

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ УКРАИНЫ
НАЦИОНАЛЬНЫЙ УНИВЕРСИТЕТ КОРАБЛЕСТРОЕНИЯ
имени адмирала Макарова

На правах рукописи

ДЫМО АЛЕКСАНДР БОРИСОВИЧ

УДК 681.5:004.9:65.012

**ПОВЫШЕНИЕ ЭФФЕКТИВНОСТИ
УПРАВЛЕНИЯ ПРОЕКТАМИ РАЗРАБОТКИ
ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ
С ОТКРЫТЫМ ИСХОДНЫМ КОДОМ**

05.13.22 – Управление проектами и программами

Диссертация на соискание ученой степени
кандидата технических наук

Научный руководитель
Шевцов Анатолий Павлович,
доктор технических наук, профессор

Николаев – 2007

СОДЕРЖАНИЕ

Введение.....	4
РАЗДЕЛ 1. АНАЛИЗ МЕТОДОВ И СРЕДСТВ УПРАВЛЕНИЯ ПРОГРАММНЫМИ ПРОЕКТАМИ С ОТКРЫТЫМ КОДОМ. ПОСТАНОВКА ЗАДАЧ ИССЛЕДОВАНИЯ.....	14
1.1. Современное состояние и проблемы управления программными проектами.....	14
1.2. Анализ способов усовершенствования процессов управления разработкой программного обеспечения с открытым кодом.....	29
1.3. Постановка задач исследования.....	46
РАЗДЕЛ 2. МОДЕЛИРОВАНИЕ ЖИЗНЕННОГО ЦИКЛА И СИСТЕМЫ УПРАВЛЕНИЯ ПРОГРАММНЫХ ПРОЕКТОВ С ОТКРЫТЫМ КОДОМ.	47
2.1. Разработка модели жизненного цикла проектов с открытым кодом.....	47
2.2. Разработка модели динамики системы управления проектами с открытым кодом.....	65
2.3. Основные выводы по разделу 2.....	77
РАЗДЕЛ 3. ПРИМЕНЕНИЕ ТЕОРИИ ПОДОБИЯ ДЛЯ УПРАВЛЕНИЯ ВРЕМЕНЕМ И СТОИМОСТЬЮ В ПРОГРАММНЫХ ПРОЕКТАХ.....	79
3.1. Подобие процессов выполнения программных проектов.....	79
3.2. Классы и группы подобных процессов выполнения проектов. Условия подобия.....	80
3.3. Обобщенная методика построения аналоговой модели для оценки времени и стоимости программных проектов.....	84
3.4. Получение абстрактных аналоговых моделей для программных проектов.....	86

3.5. Модельные критериальные уравнения для основных классов программных проектов.....	99
3.6. Основные выводы по разделу 3.....	101
РАЗДЕЛ 4. ИССЛЕДОВАНИЕ ПРИМЕНИМОСТИ АНАЛОГОВЫХ МОДЕЛЕЙ ДЛЯ УПРАВЛЕНИЯ ВРЕМЕНЕМ.....	102
4.1. Описание процесса исследования данных ISBSG.....	102
4.2. Критериальные уравнения оценки времени проектов ISBSG и обоснование их достоверности.....	107
4.3. Критериальные уравнения оценки времени программных проектов с открытым кодом.....	116
4.4. Основные выводы по разделу 4.....	118
РАЗДЕЛ 5. МЕТОДИКА УПРАВЛЕНИЯ ПРОГРАММНЫМИ ПРОЕКТАМИ С ОТКРЫТЫМ КОДОМ.....	119
5.1. Обоснование выбора открытой модели методом анализа отличительных категорий.....	119
5.2. Анализ риска выбора открытой модели, определение объема планирования и открытости.....	123
5.3. Выбор типа лицензирования продукта.....	130
5.4. Прототипирование и инициализация среды выполнения проекта.....	133
5.5. Применение методики для управления программными проектами с открытым кодом.....	137
5.6. Основные выводы по разделу 5.....	154
ВЫВОДЫ.....	156
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.....	159
ПРИЛОЖЕНИЯ.....	169

ВВЕДЕНИЕ

Актуальность исследования.

Управление проектами разработки программного обеспечения на настоящий момент стало обширным полем как научной так и практической деятельности. В области производства программного обеспечения задействовано более двух миллионов инженеров по всему миру, 20% которых составляют менеджеры. В Украине по оценкам ассоциации "IT-України" количество инженеров и менеджеров на 2007 год составляет 75 тыс. человек, при чем каждый год учебные заведения выпускают более 13 тыс. человек. При этом приблизительно 38% всех разработчиков участвует в выполнении проектов разработки программного обеспечения с открытым исходным кодом (в дальнейшем, программных проектов с открытым кодом).

Анализ последних исследований в отрасли управления программными проектами показал, что эффективное управление ими является основным способом их успешного завершения в рамках ограничений времени, стоимости и качества. По данным исследовательской группы Standish Group, по состоянию на конец 2004 года [1], из 50000 отслеженных программных проектов из-за неэффективного управления 18% завершились неудачно, 53% потребовали дополнительных затрат времени и финансов и только 29% успешно завершились. По другим данным [2], более 50% проектов завершаются неудачно. По результатам того же исследования [1], в среднем бюджет проектов превышает на 189%, а затраченное время на 222% превышает оцененное. При этом реализуется в среднем всего 69% заявленной в спецификации функциональности.

Описанная выше ситуация объясняется тем, что, цитируя Р.Л. Гласса [3], "... разработка программ является *концептуально сложным занятием*" (курсив мой. – А.Д.). Брукс добавляет [4], что трудность создания программных систем возникает благодаря их "несократимой сущности", неотъемлемыми свойствами которой являются обусловленная размерами продукта сложность и обусловленная

многочисленными связями между участниками проекта согласованность и изменяемость.

Однако, за последнее время появилось много новых моделей управления программными проектами, которые несмотря на невозможность революционного прорыва в дисциплине управления программными проектами, позволяют повысить эффективность управления. Отмечается [5], что в 2004-м году по сравнению с 1994-м, использование новых способов управления привело к повышению процентного отношения успешных проектов на 13%, и снижению количества неудач на 14%. Одним из таких способов стала модель разработки программного обеспечения с открытым кодом, которая появилась в середине 1980-х годов и стала популярной в начале XXI века.

Программные проекты с открытым кодом известны на практике как успешные проекты, которые завершаются вовремя, с минимальными затратами и в которых производится качественный продукт. Согласно [6] приблизительно 27% всех фирм-разработчиков программного обеспечения выполняют такие проекты. Кроме того, существует более 170 000 некоммерческих открытых проектов в которых участвуют более миллиона человек. Из этих проектов около 7% являются украинскими. В Украине создана Ассоциация пользователей и разработчиков открытого программного обеспечения (UAFOSS), целью которой является распространение знаний об открытом программном обеспечении, поддержка отечественных проектов с открытым кодом и проведение законодательных инициатив, связанных с их использованием в государственных структурах. Так, например, на рассмотрение Верховного Совета Украины внесен проект закона "Про використання Відкритих і Вільних форм інтелектуальної власності, Відкритих форматів даних та Відкритого (Вільного) програмного забезпечення в державних установах і державному секторі економіки". Также UAFOSS совместно с представительством ООН в Украине, Сетью разработчиков открытого программного обеспечения Украины (OSDN) и Комитетом по вопросам науки и образования Верховного Совета Украины проводят круглые столы по проблемам распространения открытого программного обеспечения я

других объектов открытой интеллектуальной собственности в Украине, материалы которых используются во время парламентских слушаний по вопросам развития информационного общества в Украине.

В связи с такой популярностью программных проектов с открытым кодом и их значением для индустрии разработки программного обеспечения Украины, возникает необходимость владеть знаниями по их управлению. Однако отмечается [7], что последние исследования были направлены лишь на описание модели и на исследование конкретных случаев ее использования. Вследствие этого вопросы изучения жизненного цикла и системы управления, которые обеспечивают основу знаний для эффективного управления проектами остаются открытыми. Незнание существующих методов выполнения процессов управления, в особенности управления временем и стоимостью, также мешает эффективному управлению и успешному завершению проектов в рамках заданных ограничений времени, стоимости и качества.

Все это приводит к необходимости решения научной задачи: изучению способов повышения эффективности управления программными проектами с открытым кодом путем разработки моделей жизненного цикла и системы управления, новых моделей оценки стоимости и времени, новой методики управления.

Работа, решающая эти задачи, является актуальной и позволит усовершенствовать выполнение процессов управления и успешно завершать программные проекты с открытым кодом в рамках заданных ограничений времени, стоимости и качества.

Связь работы с научными программами, планами, темами.

Диссертационная работа выполнена в рамках научно-исследовательских работ, которые производились на кафедре Информационных управляющих систем и технологий Национального университета кораблестроения имени адмирала Макарова совместно с ГП НПКГ «Заря-Машпроект» на тему «Анализ перспектив и возможностей использования открытого программного обеспечения

на предприятиях машиностроительной отрасли» и совместно с ОАО НИИ «Центр» на тему «Усовершенствование методов оценки стоимости и времени разработки программного обеспечения». Модели системы управления и жизненного цикла разработаны согласно заданиям, указанным в статуте Украинской ассоциации разработчиков и пользователей свободного программного обеспечения UAFOSS. Разработка методики управления проведена во время выполнения программного проекта с открытым кодом "KDE-Eclipse" в рамках гранта программы "Google Summer of Code", где Дымо А.Б. выступал менеджером и ответственным исполнителем проекта.

Целью научного исследования является повышение эффективности управления программных проектов с открытым исходным кодом.

Основные задачи научного исследования:

1. Разработка модели жизненного цикла проектов с открытым исходным кодом, выделение фаз модели и установление перечня действий, выполняемых в цикле.
2. Разработка математической модели динамики системы управления программными проектами с открытым исходным кодом и исследование устойчивости и самоорганизации системы.
3. Разработка модели оценки времени и стоимости программных проектов с открытым исходным кодом.
4. Проведение эмпирических исследований моделей оценки времени и стоимости для подтверждения их точности и применимости.
5. Разработка методики управления программными проектами с открытым кодом.

Объектом исследования являются процессы управления программными проектами с открытым исходным кодом.

Предметом исследования являются модели жизненного цикла и системы управления программными проектами с открытым кодом.

Методы исследований. Проведенные исследования базируются на методах теории систем, теории подобия, математической статистики, математического моделирования, использования компьютерных систем и современных программных комплексов.

Научная новизна полученных результатов. Основным научным результатом диссертационной работы является множество знаний по управлению программными проектами с открытым кодом, которое позволяет повысить эффективность управления благодаря учету специфики этих проектов.

Научная новизна результатов исследования, которые выносятся на защиту:

Впервые:

- разработана модель жизненного цикла программных проектов с открытым кодом на основе анализа существующих проектов, которая учитывает специфический для проектов с открытым кодом процесс обработки и управления внешними изменениями и дополнительные действия по выбору схемы лицензирования и распределенной разработки; и которая может быть использована для обоснования принятия решений по управлению проектом и как эффективный способ контроля реализации проекта;
- разработана математическая модель анализа динамики системы управления программными проектами с открытым кодом на основании обобщенной структурной схемы модели динамики программных проектов, которая учитывает дополнительные переменные состояния и обратные связи между параметрами системы и среды и позволяет исследовать устойчивость и самоорганизацию в проектах с открытым кодом;
- получены аналоговые модели оценки времени выполнения программных проектов на основе принципа подобия программных проектов и использования метода анализа размерностей теории подобия, которые учитывают существенные закономерности между определяющими время параметрами и точность которых превышает точность существующих моделей.

Усовершенствованы:

- способы и критерии обоснования выбора модели разработки программного обеспечения с открытым кодом на основе адаптированных методов анализа различительных категорий и рисков, которые обеспечивают возможность принять решение о выполнимости проекта уже в начале его жизненного цикла.

Получило дальнейшее развитие:

- исследование явления самоорганизации с использованием теории сверхустойчивых систем, на основе которого определяются механизмы обеспечения и управления самоорганизацией;
- подход к исследованию проектов с открытым кодом как к аналогам быстрых проектов, на основе чего становится возможным применение уже известных методов и способов управления быстрыми проектами для случая проектов с открытым кодом.

Обоснованность и достоверность полученных результатов, выводов и рекомендаций обеспечена соответствием разработанной модели жизненного цикла стандартам ISO 12207 и ДСТУ 3918-1999; удовлетворительным согласованием результатов оценки стоимости и времени разработанными моделями с реальными данными проектов из базы данных ISBSG; успешным выполнением проектов с открытым кодом согласно с предлагаемой методикой управления.

Практическая ценность результатов работы и их практическое использование определяется применением разработанных моделей и методик как основы знаний в управлении программными проектами с открытым кодом; выполнением оценок времени и стоимости разработки программного обеспечения с большей на 55% точностью чем у модели COCOMO. Выполнены проекты с открытым кодом MediaCloth, KDevelop, ActiveReCpp, KDE Eclipse в рамках заданных ограничений времени, стоимости и качества с использованием предложенной методики управления. При этом практически подтверждается

точность оценок времени и ценность рекомендаций по обоснованию открытости кода и выбора типа лицензирования. Результаты диссертационной работы использовались при выполнении свободного проекта с открытым кодом KDE, были внедрены на предприятиях Pluron и UkrInvent, в Национальном университете кораблестроения имени адмирала Макарова.

Апробация работы.

Основные положения и результаты диссертационной работы докладывались на:

- конференциях разработчиков KDE «Kastle» (Novi Hrad, Czech Rep., 2003), «Akademy 2004» (Ludwigsburg, Germany, 2004), «Akademy 2005» (Malaga, Spain, 2005), "Akademy 2006" (Dublin, Ireland, 2006), "Akademy 2007" (Glasgow, United Kingdom, 2007).
- конференции “Сучасні інформаційні технології та інноваційні методики навчання в підготовці фахівців: методологія, теорія, досвід, проблеми” Винницького державного педагогічного університету (Винница, 2004);
- конференции “Информационные технологии в XXI веке” (Днепропетровск, 2004);
- конференции “Інноваційний розвиток на основі технологічної зрілості в управлінні проектами” КНУБА (Киев, 2004);
- конференции “Автоматизация: проблемы, идеи, решения” (Севастополь, 2004);
- конференции “Світ інформації та телекомунікацій-2005” ГУИКТ (Киев, 2005);
- конференции FOSDEM – Free and Open Source Developers Meeting (Brussels, Belgium, 2005);
- конференции Linux Desktop Summit (Ottawa, Canada, 2005);
- конференциях LinuxSymposium (Ottawa, Canada, 2005, 2007);
- конференции Open Source Security (Warsaw, Poland, 2005);
- II Open Source World Conference (Malaga, Spain, 2006);

- II Международной научно-практической конференции "Управление проектами: состояние и перспективы (Николаев, 2006);
- IX Научно-практической международной конференции «Информационные технологии в образовании и управлении» (Новая Каховка, 2007)

Личный вклад автора. Основные научные разработки и выводы диссертационной работы, получены автором самостоятельно. Вклад автора в коллективно опубликованные работы конкретизирован в списке публикаций.

Публикации.

1. Дымо А.Б., Морозова А.С. Открытая модель жизненного цикла программных проектов. // Збірник наукових праць НУК. - Миколаїв: НУК, 2005. - №5 (404). - С.134-141.

Личный вклад автора: разработана модель жизненного цикла программных проектов с открытым кодом и определен перечень фаз, которые составляют жизненный цикл.

2. Дымо А.Б. Применение теории подобия для оценки стоимости разработки программных продуктов. // Збірник наукових праць НУК. - Миколаїв: НУК, 2006. - №3 (408). - С.162-167.

3. Дымо А.Б. Модели оценок времени выполнения программных проектов на основе базы данных метрик проектов ISBSG // Збірник наукових праць НУК. - Миколаїв: НУК, 2006. - № 5 (410). - С.53-56.

4. Дымо А.Б., Олейник А.И. Определение объема открытости исходного кода в программном проекте методом анализа рисков // Вестник ХНТУ. - Херсон: ХНТУ, 2007. - №4(27). - С. 248 - 251.

Личный вклад автора: идентифицированы риски открытости кода программного проекта и адаптирован алгоритм выполнения анализа рисков для проектов с открытым кодом.

5. Дымо А.Б. Исследование динамики системы управления программными проектами с открытым кодом // Збірник наукових праць НУК. - Миколаїв: НУК, 2007. - № 3 (414). - С.174-179.

6. Дымо А.Б., Кошкин К.В. Организация работы студентов в рамках проекта автоматизации деятельности университетов при выполнении дипломного проектирования // Сучасні інформаційні технології та інноваційні методики навчання в підготовці фахівців: методологія, теорія, досвід, проблеми. - Київ-Вінниця: ДОВ Вінниця, 2004. - №6. - С. 343 - 349.
Личный вклад автора: обоснована применимость программного обеспечения с открытым кодом для разработки систем автоматизации деятельности университетов и предложен перечень программных средств с открытым кодом, которые можно применять при разработке.
7. Дымо А.Б. CASE средства создания программного обеспечения автоматизации деятельности университетов // Международная научно-техническая конференция “Автоматизация: проблемы, идеи, решения”: Материалы конференции. - Севастополь: СевНТУ, 2004. - С. 58 - 59.
8. Дымо А.Б. Инструменты поддержки жизненного цикла разработки программного обеспечения автоматизации деятельности университетов // “Информационные технологии в XXI веке”: Материалы конференции. - Днепропетровск: ИПК ИнКомЦентра УГХТУ, 2004. - С. 67-68.
9. Дымо А.Б. Интенсификация разработки программного проекта: открытая модель // “Світ інформації та телекомунікацій-2005”: Матеріали конференції. - Київ: ДУІКТ, 2005. - с. 132.
10. Дымо А.Б. Аналоговое моделирование: инновация в области оценки показателей программных проектов // “Світ інформації та телекомунікацій-2005”: Матеріали конференції. - Київ: ДУІКТ, 2005. - с. 133.
11. Dymo A. KDevelop – the Comprehensive Tool for Development Tasks // Linux Symposium 2005: Programme. - Ottawa, Canada, 2005. - p. 56.
12. Dymo A. Rapid Linux Desktop Development with KDE and KDevelop // Desktop Developers' Conference: Programme. - Ottawa, Canada, 2005. - p. 24.
13. Dymo A. Automated search for security problems inside the C++ code // Open Source Security conference: Conference Proceedings. - Warsaw, Poland, 2005. - pp. 21-36.

14. Dymo A. Open Source Software Engineering. // II Open Source World Conference: Proceedings Book. - Malaga, Spain, 2006. - pp. 59-63.
15. Dymo A. Accomplishments And Challenges of The KDevelop Team. // "aKademy 2006" KDE Contributors Conference: Proceedings of the Conference. - Dublin, Ireland, 2007. - p. 9.
16. Dymo A. Ruby on Rails Application Development with KDevelop and RadRails // Linux Symposium: Event Programme. - Ottawa, Canada, 2007. - p. 16.

Структура и объем работы.

Диссертация состоит из введения, 5 разделов, выводов и приложений. Включает 158 страниц основного текста, 29 иллюстраций, 45 таблиц. Список литературы включает 121 наименование на 9 страницах. 5 приложений размещено на 59 страницах.

Автор выражает благодарность профессору Кошкину К.В., оказавшему поддержку при выполнении предоставляемого диссертационного исследования.

РАЗДЕЛ 1.

АНАЛИЗ МЕТОДОВ И СРЕДСТВ УПРАВЛЕНИЯ ПРОГРАММНЫМИ ПРОЕКТАМИ С ОТКРЫТЫМ КОДОМ.

ПОСТАНОВКА ЗАДАЧ ИССЛЕДОВАНИЯ

1.1. Современное состояние и проблемы управления программными проектами

Управление программными проектами с момента своего становления как инженерной дисциплины на конференции NATO Software Engineering Conference в 1968 году всегда считалось концептуально сложным занятием [3]. Ф. Брукс, один из пионеров инженерии программного обеспечения указывал на следующие существенные трудности в управлении программными проектами [4]:

- сложность управления, обусловленная размерами создаваемого продукта;
- сложность задачи отображения (моделирования) процессов и явлений реального мира в программном продукте;
- проблема изменяемости и согласования обусловленная многочисленными связями между участниками проекта.

Ройс отмечает в [8] еще несколько причин возникновения трудностей в управлении программными проектами:

- гибкость создаваемого продукта, возможность запрограммировать "практически все что угодно", затрудняющая планирование, мониторинг и управление разработкой;
- непредсказуемость процесса управления, малая доля проектов с успешным завершением без превышения первоначальных бюджетных и временных ограничений;
- большая степень влияния качества управления на успех или неудачу;

- незрелость процессов разработки, большая доля незаконченного, неиспользованного и переделанного программного обеспечения, производимого в рамках проекта.

Согласно данным исследовательской группы Standish Group, по состоянию на конец 2004 года [1], из 50000 отслеженных программных проектов 18% завершились неудачно, 53% потребовали дополнительных затрат времени и финансов и только 29% успешно завершились. По другим данным [2], более 50% проектов завершаются неудачно. По результатам того же исследования [Standish], в среднем бюджет проектов превышает на 189%, а затраченное время на 222% превышает оцененное. При этом реализуется в среднем всего 69% заявленной в спецификации функциональности а общая сумма потерь из-за неудачного управления проектами составляет 78 млрд. долларов в год. Графически приведенные данные иллюстрируются на рис. 1.1.

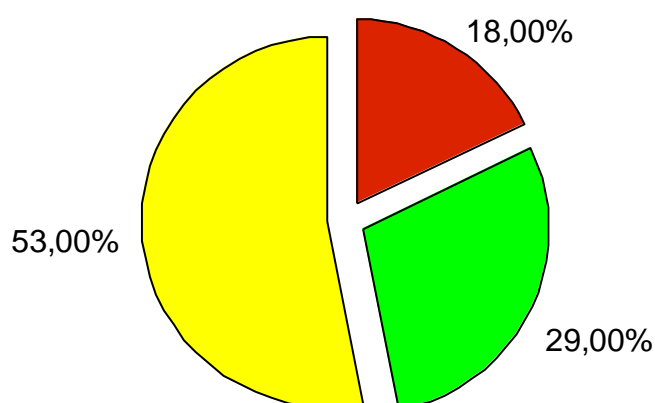


Рис. 1.1. Трудности управления программными проектами согласно исследованию Standish Group

- проекты испытывали трудности
- проекты завершились удачно
- проекты завершились неудачно

На протяжении многих лет предпринимались попытки изменить эту ситуацию и решить некоторые из проблем управления программными проектами.

Ройс [8] приводит следующий перечень направлений в совершенствовании процессов управления:

1. уменьшение размера или сложности того, что предстоит разрабатывать;
2. усовершенствование процесса разработки;
3. использование более квалифицированного персонала или хороших команд;
4. использование лучшей среды (инструментария для автоматизации процесса);
5. достижение уступок и компромиссов для пороговых значений качества.

За последние 25 лет были созданы и получили свое развитие большое количество моделей и методов управления программными проектами, каждый из которых по разному решал пять вышеперечисленных задач. Наиболее часто используемые подходы к решению этих задач в современной практике управления программными проектами следующие.

Уменьшение размера или сложности того, что предстоит разрабатывать.

В рамках классической водопадной а также спиральной модели разработки предполагается два способа решения этой задачи, предложенные Боэмом [9, 10, 11]:

- декомпозиция задач, составление детализированных WBS (Work Breakdown Structure, иерархическая структура работ) и распределение этих задач между участниками проекта;
- использование COTS (Common Off The Shelf, готовых к использованию) компонентов, покупаемых специально для решения определенных задач из WBS.

Группа исследователей, создавшая быструю (agile) модель разработки [12, 13, 14] акцентирует внимание на сохранении "простоты" при разработке программного обеспечения. Под этим они подразумевают "...искусство максимизации объема невыполняемых работ" при сохранении соответствия разрабатываемого продукта поставленным требованиям. Согласно такому

подходу, проектная команда должна прилагать все усилия по сохранению простоты и расширяемости реализуемых моделей и архитектуры.

Усовершенствование процесса разработки.

Основное внимание при решении этой задачи уделяется вопросам количественного предсказания и анализа параметров процесса разработки. Наиболее важными признаются:

- Вопросы оценки размера создаваемого продукта, для чего наиболее часто применяемыми методами являются метод аналогий Вайнберга и Шульмана [15], метод PERT Путнама и Фитцсиммонса [16], метод функциональных точек [17, 18, 19, 20] а также многие другие методы, основанные на оценке тех или иных характеристик процесса и продукта [21].
- Вопросы оценки стоимости, трудоемкости и времени выполнения программного проекта, для которых наиболее часто используемым методом является метод СОСОМО [9] и его развитие СОСОМО2000 [11]. Самой популярной альтернативой СОСОМО является метод экспертных оценок Wideband Delphi [22, 23, 24]. Менее распространенными – методы оценки сверху вниз и снизу вверх, метод задача-модуль, описанные Бозмом [9].

Использование более квалифицированного персонала или хороших команд.

Эта задача достаточно долго не привлекала к себе внимания несмотря на ряд исследований в этой области, выполненных ДеМарко [25] и Константином [26]. Однако с увеличением интереса к быстрым способам разработки увеличилось и внимание ко всему в процессе выполнения программных проектов, что связано с человеческими факторами. Прежде всего это качество и продуктивность, работа в команде, динамика поведения коллектива, личность и программирование, руководство проектом и организационные вопросы, разработка интерфейса и взаимодействие между человеком и машиной, познавательная деятельность, психология, процессы мышления. В быстрых моделях разработки решение проблем, связанных с человеческим фактором, предполагается увеличением

внутрикомандного взаимодействия, ускорением обмена информацией и самоорганизацией команд [12].

Использование лучшей среды (инструментария для автоматизации процесса).

В настоящее время этот вопрос не стоит столь остро, как это было 10-15 лет назад. Были созданы языки программирования 4-го поколения, быстрые и эффективные компиляторы, библиотеки компонентов, редакторы, интегрированные среды разработки. Вопрос использования лучшей среды сводится сейчас в основном к вопросу о политике компании и размеру средств, вкладываемых в среду разработки [26].

Достижение уступок и компромиссов для пороговых значений качества.

Проекты по разработке программного обеспечения, как и любые другие проекты представляет собой компромисс между соответствием требованиям, стоимостью и временем [27]. Поэтому в управлении программными проектами всегда остро стоит вопрос достижения максимального качества в условиях поставленных ограничений по стоимости и времени. Классическим подходом к решению этой задачи является метод GOALS (Goal Oriented Approach to Life-cycle Software, подход к разработке программного обеспечения ориентированный на цели), предложенный Боэмом [9] и основанный на декомпозиции целей, выделения наиболее важных и первостепенных целей, а также количественной оценки и выделении ресурсов для достижения конкретных подцелей. Для быстрой разработки программного обеспечения используется подход к выполнению программных проектов с точки зрения теории игр. Так Коберн рассматривает проект как кооперативную игру изобретения и коммуникации [12] и основное внимание уделяет достижению баланса в архитектуре разрабатываемого продукта, достижению эффективности в процессах проекта на основании критерия достаточности, называемого им YAGNI (You Aren't Going to Need It, вам это не потребуется).

Как видно из вышеприведенного обзора, для программных проектов выполняемых согласно разным моделям, используются разные методы и способы совершенствования процессов управления решения. Очевидно, что для каждой модели разработки будет применимым только ограниченный набор методов. Кроме того, может возникнуть ситуация, когда существующие методы окажутся и вовсе неприменимыми.

Модель разработки программного обеспечения с открытым кодом.

Программное обеспечение с открытыми исходными кодами существовало на протяжении всего периода развития программной индустрии. Однако как отдельное явление оно начало формироваться когда Ричард Столлман объявил о начале проекта GNU (Gnu Not Unix) в январе 1984 г. – проекта по созданию "свободной" UNIX-подобной операционной системы с открытым исходным кодом. Под понятием "свобода" подразумевалась:

- свобода запуска программ для любых целей;
- свобода модификации программ;
- свобода распространения копий программ (как платного так и бесплатного);
- свобода распространения модифицированных версий программы.

Еще одной характеристикой этого проекта стала невозможность закрытия исходного кода, обеспечиваемая лицензией GNU GPL (General Public License, публичная лицензия). При этом не подразумевалось наличие инвестиций в проект, так как все работы по проекту выполнялись на добровольной основе.

Несмотря на первичную установку на декоммерциализацию, модель также нашла свое применение и для коммерческих программных проектов. Началу ее использования положила фирма Netscape в 1998 году открыв исходный код интернет-коммуникатора Netscape Navigator и начав один из наиболее успешных проектов с открытым кодом – проект Mozilla (по состоянию на 2007 год браузер Firefox, произведенный в рамках проекта Mozilla согласно различным исследованиям занимает от 14% [28] до 31% [29] рынка браузеров). Вслед за этим

была основана организация OSI (Open Source Initiative), задачами которой стали продвижения идей открытого программного обеспечения. OSI в 1997 г. опубликовала определение открытого программного обеспечения [30, 31], которое перечисляет следующие основные критерии открытости:

1. Свободное повторное распространение продукта.

Лицензия не должна ограничивать право какого-либо субъекта на продажу или бесплатное распространение программного обеспечения как компонента совокупного набора программ, полученных из разных источников. Лицензия не должна требовать отчислений или других выплат за подобное распространение.

Обязательность требования свободного распространения устраняет соблазн кратковременного дохода за счет продаж в ущерб многим долгосрочным выгодам. При отсутствии этого требования стала бы невозможной эффективная кооперация участников проекта (как индивидуумов, так и компаний).

2. Исходный код.

Программа с открытым кодом должна включать исходный код, и должно допускаться ее распространение как в виде исходного кода, так и в откомпилированном виде. В тех случаях, когда какой-либо вид продукта распространяется без исходного кода, должны существовать широко оглашенные способы получения исходного кода, стоимость которых не превышает обоснованных расходов на воспроизведение; предпочтительным способом является бесплатное получение из Internet. Исходный код должен быть предпочтительной формой, в которой программисты могли бы модифицировать программу. Преднамеренное запутывание исходного кода не допускается. Распространение промежуточных форм представления программ, такие как результаты вывода препроцессора или транслятора, не допускаются.

Требуется доступ к оригинальному исходному коду, потому что невозможно совершенствовать программы без их модификации. Поскольку

цель открытого подхода состоит в облегчении процесса совершенствования программ, требуется, чтобы модификация кода производилась простым образом.

3. Производные продукты.

Лицензия должна допускать создание модифицированных и производных продуктов и должна разрешать их распространение на таких же условиях, что и оригинального программного продукта.

Возможность только чтения исходного кода не является достаточной для проведения независимой экспертизы и быстрого эволюционного отбора. Для обеспечения быстрой эволюции требуется возможность проведения экспериментов с исходным кодом и распространения его модифицированных вариантов.

4. Сохранность авторского исходного кода.

Лицензия может ограничивать распространение исходного кода в модифицированной форме, только если лицензия допускает распространение вместе с исходным кодом «файлов-патчей» для модификации программы во время создания исполняемой системы. Лицензия должна явным образом разрешать распространение программного обеспечения, созданного на основе модифицированного исходного кода. Лицензия может требовать, чтобы названия или номера версий производных продуктов отличались от тех, которые имелись у исходного программного обеспечения.

Пользователи имеют право знать, кто отвечает за используемое ими программное обеспечение. Разработчики и люди, поддерживающие код, имеют право знать, за поддержку чего они отвечают, и защищать свою репутацию и авторское право. Соответственно, лицензия программного обеспечения с открытым кодом должна гарантировать простую доступность исходного кода, но может требовать его распространения в виде изначального исходного кода плюс патчи. При этом можно производить

«неофициальные» изменения, но они легко отличимы от базового исходного кода.

5. Отсутствие пристрастий по отношению к отдельным лицам или группам лиц.

Для получения максимальной пользы от процесса разработки программного обеспечения с открытым кодом у как можно большего числа людей и групп должны иметься одинаковые права на свой вклад в проект. Поэтому запрещается наличие в любой лицензии на программное обеспечение с открытым кодом условий, не допускающих кого бы то ни было к участию в этом процессе. В некоторых странах, включая США, имеются экспортные ограничения на некоторые типы программного обеспечения. В лицензиях, соответствующих данному определению концепции, могут содержаться предупреждения о применимых ограничениях и напоминания об обязанности соблюдения закона; однако в них самих не должны содержаться подобные ограничения.

6. Отсутствие пристрастий к областям применения и деятельности.

Лицензия не должна накладывать ограничения на применение программы какой-либо области применения. Например, лицензия не может ограничивать использование в области бизнеса или в области генетических исследований.

Основной смысл этого пункта состоит в запрете лицензионных ловушек, которые препятствуют коммерческому использованию программного обеспечения с открытым кодом.

7. Распространение лицензии.

Права, приписанные данной программе, должны распространяться на всех ее получателей без потребности оформления ими какой-либо дополнительной лицензии.

Смысл этого пункта состоит в запрете закрытия программного обеспечения косвенными способами, такими как требование соглашения о неразглашении.

8. Лицензия не должна специализироваться для каких-либо продуктов.

Права, приписываемые данной программе, не должны зависеть от того, является ли эта программа частью какого-либо специального набора программ. Если программа извлекается из этого набора и используется или распространяется в соответствии с условиями ее лицензии, то все субъекты, которым распространяется данная программа, должны иметь такие же права, как и те, которые предоставляются вместе с исходным набором программ.

9. Лицензия не должна ограничивать другое программное обеспечение.

В лицензии не должны присутствовать ограничения на другое программное обеспечение, распространяемое вместе с лицензируемым программным обеспечением. Например, в лицензии не должно требоваться, чтобы все остальные программы, распространяемые на том же носителе, что и данная программа, представляли собой программное обеспечение с открытым кодом.

Дистрибьюторы программного обеспечения с открытым кодом имеют право собственного выбора по отношению к своему собственному программному обеспечению.

10. Лицензия должна быть нейтральной по отношению к технологии.

Ни одно из положений лицензии не должно основываться на какой-либо индивидуальной технологии или стиле интерфейса.

Этот пункт в особенности затрагивает лицензии, в которых требуется явные действия для установления контакта между лицензиатом и лицензиатом. Методы, в которых принятие лицензии подтверждается нажатием «согласительной» кнопки (click-wrap), могут конфликтовать с такими важными методами распространения программного обеспечения, как получение по FTP, антологии CD-ROM и зеркалирование Web-сайтов; такие методы могут также затруднять повторное использование кода. Лицензии, соответствующие данному определению, должны допускать возможность того, что (а) распределение программного обеспечения может

производиться через каналы, отличные от Web, не поддерживающие методы click-wrap, и (b) защищенный лицензией код (или его повторно используемая часть) может выполняться в среде без графического пользовательского интерфейса, в которой невозможна поддержка диалоговых окон.

Такая модель разработки стала неожиданно популярной. С 1984 года возникло множество таких проектов, поставивших перед собой самые различные цели.

Из наиболее успешных некоммерческих проектов следует отметить:

1. Linux – проект разработки ядра операционной системы. Разработка начата в 1991 году. Согласно отчетам IDC [32] 27% серверов по всему миру к ноябрю 2006 г. работали под управлением Linux. Доля Linux на рабочих столах пользователей составила 2% в Северной Америке и 5% в Европе. 46% всех государственных учреждений Европы использует Linux на своих компьютерах.
2. KDE – проект разработки графической среды пользователя. Разработка начата в 1996 году. В настоящее время занимает 64.9% рынка графических сред для операционных систем UNIX.
3. Apache – проект разработки веб-сервера. Разработка начата в 1995 г. Согласно отчетам Netcraft [33] по состоянию на май 2005 г. 70% из более чем 63 миллионов веб-серверов в Internet обслуживались Apache.

Наиболее успешными коммерческими проектами являются:

1. Eclipse – проект платформы для создания интегрированных сред разработчика и сред разработки на языках Java, C++ и др. Разработка началась в 2001 г. На сегодняшний день в рамках проекта ведется разработка более 50 подпроектов, разрабатываемых более 115 компаний. Этот проект является беспрецедентным примером кооперации производителей коммерческого программного обеспечения в рамках одного проекта.

2. OpenOffice – проект разработки пакета офисных приложений – текстового, табличного и графического редакторов. Разработка начата в 2000г. на основании кода StarOffice, открытого компанией Sun Microsystems.

Вышеперечисленные проекты являются только малой долей среди успешных проектов с открытым кодом. Важно отметить также и успех коммерческих проектов несмотря на их сравнительно небольшое количество. Так по состоянию на декабрь 2006 года доля коммерческих проектов составила 15% всего количества открытых проектов [34]. Еще 20% разрабатываются университетами и другими образовательными учреждениями. Остальная часть является некоммерческими проектами.

Примерная оценка количества проектов с открытым кодом и количества разработчиков в этих проектах по состоянию на январь 2007 г. [35] приведена в диаграмме на рис. 1.2. Были использованы данные крупнейшего репозитория открытых проектов Sourceforge и других репозитариев – Google Code, Berlios, RubyForge, Savannah.

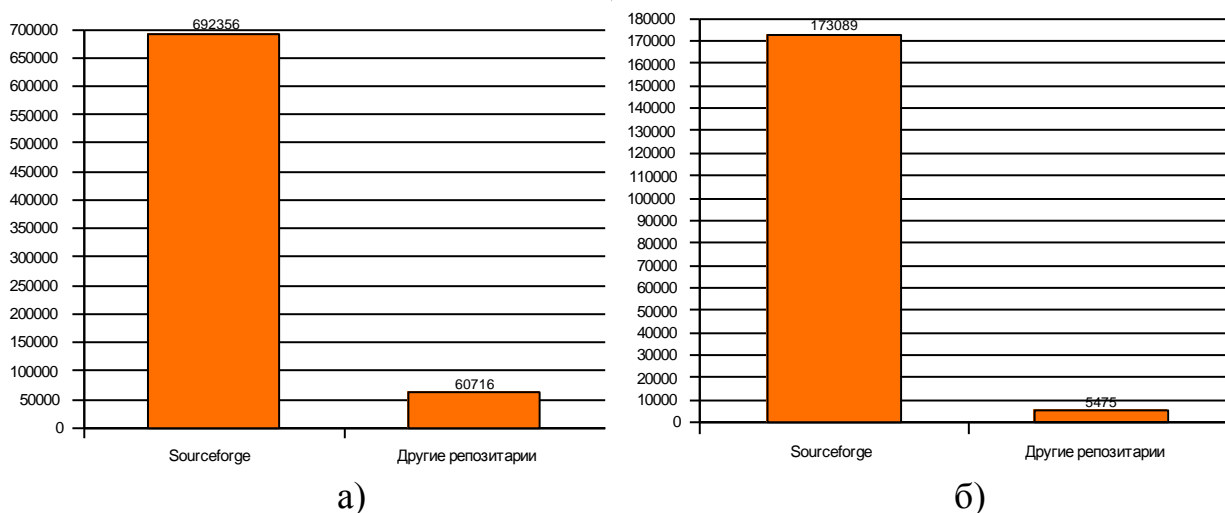


Рис. 1.2. Данных репозитариев проектов с открытым кодом:

а) о количестве разработчиков

б) о количестве проектов

Таким образом, согласно данным репозитариев общее количество проектов с открытым кодом превышает 170 000, а общее количество людей, участвующих в них – 700 000. В то же время согласно данным корпорации Evans [36], количество

участников проектов с открытым кодом по всему миру превышает 1 100 000 человек.

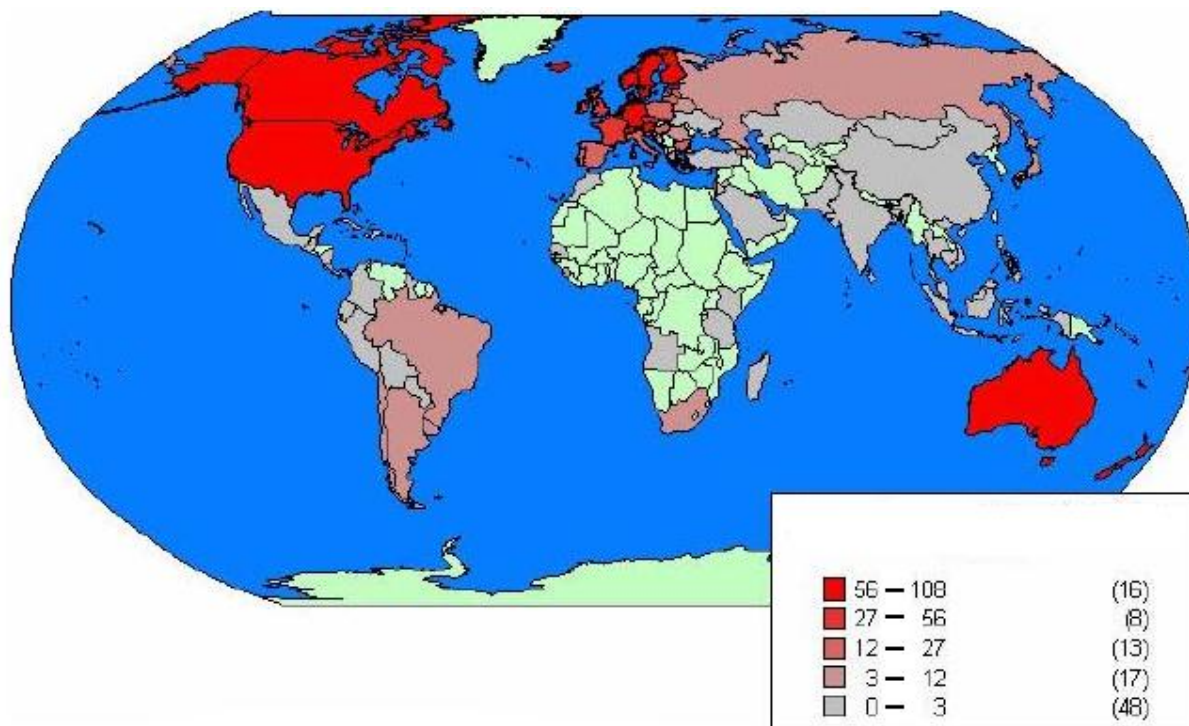


Рис. 1.3. Количество разработчиков проектов с открытым кодом на один миллион населения страны

Согласно данным портала Berlios [37] количество разработчиков из Украины составляет 0.6% от всего количества участников проектов с открытым кодом. Исследования, проводившиеся университетом UNI-MERIT (Нидерланды) по контракту с Еврокомиссией в 2006 г. утверждают [34], что количество участников из Украины составляет до 3х человек на 1 миллион населения страны (рис. 1.3).

Эти данные позволяют оценить количество участников из Украины в диапазоне от 150 до 7 000 человек. Такая сравнительно небольшая цифра тем не менее является неплохим показателем если соотнести ее со средним уровнем дохода (рис. 1.4) или с количеством подключенных пользователей Internet (рис. 1.5). Из отчета UNI-MERIT известно, что наибольшее распространение проекты с открытым кодом получили в странах Евросоюза (45% от общего количества участников) и США (27%), т.е. в странах с высоким уровнем дохода и доступом населения в Internet.

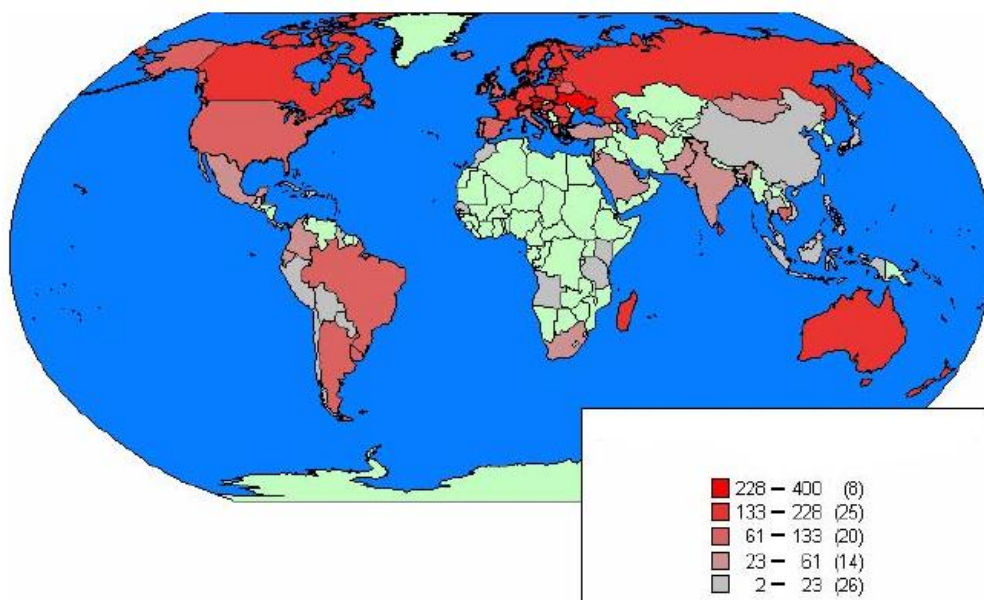


Рис. 1.4. Количество разработчиков проектов с открытым кодом на один миллион подключенного к Internet населения

Данные рис. 1.4 и 1.5 позволяют оценить количество украинских разработчиков в 1500 – 3600 чел. на основании ВВП на душу населения (из расчета 6000\$ согласно данным Всемирного Банка [38]) или 1800 – 3200 чел. (из оценки 3-6 млн. пользователей Internet в Украине [39]).

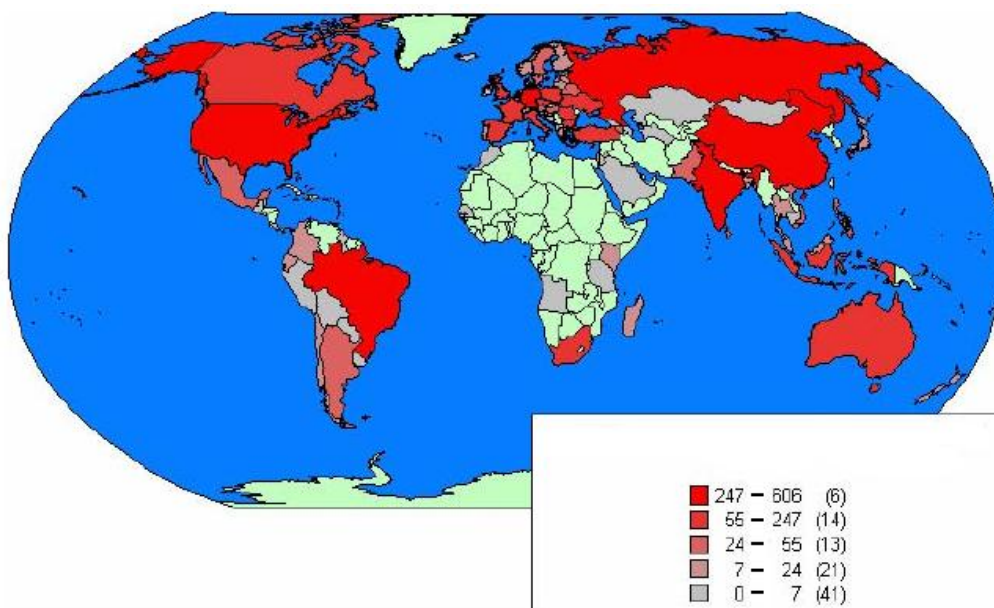


Рис. 1.5. Количество разработчиков проектов с открытым кодом на 1000\$ валового внутреннего продукта на душу населения.

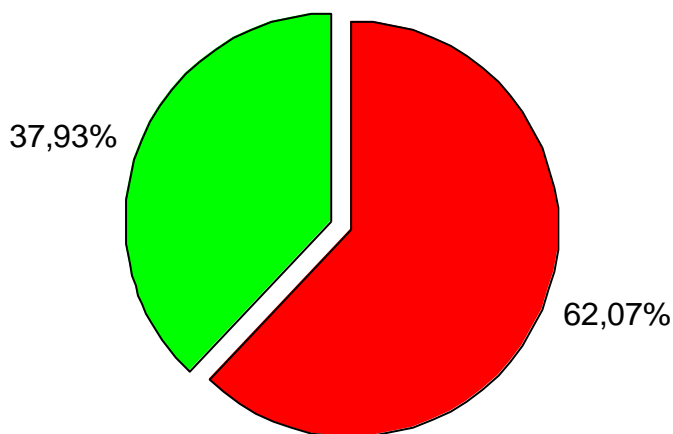
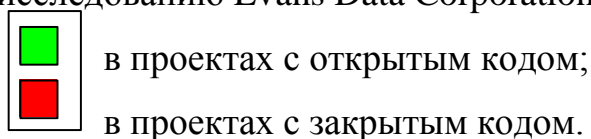


Рис. 1.6. Занятость разработчиков в программных проектах согласно исследованию Evans Data Corporation:



Учитывая данные исследования Evans Data Corporation [36], которые показывают, что более 35% разработчиков по всему миру задействованы в проектах с открытым кодом (рис. 1.6) и эта цифра растет на 2-5% в год, а также учитывая огромное количество и успешность как коммерческих так и некоммерческих выполняемых проектов, следует признать важность открытой модели разработки программного обеспечения и необходимость исследований как самой модели так и способов повышения эффективности управления программными проектами, выполняемыми в этой модели. Однако следует обратить внимание на сравнительно небольшую долю коммерческих проектов (15%), что объясняется многими аналитиками [40, 41] отсутствием теоретического обоснования основных принципов выполнения таких проектов. Как отмечается в [41], "самая большая проблема индустрии открытых исходников, препятствующая ее дальнейшему развитию, сегодня состоит в том, что даже сторонники открытого кода в большинстве своем еще не понимают, что открытая модель является действительно самостоятельной. Никто из сторонников открытого кода не понимает, какой инструмент оказался у них в руках. А отсюда

уже вытекает физическая невозможность использовать его эффективно". Это подтверждается и многочисленными опасениями, высказываемыми менеджерами и руководством компаний. Так, например, в Sun Microsystems несмотря на положительный опыт открытия исходного кода OpenOffice и 6-ти летний опыт его дальнейшей разработки в открытой модели более 2х лет принимали решение об открытии исходного кода Java и перевода проекта Java на открытую модель [42, 43, 44, 45].

Исходя из вышесказанного, очевидно, что задача по изучению модели разработки открытого программного обеспечения является важной и своевременной. Решением такой задачи будет являться формальное описание модели открытых исходников, составляющих ее фаз и действий, а также формулирование основных принципов управления открытыми проектами. Иными словами необходима модель процессов разработки (модель жизненного цикла) и модель системы управления. Все это предоставит необходимую основу знания для организаций, планирующих или разрабатывающих программные проекты с открытым кодом.

1.2 Анализ способов усовершенствования процессов управления разработкой программного обеспечения с открытым кодом

Как было отмечено в разделе 1.1 перед менеджерами программных проектов стоит пять существенных проблем и каждая известная модель разработки по своему подходит к их разрешению. Поэтому ниже для проектов с открытым кодом приведен анализ решения их специфических задач с целью повышения эффективности управления этими проектами.

Уменьшение размера или сложности того, что предстоит разрабатывать.

За счет открытости кода и гибких правил его распространения количество повторно используемого кода доходит до 40% от размера создаваемого продукта.

Это подтверждается, например, данными исследования MERIT [34], приведенными в таблице 1.1.

Таблица 1.1

**Объем повторно используемого кода на основании
анализа программ из дистрибутива Debian 3.1 (2006г.)**

Параметры кода	Значения для Debian 3.1
А – количество строк исходного кода	247 809 088
Б – количество строк повторно использованного кода	157 434 545
Степень повторного использования (А / Б)	1.57
Доля повторного использования ((А – Б) / А)	36%

Такая высокая степень повторного использования позволяет говорить о модели разработки программного обеспечения с открытым кодом как о способе избежать затрат на исследование несвязанных с производимым продуктом технологий для разработчиков и компаний, выполняющих проекты. Повторное использование готовых компонентов позволяет привлечь ресурсы к осуществлению инноваций только в необходимых областях. Так, например, Университет Нью-Йорка (NYU) получил грант от Министерства Обороны США в размере 3 000 000\$ и разработал нового компилятора языка Ada (GNAT), . В тоже время аналогичные проекты затратили более 20 000 000\$. Целенаправленность инвестиций обусловлена тем, что компилятор создавался на базе уже имеющихся компиляторов GNU C и GNU C++. В дальнейшем компилятор разрабатывался по открытой модели с периодическим дофинансированием. По оценкам на 2006 г. трудоемкость его разработки составила 4764 человеко-месяцев а оцененная стоимость – 45 000 000\$.

Еще одной положительной характеристикой модели является четкое определение правил совместной работы над проектом, что позволяет увеличивать эффективность вложений в исследование и разработку коммерческим компаниям а также обеспечить надежные пути взаимодействия коммерческих и

некоммерческих участников проекта. Примерами такой совместной работы являются проекты Eclipse и Maemo. Maemo – операционная система нового поколения для мобильных телефонов и интернет-коммуникаторов Nokia. Произведенный в рамках этого проекта продукт является результатом совместных усилий 8 коммерческих компаний и 6 некоммерческих организаций, что иллюстрирует таблица 1.2.

Таблица 1.2

**Распределение объема кода
созданного в проекте Maemo по участникам проекта**

Участник проекта	Объем кода, строк	Процент кода
Коммерческие компании		
RedHat Corporation	415 000	8.6%
Silicon Graphics Corporation	275 000	5.7%
IBM Corporation	220 000	4.6%
Nokia Corporation	200 000	4.1%
Intel Corporation	160 000	3.3%
Sun Microsystems	130 000	2.7%
Digital Equipment Corporation	130 000	2.7%
Hewlet-Packard Corporation	115 000	2.4%
Некоммерческие организации		
Free Software Foundation	2 785 000	58.3%
The Open Group	200 000	4.1%
The OpenSSL Project	75 000	1.6%
The Purdue Research Foundation	55 000	1.1%
XFree	22 000	0.5%
The Python Foundation	15 000	0.3%
Итого:	4 797 000	100

Из вышесказанного следует, что задача уменьшения размера и сложности разрабатываемого продукта в рамках модели разработки программного обеспечения с открытым кодом решена успешно и эффективность модели доказана.

Совершенствование процесса разработки.

Совершенствование процесса разработки невозможно без количественного анализа процесса. Для проектов с открытым кодом оценка размера создаваемого продукта не представляет значительных трудностей и может быть выполнена с помощью известных методов [11]. Одновременно оценки стоимости, трудоемкости и времени разработки вызывают трудности. Источником трудностей в оценке являются:

- приток и отток участников проекта, в том числе индивидуумов и некоммерческих организаций, что затрудняет оценку стоимости ввиду неопределенности размера проектной команды;
- неадекватность трудоемкости реальных проектов ее оценкам по существующим моделям.

Одним из подтверждений отсутствия надежных оценок стоимости проектов является проект разработки компилятора Ada, потребовавший в 10 раз меньше инвестиций чем другие подобные проекты.

Другим подтверждением являться результат сравнения реальных затрат для двух проектов с открытым кодом Quanta и KDevelop с их оценкой согласно модели СОСОМО [46]. Результаты сравнения приведены на рис. 1.7.

Приведенные выше данные о существенной погрешности (до 80%) имеющихся моделей оценки времени, трудоемкости и стоимости подтверждают необходимость исследований в области оценок этих процессов разработки.

Оценка трудоемкости (количества труда, использованного при выполнении программного проекта), длительности и стоимости выполняется на ранних стадиях планирования проекта, непосредственно после оценки размера создаваемого в проекте продукта (программы).

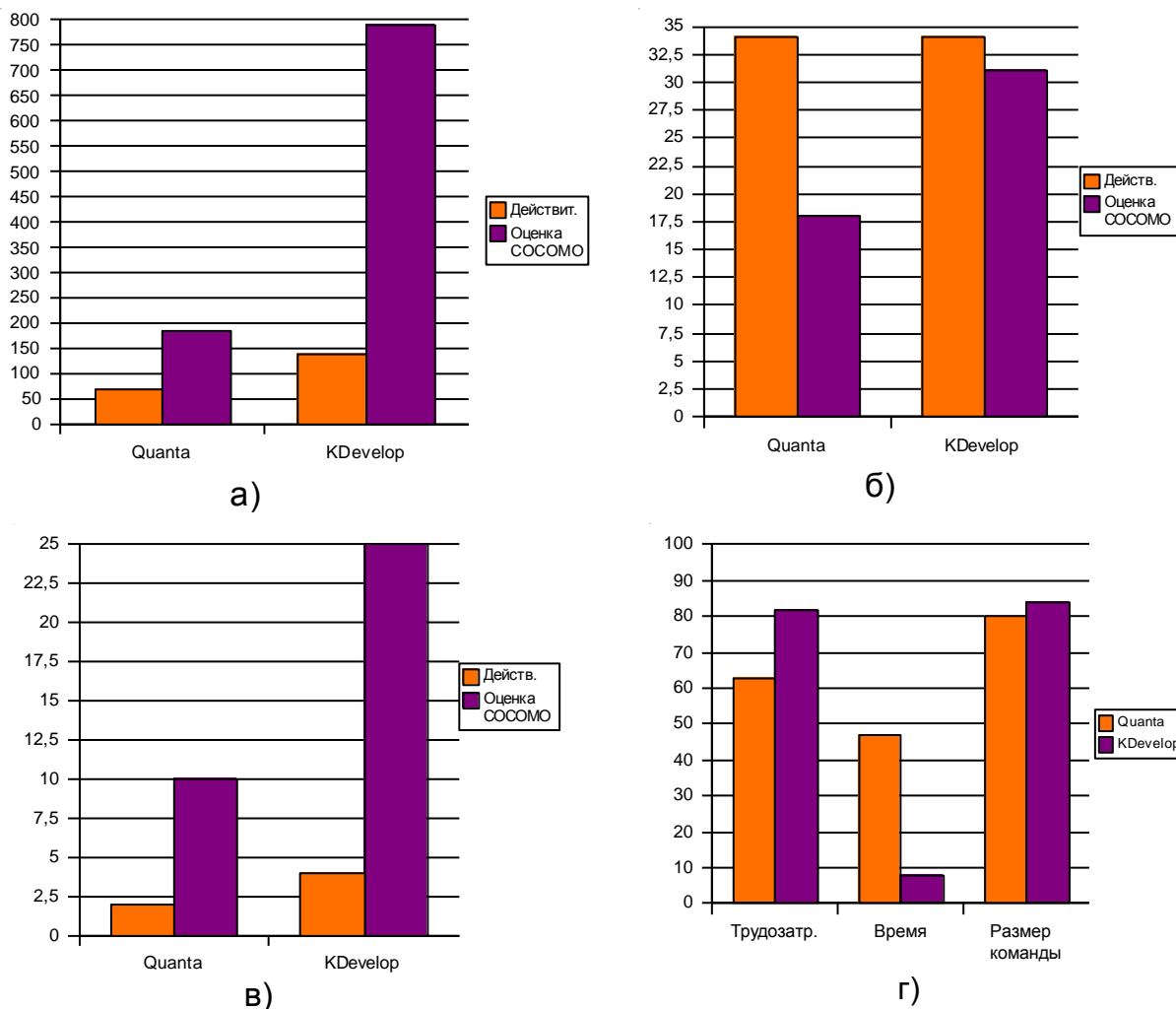


Рис. 1.7. Сравнительный анализ оценок модели СОСОМО и реальных проектных данных для проектов с открытым кодом:

- а) Оценка трудоемкости (чел*мес); б) Оценка времени (мес);
 в) Оценка размера команды, чел; г) Средняя ошибка оценок СОСОМО, %.

Место выполнения оценок можно увидеть на рис. 1.8. Такая оценка выполняется на фазах исследования концепции, изучения системы, а также на фазе формирования требований. Результаты оценки трудоемкости кладутся в основу процесса дальнейшего менеджмента ресурсов, а именно, составления графика и планирования остальных затрат программного проекта. Очевидно, что точность планирования будет напрямую зависеть от точности выполнения оценок трудоемкости, длительности и стоимости.

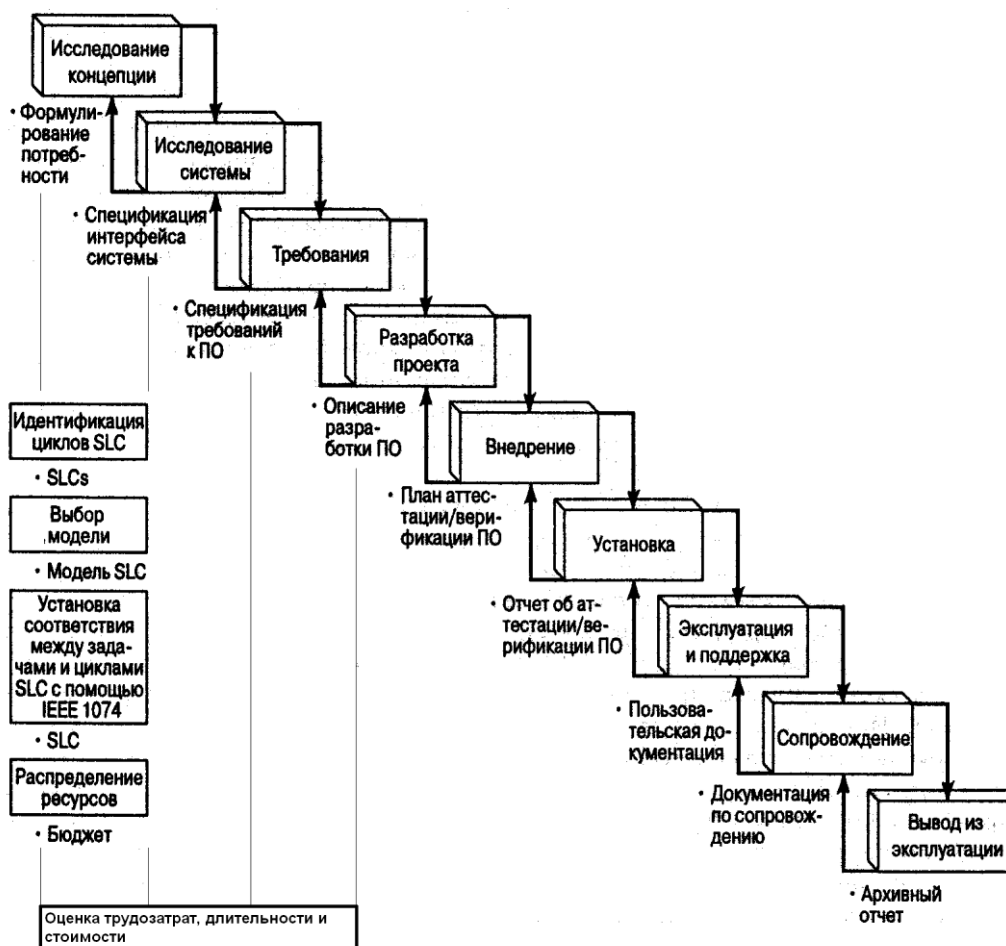


Рис. 1.8. Место оценок стоимости и длительности при выполнении программного проекта

На длительность и стоимость выполнения программного проекта будет влиять огромное количество факторов. Так, например, в [77] перечисляются следующие типичные факторы (полный список привести практически невозможно):

- сложность выполняемых задач, включенных в WBS (Work Breakdown Structure, структура работ проекта), а именно задач по разработке ПО (проект, код, тестирование), дополнительных задач по разработке (требования, система), задач поддержки (СМ, QA, менеджмент), задач, требующих дополнительных трудозатрат (документы);
- объем сопутствующих и дополнительных затрат (поездки, оборудование и т.д.);
- размер программного продукта;

- хронологические данные по затратам и производительности;
- график высокого уровня (обобщенный предварительный график разработки);
- процесс и методы разработки;
- язык программирования;
- платформа для целевой системы;
- используемые инструменты;
- уровень профессионального опыта.

Иными словами, как длительность, так и стоимость создания программного продукта будут результатов воздействия огромного количества эффектов, причем многие из них зачастую сложно измерить количественно.

Подытоживая, можно очертить проблематику выполнения оценок следующими двумя положениями:

1. необходимость учета воздействия огромного количества факторов;
2. трудность определения количественной меры тех или иных эффектов.

Естественно предположить, что задача установления связи между величинами, выбранными в качестве количественной меры воздействующих эффектов будет не только сложна, но и трудновыполнима на научном уровне. И действительно, в рекомендациях по применению методов оценки [62] говорится, что оценка размера и затрат не относится к области точных наук.

В настоящее время для оценки трудозатрат применяются эмпирические, регрессионные (COCOMO [9, 11]) и математические модели (SLIM [78]), а также экспертные оценки (Delphi [79], Wideband Delphi [80]).

Эмпирические зависимости как в дисциплине управления проектами, так и в других областях науки, представляющие собой отношения по сути случайного характера, не являются основой изучения процесса. Так, они могут достоверно охарактеризовать уже завершённый процесс разработки какой-либо программного продукта, но не могут предвидеть все эффекты, влияющие на новый процесс. Степень тяжести недостатков эмпирических методов косвенно

подтверждается тем фактом, что в современной практике управления программными проектами они не находят применения.

Регрессионные модели, предоставляющие экспоненциальные уравнения для создания количественного прогноза, в отличие от эмпирических предоставляют механизмы калибровки и подгонки моделей под процессы в конкретной организации, выполняющей разработку. Здесь стоит отметить, что процесс калибровки носит, во первых, характер преодоления частных трудностей применения моделей, нежели концептуальных, потому как поиск взаимосвязи между факторами производится путем всего-лишь статистической интерпретации хронологических данных, а не путем выявления существенных закономерностей. Кроме того, не делается различий между типами (классами) разрабатываемых продуктов.

Имеющиеся математические модели отличают общие трудности при преодолении уже упомянутой проблемы определения количественной меры эффектов, влияющих на оцениваемые параметры. В результате чего, математические модели становятся усложненными, одновременно не отражая адекватно изучаемые процессы. В [62], например, отмечается, что при использовании модели часто получается так, что общая сумма трудозатрат при выполнении группы малых проектов будет меньшей, чем трудозатраты при выполнении большого проекта, вследствие чего приводится рекомендация проявлять осторожность при применении математических моделей при разбиении больших проектов на меньшие структурные единицы. Йордон в [102] также указывает на преимущества экспертных оценок перед математическими.

Общей проблемой для всех трех вышеперечисленных типов моделей является точность моделирования на ранних (и наиболее важных с точки зрения планирования) фазах выполнения проектов, ошибка которых может достигать до 400%, что иллюстрирует график из [9] приведенный на рис. 1.9. Как видно, в каждой новой фазе жизненного цикла степень достоверности оценки повышается, но наиболее важные предварительные оценки будут тем не менее иметь невысокую степень достоверности.

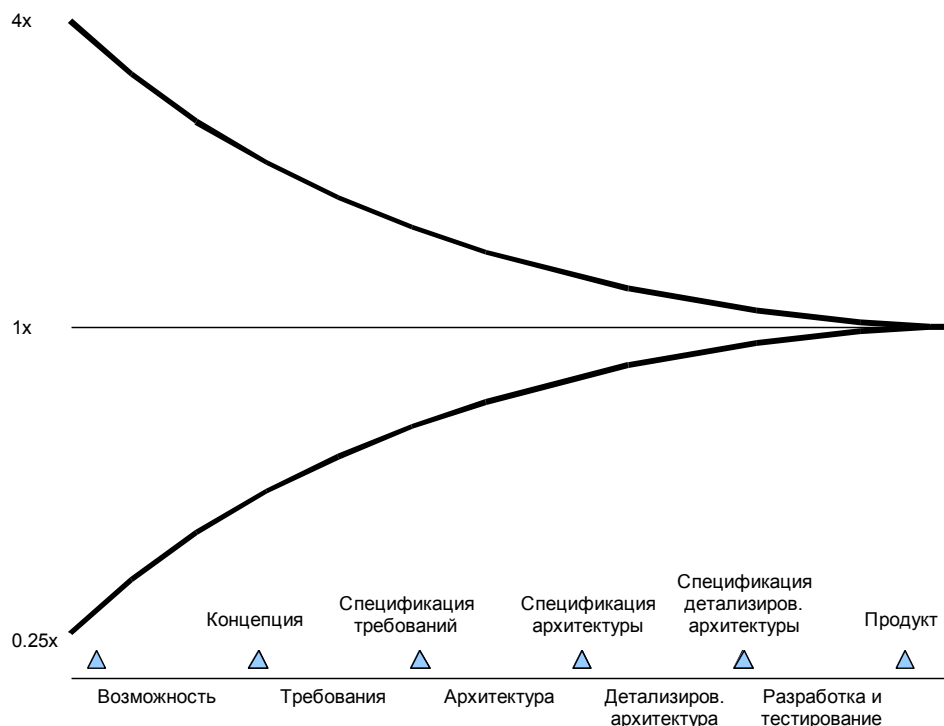


Рис. 1.9. Точность оценок модели COSOMO в зависимости от фазы, в которой производится оценка

Также модели построены и калиброваны в допущении, что основным фактором, влияющим на стоимость, является размер разрабатываемого программного продукта [9]. Влияние других параметров учитывается только поправочными коэффициентами, определенными с помощью методов статистического анализа.

Как видно из вышеизложенного, используемая методология оценивания показателей процессов выполнения программных проектов обладает следующими существенными недостатками:

- не учитывается влияние класса (типа) разрабатываемого программного продукта на значение оцениваемых показателей;
- точность моделирования на ранних этапах выполнения проектов является невысокой.

В то же время, математические модели обладают существенными достоинствами [9], такими как повторяемость, легкость анализа, простота

применения и др., которые не присущи методам экспертных оценок, поэтому вопрос преодоления вышеозначенных методологических трудностей является весьма важным. Иными словами, должен быть предложен способ оценки показателей процессов выполнения программных проектов, преодолевающий указанные выше существенные недостатки.

В [81] указывается, что идея аналогии как основы исследования есть весьма плодотворной ввиду возможности охватить в одном количественном представлении закономерности поведения системы. Иными словами, необходимо обратиться к аналоговому моделированию, как к аппарату обобщения "опытных данных". Руководство РМВОК [27] также указывает на преимущества оценок по аналогии, хотя и не предлагает конкретных методов, а в [9] описывается простой метод оценок по аналогии. Однако, наиболее совершенный и проверенный на практике аппарат аналогового моделирования представляет теория подобия, основные положения которой были разработаны еще в начале прошлого века и наиболее полно обобщены А.А. Гухманом [82].

Как указано в [83], теория подобия является научной базой постановки экспериментов и аналитической обработки их результатов, позволяя обобщать полученные данные в пределах классов и групп процессов, систем или явлений. Таким образом, применение теории подобия для оценки показателей процессов выполнения программных проектов позволяет решить один из существенных недостатков имеющихся методологий оценки: разделение на классы и обобщение опыта выполнения похожих проектов.

Основная идея теории подобия заключается в том, что в пределах некоторого класса явлений или процессов выделяются группы, в которых возможно обобщение данных единичного опыта [82].

Иными словами, обобщение опыта выполнения программного проекта можно выполнить путем создания аналоговой модели процесса выполнения, т.е. системы, отражающей и воспроизводящей структуру, свойства, взаимосвязи и отношения оригинала [83]. Теория подобия в данном случае выступает как математический механизм аналогового моделирования, который предполагает

определение множества безразмерных признаков (критериев подобия), характеризующих свойства системы и определение соотношения между этими признаками (критериями) в форме произведения их степеней на постоянные множители [82].

Такое соотношение будет иметь следующий общий вид:

$$\text{определяемый критерий} = f \left[\text{определяющие критерии} \right].$$

Эти модели имеют следующие основные преимущества, решающие проблемы оценки времени и стоимости программных проектов:

- каждому классу подобных проектов будет соответствовать критериальное уравнение с одинаковыми критериями и коэффициентами пропорциональности;
- внутри каждого класса для группы подобных проектов численные значения критериев будут одинаковы, что позволит делать оценку времени новых проектов используя значения критериев старых аналогичных проектов;
- критерии группируют определяющие время параметры, упрощая эмпирическое определение функции отношения между ними и одновременно позволяя учесть в моделях оценки большее количество определяющих параметров.

Использование более квалифицированного персонала или хороших команд.

В проектах с открытым кодом согласно многим исследованиям никогда не наблюдался недостаток квалифицированного персонала. Так, например, результаты опроса разработчиков, проведенного Boston Consulting Group в 2002 г. показывают, что профессиональные программисты со средним стажем 11 лет составляют большинство участников проектов с открытым кодом (см. рис. 1.10). При этом подавляющее их большинство рассматривает эти проекты как стимулирующие интеллектуальную деятельность и приводящие к повышению профессионализма (рис. 1.11).

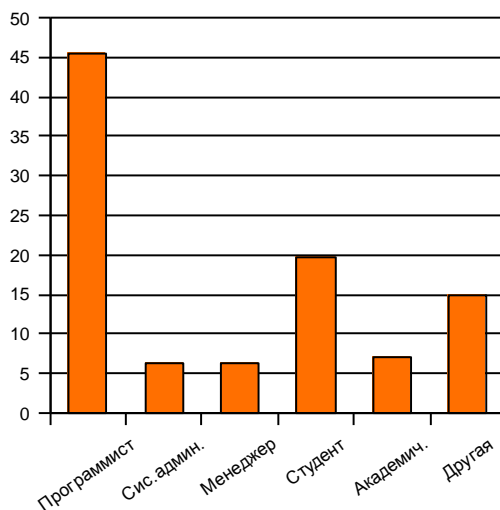


Рис. 1.10. Профессия разработчиков программного обеспечения с открытым кодом согласно опроса VCG.

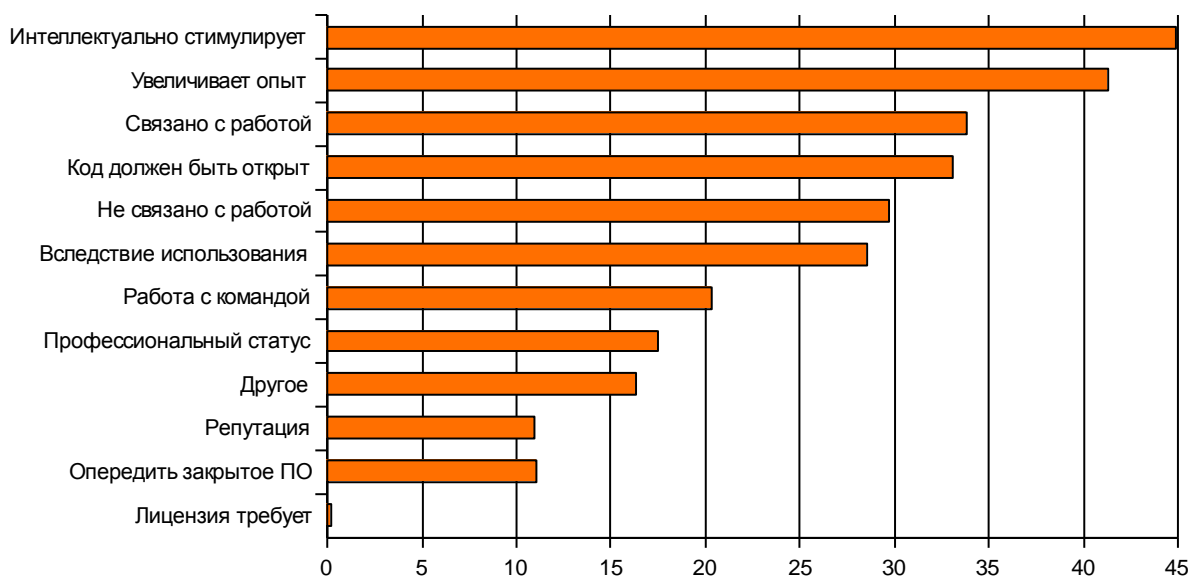


Рис. 1.11. Мотивация участников проектов с открытым кодом согласно опроса VCG.

Таким образом, профессионализм команды, выполняющей проект с открытым кодом, является существенным фактором для успешного завершения проекта.

Другим существенным фактором успеха команд является "самоорганизация" или адаптивное поведение [47, 48, 49, 50]. Под самоорганизацией

подразумевается как особые способы реакции на изменения в среде выполнения проекта [48], так и самоорганизация внутренней структуры команды, выполняющий проект [49].

В проведенных исследованиях основное внимание уделено изучению последствий самоорганизации, а не причинам ее возникновения или механизмам ее обеспечения. Как основные последствия самоорганизации выделяются:

- отсутствие формального менеджмента многих проектов с открытым кодом не вызывающее тем не менее существенных проблем в коммуникации целей;
- отсутствие формального способа назначения задач участникам проекта в условиях постоянного увеличения количества выполняемых и выполненных задач;
- отсутствие специально выделенных команд для коммуникаций с внешним миром при сохранении как объема так и качества этой коммуникации;
- "выживаемость" команды проекта даже при радикальных изменениях условий, позволяющая продолжать выполнение проекта в критической обстановке.

Таким образом, последствия самоорганизации понятны и довольно хорошо изучены, чего нельзя сказать о механизмах. Единственным исследованием механизмов самоорганизации в проектах с открытым кодом является работа Валверде и др. [48], в которой рассматривается самоорганизация в социальных сетях и проводится аналогия между самоорганизацией в пчелином улье и проектной команде.

Валверде предлагает модель социальной сети с положительной обратной связью (рис. 1.12), где в узлах сети располагаются индивидуумы, реакция которых определяется взвешенным произведением входных воздействий, причем вес связи определяется как

$$\langle w_{i,j} \rangle = (k_i k_j)^\theta,$$

где $w_{i,j}$ – вес связи между узлами сети i и j ,

k_i, k_j – реакции узлов i и j ,

θ – показатель степени, определяемый экспериментально для каждой сети.

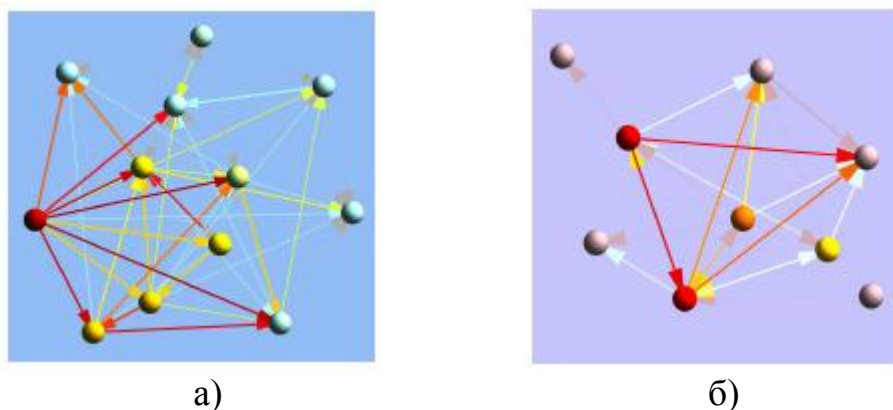


Рис. 1.12. Социальная сеть:

а) пчел в колонии из 13 особей;

б) разработчиков программного обеспечения.

Такая сеть будет самоорганизующейся если $\theta > 0$. В проведенных Валверде эмпирических исследованиях значение показателя степени было 0.39 – 0.5, что доказывало гипотезу про самоорганизацию.

Вышеприведенные исследования позволяют делать выводы о самоорганизации внутри проектной команды, однако не помогают установить причины выживаемости команды при резких изменениях внешних условий. Модель социальной сети, указывая на участников проекта как на источники возникновения самоорганизации, не делает заключения о самом механизме самоорганизации – о его надежности, о параметрах, влияющих на него, и, что наиболее важно для проектного менеджмента, о способах управления этим механизмом.

В связи с этим модель социальной сети нельзя признать удовлетворительной и задачу изучения самоорганизации решенной. Поэтому необходимо разработать такую модель, которая бы объективно отвечала на вопросы о возникновении самоорганизации и ее управлении. То есть необходима такая модель системы

управления проектами, которая бы позволила изучить систему в динамике и выделить набор параметров, управляющих механизмом самоорганизации.

Такая модель может быть построена на основании описания принципа самоорганизации Эшби [57]. Эшби показал, что самоорганизация (адаптация) присуща сверхустойчивым системам. Сверхустойчивой Эшби называет систему из двух систем с непрерывно изменяющимися переменными (среды и реагирующей части), взаимодействующих таким образом, что между ними существует первичная обратная связь и, одновременно, вторичная обратная связь действующая ступенчато и со скоростью более низкого порядка чем первичная. Эта вторая обратная связь идет от среды к некоторым “существенным” переменным, которые влияют на некоторые ступенчатые функции таким образом, что эти функции меняют свои значения тогда и только тогда, когда существенные переменные выходят из заданных пределов. Ступенчатые механизмы воздействуют на реагирующую часть, определяя с помощью ее параметров реакцию на среду.

Под существенными переменными подразумеваются такие переменные, выход которых из границ может привести к прекращению существования реагирующей части системы. Адаптацией будет такое поведение системы, при котором система будет вырабатывать и сохранять реакцию на изменение существенных параметров.

Для количественного анализа сверхустойчивости Эшби предложил в [57] математическую модель сверхустойчивой системы, состоящей из переменных состояния \mathbf{x} и ступенчатых функций \mathbf{w} , причем целое представляет собой систему, определяемую состоянием:

$$\begin{aligned}\dot{\mathbf{x}} &= \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{w} \\ \dot{\mathbf{w}} &= \mathbf{F}\mathbf{x} + \mathbf{G}\mathbf{w}\end{aligned}$$

где \mathbf{x} – вектор переменных состояния,

\mathbf{w} – вектор ступенчатых функций (параметров системы),

\mathbf{C} , \mathbf{D} , \mathbf{F} , \mathbf{G} – матрицы коэффициентов взаимного влияния переменных состояния и параметров системы.

Такое представление позволяет исследовать систему управления программным проектом не только как изолированную систему или систему с постоянным входом, но и как систему, на которую влияют внешние воздействия или на вход которых подаются изменения.

Эти изменения, представленные в модели ступенчатыми функциями w , Эшби называет *параметрами* системы. До настоящего времени ни модели динамики процессов управления программными проектами ни сами эти процессы не учитывали влияние параметров на систему. Это подразумевало, что любое изменение параметров или входных сигналов приводило к необходимости проведения повторных моделирования и исследования системы на устойчивость и внесения коррекций в систему. Применительно к программным проектам это подразумевало выполнение программных проектов при допущении, что среда проекта остается неизменной.

Наличие же сверхустойчивости подразумевает что система сама сможет находить новые поля (области устойчивости) при изменении каких-либо параметров. Эшби обосновал, что такие возможности предоставляет ступенчатое случайное изменение других параметров с целью найти новое устойчивое поле.

Из вышесказанного следует, что применение теории Эшби позволит получить математическую модель системы управления и с использованием известных методов теории систем исследовать устойчивость и самоорганизацию.

Использование лучшей среды (инструментария для автоматизации процесса).

Как уже было сказано в разделе 1.1, по состоянию на 2007г. вопрос среды и инструментария не стоит столь остро и, соответственно, не требует особого внимания. Однако, факт, что более 170 000 проектов с открытым кодом (97% от всего количества проектов) используют услуги репозитория Sourceforge (рис. 1.2) для построения инфраструктуры процесса и создания среды выполнения проекта, требует изучения. В настоящее время разработано огромное количество

инструментальных средств. Правильный их выбор будет влиять на успех или неудачу проекта. Это в особенности касается проектов с открытым кодом и объясняется их сильной зависимостью от эффективности распространения информации [49] и коммуникаций.

Очевидно, задачей такого изучения будет обобщение опыта успешных проектов и его формализация в виде части методики управления, которая отвечает за построение инфраструктуры выполнения процессов проекта.

Достижение уступок и компромиссов для пороговых значений качества.

Программные проекты с открытым кодом отличаются повышенными соотношениями качества к стоимости по сравнению с аналогичными проектами с закрытым кодом. Требуемый уровень качества достигается в проектах с открытым кодом при меньших вложениях средств. Исследования MERIT, проводимые для Еврокомиссии, показывают что в среднем соотношение качество/стоимость для проектов с открытым кодом на 10% выше [34]. Однако, отклонения значений этого соотношения весьма значительны и составляют от -200% до 1000%. В отчете MERIT отмечается, что на величину этого соотношения влияет не столько качество управления проектом, сколько применимость модели открытых кодов вообще, что полностью соответствует тезису Брукса о том, что не существует одного универсального способа решения всех проблем управления программными проектами. В настоящее время четкого способа определения применимости модели не существует. Имеющиеся публикации противоречивы. Одни говорят об универсальной применимости модели [51] другие - о ее полном несоответствии [52, 53].

Таким образом, для решения проблемы достижения максимального качества в условиях поставленных ограничений по стоимости и времени применительно к программным проектам с открытым кодом необходима разработка методики управления, которая бы объективно оценивала применимость модели к конкретному проекту.

1.3. Постановка задач исследования

Целью настоящей работы является улучшение повышение эффективности управления программными проектами с открытым исходным кодом. Как было показано в разделах 1.1 и 1.2 для этого необходимо решить следующие задачи:

1. Разработать модель жизненного цикла программных проектов с открытым кодом, выделить фазы модели и установить перечень действий, которые выполняются в цикле.
2. Разработать математическую модель системы управления программными проектами с открытым кодом и выполнить исследования устойчивости и самоорганизации в системе.
3. Разработать модель оценки времени и стоимости проектов разработки программного обеспечения с открытым кодом.
4. Провести эмпирическое исследование модели оценки времени и стоимости для подтверждения их точности и применимости.
5. Разработать методику управления программными проектами с открытым исходным кодом.

РАЗДЕЛ 2. МОДЕЛИРОВАНИЕ ЖИЗНЕННОГО ЦИКЛА И СИСТЕМЫ УПРАВЛЕНИЯ ПРОГРАММНЫХ ПРОЕКТОВ С ОТКРЫТЫМ КОДОМ

2.1. Разработка модели жизненного цикла проектов с открытым кодом

Описание модели жизненного цикла и ее составляющих необходимо для успешного управления проектами [27]. Формальное описание такой модели состоит из:

- схематического описания для определения фаз и их последовательности;
- диаграмм распределения работ по фазам проекта;
- таблиц идентификации задач и действий, выполняемых в жизненном цикле, служащие для последующего распределения ресурсов проекта.

Детальное формальное описание в таком виде определяет методологию управления проектами, которая обязана выполнять все процессы жизненного цикла согласно стандартам ISO/IEC 12207 [54] и ДСТУ 3918-1999 [55].

Модель жизненного цикла программных проектов с открытым кодом, далее открытая модель, определяется совокупностью общих и специфических признаков. Общие признаки модели определяются как:

- множество фаз выполнения проекта;
- содержание технических работ по каждой фазе;
- распределение инженерных и технических ресурсов по фазам;
- последовательность выполнения обозначенных фаз;
- переходные процессы между фазами.

Специфические признаки открытой модели следующие:

- фазы выполнения проекта организованы в спиралевидной повторяющейся последовательности;
- среда выполнения открытого проекта значительно шире, чем у любых других программных проектов;

- нормальной практикой выполнения проекта является постоянное привлечение сторонних ресурсов как материальных, так и трудовых;
- множество исполнителей проекта не является ни фиксированным ни четко определенным.

2.1.1. Определение открытой модели. Определение фаз открытой модели и их последовательности произведено путем анализа хода выполнения уже завершенных проектов.

Для анализа были выбраны наиболее характерные проекты, представляющие все типы открытых проектов:

- чистые: GNOME, KDE, Linux (ядро);
- конверсионные: Mozilla, Blender, OpenOffice;
- управляемые: Qt, Eclipse, OpenSolaris.

Анализ проводился с помощью:

- хронологических данных о проектах из электронной энциклопедии Wikipedia;
- информации из списков рассылки и веб-сайтов проектов;
- планов выпуска и анонсов.

Результаты анализа приведены в таблице 2.1, где собраны аннотированные хронологические данные об управленческих действиях в проекте KDE. Данные о других проектах показывают во многом аналогичную картину, поэтому приводятся в приложении А.

Аннотацией служила краткая характеристика действия, классифицирующая его как веху (начало либо завершение) определенной фазы жизненного цикла. Аннотирование выполнялось с целью определения фаз жизненного цикла.

Таблица 2.1

**Хронологические данные об управленческих
действиях в открытом проекте KDE**

Дата	Действие	Аннотация
14.10.1996	Письмо [67] основателя проекта Mattias Ettrich с сообщением о начале проекта, перечнем выполняемых задач и целей.	Определение проекта
Октябрь 1996	Создан список рассылки kde@kde.org для разработчиков и пользователей проекта, веб-сайт проекта. Опробована, а затем запущена система управления версиями кода для хранения кода проекта.	Прототипирование и реализация инфраструктуры макропроцесса
Ноябрь 1996	Начальной группой разработчиков используя список рассылки выполнен анализ и обзор существующих альтернатив.	Обзор существующих аналогичных и похожих программных систем
Ноябрь 1996	Определены первостепенные цели проекта.	Определение структуры целей и задач проекта
12.07.1998	Завершена разработка первой версии продукта KDE 1.0.	Прототипирование и разработка
Начало 1999	Введение в действие системы управления запросами на внесение изменений (Bugzilla) как ответ на пожелания пользователей оценки.	Пользовательская оценка, усовершенствование макропроцесса как внедрение целей
1998-1999	Доработка выпущенной версии продукта в соответствии с требованиями пользователей, выпуск обновлений.	Внедрение и улучшение

Продолж. табл. 2.1

Дата	Действие	Аннотация
15.12.1999	Завершено прототипирование следующей версии продукта.	Прототипирование новой версии
23.10.2000	Завершена разработка следующей версии продукта.	Разработка новой версии

Каждая фаза разработки из тех, что аннотированы в таблице 2.1, исходя из данных системы управления версиями и планов выпуска новых версий (приведенных, например, в документах [58, 59, 60]) разделяется на 3 основных этапа. Первый этап – выполнение одновременного прототипирования, кодирования и документирования с целью выпуска нестабильной и не полностью функциональной "α" версии продукта. Следующий этап после выпуска "α" версии включает в себя одновременное кодирование, документирование и модульное тестирование с целью выпуска более стабильной и полнофункциональной "β" версии продукта. И, наконец, в последний этап входит интеграция и квалификационное тестирование продукта с целью выпуска стабильной и полнофункциональной следующей версии продукта (с промежуточными опциональными выпусками версий вида "release candidate", т.е. "почти готовых к выпуску" версий).

По результатам проведенного выше анализа и содержанию проектной информации, представленной в Приложении А, выполнено схематическое представление фаз жизненного цикла открытой модели и определено содержание процессов, выполняемых в каждой из фаз. Такое представление показано на рис. 2.1.

Визуальный анализ модели показывает, что она имеет процессы, присущие каскадной [61] и спиральной [9] модели. Как и в спиральной модели, в нее включены процессы анализа и управления рисками и поддержки менеджмента. Предусмотрена разработка программного продукта при использовании метода прототипирования или быстрой разработки приложений посредством применения

языков программирования и средств разработки четвертого и выше поколения. Одновременно, каждый цикл модели отображает базовую концепцию, которая заключается в том, что цикл представляет собой набор операций, которому соответствует такое же количество стадий, что и в каскадной модели.

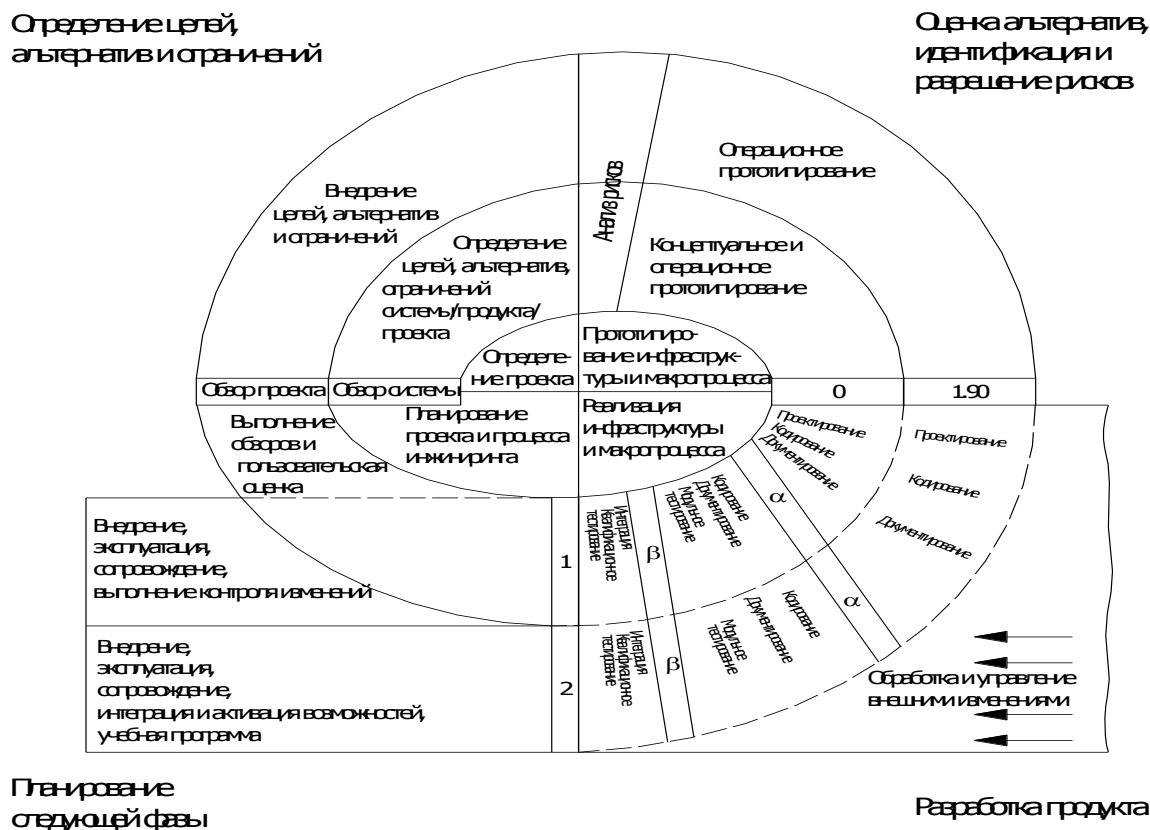


Рис. 2.1. Открытая модель жизненного цикла программных проектов

С другой стороны, открытая модель вносит новую концепцию обработки и управления внешними изменениями. Они объявляются неотъемлемой частью процесса выполнения проекта и их задачами становятся, в первую очередь, решение проблемы выполнимости проекта, а, во-вторых, разрешение рисков, связанных с персоналом и сложностью проекта. Также циклы определения проекта и обзора требований совмещены в открытой модели в целях упрощения ее использования и ускорения процесса выполнения проекта.

Как видно из рис. 2.1, модель делится на квадранты, каждому из которых соответствует цель и каждый из которых состоит из множества фаз для достижения этой цели.

Цели, соответствующие квадрантам модели, могут быть определены как:

- **определение целей, альтернатив и ограничений.** Здесь выполняется определение целей всего проекта, определяются рабочая характеристика, перечень выполняемых функций, решающие факторы достижения успеха, программно-аппаратный интерфейс. Определяются альтернативные способы реализации продукта (частей продукта): повторное использование, разработка, кооперация, приобретение. Определяются ограничения, налагаемые на использование альтернативных вариантов: время, интерфейс, ограничения на использующий продукт, внешнюю среду, и т.д.;
- **оценка альтернатив, идентификация и разрешение рисков.** Выполняется апробация (прототипирование) и оценка альтернатив, определенных в предыдущем квадранте, оцениваются и разрешаются риски, принимается решение о продолжении (либо прекращении) работ над проектом;
- **разработка продукта.** Выполняется, собственно, разработка кода, тестирование, документирование, сборка и выпуск продукта. Здесь продукт попадает к заказчику, его (продукта) жизненный цикл трансформируется в цикл с первичной целью улучшения и доработки;
- **планирование следующей фазы.** Поступивший к заказчику продукт, в своем новом жизненном цикле проходит фазы внедрения, сопровождения и эксплуатации, тогда как в жизненном цикле проекта проводится подготовка к выпуску следующей версии продукта. На основании данных о внедрении, сопровождении и эксплуатации выпущенного продукта, а также на основании экспертных оценок и обзоров продукта и процесса разработки выполняется планирование модификации процесса и продукта, внедряемых на следующем цикле.

2.1.2. Процессы управления проектом. Диаграммы распределения работ по фазам жизненного цикла. Согласно [27] процессы делятся на две категории –

процессы, ориентированные на продукт, сосредоточенные на определении и создании продукта проекта и процессы управления проектами, которые описывают и упорядочивают работы в проекте. Процессы, ориентированные на продукт были определены выше в формальном описании жизненного цикла. Процессы же управления проектом должны быть описаны отдельно, но при условии временного наложения и взаимосвязи с процессами, ориентированными на продукт.

Для описания процессов управления проектами, предлагается особый вид диаграмм (см. рис. 2.2), в которых первые два цикла спирали открытой модели жизненного цикла линейризируются и представляются как последовательность фаз. Для каждой фазы указывается сгруппированные по процессам управления проектами диаграммы распределения работ по фазам.

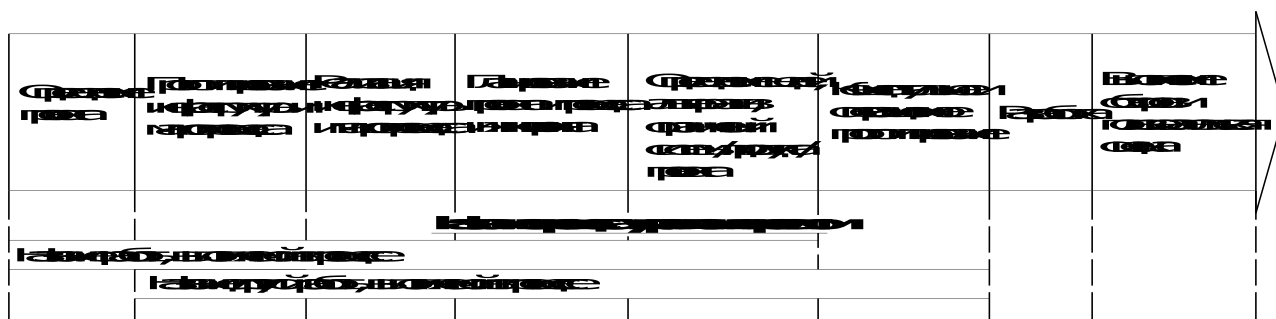


Рис. 2.2. Общий вид диаграммы процессов управления проектом и распределения работ

При разработке программного обеспечения, в том числе с открытым кодом, выделяются [62] несколько основных процессов управления, соответствующих 5-ти группам процессов управления, определенных в [27]. Остальные процессы, относящиеся к продукту, и процессы управления ресурсами, которые выполняются в течение всего жизненного цикла, рассматривать на диаграммах не целесообразно

Определение цели и области действия программного проекта. Этот процесс воплощает процессы инициализации и планирования. При выполнении проекта определяется одна или несколько целей, достижение которых закладывается в

плане менеджмента (SPMP, Software Project Management Plan – План управления программным проектом) при условии ограничений по ресурсам и достигаемого качества. План менеджмента и техническое задание проекта определяют методологическую сторону среды проекта, а техническая сторона определяется жизненным циклом.

Графически процесс представлен на рис. 2.3.

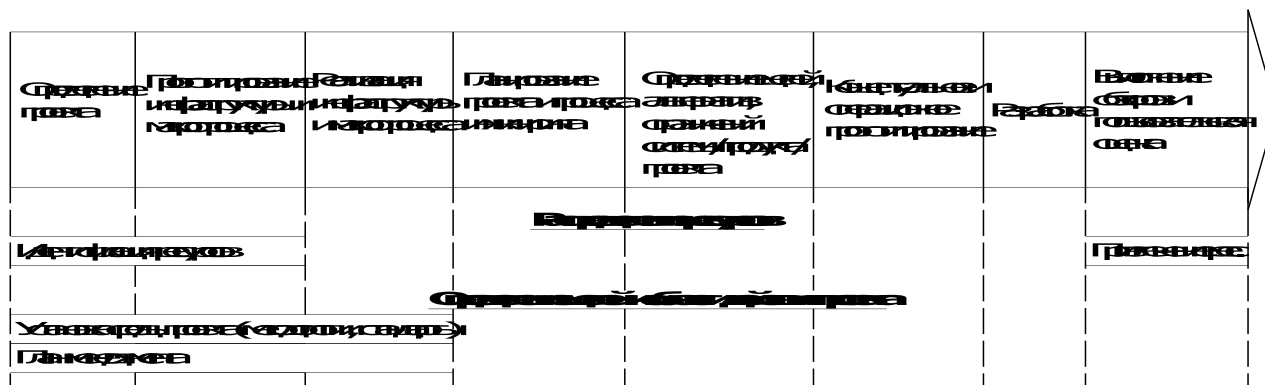


Рис 2.3. Определение цели и области действия программного проекта

Отбор и управление командой разработчиков. Отбор команды – это наиболее важный процесс из группы процессов выполнения открытого проекта. Обычно селекция команды, формирование коллектива разработчиков оказывает немалое влияние на остальные действия жизненного цикла разработки программного обеспечения. В открытой модели важным является предварительное планирование и формальное описание процесса присоединения сторонних разработчиков. Выполнение этого процесса и менеджмент стороннего персонала есть неотъемлемая часть фаз разработки в открытом проекте.

Графически процесс представлен на рис. 2.4.

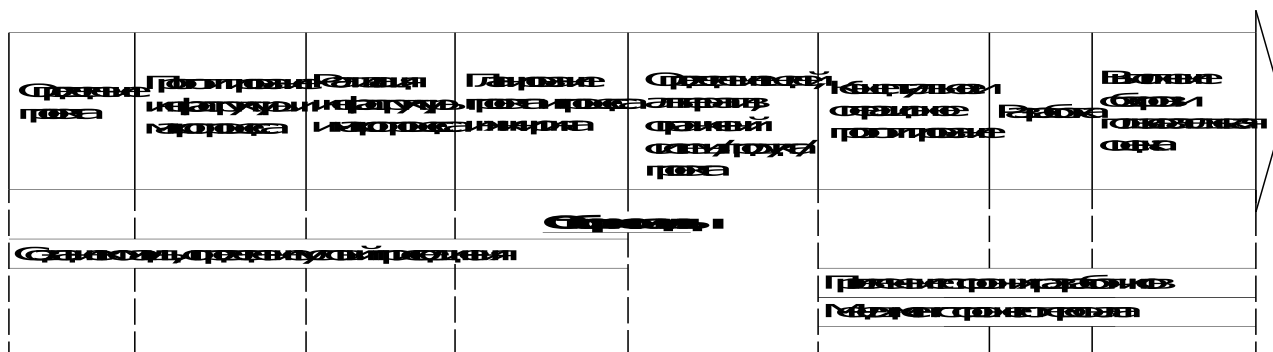


Рис. 2.4. Отбор команды разработчиков

Обеспечение надежности. Этот процесс принадлежит большой группе процессов осуществления контроля за проектом. Процессу уделяется особое внимание ввиду его важности для конечного пользователя, и он напрямую влияет на успешность проекта. Процесс создания надежного программного обеспечения должен включать в себя, согласно [62] прогнозирование, предотвращение и устранение ошибок, а также реализацию методик обеспечения устойчивости к отказам.

Графически процесс представлен на рис. 2.5.

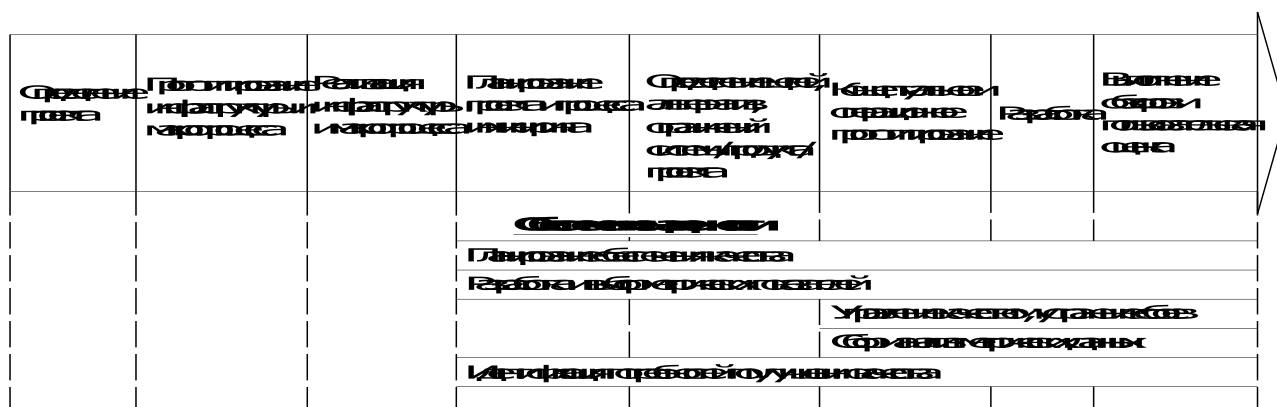


Рис. 2.5. Обеспечение надежности.

2.1.3. Идентификация задач и действий, выполняемых в жизненном цикле.

Проход №1. Цикл определения проекта.

Квадрант №1. Определение целей, альтернатив и ограничений – определение требований и спецификаций для системы в целом и ее критических частей в частности с точки зрения производительности, функциональных свойств, способности к аккомодации изменений, программного и аппаратного интерфейса, критических факторов успеха и т.д.

Фаза №1. Определение проекта – разработка предварительной цели и плана проекта, с помощью которых приближенно описываются графики и поставляемые продукты.

Действия и задачи, выполняемые на фазе определения проекта приведены в таблице 2.2.

Таблица 2.2

Действия и потенциальные задачи фазы определения проекта

Действия	Задачи
начало выполнения проекта	распределение ресурсов проекта
	управление планом проекта
исследование концепции	идентификация идей либо потребностей
	формулирование потенциальных подходов к удовлетворению потребностей

Квадрант №2. Оценка альтернатив, идентификация и разрешение рисков – оценивание альтернативных форм макропроцесса исходя из целей и ограничений.

Фаза №2. Прототипирование инфраструктуры и макропроцесса.

Действия и задачи, выполняемые на этой фазе приведены в таблице 2.3.

Таблица 2.3

**Действия и потенциальные задачи фазы
прототипирования инфраструктуры и макропроцесса**

Действия	Задачи
прототипирование среды проекта	обзор сред конфигурирования проекта
	установка и апробация сред конфигурирования проекта

Квадрант №3. Разработка продукта – создание инфраструктуры и макропроцесса.

Фаза №3. Реализация инфраструктуры и макропроцесса

Действия и задачи, выполняемые на этой фазе приведены в таблице 2.4.

Таблица 2.4

**Действия и потенциальные задачи фазы реализации
инфраструктуры и макропроцесса**

Действия	Задачи
установка среды проекта	выбор и установка среды конфигурирования проекта
создание прототипа системы	реализация демонстрационного прототипа, реализующего основные системные концепции на установленной среде проекта

Квадрант №4. Планирование следующей фазы – применение информации о ходе выполнения проекта к планированию следующего цикла.

Фаза №4. Планирование проекта и процесса инжиниринга

Действия и задачи, выполняемые на этой фазе приведены в таблице 2.5.

Таблица 2.5

**Действия и потенциальные задачи фазы планирования проекта
и процесса инжиниринга**

Действия	Задачи
повторное планирование управления проектом	повторное формулирование потенциальных подходов
	уточнение идей либо потребностей
приемка среды проекта	апробация демонстрационного прототипа, реализующего основные системные концепции

Проход №2. Цикл реализации проекта.

Квадрант №1. Определение целей, альтернатив и ограничений

Фаза №5. Определение целей, альтернатив и ограничений системы/продукта/проекта – определение требований и спецификаций для

критических частей системы с точки зрения производительности, функциональных свойств, способности к аккомодации изменений, программного и аппаратного интерфейса, критических факторов успеха и т.д.

Действия и задачи, выполняемые на этой фазе приведены в таблице 2.6.

Таблица 2.6

Действия и потенциальные задачи фазы определения целей, альтернатив и ограничений системы/продукта/проекта

Действия	Задачи
системный анализ	анализ функций на уровне системы/продукта
	разработка системной архитектуры
	декомпозиция системных требований
специфицирование требований к продукту	уточнение и разработка требований к ПО
	определение требований к интерфейсу
	расстановка приоритетов и интеграция программных требований на уровне системы / продукта

Квадрант №2. Оценка альтернатив, идентификация и разрешение рисков

Фаза №6. Анализ рисков

Действия и задачи, выполняемые на этой фазе приведены в таблице 2.7.

Таблица 2.7

Действия и потенциальные задачи фазы анализа рисков

Действия	Задачи
идентификация рисков	определение источников возникновения рисков

Фаза №7. Концептуальное и операционное прототипирование – уточнение требований и спецификаций для потенциально наиболее критичных частей системы с целью изучения степени выполнимости этих частей.

Действия и задачи, выполняемые на этой фазе приведены в таблице 2.8.

Таблица 2.8

Действия и потенциальные задачи фазы концептуального и операционного прототипирования

Действия	Задачи
изучение выполнимости	выполнение имитаций
	создание и выполнение сравнительных тестов
исследование критичных частей системы	эскизное проектирование критичных частей продукта
	прототипирование и реализация критичных частей продукта
идентификация дальнейших требований к продукту	уточнение и дальнейшая разработка требований
	определение требований к интерфейсу

Квадрант №3. Разработка продукта

Фаза №8. Разработка продукта "α" – одновременное проектирование, кодирование, и документирование с целью выпуска нестабильной и функционально незавершенной версии "α".

Действия и задачи, выполняемые на этой фазе приведены в таблице 2.9.

Таблица 2.9

Действия и потенциальные задачи фазы разработки продукта "α"

Действия	Задачи
проектирование	разработка проекта архитектуры
	разработка проекта интерфейса
кодирование	создание исходного кода
	генерирование объектного кода
документирование	генерация оперативной проектной документации

Фаза №9. Разработка продукта "β" – одновременное кодирование, документирование и модульное тестирование с целью выпуска нестабильной, но функционально завершенной версии "β".

Действия и задачи, выполняемые на этой фазе приведены в таблице 2.10.

Таблица 2.10

Действия и потенциальные задачи фазы разработки продукта "β"

Действия	Задачи
кодирование	создание исходного кода
	генерирование объектного кода
документирование	генерация оперативной проектной документации
модульное тестирование	разработка и сопровождение плана тестирования
	разработка тестовых требований
	создание тестовых данных
	выполнение тестов

Фаза №10. Разработка следующей версии продукта – интеграция и квалификационное тестирование с целью выпуска следующей версии продукта.

Действия и задачи, выполняемые на этой фазе приведены в таблице 2.11.

Таблица 2.11

Действия и потенциальные задачи фазы разработки следующей версии продукта

Действия	Задачи
интеграция	разработка и сопровождение плана интеграции
	выполнение интеграции
квалификационное тестирование	разработка и сопровождение плана тестирования
	выполнение тестов

Фаза №11. Обработка и управление внешними изменениями

Действия и задачи, выполняемые на этой фазе приведены в таблице 2.12.

Таблица 2.12

Действия и потенциальные задачи фазы обработки и управления внешними изменениями

Действия	Задачи
управление внешними изменениями	создание и сопровождение документов, регламентирующих возможность и процедуру внесения внешних изменений
	управление и отслеживание внесенных изменений
обработка внешних изменений	отбор изменений согласно критериям качества
	интеграция изменений
	корректировка изменений

Квадрант 4. Планирование следующей фазы

Фаза №12. Выполнение обзоров и пользовательская оценка.

Действия и задачи, выполняемые на этой фазе приведены в таблице 2.13.

Таблица 2.13

Действия и потенциальные задачи фазы выполнения обзоров и пользовательской оценки

Действия	Задачи
анализ структуры системы	уточнение требований к архитектуре системы
идентификация исправленных требований	проведение пользовательской оценки
	управление результатами пользовательской оценки
	управление базой данных запросов на изменения и исправления

Фаза №13. Внедрения, эксплуатации, сопровождения и выполнения контроля изменений.

Действия и задачи, выполняемые на этой фазе приведены в таблице 2.14.

Таблица 2.14

Действия и потенциальные задачи фазы внедрения, эксплуатации, сопровождения и выполнения контроля изменений

Действия	Задачи
установка производственной системы	создание и сопровождение плана установки
	распределение ПО
	установка ПО
	приемка ПО
эксплуатация и поддержка	выполнение операций в системе
	обеспечение технической помощи и выполнение консультаций
	поддержка журналов запросов о помощи
процесс сопровождения	анализ запросов
	внесение изменений

2.1.4. Соответствие модели стандартам ISO/IEC 12207 и ДСТУ 3918-1999. В настоящее время регламентирующими стандартами в области жизненных циклов программных продуктов являются международные стандарты ISO/IEC 12207 "Software Lifecycle Processes" [54] и ДСТУ 3918-1999 "Процессы жизненного цикла программного обеспечения" [55].

Вышеперечисленные стандарты определяют содержание жизненного цикла. Форма, последовательность процессов и временные рамки не регламентируются. Также стандарт не требует формального признания соответствия. Соответствие определяется как выполнение определенных стандартами процессов, действий и задач, специально подогнанных под нужды конкретной организации [63].

Подобные гибкие требования позволяют использовать наиболее современные жизненные циклы при сохранении соответствия стандарту [64, 65].

В то же время важно применение общего языка [66], определяемого стандартом, для обозначения основных процессов жизненного цикла программного обеспечения. Адаптация к стандарту является средством идентификации бизнес-процессов предприятий и интеграции бизнес-процессов множества предприятий, участвующих в проекте разработки программного обеспечения [68].

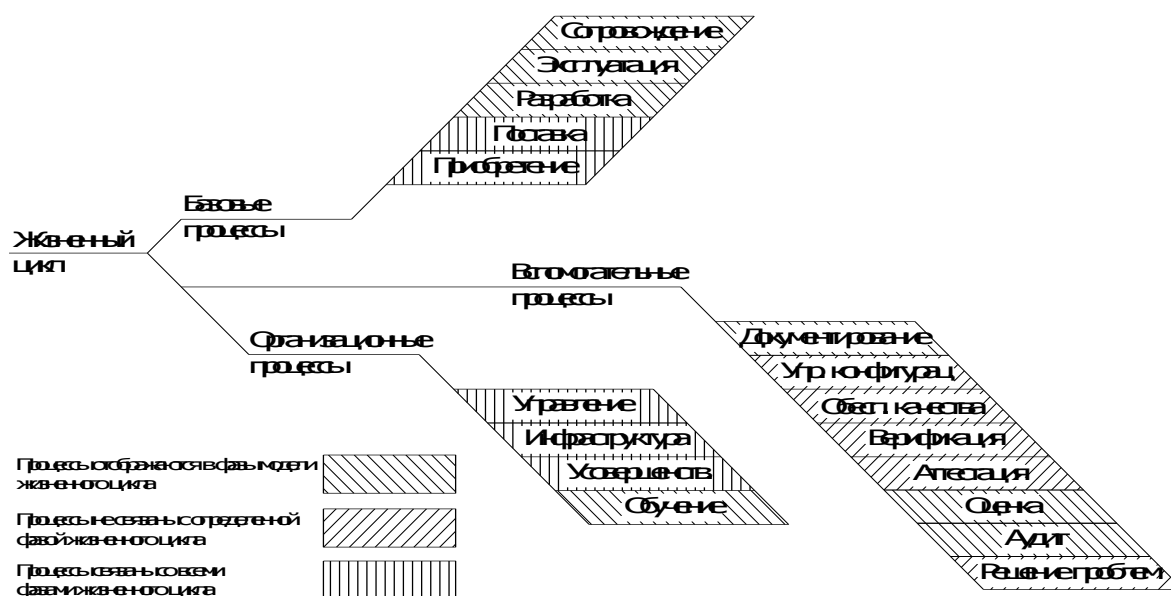


Рис. 2.6. Место процессов жизненного цикла по стандартам ISO/IEC 12207 и DSTU 3918-1999 в открытой модели жизненного цикла

Иными словами, жизненный цикл должен определять и реализовывать процессы, описанные стандартами для выполнения соответствия. Диаграмма, показывающая место определенных стандартами процессов в жизненном цикле, показана на рис. 2.6. Из диаграммы следует, что все такие процессы находят свое отражение в модели жизненного цикла, а, следовательно, будут выполнены при выполнении проекта. Поэтому открытая модель жизненного цикла соответствует стандартам ISO/IEC 12207 и DSTU 3918-1999.

2.1.5. Связь открытой и быстрой моделей. Открытая модель жизненного цикла имеет многие характерные черты обобщенной (каскадной) и спиральной моделей. Это сходство наблюдается только по форме организации фаз и перечню

действий, но не по содержанию действий, выполняемых в этих фазах. В отличие от обобщенной и спиральной моделей, открытая модель регламентирует только перечень действий, не регламентируя их содержание, которое определяется скорее практикой выполнения открытых проектов, а не формальным документом. На практике действия открытой модели отличаются существенно меньшим объемом производимой в них формальной проектной документации, отсутствием или значительным ослаблением требований к процессу проектирования разработки и документирования, отсутствием контрольных точек и обзоров по окончанию фаз, кроме фазы выполнения обзоров и пользовательской оценки.

Таким образом методологически открытая модель существенно отличается от обобщенной и спиральной.

Одновременно отмеченная выше характерная особенность низкого уровня формализации открытых проектов присуща и так называемым «быстрым» моделям [12, 13], объединенных общими принципами, определенных в манифесте быстрой разработки программных систем [69, 70]. Можно отметить сходство следующих принципов открытых и быстрых проектов:

1. Индивидуумы и взаимодействия между ними ценятся выше процессов и инструментов.

Открытые проекты возникли не как результат деятельности исследовательских групп или организаций по стандартизации а как результат практической деятельности прежде всего индивидуумов и команд, ценящих прежде всего взаимодействие и сотрудничество.

2. Работающее ПО ценится выше всеобъемлющей документации.

Преимущественной задачей открытых проектов также ставится выполнение задач, а не документирование процесса.

3. Сотрудничество с заказчиками ценится выше формальных договоров.

Открытые проекты основываются на предположении о высокой степени сотрудничества с пользователем, более того наилучшие условия для открытого проекта обеспечиваются при раннем и активном участии пользователей в разработке.

4. Реагирование на изменения ценится выше строгого следования плану.

План в открытом проекте создается для описания процедуры реакции на изменения, таким образом предвосхищая и привлекая их.

Кроме того, в открытых проектах используются такие принципы быстрой разработки как [71] упрощенное проектирование, подразумеваемое знание, тест-ориентированная разработка, рефакторинг, быстрый цикл разработки и поставки и другие.

Все вышеперечисленное позволяет заключить, что открытая модель разработки ПО является аналогом быстрой, т.к. соответствует всем принципам организации быстрых проектов. Следовательно, открытая модель разработки будет применима в тех случаях, где применима быстрая модель, т.е. в проектах с низкой и средней критичностью [12, 70] когда выполняются два условия:

- дефекты в производимом продукте вызывают потерю удобства;
- дефекты вызывают потерю возместимых средств (материальных или финансовых).

Несмотря на то, что быстрые методологии применимы для проектов малого и среднего масштаба (от 1 до 20 разработчиков), практика показывает применимость открытой методологии для проектов большого масштаба. Например, Linux Kernel – ок. 50 основных разработчиков, более 1000 внешних, KDE – ок. 70 основных разработчиков, более 1200 внешних.

Применимость открытой модели для конкретного проекта при условии соответствия критичности проекта требуемым, можно определить методами анализа отличительных категорий или анализа риска, приведенных в подразделах 5.3 и 5.4.

2.2. Разработка модели динамики системы управления проектами с открытым кодом

Модель жизненного цикла и его составляющих фаз и действий позволяет изучить вопросы управления открытыми программными проектами в статике. Для

определения насколько удачна модель, какой устойчивостью обладают системы управления открытыми программными проектами, какие ограничения на параметры системы накладываются моделью, необходимо исследовать динамику этой системы.

Система управления открытыми проектами является сложной нестационарной нелинейной системой, что значительно затрудняет ее анализ. Как показано Абдель-Хамидом возможно построить упрощенную, но не менее точную модель динамики управления программными проектами [56]. Согласно вышеприведенному исследованию, модель динамики позволяет:

- исследовать систему управления как социо-техническую систему, интегрируя знания по управлению персоналом и знания по разработке программ в одну целое;
- использовать принцип обратной связи для структурирования и прояснения сложных взаимосвязей между переменными системы.

Системный подход к детальному анализу динамики процессов, протекающих в открытых программных проектах, также позволит теоретически обосновать явление самоорганизации, наблюдаемое во многих программных проектах с открытым кодом [47, 48] и указать на его истоки. При этом теория сверхустойчивых адаптивных систем Эшби [57] позволит понять сущность и структуру самоорганизации и выделить дополнительные параметры и связи, присутствующие в открытых проектах и не нашедшие отражение в оригинальной модели динамики выполнения программных проектов Абдель-Хамида. Исследование такой системы на устойчивость позволит выяснить как дополнительные механизмы самоорганизации влияют на проект и насколько более устойчивыми являются системы управления открытыми программными проектами. Для этого необходимо построить математические модели как системы как без самоорганизации, так с ней и, получить вероятность устойчивости системы и показать приводит ли самоорганизация к поиску нового устойчивого состояния системы при выведении ее параметров за границы устойчивости.

Таким образом, моделирование динамики должно состоять из следующих этапов:

1. построение математической модели изолированной системы;
2. получение критерия устойчивости системы без самоорганизации;
3. исследование вероятности устойчивости системы;
4. расширение модели для случая самоорганизации;
5. исследование механизма поиска нового устойчивого состояния в системе с самоорганизацией.

2.2.1. Система управления открытого проекта как самоорганизующаяся сверхустойчивая система. В предыдущем разделе было указано на методологическую общность открытой и быстрой модели. Несмотря на это, существуют два отличия между ними – отличия в методах оценки стоимости и времени [62] и отличия в принципах управления командами [12]. Методические проблемы оценки стоимости подробно рассматриваются в разделе 3. В этом разделе решаются вопросы управления командами.

Основатели быстрой методологии Коберн, Бек и Хайсмит считают, что успех быстрой разработки определяется гибкой организацией команд, выполняющих проект. Основные характеристики, присущие гибким командам, следующие [12, 13, 14]:

- наименьшее количество разработчиков, необходимое для выполнения проекта;
- наименьшее возможное расстояние между участниками проекта (предпочтительное размещение команды – одно помещение);
- представители пользователей участвуют в разработке и, предпочтительно, доступны постоянно;
- одномесячные циклы инкрементальной разработки;
- наиболее опытные разработчики;
- самоорганизация команды.

В открытых же проектах, наблюдаются следующие принципы функционирования команд:

- неограниченное количество разработчиков, разделенных на две группы – основных и внешних. Основные разработчики (в соответствии с принципом Парето) разрабатывают 80% процентов всего кода в проекте и владеют его архитектурой. Внешние разработчики работают над частичными улучшениями, вносят дополнительные изменения и функциональность, исправляют ошибки с низким уровнем критичности, выполняют документирование и другие работы по сопровождению инфраструктуры проекта;
- виртуальная организация работы и децентрализация, в которой участники проекта не обязаны находиться рядом географически. Такая организация, однако выдвигает дополнительные требования к инфраструктуре для обеспечения свободной и прозрачной (видимой как для участников проекта, так и для внешних наблюдателей) передачи информации;
- в качестве разработчиков привлекаются не только профессионалы, но и специалисты любого уровня. Это правило, однако, не выполняется для оплачиваемых разработчиков.

Тем не менее, идеи участия пользователей в разработке и самоорганизации нашли свое место в открытых проектах. И, если участие пользователей с точки зрения методологии не представляет интереса, то самоорганизация требует изучения.

В быстрых моделях самоорганизация обеспечивается согласно [12] следующими тремя механизмами:

1. Рефлексией – обсуждением процесса разработки членами команды (предпочтительно еженедельным);
2. Настройкой методологии в процессе ее применения – изменениями структуры команд, их географического размещения, введение дополнительных коммуникационных каналов, внедрением лучших инструментальных средств;
3. Ретроспективным обзором – анализом результатов выполнения проекта.

Все три механизма применяются и в открытых проектах, однако содержание второго из них, естественно, будет отличным.

Исходя из вышеизложенного очевидно, что самоорганизация определяется наличием дополнительных механизмов влияния на параметры системы. Для изучения этих механизмов и определения параметров была применена теория адаптивных систем Эшби.

Схематическое изображение сверхустойчивой системы управления программными проектами с открытым кодом приведено на рис. 2.7.

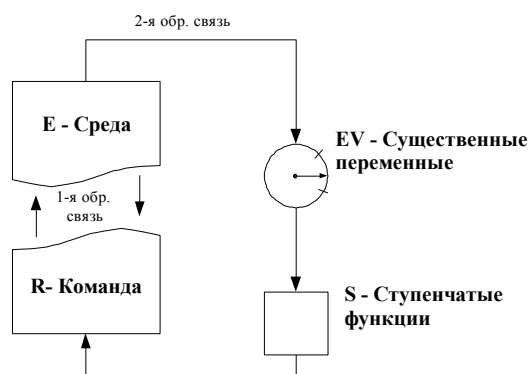


Рис. 2.7. Сверхустойчивая самоорганизующаяся система управления программными проектами с открытым кодом

В результате анализа данных о процессах в открытых проектах, а также архивных данных об успешных и неудачных проектах портала SourceForge [72], было идентифицировано шесть существенных переменных, для каждой из которых в практике управления открытыми проектами были выработаны соответствующие ступенчатые функции.

Области определения этих существенных переменных представлены на рис. 2.8 в виде полярного графа. На графе цветом выделена область допустимых значений существенных переменных, внутри которой ступенчатые функции не активируются. При нахождении этих существенных переменных вне представленного интервала, система потеряет адаптивное поведение.

Сами ступенчатые функции определены в таблице 2.15 как перечень параметров, которые эти функции изменяют при выведении существенных

переменных за границы. Толстыми линиями в таблице выделены диапазоны допустимых значений существенных переменных.

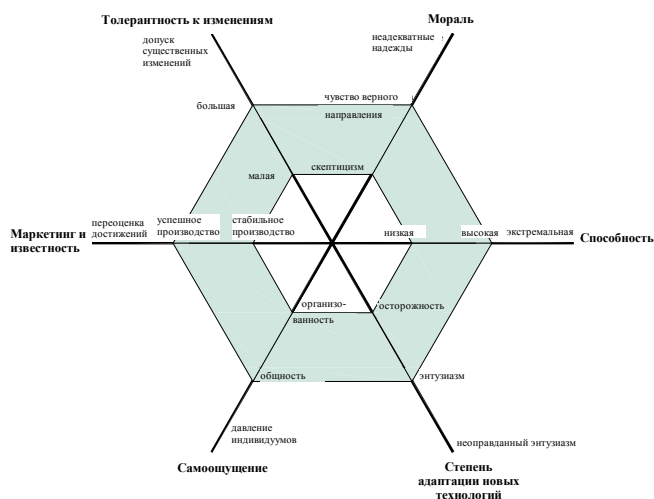


Рис. 2.8. Область определения существенных переменных

Таблица 2.15

Ступенчатые функции, обеспечивающие сверхустойчивость системы из команды и среды

Существенная переменная системы команды и среды	
Изменяемый параметр при низких значениях переменной	Изменяемый параметр при высоких значениях переменной
1. Толерантность к изменениям (C_A)	
Больше исследователей	Больше прагматичных разработчиков
2. Мораль (T_M)	
Новый план с детализированными шагами к выполнению проекта	Прагматичный план, прототипирование и доказательство верности изменений

Существенная переменная системы команды и среды	
Изменяемый параметр при низких значениях переменной	Изменяемый параметр при высоких значениях переменной
<p>3. Степень адаптации новых технологий (R_{NTA})</p>	
Обзор успешных случаев использования	Обзор неудачных случаев использования, призыв к дискуссиям, прототипирование
<p>4. Способность (T_C)</p>	
Больше изменений, в том числе «искусственных»	Требование написать документацию по API, обзор функциональности
<p>5. Самоощущение (T_{SP})</p>	
Выборы нового лидера команды с применением правил меритократии	Больше акцент на командный дух, важность всех членов команды
<p>6. Маркетинг и известность (M_E)</p>	
Больше пользователей, обзоров, участие команды в конференциях	Больше критицизма, настройка команды на положительное восприятие критики

2.2.2. Математическая модель динамики изолированной системы управления программными проектами. Для разработки математической модели динамики изолированной системы управления программными проектами за основу была

взята упрощенная модель динамики Абдель-Хамида [56], которая рассматривает систему управления как нестационарную линейную систему (изолированную или с постоянным входом) и для которой имеется структурная схема модели в переменных состояния, представленная на рис. 2.9.

Для сравнительного исследования системы без самоорганизации и адаптивной системы на устойчивость, необходимо получить математическую модель и выбрать критерий устойчивости для обеих систем.

Математическая модель системы определяемой состояниями с постоянным входом имеет следующий вид дифференциального уравнения состояния [73]:

$$\dot{\mathbf{x}} = \mathbf{A} \mathbf{x},$$

где \mathbf{x} – вектор производных переменных состояния по времени;

\mathbf{A} – матрица коэффициентов влияния переменных на скорость изменения каждой переменной состояния;

\mathbf{x} – вектор состояния, составленный из переменных состояния.

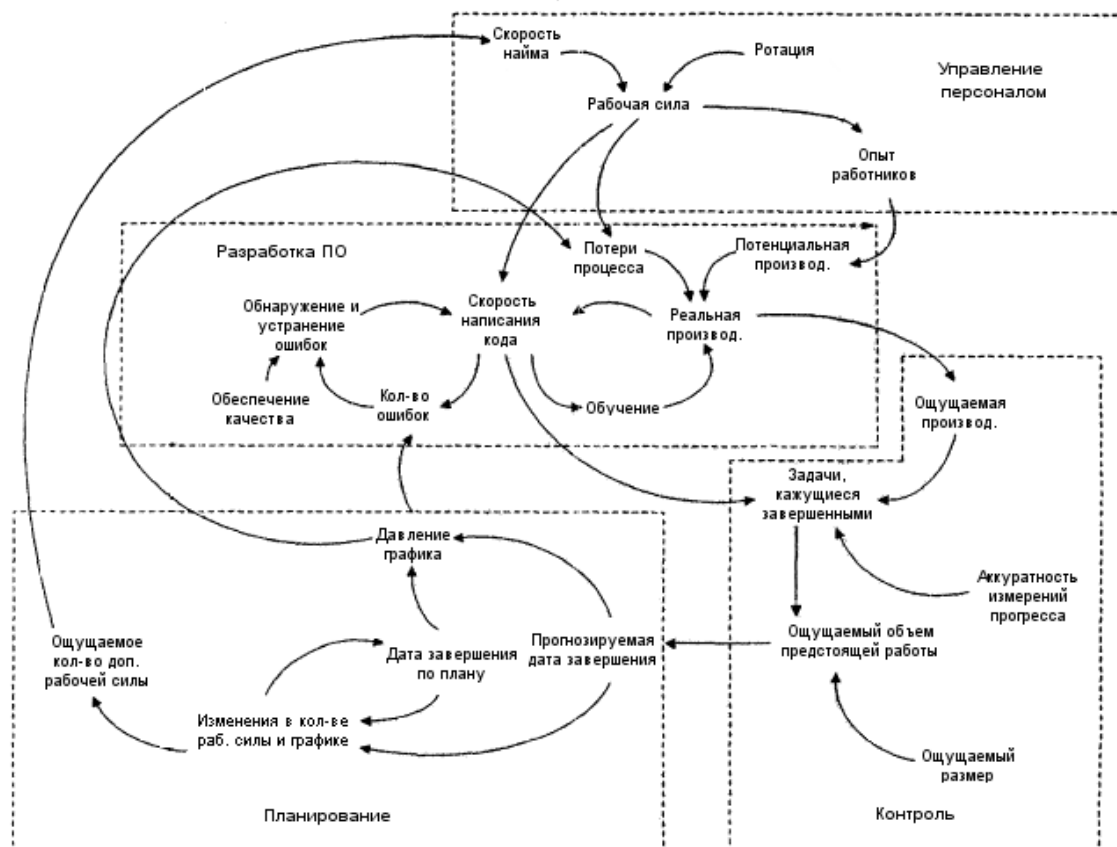


Рис. 2.9. Структурная схема модели динамики системы управления программным проектом

Математическая модель исследуемой системы получена преобразованием структурной схемы (рис 2.9) в сигнальный граф (рис 2.10) и записи согласно этому графу системы дифференциальных уравнений состояния

$$\left\{ \begin{array}{l} \dot{x}_p = \alpha_{dx} d \\ \dot{d} = \alpha_{td} t \\ \dot{t} = \alpha_{rt} r_c - \alpha_{tt} t \\ \dot{p} = \alpha_{xp} x_p - \alpha_{dp} d + \alpha_{pp} p, \\ \dot{s} = \alpha_{ds} d - \alpha_{bs} b + \alpha_{ps} p \\ \dot{b} = \alpha_{tb} t + \alpha_{sb} s \\ \dot{r}_c = -\alpha_{pr} p - \alpha_{sr} s \end{array} \right. \quad (2.1)$$

где x_p – опыт разработчиков;

d – количество разработчиков;

t – прогнозируемая длительность разработки;

p – продуктивность разработчиков;

s – размер создаваемого программного продукта;

b – количество обнаруженных ошибок в программном продукте;

r_c – количество функциональных требований;

$\alpha_{dx}, \alpha_{td}, \alpha_{rt}, \alpha_{tt}, \alpha_{xp}, \alpha_{dp}, \alpha_{pp}, \alpha_{ds}, \alpha_{bs}, \alpha_{ps}, \alpha_{tb}, \alpha_{sb}, \alpha_{pr}, \alpha_{sr}$ – коэффициенты влияния переменных состояния на скорость их изменения,

$\dot{x}_p, \dot{d}, \dot{t}, \dot{p}, \dot{s}, \dot{b}, \dot{r}_c$ – скорости изменения переменных состояния во времени τ , выраженные их производными.

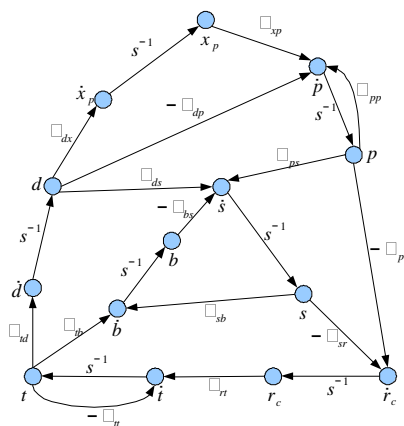


Рис. 2.10. Сигнальный граф представляющий модель динамики изолированной системы управления программным проектом

Система (2.1) в виде матричного дифференциального уравнения состояния представляется как:

$$\frac{d}{d\tau} \begin{bmatrix} x_p \\ d \\ t \\ p \\ s \\ b \\ r_c \end{bmatrix} = \begin{bmatrix} 0 & \alpha_{dx} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \alpha_{td} & 0 & 0 & 0 & 0 \\ 0 & 0 & -\alpha_{tt} & 0 & 0 & 0 & \alpha_{rt} \\ \alpha_{xp} & -\alpha_{dp} & 0 & \alpha_{pp} & 0 & 0 & 0 \\ 0 & \alpha_{ds} & 0 & \alpha_{ps} & 0 & -\alpha_{bs} & 0 \\ 0 & 0 & \alpha_{tb} & 0 & \alpha_{sb} & 0 & 0 \\ 0 & 0 & 0 & -\alpha_{pr} & -\alpha_{sr} & 0 & 0 \end{bmatrix} \begin{bmatrix} x_p \\ d \\ t \\ p \\ s \\ b \\ r_c \end{bmatrix}. \quad (2.2)$$

Так как уравнение (2.2) системы является линейным и представлено в матричной форме, то для исследования устойчивости системы выбран критерий Раусса-Гурвица [74, 75], а точнее его вариант, основанный на анализе определителя Гурвица, составленного из коэффициентов характеристического уравнения системы. Такой выбор обуславливается простотой получения критерия на основе выражения (2.2), аperiodическим характером входного воздействия и взаимного влияния переменных состояния, а также относительной легкостью программной реализации расчета значений критериев. Для получения численного значения критерия Гурвица необходимо только выполнить операции вычисления миноров и определителей матриц и простые алгебраические вычисления.

Для получения аналитического выражения характеристического уравнения написана программа для пакета символьных алгебраических вычислений *Mathia* [76]. Текст программы приводится в Приложении Б.1.

Полученное при помощи этой программы характеристическое уравнение системы относительно переменной λ имеет вид:

$$\begin{aligned} & \lambda^7 - (\alpha_{pp} - \alpha_{tt})\lambda^6 - (\alpha_{pp}\alpha_{tt} - \alpha_{sb}\alpha_{bs})\lambda^5 - \alpha_{bs}(\alpha_{pp}\alpha_{sb} - \alpha_{sb}\alpha_{tt})\lambda^4 - \\ & - ((\alpha_{pp}\alpha_{sb}\alpha_{tt} + \alpha_{rt}\alpha_{sr}\alpha_{tb})\alpha_{bs} + (\alpha_{dp}\alpha_{pr}\alpha_{rt} - \alpha_{bs}\alpha_{rt}\alpha_{sr})\alpha_{td})\lambda^3 - \\ & - (-\alpha_{pp}\alpha_{rt}\alpha_{sr}\alpha_{tb}\alpha_{bs} - \alpha_{dx}\alpha_{pr}\alpha_{rt}\alpha_{td}\alpha_{xp} + (\alpha_{dp}\alpha_{ps} + \alpha_{ds}\alpha_{pp})\alpha_{rt}\alpha_{sr}\alpha_{td})\lambda^2 - \\ & - (\alpha_{dp}\alpha_{pr}\alpha_{rt}\alpha_{sr}\alpha_{tb}\alpha_{bs} - \alpha_{dx}\alpha_{pr}\alpha_{rt}\alpha_{td}\alpha_{xp})\lambda + \\ & + \alpha_{dx}\alpha_{pr}\alpha_{rt}\alpha_{sb}\alpha_{td}\alpha_{xp}\alpha_{bs} = 0, \end{aligned} \quad (2.3)$$

Замена коэффициентов при λ в уравнении (2.3) на

$$\begin{aligned}
 m_0 &= 1, \\
 m_1 &= \alpha_{tt} - \alpha_{pp}, \\
 m_2 &= \alpha_{sb}\alpha_{bs} - \alpha_{pp}\alpha_{tt}, \\
 m_3 &= \alpha_{bs}(\alpha_{sb}\alpha_{tt} - \alpha_{pp}\alpha_{sb}), \\
 m_4 &= (\alpha_{dp}\alpha_{pr}\alpha_{rt} - \alpha_{bs}\alpha_{rt}\alpha_{sr})\alpha_{td} - (\alpha_{pp}\alpha_{sb}\alpha_{tt} + \alpha_{rt}\alpha_{sr}\alpha_{tb})\alpha_{bs}, \\
 m_5 &= \alpha_{pp}\alpha_{rt}\alpha_{sr}\alpha_{tb}\alpha_{bs} + \alpha_{dx}\alpha_{pr}\alpha_{rt}\alpha_{td}\alpha_{xp} - (\alpha_{dp}\alpha_{ps} + \alpha_{ds}\alpha_{pp})\alpha_{rt}\alpha_{sr}\alpha_{td}, \\
 m_6 &= \alpha_{dx}\alpha_{pr}\alpha_{rt}\alpha_{td}\alpha_{xp} - \alpha_{dp}\alpha_{pr}\alpha_{rt}\alpha_{sr}\alpha_{tb}\alpha_{bs}, \\
 m_7 &= \alpha_{dx}\alpha_{pr}\alpha_{rt}\alpha_{sb}\alpha_{td}\alpha_{xp}\alpha_{bs}
 \end{aligned}$$

позволяет записать характеристическое уравнение в упрощенном виде:

$$m_0\lambda^7 + m_1\lambda^6 + m_2\lambda^5 + m_3\lambda^4 + m_4\lambda^3 + m_5\lambda^2 + m_6\lambda + m_7 = 0. \quad (2.4)$$

На основании уравнения (2.4) матрица Гурвица имеет вид:

$$\left. \begin{array}{ccccccc}
 m_1 & m_3 & m_5 & m_7 & 0 & 0 & 0 \\
 m_0 & m_2 & m_4 & m_6 & 0 & 0 & 0 \\
 0 & m_1 & m_3 & m_5 & m_7 & 0 & 0 \\
 0 & m_0 & m_2 & m_4 & m_6 & 0 & 0 \\
 0 & 0 & m_1 & m_3 & m_5 & m_7 & 0 \\
 0 & 0 & m_0 & m_2 & m_4 & m_6 & 0 \\
 0 & 0 & 0 & m_1 & m_3 & m_5 & m_7
 \end{array} \right\}. \quad (2.5)$$

Согласно критерия Гурвица, необходимым и достаточным условием устойчивости системы будет положительный знак значений определителей всех диагональных миноров матрицы (2.5), т.е.:

$$\begin{aligned}
 m_1 &> 0; \\
 \begin{bmatrix} m_1 & m_3 \\ m_0 & m_2 \end{bmatrix} &> 0; \\
 \begin{bmatrix} m_1 & m_3 & m_5 \\ m_0 & m_2 & m_4 \\ 0 & m_1 & m_3 \end{bmatrix} &> 0;
 \end{aligned}$$

$$\begin{bmatrix} m_1 & m_3 & m_5 & m_7 \\ m_0 & m_2 & m_4 & m_6 \\ 0 & m_1 & m_3 & m_5 \\ 0 & m_0 & m_2 & m_4 \end{bmatrix} > 0;$$

$$\begin{bmatrix} m_1 & m_3 & m_5 & m_7 & 0 \\ m_0 & m_2 & m_4 & m_6 & 0 \\ 0 & m_1 & m_3 & m_5 & m_7 \\ 0 & m_0 & m_2 & m_4 & m_6 \\ 0 & 0 & m_1 & m_3 & m_5 \end{bmatrix} > 0;$$

$$\begin{bmatrix} m_1 & m_3 & m_5 & m_7 & 0 & 0 \\ m_0 & m_2 & m_4 & m_6 & 0 & 0 \\ 0 & m_1 & m_3 & m_5 & m_7 & 0 \\ 0 & m_0 & m_2 & m_4 & m_6 & 0 \\ 0 & 0 & m_1 & m_3 & m_5 & m_7 \\ 0 & 0 & m_0 & m_2 & m_4 & m_6 \end{bmatrix} > 0;$$

$$\begin{bmatrix} m_1 & m_3 & m_5 & m_7 & 0 & 0 & 0 \\ m_0 & m_2 & m_4 & m_6 & 0 & 0 & 0 \\ 0 & m_1 & m_3 & m_5 & m_7 & 0 & 0 \\ 0 & m_0 & m_2 & m_4 & m_6 & 0 & 0 \\ 0 & 0 & m_1 & m_3 & m_5 & m_7 & 0 \\ 0 & 0 & m_0 & m_2 & m_4 & m_6 & 0 \\ 0 & 0 & 0 & m_1 & m_3 & m_5 & m_7 \end{bmatrix} > 0$$

2.2.3. Математическая модель динамики системы управления программными проектами с механизмами самоорганизации. Для построения математической модели системы управления с самоорганизацией использована модель Эшби (см. подраздел 1.2), в которой учтено влияние существенных переменных среды, определенных в таблице 2.15. Соответствие между обозначениями существенных переменных и ступенчатых функций (параметров) приведено в таблице 2.16.

С использованием δ -функции Дирака для введения ступенчатых параметров из таблицы 2.16 в модель (2.2), получено уравнение состояния системы управления проектами с самоорганизацией:

$$\begin{aligned}\dot{\mathbf{x}} &= \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{w} \\ \dot{\mathbf{w}} &= \delta(\mathbf{x}, \mathbf{w}),\end{aligned}\quad (2.7)$$

где \mathbf{w} – вектор параметров системы, реагирующих на существенные переменные;

δ – функция Дирака, определенная следующими условиями:

$$\delta(u) = 0 \text{ при } u \neq 0; \delta(u) \rightarrow \infty \text{ при } u = 0 \text{ так, что } \int_{-\infty}^{\infty} \delta(u) du = 1.$$

Таблица 2.16

Обозначения параметров системы управления с самоорганизацией

Существенная переменная	Описание параметра	Обозначение и размерность
Толерантность к изменениям (C_A)	Прагматизм разработчиков	o , прототипов/день
Мораль (T_M)	Степень детализации плана разработки продукта	z , задач/цель
Степень адаптации новых технологий (R_{NTA})	Количество прототипов	n_p , прототипов
Способность (T_C)	Атомарность изменений	a , изменений/день
Самоощущение (T_{SP})	Степень контроля действий	v , обзоров/именение
Маркетинг и известность (M_E)	Количество публичных мероприятий, связанных с проектом	l , мероприятий

2.3. Основные выводы по разделу 2

1. Разработана модель жизненного цикла программных проектов с открытым кодом, которая определяет перечень и последовательность действий для

выполнения в проекте, определяет место процессов управления в жизненном цикле.

2. Доказано, что разработанная модель жизненного цикла соответствует требованиям стандартов ISO/IEC 12207 и ДСТУ 3918-1999 и включает все регламентированные стандартами процессы, выполняющиеся в одной или нескольких фазах жизненного цикла.
3. Показано, что открытая модель является аналогом быстрых моделей, так как имеет ряд сходственных принципов управления командами и организации разработки программного продукта.
4. Разработана модель динамики системы управления программными проектами с открытым кодом для случаев изолированной и адаптивной сверхустойчивой систем и создано программное обеспечение, которое для данных ограничений параметров состояния системы и коэффициентов их взаимного влияния рассчитывает вероятность устойчивости системы.
5. Обоснованы управляемость проекта и возможность его успешного завершения наличием поля устойчивости системы управления, которое позволяет принять решение про выполнимость проекта еще до начала его выполнения.
6. Определено, что наличие в модели системы управления ступенчатых функций обеспечивает самоорганизацию в проектах с открытым кодом, что повышает вероятность их успешного завершения при влиянии внешних дестабилизирующих изменений.

РАЗДЕЛ 3. ПРИМЕНЕНИЕ ТЕОРИИ ПОДОБИЯ ДЛЯ УПРАВЛЕНИЯ ВРЕМЕНЕМ И СТОИМОСТЬЮ В ПРОГРАММНЫХ ПРОЕКТАХ

3.1. Подобие процессов выполнения программных проектов

Так как создаваемые программные продукты, очевидно, относятся к различным "классам", обобщение опыта их создания возможно только внутри одного класса или внутри еще более узких групп. Разделение на классы и группы, внутри которых возможно обобщение, позволит точнее осуществлять моделирование на ранних этапах выполнения новых проектов, "аналогичных" предыдущим.

Наиболее совершенный и проверенный на практике аппарат аналогового моделирования представляет теория подобия, основные положения которой были разработаны еще в начале прошлого века и наиболее полно обобщены А.А. Гухманом [82].

Основная идея теории подобия заключается в том, что в пределах некоторого класса явлений или процессов выделяются группы, в которых возможно обобщение данных единичного опыта.

При производстве программных продуктов, то под понятием "единичный опыт" следует понимать:

- производство одной (первой) версии программной системы компанией-разработчиком;
- производство аналогичной системы другой компанией (в случае закрытой модели разработки) или группой разработчиков (в случае открытой модели разработки);
- производство демонстрационного прототипа системы.

Обобщить опыт выполнения проекта (создания программной системы) – значит дать возможность оценить трудоемкость, стоимость и длительность выполнения аналогичного проекта, т.е. создать аналогичную (подобную) систему.

Это возможно, если соотношение между безразмерными признаками (критериями подобия), характеризующими свойства системы представить в виде аппроксимирующей наиболее часто используемой степенной функции:

$$\pi = A \cdot \pi_1^a \cdot \pi_2^b \cdot \dots \cdot \pi_n^z, \quad (3.1)$$

где $\pi, \pi_1, \pi_2, \dots, \pi_n$ – безразмерные величины – критерии подобия,

A – коэффициент пропорциональности между определяющими и определяемым критериями,

a, b, \dots, z – показатели степени при критериях.

При анализе подобия программных продуктов с учетом основного правила аналогового моделирования [82] использовано следующее:

1. внутри одного класса процессов вид зависимости (3.1) будет оставаться неизменным, т.е.:

$$a = \text{idem}; b = \text{idem}; \dots z = \text{idem}; A = \text{idem};$$

2. для группы подобных систем критерии подобия $\pi, \pi_1, \pi_2, \dots, \pi_n$ будут равны, а составляющие критерии показатели – пропорциональны [83].

Таким образом, аналоговая модель позволяет:

1. исходя из значений критериев нескольких групп одного класса процессов, собранных в ходе выполнения проектов, получить аналоговую модель этого класса в виде уравнения (3.1) с известными значениями показателей степени a, b, \dots, z и коэффициента пропорциональности A ;
2. получить значения критериев подобия $\pi, \pi_1, \pi_2, \dots, \pi_n$ для группы подобных процессов на основании единичного опыта.

3.2. Классы и группы подобных процессов выполнения проектов. Условия подобия

Для аналогового моделирования программных проектов необходимо определить:

1. какие процессы будут входить в класс;

2. какие процессы будут формировать группы, т.е. какое множество условий определяет подобие процессов выполнения программных проектов.

Класс процесса выполнения проекта определяется двумя характеристиками – характеристиками проекта и продукта, соответственно:

- методом разработки: быстрая разработка (eXtreme Programming, Agile Methods), разработка по водопадной, спиральной, инкрементальной, открытой и т.д. модели;
- типом разрабатываемого продукта: система обработки транзакций, система поддержки принятия решений, среда разработки, офисное приложение, учетное приложение и т.д.

Формирование групп подобных проектов внутри класса описывается условиями подобия. При рассмотрении физических процессов такими условиями будут являться:

- принадлежность процессов и/или явлений к одному классу;
- геометрическое подобие, когда между геометрическими характеристиками существуют масштабные соотношения;
- временное подобие (гомохронность, когда каждому моменту той временной области, в которой протекает первое явление, отвечают сходственные моменты тех временных областей, в которых протекают другие явления той же группы);
- физическое подобие средин, когда между константами подобных явлений существуют масштабные соотношения либо равенство;
- подобие начальных и граничных условий, когда между начальными (либо граничными) параметрами явления существуют масштабные соотношения.

На первый взгляд процессы выполнения программных проектов отличаются от физических процессов и явлений, но при детальном их сопоставлении можно отметить следующие сходные качества:

- обязательный временной характер: как физическое явление так и программный проект происходят во времени, причем внутри определенного, конечного [27] интервала времени;

- наличие закономерностей, устанавливающих зависимость параметров процессов друг от друга и от времени: физические процессы моделируются с помощью универсальных соотношений физических параметров – законов сохранения, в программных же проектах также наблюдается зависимость параметров друг от друга, например, времени разработки продукта от его размера;
- возможность определения завершающего состояния системы, в которой выполняется процесс без знания ее предыстории, а основываясь только на знании начального состояния: справедливо для физических систем т.к. универсальные соотношения выражаются в общем виде в дифференциальной форме; также справедливо и для программных проектов, т.к., например, зачастую предыстории системы не существует.

Эти сходные качества приводят к выводу о том, что теория подобия рассматривает один класс систем, а именно класс функциональных, временных и статических систем. Системы, в которых происходят физические явления и системы, в которых выполняются программные проекты, подпадают под определение этого класса, т.к. они являются::

- функциональными (между входными и выходными объектами системы существует функциональная зависимость);
- временными (входные и выходные объекты системы определены на одном и том же множестве моментов времени);
- статическими (выходные объекты системы зависят только от состояния, в котором началась эволюция).

Следовательно, можно утверждать, что общность систем с точки зрения теории подобия, определяет общность условий подобия, которые теория накладывает на процессы в рассматриваемых системах. Исходя из этого возможно условия подобия для физических явлений применять к программным проектам.

Требование принадлежности процессов к одному классу является универсальным.

Геометрическое подобие применительно к процессам в проектах будет означать подобие размеров продуктов, производимых в этих проектах, определяемых либо в функциональных (объектных) точках либо числом строк программного кода. При этом в программных проектах условие геометрического подобия будет выполняться всегда при выборе одной и той же единицы измерения вследствие одномерности программного кода.

Временное подобие будет выражаться в требовании подобия временных характеристик жизненных циклов программных проектов. Только в этом случае будет соблюдаться гомохронность в пределах одной и той же группы, так как жизненные циклы определяют последовательность и период выполнения определенных фаз разработки, которые и будут являться требуемыми сходственными временными областями.

Физическое подобие средин физических систем и явлений выражается в управлении программными проектами как подобие констант, связанных с внешней средой, в которой выполняется проект. Такими константами, например, могут являться экономические показатели, например, средний размер ставки разработчика, а также некоторые технологические показатели, как используемые инструменты и языки программирования.

Так как проект выполняется во времени и не имеет пространственных характеристик, то из условий подобия начальных и граничных условий применимыми остаются только условия подобия начальных условий. Такие начальные условия для программных систем как:

- процесс и методы разработки продукта;
- язык программирования (или поколение языка программирования);
- платформа для целевой системы;
- используемые инструменты,

должны быть подобны для систем одной группы.

Из вышесказанного следует, что условиями подобия программных проектов (процессов их выполнения) являются:

1. принадлежность к одному классу;

2. подобие жизненных циклов разработки;
3. подобие констант внешней среды выполнения разработки;
4. подобие начальных условий вхождения в жизненный цикл.

3.3. Обобщенная методика построения аналоговой модели для оценки времени и стоимости программных проектов

Методика применения аппарата теории подобия для построения аналоговых моделей программных систем, очевидно, будет во многом аналогична случаю физических систем. Для построения аналоговой модели необходимо применить метод анализа размерностей, для чего выполнить следующую последовательность действий.

1. Ввести множество показателей, определяющих состояние программной системы.

Эти показатели должны количественно описывать состояние системы и быть измеримыми либо вычислимыми.

2. Построить систему единиц измерения и задать размерности выбранных показателей.

Для этого произвольно выбираются основные единицы измерения и на основании них согласно известным закономерностям получаются производные единицы измерения (если таковые имеются). После этого задается размерность каждого количественного показателя, введенного в п.1 методики.

3. Представить соотношение между размерными величинами в виде функции

$$a = A \cdot v_1^a \cdot v_2^b \cdot \dots \cdot v_k^l \cdot v_{k+1}^{l+1} \cdot \dots \cdot v_n^z, \quad (3.2)$$

где a – определяемая величина,

v_1, v_2, \dots, v_n – определяющие величины,

n – количество размерных определяющих величин,

k – количество первичных размерностей.

4. Найти соотношение между безразмерными величинами, представляющими количественные соотношения той же системы в виде, определенном π -теоремой (теоремой Бэкингема):

$$\pi = f_1(\pi_1, \pi_2, \dots, \pi_{n-k}), \quad (3.3)$$

где $\pi, \pi_1, \pi_2, \dots, \pi_{n-k}$ – безразмерные величины.

Для этого необходимо выполнить следующие действия:

- подставить размерности вместо самих величин в уравнение (3.2);
- просуммировать показатели степени при одинаковых основных единицах измерения и записать выражения с суммой в систему уравнений;
- решить полученную систему уравнений, выразив через произвольно выбранные k показателей степени остальные $n-k$;
- подставить выраженные показатели степени в (3.2) и преобразовать зависимость к виду (3.3), группируя величины v_i при одинаковых показателях степени.

5. Вычислить коэффициенты пропорциональности и показатели степени критериев модели

В критериальном уравнении (3.3) вид функциональной зависимости, а именно коэффициенты пропорциональности и показатели степени критериев модели определяются по данным эксперимента, т.е. по данным уже выполненного проекта(ов). Для каждого класса процессов будет наблюдаться равенство показателей степеней и коэффициентов пропорциональности, а также для каждой группы процессов будет наблюдаться равенство критериев.

Уравнение (3.3), полученное на 4-м шаге указанной выше методики будет являться абстрактной аналоговой моделью процессов выполнения программных проектов.

Уравнение (3.3) с конкретными значениями коэффициентов пропорциональности и показателей степени критериев будет являться конкретной аналоговой моделью процессов выполнения программных проектов одного класса.

3.4. Получение абстрактных аналоговых моделей для программных проектов

Пользуясь методикой, изложенной в разделе 3.3, становится возможным получить абстрактные аналоговые модели для открытых и других программных проектов с целью их сравнения и сопоставления.

Аналоговое моделирование проектов обобщенной модели жизненного цикла

Обобщенная модель жизненного цикла [62] – это развитие классической каскадной модели [61], включающей процессы аттестации и верификации. Существует несколько разновидностей каскадной модели, однако ее общими чертами является последовательное выполнение фаз жизненного цикла, будь то для целей прототипирования или для разработки готового продукта. Так как существует две разновидности модели – с прототипированием и без, ниже приводятся соответствующие им две аналоговые модели.

Аналоговое моделирование проектов обобщенной модели жизненного цикла с прототипированием.

Аналоговое моделирование производилось согласно вышеизложенного алгоритма.

1. Определение множества показателей, определяющих состояние программной системы, выбор системы единиц измерения и задание размерностей.

Принимая за основу список факторов, влияющих на длительность разработки, определенный в [77], выделяются несколько размерных параметров.

Одним из важных (и трудноизмеримым) параметром, очевидно, будет размер программной системы – s . Существует множество методик определения размера, каждая из которых определяет свою единицу измерения. Классический подход к измерению размера, как он определен в [88] предлагает использовать в качестве единицы измерения количество строк кода, SLOC (Source Lines Of Code). [89] и [90] предлагают метод измерения размера программ в функциональных точках. [91] предлагает объектные точки и точки свойств в качестве размерности.

Использование каждой из вышеозначенных единиц измерения имеет свои недостатки и преимущества (см., например, [62]). С точки зрения же предстоящего аналогового моделирования, более существенными оказываются недостатки SLOC как единицы измерения. Так, количество строк кода может произвольно меняться в зависимости от типов языков программирования, методов проектирования, стиля и способностей программиста. Использование генераторов кода может привести к искажению показателя SLOC, т.к. размер генерируемого кода не влияет ни на длительность, ни на стоимость разработки программной системы. Функциональным (и другим) точкам вышеозначенные недостатки не присущи. Таким образом, наиболее распространенной единицей измерения является все же функциональная точка (ее измерение наилучшим образом формализовано в [17, 18, 19, 20]).

При дальнейшем построении аналоговых моделей размер s будет указываться просто в точках без уточнения типа точек [89, 90, 17, 18, 19, 20]. Все остальные единицы измерения при необходимости будут также определены относительно точки как единицы размера. Если в проекте параметр (метрика) размера выражается не в функциональных точках, а в строках кода [88], то для целей моделирования можно воспользоваться таблицей преобразования размера в SLOC в функциональные точки из [11].

Сложность разработки – c_x . измеряется в точках, отнесенных к человеко-дню и определяет количество кода, необходимого для написания одним человеком за один день при условии успешного завершения проекта за некоторое определенное время [62].

Сложность требований выражается двумя параметрами. Параметрами r_c – количество устойчивых, необходимых, функциональных требований (единица измерения – функц) [62] и r_v , – соотношение между имеющимися готовыми компонентами программной системы (единица измерения – комп) и количеством функциональных требований. Параметр r_v определяет степень повторного использования компонентов и, одновременно, степень покрытия такими

компонентами функциональных требований. Соответственно, вводится еще один показатель – число компонентов, из которых состоит программная система – c_c .

Число разработчиков d измеряется количеством человек (чел.) Их производительность p – в точках, отнесенных к человеко-дням. Подобная единица измерения производительности была предложена в [89], [91] и [92] и отражает количество кода, которое может написать один разработчик за один день.

Языковой фактор l_f учитывается показателем эффективности языка, определяемым как среднее количество программного кода, производимое программистом при использовании языка за один день. Включение языкового фактора в перечень определяющих параметров модели позволит учесть иногда значительное влияние выбранного языка программирования на стоимость и длительность разработки.

Опыт разработчиков – x_p есть также одной из самых сложных метрик. Его трудно измерить. При моделировании этот фактор либо не учитывался, либо учитывался поправочными коэффициентами вычисленными на основании измерения хронологического опыта как в регрессионной модели COSOMO [9].

Представляется, что аналоговая модель все-таки должна учитывать показатель x_p ввиду его сильного влияния на стоимость и длительность, поэтому в дальнейшем опыт разработчиков определяется по методу Куртиса [62] как количество дефектов, которое может быть обнаружено одним разработчиком в программном коде за некоторый промежуток времени с единицей измерения $\text{деф}/(\text{чел}\cdot\text{день})$.

К вышеперечисленным размерным показателям необходимо добавить еще и показатель качества – q . Качество является одним из важных ограничений на результат разработки, непосредственно влияющим на ее стоимость и длительность, т.к. на достижение заданного качества требуются ресурсы. Качество определяется как количество дефектов на единицу размера кода [62].

Рассмотренные размерные показатели определяют факторы, влияющие на любой программный проект, выполняющийся в обобщенной (каскадной) модели жизненного цикла. Для учета прототипирования необходимо к множеству

определенных размерных показателей добавить показатели концептуального прототипа, выполняемого в проекте. На прототипирование отводятся определенные ресурсы проекта, а результат прототипирования влияет на дальнейший ход выполнения проекта. Следовательно, длительность и стоимость разработки зависят от факторов, определяемых процессом прототипирования. Такими факторами будут размер прототипа, s_{cp} ; время прототипирования, t_{cp} ; число разработчиков d_{cp} , выполняющих прототипирование и их производительность p_{cp} . Размерности этих показателей будут аналогичны рассмотренным выше показателям собственно программной системы.

Результаты выбора размерных показателей и системы единиц измерения приведены в таблице 3.1.

Таблица 3.1

Размерные показатели и система единиц измерения для аналогового моделирования программной системы разрабатываемой по каскадной модели с прототипированием

Показатель	Обозн.	Размерность
Длительность разработки (определяемый параметр)	t	день
Размер	s	точки
Сложность разработки	c_x	точки / (чел·день)
Количество устойчивых, необходимых, функциональных требований	r_c	функц
Степень покрытия готовыми комп. Функц. требований	r_v	комп / функц
Число компонентов	c_c	комп
Число разработчиков	d	чел
Производительность разработчиков	p	точки / (чел·день)
Языковой фактор	l_f	точки / день
Опыт разработчиков	x_p	деф / (чел·день)
Качество	q	деф / точки
Размер прототипа	s_{cp}	точки
Время прототипирования	t_{cp}	день
Число разработчиков, выполняющих прототипирование	d_{cp}	чел
Производ. разработчиков, выполняющих прототипирование	p_{cp}	точки / (чел·день)

2. Построение аналоговой модели

Зависимость между определяемыми и определяющими показателями эволюции программной системы аппроксимирована как

$$t = A \cdot c_x^a \cdot r_c^b \cdot r_v^c \cdot s^d \cdot d^e \cdot p^f \cdot s_{cp}^g \cdot t_{cp}^h \times \\ \times d_{cp}^i \cdot p_{cp}^j \cdot q^k \cdot l_f^l \cdot x_p^m \cdot c_c^n. \quad (3.4)$$

Критериальное уравнение определяется, используя метод анализа размерностей. Для этого уравнение (3.4) представляется в терминах размерностей входящих в уравнение величин и получается соотношение вида

$$\text{день} = \frac{\text{точки}^a}{\text{чел}^a \cdot \text{день}^a} \cdot \text{функц}^b \cdot \frac{\text{комп}^c}{\text{функц}^c} \cdot \text{точки}^d \cdot \text{чел}^e \times \\ \times \frac{\text{точки}^f}{\text{чел}^f \cdot \text{день}^f} \cdot \text{точки}^g \cdot \text{день}^h \cdot \text{чел}^i \cdot \frac{\text{точки}^j}{\text{чел}^j \cdot \text{день}^j} \cdot \frac{\text{деф}^k}{\text{точки}^k} \times \\ \times \frac{\text{точки}^l}{\text{день}^l} \cdot \frac{\text{деф}^m}{\text{чел}^m \cdot \text{день}^m} \cdot \text{комп}^n. \quad (3.5)$$

Здесь количество размерных величин $n = 15$, количество независимых размерностей $k = 6$. Соответственно, необходимо получить $n-k=9$ критериев.

Система уравнений, составленная из показателей степеней для каждой размерности из уравнения (3.5):

$$\begin{aligned} \text{день: } 1 &= -a - f + h - j - l - m \\ \text{точки: } 0 &= a + d + f + g + j - k + l \\ \text{чел: } 0 &= -a + e - f + i - j - m \\ \text{комп: } 0 &= c + n \\ \text{деф: } 0 &= k + m \\ \text{функц: } 0 &= b - c \end{aligned} \quad (3.6)$$

Складывая первое уравнение со вторым и второе с третьим в системе (3.6), а также выражая остальные величины из последних трех уравнений, получаем решение системы:

$$\begin{aligned} k &= -m \\ c &= -n \\ b &= -n \\ a &= e - f + i - j - m \\ h &= 1 - d - g \\ l &= -d - e - g - i \end{aligned} \quad (3.7)$$

Подставляя значения показателей из (3.7) в (3.4), получаем

$$t = A \cdot \frac{c_x^e \cdot c_x^i}{c_x^f \cdot c_x^j \cdot c_x^m} \cdot \frac{1}{r_c^n} \cdot \frac{1}{r_v^n} \cdot s^d \cdot d^e \cdot p^f \cdot s_{cp}^g \times \\ \times \frac{t_{cp}}{t_{cp}^d \cdot t_{cp}^g} \cdot d_{cp}^i \cdot p_{cp}^j \cdot \frac{1}{q^m} \cdot \frac{1}{l_f^d \cdot l_f^e \cdot l_f^g \cdot l_f^i} \cdot x_p^m \cdot c_c^n. \quad (3.8)$$

Группируя в (3.8) величины с одинаковыми показателями степени, записываем

$$\frac{t}{t_{cp}} = A \cdot \left(\frac{c_x \cdot d}{l_f} \right)^e \cdot \left(\frac{p}{c_x} \right)^f \cdot \left(\frac{c_x \cdot d_{cp}}{l_f} \right)^i \cdot \left(\frac{p_{cp}}{c_x} \right)^j \cdot \left(\frac{c_c}{r_c \cdot r_v} \right)^n \times \\ \times \left(\frac{s}{t_{cp} \cdot l_f} \right)^d \cdot \left(\frac{s_{cp}}{t_{cp} \cdot l_f} \right)^g \cdot \left(\frac{x_p}{q \cdot c_x} \right)^m, \quad (3.9)$$

где комплексы в скобках безразмерны и являются критериями подобия.

Комплексы обозначаются символьными именами и проверяется их размерность:

$$D_t = \frac{t}{t_{cp}}; [D_t] = \frac{\text{день}}{\text{день}} = 1;$$

$$D_{com} = \frac{c_x \cdot d}{l_f}; [D_{com}] = \frac{\text{точки}}{\text{чел} \cdot \text{день}} \cdot \text{чел} \cdot \frac{\text{день}}{\text{точки}} = 1;$$

$$D_{prod} = \frac{p}{c_x}; [D_{prod}] = \frac{\text{точки}}{\text{чел} \cdot \text{день}} \cdot \frac{\text{чел} \cdot \text{день}}{\text{точки}} = 1;$$

$$D_{compro} = \frac{c_x \cdot d_{cp}}{l_f}; [D_{compro}] = \frac{\text{точки}}{\text{чел} \cdot \text{день}} \cdot \text{чел} \cdot \frac{\text{день}}{\text{точки}} = 1;$$

$$D_{quapro} = \frac{p_{cp}}{c_x}; [D_{quapro}] = \frac{\text{точки}}{\text{чел} \cdot \text{день}} \cdot \frac{\text{чел} \cdot \text{день}}{\text{точки}} = 1;$$

$$D_{reus} = \frac{c_c}{r_c \cdot r_v}; [D_{reus}] = \text{комп} \cdot \frac{1}{\text{функц}} \cdot \frac{\text{функц}}{\text{комп}} = 1;$$

$$D_{size} = \frac{s}{t_{cp} \cdot l_f}; [D_{size}] = \text{точки} \cdot \frac{1}{\text{день}} \cdot \frac{\text{день}}{\text{точки}} = 1;$$

$$D_{sizepro} = \frac{s_{cp}}{t_{cp} \cdot l_f}; [D_{sizepro}] = \text{точки} \cdot \frac{1}{\text{день}} \cdot \frac{\text{день}}{\text{точки}} = 1;$$

$$D_{qua} = \frac{x_p}{q \cdot c_x}; [D_{qua}] = \frac{\text{деф}}{\text{чел} \cdot \text{день}} \cdot \frac{\text{точки}}{\text{деф}} \cdot \frac{\text{чел} \cdot \text{день}}{\text{точки}} = 1.$$

Таким образом, абстрактная аналоговая модель эволюции программной системы общего назначения, разрабатываемой по обобщенной модели с учетом прототипирования будет выражаться критериальным уравнением

$$D_t = A \cdot D_{com}^e \cdot D_{prod}^f \cdot D_{compro}^i \cdot D_{prodpro}^j \times \\ \times D_{reus}^n \cdot D_{size}^d \cdot D_{sizepro}^g \cdot D_{qua}^m \quad (3.10)$$

Аналоговое моделирование проектов обобщенной модели жизненного цикла без прототипирования

Моделирование включало в себя следующие операции.

1. Определение множества показателей, определяющих состояние программной системы, выбор системы единиц измерения и задание размерностей.

Множество размерных показателей, определяющих эволюцию программной системы при ее разработке по каскадной модели жизненного цикла является подмножеством аналогичного множества для открытой модели, исключая метрики прототипа. Система единиц измерения остается аналогичной и приведена в таблице 3.2.

Таблица 3.2

Размерные показатели и система единиц измерения для аналогового моделирования программной системы, разрабатываемой по каскадной модели без прототипирования

Показатель	Обозначение	Размерность
Длительность разработки (определяемый параметр)	t	день
Размер	s	точки
Сложность разработки	c_x	точки / (чел·день)
Количество устойчивых, необходимых, функциональных требований	r_c	функц
Степень покрытия готовыми компонентами функциональных требований	r_v	комп / функц
Число компонентов	c_c	комп

Показатель	Обозначение	Размерность
Число разработчиков	d	чел
Производительность разработчиков	p	точки / (чел·день)
Языковой фактор	l_f	точки / день
Опыт разработчиков	x_p	деф / (чел·день)
Качество	q	деф / точки

2. Построение аналоговой модели

Зависимость между определяемыми и определяющими показателями эволюции программной системы аппроксимируется как

$$t = A \cdot c_x^a \cdot r_c^b \cdot r_v^c \cdot s^d \cdot d^e \cdot p^f \cdot q^g \cdot l_f^h \cdot x_p^i \cdot c_c^j. \quad (3.11)$$

Уравнение (3.11) записанное в терминах размерностей входящих в него величин дает соотношение вида

$$\begin{aligned} \text{день} &= \frac{\text{точки}^a}{\text{чел}^a \cdot \text{день}^a} \cdot \text{функц}^b \cdot \frac{\text{комп}^c}{\text{функц}^c} \cdot \text{точки}^d \cdot \text{чел}^e \times \\ &\times \frac{\text{точки}^f}{\text{чел}^f \cdot \text{день}^f} \cdot \frac{\text{деф}^g}{\text{точки}^g} \cdot \frac{\text{точки}^h}{\text{день}^h} \cdot \frac{\text{деф}^i}{\text{чел}^i \cdot \text{день}^i} \cdot \text{комп}^j. \end{aligned} \quad (3.12)$$

Здесь количество размерных величин $n = 11$, количество независимых размерностей $k = 6$ и, соответственно, количество критериев $n-k=5$.

Система уравнений, составленная из показателей степеней для каждой размерности из уравнения (3.12):

$$\begin{aligned} \text{день} : 1 &= -a - f - h - i \\ \text{точки} : 0 &= a + d + f - g + h \\ \text{чел} : 0 &= -a + e - f - i \\ \text{комп} : 0 &= c + j \\ \text{деф} : 0 &= g + i \\ \text{функц} : 0 &= b - c \end{aligned} \quad (3.13)$$

Складывая первое уравнение со вторым и второе с третьим в системе (3.13), а также выражая остальные величины из последних трех уравнений, получается решение системы:

$$\begin{aligned}
g &= -i \\
c &= -j \\
b &= -j \\
a &= e - f - i \\
d &= 1 \\
h &= -1 - e
\end{aligned}
\tag{3.14}$$

Подставляя значения показателей из (3.14) в (3.11), получается:

$$t = A \cdot \frac{c_x^e}{c_x^f \cdot c_x^i} \cdot \frac{1}{r_c^j} \cdot \frac{1}{r_v^j} \cdot s \cdot d^e \cdot p^f \cdot \frac{1}{q^i} \cdot \frac{1}{l_f \cdot l_f^e} \cdot x_p^i \cdot c_c^j.
\tag{3.15}$$

Группируя в (3.15) величины с одинаковыми показателями степени:

$$\frac{t \cdot l_f}{s} = A \cdot \left(\frac{c_x \cdot d}{l_f} \right)^e \cdot \left(\frac{p}{c_x} \right)^f \cdot \left(\frac{c_c}{r_c \cdot r_v} \right)^j \cdot \left(\frac{x_p}{q \cdot c_x} \right)^i,
\tag{3.16}$$

где единственный критерий, не встречавшийся ранее – это определяемый критерий $D_{ts} = \frac{t \cdot l_f}{s}$; $[D_{ts}] = \frac{\text{день}}{\text{точки}} \cdot \frac{\text{точки}}{\text{день}} = 1$.

Абстрактная аналоговая модель эволюции программной системы общего назначения, разрабатываемой по обобщенной модели без прототипирования выражается критериальным уравнением

$$D_{ts} = A \cdot D_{com}^e \cdot D_{prod}^f \cdot D_{reus}^j \cdot D_{qua}^i.
\tag{3.17}$$

Аналоговое моделирование проектов быстрой (agile) модели жизненного цикла

Моделирование включало в себя следующие операции.

1. Определение множества показателей, определяющих состояние программной системы, выбор системы единиц измерения и задание размерностей.

Несмотря на то, что существует несколько вариантов быстрых моделей жизненных циклов (их полный обзор можно найти в [71]), метрики, собираемые в быстрых проектах совпадают. Основные методологические труды [11, 12, 13] предлагают одинаковые средства измерения и оценки.

Быстрые проекты отличаются специфическим подходом к оценкам. Размер продукта и требования не являются собираемыми метриками. Вместо этого

оцениваются количество пользовательских историй s_t и число идеальных инженерных дней i_{ed} , которые представляют собой количество дней, за которые разработчик может реализовать одну пользовательскую историю при отсутствии влияния внешних факторов [93]. Кроме этого, наиболее часто собираемыми метриками будут количество тестов n_t , количество разработчиков d и количество работающих оттестированных возможностей r_{tf} [94]. Последняя величина измеряется в количестве пользовательских историй, которые проходят тесты и является основным показателем гибкости проекта.

Выбранная система единиц измерения приведена в таблице 3.3.

Таблица 3.3

Размерные показатели и система единиц измерения для аналогового моделирования программной системы, разрабатываемой по быстрой модели

Показатель	Обозначение	Размерность
Длительность разработки (определяемый параметр)	t	день
Количество пользовательских историй	s_t	история
Число идеальных инженерных дней	i_{ed}	день / (чел·история)
Количество работающих оттестированных возможностей	r_{tf}	история / тест
Количество тестов	n_t	тест
Число разработчиков	d	чел

2. Построение аналоговой модели

Зависимость между определяемыми и определяющими показателями эволюции программной системы аппроксимируется как

$$t = A \cdot s_t^a \cdot i_{ed}^b \cdot r_{tf}^c \cdot n_t^e \cdot d^f, \quad (3.18)$$

Уравнение (3.18) записанное в терминах размерностей входящих в него величин приводит к соотношению вида

$$\text{день} = \text{история}^a \cdot \frac{\text{день}^b}{\text{история}^b \cdot \text{чел}^b} \cdot \frac{\text{история}^c}{\text{тест}^c} \cdot \text{тест}^e \cdot \text{чел}^f. \quad (3.19)$$

Здесь количество размерных величин $n = 6$, количество независимых размерностей $k = 4$. Соответственно, необходимо получить $n-k=2$ критерия.

Система уравнений, составленная из показателей степеней для каждой размерности из уравнения (3.19) имеет вид:

$$\begin{aligned} \text{день} : 1 &= b \\ \text{точки} : 0 &= a - b + c \\ \text{чел} : 0 &= -c + e \\ \text{комп} : 0 &= -b + f \end{aligned} \quad (3.20)$$

Складывая первое уравнение со вторым и второе с третьим в системе (3.20), а также выражая остальные величины из последних трех уравнений, получается решение системы:

$$\begin{aligned} b &= 1 \\ e &= c \\ f &= b = 1 \\ a &= b - c = 1 - c \end{aligned} \quad (3.21)$$

Подставляя значения показателей из (3.21) в (3.18), получается

$$t = A \cdot \frac{s_t}{s_t^c} \cdot i_{ed} \cdot r_{if}^c \cdot n_t^c \cdot d. \quad (3.22)$$

Группируя в (3.22) величины с одинаковыми показателями степени:

$$\frac{t}{s_t \cdot i_{ed} \cdot d} = A \cdot \left(\frac{r_{if} \cdot n_t}{s_t} \right)^c. \quad (3.23)$$

где выделяются два критерия:

$$A_{ts} = \frac{t}{s_t \cdot i_{ed} \cdot d}; [A_{ts}] = \frac{\text{день} \cdot \text{история} \cdot \text{чел}}{\text{история} \cdot \text{день} \cdot \text{чел}} = 1; A_{pq} = \frac{r_{ts} \cdot n_t}{s_t}; [A_{pq}] = \frac{\text{история}}{\text{тест}} \cdot \frac{\text{тест}}{\text{история}} = 1.$$

Абстрактная аналоговая модель эволюции программной системы общего назначения, разрабатываемой по быстрой модели выражается критериальным уравнением

$$A_{ts} = A \cdot A_{pq}^c. \quad (3.24)$$

Аналоговое моделирование проектов открытой модели жизненного цикла

Моделирование включало в себя следующие операции.

1. Определение множества показателей, определяющих состояние программной системы, выбор системы единиц измерения и задание размерностей

Управление открытыми проектами предполагает, как и при управлении быстрыми проектами, сбор особого типа метрик. Размер программного кода s оценивается либо в функциональных точках либо в числе строк кода. Количество разработчиков учитывается двумя параметрами – количеством основных разработчиков d_c и объемом привлечения сторонних разработчиков δ_d , измеряемым в числе разработчиков, привлеченных к проекту в единицу времени. Качество контролируется параметрами b – количеством вносимых ошибок и темпом исправления ошибок r_b – количеством ошибок, исправляемых в единицу времени. Производительность определяется группой параметров, учитывающих внешние и внутренние изменения. Учитывается средний объем внутренних изменений (т.е. вносимых основными разработчиками) c_v – количество единиц кода, производимых человеком в единицу времени, и внешние изменения как объем поступления доработок в единицу времени r_p и средний объем этих доработок p_v – количество единиц кода на одну доработку.

Выбранная система единиц измерения приведена в таблице 3.4.

Таблица 3.4

Размерные показатели и система единиц измерения для аналогового моделирования программной системы, разрабатываемой по открытой модели

Показатель	Обозначение	Размерность
Длительность разработки (определяемый параметр)	t	день
Размер программного кода	s	SLOC
Количество основных разработчиков	d_c	чел
Количество вносимых ошибок	b	деф
Объем привлечения сторонних разработчиков	δ_d	чел / день
Темп исправления ошибок	r_b	деф / день
Объем внутренних изменений	c_v	SLOC / (чел день)
Объем поступления доработок	r_p	дор / день
Размер доработок	p_v	SLOC / дор

2. Построение аналоговой модели

Зависимость между определяемыми и определяющими показателями эволюции программной системы аппроксимируется как

$$t = A \cdot s^a \cdot d_c^b \cdot \delta_d^c \cdot r_p^d \cdot p_v^e \cdot b^f \cdot r_b^g \cdot c_v^h, \quad (3.25)$$

Уравнение (3.25) записанное в терминах размерностей входящих в него величин приводит к соотношению вида

$$\text{день} = \text{SLOC}^a \cdot \text{чел}^b \cdot \frac{\text{чел}^c}{\text{день}^c} \cdot \frac{\text{дор}^d}{\text{день}^d} \cdot \frac{\text{SLOC}^e}{\text{дор}^e} \cdot \text{деф}^f \cdot \frac{\text{деф}^g}{\text{день}^g} \cdot \frac{\text{SLOC}^h}{\text{день}^h \cdot \text{чел}^h}. \quad (3.26)$$

Здесь количество размерных величин $n = 9$, количество независимых размерностей $k = 5$. Соответственно, необходимо получить $n-k=4$ критерия.

Система уравнений, составленная из показателей степеней для каждой размерности из уравнения (3.26) имеет вид:

$$\begin{aligned} \text{день} : 1 &= -c - d - g - h \\ \text{SLOC} : 0 &= a + e + h \\ \text{чел} : 0 &= b + c - h \\ \text{деф} : 0 &= f + g \\ \text{дор} : 0 &= d - e \end{aligned} \quad (3.27)$$

Складывая первое уравнение со вторым и второе с третьим в системе (3.27), а также выражая остальные величины из последних трех уравнений, получается решение системы:

$$\begin{aligned} d &= e \\ g &= -f \\ c &= a + f - 1 \\ b &= 1 - e - f - 2a \\ h &= -a - e \end{aligned} \quad (3.28)$$

Подставляя значения показателей из (3.28) в (3.25), получается

$$t = A \cdot s^a \cdot \frac{d_c}{d_c^{2a} \cdot d_c^e \cdot d_c^f} \cdot \frac{\delta_d^a \cdot \delta_d^f}{\delta_d} \cdot r_p^e \cdot b^f \cdot p_v^e \cdot \frac{1}{r_b^f} \cdot \frac{1}{c_v^a \cdot c_v^e}. \quad (3.29)$$

Группируя в (3.29) величины с одинаковыми показателями степени:

$$\frac{t \cdot \delta_d}{d_c} = A \cdot \left(\frac{s \cdot \delta_d}{d_c^2 \cdot c_v} \right)^a \cdot \left(\frac{\delta_d \cdot b}{d_c \cdot r_b} \right)^f \cdot \left(\frac{r_p \cdot p_v}{d_c \cdot c_v} \right)^e. \quad (3.30)$$

где выделяются четыре критерия:

$$F_t = \frac{t \cdot \delta_d}{d_c}; [F_t] = \frac{\text{день} \cdot \text{чел}}{\text{чел} \cdot \text{день}} = 1;$$

$$F_{scope} = \frac{s \cdot \delta_d}{d^2 \cdot c_v}; [F_{scope}] = \frac{\text{SLOC} \cdot \text{чел} \cdot \text{чел} \cdot \text{день}}{\text{день} \cdot \text{чел}^2 \cdot \text{SLOC}} = 1;$$

$$F_{qua} = \frac{\delta_d \cdot b}{d_c \cdot r_b}; [F_{qua}] = \frac{\text{чел} \cdot \text{деф} \cdot \text{день}}{\text{день} \cdot \text{деф} \cdot \text{чел}} = 1;$$

$$F_{ext} = \frac{r_p \cdot p_v}{d_c \cdot c_v}; [F_{ext}] = \frac{\text{дор} \cdot \text{SLOC} \cdot \text{чел} \cdot \text{день}}{\text{день} \cdot \text{дор} \cdot \text{чел} \cdot \text{SLOC}} = 1.$$

Абстрактная аналоговая модель эволюции программной системы общего назначения, разрабатываемой по открытой модели выражается критериальным уравнением

$$F_t = A \cdot F_{scope}^a \cdot F_{qua}^f \cdot F_{ext}^e. \quad (3.31)$$

3.5. Модельные критериальные уравнения для основных классов программных проектов

Модельные критериальные уравнения для основных классов программных проектов, разрабатываемых по обобщенной, быстрой и открытой модели представлены ниже:

- обобщенная (каскадная) модель с прототипированием (3.10):

$$D_t = A \cdot D_{com}^e \cdot D_{prod}^f \cdot D_{compro}^i \cdot D_{prodpro}^j \cdot D_{reus}^n \cdot D_{size}^d \cdot D_{sizepro}^g \cdot D_{qua}^m;$$

- обобщенная (каскадная) модель без прототипирования (3.17):

$$D_{ts} = A \cdot D_{com}^e \cdot D_{prod}^f \cdot D_{reus}^j \cdot D_{qua}^i;$$

- быстрая (agile) модель (3.24):

$$A_{ts} = A \cdot A_{pq}^c;$$

- открытая модель (3.31):

$$F_t = A \cdot F_{scope}^a \cdot F_{qua}^f \cdot F_{ext}^e.$$

Смысл и содержание критериев подобия можно обобщить путем анализа составляющих критерии параметров. Результаты такого обобщения приводятся в таблице 3.5.

Таблица 3.5

Критерии подобия программных систем

Крит. и форм. для расчета	Пояснение
Критерии для обобщенной модели	
$D_t = \frac{t}{t_{cp}}$	Характеристика времени разработки при прототипировании
$D_{ts} = \frac{t \cdot l_f}{s}$	Характеристика времени разработки без прототипирования
$D_{com} = \frac{c_x \cdot d}{l_f}$	Характеристика сложности проекта
$D_{compro} = \frac{c_x \cdot d_{cp}}{l_f}$	Характеристика сложности прототипа
$D_{prod} = \frac{p}{c_x}$	Характеристика производительности разработчиков
$D_{prodpro} = \frac{p_{cp}}{c_x}$	Характеристика производительности разработчиков при прототипировании
$D_{qua} = \frac{x_p}{q \cdot c_x}$	Характеристика качества продукта
$D_{reus} = \frac{c_c}{r_c \cdot r_v}$	Характеристика повторного использования кода
$D_{size} = \frac{s}{t_{cp} \cdot l_f}$	Характеристика размера продукта
$D_{sizepro} = \frac{s_{cp}}{t_{cp} \cdot l_f}$	Характеристика размера прототипа
Критерии для быстрой модели	
$A_{ts} = \frac{t}{s_t \cdot i_{ed} \cdot d}$	Характеристика времени и размера быстрого проекта
$A_{pq} = \frac{r_{ts} \cdot n_t}{s_t}$	Характеристика быстроты и гибкости
Критерии для открытой модели	
$F_t = \frac{t \cdot \delta_d}{d_c}$	Характеристика времени открытого проекта
$F_{scope} = \frac{s \cdot \delta_d}{d^2 \cdot c_v}$	Характеристика масштаба проекта

Крит. и форм. для расчета	Пояснение
$F_{qua} = \frac{\delta_d \cdot b}{d_c \cdot r_b}$	Характеристика качества продукта
$F_{ext} = \frac{r_p \cdot p_v}{d_c \cdot c_v}$	Характеристика объема внешних изменений

3.6. Основные выводы по разделу 3

1. Показано, что теорию подобия можно использовать как математический аппарат исследования процессов выполнения программных проектов по принципу аналогии.
2. Разработаны аналоговые модели оценки времени и стоимости для основных классов программных проектов, которые учитывают не только размер продукта, а его качество, темп разработки и доработки, показатели эффективности проектной команды и другие параметры, определяющие время и стоимость.
3. Сформулированы следующие условия подобия проектов разработки программного обеспечения внутри одного класса:
 - временного подобия – подобия жизненных циклов;
 - подобия констант проекта, связанных с внешней средой;
 - подобия начальных условий при инициализации.
4. Установлено, что показатели степени в критериальных уравнениях будут одинаковы для всех проектов внутри класса проектов, а критерии подобия будут численно равны для всех подобных проектов внутри класса.
5. Показано, что применение аналоговых моделей оценки времени и стоимости разработки программного обеспечения упрощает анализ влияния параметров проекта на время и стоимость.

РАЗДЕЛ 4. ИССЛЕДОВАНИЕ ПРИМЕНИМОСТИ АНАЛОГОВЫХ МОДЕЛЕЙ ДЛЯ УПРАВЛЕНИЯ ВРЕМЕНЕМ

4.1. Описание процесса исследования данных ISBSG

От ISBSG были получены параметры (метрики) 3000 выполненных программных проектов. В приложении В в документе “Repository Data Release 9 - Field Descriptions” описано содержание собранных данных. Этот документ вместе с лицензионным соглашением определяет порядок анализа полученной информации. Ниже описываются действия, выполненные в процессе исследования данных ISBSG.

4.1.1. Разбиение проектов на классы. Все проекты из базы данных ISBSG выполнены по каскадной (обобщенной) модели жизненного цикла и различаются типом производимого продукта. В базе присутствуют проекты доработки имеющегося программного обеспечения, которые необходимо отличать от проектов новых разработок.

Согласно вышеизложенному, группировка проектов в классы производится с помощью двух полей:

- “*Development Type*” – тип разработки (новая, улучшение, переработка);
- “*Application Type*” – тип приложения (информационная система, система обработки транзакций, управление процессами и т.д.).

Эти два поля определяют характеристики соответственно проекта и продукта, а фильтрация (выбор проектов с одинаковыми значениями полей) по ним выделяет проекты внутри одного класса из базы данных.

4.1.2. Определение групп подобных проектов. Выделяется 4 группы полей, фильтрация по которым обеспечивает выполнение условий подобия:

- “*Project Activity Scope*” – способ измерений трудозатрат;

- “*EffortPlan*”, “*EffortSpecify*”, “*EffortDesign*”, “*EffortBuild*”, “*EffortTest*”, “*EffortImplement*”, “*EffortUnphased*” – процентное соотношение времени, выделенного в жизненном цикле на планирование, спецификацию, проектирование, разработку, тестирование, внедрение и другие действия;
- “*Architecture*”, “*ClientServer*”, “*WebDevelopment*” – характеристики архитектуры целевого окружения, для которого разрабатывается продукт;
- “*LanguageType*”, “*PrimaryProgrammingLanguage*” – тип и название используемого языка программирования.

Первые две группы полей обеспечивают выполнение условий временного подобия, третья группа – подобие констант внешней среды, а четвертая – подобие начальных условий.

4.1.3. Дополнительные условия фильтрации данных. База данных ISBSG содержит записи о проектах с неполной или недостаточно корректной информацией. Для анализа выбираются только проекты с полной и достоверной информацией, что обеспечивается фильтрацией по следующим полям:

- “*Data Quality Rating*” – качество проектных данных;
- “*Unadjusted Function Point Rating*” – рейтинг качества измерения размера программного кода в функциональных точках.

Для обоих полей выбираются только проекты с наиболее высоким рейтингом А, для которых отсутствуют известные факторы, влияющие на качество.

4.1.4. Выбор метрик для аналогового моделирования. Для моделирования оценки времени из базы данных ISBSG выбираются следующие размерные показатели, влияющие на время разработки:

- “*Functional Size*” – размер производимого продукта в функциональных точках;
- “*Normalised Productivity Delivery Rate (functional size units)*” – производительность разработчиков измеряемая в часах, затраченных на одну функциональную точку;
- “*Average Team Size*” – средний размер команды разработчиков;
- “*Total Defects Delivered*” – количество дефектов в готовом продукте.

Используя эти показатели учитывался опыт разработчиков согласно [62].
Дополнительными показателями “*LanguageType*” и “*PrimaryProgrammingLanguage*” учитывался фактор языка программирования согласно [11, 95].

Таким образом, для аналогового моделирования проектов из базы данных ISBSG выбраны следующие размерные показатели (см. таблицу 4.1).

Таблица 4.1

Размерные показатели и система единиц измерения для аналогового моделирования проектов из базы данных ISBSG

Показатель	Обозначение	Размерность
Длительность разработки (определяемый параметр)	<i>t</i>	час
Размер	<i>s</i>	точки
Число разработчиков	<i>d</i>	чел
Производительность разработчиков	<i>p</i>	час / точки
Языковой фактор	<i>l</i>	точки / (чел день)
Опыт разработчиков	<i>f</i>	деф / точка
Качество	<i>q</i>	деф

Из сопоставления параметров из таблицы 4.1 с параметрами для проектов в обобщенной модели из таблицы 3.2 следует, что в уравнении для проектов в обобщенной модели (3.17) количество параметров больше, чем имеется в наличии. Таким образом, необходимо упростить эту модель для анализа.

4.1.5. Упрощенная аналоговая модель для проектов из базы данных ISBSG. Зависимость между определяемыми и определяющими показателями аппроксимируется как

$$t = A \cdot s^a \cdot d^b \cdot q^c \cdot p^e \cdot f^g \cdot l^h \quad (4.1)$$

Уравнение (4.1) в терминах размерностей входящих в уравнение величин представляется соотношением вида

$$\text{час} = A \cdot \text{точки}^a \cdot \text{чел}^b \cdot \text{деф}^c \cdot \frac{\text{час}^e}{\text{точки}^e} \cdot \frac{\text{деф}^g}{\text{точки}^g} \cdot \frac{\text{точки}^h}{\text{чел}^h \cdot \text{час}^h} \quad (4.2)$$

Здесь количество размерных величин $n = 7$, количество независимых размерностей $k = 4$, соответственно было получено $n-k=3$ критерия.

Система уравнений, составленная из показателей степеней для каждой размерности из уравнения (4.2):

$$\begin{aligned}
 \text{час: } & 1 = e - h \\
 \text{точки: } & 0 = 1 + h - e - g \\
 \text{чел: } & 0 = c + g \\
 \text{деф: } & 0 = b - h
 \end{aligned}
 \tag{4.3}$$

Система решается складывая первое уравнение со вторым и второе с третьим в системе (3.21), а также выражая остальные величины из последних трех уравнений:

$$\begin{aligned}
 e &= h + 1 \\
 c &= -g \\
 b &= h \\
 a &= 1 + g
 \end{aligned}
 \tag{4.4}$$

Подставляя значения показателей из (4.4) в (4.1):

$$t = A \cdot s \cdot s^g \cdot d^h \cdot \frac{1}{q^g} \cdot p^h \cdot p \cdot f^g \cdot l^h
 \tag{4.5}$$

Группируя в (4.5) величины с одинаковыми показателями степени:

$$\frac{t}{sp} = A \cdot \left(\frac{sf}{q} \right)^g \cdot (pld)^h
 \tag{4.6}$$

где критерии подобия:

$$D_t = \frac{t}{sp} \quad \text{– критерий времени;}$$

$$D_c = \frac{sf}{q} \quad \text{– критерий сложности разрабатываемого продукта;}$$

$$D_p = pld \quad \text{– критерий производительности команды разработчиков.}$$

Полученные критерии безразмерны:

$$[D_t] = \frac{\text{hour} \cdot \text{ufps}}{\text{ufps} \cdot \text{hour}} = 1,$$

$$[D_c] = \frac{\text{ufps} \cdot \text{bugs}}{\text{bugs} \cdot \text{ufps}} = 1,$$

$$[D_p] = \frac{\text{hour}}{\text{ufps}} \cdot \frac{\text{ufps}}{\text{man} \cdot \text{hour}} \cdot \text{man} = 1.$$

Аналоговая модель для проектов ISBSG выражается уравнением

$$D_i = A \cdot (D_c)^g \cdot (D_p)^h. \quad (4.7)$$

4.1.6. Определение показателей степени критериального уравнения. Для каждой группы подобных проектов вычислены значения критериев D_i , D_p и D_c .

Значения каждого из полученных критериев D_i рассмотрены как члены статистической совокупности с распределением Стьюдента, для которой рассчитано выборочное среднее, исправленное среднеквадратическое отклонение и оценивается математическое ожидание при помощи доверительного интервала с надежностью $\gamma = 0.95$. Такое распределение выбрано согласно рекомендациям [96] для малых выборок (с $n < 30$, где n – объем выборки). Расчет выборочной средней, исправленного среднеквадратического отклонения и границ доверительного интервала производился согласно [96]:

$$\begin{aligned} \bar{x} &= \frac{\sum_{i=1}^n x_i}{n}; \\ s &= \sqrt{\frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n-1}}; \\ a_{min} &= \bar{x} - t_\gamma \cdot \frac{s}{\sqrt{n}}; a_{max} = \bar{x} + t_\gamma \cdot \frac{s}{\sqrt{n}}, \end{aligned} \quad (4.8)$$

где \bar{x} – выборочное среднее,

x_i – i -й элемент выборки,

n – число элементов в выборке,

s – исправленное среднеквадратическое отклонение,

t_γ – функция точности оценки, зависящая от числа элементов в выборке и надежности γ ; выбирается по таблицам из [96];

a_{min} – нижняя граница доверительного интервала;

a_{max} – верхняя граница доверительного интервала.

В случае, если количество выбранных проектов группы не превышает трех, доверительный интервал не вычислялся.

Из средневыборочных значений критериев для всех групп составлена система m (m -число групп) нелинейных уравнений вида

$$\begin{aligned}
 D_{i1} &= A \cdot (D_{c1})^g \cdot (D_{p1})^h \\
 D_{i2} &= A \cdot (D_{c2})^g \cdot (D_{p2})^h \\
 &\dots \\
 D_{im} &= A \cdot (D_{cm})^g \cdot (D_{pm})^h
 \end{aligned}
 \tag{4.9}$$

Эта система решена методом взвешенных градиентов [97, 98, 99], в результате чего получены численные значения коэффициента A и показателей степени g и h .

4.1.7. Оценка времени и анализ точности оценок. Время t_{am} , оцененное аналоговой моделью (4.7) определялось как:

$$t_{am} = A \cdot s \cdot p \cdot D_c^g \cdot D_p^h. \tag{4.10}$$

Точность оценки определялась сопоставлением t_{am} с реальным временем разработки t_r и t_{cocomo} – временем, оцененным моделью COCOMO [9]. Реальное время разработки в базе данных ISBSG выбрано из поля “Normalised Work Effort”. Для сопоставления определялось максимальное относительное отклонение и строились зависимости $t_r = f(t_r)$, $t_r = f(t_{am})$ и $t_r = f(t_{cocomo})$ на одном графике.

Также вычислялись относительные погрешности оценок, математическое ожидание и его доверительный интервал с надежностью $\gamma = 0.95$. Математическое ожидание погрешности модели (4.7) сравнивалось с аналогичным показателем для модели COCOMO.

4.2. Критериальные уравнения оценки времени проектов ISBSG и обоснование их достоверности

Критериальные уравнения оценки времени получены для пяти классов проектов:

1. проекты разработки систем обработки транзакций (управление, контроль информации) со значениями полей:

Development Type = New Development

Application Type = ManagementInformationSystem; Transaction/Production System;

2. проекты доработки систем обработки транзакций (управление, контроль информации) со значениями полей:

Development Type = Enhancement; Re-Development

Application Type = ManagementInformationSystem; Transaction/Production System;

3. проекты разработки систем поддержки принятия решений:

Development Type = New Development

Application Type = Decision Support System;

4. проекты разработки систем автоматизации деятельности офисов:

Development Type = New Development

Application Type = Office Information System;

5. проекты разработки систем управления производственными процессами:

Development Type = New Development

Application Type = Process Control;

Результаты разбиения проектов на группы для всех классов, соответствующие им значения критериев и их доверительные интервалы приведены в таблицах 4.2 – 4.11, а показатели степени критериальных уравнений – в таблице 4.12.

В результате анализа для каждого класса проектов получены критериальные уравнения:

1. проекты разработки систем обработки транзакций:

$$D_t = 308 \cdot D_c^{-0.669} \cdot D_p^{-0.209} \quad (4.11)$$

2. проекты доработки систем обработки:

$$D_t = 843 \cdot D_c^{-1.114} \cdot D_p^{-0.107} \quad (4.12)$$

3. проекты разработки систем поддержки принятия решений:

$$D_t = 631 \cdot D_c^{-0.82} \cdot D_p^{-0.239} \quad (4.13)$$

4. проекты разработки систем автоматизации деятельности офисов:

$$D_t = 857 \cdot D_c^{-0.427} \cdot D_p^{-0.757} \quad (4.14)$$

5. проекты разработки систем управления производственными процессами:

$$D_t = 219 \cdot D_c^{-0.572} \cdot D_p^{-0.307} \quad (4.15)$$

Для обоснования достоверности полученных моделей (4.11) – (4.15) согласно 4.1.7 выполнены оценки времени t_{am} для каждого проекта, сопоставление которых приведено на рис. 4.1–4.5.

Сопоставление относительных погрешностей оценок показывает, что средняя погрешность составила 32% для аналоговой модели и 87% для модели СОСОМО. Математическое ожидание погрешности с достоверностью 0.95 находилось в интервале [25%..39%] для аналоговой модели и [60%..114%] для модели СОСОМО. При этом наибольшее отклонение оценок от действительных данных для аналоговой модели составило 91.6%, а для СОСОМО – 621%.

Оценки аналоговой модели точнее для 80% проектов, а для остальных 20% разница между оценкой аналоговой модели и модели СОСОМО не превышает 51% и в среднем составляет 21%. Аналоговая модель в среднем дает оценку на 76% лучшую по сравнению с моделью СОСОМО.

Таблица 4.2

Класс 1. Проекты разработки систем обработки транзакций (управление, контроль информации). Критерии подобия, оценки времени, отклонения

№ п/п	t_r , час	D_c	D_p	D_t	t_{am} , час	t_{cocomo} , час	δ_{am} , %	δ_{cocomo} , %
Группа 1								
1	6924	658	925	0,999	6671	3275	3,7	52,7
2	21014	809	516	0,989	20123	5672	4,2	73
3	15165	621	1366	1,004	13937	1412	8	90,6
4	5078	305	855	1,000	8298	2630	63,2	48,2
5	2449	1049	410	1,000	2044	2619	16,5	6,9
Группа 2								
6	13528	546	14128	1,000	8261	4060	38,3	70
7	18521	367	13277	1,000	14960	6390	18,5	66
Группа 3								
8	25401	164	45360	0,999	27845	13523	8,2	46,7
Группа 4								
9	1076	1229	156	1,004	963	1259	10,5	17
Группа 5								
10	2385	437	1950	1,004	2524	220	5,8	90,7
Группа 6								
11	4900	514	340	1,004	6694	2831	36,6	42,2
12	6068	665	784	0,999	5894	1561	2,86	74,3
13	7824	942	359	0,996	7095	4306	9,32	44,9

Таблица 4.3

Класс 1. Проекты разработки систем обработки транзакций (управление, контроль информации). Расчетные критерии и их доверительные интервалы

Группа	D_c	D_{cmin}	D_{cmax}	D_p	D_{min}	D_{max}	D_t	D_{tmin}	D_{tmax}
1	688	449	927	814	483	1145	1	0.995	1.005
2	457	–	–	13703	–	–	1	–	–
3	164	–	–	45360	–	–	1	–	–
4	1229	–	–	156	–	–	1	–	–
5	2385	–	–	1950	–	–	1	–	–
6	579	273	885	555	322	788	1	0.997	1.003

Таблица 4.4

Класс 2. Проекты доработки систем обработки транзакций (управление, контроль информации). Критерии подобия, оценки времени, отклонения

№ п/п	t_r , час	D_c	D_p	D_t	$t_{ам}$, час	$t_{сосопо}$, час	$\delta_{ам}$, %	$\delta_{сосопо}$, %
Группа 1								
1	7231	244	1733	1,000	6006	2960	16,9	59,1
2	450	187	2491	0,997	484	79	7,7	82,3
3	3890	287	1976	0,999	2661	1650	31,6	57,6
4	23491	179	1848	0,998	27413	6386	16,7	72,8
5	11045	170	1726	1,001	13721	964	24,2	91,2
Группа 2								
6	1171	225	290	0,999	1290	1838	10,2	57,0
7	1417	204	348	1,002	1701	586	20,1	58,6
Группа 3								
8	1464	413	440	1,004	780	1126	46,7	23,1
Группа 4								
9	1238	1482	66	0,987	198	8925	84	620
Группа 5								
10	3639	193	3516	1,001	3626	338	0,3	90
Группа 6								
11	201	299	77	1,028	180	811	9,9	303,6
12	1774	179	115	1,004	2763	2595	55,8	46,3
13	247	152	98	1,000	473	760	91,6	208
Группа 7								
14	1717	761	282	1,000	488	2768	71,5	61,3
15	799	576	293	1,030	299	415	62,5	48,1

Таблица 4.5

Класс 2. Проекты доработки систем обработки транзакций (управление, контроль информации). Расчетные критерии и их доверительные интервалы

Группа	D_c	D_{cmin}	D_{cmax}	D_p	D_{min}	D_{max}	D_t	D_{tmin}	D_{tmax}
1	213	169	257	1955	1677	2233	1	0,998	1,002
2	215	–	–	319	–	–	1	–	–
3	413	–	–	440	–	–	1	–	–
4	1482	–	–	66	–	–	1	–	–
5	193	–	–	3516	–	–	1	–	–
6	210	133	287	96	77	115	1	0,985	1,015
7	668	–	–	287	–	–	1	–	–

Таблица 4.6

Класс 3. Проекты разработки систем поддержки принятия решений.

Критерии подобию, оценки времени, отклонения

№ п/п	t_r , час	D_c	D_p	D_t	$t_{ам}$, час	$t_{сосоm}$, час	$\delta_{ам}$, %	$\delta_{сосоm}$, %
Группа 1								
1	700	710	24	1,012	940	994	34,3	42,0
Группа 2								
2	950	1495	153	0,971	462	1005	51,3	5,8
Группа 3								
3	13661	1045	1407	1,002	5084	3458	62,8	74,7
4	2538	1119	1214	1,004	922	2215	63,6	12,7
Группа 4								
5	3877	486	2340	0,998	2406	547	37,9	85,9
6	7654	634	2256	1,001	3837	542	49,8	92,9
Группа 5								
7	4551	129	7137	0,999	6392	313	40,5	93,1
8	3323	151	8679	0,999	3920	52	17,9	98,4

Таблица 4.7

Класс 3. Проекты разработки систем поддержки принятия решений.**Расчетные критерии и их доверительные интервалы**

Группа	D_c	D_{cmin}	D_{cmax}	D_p	D_{min}	D_{max}	D_t	D_{tmin}	D_{tmax}
1	710	–	–	24	–	–	1	–	–
2	1495	–	–	153	–	–	1	–	–
3	1082	–	–	1311	–	–	1	–	–
4	560	–	–	2298	–	–	1	–	–
5	140	–	–	7907	–	–	1	–	–

Таблица 4.8

Класс 4. Проекты разработки систем автоматизации деятельности офисов.**Критерии подобия, оценки времени, отклонения**

№ п/п	t_r , час	D_c	D_p	D_t	$t_{ам}$, час	$t_{сосопо}$, час	$\delta_{ам}$, %	$\delta_{сосопо}$, %
Группа 1								
1	7490	976	1050	0,997	1758	994	76,5	86,7
Группа 2								
2	1754	136	314	0,997	2383	1857	35,9	5,9
Группа 3								
3	10576	350	1019	0,995	3941	2641	62,8	75,0
Группа 4								
4	7060	3186	1032	1,004	1005	1128	85,7	84,0
Группа 5								
5	1050	739	129	0,989	1369	1529	30,5	45,7
6	1058	669	150	1,012	1254	1078	18,6	1,96
7	3930	636	185	1,002	4103	1801	4,4	54,1

Таблица 4.9

Класс 4. Проекты разработки систем автоматизации деятельности офисов.**Расчетные критерии и их доверительные интервалы**

Группа	D_c	D_{cmin}	D_{cmax}	D_p	D_{min}	D_{max}	D_t	D_{tmin}	D_{tmax}
1	976	–	–	1050	–	–	1	–	–
2	136	–	–	314	–	–	1	–	–
3	350	–	–	1019	–	–	1	–	–
4	3186	–	–	1032	–	–	1	–	–
5	681	635	727	155	130	180	1	–	–

Таблица 4.10

Класс 5. Проекты разработки систем управления производственными процессами. Критерии подобия, оценки времени, отклонения

№ п/п	t_r , час	D_c	D_p	D_t	$t_{ам}$, час	$t_{сосоmо}$, час	$\delta_{ам}$, %	$\delta_{сосоmо}$, %
Группа 1								
1	22500	106	8580	0,999	21253	994	5,5	95,6
Группа 2								
2	3742	1292	149	1,004	2920	12255	21,9	227,5
Группа 3								
3	1200	2359	22	0,992	1214	2773	1,17	131,16
Группа 4								
4	3214	440	264	0,995	3925	4260	22,1	32,6
5	4630	459	256	1,006	5512	9434	19,1	103,8

Таблица 4.11

Класс 5. Проекты разработки систем управления производственными процессами. Расчетные критерии и их доверительные интервалы

Группа	D_c	$D_{сmin}$	$D_{сmax}$	D_p	D_{min}	D_{max}	D_t	D_{tmin}	D_{tmax}
1	106	–	–	8580	–	–	1	–	–
2	1292	–	–	149	–	–	1	–	–
3	2359	–	–	22	–	–	1	–	–
4	450	–	–	260	–	–	1	–	–

Таблица 4.12

Значения коэффициентов пропорциональности и показателей степени в критериальных уравнениях для классов проектов

Класс проектов	g	h	A
Класс 1. Проекты разработки систем	-0,669	-0,209	308
Класс 2. Проекты доработки СОТ	-1,114	-0,107	843
Класс 3. Проекты разработки систем	-0,82	-0,239	631
Класс 4. Проекты разработки систем	-0,427	-0,757	857
Класс 5. Проекты разработки систем	-0,572	-0,307	219

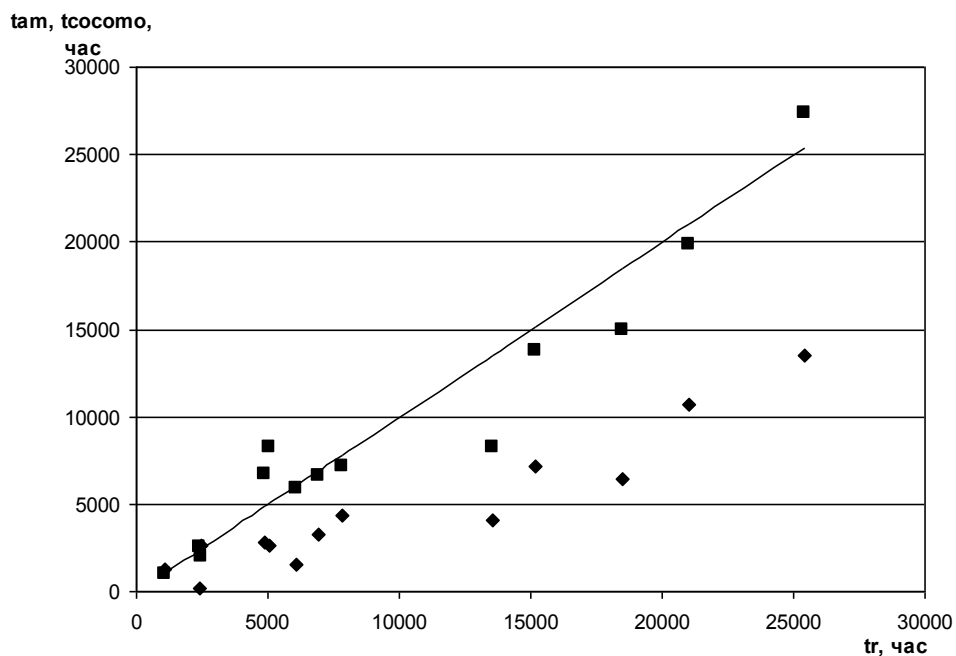


Рис. 4.1. Сопоставление оценок аналоговой модели, модели COSCOMO и действительных проектных данных проектов класса 1:

□ – оценки аналоговой модели (t_{am});

◇ – оценки модели COSCOMO (t_{cocomo}).

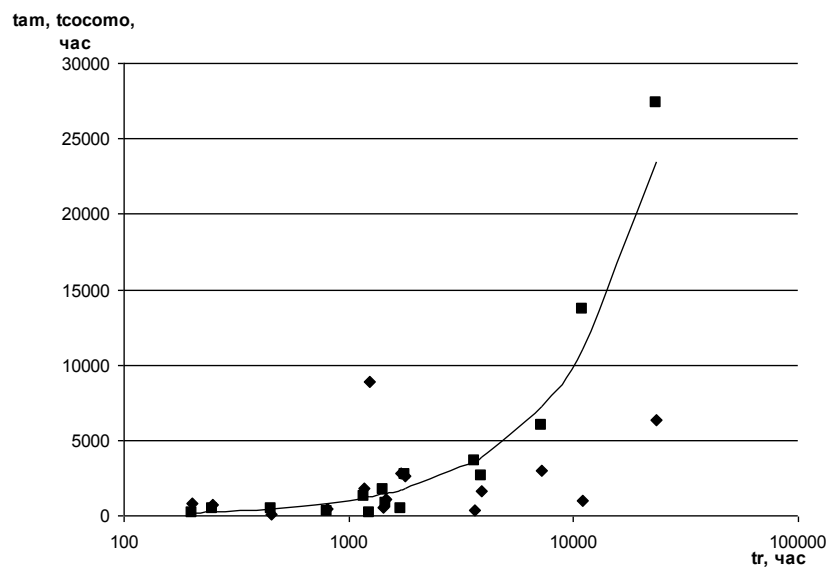


Рис. 4.2. Сопоставление оценок аналоговой модели, модели COSCOMO и действительных проектных данных проектов класса 2:

□ оценки аналоговой модели (t_{am});

◇ – оценки модели COSCOMO (t_{cocomo}).

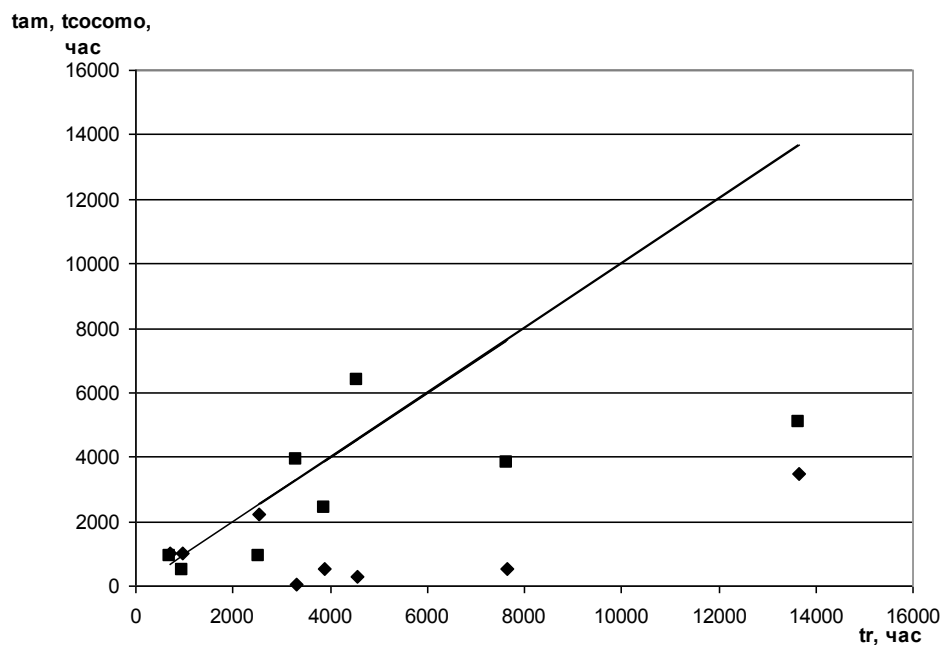


Рис. 4.3. Сопоставление оценок аналоговой модели, модели COSCOMO и действительных проектных данных проектов класса 3:

- – оценки аналоговой модели (t_{am});
- ◇ – оценки модели COSCOMO (t_{cocomo}).

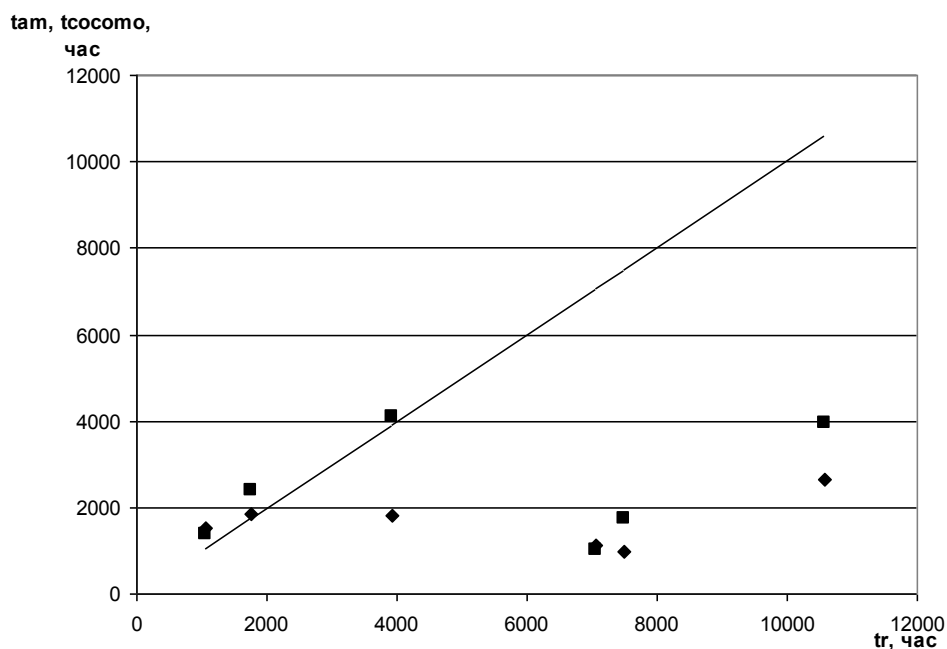


Рис. 4.4. Сопоставление оценок аналоговой модели, модели COSCOMO и действительных проектных данных проектов класса 4:

- – оценки аналоговой модели (t_{am});
- ◇ – оценки модели COSCOMO (t_{cocomo}).

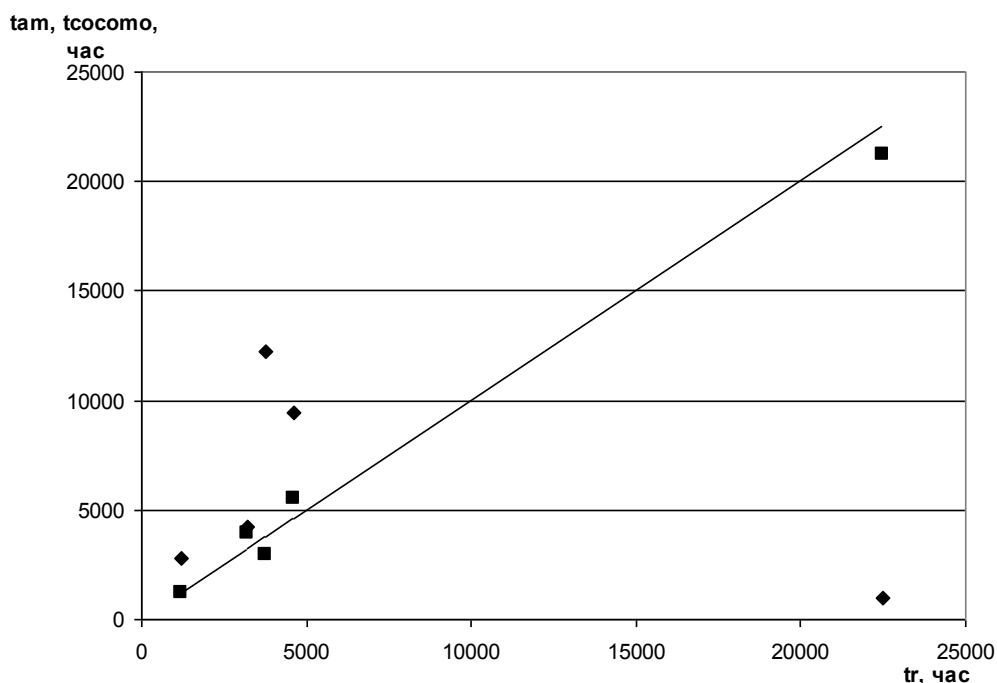


Рис. 4.5. Сопоставление оценок аналоговой модели, модели COSCOMO и действительных проектных данных проектов класса 5:

- – оценки аналоговой модели (t_{am});
- ◇ – оценки модели COSCOMO (t_{cocomo}).

4.3. Критериальные уравнения оценки времени программных проектов с открытым кодом

Сбор информации о свободных проектах с открытым кодом произведен на основании данных, предоставляемых проектами на их веб-сайтах и путем поиска в репозиториях исходного кода.

Для исследования выбраны 2 группы подобных проектов с открытым кодом:

1. проекты разработки интегрированных средств разработки;
2. проекты разработки инструментальных библиотек.

Собранные данные приведены в приложении Г.

Используя данные из приложения Г, вычислены критерии подобия открытой модели и определены их доверительные интервалы.

Для группы проектов разработки интегрированных средств разработки:

- критерий времени $F_t = 0.26$, его доверительный интервал – [0.24..0.28];
- критерий масштаба проекта $F_{scope} = 0.33$, его доверительный интервал – [0.30..0.36];
- критерий качества продукта $F_{qua} = 0.0003$, его доверительный интервал – [0.00022..0.00038];
- критерий объема внешних изменений $F_{ext} = 1.25$, его доверительный интервал – [1.23..1.27].

Для группы проектов разработки инструментальных библиотек:

- критерий времени $F_t = 0.61$, его доверительный интервал – [0.58..0.64];
- критерий масштаба проекта $F_{scope} = 0.76$, его доверительный интервал – [0.73..0.79];
- критерий качества продукта $F_{qua} = 0.00015$, его доверительный интервал – [0.00012..0.00018];
- критерий объема внешних изменений $F_{ext} = 1.24$, его доверительный интервал – [1.23..1.25].

Средняя погрешность оценок времени относительно реальных данных проектов составила:

- для проектов разработки интегрированных средств разработки – 14%;
- для проектов разработки инструментальных библиотек – 16%.

Аналоговые модели для проектов с открытым кодом приводятся ниже:

- для проектов разработки интегрированных средств разработки:

$$F_t = 784 \cdot F_{scope}^{1.97} \cdot F_{qua}^{0.64} \cdot F_{ext}^{-3.1}. \quad (4.16)$$

- для проектов разработки инструментальных библиотек:

$$F_t = 911 \cdot F_{scope}^{2.1} \cdot F_{qua}^{0.68} \cdot F_{ext}^{-2.8}. \quad (4.17)$$

4.4. Основные выводы по разделу 4

1. Определены численные значения параметров аналоговых моделей проектов разработки и доработки систем обработки транзакций, поддержки принятия решений, автоматизации деятельности офисов и управления процессами на производстве с средней погрешностью оценок времени ниже на 55% погрешности существующей модели СОСОМО.
2. Установлено, что оценки, выполненные с помощью аналоговых моделей точнее, чем оценки используемой в данное время модели СОСОМО, так как для 80% выбранных для анализа проектов оценка аналоговой модели точнее, а для остальных 20% ее значение в среднем на 20% хуже. При этом максимальная погрешность оценок аналоговых моделей не превышает 92%, тогда как модель СОСОМО давала результат с погрешностью 612% (при заявленной погрешности до 400% [9] для предварительных оценок).
3. Получены аналоговые модели проектов разработки программного обеспечения с открытым кодом для двух групп подобных проектов – разработки интегрированных средств разработчика и инструментальных библиотек, которые за счет повышенной точности оценки времени и стоимости способны повысить эффективность управления.

РАЗДЕЛ 5. МЕТОДИКА УПРАВЛЕНИЯ ПРОГРАММНЫМИ ПРОЕКТАМИ С ОТКРЫТЫМ КОДОМ

Усовершенствования методика управления программными проектами с открытым кодом включает разделы:

- обоснование выбора открытой модели методом анализа отличительных категорий;
- анализ риска выбора открытой модели, определение объема планирования и открытости;
- выбор типа лицензирования продукта;
- прототипирование и инициализация среды выполнения проекта.

5.1. Обоснование выбора открытой модели методом анализа отличительных категорий

Для определения возможности успешного завершения проекта необходимо доказать применимость открытой модели.

Обоснование применимости модели выполняется по уточненному методу анализа отличительных категорий [62], в сравнительные таблицы которого добавляется информация об открытой модели (см. таблицу 5.1). При этом выполняются следующие действия:

1. Анализ отличительных категорий проекта на основе сравнительных таблиц:
 - 1.1. категории требований;
 - 1.2. категории команд разработчиков;
 - 1.3. категория коллектива разработчиков;
 - 1.4. категории типа проекта и рисков.

2. Расположение по степени важности категорий или вопросов, относящихся к каждой категории, относительно проекта, для которого выбирается приемлемая модель.
3. Подсчет показателей (за/против) каждой модели и разрешение противоречий при сходных показателях с помощью градации категорий и вопросов по степени важности.

Таблица 5.1

Выбор модели жизненного цикла на основе

Требования	Каскадная	V-образная	Прототипирование	Спиральная	RAD	Инкрементная	Открытая
Характеристики требований:							
Являются ли требования легко определяемыми и/или хорошо известными?	Да	Да	Нет	Нет	Да	Нет	Нет
Могут ли требования заранее определяться в цикле?	Да	Да	Нет	Нет	Да	Да	Нет
Часто ли будут изменяться требования в цикле?	Нет	Нет	Да	Да	Нет	Нет	Да
Нужно ли демонстрировать требования с целью определения?	Нет	Нет	Да	Да	Да	Нет	Да
Требуется ли для демонстрации возможностей проверка концепции?	Нет	Нет	Да	Да	Да	Нет	Да
Будут ли требования отражать сложность системы?	Нет	Нет	Да	Да	Нет	Да	Да
Обладает ли требование функциональными свойствами на раннем этапе?	Нет	Нет	Да	Да	Да	Да	Да
Характеристики команд разработчиков							
Являются ли проблемы предметной области проекта новыми для большинства разработчиков?	Нет	Нет	Да	Да	Нет	Нет	Нет

Требования	Каскадная	V-образная	Протогиширование	Спиральная	RAD	Инкрементная	Открытая
Является ли технология предметной области новой для большинства разработчиков?	Да	Да	Нет	Да	Нет	Да	Нет
Являются ли инструменты, используемые проектом, новыми для большинства разработчиков?	Да	Да	Нет	Да	Нет	Нет	Да
Изменяются ли роли участников проекта во время жизненного цикла?	Нет	Нет	Да	Да	Нет	Да	Да
Могут ли разработчики проекта пройти обучение?	Нет	Да	Нет	Нет	Да	Да	Да
Является ли структура более значимой для разработчиков, чем гибкость?	Да	Да	Нет	Нет	Нет	Да	Нет
Будет ли менеджер проекта строго отслеживать прогресс команды?	Да	Да	Нет	Да	Нет	Да	Нет
Важна ли легкость распределения ресурсов?	Да	Да	Нет	Нет	Да	Да	Да
Приемлет ли команда равноправные обзоры и инспекции, менеджмент/обзоры заказчиков, а также стадии?	Да	Да	Да	Да	Да	Нет	Да
Имеет ли команда разработчиков опыт виртуальной/распределенной работы	Да	Да	Нет	Да	Нет	Нет	Да
Имеются ли в распоряжении инструментальные средства организации распределенной работы	Да	Да	Нет	Да	Нет	Нет	Да
Входит ли в политику компании предоставлять общий доступ к репозитарию кода для разработчиков	Нет	Нет	Да	Нет	Да	Да	Да
Характеристики коллектива пользователей							
Будет ли присутствие пользователей ограничено в жизненном цикле?	Да	Да	Нет	Да	Нет	Да	Нет
Будут ли пользователи знакомы с определением системы?	Нет	Нет	Да	Да	Нет	Да	Да

Требования	Каскадная	V-образная	Прототипирование	Спиральная	RAD	Инкрементная	Открытая
Будут ли пользователи знакомы с проблемами предметной области?	Нет	Нет	Да	Нет	Да	Да	Да
Будут ли пользователи вовлечены во все фазы жизненного цикла?	Нет	Нет	Да	Нет	Да	Нет	Да
Будет ли заказчик отслеживать ход выполнения проекта?	Нет	Нет	Да	Да	Нет	Нет	Да
Характеристики типа проекта и рисков:							
Будет ли проект идентифицировать новое направление для организации?	Нет	Нет	Да	Да	Нет	Да	Да
Будет ли проект иметь тип системной интеграции?	Нет	Да	Да	Да	Да	Да	Да
Будет ли проект являться расширением существующей системы?	Нет	Да	Нет	Нет	Да	Да	Да
Будет ли финансирование проекта стабильным на всем протяжении?	Да	Да	Да	Нет	Да	Нет	Нет
Ожидается ли длительная эксплуатация продукта в организации?	Да	Да	Нет	Да	Нет	Да	Да
Должна ли быть высокая степень надежности?	Нет	Да	Нет	Да	Нет	Да	Да
Будет ли система изменяться, возможно, с применением непредвиденных методов, на этапе сопровождения?	Нет	Нет	Да	Да	Нет	Да	Нет
Является ли график ограниченным?	Нет	Нет	Да	Да	Да	Да	Нет
Являются ли "прозрачными" интерфейсные модули?	Да	Да	Нет	Нет	Нет	Да	Нет
Доступны ли повторно используемые компоненты?	Нет	Нет	Да	Да	Да	Нет	Да
Являются ли достаточными ресурсы (время, деньги, инструменты, персонал)?	Нет	Нет	Да	Да	Нет	Нет	Нет

5.2. Анализ риска выбора открытой модели, определение объема планирования и открытости

После обоснования выбора открытой модели для программного проекта, необходимо определить риски, связанные с этим выбором. В случае значительных рисков необходимо составить план их разрешения либо пересмотреть выбор модели.

Для анализа риска и определения объема открытости и планирования применяется метод анализа риска Боэма и Тюнера [71], дополненный рисками, связанными с открытостью кода проекта.

Объем планирования определяется путем выполнения последовательности действий, приведенной в таблице 5.2 и изображенной в виде алгоритма на рис. 5.1. Метод анализа рисков позволяет определить объем планирования как качественно так и количественно. В результате применения метода для проекта либо для его отдельных подпроектов определяется необходимость в увеличении либо уменьшении объема планирования. В случае если риски открытости доминируют и для проекта не существует хорошей стратегии уменьшения таких рисков, открытая модель разработки применяться не должна и решение об открытости проекта должно быть пересмотрено.

Таблица 5.2

Последовательность действий для определения объема планирования

Действие	Описание
1. Оценка риска	Оценить риски среды, гибкости, планирования и открытости. В случае неоднозначности либо неопределенности в оценках, применить прототипирование или сбор и анализ данных о похожих проектах.
2а. Сравнение	В случае доминирования рисков гибкости, увеличить объем планирования.

Действие	Описание
2б. Сравнение	В случае доминирования рисков планирования, увеличить гибкость, уменьшая планирование.
2в. Сравнение	В случае доминирования рисков открытости, пересмотреть решение об открытости проекта.
3. Анализ	В случае если в проекте существуют элементы, для каждого которых удовлетворяется условия 2а, 2б или 2в, разбить проект на подпроекты.
4. Модификация жизненного цикла	Разработать стратегию разрешения рисков проекта интегрируя планы разрешения каждого из идентифицированных рисков
5. Отслеживание	Отслеживать риски по ходу выполнения проекта и в случае появления дисбаланса, выполнить переоценку объема планирования и открытости

Изложенный в таблице 5.2 алгоритм предоставляет способ интеграции процессов идентификации и оценки риска в стратегию выполнения проекта, однако не позволяет оценить риск количественно, следовательно зависит от квалификации менеджера и его способности к качественной оценке рисков.

Для программных проектов с открытым кодом выделяются следующие источники рисков:

- риски среды, связанные со средой выполнения проекта:
 - $E_{технол}$ – неопределенности используемых технологий;
 - $E_{коорд}$ – координация множества участников проекта;
 - $E_{сложн}$ – сложность системы;
- риски гибкости, связанные с использованием быстрых методов разработки:
 - $A_{масшт}$ – масштабируемость и критичность;
 - $A_{прост}$ – упрощенность проектирования;

- $A_{рот}$ – ротация и текучесть кадров;
- $A_{снос}$ – недостаточные навыки персонала в быстрых методах;



Рис. 5.1. Алгоритм выполнения анализа рисков

- риски планирования, связанные с использованием методов разработки с преобладающей долей планирования:
 - $P_{изм}$ – частота изменений;
 - $P_{скор}$ – необходимость получения быстрых результатов;
 - $P_{возн}$ – возникновение новых требований;
 - $P_{снос}$ – недостаточные навыки персонала в методах с планированием;
- риски открытости, связанные с применением открытых методов разработки:
 - $O_{сист}$ – несистемность производимого продукта;
 - $O_{код}$ – опасность открытости кода;
 - $O_{утеч}$ – возможность появления порожденных проектов;

- $O_{раз}$ – разовый характер производимого продукта;
- $O_{снос}$ – недостаточные навыки персонала в открытых методах.

Выполнение действия 1 обеспечивает основу для принятия решений о стратегии разработки в дальнейшем. Если в проекте риски слишком неопределены, необходимо получить дополнительную информацию с помощью прототипирования либо собрать (приобрести) дополнительные данные. Для выполнения действия следует иметь контрольные перечни рисков и график потоков в виде, рекомендуемом РМВОК [27].

В процессе выполнения действия 2, риски из приведенного выше списка оцениваются и сравниваются. Задачей сравнения является выяснение преобладающих рисков. Само сравнение выполняется на качественном уровне, и по его результатам рискам присваивается рейтинг – значение от 0 до 4, где

- 0 – минимальный риск;
- 1 – средний риск;
- 2 – серьезный, но управляемый риск;
- 3 – очень серьезный, но управляемый риск;
- 4 – неуправляемый риск.

Результаты сводятся в форму, общий вид которой приведен в таблице 5.3.

Таблица 5.3

Общий вид формы оценки рисков

Риск	Рейтинг
$E_{технол}$ – неопределенности используемых технологий	0-4
$E_{коорд}$ – координация множества участников проекта	0-4
...	...

Действие 3 выполняется в том случае, когда ни одна группа рисков для всего проекта не является доминирующей. Это происходит вследствие того, что разные части проекта имеют разные риски. Задачей анализа является разбиение проекта

на подпроекты с преобладающими рисками гибкости, открытости и планирования и обеспечение возможности выработки новой стратегии управления, минимизирующей риски для каждого подпроекта.

При выполнении 4-го действия вырабатывается стратегия управления проектом либо группой подпроектов, минимизирующая идентифицированные риски. Для каждой группы рисков – открытости, гибкости и планирования, принимаются решения об, соответственно, закрытии частей или всего проекта, увеличения и уменьшения объема планирования. Затем для каждого источника риска разрабатывается стратегия его разрешения и вносятся изменения в жизненный цикл проекта путем добавления необходимых процессов либо разрешения либо дополнительного контроля риска.

Использование рисков для определения объема планирования. Для количественного определения объема планирования необходимо рассчитать ожидаемое денежное значение двух групп рисков – рисков недостаточного планирования $R_{план}$ и раннего выведения продукта на рынок $R_{рын}$:

$$R_{план} = P(L) \cdot S(L),$$

$$R_{рын} = P(L) \cdot S(L),$$

где $P(L)$ – вероятность потерь;

$S(L)$ – размер потерь.

Риски $R_{план}$ и $R_{рын}$ рассчитываются для разных величин затраченного времени и трудозатрат на планирование с целью получения зависимостей

$$R_{план} = f(t),$$

$$R_{рын} = f(t),$$

где t – объем трудозатрат либо времени на планирование.

Минимум функции суммарного риска $R(t)$ будет определять необходимый объем планирования $t_{план}$:

$$R(t) = R_{план}(t) + R_{рын}(t);$$

$$R(t_{план}) = \min;$$

$$\left(\frac{\partial R}{\partial t} \right)_{t=t_{план}} = 0.$$

Примерный вид получаемой зависимости и величина объема необходимого планирования изображены на рис. 5.2.

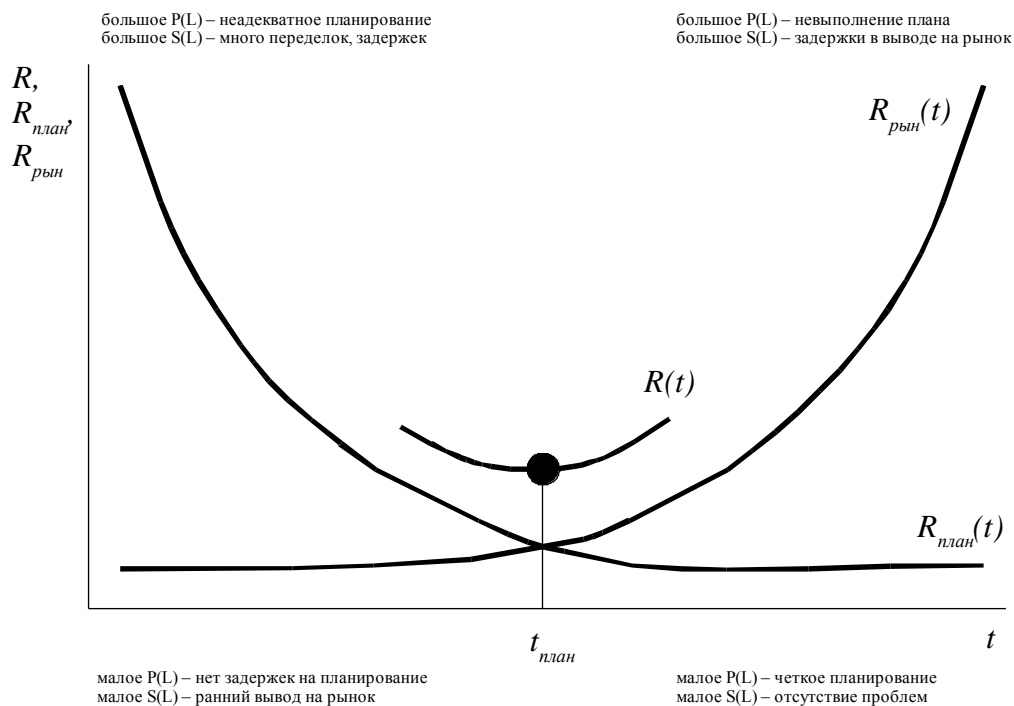


Рис. 5.2. Функция суммарного риска и величина объема необходимого планирования

Анализ зависимости $R_{план}$ на рис. 5.2 показывает, что чем меньше инвестируется в планирование, тем больше вероятность того, что планы будут несовершенными, неоднозначными, иметь пробелы. Также увеличиваются возможные финансовые потери из-за постоянных переработок и уточнений, потерь или ротации персонала. В то же время, чем точнее план, тем меньше вероятность потерь из-за планирования и меньше потери.

Зависимость $R_{рын}$ на рис. 5.2 указывает на то, что малые объемы планирования приводят к раннему выводу разрабатываемого в проекте продукта на рынок и, следовательно, получению большей прибыли. Избыточное планирование приведет к задержкам в выпуске продукта и, следовательно, позволит конкурентам перехватить инициативу и приведет к повышению потерь $S(L)$.

На кривой $R(t)$ суммы ожидаемого денежного значения рисков находится искомая точка минимальных ожидаемых потерь со значением абсциссы $t_{план}$, которая и является искомым количественным выражением необходимого объема планирования.

Рекомендации по выработке стратегий разрешений рисков. Для завершения выполнения действия 4 необходимо разработать план разрешения рисков. Рекомендации для разрешения каждого из источников рисков, дополненные по сравнению с [71] для целей управления проектами с открытым кодом приведены в таблице 5.4.

Таблица 5.4

Рекомендации по выработке стратегий разрешений рисков

Риск	Рекомендация
$E_{технол}$	Более интенсивное прототипирование с целью изучения производительности и масштабируемости технологий. Отслеживание развития технологий и выполнения работ подрядчиками.
$E_{коорд}$	Организация взаимовыгодного сотрудничества, выработка соглашений без ущемления интересов участников проекта, вознаграждения за сотрудничество.
$E_{сложн}$	Архитектурная декомпозиция, уменьшение сложности.
$A_{масшт}$	Архитектурная декомпозиция и разработка интерфейсов для каждой части системы с целью обеспечения параллельности разработки.
$A_{прост}$	Использовать простой дизайн только в подпроектах, где это возможно. Использовать планирование в объеме, определенном методом анализа рисков.
$A_{рот}$	Премии по завершении проекта, постоянное проведение обзоров, парное программирование при возможности.
$A_{спос}$	Ужесточение критериев подбора персонала.
$P_{изм}$	Разработка архитектуры с учетом возможных изменений.
$P_{скор}$	Архитектурная декомпозиция и разработка интерфейсов для каждой части системы с целью обеспечения параллельности разработки.
$P_{возн}$	Спиральная эволюционная разработка.

Риск	Рекомендация
$R_{спос}$	Ужесточение критериев подбора персонала.
$O_{сист}$	Выделение подпроектов с признаками системности производимых продуктов и применение открытой модели для них.
$O_{код}$	Декомпозиция производимого продукта на модули и закрытие кода критических модулей. Разработка архитектуры, допускающей закрытые компоненты и модули.
$O_{утеч}$	Обеспечение прозрачности процесса разработки, доступности проектной информации, легкости интеграции внешних изменений.
$O_{раз}$	Декомпозиция производимого продукта на модули и выделение модулей с возможностью повторного использования.
$O_{спос}$	Ужесточение критериев подбора персонала. Руководство со стороны опытных разработчиков.

5.3. Выбор типа лицензирования продукта

Правильное лицензирование производимого продукта обеспечивает:

- возможность построения инфраструктуры совместной работы над проектом с участием проектной команды и сторонних разработчиков;
- отсутствие юридических трудностей при распространении продукта, произведенного в проекте.

Рекомендуется использовать уже применяемые и утвержденные организациями OSI и FSF открытые и свободные лицензии.

Если производимый продукт предназначен как для свободного распространения, так и для продажи, рекомендуется двойное лицензирование, когда открытость обеспечивается открытой лицензией, а условия платного распространения с коммерческой поддержкой – коммерческой лицензией.

Ниже приводятся наиболее известные и часто используемые лицензии и рекомендации к их использованию.

GPL - публичная лицензия GNU (GNU General Public License).

Наиболее совершенная с юридической точки зрения открытая лицензия. Лицензия разрешает только открытые внешние изменения, требуя применения для них GPL. Благодаря этому, создается благоприятная среда, в которой в рамках проекта происходит интеграция абсолютно всех внешних изменений. В то же время, GPL не позволяет включение лицензированного продукта в более крупную систему если лицензия этой системы ограничивает права относительно GPL. Таким образом, интеграция системы из компонентов с разной лицензией затрудняется. Отличительной чертой GPL является также то, что автор вправе нарушить условия лицензии, что делает возможным двойное лицензирование кода.

LGPL – облегченная лицензия GNU (GNU Lesser General Public License).

LGPL – производная от GPL лицензия, позволяющая интеграцию продукта с другими, имеющими отличающиеся лицензии. Используется в основном в проектах по производству программных библиотек и компонентов.

Лицензии X, Apache, BSD.

Эти лицензии практически не накладывают ограничений на использование продуктов. Позволяется внесение непубличных внешних изменений и перепродажа с закрытием кода. Лицензия в основном используется в открытых проектах, выполняемых как государственные заказы.

Лицензия Artistic.

Требует публичное распространение изменений, однако в то же время позволяет продажу измененного продукта как части большей программной системы. В настоящее время из-за юридических проблем не используется либо применяется совместно с GPL.

MPL - публичная лицензия Mozilla (Mozilla Public License).

Использованная впервые как лицензия для веб-браузера Mozilla, является открытой лицензией, позволяющей непубличные модификации и перепродажу.

Алгоритм выбора лицензии.

Для принятия решение о выборе лицензии для конкретного открытого проекта следует рассмотреть следующие 3 вопроса:

1. Разрешено ли внесение внешних непубличных изменений? Если решается, что все изменения должны выполняться в среде проекта и быть интегрированы в производимый продукт, тогда следует выбирать GPL или LGPL.
2. Необходимо ли продавать производимый продукт под коммерческой лицензией? Если да, то можно применить технику двойного лицензирования.
3. Допускается ли использование производимых в проекте продуктов в других не открытых продуктах и системах? Если да, наилучшим выбором будет либо LGPL либо лицензии X и Apache в зависимости от решения 1-го вопроса.

Сравнительная характеристика, обобщающая сказанное выше приводится в таблице 5.5.

Таблица 5.5

Сравнительная характеристика открытых лицензий

Лицензия	Возможна интеграция с закрытыми продуктами	Возможны непубличные внешние изменения	Позволяется релицензирование	Имеются привилегии для владельцев авторских прав
GPL	нет	нет	нет	да
LGPL	да	нет	нет	нет
X, Apache, BSD	да	да	нет	нет
MPL	да	да	нет	нет

5.4. Прототипирование и инициализация среды выполнения проекта

5.4.1. Определение проекта. Определением открытого проекта начинается с составленного документа спецификации намерений. Документ должен отвечать на два вопроса – что производится в проекте и какова целевая аудитория производимого продукта. Местом этого документа должны быть новостные страницы, публичные списки рассылки, новостные группы. Инициатор проекта должен быть готов к публичному обсуждению спецификации намерений.

Как только выяснены цели проекта, необходимо разработать предварительный план и обобщенные инженерные решения, позволяющие реализовать цели проекта. На этом этапе производится предварительный выбор инструментальных средств, определяющих инфраструктуру проекта.

5.4.2. Инициализация среды проекта. Целью этого этапа является создание “здоровой” среды проекта, которая обеспечит его развитие и способность реагировать и интегрировать изменения. Среда выполнения состоит из двух важных составляющих – инструментальных средств и основного персонала.

Перечень инструментальных средств определяется на основании практики управления наиболее успешными программными проектами и состоит из:

1. Система управления версиями – обязательная часть инфраструктуры открытого проекта, позволяющая простую интеграцию изменений кода продукта, их отслеживание и обзор.

Наиболее популярные системы управления версиями, рекомендуемые для открытых проектов:

Subversion: <http://subversion.tigris.org>;

CVS: <http://cvshome.org>.

2. Списки рассылки отдельно для разработчиков и отдельно для пользователей – основное средство обеспечения распространения информации между участниками проекта.

Рекомендуемые инструментальные средства:

Mailman: <http://www.list.org/>.

3. IRC – чат-каналы для обеспечения коммуникаций между участниками проекта в режиме реального времени.

Рекомендуемые способы обеспечения чат-каналов:

InspIRCd (IRC сервер): <http://www.inspircd.org/>;

irc.freenode.org: общедоступный IRC сервер где для любого открытого проекта может быть созданы чат-каналы.

4. Веб-сервер и форумы – необходимы для распространения информации о проекте и публикации документов, связанных с проектом. Для целей открытого проекта возможна как настройка собственного веб-сервера, так и использование публичных серверов, таких как:

<http://www.sourceforge.org>;

<http://savannah.nongnu.org/>.

Перечисленные выше сервера предоставляют также доступ к системе управления версиями, спискам рассылки и системе управления запросами на изменения. Для использования этих сервисов проект должен полностью соответствовать условиям, определенным в разделе 5.2 и пройти формальную процедуру проверки.

5. Система управления запросами на изменения – средство, позволяющая пользователям и разработчикам вносить и отслеживать сообщения об ошибках, запросы на новую функциональность и другие задачи.

Рекомендованные системы:

Bugzilla: <http://www.bugzilla.org/>.

Для поддержки инфраструктуры и выполнения задач по управлению проектом потребуется персонал, функции которого должны соответствовать следующим ролям [101]:

1. Поддержка инфраструктуры: установка и администрирование системы управления версиями, запросами на изменения, веб-сервера и списка рассылки.

Приблизительная оценка затрачиваемого времени:

на установку и настройку – 100 часов;

на поддержку и администрирование – 20 часов в неделю.

2. Менеджер проекта, в терминологии открытых проектов мэйнтэйнер (maintainer) или координатор выпусков (release coordinator). Выполняемыми задачами являются обзор всех поступающих изменений и их интеграция при необходимости, контроль качества производимого программного кода, исправление ошибок, принятие проектных решений.

Приблизительная оценка затрачиваемого времени:

на инициализацию проекта – 40-200 часов;

на поддержку и администрирование – 20 часов в неделю.

3. Управление запросами на изменения – просмотр и классификация сообщений об ошибках и запросах на новую функциональность.

Приблизительная оценка затрачиваемого времени:

на установку и настройку – 40-70 часов;

на поддержку и администрирование – 20 часов в неделю.

4. Поддержка документации и веб-сайта.

Приблизительная оценка затрачиваемого времени:

на установку и настройку – 60 часов;

на поддержку и администрирование – 10 часов в неделю.

5. Распространение информации о проекте, выработка стратегии, отношения с пользователями и заказчиками.

Приблизительная оценка затрачиваемого времени:

на получение предварительных знаний – 15 часов;

на выполнение – 20 часов в неделю.

5.4.3. Системный анализ и спецификация требований. После определения проекта и его среды, необходимо пересмотреть план выполнения проекта и разработать системную архитектуру.

В плане уточняются даты выпуска версий продуктов, даты наиболее важных вех и определяются требования и ограничения для каждой из них. На этом этапе выполняются предварительные оценки времени выполнения, в соответствии с которыми и уточняется план. Оценки времени выполняются с использованием аналоговой модели (3.31) для проектов с открытым кодом.

Системная архитектура разрабатывается в открытых проектах преимущественно эволюционным путем, с помощью интенсивного прототипирования. Необходимо удостовериться, что к моменту выполнения 2-й итерации жизненного цикла, существует формальное описание системной архитектуры.

5.5. Применение методики для управления программными проектами с открытым кодом

Разработанная методика управления использована при выполнении свободного проекта с открытым кодом KDE (проекты KDevelop и KDE Eclipse), проектов компаний Pluron и UkrInvent (проекты MediaCloth и ActiveReCpp) и в Национальном университете кораблестроения имени адмирала Макарова (проект разработки программного комплекса автоматизации деятельности университета) (см. приложение Д).

5.5.1. Проект KDevelop

Методика применялась для управления выполнением проектом KDevelop в цикле разработки версии 3.4 продукта.

Общие сведения о проекте KDevelop.

Производитель: группа независимых разработчиков, координируемая в рамках проекта KDE некоммерческой организацией KDE e.V.

Назначение: разработка версии 3.4 интегрированной среды разработчика для платформ Unix.

Критерии выпуска:

- переработанная поддержка системы сборки QMake, включающая новый синтаксический анализатор QMake файлов и новый интерфейс пользователя;
- новый пользовательский интерфейс среды редактирования, поддерживающий только режим «Ideal» и не поддерживающий устаревшие режимы интерфейса «MDI Childframe», «Toplevel» и «Tabbed»;
- улучшенная поддержка Ruby и RubyOnRails проектов;
- поддержка шаблонов, умных указателей и определений типов в автодополнении кода для языка C++

Начало выполнения проекта: 31.05.2006.

Имеющаяся инфраструктура на момент начала выполнения проекта:

- система управления версиями SVN, расположенная в Internet по адресу <https://svn.kde.org/home/kde/branches/kdevelop/3.4>
- списки рассылки:
kdevelop@kdevelop.org
kdevelop-devel@kdevelop.org
- IRC: irc.freenode.org, порт 6667
- Web-сервер и форумы:
<http://www.kdevelop.org>
<http://www.kdevelop.org/phorum5>
- система управления запросами на изменения: <http://bugs.kde.org> с выделенным компонентом «kdevelop».

Так как в проекте производилась новая версия уже существующего продукта, действия методики по обоснованию выбора модели разработки с открытым кодом не выполнялись. Реинициализация инфраструктуры также не производилась ввиду ее соответствия требованиям п.п. 5.6.2 данной методики.

Выполнялась лишь предварительная оценка времени разработки с целью создания плана выпуска продукта с учетом имеющихся ресурсов и наблюдаемых ранее темпов поступления внешних изменений.

Так как проект KDevelop входит в группу проектов разработки интегрированных средств разработки, оценка времени производилась по модели (4.16):

$$F_t = 784 \cdot F_{scope}^{1.97} \cdot F_{qua}^{0.64} \cdot F_{ext}^{-3.1}.$$

Для предварительной оценки критерии масштаба проекта, качества продукта и объема внешних изменений выбирались равными критериям остальных проектов этой группы:

$$F_t = 784 \cdot 0,33^{1.97} \cdot 0,0003^{0.64} \cdot 1,25^{-3.1}.$$

Раскрывая критерий времени, модель оценки времени приобретает вид

$$t = 784 \cdot 0,33^{1,97} \cdot 0,0003^{0,64} \cdot 1,25^{-3,1} \cdot \frac{d_c}{\delta_d}.$$

Экспертная оценка на основании хронологических проектных данных количества разработчиков $d_c = 55$, а оценка объем привлечения сторонних разработчиков $\delta_d = 0.06$. Согласно этому, оценка времени:

$$t = 784 \cdot 0,33^{1,97} \cdot 0,0003^{0,64} \cdot 1,25^{-3,1} \cdot \frac{55}{0.06} = 225,4.$$

Используя эту оценку времени, был составлен план выполнения проекта, предусматривающий его завершение через 225 дней после начала, т.е. 10.01.2007г. Также, эта оценка позволила принять решение про отмену выпуска версии продукта 3.4 в составе группы программ рабочего стола пользователя KDE 3.5.5 11.10.2006 ввиду невозможности окончания проекта к этому сроку согласно оценке.

В действительности проект завершился с опозданием на 15 дней, т.е. 25.01.2007г. Таким образом, относительная погрешность оценки времени выполнения проекта KDevelop 3.4 составила 6%.

Учитывая успешное завершение проекта KDevelop 3.4, был сделан вывод о успешном применении разработанной в работе методики для управления программным проектом KDevelop 3.4.

5.5.2. Проект MediaCloth

Методика применялась для управления выполнения проектом MediaCloth в цикле разработки первой версии продукта.

Общие сведения о проекте MediaCloth.

Производитель: Компания Plurion Inc, США.

Назначение: новая разработка инструментальной библиотеки поддержки формата документов MediaWiki на языке Ruby.

Критерии выпуска:

- лексический анализатор с поддержкой 80% лексем формата MediaWiki;

- синтаксический анализатор с поддержкой 80% синтаксических конструкций формата MediaWiki;
- генератор HTML документов;
- модульные тесты.

Начало выполнения проекта: 12.05.2006.

Так как MediaCloth являлся проектом новой разработки, методика управления применялась полностью и были выполнены следующие действия.

- 1) Обоснование выбора открытой модели методом анализа отличительных категорий.

Обоснование выбора проводилось с использованием алгоритма, приведенного в разделе 5.1. Результаты анализа приведены в таблице 5.6.

Таблица 5.6

Применение метода анализа отличительных категорий для обоснования выбора открытой модели разработки проекта MediaCloth

Группы характеристик	Каскадная	V-образная	Прототипирование	Спиральная	RAD	Инкрементная	Открытая
Требования	0	0	5	5	3	3	5
Команда разработчиков	7	8	5	7	7	6	11
Пользователи	0	0	5	2	3	2	5
Тип проекта и рисков	6	5	4	7	2	7	7
Итого:	13	13	19	21	15	18	29

Как видно из анализа, открытая модель набрала больше всего баллов соответствия для каждой группы характеристик проекта и больше всего баллов в итоговом сравнении. Таким образом, было принято решение использовать открытую модель для разработки проекта MediaCloth.

2) Анализ риска выбора открытой модели, определение объема планирования и открытости.

Анализ рисков производился на основании экспертных оценок согласно алгоритма, приведенного на рис. 5.1. Результаты анализа приведены в таблице 5.7.

Таблица 5.7

Форма оценки рисков проекта MediaCloth

Риск	Рейтинг
$E_{технол}$ – неопределенности используемых технологий	3
$E_{коорд}$ – координация множества участников проекта	1
$E_{сложн}$ – сложность системы	0
$A_{масшт}$ – масштабируемость и критичность	0
$A_{прост}$ – упрощенность проектирования	0
$A_{рот}$ – ротация и текучесть кадров	1
$A_{спос}$ – недостаточные навыки персонала в быстрых методах	0
$P_{изм}$ – частота изменений	0
$P_{скор}$ – необходимость получения быстрых результатов	0
$P_{возн}$ – возникновение новых требований	1
$O_{сист}$ – несистемность производимого продукта	0
$O_{код}$ – опасность открытости кода	0
$O_{утеч}$ – возможность появления порожденных проектов	0
$O_{раз}$ – разовый характер производимого продукта	1
$O_{спос}$ – недостаточные навыки персонала в открытых методах	0

Как показал анализ рисков, из рисков открытости только $O_{раз}$ получил оценку как "средний" ввиду возможного разового производства продукта. Поэтому было принято решения объем открытости не рассчитывать и открыть весь исходный код продукта. Практика выполнения проекта показала, что риск $O_{раз}$ был переоценен, т.к. произведенный продукт используется компаниями Sun и Motiro.

3) Выбор типа лицензирования продукта.

Продукт, создаваемый в проекте, предназначался в первую очередь для использования как часть коммерческой разработки Acunote компании Pluron. Поэтому важными критериями выбора лицензии были:

- возможность интеграции с закрытыми продуктами;
- возможность внесения непубличных внешних изменений.

По этим двум критериям согласно таблице 5.5 была выбрана MPL-подобная лицензия "MIT License".

4) Прототипирование и инициализация среды выполнения проекта.

4.1) Определение проекта

Определение проекта было выполнено в документе, помещенном на веб-странице <http://rubyforge.org/projects/mediacloth/> и содержало в себе описание проекта, описание продукта и перечень задач с предварительным планом их выполнения.

4.2) Инициализация среды проекта

Для инициализации среды проекта был выбран портал RubyForge, который предоставляет инструментарий для быстрого прототипирования и создания инфраструктуры проекта. В качестве альтернативы рассматривался также портал SourceForge, который был отклонен ввиду малого количества размещенных на нем проектов, в которых производится продукт на языке программирования Ruby.

В соответствии с методикой были установлены и настроены следующие средства:

Система управления версиями Subversion по адресу:

<http://mediacloth.rubyforge.org/svn/>

Списки рассылки отдельно для разработчиков и отдельно для пользователей:

<http://rubyforge.org/pipermail/mediacloth-devel/>

<http://rubyforge.org/pipermail/mediacloth-talk/>

IRC – чат-канал для обеспечения коммуникаций между участниками проекта:

чат-канал #mediacloth на сервере irc.freenode.org

Веб-сервер и форумы:

<http://mediacloth.rubyforge.org/>

http://rubyforge.org/forum/forum.php?forum_id=6882

http://rubyforge.org/forum/forum.php?forum_id=6883

Система управления запросами на изменения:

http://rubyforge.org/pm/?group_id=1668

4.3) Системный анализ и спецификация требований

Выполнялась предварительная оценка времени разработки с целью создания плана выпуска продукта с учетом имеющихся ресурсов и наблюдаемых ранее темпов поступления внешних изменений.

Так как проект MediaCloth входит в группу проектов разработки инструментальных библиотек, оценка времени производилась по модели (4.17):

$$F_t = 911 \cdot F_{scope}^{2,1} \cdot F_{qua}^{0,68} \cdot F_{ext}^{-2,8}.$$

Для предварительной оценки критерии масштаба проекта, качества продукта и объема внешних изменений выбирались равными критериям остальных проектов этой группы:

$$F_t = 911 \cdot 0,76^{2,1} \cdot 0,00015^{0,68} \cdot 1,24^{-2,8}.$$

Раскрывая критерий времени, модель оценки времени приобретает вид

$$t = 911 \cdot 0,76^{2,1} \cdot 0,00015^{0,68} \cdot 1,24^{-2,8} \cdot \frac{d_c}{\delta_d}.$$

Количество разработчиков, выделенных на проект $d_c = 2$, экспертная оценка объема привлечения сторонних разработчиков на основании хронологических данных аналогичных проектов $\delta_d = 0.01$. Согласно этому, оценка времени:

$$t = 911 \cdot 0,76^{2,1} \cdot 0,00015^{0,68} \cdot 1,24^{-2,8} \cdot \frac{2}{0,01} = 140,7.$$

Используя эту оценку времени, был составлен план выполнения проекта, предусматривающий его завершение через 123 дня после начала, т.е. 12.09.2006г.

В действительности проект благодаря превышению объема привлечения сторонних разработчиков завершился раньше на 18 дней, т.е. 25.08.2006г. Таким образом, относительная погрешность оценки времени выполнения проекта MediaCloth составила 17%.

Учитывая успешное завершение проекта MediaCloth, был сделан вывод о успешном применении разработанной в работе методики для управления программным проектом MediaCloth.

5.5.3. Проект KDE-Eclipse

Методика применялась для управления выполнением проектом KDE-Eclipse в цикле разработки первой версии продукта.

Общие сведения о проекте KDE-Eclipse.

Производитель: Дымо А.Б. в рамках гранта "Google Summer of Code"; координация осуществлялась некоммерческой организацией KDE e.V.

Назначение: новая разработка модулей интегрированной среды разработки Eclipse для поддержки разработки приложений Qt и KDE.

Критерии выпуска:

- модуль поддержки системы сборки Autotools;
- модуль поддержки системы сборки QMake;
- мастера создания проектов Qt и KDE;
- модуль поддержки средства создания графических интерфейсов пользователя QtDesigner;
- помощники создания проектов Qt и KDE.

Начало выполнения проекта: 15.06.2005.

Конечный срок выполнения проекта: 01.09.2005.

Проект выполнялся в рамках гранта программы Google Summer of Code, вследствие чего он обладал следующими особенностями:

- обязательность применения открытой модели разработки (отсюда отсутствие необходимости обоснования выбора модели);
- открытость всего исходного кода (отсюда отсутствие необходимости выполнять анализ рисков открытости кода);
- конечный срок выполнения проекта - дата окончания программы Google Summer of Code (отсюда необходимость применять оценки времени разработки не для составления плана проекта, а для определения его выполнимости в рамках определенных сроков).

Поэтому, при применении методики управления были выполнены следующие действия.

1) Выбор типа лицензирования продукта.

Продукт, создаваемый в проекте, предназначался как вспомогательное средство для разработчиков прикладных программ. Поэтому одним из критериев выбора лицензии было обеспечение невозможности релицензирования с закрытием кода. Это требование также было указано в правилах программы Google Summer of Code.

Согласно таблице 5.5, для проекта подходила лицензия GPL, однако, вследствие ее несовместимости с лицензией среды Eclipse, модули для которой предстояло разработать, была выбрана Eclipse Public License как наиболее близко отвечающая поставленным требованиям.

2) Прототипирование и инициализация среды выполнения проекта.

2.1) Определение проекта

Определение проекта было выполнено в документе, помещенном на веб-странице <http://developer.kde.org/summerofcode/eclipse.html> и содержало в себе описание проекта, описание продукта и перечень задач с планом их выполнения.

Определение проекта полностью соответствовало требованиям программы "Google Summer of Code".

2.2) Инициализация среды проекта

В соответствии с методикой были установлены и настроены следующие средства:

Система управления версиями Subversion по адресу:

<https://svn.kde.org/home/kde/trunk/playground/devtools/eclipse/>

Список рассылки:

http://sourceforge.net/mailarchive/forum.php?forum_name=kde-eclipse-talk

IRC – чат-канал для обеспечения коммуникаций между участниками проекта:

чат-канал #kde-eclipse на сервере irc.freenode.org

Веб-сервер и форумы:

<http://kde-eclipse.pwsp.net/>

Система управления запросами на изменения:

<http://bugs.kde.org>, компонент "kde-eclipse".

2.3) Системный анализ и спецификация требований

Ввиду ограничения срока разработки, проводилась оценка времени разработки с целью определения выполнимости проекта до его начала.

Так как проект KDE-Eclipse входит в группу проектов разработки интегрированных средств разработки, оценка времени производилась по модели (4.16):

$$F_t = 784 \cdot F_{scope}^{1.97} \cdot F_{qua}^{0.64} \cdot F_{ext}^{-3.1}$$

Для предварительной оценки критерии масштаба проекта, качества продукта и объема внешних изменений выбирались равными критериям остальных проектов этой группы:

$$F_t = 784 \cdot 0,33^{1.97} \cdot 0,0003^{0.64} \cdot 1,25^{-3.1}$$

Раскрывая критерий времени, модель оценки времени приобретает вид

$$t = 784 \cdot 0,33^{1.97} \cdot 0,0003^{0.64} \cdot 1,25^{-3.1} \cdot \frac{d_c}{\delta_d}$$

Количество разработчиков в проекте было $d_c = 1$, а экспертная оценка объема привлечения сторонних разработчиков $\delta_d = 0.01$ (1 новый разработчик за 77 дней выполнения проекта). Согласно этому, оценка времени:

$$t = 784 \cdot 0,33^{1,97} \cdot 0,0003^{0,64} \cdot 1,25^{-3,1} \cdot \frac{1}{0,01} = 24,5.$$

Для выполнения проекта было отведено 77 дней. Оценка времени говорила о возможности завершения за более короткий срок, таким образом было принято решение о выполнимости проекта.

В действительности проект начался 11.07.2005г. Причиной сдвига даты начала разработки стало промедление в оформлении документов на получение гранта.

Проект был завершён 27.08.2005г. через 41 день после начала разработки. Погрешность оценки времени, таким образом, составила 40%. Такая значительная цифра отклонения объясняется малым масштабом проекта. В таких условиях погрешность оценок времени возрастает как для существующих моделей СОСОМО так и для разработанных аналоговых моделей.

5.5.4. Проект ActiveReCpp

Методика применялась для управления выполнением проектом ActiveReCpp в цикле разработки первой версии продукта.

Общие сведения о проекте ActiveReCpp.

Производитель: Компания UkrInvent (УкрИнвент), Украина.

Назначение: новая разработка инструментальной библиотеки обеспечения коммуникаций между программами на языке C++ и веб-сервисами построенными по архитектуре REST.

Критерии выпуска:

- поддержка аналога ActiveResource для языка C++;
- модульные тесты;
- пример приложения использующего библиотеку.

Начало выполнения проекта: 17.10.2006.

Так как ActiveReCpP являлся проектом новой разработки, методика управления применялась полностью и были выполнены следующие действия.

1) Обоснование выбора открытой модели методом анализа отличительных категорий.

Обоснование выбора проводилось с использованием алгоритма, приведенного в разделе 5.1. Результаты анализа приведены в таблице 5.8.

Как видно из анализа, открытая модель набрала больше всего баллов соответствия для каждой группы характеристик проекта и больше всего баллов в итоговом сравнении. Таким образом, было принято решение использовать открытую модель для разработки проекта ActiveReCpP.

Таблица 5.8

Применение метода анализа отличительных категорий для обоснования выбора открытой модели разработки проекта ActiveReCpP

Группы характеристик	Каскадная	V-образная	Прототипирование	Спиральная	RAD	Инкрементная	Открытая
Требования	0	0	5	5	3	3	5
Команда разработчиков	6	7	4	8	8	7	12
Пользователи	0	0	5	2	3	2	5
Тип проекта и рисков	5	4	5	6	1	8	8
Итого:	11	11	19	21	15	20	30

2) Анализ риска выбора открытой модели, определение объема планирования и открытости.

Анализ рисков производился на основании экспертных оценок согласно алгоритма, приведенного на рис. 5.1. Результаты анализа приведены в таблице 5.9.

Таблица 5.9

Форма оценки рисков проекта ActiveReCpp

Риск	Рейтинг
$E_{технол}$ – неопределенности используемых технологий	3
$E_{коорд}$ – координация множества участников проекта	1
$E_{сложн}$ – сложность системы	1
$A_{масшт}$ – масштабируемость и критичность	0
$A_{прост}$ – упрощенность проектирования	1
$A_{рот}$ – ротация и текучесть кадров	1
$A_{спос}$ – недостаточные навыки персонала в быстрых методах	0
$P_{изм}$ – частота изменений	0
$P_{скор}$ – необходимость получения быстрых результатов	0
$P_{возн}$ – возникновение новых требований	1
$O_{сист}$ – несистемность производимого продукта	0
$O_{код}$ – опасность открытости кода	0
$O_{утеч}$ – возможность появления порожденных проектов	0
$O_{раз}$ – разовый характер производимого продукта	0
$O_{спос}$ – недостаточные навыки персонала в открытых методах	0

Как показал анализ рисков, из рисков открытости ни один не получил оценку выше минимального, поэтому было принято решения объем открытости не рассчитывать и открыть весь исходный код продукта.

3) Выбор типа лицензирования продукта.

Продукт, создаваемый в проекте, предназначался для использования в разработке коммерческих проектов компании UkrInvent (одним из таких проектов являлся модуль связи с системой управления проектами Acunote, предоставляющей веб-сервисы). Поэтому важными критериями выбора лицензии были:

- возможность интеграции с закрытыми продуктами;
- возможность внесения непубличных внешних изменений.

По этим двум критериям согласно таблице 5.5 была выбрана MPL-подобная лицензия "MIT License".

4) Прототипирование и инициализация среды выполнения проекта.

4.1) Определение проекта

Определение проекта было выполнено в документе, помещенном на веб-странице <http://code.google.com/p/activerecpp/> и содержало в себе описание проекта, описание продукта и перечень задач с предварительным планом их выполнения.

4.2) Инициализация среды проекта

Для инициализации среды проекта был выбран портал Google Code, который предоставляет инструментарий для быстрого прототипирования и создания инфраструктуры проекта. В качестве альтернативы рассматривался также портал SourceForge, который был отклонен ввиду неудовлетворительной скорости работы веб-интерфейсов управления сайтом и списками рассылки.

В соответствии с методикой были установлены и настроены следующие средства:

Система управления версиями Subversion по адресу:

<https://activerecpp.googlecode.com/svn/>

Wiki для разработчиков и пользователей:

<http://code.google.com/p/activerecpp/w/list>

IRC – чат-канал для обеспечения коммуникаций между участниками проекта:

чат-канал #activerecpp на сервере irc.freenode.org

Веб-сервер и форумы:

<http://code.google.com/p/activerecpp/>

Система управления запросами на изменения:

<http://code.google.com/p/activerecpp/issues/list>

4.3) Системный анализ и спецификация требований

Выполнялась предварительная оценка времени разработки с целью создания плана выпуска продукта с учетом имеющихся ресурсов и наблюдаемых ранее темпов поступления внешних изменений.

Так как проект ActiveReCpp входит в группу проектов разработки инструментальных библиотек, оценка времени производилась по модели (4.17):

$$F_t = 911 \cdot F_{scope}^{2,1} \cdot F_{qua}^{0,68} \cdot F_{ext}^{-2,8}.$$

Для предварительной оценки критерии масштаба проекта, качества продукта и объема внешних изменений выбирались равными критериям остальных проектов этой группы:

$$F_t = 911 \cdot 0,76^{2,1} \cdot 0,00015^{0,68} \cdot 1,24^{-2,8}.$$

Раскрывая критерий времени, модель оценки времени приобретает вид

$$t = 911 \cdot 0,76^{2,1} \cdot 0,00015^{0,68} \cdot 1,24^{-2,8} \cdot \frac{d_c}{\delta_d}.$$

Количество разработчиков, выделенных на проект $d_c = 3$, экспертная оценка объема привлечения сторонних разработчиков на основании хронологических данных аналогичных проектов $\delta_d = 0.01$. Согласно этому, оценка времени:

$$t = 911 \cdot 0,76^{2,1} \cdot 0,00015^{0,68} \cdot 1,24^{-2,8} \cdot \frac{3}{0,01} = 211,1.$$

Используя эту оценку времени, был составлен план выполнения проекта, предусматривающий его завершение через 211 дней после начала, т.е. 16.05.2007г.

В действительности проект завершился раньше на 2 дня, т.е. 14.05.2007г. Таким образом, относительная погрешность оценки времени выполнения проекта ActiveReCpp составила менее 1%!

Учитывая успешное завершение проекта ActiveReCpp, был сделан вывод о успешном применении разработанной в работе методики для управления программным проектом ActiveReCpp.

5.5.5. Подпроект автоматизации бухгалтерского учета проекта разработки программного комплекса автоматизации деятельности Национального университета кораблестроения имени адмирала Макарова

Методика применялась для управления выполнением подпроектом автоматизации бухгалтерского учета в цикле разработки третьей версии продукта "Первичные документы".

Общие сведения о проекте.

Производитель: Национальный университет кораблестроения, Украина.

Назначение: разработка новой версии продукта "Первичные документы".

Критерии выпуска:

- управление юридическими и финансовыми обязательствами университета;
- управление платежными поручениями;
- коммуникации с системой электронных платежей Государственной налоговой администрации Украины.

Начало выполнения проекта: 10.03.2004.

Завершение выполнения проекта: 23.10.2006.

В проекте разрабатывался программный код для внутреннего использования университетом. Было высказано предположение о высоком степени риска открытости исходного кода проекта, поэтому предложенным в методике методом анализа риска обосновывался выбор открытой модели разработки и определялся объем открытости кода.

Анализ рисков производился на основании экспертных оценок согласно алгоритма, приведенного на рис. 5.1. Результаты анализа приведены в таблице 5.10.

Таблица 5.10

Форма оценки рисков подпроекта автоматизации бухгалтерского учета

Риск	Рейтинг
$E_{технол}$ – неопределенности используемых технологий	2
$E_{коорд}$ – координация множества участников проекта	3
$E_{сложн}$ – сложность системы	2
$A_{масшт}$ – масштабируемость и критичность	3
$A_{прост}$ – упрощенность проектирования	2
$A_{рот}$ – ротация и текучесть кадров	1
$A_{спос}$ – недостаточные навыки персонала в быстрых методах	3
$P_{изм}$ – частота изменений	0
$P_{скор}$ – необходимость получения быстрых результатов	0
$P_{возн}$ – возникновение новых требований	1
$O_{сист}$ – несистемность производимого продукта	0
$O_{код}$ – опасность открытости кода	3
$O_{утеч}$ – возможность появления порожденных проектов	0
$O_{раз}$ – разовый характер производимого продукта	2
$O_{спос}$ – недостаточные навыки персонала в открытых методах	2

Как показал анализ рисков, риски открытости являлись хотя и управляемыми, но серьезными. В то же время, риски других групп также являлись серьезными. Таким образом, ни одна из групп рисков не являлась доминирующей, поэтому в соответствии с предложенной методикой управления, подпроект автоматизации бухгалтерского учета был разбит на два:

- подпроект разработки конструктора программ автоматизации учета (названный SAGA);
- подпроект разработки собственно средств автоматизации (с названием "Первичные документа").

Для подпроекта SAGA риски открытости не являлись доминирующими, т.к. $O_{сист}$, $O_{код}$, $O_{утеч}$, $O_{раз}$ были оценены экспертами как минимальные. Только оценка $O_{спос}$ оставалась неизменной (серьезный, но управляемый риск). Поэтому, в соответствии с методикой управления, было принято решение об открытии исходного кода подпроекта SAGA.

Выбор типа лицензирования продукта. Продукт, создаваемый в проекте, создавался в рамках проекта автоматизации деятельности университета, поэтому важными критериями выбора лицензии были:

- гарантия авторских прав на исходный код для обеспечения возможности закрытия части кода в будущем;
- обеспечение невозможности внесения непубличных внешних изменений.

По этим двум критериям согласно таблице 5.5 была выбрана лицензия GPL.

Учитывая успешное завершение подпроектов SAGA и "Первичные документы", был сделан вывод о успешном применении разработанной в работе методики для управления программным проектом автоматизации бухгалтерского учета Национального университета кораблестроения имени адмирала Макарова.

5.6 Основные выводы по разделу 5

1. Разработана методика управления программными проектами с открытым кодом, которая включает алгоритм обоснования выбора открытой модели с

использованием методов анализа различительных категорий и анализа рисков; рекомендации по типу лицензирования продукта и инициализации инфраструктуры проекта.

2. Обосновано, что использование методики повысило эффективность управления проектами благодаря правильному выбору модели разработки программного обеспечения с открытым кодом и схемы лицензирования для проектов MediaCloth и ActiveReCpp, успешному обеспечению инициализации и завершения проектов MediaCloth, ActiveReCpp, KDevelop и KDE Eclipse в рамках заданных ограничений качества, стоимости и времени.
3. Доказано, что применение разработанных моделей оценки времени при выполнении программных проектов с открытым кодом KDevelop и MediaCloth снизило погрешность оценки времени соответственно до 6% и 17%, что позволило корректно определить размер затрат для успешного завершения проектов с заданным качеством и подтвердить повышение эффективности управления проектами.
4. Подтверждено практическое значение полученных результатов на примерах их использования при выполнении свободного проекта с открытым кодом KDE, проектов компаний Pluron и UkrInvent и Национального университета кораблестроения имени адмирала Макарова.

ВЫВОДЫ

В диссертационной работе решено важное научно-техническое задание – повышена эффективность управления программными проектами с открытым кодом и обеспечено их завершение в рамках заданных ограничений времени, стоимости и качества на основании разработанной модели жизненного цикла и системы управления проектами разработки программного обеспечения с открытым кодом и разработанной и эмпирически подтвержденной модели оценки времени и стоимости выполнения таких проектов.

При этом получены следующие результаты:

1. Разработана модель жизненного цикла программных проектов с открытым кодом, которая определяет перечень и последовательность действий для выполнения в проекте, определяет место процессов управления в жизненном цикле и отвечает требованиям стандартов ISO 12207 і ДСТУ 3918-1999.
2. Разработана математическая модель системы управления программными проектами с открытым кодом и создано программное обеспечение, которое для данных ограничений параметров состояния системы и коэффициентов их взаимного влияния рассчитывает вероятность устойчивости системы.
3. Обоснованы управляемость проекта и возможность его успешного завершения наличием поля устойчивости системы управления, которое позволяет принять решение про выполнимость проекта еще до начала его выполнения.

4. Определено, что наличие в модели системы управления ступенчатых функций обеспечивает самоорганизацию в проектах с открытым кодом, что повышает вероятность их успешного завершения при влиянии внешних дестабилизирующих изменений.
5. Разработаны аналоговые модели оценки времени и стоимости разработки программного обеспечения, которые учитывают не только размер продукта, а его качество, темп разработки и доработки, показатели эффективности проектной команды и другие параметры, определяющие время и стоимость.
6. Определены численные значения параметров аналоговых моделей проектов разработки и доработки систем обработки транзакций, поддержки принятия решений, автоматизации деятельности офисов и управления процессами на производстве с средней погрешностью оценок времени ниже на 55% погрешности существующей модели COCOMO.
7. Доказано, что применение разработанных моделей оценки времени при выполнении программных проектов с открытым кодом KDevelop и MediaCloth снизило погрешность оценки времени соответственно до 6% и 17%, что позволило корректно определить размер затрат для успешного завершения проектов с заданным качеством и подтвердить повышение эффективности управления проектами.
8. Разработана методика управления программными проектами с открытым кодом, которая включает алгоритм обоснования выбора открытой модели с использованием методов анализа различительных категорий и анализа

рисков; рекомендации по типу лицензирования продукта и инициализации инфраструктуры проекта.

9. Обосновано, что использование методики повысило эффективность управления проектами благодаря правильному выбору модели разработки программного обеспечения с открытым кодом для проектов MediaCloth и ActiveReCpp, успешному обеспечению инициализации и завершения проектов MediaCloth, ActiveReCpp, KDevelop и KDE Eclipse в рамках заданных ограничений качества, стоимости и времени.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. CHAOS Report [Электронный ресурс] // The Standish Group International, Inc. – 2004. – Режим доступа к отчету:
<http://www.projectsart.co.uk/docs/chaos-report.pdf>
2. Jian Z. Why IT Projects Fail // Computerworld. – 2005. – Vol. 39, № 6. – P. 31-32.
3. Glass R. L. System Development Glass Column // System Development. – 1988. – Vol. 1, № 1. – P. 4-5.
4. Брукс Ф. Мифический человеко-месяц или как создаются программные системы. – СПб.: Символ-Плюс, 2006. – 304с.
5. CHAOS Report [Электронный ресурс] // The Standish Group International, Inc. – 1994. – Режим доступа к отчету:
http://www.standishgroup.com/sample_research/chaos_1994_1.php
6. Mystery Solved! Linux is Cheaper – PERIOD. / The Standish Group International, Inc. // VirtualBEACON. – 2004. № 347. – P. 1-3.
7. Ройс У. Управление проектами по созданию программного обеспечения. Унифицированный подход. – М.: Издательство "Лори", 2002. – 424 с.
8. Boehm B.W. Software Engineering Economics. – Upper Saddle River, New Jersey: Prentice Hall PTR, 1981. – 768p.
9. Boehm B.W. A Spiral Model of Software Development and Enhancement // IEEE Computer. – 1988. – № 21(5). – P. 61-72.
10. Boehm B.W. Software Cost Estimation with COCOMO II. – Upper Saddle River, New Jersey: Prentice Hall PTR, 2000. – 502p.
11. Cockburn A. Agile Software Development. – Boston: Addison-Wesley, 2005. – 278p.
12. Highsmith J. Agile Software Development Ecosystems. – Boston: Addison-Wesley, 2006. – 448p.
13. Beck K. Extreme Programming Explained. – Reading, MA: Addison-Wesley, 2004. – 224p.

14. Weinberg G.M., Schulman E.L. Goals and Performance in Computer Programming // Human Factors. – 1974. – № 16(1). – P. 70-77.
15. Putnam L.H., Fitzsimmons A. Estimating Software Costs // Datamation. – 1979. Vol. 3. – P. 171-178.
16. Garmus D., Herron D. Function Point Analysis: Measurement Practices for Successful Software Projects. – Boston, MA: Addison-Wesley, 2000. – 400p.
17. Function Points as Asset Reporting to Management [Электронный ресурс] / International Function Point Users Group // IFPUG Standards. – 1990. – Режим доступа: <http://www.ifpug.org>.
18. Function Point Counting Practices Manual [Электронный ресурс] / International Function Point Users Group // IFPUG Standards. – 1994. – Release 4.0. – Режим доступа: <http://www.ifpug.org>.
19. Guidelines to Software measurement [Электронный ресурс] / International Function Point Users Group // IFPUG Standards. – 1994. – Release 1.0. – Режим доступа: <http://www.ifpug.org>.
20. Бобровский С.И. Программная инженерия. – СПб.: Питер, 2003. – 222с.
21. Управление проектами. Основы профессиональных знаний и система оценки компетенции проектных менеджеров (National Competence Baseline, NCB UA Version 3.0) : / Бушуев С.Д., Бушуева Н.С. К.:ІРІДІУМ, 2006. – 208 с.
22. Денисов Ю.Д. Японские прогнозы мирового инновационного развития // Наука Москвы и Регионов. – 2004. – №3. – С. 49-55.
23. Бершадская Е.Г. Компьютерная поддержка обучения моделированию и анализу информационно-вычислительных систем // Тезисы докладов XXIII международной конференции "Новые информационные технологии в науке, образовании и бизнесе". – Украина, Крым, 1996. – с. 211.
24. Farquhar J.A. A Preliminary Inquiry Into the Software Estimation Process / The Rand Corp. – RM-621PR. – 1970. – 186p.
25. Демарко Т., Листер Т. Человеческий фактор: успешные проекты и команды. – СПб.: Символ-Плюс, 2005. – 256с.

26. Константин Л. Человеческий фактор в программировании. – СПб.: Символ-плюс, 2004. – 384с.
27. Керівництво з питань проектного менеджменту / Під ред. С.Д. Бушуєва. – К.: Видавничий дім "Деловая Украина", 2000. – 198 с.
28. Market Share for Browsers, Operating Systems and Search Engines [Электронный ресурс] // Market Share by NET Applications. – 2007. – Режим доступа: <http://marketshare.hitslink.com/report.aspx?qprid=0>.
29. Browser Statistics [Электронный ресурс] // W3C Corporation. – 2007. – Режим доступа: http://www.w3schools.com/browsers/browsers_stats.asp.
30. Coar K. The Open Source Definition (annotated) [Электронный ресурс] // Open Source Initiative. – 2006. – Version 1.9. – Режим доступа: <http://www.opensource.org/docs/definition.php>
31. Киреев А. Определение концепции Открытого Исходного Кода (Open Source) [Электронный ресурс] // Open Source Initiative. – 2006. – Version 1.9. – Режим доступа: <http://www.opensource.org/docs/osd-russian.php>.
32. Динамическое лидерство в управлении проектами :Бушуев С.Д, Морозов В.В. Монография / Украинская ассоциация управления проектами. - 2-е изд. - К., 2000. - 312с.
33. Little G., Healey M. Worldwide Open Source Services 2007–2011 Forecast [Электронный ресурс] // IDC. – 2007. – № 208255. – Режим доступа: <http://www.idc.com/getdoc.jsp?containerId=208255>.
34. NetCraft Internet Research Analysis [Электронный ресурс]. – 2006. – Режим доступа: <http://www.netcraft.com>.
35. Ghosh R.A. Study on the economic impact of FLOSS on innovation and competitiveness of the EU ICT sector: Final Report / UNU-MERIT. – Contract ENTR/04/112. – The Netherlands, 2006. – 287p.
36. Conklin M. Collaborative collection and analysis of free/libre/open source project data [Электронный ресурс] // FLOSSmole. – 2007. – № 4. – Режим доступа: <http://ossmole.sourceforge.net/index.html>.

37. Садовніков О.О., Рач О.М. Неопределенность и риск в выработке стратегии аудита при проверках финансовой отчетности предприятий // Збірник наукових праць “Управління проектами та розвиток виробництва. – Луганск: Вид-во СНУ ім. В. Даля, 2006. – №1(17). С. 106-109.
38. Schindler E. OSS/Linux Development Survey [Электронный ресурс] // Evans Data Corporation Strategic Reports. – 2007. – № 1. – Режим доступа: <http://evansdata.com/reports/viewRelease.php?reportID=7>.
39. BerliOS: The Open Source Mediator. – Режим доступа: <http://www.berlios.de>. – Заголовок с экрана.
40. Список стран по ВВП [Электронный ресурс] // Википедия – свободная энциклопедия. – 2007. – Режим доступа: http://ru.wikipedia.org/wiki/Список_стран,_сортировка_по_ВВП.
41. Интернет-маркетинг в Украине 2006 (официальный пресс-релиз для СМІ II-й Международной конференции "Интернет-маркетинг в Украине 2006") [Электронный ресурс] / Шевченко Е. // ІМА UaMaster. – 2006. – Режим доступа: <http://internet-marketing.org.ua/ru/2006>.
42. Золотов Е. Теоретическая часть [Электронный ресурс] // Копьютерра Онлайн. – 2003. – № 12. – Режим доступа: <http://www.computerra.ru/focus/coment/31201/>.
43. Тесля Ю.Н. Несилова взаємодія. – К.: Кондор, 2005. – 196 с.
44. Malcolm J. Problems in Open Source Licensing [Электронный ресурс] // iLaw. – Режим доступа: <http://www.ilaw.com.au/public/licencearticle.html>.
45. Chase B. Sun to open-source Java [Электронный ресурс] // ZDNet Australia. – 2004. – №1. – Режим доступа: http://www.zdnet.com.au/news/software/soa/Sun_to_open_source_Java/0,130061733,139149502,00.htm.
46. Kirk J. Sun says open-source Java possible in 'months' [Электронный ресурс] // IDG News Service. – 2006. – № 6. – Режим доступа: http://www.infoworld.com/article/06/06/27/79685_HNsunopensourcejava_1.html.

47. LaMonica M. IBM urges Sun to make Java open source [Электронный ресурс] // CNET News. – 2004. – № 2. – Режим доступа: http://news.com.com/2100-1007_3-5165427.html.
48. Roberts A. The Java open source debate [Электронный ресурс] // OSNews. – 2005. – Режим доступа: http://www.osnews.com/story.php?news_id=10806.
49. Dymo A. Open Source Software Engineering. // II Open Source World Conference: Proceedings Book. – Malaga, Spain, 2006. – P. 59-63.
50. Valverde S., Theraulaz G. Emergent behavior in agent networks: Self-Organization in Wasp and Open Source Communities // Complexity. – 2002. – № 8 (1). – P. 20-26.
51. Valverde S., Solé R.V. Self-Organization and Hierarchy in Open Source Social Networks // Santa Fe Institute Scientific Works. – 2006. – № 06-12-053. – P. 3-9.
52. Crowston K., Wei K., Li Q., Eseryel U. Y. Coordination of Free/Libre Open Source Software development // Proceedings of the International Conference on Information Systems (ICIS 2005). Las Vegas, NV, USA, December 2005.
53. Goldman R., Gabriel R.P. Innovation Happens Elsewhere: Open Source as Business Strategy. – San Francisco: Morgan Kaufmann, 2005. – 424 p.
54. Raymond E.S. The Cathedral and the Bazaar. – Sebastopol, CA: O'Reilly & Associates, 1999. – 281 p.
55. Словник-довідник з питань управління проектами / Бушуев С.Д. Українська асоціація управління проектами. - К.: Издательский дом "Ділова Україна", 2001. - 640с.
56. Кэрролл Дж. Очень ощутимый недостаток open-source [Электронный ресурс] // ZDNet. – 2002. – № 9. – Режим доступа: <http://zdnet.ru/?ID=203472>.
57. Лушня Ю. "Мода" на Open Source [Электронный ресурс] // LinuxRSP. – 2005. – Режим доступа: <http://citkit.ru/articles/45/>.
58. ISO/IEC 12207:1995. Software life cycle processes. – Geneva, Switzerland: International Organization for Standardization. – 57p.

59. ДСТУ 3918-1999 (ISO/IEC 12207:1995) Державний стандарт України. Інформаційні технології. Процеси життєвого циклу програмного забезпечення. – К.: Держстандарт України, 2000. – 49 с.
60. Кошкин К.В., Яни А.Ю., Романчук Н.П. Информационная поддержка процессов жизненного цикла постройки судна // Збірник наукових праць НУК. – Миколаїв: НУК, 2006. – № 5 (410). – С.62-68.
61. Abdel-Hamid Tarek K., Madnick S.E. Lessons Learned from Modeling the Dynamics of Software Development // Communications of the ACM. – 1989. – №4. – P. 1426-1455.
62. Эшби У.Р. Конструкция мозга. Происхождение адаптивного поведения. – М.: Мир, 1964. – 411с.
63. KDE 3.0 Release Plan [Электронный ресурс]. – Режим доступа: <http://developer.kde.org/development-versions/kde-3.0-release-plan.html>.
64. KDE Release Schedules [Электронный ресурс]. – Режим доступа: <http://developer.kde.org/development-versions/release.html>.
65. KDE [Электронный ресурс] // Wikipedia, the free encyclopedia. – Режим доступа: <http://en.wikipedia.org/wiki/KDE>.
66. Royce W.W. Managing the Development of Large Software Systems: Concepts and Techniques // Proceedings WESCON. LosAlamitos, CA, 1970. – P. 12-18.
67. Шафер Д.Ф., Фартрелл Р.Т., Шафер Л.И. Управление программными проектами: достижение оптимального качества при минимуме затрат. – М.: Издательский дом “Вильямс”, 2003. – 1136 с.
68. Дымо А.Б., Морозова А.С. Открытая модель жизненного цикла программных проектов. // Збірник наукових праць НУК. – Миколаїв: НУК, 2005. – №5 (404). – С.134-141.
69. Moore J. ISO 12207 and Related Software Life-Cycle Standards // Communications of the ACM. – 2005. – № 4. – P. 134-139.
70. Gray L. Using MIL-STD-498 and ISO/IEC 12207 for OOD and RAD // Proceedings of the Eighth Annual Software Technology Conference. Salt Lake City, Utah, April 1996. – P. 1187-1195.

71. Gray L. ISO/IEC 12207 Software Lifecycle Processes // Communications of the ACM. – 2005. – № 8. – P. 65-69.
72. Singh R. International Standard ISO/IEC 12207 Software Lifecycle Processes // Tutorial slides. – 1995.
73. KDE Project Announced. New Project: Kool Desktop Environment (KDE) [Электронный ресурс] // KDE Web Site. – 1996. – Режим доступа: <http://www.kde.org/announcements/announcement.php>.
74. Писаревский М. Решат ли новые стандарты проблемы ЖЦ ИСУ? // ComputerWorld Украина. – 2004. – № 6. – с. 23.
75. Manifesto for Agile Software Development [Электронный ресурс]. – Режим доступа: <http://agilemanifesto.org/>.
76. Вендров А.М. Современные технологии создания программного обеспечения // Информационный бюллетень JetInfo. – 2004. – №4(131). – с. 64.
77. Boehm V.W., Turner R. Balancing Agility and Discipline. A Guide for the Perplexed. – Boston: Addison-Wesley, 2005. – 266p.
78. Чередниченко А.Н. Моделі і методи аналізу ризиків проекту на етапах науково-дослідних та дослідно-конструкторських робіт: Автореферат дисертації на здобуття наукового ступеня кандидата технічних наук / ХАІ. – Харків: Видавничий центр "ХАІ", 2004. – 19с.
79. SourceForge.net: Create, Participate, Evaluate. – Режим доступа: <http://www.sourceforge.net>. – Заголовок с экрана.
80. Дорф Р., Бишоп Р. Современные системы управления. – М.: Лаборатория Базовых Знаний, 2002. – 832 с.
81. Hurwitz A. On the Conditions under which an Equation has only Roots with Negative Real Parts // Selected Papers on Mathematical Trends in Control Theory. – New York, 1964. – P. 34-38.
82. Routh E. J. A Treatise on the Stability of a Given State of Motion. – London: Taylor & Francis, 1975. – 297 p.

83. Paulo Ney de Souza, Fateman R.J., Moses J. The Maxima Book. – Berkeley, CA: New Riders, 2004. – 173 p.
84. Fairley R.E. Recent Advances in Software Estimation Techniques // Proceedings of the 14th international IEEE software engineering conference. – NY: IEEE Computer Society Press. – P. 382-391.
85. Putnam L.H. A General Empirical Solution to the Macro Software Sizing and Estimating Problem // IEEE Transactions on Software Engineering. – 1978. – № 4. – P. 345-361.
86. Helmer O. Social Technology. – New York: Basic Books, 1966. – 324 p.
87. Boehm B.W. The Impact of New Technologies on Software Configuration Management: TRW Report to USAF-ESD. – Contract F19628-74-C-0154. – 1974.
88. Гухман А.А. Применение теории подобия к исследованию процессов тепло-массообмена. – М.: "Высшая школа", 1974. – 328с.
89. Гухман А.А. Физические основы теплопередачи. Том первый. Теория подобия и ее приложения. – М.: Государственное энергетическое издательство, 1934. – 316с.
90. Дымо А.Б. Применение теории подобия для оценки стоимости разработки программных продуктов. // Збірник наукових праць НУК. – Миколаїв: НУК, 2006. – №3 (408). – С.162-167.
91. Словарь по кибернетике / Под ред. Глушкова В.М. – К.: Главная редакция УСЭ, 1979. – 624с.
92. Месарович М., Тахакара Я. Общая теория систем: математические основы. – М.: Мир, 1978. – 312с.
93. Мигай В.К. Моделирование теплообменного энергетического оборудования. – Л.: "Энергоатомиздат", 1987. – 286с.
94. Дульнев Г.Н. Тепло- и массообмен в радиоэлектронной аппаратуре. – М.: "Высшая школа", 1984. – 247с.
95. Кошкин В.К., Калинин Э.К. Теплообменные аппараты и теплоносители. – М. "Машиностроение", 1971. – 200с.

96. Блинцов В.С., Любимцев В.О. Современные задачи управления проектами освоения морских газовых месторождений Украины // Збірник наукових праць НУК. – Миколаїв: НУК, 2005. – № 3 (402). – С.136-143.
97. DeMarco T., Boehm V.W. Controlling Software Projects: Management, Measurement, and Estimates. Englewood Cliffs, NJ: Prentice Hall PTR/Sun Microsystems Press, 1998. – 212p.
98. Albrecht A.J. Measuring Application Developers Productivity // Proceedings of the IBM Application Dev.Symposium. October 1979, CA. – P. 83-92.
99. Albrecht A.J. Gaffney J.E. Software Function, Source Lines of Code, and Developer Effort Prediction // IEEE Transactions on Software Engineering. – 1993. – SE-9(6). – P. 639-648.
100. Кошкин К.В., Павлов А.А. Алгоритмическое обеспечение управления проектами виртуальных производств в судостроении. – Херсон: Олди-Плюс, 2001. – 178с.
101. Capers J. Programming Productivity. – NY: McGraw-Hill, 1986. – 239p.
102. Capers J. Applied Software Measurement: Assuring Productivity and Quality, 2nd ed. – NY: McGraw-Hill, 1997. – 301p.
103. Fuqua A.M. Using Function Points in Extreme Programming // AgileAlliance. – 2005. – № 2. – P. 10-13.
104. Jeffries R. A Metric Leading to Agility // XP Magazine. – 2004. – № 6. – p.64.
105. Capers J. Programming Languages Table. – NY: Software Productivity Research, 1996. – 155p.
106. Гмурман В.Е. Введение в теорию вероятностей и математическую статистику. – М.: «Высшая Школа», 1963. – 238с.
107. Press W.H., Flannery W.T., Teukolsky S.A. Numerical Recipes in C. – New York: Cambridge University Press, 1992. – 56p.
108. Дымо А.Б. Модели оценок времени выполнения программных проектов на основе базы данных метрик проектов ISBSG // Збірник наукових праць НУК. – Миколаїв: НУК, 2006. – № 5 (410). – С.53-56.

109. Polak E. Optimization Algorithms and Consistent Approximations. – New York: Springer-Verlag, 1997. – 198p.
110. Winston W. Operations Research: Applications and Algorithms. – Belmont: Wadsworth, 1994. – 214p.
111. Coar K. The Open Source Definition (plain) [Электронный ресурс] // Open Source Initiative. – 2006. – Режим доступа: <http://opensource.org/docs/osd>.
112. DiBona C., Ockman S., Stone M. Open Sources: Voices from the Open Source Revolution. – Sebastopol, CA: O'Reilly, 1999. – 280 p.
113. Йордон Э. Путь камикадзе. – М.: "ЛОРИ", 2004. – 286с.
114. Fente J., Knutson K., Schexnayder C. Defining a Beta Distribution Function for Construction Simulation // Proceedings of the 1999 Winter Simulation Conference.
115. Geoghegan M.C., Pangaro P. Design for a Self-Regenerating Organization // Ashby Centenary Conference. University of Illinois, Urbana, March 4-6, 2004.
116. Павлов А.А., Теленик С.Ф. Информационные технологии и алгоритмизация в управлении. – К.: Техника, 2002. – 344с.
117. Э. Хант, Д. Томас. Программист-прагматик. – М.: «Лори», 2004, 270с.
118. Дымо А.Б., Олейник А.И. Определение объема открытости исходного кода в программном проекте методом анализа рисков // Вестник ХНТУ. – Херсон: ХНТУ, 2007. – №4(27). – С. 248 - 251.
119. Дымо А.Б. Исследование динамики системы управления программными проектами с открытым кодом // Збірник наукових праць НУК. – Миколаїв: НУК, 2007. – № 3 (414). – С.174-179.
120. Wheeler D.A. Why Open Source Software / Free Software (OSS/FS, FLOSS, or FOSS)? Look at the Numbers! // ComputerWorld. – 2005. – № 7. – p. 26.
121. Evans P., Gray D., Gittlein K. The Boston Consulting Group Hacker Survey: BGG Report. – 2002. – Release 0.73.

ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ А.

ХРОНОЛОГИЯ УПРАВЛЕНЧЕСКИХ ДЕЙСТВИЙ
В ПРОЕКТАХ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ
С ОТКРЫТЫМ ИСХОДНЫМ КОДОМ

**Хронология управленческих действий. KDE – проект по разработке
комплекса программного обеспечения с графическим интерфейсом для
"рабочего стола" пользователя**

Дата	Действие	Аннотация
14.10.1996	Письмо основателя проекта Mattias Ettrich с сообщением о начале проекта, перечнем выполняемых задач и целей.	Определение проекта
Октябрь 1996	Создан список рассылки kde@kde.org для разработчиков и пользователей проекта, веб-сайт проекта. Опробована, а затем запущена система управления версиями кода для хранения кода проекта.	Прототипирование и реализация инфраструктуры макропроцесса
Ноябрь 1996	Начальной группой разработчиков используя список рассылки выполнен анализ и обзор существующих альтернатив.	Обзор существующих аналогичных систем
Ноябрь 1996	Определены первостепенные цели проекта.	Определение структуры целей и задач проекта
12.07.1998	Завершена разработка первой версии продукта KDE 1.0.	Прототипирование и разработка
Начало 1999	Введение в действие системы управления запросами на внесение изменений (Bugzilla) как ответ на пожелания пользователей оценки.	Пользовательская оценка, усовершенствование макропроцесса
1998-1999	Доработка продукта в соответствии с требованиями пользователей, выпуск обновлений.	Внедрение и улучшение
15.12.1999	Завершено прототипирование следующей версии продукта.	Прототипирование новой версии
23.10.2000	Завершена разработка следующей версии продукта.	Разработка новой версии

Хронология управленческих действий. GNOME – проект по разработке комплекса программного обеспечения с графическим интерфейсом для "рабочего стола" пользователя

Дата	Действие	Аннотация
Август 1997	Аннонс основателя проекта Miguel de Icaza с сообщением о начале проекта, перечнем выполняемых задач и целей.	Определение проекта
Август-Ноябрь 1997	Зарегистрирован домен gnome.org, запущена система управления версиями кода для хранения кода проекта, создания списка рассылки gnome-devel-list@gnome.org.	Прототипирование и реализация инфраструктуры макропроцесса
Август 1997	Дискуссия Miguel de Icaza, Richard Stallman, Eric S. Raymond и компании Trolltech о лицензировании аналогичного проекта KDE.	Обзор существующих аналогичных систем
Декабрь 1997	Совещание главных разработчиков Miguel de Icaza, Marc Ewing и DrMike в офисе компании Red Hat.	Определение структуры целей и задач проекта
Декабрь 1997 - Март 1999	Выпуск версий 0.13, 0.20, 0.30, 0.99 и 1.0	Прототипирование и разработка
Октябрь 1999	Выпуск затребованных пользователями улучшений в версии 1.0.53 и установка системы обработки запросов на изменения Gnome Bug Tracking System	Пользовательская оценка, усовершенствование макропроцесса
Октябрь 1999 – Июнь 2002	Выпуск обновлений Gnome 1.x	Внедрение и улучшение
Апрель 2001 – Июнь 2002	Прототипирование и разработка версии 2.0	Прототипирование и разработка новой версии

Хронология управленческих действий. KDevelop – проект по разработке интегрированной среды разработчика

Дата	Действие	Аннотация
Апрель 1998	Аннонс Sandy Meier про начало работы над KDevelop.	Определение проекта
Апрель 1998 – Март 1999	Настройка списка рассылки kdevelop@kdevelop.org , инициализация модуля kdevelop в репозитории кода KDE CVS.	Прототипирование и реализация инфраструктуры макропроцесса
Март 1999 – Апрель 1999	Обзоры других IDE и обсуждение необходимой функциональности в списке рассылки kdevelop@kdevelop.org.	Обзор существующих аналогичных систем и определение структуры целей и задач проекта
11.12.1999	Выпуск версии 1.0 продукта.	Прототипирование и разработка
Начало 2000г.	Начало работы с компонентом "kdevelop" системы управления запросами на изменения KDE Bugzilla.	Пользовательская оценка, усовершенствование макропроцесса
28.02.2000 – 04.01.2001	Доработка продукта и выпуск улучшенных версий 1.1, 1.2, 1.3 и 1.4	Внедрение и улучшение
04.01.2001 – 15.08.2001	Разработка и выпуск версии 2.0 продукта.	Прототипирование и разработка новой версии

Хронология управленческих действий. – Kontakt проект по разработке программного обеспечения групповой работы

Дата	Действие	Аннотация
22.03.2000	Обсуждение интеграции компонентов групповой работы в единый проект.	Определение проекта
Октябрь 2000	Начало сбора существующего кода в группу "kderim".	Прототипирование и реализация инфраструктуры макропроцесса
19.01.2001	Документ "Proposal for a KDE syncing Library – libksync" (автор Cornelius Schumacher), обозначивший цели и пути развития проекта.	Обзор существующих аналогичных систем и определение структуры целей и задач проекта
Январь 2001 – Апрель 2002	Разработка начальной версии продукта (названного в то время "Kaplan"), размещение в KDE CVS.	Прототипирование и разработка
Апрель 2002 – Январь 2003	Выполнение действий по реорганизации всего существующего кода в модуль "kderim" в KDE CVS, интеграции команд разработчиков в проект "Kontakt" и начало применения системы управления запросами на изменения KDE Bugzilla.	Пользовательская оценка, усовершенствование макропроцесса, внедрение и улучшение
Январь 2003 – 03.02.2004	Разработка первой версии "Kontakt", включенной в выпуск KDE 3.2.	Прототипирование и разработка новой версии

Хронология управленческих действий. – Eclipse проект по разработке программного обеспечения интегрированной среды разработчика

Дата	Действие	Аннотация
Ноябрь 2001	Формирование "eclipse.org Board of Stewards" и выпуск документа о намерениях.	Определение проекта
7.11.2001	Создание CVS сервера с исходным кодом, размещение и выпуск версии 1.0 продукта.	Прототипирование и реализация инфраструктуры макропроцесса
Ноябрь 2001	Формирование плана действий 9-ю участниками Eclipse Board of Stewards.	Обзор существующих аналогичных систем, определение структуры целей и задач проекта
Ноябрь 2001 – 27.06.2002	Разработка и выпуск версии 2.0 продукта.	Прототипирование и разработка
29.08.2002, 7.11.2002	Доработка продукта и выпуск исправленных версий 2.0.1 и 2.0.2. Привлечение новых членов в Eclipse Board of Stewards (80 участников).	Пользовательская оценка, усовершенствование макропроцесса, внедрение и улучшение
Август 2002 – 27.03.2003	Прототипирование и разработка версии 2.1 продукта.	Прототипирование и разработка новой версии

Хронология управленческих действий. MinGW – проект по разработке компилятора GCC для платформы Windows

Дата	Действие	Аннотация
01.07.1998	Аннонс Колина Петерса про начало работы над проектом. Первый выпуск продукта.	Определение проекта, определение структуры целей и задач проекта, прототипирование и разработка
Июль 1998	Создание web-сайта Колина Петерса с исходным кодом MinGW, привлечение новых разработчиков, начало использования списка рассылки родственного проекта Cygwin для целей проекта MinGW.	Прототипирование и реализация инфраструктуры макропроцесса
1998 – 2001	Выпуск исправлений и улучшений в версиях 0.4 и 0.5. Работа Jan-Jaap van der Heijden и Mumit Khan над координацией проекта.	Внедрение и улучшение
Август 1999 – Март 2000	Установка собственного списка рассылки, начало использование инфраструктуры Sourceforge.net для управления проектом.	Пользовательская оценка, усовершенствование макропроцесса
Март 2000 – 08.06.2001	Разработка версии 1.0 продукта.	Прототипирование и разработка новой версии

**Хронология управленческих действий. FVWM – проект по разработке
оконного менеджера для среды UNIX/X11**

Дата	Действие	Аннотация
Июнь 1993	Аннонс основателя проекта Robert Nation про начало разработки.	Определение проекта
Июнь 1993 – Октябрь 1993	Привлечение сторонних разработчиков, организация инфраструктуры проекта, выделение задач проекта FVWM и разделение его на два независимых подпроекта – разработки оконного менеджера и терминала.	Прототипирование и реализация инфраструктуры макропроцесса, обзор существующих аналогичных систем, определение структуры целей и задач проекта
Июнь 1993 – Октябрь 1993	Разработка и выпуск версии 1.0 продукта.	Прототипирование и разработка
1993-1995	Дискуссии в новостной группе comp.windows.x.apps, начало использования списка рассылки fvwm@shrug.org.	Пользовательская оценка, усовершенствование макропроцесса
1993-1995	Выпуск исправленных и улучшенных версий 1.x продукта.	Внедрение и улучшение
1995 – Май 1998	Разработка версии 2.0 продукта.	Прототипирование и разработка новой версии

Хронология управленческих действий. Ruby – проект по разработке интерпретатора языка программирования с динамической типизацией

Дата	Действие	Аннотация
24.12.1993	Yukihiro Matsumoto объявил о начале работ по проекту.	Определение проекта
1993-1994	Организация команды разработчиков и необходимой инфраструктуры (CVS сервер).	Прототипирование и реализация инфраструктуры макропроцесса
1994	Сравнение языка Ruby с языками Python и Perl, определение направлений разработки и целей проекта (решение про одновременную простоту и сложность языка).	Обзор существующих аналогичных систем, определение структуры целей и задач проекта
1994-1995	Разработка версии 1.0 продукта.	Прототипирование и разработка
1995 Декабрь 1998	Распространение информации про проект за пределами Японии, начало работы англоязычного списка рассылки ruby-talk@ruby-lang.org.	Пользовательская оценка, усовершенствование макропроцесса, внедрение и улучшение
1999-2007	Разработка новых версий (1.4, 1.6, 1.8) продукта.	Прототипирование и разработка новой версии

Хронология управленческих действий. Perl – проект по разработке языка программирования с динамической типизацией

Дата	Действие	Аннотация
18.12.1987	Аннонс Larry Wall про разработку проекта Perl и выпуск прототипа 1.0 продукта.	Определение проекта, определение структуры целей и задач проекта
1988	Использование CVS сервера и новостной группы comp.sources.misc.	Прототипирование и реализация инфраструктуры макропроцесса
1998- 17.10.1994	Разработка версий 2.0, 3.0, 4.0 и 5.0 продукта.	Прототипирование и разработка
26.10.1995	Создание Comprehensive Perl Archive Network (CPAN) с репозитариями для проекта Perl и проектов разработки модулей и расширений к Perl.	Пользовательская оценка, усовершенствование макропроцесса
1995-2000	Разработка версий 5.x продукта, разработка и включение в проект сторонних расширений.	Внедрение и улучшение
11.03.2000	Выпуск версии 5.6 продукта.	Прототипирование и разработка новой версии

**Хронология управленческих действий. GIMP – проект по разработке
графического редактора**

Дата	Действие	Аннотация
Февраль 1996	Peter Mattis и Spencer Kimball принимают решение про начало работ над проектом.	Определение проекта
Февраль 1996 – Июль 1996	Работа над прототипированием продукта, обзор и выделение функциональности, необходимой растровому редактору, выбор библиотеки построения графического интерфейса пользователя (Motif, GTK).	Прототипирование и реализация инфраструктуры макропроцесса, обзор существующих аналогичных систем, определение структуры целей и задач проекта
Июль 1996 – 05.06.1998	Прототипирование и разработка новой версии продукта 1.0.	Прототипирование и разработка
1998-2000	Выделение в отдельный проект библиотеки графического интерфейса пользователя GTK.	Пользовательская оценка, усовершенствование макропроцесса
25.12.2000	Доработка и улучшения продукта, выпуск обновлений в версии 1.2.	Внедрение и улучшение
2001 – Июнь 2006	Разработка версии 2.0 продукта.	Прототипирование и разработка новой версии

**Хронология управленческих действий. – Amarok проект по разработке
проигрывателя звуковых файлов**

Дата	Действие	Аннотация
20.06.2003	Выпуск первой публичной версии продукта, начало использования инфраструктуры KDE (CVS сервер и KDE Bugzilla) для разработки.	Определение проекта, прототипирование и реализация инфраструктуры макропроцесса
2003	Обзор usability проблем плеера XMMS и выбор путей их решения для проекта Amarok. Выбор парадигмы пользовательского интерфейса на основе Midnight Commander.	Обзор существующих аналогичных систем, определение структуры целей и задач проекта
20.09.2003 – 17.06.2004	Разработка прототипов и новой версии 1.0 продукта.	Прототипирование и разработка
2004-2007	Выпуск улучшений и дополнений в релизах 1.1, 1.2, 1.3, 1.4 продукта. Переход с репозитария CVS на репозитарий SVN. Организация системы сбора пожертвований пользователей для обеспечения бюджета проекта.	Пользовательская оценка, усовершенствование макропроцесса, внедрение и улучшение
2006-2007	Разработка версии 2.0 продукта.	Прототипирование и разработка новой версии

**Хронология управленческих действий. GNUStep – проект по разработке
NextStep подобной среды разработки**

Дата	Действие	Аннотация
11.05.1991	Barry Merriman публикует идею создания NextStep подобной среды разработки с открытым исходным кодом.	Определение проекта
27.01.1994	Выпуск «Call for volunters», организация репозитариев кода, команды разработчиков.	Прототипирование и реализация инфраструктуры макропроцесса
1991 - 1994	Обзор функциональности NextStep и возможности открытия имеющегося исходного кода. Принятие решения про необходимость разработки библиотеки коллекций objective-C и графического	Обзор существующих аналогичных систем, определение структуры целей и задач проекта
1991 08.11.1994	– Выпуск версии 0.1 продукта.	Прототипирование и разработка
Март 1995	Начало использования репозитария исходного кода GNU CVS.	Пользовательская оценка, усовершенствование макропроцесса
1995-2002	Выпуск улучшенных и дополненных версий 0.x продукта	Внедрение и улучшение
16.03.1998 27.06.2003	– Разработка версии 1.0 продукта.	Прототипирование и разработка новой версии

**Хронология управленческих действий. Ruby on Rails – проект по разработке
Web Development Framework**

Дата	Действие	Аннотация
Июль 2004	Решение David Heinemeier Hanssen про начало работы над проектом.	Определение проекта
2004 - 2005	Использование инфраструктуры компании 37 Signals и части исходного кода проекта Basecamp для начала работ над проектом.	Прототипирование и реализация инфраструктуры макропроцесса
2004 - 2005	Обзор Web Development Framework для языка Java, принятие решений про архитектуру MVC и ORM без конфигурирования.	Обзор существующих аналогичных систем, определение структуры целей и задач проекта
13.12.2005	Разработка и выпуск версии 1.0 продукта.	Прототипирование и разработка
2005	Установка и настройка средства управления проектом Trac с системой управления версиями и запросами на изменения.	Пользовательская оценка, усовершенствование макропроцесса
2007	Разработка версии 2.0 продукта	Разработка новой версии

ПРИЛОЖЕНИЕ Б.

ПРОГРАММЫ ЧИСЛЕННОГО МОДЕЛИРОВАНИЯ СИСТЕМЫ УПРАВЛЕНИЯ
ПРОЕКТОМ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ С
ОТКРЫТЫМ ИСХОДНЫМ КОДОМ

Б.1. Программа моделирования системы без механизмов самоорганизации

```

/*АНАЛИТИЧЕСКИЙ РАСЧЕТ*/

/*Исходная матрица коэффициентов уравнения состояния*/
A:matrix(
[ 0, adx, 0, 0, 0, 0, 0],
[ 0, 0, atd, 0, 0, 0, 0],
[ 0, 0, -att, 0, 0, 0, art],
[ axp, -adp, 0, app, 0, 0, 0],
[ 0, ads, 0, aps, 0, -abs, 0],
[ 0, 0, atb, 0, asb, 0, 0],
[ 0, 0, 0, -apr, -asr, 0, 0]
);

/*Флаг для приведения характеристического уравнения к полиномиальной форме*/
ratmx: true$

/*Символьное вычисление характеристического уравнения*/
d: charpoly(A, lambda)$

/*Умножаем обе части характеристического уравнения на -1 т.к.
число коэффициентов нечетно (7)*/
d: -1*d;

/*Извлечение коэффициентов уравнения при отдельных показателях степени*/
m0: coeff(d, lambda, 7);
m1: coeff(d, lambda, 6);
m2: coeff(d, lambda, 5);
m3: coeff(d, lambda, 4);
m4: coeff(d, lambda, 3);
m5: coeff(d, lambda, 2);
m6: coeff(d, lambda, 1);
m7: coeff(d, lambda, 0);

/*Составление матрицы Гурвица*/
hurwitz: matrix(
[m1, m3, m5, m7, 0, 0, 0],
[m0, m2, m4, m6, 0, 0, 0],
[ 0, m1, m3, m5, m7, 0, 0],
[ 0, m0, m2, m4, m6, 0, 0],
[ 0, 0, m1, m3, m5, m7, 0],
[ 0, 0, m0, m2, m4, m6, 0],
[ 0, 0, 0, m1, m3, m5, m7]
)$

/*Вычисление диагональных миноров матрицы Гурвица*/
h7: hurwitz$
h6: minor(h7, 7, 7)$
h5: minor(h6, 6, 6)$
h4: minor(h5, 5, 5)$
h3: minor(h4, 4, 4)$
h2: minor(h3, 3, 3)$
h1: minor(h2, 2, 2)$

```

```

/*Вычисление определителей диагональных миноров матрицы Гурвица*/
deth1: determinant(h1)$
deth2: determinant(h2)$
deth3: determinant(h3)$
deth4: determinant(h4)$
deth5: determinant(h5)$
deth6: determinant(h6)$
deth7: determinant(h7)$

/*Критерии устойчивости*/
deth1 > 0;
deth2 > 0;
deth3 > 0;
deth4 > 0;
deth5 > 0;
deth6 > 0;
deth7 > 0;

/*ЧИСЛЕННЫЙ РАСЧЕТ*/

/*Количество шагов для моделирования*/
steps: 10$

/*Диапазоны значений коэффициентов уравнения состояния*/
adx_min: 0$
adx_max: 1$
adx_step: (adx_max - adx_min) / steps$

atd_min: -1/8$
atd_max: 1$
atd_step: (atd_max - atd_min) / steps$

art_min: 0.73$
art_max: 1.74$
art_step: (art_max - art_min) / steps$

att_min: 0$
att_max: 5.07$
att_step: (att_max - att_min) / steps$

axp_min: 0$
axp_max: 0.8$
axp_step: (axp_max - axp_min) / steps$

adp_min: 0$
adp_max: 0.1$
adp_step: (adp_max - adp_min) / steps$

app_min: 0$
app_max: 10$
app_step: (app_max - app_min) / steps$

ads_min: 0.2$
ads_max: 1$
ads_step: (ads_max - ads_min) / steps$

abs_min: -1$
abs_max: 1$
abs_step: (abs_max - abs_min) / steps$

aps_min: 1$
aps_max: 4$
aps_step: (aps_max - aps_min) / steps$

```

```

asb_min: 0$
asb_max: 1$
asb_step: (asb_max - asb_min) / steps$

```

```

atb_min: 0$
atb_max: 1$
atb_step: (atb_max - atb_min) / steps$

```

```

asr_min: 0.5$
asr_max: 2$
asr_step: (asr_max - asr_min) / steps$

```

```

apr_min: 0$
apr_max: 1$
apr_step: (apr_max - apr_min) / steps$

```

```

/*Счетчики количества устойчивых и неустойчивых систем*/
total_count:0$
stable_count: 0$

```

```

adx_stable_min:adx_max$
adx_stable_max:adx_min$
atd_stable_min:atd_max$
atd_stable_max:atd_min$
art_stable_min:art_max$
art_stable_max:art_min$
att_stable_min:att_max$
att_stable_max:att_min$
axp_stable_min:axp_max$
axp_stable_max:axp_min$
adp_stable_min:adp_max$
adp_stable_max:adp_min$
app_stable_min:app_max$
app_stable_max:app_min$
ads_stable_min:ads_max$
ads_stable_max:ads_min$
aps_stable_min:aps_max$
aps_stable_max:aps_min$
abs_stable_min:abs_max$
abs_stable_max:abs_min$
asb_stable_min:asb_max$
asb_stable_max:asb_min$
atb_stable_min:atb_max$
atb_stable_max:atb_min$
asr_stable_min:asr_max$
asr_stable_max:asr_min$
apr_stable_min:apr_max$
apr_stable_max:apr_min$

```

```

/*Определение диапазонов устойчивости и количества устойчивых систем*/
for adx: adx_min step adx_step thru adx_max do (
for atd: atd_min step atd_step thru atd_max do (
for art: art_min step art_step thru art_max do (
for att: att_min step att_step thru att_max do (
for axp: axp_min step axp_step thru axp_max do (
for adp: adp_min step adp_step thru adp_max do (
for app: app_min step app_step thru app_max do (
for ads: ads_min step ads_step thru ads_max do (
for aps: aps_min step aps_step thru aps_max do (
for abs: abs_min step abs_step thru abs_max do (
for asb: asb_min step asb_step thru asb_max do (
for atb: atb_min step atb_step thru atb_max do (

```

```

for asr: asr_min step asr_step thru asr_max do (
for apr: apr_min step apr_step thru apr_max do (
  total_count: total_count + 1,
  c1: ev(deth1),
  c2: ev(deth2),
  c3: ev(deth3),
  c4: ev(deth4),
  c5: ev(deth5),
  c6: ev(deth6),
  c7: ev(deth7),
  if c1 > 0 and c2 > 0 and c3 > 0 and c4 > 0 and c5 > 0
    and c6 > 0 and c7 > 0 then [
    stable_count: stable_count + 1,
    if adx < adx_stable_min then adx_stable_min:adx,
    if adx > adx_stable_max then adx_stable_max:adx,
    if atd < atd_stable_min then atd_stable_min:atd,
    if atd > atd_stable_max then atd_stable_max:atd,
    if art < art_stable_min then art_stable_min:art,
    if art > art_stable_max then art_stable_max:art,
    if att < att_stable_min then att_stable_min:att,
    if att > att_stable_max then att_stable_max:att,
    if axp < axp_stable_min then axp_stable_min:axp,
    if axp > axp_stable_max then axp_stable_max:axp,
    if adp < adp_stable_min then adp_stable_min:adp,
    if adp > adp_stable_max then adp_stable_max:adp,
    if app < app_stable_min then app_stable_min:app,
    if app > app_stable_max then app_stable_max:app,
    if ads < ads_stable_min then ads_stable_min:ads,
    if ads > ads_stable_max then ads_stable_max:ads,
    if abs < abs_stable_min then abs_stable_min:abs,
    if abs > abs_stable_max then abs_stable_max:abs,
    if aps < aps_stable_min then aps_stable_min:aps,
    if aps > aps_stable_max then aps_stable_max:aps,
    if asb < asb_stable_min then asb_stable_min:asb,
    if asb > asb_stable_max then asb_stable_max:asb,
    if atb < atb_stable_min then atb_stable_min:atb,
    if atb > atb_stable_max then atb_stable_max:atb,
    if asr < asr_stable_min then asr_stable_min:asr,
    if asr > asr_stable_max then asr_stable_max:asr,
    if apr < apr_stable_min then apr_stable_min:apr,
    if apr > apr_stable_max then apr_stable_max:apr
  ],
  display(total_count),
  display(stable_count)
)))))))))$

```

```
/*Вывод полученных результатов*/
```

```
total_count;
stable_count;
```

```

adx_stable_min;
adx_stable_max;
atd_stable_min;
atd_stable_max;
art_stable_min;
art_stable_max;
att_stable_min;
att_stable_max;
axp_stable_min;
axp_stable_max;
adp_stable_min;
adp_stable_max;

```

```
app_stable_min;  
app_stable_max;  
ads_stable_min;  
ads_stable_max;  
aps_stable_min;  
aps_stable_max;  
abs_stable_min;  
abs_stable_max;  
asb_stable_min;  
asb_stable_max;  
atb_stable_min;  
atb_stable_max;  
asr_stable_min;  
asr_stable_max;  
apr_stable_min;  
apr_stable_max;
```

Б.2. Результаты работы программы моделирования системы без механизмов самоорганизации

```
earth:~/projects/system gremlin$ maxima -b system1.max  
total_count  
1000000000000000  
stable_count  
24918621107549  
adx_stable_min  
0  
adx_stable_max  
0.9  
atd_stable_min  
-0.13  
atd_stable_max  
0.89  
art_stable_min  
0.83  
art_stable_max  
1.74  
att_stable_min  
0  
att_stable_max  
5.07  
axp_stable_min  
0.08  
axp_stable_max  
0.8  
adp_stable_min  
0  
adp_stable_max  
0.08  
app_stable_min  
0  
app_stable_max  
10  
ads_stable_min  
0.2
```

```
ads_stable_max
1
aps_stable_min
1.3
aps_stable_max
3.7
abs_stable_min
-1
abs_stable_max
1
asb_stable_min
0.1
asb_stable_max
1
atb_stable_min
0.2
atb_stable_max
1
asr_stable_min
0.5
asr_stable_max
1.85
apr_stable_min
0.1
apr_stable_max
0.9
```

ПРИЛОЖЕНИЕ В.

ИНФОРМАЦИЯ О РЕПОЗИТОРИИ ISBSG И ЛИЦЕНЗИОННОЕ СОГЛАШЕНИЕ
ОБ ИСПОЛЬЗОВАНИИ ДАННЫХ ДЛЯ ЦЕЛЕЙ ИССЛЕДОВАНИЯ



Repository Data CD – Release 9 - License Agreement

Licensee Information

Licensee Name:	Oleksandr Dymo
Licensee Address:	National University of Shipbuilding
	Karpenko 32, flat 23
	Mykolayiv 54038
	Ukraine
Number of Users Licensed:	1 for approved research purposes only

Please read this document carefully before using the ISBSG Repository Data contained on the enclosed CD.

This License Agreement is entered into between the International Software Benchmarking Standards Group Limited, the owner of all rights in respect of the Data referred to above (herein referred to as : “Licensor”) of the one part and you, the Licensee, on the other.

By using the Data on the enclosed CD, you agree to become bound by the terms of this Agreement, which includes the Data License and disclaimer of warranty.

If you do not agree with the terms of this Agreement, do not use the Data and promptly return the CD to the organisation from which you obtained it or to ISBSG Limited for a full refund.

This document constitutes a License to use the enclosed Data on the terms and conditions appearing below.

The Data referred to above, and related documentation and materials (herein collectively referred to as “the Data”) are Licensed, not sold, to you for use only upon the terms of the License, and Licensor reserves all rights not expressly granted to you. You own the CDs on which this Data is originally or subsequently recorded or fixed, but Licensor retains ownership of all copies of the Data themselves.

1. License

This License allows you to

- a) Use the Data solely as an end user (that is, for individual use by the number of licensed users contracted and not for use by others, or for marketing or redistribution, alone or as a component of any other product, to any other person or company). You must enter into an additional License Agreement with Licensor and thereby purchase further licences for the Data, before use of the Data by additional users.
- b) Install the Data on a hard disk for use in accordance with the Agreement and make appropriate back-up copies. You must reproduce on any backup copies the Licensor copyright notice and any other proprietary legends that were on the original copy supplied by Licensor. The Data is protected by copyright law. You are not authorised to make any copies of the Data, except as permitted by this paragraph.
- c) Transfer the Data (including back-up copies) and all rights under this License to another party together with a copy of the Agreement provided you give Licensor written notice of the transfer and other party reads and agrees to the terms and conditions of this agreement.

2. Restrictions

- a) You may not market, distribute or transfer (other than in agreement with paragraph 1(c) above any copy of the Data to others or electronically transfer the Data from one computer to another over a network, either on its own or with or as part of any other product without an express distribution License from licensor. **YOUR MAY NOT MODIFY, ADAPT, RENT, LEASE, LOAN, SELL, DISTRIBUTE, OR NETWORK THE DATA OR ANY PART THEREOF.**

3. Termination

This License is effective until terminated. The License will terminate automatically without notice from Licensor if you fail to comply with any provision of the License. Upon termination you must destroy the Data and all copies thereof. You may terminate this License at any time by destroying the Data and all copies thereof. Upon termination of this License for any reason:

- a) you shall have no right to refund of the whole or any part of the License fees or other amounts paid for this License and the Data Licensed hereunder (except in the circumstances and expressly as provided in Section 6 below); and
- b) you shall continue to be bound by the provisions of Section 2 above.

Termination shall be without prejudice to any rights Licensor may have as a result of breach of this Agreement.

4. Limited Warranty

Licensors warrants the CD on which the Data is recorded to be free from defects in materials and faulty workmanship in normal use for a period of ninety (90) days from the date of shipment. During this warranty period, Licensor will replace, free of charge, defective CD upon which the Data has been supplied. All CD replaced under this warranty shall become the property of the Licensor.

5. Disclaimer of Warranty, Limitation of Remedies

TO THE FULL EXTENT PERMITTED BY LAW, LICENSOR HEREBY EXCLUDES ALL CONDITIONS AND WARRANTIES, WHETHER IMPOSED BY STATUTE OR BY OPERATION OF LAW OR OTHERWISE, NOT EXPRESSLY SET OUT HEREIN. THE DATA IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND. LICENSOR DOES NOT WARRANT, GUARANTEE OR MAKE ANY REPRESENTATIONS REGARDING THE USE, OR THE RESULTS OF THE USE, OF THE DATA WITH RESPECT TO ITS CORRECTNESS, ACCURACY, RELIABILITY, CURRENTNESS OR OTHERWISE.

THE ENTIRE RISK AS TO THE USE, OR THE RESULTS OF THE USE, OF THE DATA IS ASSUMED BY YOU. IF THE DATA IS DEFECTIVE, YOU, AND NOT LICENSOR OR ITS MEMBERS, DISTRIBUTORS, AGENTS OR EMPLOYEES, ASSUME THE ENTIRE COST OF ANY CORRECTION THEREOF.

EXCEPT AS SPECIFICALLY SET FORTH IN THIS SECTION 5, LICENSOR MAKES NO EXPRESSED OR IMPLIED WARRANTIES, CONDITIONS, INCLUDING, WITHOUT LIMITATION, THE WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DATA. NO ORAL OR WRITTEN INFORMATION OR ADVICE GIVEN BY LICENSOR, ITS MEMBERS, DISTRIBUTORS, AGENTS OR EMPLOYEES SHALL CREATE A WARRANTY OR IN ANYWAY INCREASE THE SCOPE OF THIS WARRANTY, AND YOU MAY NOT RELY ON ANY SUCH INFORMATION OR ADVICE.

Important Note: Nothing in the Agreement is intended or shall be construed as excluding or modifying any statutory rights, warranties or conditions which are applicable to the Agreement or the Data supplied hereunder, and which by virtue of any national or state Fair Trading, Trade Practices or other consumer legislation may not be modified or excluded. If permitted by such legislation, however, Licensor's liability for any breach of any such warranty or condition shall be and is hereby limited to either:

- a) the supply of such part of the Data licensed hereunder again; or
- b) the correction of any defect in such part of the Data licensed hereunder as Licensor at its sole discretion may determine to be necessary to correct the said breach.

EXCEPT AS SET OUT IN THIS SECTION 5, IN NO EVENT SHALL LICENSOR BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT OR CONSEQUENTIAL DAMAGES, (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS INFORMATION), EVEN IF LICENSOR OR LICENSOR MEMBER OR REPRESENTATIVE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Except as expressly set out in the Section 5, Licensor's maximum liability for damages arising under this Agreement shall be limited to the License fees paid by you for that part of the Data supplied by Licensor hereunder which caused the damages or that is the subject matter of, or is directly related to, the cause of action.

6. General

This Agreement will be construed under the laws of the State of Victoria, Australia. This Agreement contains the entire agreement between the parties hereto with respect to the subject matter hereof, and supersedes all prior agreements and/or understandings (oral or written). Failure or delay by Licensor in enforcing any right or provision hereof shall not be deemed a waiver of such provision or right with respect to the instant or any subsequent breach. If any provision of this Agreement shall be held by a court of competent jurisdiction to be contrary to law, that provision will be enforced to the maximum extent permissible, and the remaining provisions of this Agreement will remain in full force and effect.

ISBSG Repository Data Access Request Form

REQUESTOR DETAILS

Organisation Name: National University of Shipbuilding	Applicant Name: Dymo Oleksandr B. Applicant Title: Lecturer of the Information Systems and Technologies Department
Address: 9 Geroyiv Stalingrada avenue 54025 Mykolayiv, Ukraine	Supervising Professor's Name: Dr. Koshkin Constantine V. Supervisor's Title: Head of the Shipbuilding Institute, Head of the Information Systems and Technologies Department
Telephone Number: 380 512 359148	
Fax Number: 380 512 424652	E-mail Address: adymo@acm.org

RESEARCH PROPOSAL

Project title: Estimation of software development time and cost using similarity theory.
Research aims and objectives: The research aims to prove the adaptability of similarity theory to enhance analog methods of software development cost and time estimation. Similarity theory allows to build analog models of the development process and deduce model equations by analyzing the influence of determinant process parameters on determinate (such as cost and time) with dimension theory. The resulting model equations have a form of power dependence with unknown exponents. The objective is to gather experimental and real world data to calculate exponents of model equations and prove the correctness of the models acquired. After exponents are calculated and model correctness is proved, the model equations could be easily used for software development time and cost estimations of analog projects.
Names and organisation(s) of principal researcher(s): Dymo Oleksandr B., Information Systems and Technologies Department, National University of Shipbuilding

For approved research applications the ISBSG will provide a copy of the latest data CD.

If additional data is required a quotation will be provided to extract and supply the requested data.

Please provide a description of any additional data required on an attached sheet.

Description of how the data will be used / processed:

The projects data will be divided into several clusters grouped by type of the project, type of the product and development method.

For each cluster the effort, size, productivity, quality metric values will be used to calculate non-dimensional criterions that form the analog model equations.

Then for each project in a cluster the exponents of model equations will be calculated and compared with those of other projects withing the cluster.

In case the exponent values withing the cluster vary considerably, the cluster will be broken into smaller ones and the procedure above will be repeated.

After all data is analysed, for each cluster of projects the analog model with specific exponent values will be available.

Research project milestones and estimated completion date:

31.01.2006: All theoretical research is finished and analog model equations for appointed project clusters acquired.

15.02.2006: The development of automated tool calculate the criterions and exponents of model equations finished.

15.03.2006: All calculations finished, all results analyzed.

AGREEMENT

I/we acknowledge that ISBSG owns the data made available under the terms of this agreement and at all times will respect ISBSG's intellectual property rights and copyright.

I/we agree to acknowledge ISBSG as the source of the data in research publications.

I/we agree that any data made available by ISBSG under this agreement will not be used for any purpose outside the scope of this request without the express permission of ISBSG.

I/we agree not to publish any raw data furnished by ISBSG without the express permission of ISBSG.
I/we agree not to give the data or copies of the data to any other person or organisation.
I/we agree that ISBSG accepts no liability whatsoever for any inaccuracy in data supplied under this request nor for any subsequent use or interpretation of the data.

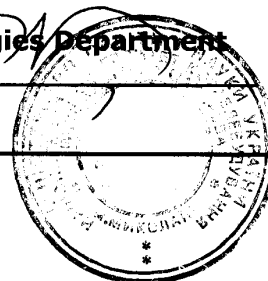
**AGREEMENT TO COMPLY WITH ISBSG POLICIES AND PROCEDURES
(Must be signed by a person duly authorised to make commitments on behalf of the requesting organisation)**

I agree to observe ISBSG policy and procedures and to abide by the terms and conditions:

Signed

**Head of the Shipbuilding Institute,
Head of the Information Systems and Technologies Department
Dr Koshkin Constantine V, Professor** _____

Date 12 / 23 / 2005





Repository Data CD Release 9 - Instructions for Licensees

1. Please read the enclosed License Agreements before using the data or programs on the enclosed CD ROM.
2. Point your web browser at ISBSG9.htm, it contains an index of the CD contents with links.
3. The disk has the following contents:
 - aa. An HTML Index of the contents, (ISBSG9.htm)
 - ab. README – Repository Data CD Release 9 – Instructions for Licensees
 - ac. License - CD Data R9 - License to use the CD Data Release 9
 - ad. License - Estimation Program - License to use the ISBSG Comparative Estimation Tool and ISBSG Estimation Reality Checker V2.
 - ae. Data CD R9 Demographics - Report on the demographics of the repository contents
 - af. Data CD R9 Field Descriptions - Data contents field descriptions
 - ag. ISBSG9.xls - CD Data R9 - the ISBSG data in an Excel Spreadsheet
 - ah. Guidance on the use of ISBSG data – a description of what the data can be used for.
 - ai. REALITY3 - Regression Estimation Tool – An executable copy of the Regression Project Estimation Tool, (“ISBSG Estimation Reality Checker V3”).
 - aj. ISBSG Reality Checker V3 User Guide – Instructions on the installation and use of the Regression Project Estimation Tool.
 - ak. Comparative Estimation Tool V3.0 User Guide - Instructions on the use of the ISBSG Comparative Estimation Tool.
 - al. Comparison.exe – An executable copy of the ISBSG Comparative Estimation Tool.
 - am. Free Project Benchmark Report - A description of the free ISBSG Project Benchmark Report
 - an. Your Comments Please - A request for your comments

- ao. ISBSG Contact Information – Contact details for the ISBSG and its members
 - ap. The Software Metrics Compendium – A description of The Software Metrics Compendium book.
 - aq. ISBSG Sponsors - Details of the Sponsors that support the ISBSG
4. After reading and agreeing to the license terms, please read the document “h. Guidance on the Use of the ISBSG Data” and the “f. Data CD R9 Field Descriptions” documents before you start using the data.

Any suggestions or comments that you may have on the data, its format or use will be welcomed. You can email comments to the administrator at: admin@isbsg.com

Note: Vendors of estimation tools who wish to embed or bundle the ISBSG data with their product must enter into a separate agreement with the ISBSG. The data contained on this CD is only for use by the contracted number of users.

ISBSG contact details:

Robyn Smith - Administration Officer

ISBSG

PO Box 127

Warrandyte 3113 Australia

Phone: +61 3 9723 5145

Fax: +61 3 9723 8545

Email: admin@isbsg.org



Repository Data Release 9 - Field Descriptions

Your ISBSG Repository Data Release contains the following data fields in this order in an MS Excel spreadsheet to allow you to perform your own analysis, estimation comparisons or benchmarking. Please remember that the terms of the licence to use this data prohibit it from being resold in any form or incorporated into any product for resale.

Field Descriptions (Project Data worksheet):

Project ID

A primary key, for identifying projects. (These Identification numbers have been 'randomised' to remove any chance of identifying a company).

Rating:

Data Quality Rating

This field contains an ISBSG rating code of A, B, C or D applied to the project data by the ISBSG quality reviewers to denote the following:

- A=** The data submitted was assessed as being sound with nothing being identified that might affect its integrity.
- B=** The submission appears fundamentally sound but there are some factors which could affect the integrity of the submitted data.
- C=** Due to significant data not being provided, it was not possible to assess the integrity of the submitted data.
- D=** Due to one factor or a combination of factors, little credibility should be given to the submitted data.

Unadjusted Function Point Rating

This field contains an ISBSG rating code of A, B, C or D applied to the Functional Size (Unadjusted Function Point count) data by the ISBSG quality reviewers to denote the following:

- A=** The unadjusted function point count was assessed as being sound with nothing being identified that might affect its integrity.
- B=** The unadjusted function point count appears sound, but integrity cannot be assured as a single figure was provided.
- C=** Due to unadjusted function point or count breakdown data not being provided, it was not possible to provide the unadjusted function point data.

D= Due to one factor or a combination of factors, little credibility should be given to the unadjusted function point data.

Sizing:

Count Approach

A description of the technique used to size the project. For most projects in the ISBSG repository this is the Functional Size Measurement Method (FSM Method) used to measure the functional size (e.g. IFPUG, MARK II, NESMA, COSMIC-FFP etc.). For projects using Other Size Measures (e.g. LOC etc.) the size data is in the section Size Other than FSM. Helps you to compare apples with apples.

Functional Size

The unadjusted function point count (before any adjustment by a Value Adjustment Factor if used). This may be reported in different units depending on the FSM Method.

Adjusted Function Points

For IFPUG, NESMA and MARK II counts this is the adjusted size (the functional size is adjusted by a Value Adjustment Factor) The resultant adjusted size is reported in adjusted function points (AFP). Where the Adjusted Size has not been supplied by the project then the Functional Size is used in the calculations that use AFPs.

Value Adjustment Factor

The adjustment to the function points, applied by the project submitter, that takes into account various technical and quality characteristics e.g.: data communications, end user efficiency etc. This data is not reported for some projects, (i.e. it equals 1).

Effort:

Summary Work Effort

Provides the total effort in hours recorded against the project.

Normalised Work Effort

For projects covering less than a full development life-cycle, this value is an estimate of the full development life-cycle effort. For projects covering the full development life-cycle, and projects where development life-cycle coverage is not known, this value is the same as Summary Work Effort.

Productivity:

Reported Productivity Delivery Rate (adjusted size units)

Project productivity delivery rate in hours per function point calculated from Summary Work Effort divided by Adjusted Function Point count. This is the delivery rate previously reported for the project used for all analysis and comparison prior to the ISBSG Software Metrics Compendium.

Project Productivity Delivery Rate (functional size units)

Project productivity delivery rate in hours per function point calculated from Summary Work Effort divided by Functional Size (Unadjusted Function Point count).

Normalised Productivity Delivery Rate (adjusted size units)

Project productivity delivery rate in hours per function point calculated from Normalised Work Effort divided by Adjusted Function Point count.

Normalised Productivity Delivery Rate (functional size units)

Project productivity delivery rate in hours per function point calculated from Normalised Work Effort divided by the Functional Size (Unadjusted Function Point count). Use of normalised effort and unadjusted count should render more comparable rates.

Schedule:

Project Elapsed Time

Total elapsed time for the project in calendar months.

Project Inactive Time

This is the number of calendar months in which no activity occurred, (e.g.. awaiting client sign off, awaiting acceptance test data). This time, subtracted from Project Elapsed Time, derives the actual time spent working on the project.

Implementation Date

Actual date of implementation. (Note: where the exact date is not known the date is shown in the data in date format 1/mm/yy).

Project Activity Scope

This indicates what tasks were included in the project work effort data recorded. These are: Planning, Specify, Design, Build, Test and Implement.

Effort Breakdown

When provided in the submission, these fields contain the breakdown of the work effort reported by six categories: Plan, Specify, Design, Build, Test and Implement.

Effort Unphased

Where no phase breakdown is provided in the submission, this field contains the same value as the Summary Work Effort. Where phase breakdown is provided in the submission, and the sum of that breakdown does not equal the Summary Work Effort, the difference is shown here.

Quality:**Defects Delivered**

Defects reported in the first month of use of the software. Three columns in the data covering the number of Minor, Major and Extreme defects reported.

Total Defects Delivered

Total number of defects reported in the first month of use of the software. This column shows the total of defects reported (Minor, Major and Extreme). Where no breakdown is available, the single value is shown here.

Grouping Attributes:**Development Type**

This field describes whether the development was a new development, enhancement or re-development.

Organisation Type

This identifies the type of organisation that submitted the project. (e.g.: Banking, Manufacturing, Retail).

Business Area Type

This identifies the type of business area being addressed by the project where this is different to the organisation type. (e.g.: Manufacturing, Personnel, Finance).

Application Type

This identifies the type of application being addressed by the project. (e.g.: information system, transaction/production system, process control.)

Package Customisation

This indicates whether the project was a package customisation. (Yes, No or Don't Know).

Degree of Customisation

If the project was based on an existing package, this field provides comments on how much customisation was involved.

Architecture:

Architecture

A derived attribute for the project to indicate if the application is Stand alone, Multi-tier, Client server, or Multi-tier with web public interface.

Client Server?

Indicator of whether the application or product requires more than one computer to operate different components or parts of it. (Yes, No or Don't Know).

Client Roles

The roles performed by the computers that provide interface to the software's external users.

Server Roles

The services provided by the host/server computer(s) to the software application or product.

Type of server

A description of the server to the software application or product. This data comes from a previous version of the questionnaire.

Client/server description

A description of the architecture of the client/server software application or product. This data comes from a previous version of the questionnaire.

Web development

A derived indicator of whether the project data includes any comment that it is a web-development.

Documents & Techniques:

Plan documents

The documents or other work products produced during the planning activity.

Specification Documents

The documents or other work products produced during the specification activity.

Specification Techniques

The techniques used during the specification of the software.

Design Documents

The documents or other work products produced during the design activity.

Design Techniques

The techniques used during the design of the software.

Build Products

The items that were produced or modified during the build activity.

Build Activity

The detailed activities that occurred during the build of the software.

Test Documents

The documents or other work products produced during the planning or performance of testing.

Test Activity

The detailed activities that occurred during the testing of the software.

Implementation Documents

The documents or other work products produced during preparation for, or performance of, the implementation activity.

Implementation Activity

The detailed activities that occurred during the implementation of the software.

Development Techniques

Techniques used during development. (e.g.: JAD, Data Modeling, OO Analysis etc.).

Functional Sizing Technique

The technology used to support the functional sizing process. Certain technologies used in function point counting can impact on the count's potential accuracy.

FP Standard (Function Size Metric Used)

The functional size metric used to record the size of the project, (e.g.. IFPUG3, IFPUG4 [version 4 series = 4.0,4.1, 4.1.1, 4.2 etc], in-house etc.). Where more than 1 standard has been recorded for the project, this has been rationalised.

FP Standards All

All functional size metrics used to record the size of the project. This column shows all standards recorded for the project, (e.g. IFPUG3;IFPUG4;in-house).

Reference Table Approach

This describes the approach used to assess tables of code or reference data, (a comment field) for their contribution to functional size.

Project Attributes:

Development Platform

Defines the primary development platform, (as determined by the operating system used). Each project is classified as: PC, Mid Range, Main Frame or Multi platform.

Language Type

Defines the language type used for the project: e.g. 3GL, 4GL, Application Generator etc.

Primary Programming Language

The primary language used for the development: JAVA, C++, PL/1, Natural, Cobol etc.

1st Hardware

Where known, this is the primary technology hardware platform used to build or enhance the software

(i.e. that used for most of the build effort).

1st Operating System

Where known, this is the primary technology operating system used to build or enhance the software

(i.e. that used for most of the build effort).

1st Language

Where known, this is the primary technology programming language used to build or enhance the software (i.e. that used for most of the build effort).

1st Data Base System

Where known, this is the primary technology database used to build or enhance the software

(i.e. that used for most of the build effort), otherwise (if known) it is whether the project used a DBMS.

1st Component Server

Where known, this is the primary technology object/component server used to build or enhance the software (i.e. that used for most of the build effort); otherwise (if known) it is whether the project used an object/component server.

1st Web Server

Where known, this is the primary technology HTML/Web server used to build or enhance the software (i.e. that used for most of the build effort); otherwise (if known) it is whether the project used an HTML/Web server.

1st Message Server

Where known, this is the primary technology E-Mail or message server used to build or enhance the software (i.e. that used for most of the build effort), otherwise (if known) it is whether the project used an E-Mail or message server.

1st Debugging tool

Where known, this is the primary technology debugging tool used to build or enhance the software

(i.e. that used for most of the build effort), otherwise (if known) it is whether the project used a debugging tool.

1st Other Platform

Where known, this is any other component of the primary technology used to build or enhance the software (i.e. that used for most of the build effort).

2nd Hardware

Where known, this is the secondary or other technology hardware platform used to build or enhance the software (i.e. that used for remainder of the build effort).

2nd Operating System

Where known, this is the secondary or other technology operating system used to build or enhance the software (i.e. that used for remainder of the build effort).

2nd Language

Where known, this is the secondary or other technology programming language used to build or enhance the software (i.e. that used for remainder of the build effort).

2nd Data Base System

Where known, this is the secondary or other technology database used to build or enhance the software (i.e. that used for remainder of the build effort), otherwise (if known) it is whether the project used a secondary DBMS.

2nd Component Server

Where known, this is the secondary or other technology object/component server used to build or enhance the software (i.e. that used for remainder of the build effort), otherwise (if known) it is whether the project used a secondary object/component server.

2nd Web Server

Where known, this is the secondary or other technology HTML/Web server used to build or enhance the software (i.e. that used for remainder of the build effort), otherwise (if known) it is whether the project used a secondary HTML/Web server.

2nd Message Server

Where known, this is the secondary or other technology E-Mail or message server used to build or enhance the software (i.e. that used for remainder of the build effort), otherwise (if known) it is whether the project used a secondary E-Mail or message server.

2nd Other Platform

Where known, this is any other component of the secondary or other technology used to build or enhance the software (i.e. that used for remainder of the build effort).

CASE Tool Used

Whether the project used any CASE tool. The full repository holds a breakdown of CASE usage for those projects that reported using a CASE tool:

- Upper CASE tool
- Lower CASE tool with code generator
- Lower CASE tool without code generator
- Integrated CASE tool

Used Methodology

States whether a development methodology was used by the development team to build the software.

How Methodology Acquired

Describes whether the development methodology was purchased or developed in-house, or a combination of these.

Product Attributes:**User Base - Business Units**

Number of business units (or project business stakeholders) serviced by the software application.

User Base - Locations

Number of physical locations being serviced/supported by the installed software application.

User Base - Concurrent Users

Number of users using the system concurrently.

Intended Market

This field describes the relationship between the project's customer, end users and development team.

Effort Attributes:

Recording Method

The method used to obtain work effort data:

Staff Hours (Recorded) – The WORK EFFORT reported comes from a “daily” record of all the WORK EFFORT expended by each person on project related tasks.

Staff Hours (Derived) – The WORK EFFORT reported is derived from time records that indicate, for example, the assignment of people to the project. This might entail estimating that, for example, only 75% of the assigned time was actually applied to the project; the rest is for holidays, education, etc.

Productive Time Only – The WORK EFFORT reported is only for the “productive time” spent by each person on the project. This often amounts to only 5-6 hours per day.

Combination – A combination of recorded and derived methods was used to obtain the WORK EFFORT.

No timesheets recorded by development team – No timesheets were recorded by the development team.

Recorded total hours each day or week – Only the total hours worked each day or week was recorded as WORK EFFORT.

Recorded hours on each project/day/week – The WORK EFFORT was recorded as hours worked on each project for each day/week.

Recorded work on project tasks each day – The WORK EFFORT was recorded for each project task for each day.

Resource Level

Data is collected about the people whose time is included in the work effort data reported. Four levels are identified in the data collection instrument.

- 1** = development team effort (e.g., project team, project management, project administration)
- 2** = development team support (e.g., database administration, data administration, quality assurance, data security, standards support, audit & control, technical support)
- 3** = computer operations involvement (e.g., software support, hardware support, information centre support, computer operators, network administration)
- 4** = end users or clients (e.g., user liaisons, user training time, application users and/or clients)

The number in this field indicates that all effort at this and preceding levels is included in the effort fields. For example, a “3” in this field for a project means that the work effort for the development team, development team support and computer operations is included in the work effort number.

Max Team Size

The maximum number of people that worked at any time on the project, (peak team size).

Average Team Size

The average number of people that worked on the project, (calculated from the team sizes per phase).

Ratio of Project Work Effort to Non-Project Activity

The ratio of Project Work Effort to Non-Project Activities.

Percentage of Uncollected Work Effort

The percentage of Work Effort not reflected in the reported data. i.e. an estimate of the work effort time not collected by the method used. The report typically is stated in the following terms:

1. less than 5% of that recorded,
2. between 5% and 10% of that recorded,
3. ___ % over that recorded, and
4. unable to estimate.

Size Attributes:

Function Point Categories

Input count / Output count / Enquiry count / File count / Interface count

When provided in the submission, the following five fields that breakdown the Functional Size are provided (note that all values are **unadjusted**):

Inputs For IFPUG & NESMA = functional size (Ufps) of External Input
 For MARK II = functional size (Ufps) of Input

Outputs For IFPUG & NESMA = functional size (Ufps) of External Output
 For MARK II = functional size (Ufps) of Output

Enquiries For IFPUG & NESMA = functional size (Ufps) of External Enquiry

Files For IFPUG & NESMA = functional size (Ufps) of Internal Logical Files
 For MARK II = functional size (Ufps) of Entity Reference

Interfaces For IFPUG & NESMA = functional size (Ufps) of External Interface

Added count / Changed count / Deleted count

When provided in the submission, the following three fields that breakdown the Functional Size are provided (note that all values are **unadjusted**):

Additions For IFPUG & NESMA = functional size (Ufps) of New or
 Added Functions

For MARK II = functional size (Ufps) of New or Added Functions

For COSMIC FFP = functional size (Cfsu) of New or
Added Functions

Changes For IFPUG, NESMA & MARK II = functional size (Ufps) of
Changed Functions

For COSMIC FFP = functional size (Cfsu) of Changed Functions

Deletions For IFPUG, NESMA & MARK II = functional size (Ufps) of
Deleted Functions

For COSMIC FFP = functional size (Cfsu) of Deleted Functions

Size Other than FSM:

Lines of code

The number of the source lines of code (SLOC) produced by the project. This is only available for some projects.

LOC not Statements

[Lines of Code not statements. The % of the source lines of code \(SLOC\) that are not program statements. In some cases this is a general comment on the counting of lines of code.](#)

ПРИЛОЖЕНИЕ Д.
МЕТРИКИ СВОБОДНЫХ ПРОЕКТОВ С
ОТКРЫТЫМ ИСХОДНЫМ КОДОМ

Таблица Д.1

Метрики проектов разработки интегрированных средств разработки с открытым исходным кодом

№	Проект	t , день	s , SLOC	d_c , чел	b , деф	δ_d , чел/день	r_b , деф/день	c_v , SLOC/(чел*день)	r_p , доп/день	p_v , SLOC/доп
1	QDevelop 0.20	27	16018	3	11	0,04	0,41	158,2	0,59	971,13
2	QDevelop 0.21	51	17333	4	74	0,02	1,45	67,97	0,27	1238,07
3	QDevelop 0.22	115	18191	5	225	0,01	1,96	25,31	0,03	5563,67
4	QDevelop 0.23	48	18295	6	138	0,04	2,88	50,82	0,04	9147,50
5	QDevelop 0.24	74	19403	8	141	0,03	1,91	26,22	0,26	1021,21
6	KDevelop 3.1	197	376683	20	4564	0,03	23,17	76,48	0,10	18984,15
7	KDevelop 3.2	218	380051	34	5321	0,04	24,41	41,02	0,09	19322,55
8	KDevelop 3.3	258	393510	45	6187	0,05	23,98	27,12	0,08	18584,57
9	Anjuta 2.1	244	205597	11	890	0,02	3,65	61,28	0,03	25569,63
10	Anjuta 2.2	162	217473	15	1514	0,02	9,35	71,60	0,02	54638,25
11	MonoDevelop 0.12	123	24559	25	1073	0,07	8,72	6,39	0,03	6016,75
12	MonoDevelop 0.13	169	30001	32	1221	0,04	7,22	4,44	0,02	7301,25
13	MonoDevelop 0.14	111	37443	42	1266	0,09	11,41	6,43	0,04	9494,75
14	MonoDevelop 0.15	54	51145	49	1422	0,20	26,33	15,46	0,07	12599,25
15	MonoDevelop 0.16	56	220071	53	1600	0,25	28,57	59,32	0,07	55017,75

Таблица Д.2

Критерии подобию проектов разработки интегрированных средств разработки и оцененные значения времени

№	Проект	F_t	F_{scope}	F_{qua}	F_{ext}	t_{orig} , день	t_{am} , день	δ_d , %
1	QDevelop 0.20	0,33	0,42	0,00014	1,21	27	19,3	28
2	QDevelop 0.21	0,25	0,31	0,00029	1,25	51	48,7	5
3	QDevelop 0.22	0,20	0,25	0,00007	1,15	115	137,3	19
4	QDevelop 0.23	0,33	0,42	0,00010	1,25	48	34,4	28
5	QDevelop 0.24	0,25	0,31	0,00047	1,25	74	70,7	5
6	KDevelop 3.1	0,30	0,38	0,00037	1,26	197	156,8	20
7	KDevelop 3.2	0,24	0,29	0,00030	1,27	218	221,2	1
8	KDevelop 3.3	0,27	0,33	0,00034	1,24	258	231,0	10
9	Anjuta 2.1	0,36	0,45	0,00005	1,24	244	160,2	34
10	Anjuta 2.2	0,27	0,33	0,00005	1,26	162	145,0	10
11	MonoDevelop 0.12	0,32	0,40	0,00046	1,22	123	91,8	25
12	MonoDevelop 0.13	0,22	0,27	0,00022	1,22	169	184,4	9
13	MonoDevelop 0.14	0,24	0,30	0,00029	1,27	111	111,3	1
14	MonoDevelop 0.15	0,22	0,28	0,00047	1,23	54	57,4	6
15	MonoDevelop 0.16	0,26	0,33	0,00014	1,25	56	50,7	10
	Медиана:	0,26	0,33	0,00029	1,25	Среднее арифметич. погрешности:		0,14
	Среднеквадратич. откл.:	0,05	0,06	0,00015	0,03			

Таблица Д.3

Метрики проектов разработки инструментальных библиотек с открытым исходным кодом

№	Проект	t , день	s , SLOC	d_c , чел	b , деф	δ_d , чел/день	r_b , деф/день	c_v , SLOC/(чел*день)	r_p , доп/день	p_v , SLOC/доп
1	RubyOnRails 0.9.5	415	50821	9	71	0,01	0,17	10,89	0,01	12861,25
2	RubyOnRails 1.0.0	317	73275	15	540	0,03	1,70	12,33	0,01	18165,75
3	RubyOnRails 1.1.0	105	82859	24	708	0,12	6,74	26,30	0,04	20837,75
4	RubyOnRails 1.2.0	292	103716	57	1142	0,11	3,91	4,99	0,01	25915,00
5	RubyOnRails 1.2.3	54	104653	57	1162	0,65	21,52	27,20	0,07	26007,25
6	RubyOnRails 1.2.4	206	105652	57	1511	0,15	7,33	7,20	0,02	26279,00
7	RubyOnRails 1.2.5	214	105587	60	1564	0,17	7,31	6,58	0,02	25850,75

Таблица Д.4

Критерии подобию проектов разработки инструментальных библиотек и оцененные значения времени

№	Проект	F_t	F_{scope}	F_{qua}	F_{ext}	t_{orig} , день	t_{am} , день	δ_d , %
1	RubyOnRails 0.9.5	0.67	0.83	0.00001	1.27	415	431.9	4
2	RubyOnRails 1.0.0	0.67	0.83	0.00006	1.24	317	329.9	4
3	RubyOnRails 1.1.0	0.54	0.68	0.00018	1.26	105	134.5	28
4	RubyOnRails 1.2.0	0.58	0.72	0.00009	1.25	292	350.0	20
5	RubyOnRails 1.2.3	0.62	0.77	0.00015	1.24	54	60.9	13
6	RubyOnRails 1.2.4	0.54	0.68	0.00015	1.24	206	263.7	28
7	RubyOnRails 1.2.5	0.61	0.76	0.00017	1.22	214	244.9	14
	Медиана:	0.61	0.76	0.00015	1.24	Среднее арифметич. погрешности:		16
	Среднеквадратич. откл.:	0.05	0.07	0.00006	0.01			

ПРИЛОЖЕНИЕ Е.
АКТЫ ВНЕДРЕНИЯ ОСНОВНЫХ РЕЗУЛЬТАТОВ
ДИССЕРТАЦИОННОЙ РАБОТЫ



ТОВ «УкрІнвент», 03058, Україна,
м.Київ, вул.Борщагівська 204 корпус 1
р/р 26004038301980
в філії УніКредит Банку ТзОВ, в м.Києві
код ЄДРПОУ 32919108, МФО 300744
тел. 044 4578760, 044 4578759 факс

Вих. №
від «_24_» жовтня 2007р.

"ЗАТВЕРДЖУЮ"

Директор компанії "УкрІнвент"
Г.В.

"24" жовтня



АКТ

про впровадження дисертаційної роботи Димо О.Б. "Підвищення ефективності управління проектами розробки програмного забезпечення з відкритим вихідним кодом" в компанії УкрІнвент.

Основні результати дисертаційної роботи Димо О.Б. використовуються компанією УкрІнвент при виконанні проекту розробки інструментальної бібліотеки ActiveReC++ , що призначена для забезпечення комунікацій між програмами написаними на мові C++ і веб-сервісами, побудованими за архітектурою REST.

Розроблена Димо О.Б. модель життєвого циклу дозволяє забезпечити успішне виконання проектів розробки програмного забезпечення з відкритим кодом завдяки чіткому опису переліку і послідовності необхідних дій з управління проектом. Розроблена автором математична модель системи управління дозволяє ініціалізувати необхідну інфраструктуру для ефективного управління проектом, а запропонована методика управління – виконувати процеси управління і забезпечувати вчасне завершення проекту.


Основні наукові результати дисертаційної роботи використовуються компанією і дозволяють підвищити ефективність управління і забезпечити успішне завершення проектів розробки програмного забезпечення з відкритим кодом.

Керівник проектів та програм "УкрІнвент"

<Новосільська Л.Б.>

pluron
Agile Project Management

969-G Edgewater Blvd # 300, Foster City, CA 94404, USA
1.650.491.3868/ fax 1.650.989.4062
www.pluron.com


Gleb Arshinov
President
Pluron, Inc.
gleb@pluron.com
1-650-357-1406 (direct)

STATEMENT

on application of the PhD dissertation by Dymo O.B.: "Enhancement of open source software project management efficiency" in Pluron Inc.

Primary results of the dissertation are used in Pluron Inc. to manage the MediaCloth project. MediaCloth is the library to help building MediaWiki-like systems using Ruby programming language.

Self-organization principles in open source software projects developed in the dissertation allow to manage the project even under destabilizing conditions. This makes it possible to react on changes faster and finish the project in time. Software development time estimation models are especially useful for project plan development during project initialization phase.

Primary scientific results of the dissertation are used by the company to effectively manage open source software projects.

АКТ

про впровадження дисертаційної роботи Димо О.Б. "Підвищення ефективності управління проектами розробки програмного забезпечення з відкритим вихідним кодом" в компанії Pluron.

Основні результати дисертаційної роботи Димо О.Б. використовуються компанією Pluron при виконанні проекту розробки інструментальної бібліотеки MediaCloth, що призначена для побудови MediaWiki-подібних систем на мові програмування Ruby.

Розроблені автором принципи забезпечення самоорганізації в програмних проектах з відкритим кодом дозволяють виконувати проект в складних умовах зовнішнього середовища і швидко реагувати на зміни, забезпечуючи успішне завершення проекту. Авторські моделі оцінки часу виконання проекту мають особливу цінність для розробки плану проекту на етапі його ініціалізації.

Основні наукові результати дисертаційної роботи використовуються компанією для забезпечення ефективного управління проектами розробки програмного забезпечення з відкритим кодом.

"ЗАТВЕРДЖУЮ"

Ректор Національного університету



доцент кафедри будівництва імені адмірала Макарова

д.т.н., професор

Романовський Г.Ф.

2007р.

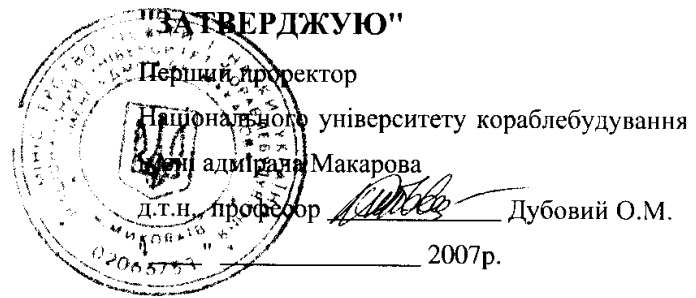
про впровадження дисертаційної роботи Димо О.Б. "Підвищення ефективності управління проектами розробки програмного забезпечення з відкритим вихідним кодом"

Дисертаційна робота Димо О.Б. "Підвищення ефективності управління проектами розробки програмного забезпечення з відкритим вихідним кодом" виконана на кафедрі інформаційних управляючих систем і технологій під керівництвом д.т.н., професора Шевцова А.П. і присвячена дослідженню життєвого циклу, системи управління і процесів управління програмними проектами з відкритим вихідним кодом з метою підвищення ефективності управління і забезпечення успішного завершення програмних проектів в рамках заданих часу, вартості і якості.

Результати дисертаційної роботи застосовувались з 2004 по 2007 рр. при виконанні підпроектів автоматизації бухгалтерського обліку проекту розробки програмного комплексу автоматизації діяльності університету. Запропонована в дисертаційній роботі методика управління використовувалась для управління розробкою підсистем "Головна книга", "Первинні документи", "Навчання за контрактом" і "Зв'язок головної книги" програмного комплексу автоматизації діяльності університету.

Головний бухгалтер

Зіміна І.А.



ДОВІДКА

про використання в навчальному процесі дисертаційної роботи Димо О.Б. "Підвищення ефективності управління проектами розробки програмного забезпечення з відкритим вихідним кодом"

Дисертаційна робота Димо О.Б. "Підвищення ефективності управління проектами розробки програмного забезпечення з відкритим вихідним кодом" виконана на кафедрі інформаційних управляючих систем і технологій під керівництвом д.т.н., професора Шевцова А.П. і присвячена дослідженню життєвого циклу, системи управління і процесів управління програмними проектами з відкритим вихідним кодом з метою підвищення ефективності управління і забезпечення успішного завершення програмних проектів в рамках заданих часу, вартості і якості.

Розроблені в дисертаційній роботі методика управління проектами з відкритим кодом і моделі оцінки часу і вартості розробки програмного забезпечення застосовуються з 2005р. в лекційних курсах і практичних заняттях з дисципліни "Методи, засоби і моделі проектного менеджменту". Розроблені програми чисельного моделювання системи управління програмними проектами використовуються при виконанні лабораторних робіт з дисципліни "Інформаційні технології проектного менеджменту".

Директор
 інституту комп'ютерних та інженерних наук,
 д.т.н., професор



Кошкін К.В.

"ЗАТВЕРДЖУЮ"



Директор з персоналу та НТГ

П.І.Н.В.К.Г. "Зоря" "Машпроект"

д.т.н., професор Чернов С.К.

" 06.11.2007р.

АКТ

про впровадження дисертаційної роботи Димо О.Б.

"Підвищення ефективності управління проектами розробки програмного забезпечення з відкритим вихідним кодом"

Результати дисертаційної роботи Димо О.Б., виконаної на кафедрі Інформаційних управляючих систем і технологій Національного університету кораблебудування імені адмірала Макарова під керівництвом д.т.н., професора Шевцова А.П., використана при виконанні дослідницьких робіт за договором про співпрацю на тему "Аналіз перспектив та можливостей використання відкритого програмного забезпечення на підприємствах машинобудівної галузі". Автором передані методичні рекомендації щодо управління проектами впровадження програмних засобів з відкритим кодом.

Заступник начальника управління
інформаційних технологій

Заїка В.П.