МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ
УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра автоматизации обработки информации (АОИ)

С.А. Мытник, Ю.Б. Гриценко

Программная инженерия

**Методические указания по выполнению лабораторных работ
и организации самостоятельной работы для студентов,
обучающихся по направлению 080700.62 «Бизнес-информатика»**

2012

СОДЕРЖАНИЕ

1. Лабораторная работа № 1. Планирование работ по выполнению проекта «Разработка и внедрение программного обеспечения»

Количество аудиторных часов: 4.

Цели занятия: Получение первичных навыков планирования работ по разработке и внедрению автоматизированных информационных систем на основе распространенных моделей жизненных циклов программных продуктов.

Цель работы: На основе технического задания на разработку и внедрение автоматизированной информационной системы сформировать календарный план выполнения работ по проекту.

Средства реализации: Ограничений не накладывается, рекомендуется использование Microsoft Project 2007.

Исходные данные: В качестве исходных данных выступает техническое задание на разработку и внедрение автоматизированной информационной системы, перечень доступных для использования трудовых ресурсов.

Для проведения успешного проекта нужно оценить объем предстоящих работ, возможный риск, требуемые ресурсы, предстоящие задачи, определить контрольные точки, стоимость и план работ, которому желательно следовать. Процесс руководство программным проектом включает решение вышеперечисленных задач. Этот процесс начинается перед технической работой, продолжается по мере развития ПО от идея к реальности и достигает наибольшей интенсивности к концу работ.

Основной задачей при планировании является определение структуры распределения работ WBS (Work Breakdown Structure) с помощью средств планирования работ (например, MS Project).

План выполнения работ составляется на основе декомпозиции проекта вплоть до постановки элементарных задач, которые могут быть выяснены по результатам предварительного анализа. При этом возможно применение содержательных моделей системного анализа. Например, использование модели декомпозиции типа «жизненный цикл» позволит разбивать отдельные задачи на подзадачи путем определения последовательности действий.

Процесс декомпозиции будет определяться принятой моделью жизненного цикла разработки программного обеспечения.

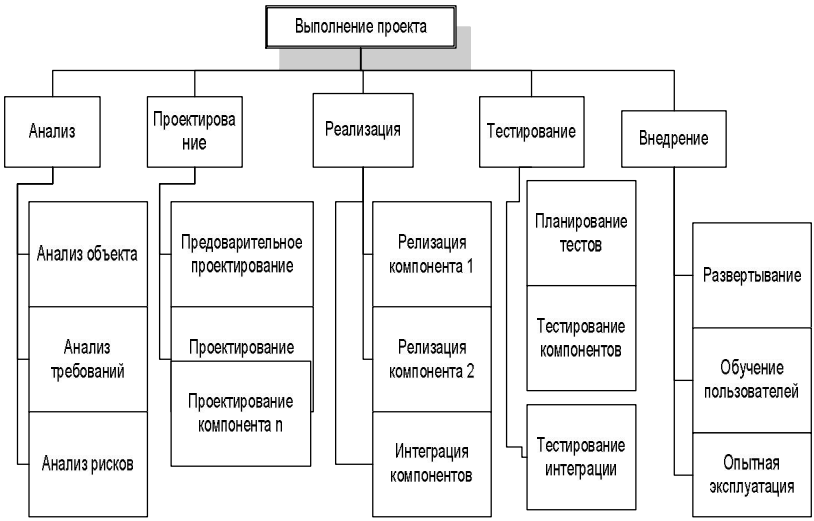

Рис. 1.1. Декомпозиция задач, которые необходимо решить в процессе выполнения проекта по разработке программного обеспечения

Для каждой элементарной задачи должны быть определены:

- ресурсы, необходимые для решения задачи (в том числе трудовые);
- объем работ, выраженный в принятой системе метрик;
- стоимость работ (может быть вычислена на основе объема работ и стоимости привлекаемых ресурсов);
- длительность работ (может быть вычислена на основе объема работ, количества привлекаемых рудовых ресурсов и принятых нормативов производительности).

Между отдельными элементарными задачами могут быть определенные зависимости, заключающиеся в том, что одни задачи могут выполняться параллельно, другие — в строгой последовательности (для выполнения одних задач могут требоваться результаты выполнения других).

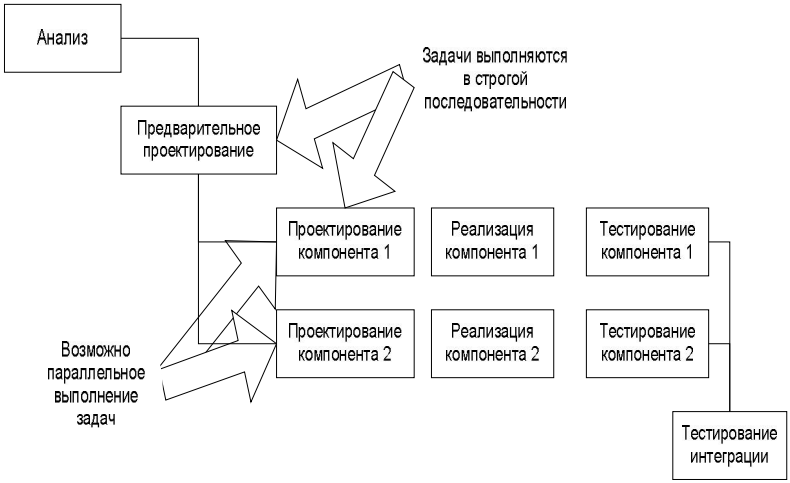

Рис. 1.2. Параллельное и последовательное выполнение задач

После определения зависимостей можно приступать к распределению элементарных задач по времени. При этом особое внимание следует остановить на задачах, выполняемых параллельно. Параллельность действий повышает требования к планированию. Необходимо четко отследить наличие ресурсов, необходимых для выполнения каждой задачи. Если план предусматривает использование ресурса 1 для выполнения задач А и Б, то эти задачи не могут выполняться параллельно, даже если между ними нет концептуальной зависимости. Кроме того, руководитель проекта должен знать задачи, лежащие на критическом пути. Для того чтобы весь проект был выполнен в срок, необходимо выполнять в срок все критические задачи.

Календарный план помимо распределения задач и ресурсов по времени должен предусматривать процедуры контроля промежуточных результатов. Такие процедуры обычно называют вехами. Очень важно, чтобы вехи были расставлены через регулярные интервалы вдоль всего процесса разработки программного обеспечения. Кроме того, желательно чтобы вехи совпадали со сроком выполнения критических задач. Это даст руководителю возможность не только регулярно получать информацию о текущем положении дел, но и объективно оценивать рис-

ки срыва сроков выполнения проекта, принимать оперативные решения по снижению этих рисков.

В первую очередь определите основные фазы выполнения проекта. В **основу** может быть положена принятая модель жизненного цикла процесса разработки программного обеспечения. Например, при использовании каскадной модели основными фазами будут являться **анализ, проектирование, реализация, тестирование, внедрение**. Далее определите состав работ внутри каждой фазы, в соответствии с сутью разработки. Таким образом будет определен состав работ по проекту. Каждая фаза должна заканчиваться вехой – специальной единицей работы, подразумевающей контроль выполнения работ по проекту и достижение некоторого промежуточного или окончательного результата.

Далее определите длительность каждой работы, входящей в план работ. Для определения длительности могут быть использованы различные регрессионные модели (например, COCOMO II), или же может применяться прямой метод оценки.

Следующим этапом является определение связей между задачами. В большинстве средств планирования (например, в MS Project), существует четыре вида связей:

Связь типа окончание – начало означает, что задача Б не может начаться раньше окончания задачи А (например, если в процессе выполнения задачи Б используются результаты, получаемые при решении задачи А).

Связь типа начало – начало означает, что задача Б не может начаться до тех пор, пока не началось выполнение задачи А. Например, тестирование программного компонента не может начинаться до того, как была начата его разработка, но, в то же время, для написания тестов не обязательно дожидаться окончания разработки этого компонента.

Связь типа окончание – окончание означает, что работа Б не может окончиться до тех пор, пока не завершится выполнение работы А. Например, проектирование базы данных не может быть закончено до того, как будет завершено семантическое моделирование предметной области.

Связь окончание – начало означает, что задача Б не может закончиться до того, как началась задача А. Обычно такая связь

используется в том случае, когда А является задачей с фиксированной датой начала, которую нельзя изменить.

После того, как определен состав работ, нужно определить, кто эти задачи будет выполнять и какое оборудование должно использоваться.

Сформируйте список ресурсов, для каждого ресурса определите название и стоимость его использования. Далее назначьте ресурсы на выполнение конкретных задач. При первом назначении ресурса будут автоматически рассчитаны трудозатраты. В тех случаях, когда необходимо ускорить выполнение тех или иных задач, на них могут быть назначены дополнительные ресурсы.

После распределения ресурсов необходимо выполнить выравнивание их нагрузки. В тех случаях, когда на параллельно выполняемые задачи назначается один и тот же ресурс, нагрузка на него может превысить максимально допустимую. Для выравнивания нагрузки установите дополнительные связи между задачами таким образом, чтобы задачи, использующие один и тот же ресурс, выполнялись последовательно.

Порядок выполнения работы.

1. Ознакомьтесь с техническим заданием
2. Выберите модель жизненного цикла процесса разработки и внедрения ПО, которая, по вашему мнению, в наибольшей степени соответствует рассматриваемой ситуации.
3. Выделите основные этапы работ.
4. Выделите основные задачи внутри отдельных этапов работ.
5. Определите зависимости между задачами.
6. Определите порядок выполнения отдельных задач.
7. Назначьте исполнителей на решаемые задачи.
8. Сбалансируйте нагрузку исполнителей.

2. Лабораторная работа № 2. Оценка трудозатрат на выполнение работ по разработке и внедрению программного обеспечения

Количество аудиторных часов: 8.

Цели занятия: получить первоначальные навыки применения инженерных подходов к процессам оценки трудоемкости выполнения работ по разработке и внедрению автоматизированных информационных систем, получить первоначальные навыки расчета стоимости выполнения работ по разработке и внедрению автоматизированных информационных систем.

Цель работы: расчет трудоемкости и стоимости выполнения работ по проекту, для которого был составлен календарный план в процессе выполнения лабораторной работы № 1. Расчет трудоемкости следует проводить методов функциональных точек.

Средства реализации: ограничений не накладывается, рекомендуется использование All Fusion Process Modeler, All Fusion Data Modeler, Microsoft Excel, Microsoft Project.

Функционально-ориентированные метрики косвенно измеряют программный продукт и процесс его разработки. Вместо подсчета LOC-оценки при этом рассматривается не размер, а функциональность или полезность продукта. В качестве количественной характеристики применяется понятие количества функциональных точек FP (function points) [7].

Для получения функционально-ориентированных метрик (FP-метрик) используются функциональные и концептуальные модели будущей системы (например, модели IDEF0-IDEF1X). Основывается процесс получения функционально-ориентированных метрик на функциональной модели системы (модели бизнес-процессов). Рассматриваются только наиболее значимые процессы, соответствующие основным функциям разрабатываемого программного продукта (например, перечисленным в техническом задании). Каждый бизнес-процесс имеет входные и выходные данные, находящие свое отражение в концептуальной модели системы (например, соответствующие сущностям в моделях «сущность-связь»).

Для расчета количества функциональных точек используется пять информационных характеристик:

1) количество внешних вводов;
2) количество внешних выводов;
3) количество внешних запросов;
4) количество внутренних логических файлов;
5) количество внешних интерфейсных файлов.

Для каждого бизнес-процесса каждой выделенной характеристике ставится в соответствие сложность. Для этого характеристике назначается низкий, средний или высокий ранг, а затем формируется числовая оценка ранга. Для внешних вводов, выводов и запросов ранжирование основано на количестве ссылок на файлы и количестве элементов данных a_{ij}, где i — номер строки, соответствующей количеству внешних вводов, выводов или запросов; j — номер, соответствующий количеству элементов данных.

Для каждого бизнес-процесса, реализуемого в разрабатываемой информационной системе, строится таблица расчета количества функциональных точек (см. таблицу 2.1).

Таблица 2.1

Таблица расчета количества функциональных точек

Наименование функции	Количество функциональных точек, соответствующее категории функции			Количество функциональных точек
	Простая	Средняя	Сложная	
1. Определение количества выводов	$\alpha_{11} \times x_{11}$	$\alpha_{12} \times x_{12}$	$\alpha_{13} \times x_{13}$	X_1
2. Определение количества вводов	$\alpha_{21} \times x_{21}$	$\alpha_{22} \times x_{22}$	$\alpha_{23} \times x_{23}$	X_2
3. Определение количества опросов вывода	$\alpha_{31} \times x_{31}$	$\alpha_{32} \times x_{32}$	$\alpha_{33} \times x_{33}$	X_3
4. Определение количества опросов ввода	$\alpha_{41} \times x_{41}$	$\alpha_{42} \times x_{42}$	$\alpha_{43} \times x_{43}$	X_4

5. Определение количества файлов	$\alpha_{51} \times x_{51}$	$\alpha_{52} \times x_{52}$	$\alpha_{53} \times x_{53}$	X_5
6. Определение количества интерфейсов	$\alpha_{61} \times x_{61}$	$\alpha_{62} \times x_{62}$	$\alpha_{63} \times x_{63}$	X_6
Общее количество функциональных точек				$\sum_{i=1}^{6} X_i$

Определение количества выводов

Под выводами при расчете FP-оценок следует понимать единицы деловой информации, получаемые на выходе бизнес-процесса. Как правило, выводами являются:

- структуры данных, создаваемых в рассматриваемом бизнес-процессе для передачи в другие бизнес-процессы или за пределы создаваемой информационной системы;
- структуры данных, создаваемых в рассматриваемом бизнес-процессе и предназначенные для вывода конечным пользователям в виде экранных форм или печатных документов.

При использовании для получения FP-метрик моделей Idef0 и Idef1x выводы можно определять на основе стрелок, исходящих из рассматриваемого процесса модели Idef0 и соответствующих им сущностей модели Idef1x.

Каждый из выводов, в соответствии с количеством формируемых структур данных и количеством элементов данных в каждой из структур, следует отнести к одной из категорий сложности: простой, средний, сложный. В таблице 2.2 приведены весовые коэффициенты сложности выводов.

Определение количества вводов

Под вводами при расчете FP-оценок следует понимать единицы деловой информации, поступающие на вход бизнес-процесса. Как правило, вводами являются:

- структуры данных, получаемые из других бизнес-процессов или из-за пределов разрабатываемой информационной системы;
- структуры данных, вводимых конечным пользователем.

Таблица 2.2

Весовые коэффициенты сложности выводов

Количество структур данных	Значение коэффициента α в зависимости от количества элементов данных		
	от 1 до 5, α_{11}	от 6 до 19, α_{12}	20 и более, α_{13}
1	4	4	5
2–3	4	5	7
4 и более	5	7	7

При использовании для получения FP-метрик моделей Idef0 и Idef1x вводы можно определять на основе стрелок, входящих в рассматриваемый процесс модели Idef0 и соответствующих им сущностей модели Idef1x.

Вводы, так же как и выводы, следует разбить по категориям: простые, средние, сложные. В таблице 2.3 приведены весовые коэффициенты сложности вводов.

Таблица 2.3

Весовые коэффициенты сложности вводов

Количество структур данных	Значение коэффициента α в зависимости от количества элементов данных		
	от 1 до 5, α_{11}	от 6 до 19, α_{12}	20 и более, α_{13}
1	4	4	5
2–3	4	5	7
4 и более	5	7	7

Определение количества запросов

Под запросами при расчете FP-оценок следует понимать диалоговый ввод, который немедленно приводит к немедленному программному ответу в виде диалогового вывода. Основным отличием запроса от пары ввод-вывод является отсутствие вычислений либо каких-то других сложных действия в рамках этой процедуры взаимодействия. Например, ввод данных о клиенте через диалоговую форму с последующим сохранением данных в БД является вводом, вывод на экран отчета по активности клиентов за последние три месяца является выводом, а поиск клиента по наименованию является запросом.

В таблице 2.4 приведены весовые коэффициенты сложности запросов.

Таблица 2.4

Весовые коэффициенты сложности запросов

Количество структур данных	Значение коэффициента α в зависимости от количества элементов данных		
	от 1 до 5, α_{11}	от 6 до 19, α_{12}	20 и более, α_{13}
1	4	4	5
2–3	4	5	7
4 и более	5	7	7

Определение количества внутренних структур данных

Под структурами данных при расчете FP-оценок будем понимать структурированные единицы информации, используемые программной системой в рассматриваемом бизнес-процессе. В основном это структуры данных, представляющие собой первичную логическую группу пользовательских данных, которые находятся внутри границ программной системы.

При использовании для получения FP-метрик моделей Idef0 и Idef1x структуры можно определять на основе стрелок, входящих в рассматриваемый процесс модели Idef0 и соответствующих им сущностей модели Idef1x. В этом случае следует учитывать каждую сущность, связанную с рассматриваемым бизнес-процессом или любом из его подпроцесов.

В таблице 2.5. приведены весовые коэффициенты сложности структур данных.

Таблица 2.5

Весовые коэффициенты сложности структур данных

Количество логических взаимосвязей	Значение коэффициента α в зависимости от количества элементов данных		
	от 1 до 5, α_{11}	от 6 до 19, α_{12}	20 и более, α_{13}
1	7	7	7
2–5	7	10	10
6 и более	10	15	15

Определение количества внешних интерфейсов

Под интерфейсами следует понимать структуры данных, получаемых из внешних программных систем и структуры данных, передаваемые во внешние программные системы. В табл. 2.6. приведены весовые коэффициенты сложности интерфейсов.

Таблица 2.6

Весовые коэффициенты сложности интерфейсов

Количество логических взаимосвязей	Значение коэффициента α в зависимости от количества элементов данных		
	от 1 до 5, α_{11}	от 6 до 19, α_{12}	20 и более, α_{13}
1	5	5	7
2–5	7	7	10
6 и более	7	10	10

Общее количество функциональных точек (FP-оценок) рассчитывается по формуле $FP = X \times (0{,}65 + 0{,}01 \times \sum_{i=1}^{14} F_i)$, где X— суммарное количество функциональных точек для каждого бизнес-процесса; F_i — коэффициенты регулировки сложности.

Таблица 2.7.

Факторы, влияющие на сложность программного продукта

Наименование фактора	Показатель оценки фактора
1. Передача данных	Количество средств связи, необходимых для передачи или обмена информацией с приложением или системой
2. Распределенная обработка данных	Количество и сложность вычислительных процессов, требующих распределенной обработки данных
3. Производительность	Потребность пользователя в фиксации времени ответа или производительности
4. Распространенность используемой конфигурации	Уровень распространенности аппаратно-программной платформы, на которой будет выполняться приложение
5. Частота транзакций	Частота выполнения транзакций (ежеминутно, ежедневно, ежемесячно)
6. Оперативный ввод данных	Доля информации, которую необходимо вводить в режиме реального времени

7. Эффективность работы конечного пользователя	Уровень повышения производительности работы пользователя, использующего приложение
8. Оперативное обновление	Количество внутренних файлов, обновляемых в онлайновой транзакции
9. Сложность обработки	Способность приложения выполнять интенсивную логическую или математическую обработку данных
10. Повторная используемость	Возможность использования отдельных компонентов программного продукта в будущих проектах
11. Простота инсталляции	Трудность преобразования и инсталляции приложений
12. Простота эксплуатации	Эффективность и/или уровень автоматизации процедур запуска, резервирования и восстановления
13. Разнообразные условия размещения	Возможность применения программного продукта в разных местах, разных организациях
14. Простота изменений	Наличие в приложении процедуры поддержки внесения изменений с максимальной простотой

После получения FP-оценок можно приступать к расчету трудоемкости отдельных задач в рамках проекта по разработке программной системы. Обычно трудоемкость измеряют в количестве часов, затрачиваемых специалистом на решение той или иной задачи. Для получения трудоемкости на основе оценок следует определить нормативную производительность труда специалиста. Можно использовать производительность труда выраженную в количестве функциональных точек, реализуемых специалистом за час работы. Но чаще производительность труда программиста выражается в количестве строк кода, которые он должен написать за час работы. Для перевода FP-оценок в LOC-оценки можно воспользоваться таблицей 2.8.

Таблица 2.8

Пересчет FP-оценок в LOC-оценки

Язык программирования	Количество LOC на один FP
Ассемблер	320
C	128
Кобол	106

Фортран	106
Паскаль	90
C++	64
Java	53
Ada 95	49
Visual Basic	32
Visual C++	34
Delphi Pascal	29
Smalltalk	22
Perl	21
HTML3	15
LISP	64
Prolog	64
Miranda	40
Haskell	38

Порядок выполнения работы

1. Ознакомьтесь с техническим заданием.
2. Определите основные процессы, подлежащие автоматизации, постройте соответствующие диаграммы Idef0.
3. Определите основные элементы данных, используемые в автоматизируемых процессах, постройте соответствующие диаграммы Idef1x.
4. Рассчитайте количество функциональных точек для каждого из автоматизируемых процессов.
5. Рассчитайте общее количество функциональных точек.
6. Проставьте полученные оценки трудозатрат в календарный план, полученный в результате выполнения лабораторной работы № 1.
7. Составьте смету затрат на выполнение работ по проекту в соответствии с вышеприведенной методикой.

3. Лабораторная работа № 3. Автоматизация управления проектом по разработке и внедрению автоматизированной информационной системы

Количество аудиторных часов: 8.

Цели занятия: получение первоначальных навыков использования Issue Tracking System для управления проектами по разработке и внедрению автоматизированных информационных систем.

Цель работы: Сформировать в Issue Tracking System план работ, имитируя выполнение отдельных задач, выполнить мониторинг работ, сформировать отчет о ходе выполнения проекта.

Средства реализации: ограничений не накладывается, рекомендуется использование TrackStudio.

Система управления проектами (СУП) - набор инструментов, методов, методологий, ресурсов и процедур, используемых для управления проектом. Она может быть как формальной, так и неформальной и помогает менеджеру проекта эффективно завершить проект. Система управления проектами - это ряд процессов и связанных с ними функций контроля, объединенных в единую целенаправленную структуру.

Система управления проектами строится на основе плана управления проектом, который описывает то, как будет использоваться система.

Содержание системы управления проектом изменяется в зависимости от области приложения, особенностей организации, сложности проекта и доступности необходимых ресурсов. Система строится так, чтобы максимально соответствовать стратегическим целям и производственным ресурсам клиентской организации.

При разработке программного обеспечения как большие, так и маленькие софтверные компании используют системы учета задач, ошибок, управления проектами (bug tracker, issue tracking system, project management application).

На данный момент такого рода продуктов существует немало. Есть простые системы, функционал которых ограничивается учетом ошибок и отслеживанием их статуса. Есть более

сложные, которые позволяют, например, строить различные графики по проектным рискам, интегрироваться с системами версионного контроля, осуществлять сложный поиск по проектной документации и так далее.

В идеале система управления проектами - это некоторое серверное приложение, которое позволяет делать следующее:

• в реальном времени отслеживать текущее состояние проектов, собирать статистику по проектам;

• вести учет ошибок, заданий, улучшений в соответствии с заданным жизненным циклом;

• хранить проектную документацию (например, иметь встроенную Wiki);

• конфигурировать права доступа пользователей, их роли, отправку нотификаций;

• интегрироваться с разными third-party продуктами (например, с системами контроля версий);

• получать доступ к функциям системы программным способом (через соответствующий API);

В качестве примера Issue Tracking System рассмотрим TrackStudio 3.5.

Пользовательский интерфейс

1. Навигационное дерево

Пользовательский интерфейс TrackStudio оптимизирован для эффективной работы с большим количеством задач и пользователей. Чтобы переместиться быстро для выполнения задачи или пользователя нужно использовать навигационное дерево в левой части. В дереве представлены задачи и их подзадачи, пользователи и подчиненные пользователей.

2. Главное меню

Главное меню управления проектами и задачами. Поле в правой части главного меню служит для поиска задач или пользователей.

3. Полный путь

Путь к текущей задаче или пользователю, чуть ниже, некоторые сведения, касающиеся текущего пользователя или задачи, в зависимости от режима работы.

4. Закладки

Страницы TrackStudio различной функциональности сгруппированы на закладках интерфейса. Вы можете управлять закладками, которые доступны для каждого из пользователей.

5. Списки объектов

Большинство объектов, связанных с задачей или пользователем, отображаются в TrackStudio в виде списков. • Чтобы просмотреть или изменить свойства объекта, нажмите на название объекта. Чтобы удалить объект, установите флажок рядом с ним и нажмите кнопку "Удалить".

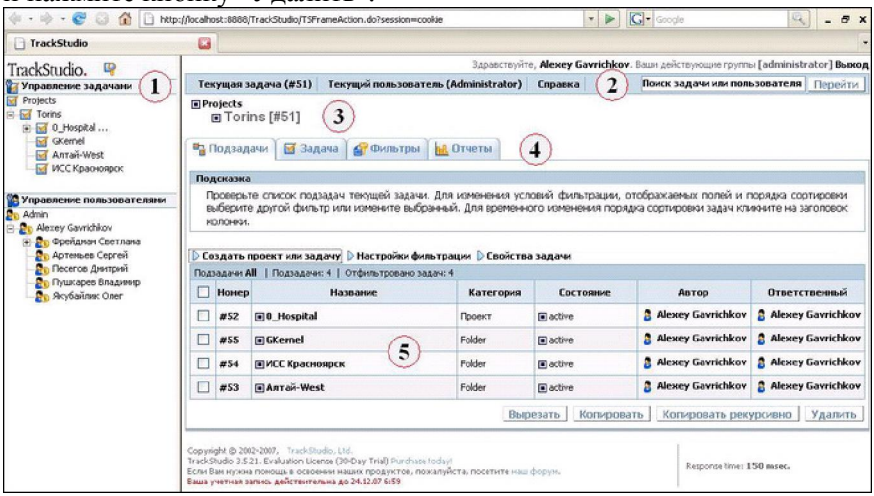

Рис. 3.1. Общий вид интерфейса TrackStudio

Рассмотрим основные понятия, используемые в TrackStudio

Процессы

Процессы используются для определения жизненного цикла задач в TrackStudio. Они позволяют настраивать права пользователей на изменение состояния задачи и содержат набор свойств: приоритеты, типы пользовательских сообщений, переходы между состояниями и настраиваемые поля. Для назначения процесса для задачи следует использовать категорию задачи.

Сообщения

Сообщения используются для изменения состояния задачи, назначения ответственного, учета времени и добавления

комментариев. В TrackStudio все эти операции можно сделать одним действием – добавлением сообщения. На добавление сообщений разных типов у пользователя или группы пользователей должны быть соответствующие разрешения.

Используйте сообщения для:

• изменения состояния задачи: с помощью сообщений вы можете перевести задачу в другое состояние. Для этого выберите Тип сообщения, который переводит задачу из текущего состояния в требуемое. Состояния и переходы между ними зависят от процесса задачи, который определяется категорий задачи.

• организации взаимодействия пользователей: поле задачи «Ответственный» определяет пользователя, реакция которого необходима для дальнейшей обработки задачи. Если в ходе работы над задачей вам требуется участие другого пользователя (например Вы написали код, который должен протестировать тестер или задали вопрос другому разработчику), то укажите в поле «Ответственный» требуемого пользователя.

• отслеживания хода работ по задаче: в ходе работы над задачей вы можете добавлять сообщения, описывающие текущее состояние дел, и указывать сколько было потрачено времени.

Категории

Категории определяют типы задач в TrackStudio, а также зависимости между ними. При создании категории необходимо выбрать процесс, который будет отражать ее поведение. Также следует указать список допустимых подкатегорий для нее и список группы пользователей, которые могут создавать, просматривать и удалять задачи данной категории. Иными словами, категории, в отличие от процессов, задают не жизненный цикл задач, а их ограничения и права пользователей на их использование.

Фильтры

Фильтры используются для поиска задач и пользователей, соответствующих определенным критериям. Также как и другие объекты в TrackStudio, фильтры наследуются от верхних уровней иерархии к нижним, что позволяет сэкономить время при конфигурации проектов. Фильтры предназначены для поиска задач по заданным критериям. Используйте фильтры для:

• поиска подзадач текущей задачи по заданным критериям (закладка «Подзадачи»)

• периодического получения списка подзадач по e-mail (пункт меню Текущая Задача -> Правила подписки на фильтры...). Если вы установите правило подписки на фильтр, то TrackStudio будет периодически отсылать вам информацию о задачах, подходящих под условия фильтрации.

• определения правила рассылки оповещений. Используйте пункт меню Текущая задача -> Правила оповещений по e-mail для создания правила оповещения. Если Вы создадите правило оповещения для фильтра, то при каждом изменении в задаче, удовлетворяющей условиям фильтра, Вам будут посылаться уведомления.

• генерация отчетов по задачам которые найдены фильтром.

Оповещения по электронной почте

Создайте правило оповещения для получения e-mail при изменении текущей задачи или ее подзадач, удовлетворяющих условиям фильтрации. Внешний вид e-mail сообщений определяется шаблонами. Создавать и редактировать шаблоны можно с помощью пункта меню Текущий пользователь -> Шаблоны e-mail....

Подписка на фильтры

Используйте подписку на фильтры, чтобы периодически сообщать пользователям о состоянии интересующих их задач. Например, заведующий может в начале рабочего дня получать по e-mail список книг с истекшим сроком возврата, а в конце рабочего дня – со списком отданных и возвращенных книг в его отделе.

Скрипты

Скрипты используются для вычисления настраиваемых полей, задания триггеров (триггер – это специализированный скрипт, который автоматически срабатывает до, вместо или после изменения задачи) и определения правил импорта из CSV файлов. Для написания скриптов используется Java-подобный язык. Он позволяет решать задачи разной сложности: от простых математических операций с полями задач до реализации

полноценных алгоритмов обработки данных с использованием циклов и ветвлений.

Конфигурирование проектов в TrackStudio

Настройка процессов и создание категорий задач

Выберите в дереве проектов родительский проект. В главном меню откройте пункт Текущая задача → Процессы…, щелкните по ссылке «Создать процесс» и укажите название процесса. После нажатия кнопки «Сохранить» появится страница со вкладками настройки процесса. Здесь можно задать состояния процесса, типы сообщений и настраиваемые поля.

Создайте состояния в которых может находится задача. Примеры состояний задачи: new (новая), resolved (решена), closed (закрыта). Среди созданных состояний задачи укажите начальное состояние. При создании задачи система автоматически переводит задачу в начальное состояние. Затем укажите конечные состояния задачи. Если задача будет переведена в конечное состояние, то система автоматически установит для нее дату закрытия. Вы можете указать несколько конечных состояний или не указывать их вообще. В дальнейшем начальное состояние будет отмечаться знаком ◦ внутри цветного квадрата, а конечные состояния - знаком x.

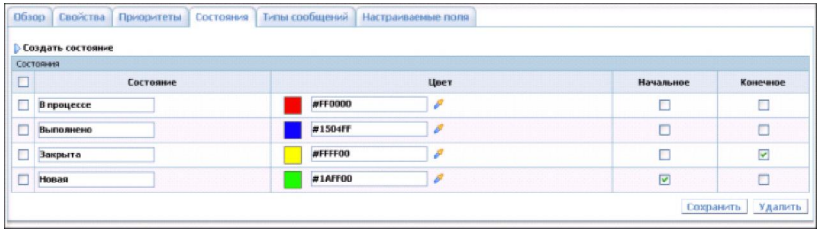

Рис. 3.2. Настройка состояний задач в TrackStudio

Для перевода задач из одного состояния в другое пользователь должен создать сообщение и указать тип сообщения. Тип сообщения определяет переход, условия перехода и конечное состояние задачи.

Для создания категорий выберите пункт меню Текущая задача → Категории…, затем перейдите по ссылке «Создать категорию». Отметьте галочкой, если для данной категории задач

необходим ответственный и разрешено задавать группу в качестве ответственного. После нажатия кнопки «Сохранить» появится страница, на вкладках которой можно редактировать свойства, отношения, триггеры, шаблоны и разрешения для только что созданной категории. Разрешения на создание, просмотр, редактирование и удаление задач данной категории задаются с помощью матрицы.

Вы создали новые категории, но они не появились в выпадающем списке категорий при создании задачи. Это возможно, если вы пропустили важный шаг после создания категории. Добавьте эту новую категорию в качестве возможной подкатегории в другие. Для этого выберете «родительскую» категорию, т.е. ту, в задачах которой можно будет создавать подзадачи с новой категорией. Перейдите на закладку «Отношения» и нажмите «Добавить подкатегорию». В появившемся списке выберите созданную Вами категорию и нажмите кнопку «Добавить подкатегорию». Аналогично добавьте Вашу категорию в другие нужные.

Создание проекта или задачи

Для создания проекта:

1. Выберите родительскую задачу вашего проекта.
2. Перейдите на закладку «Подзадачи».
3. Разверните панель «Создать проект или задачу».
4. Выберите категорию для этого проекта.
5. Введите название проекта.
6. Нажмите «Создать проект или задачу».
7. Заполните свойства проекта.
8. Нажмите кнопку «Сохранить».

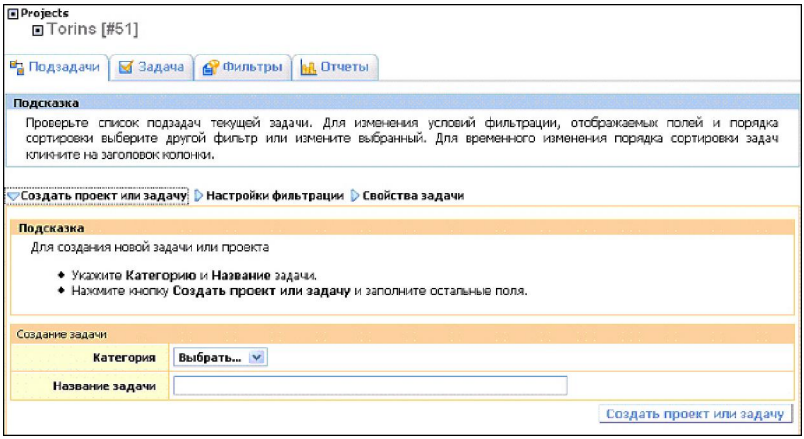

Рис. 3.3. Создание проекта в TrackStudio

Условия

Вы не можете создать проект или задачу, если у вас нет разрешения для создания подзадач для текущего задачи. Для уточнения вашего разрешения, зайдите в пункт меню Текущая задача – Правила доступа....

Правила доступа к задаче

Что бы организовать доступ:

1. Нажмите пункт меню Текущий пользователь → Правила доступа ...
2. Выберете закладку «Назначенные группы».
3. Создайте правило для пользователя или группы пользователей.
4. Нажмите кнопку «Сохранить».

Редактирование задачи

Чтобы изменить свойства задачи:

1. Выберите задачу, которую вы хотите изменить.
2. Выберите закладку «Задача».
3. Нажмите ссылку «Редактировать».
4. Отредактируйте задачу.
5. Нажмите кнопку «Сохранить».

Изменение состояния задачи

Чтобы изменить состояние задачи:

1. Выберите закладку «Задача».
2. Разверните панель «Создать сообщение».

3. Выберите тип сообщения, который переведет вашу задачу до требуемого состояния.

4. Нажмите кнопку «Сохранить».

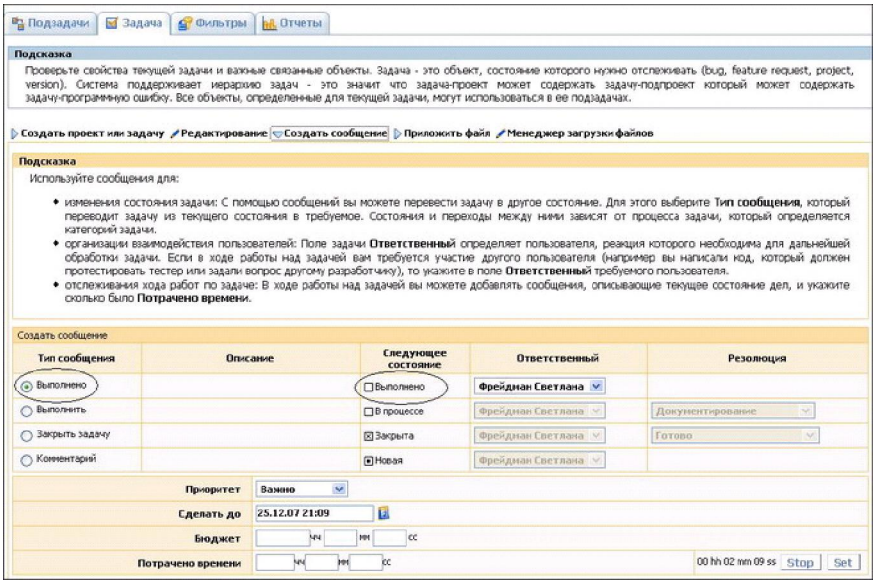

Рис. 3.4. Создание сообщения в TrackStudio

Условия

Вы не сможете изменить состояние задачи, если у Вас нет разрешения на добавление сообщений в задачу или не прописаны переходы в процессе, выбранного для этой задачи.

Добавление описания к задаче

Чтобы добавлять комментарии к задаче:

1. Выберите закладку «Задача».
2. Разверните панель «Создать сообщение».
3. Выберите тип сообщения и введите описание к задаче.
4. Нажмите кнопку «Сохранить».

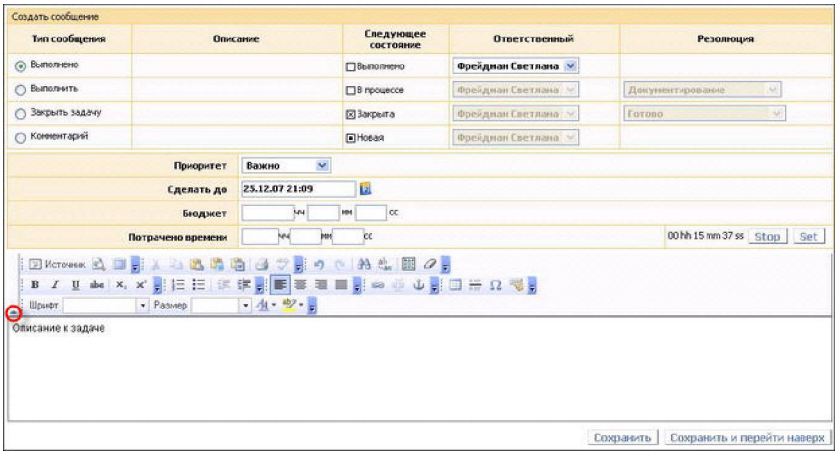

Рис. 3.5. Добавление описания к задаче в TrackStudio

Условия

Вы не сможете добавить описание к задаче, если у Вас нет разрешения на добавление сообщений в задачу.

Примечание

Для использования сложного форматирования нажмите на треугольник в верхнем левом углу.

Вложение файлов в задачу

Для вложения файла в задачу:

1. Выберите закладку «Задача».
2. Разверните панель «Приложить файл».
3. Укажите название приложенного файла.
4. Введите описание.
5. Нажмите кнопку «Приложить файл».

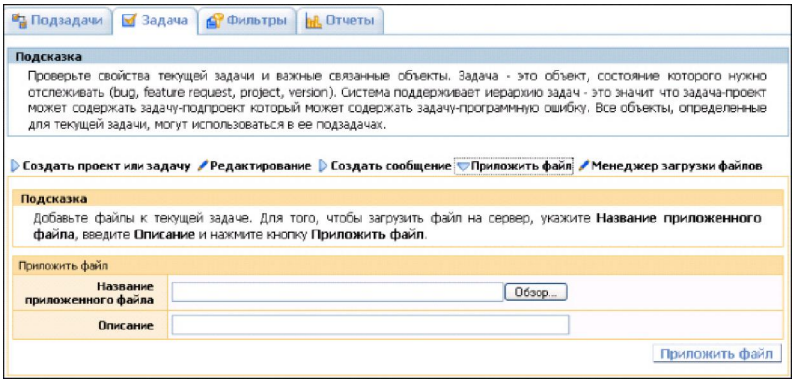

Рис. 3.6. Прикрепление файла к задаче в TrackStudio

Для вложения снимка экрана в задачу (Microsoft Windows):

1. Выберите закладку «Задача».
2. Нажмите «Менеджер загрузки файлов».
3. Скопируйте экран в буфер обмена с помощью Alt - PrtScr.
4. Нажмите кнопку «Вставить изображение».
5. Нажмите кнопку «Загрузить и обновить».

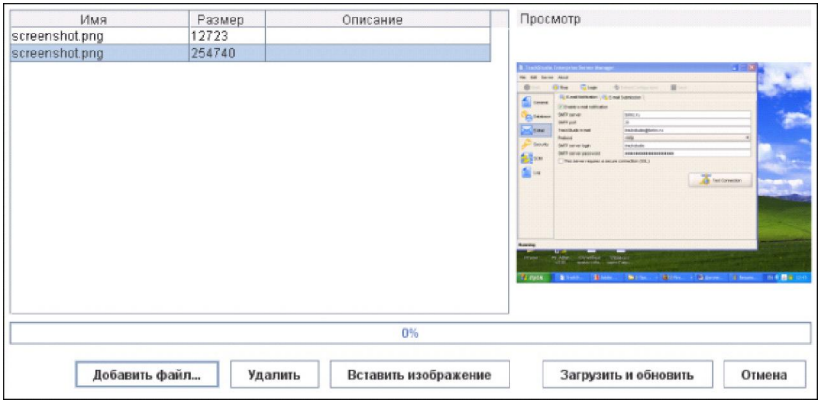

Рис. 3.7. Вложение снимка экрана в задачу в TrackStudio

Условия

Вы не можете приложить файл с задачей, если:

• У Вас нет разрешения для загрузки файлов с этой задачей.

• Файл превышает максимальный размер загрузки, определенный администратором TrackStudio.

Перемещение задачи

Чтобы переместить задачи:

1. Выберите задачу, которую нужно переместить.
2. Нажмите Текущая задача → Вырезать задачу.
3. Выберете задачу, которая будет новым родителем.
4. Нажмите Текущая задача → Вставить.

Создание фильтров

В качестве примера рассмотрим процедуру создания фильтра, позволяющего отображать все задачи, обновленные за последние сутки. Для того чтобы фильтр был доступен всем подразделениям, необходимо выбрать родительский проект в дереве «Управление задачами». Затем перейдите на вкладку «Фильтры» и щелкните мышью по ссылке «Создать фильтр». На появившейся странице заполните основные свойства фильтра. В качестве типа фильтра выберите normal, тогда фильтр можно будет использовать для фильтрации задач, рассылки оповещений и для генерации отчетов. Снимите галочку «Закрытый», чтобы дать возможность другим пользователям пользоваться этим фильтром. Также отметьте галочкой «Глубокий поиск», который позволит искать задачи по всей иерархии проектов. После нажатия на кнопку «Сохранить» появляется окно со вкладками настройки всех параметров фильтра. Для того чтобы отфильтровать все задачи с изменениями за последние сутки, необходимо в поле «Дата обновления» выбрать соответственно «24 часов назад или позже».

Генерация отчета

Прежде чем начать создание отчета надо создать соответствующий фильтр для отчета. Для генерации отчета:

1. Нажмите закладку Задача → Отчеты.
2. Выберите существующий отчет или создайте новый.
3. Заполните параметры отчета.
4. Нажмите кнопку «Создать отчет».

Импорт задач из CSV файла

Для импорта задач из CSV файла вначале необходимо создать скрипт импорта. Выбирите тип скрипта – CSV Import. После задания общих свойств скрипта можно приступить к редактированию его кода.

Теперь выберите в дереве «Управление задачами» проект, в который мы будем импортировать журналы из CSV файла. Затем в главном меню перейдите по ссылке Текущая задача → CSV импорт… В появившемся окне выберите ранее созданный скрипт «CSV импорт» и файл с перечнем журналов. Обратите внимание на поле с кодировкой – она должна совпадать с кодировкой импортируемого файла. После выполнения импорта отображается количество импортированных строк и наличие ошибок в них.

Аналогичным образом в TrackStudio можно импортировать пользователей и их сообщения.

Удаление задачи

Чтобы удалить задачу:

1. Выберите задачу, которую вы хотите удалить.
2. Нажмите кнопку «Удалить».

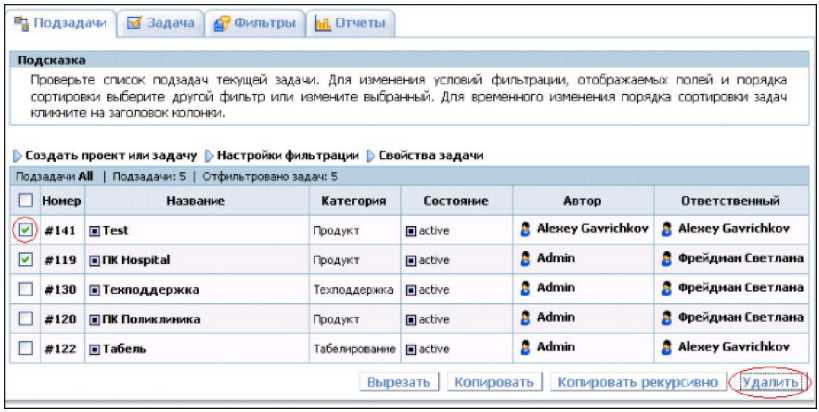

Рис. 3.8. Удаление задачи в TrackStudio

Создание учетной записи пользователя

Для создания учетной записи пользователя:

1. Выберите пункт меню Текущий пользователь → Пользователь ...

2. Выберете группу, заполните логин и имя пользователя.

3. Нажмите кнопку «Создать пользователя».

4. Заполните остальные поля.

5. Нажмите кнопку «Сохранить».

6. Нажмите кнопку «Изменить пароль».

7. Введите новый пароль дважды.

8. Нажмите кнопку «Установить пароль».

9. Укажите задачи и проекты, к которым пользователь будет иметь доступ, используя пункт меню Текущая задача – Правила доступа ...

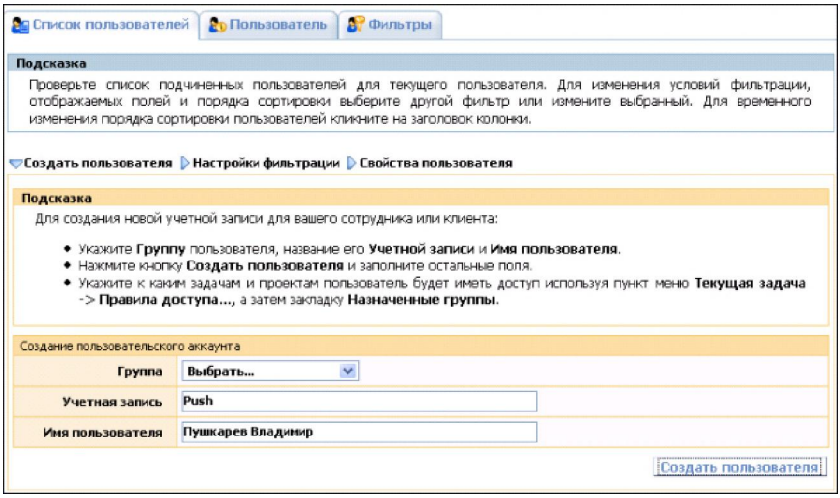

Рис. 3.9. Создание пользователя в TrackStudio

Редактирование учетной записи пользователя

Чтобы изменить свойства учетной записи пользователя:

1. Выберите пункт меню Текущий пользователь → Пользователь ...

2. Нажмите ссылку «Изменить».

3. Введите свойств учетной записи пользователя.

4. Нажмите кнопку «Сохранить».

Задание параметров саморегистрации пользователей

Иногда очень удобно предоставить некоторым группам пользователей право самим регистрироваться в TrackStudio без участия администратора. В этом случае при первом входе в систему появляется окно с формой регистрации, где пользователь может указать свой логин, имя пользователя, адрес электронной почты и другие параметры. После нажатия на кнопку «Register» можно войти в систему под созданным логином и полученным по электронной почте паролем.

Для задания правил саморегистрации читателей зайдите в TrackStudio под учетной записью администратора TrackStudio. Выберите пункт меню Текущий пользователь → Правила саморегистрации… Затем последовательно укажите Название правила, Группу и Задачу (здесь надо указать номер задачи). После нажатия кнопки «Сохранить» будет отображен URL для саморегистрации. При переходе по этой ссылке появляется окно для саморегистрации. Необходимо указать действующий адрес электронной почты, потому что на него будет отослан пароль.

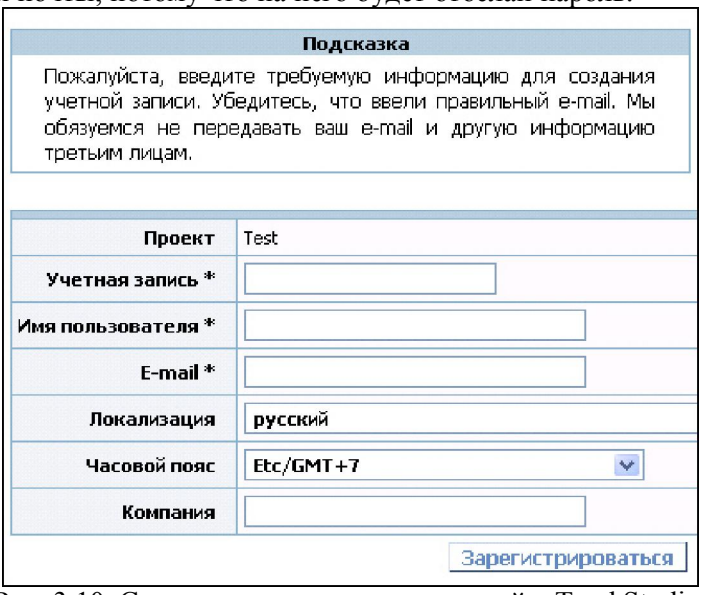

Рис. 3.10. Саморегистрация пользователей в TrackStudio.

Правила оповещений по e-mail

Для отправки оповещений о событиях, происходящих во всем проекте, выберите в дереве «Управление задачами» проект «Библиотека». Затем в главном меню выберите «Текущая задача» → Правила оповещений по e-mail… Щелкните мышью по ссылке «Создать правило оповещения по e-mail» и в поле «Создать» выберите группу «библиотекарь». На открывшейся странице поставьте галочку только для поля «Реагировать на создание задачи».

Примечание

Для того что бы оповещение приходило только автору задания нужно в условиях фильтрации указать Submitter = current user.

Пример

Для отправки оповещений об изменении состояния задач за последние сутки необходимо выбрать Текущая задача → Правила подписки на фильтры… На странице со свойствами правила оповещения выберите фильтр «Все сообщения за последние сутки», поставьте интервал – «каждый день». Не забудьте проверить время действия фильтра.

Правила импорта e-mail

Настройте правила, по которым письма, поступившие в специальный почтовый ящик (указывается администратором при инсталляции системы) будут импортироваться как подзадачи текущей задачи. Если вы хотите импортировать задачи в несколько проектов - создайте для каждого проекта правило импорта задач и укажите имя проекта в качестве ключевого слова. Ответственным в новых задачах будет автоматически назначен ответственный за текущую задачу (на момент обработки e-mail). Если у вас есть права на загрузку файлов, то вы можете загрузить файлы прикрепив их к письму.

Что бы создать правила импорта e-mail

1. Выберите пункт меню Текущий пользователь → Правила импорта e-mail ...
2. Нажмите ссылку «Создать правило импорта e-mail».
3. Заполните свойства.
4. Нажмите кнопку «Сохранить».

Создание сообщений с помощью e-mail

Добавление сообщений, используя обычный текстовый e-mail:

1. Создайте новое сообщение в вашем e-mail клиенте.
2. Введите число задачи (например, # 23).
3. Введите описание сообщения.
4. Дополнительно: для изменения состояния задачи, введите имя и адрес электронной почты нового ответственного в CC поле.
5. Отправьте на электронную почту TrackStudio.

Условия

• Пользователь должен быть зарегистрирован в TrackStudio с тем же именем или электронной почтой.

• Пользователь, который посылает по электронной почте сообщения, должен иметь разрешения на добавление сообщений к указанной задаче.

Порядок выполнения лабораторной работы

1. Создайте новый проект в системе управления проектами.
2. Зарегистрируйте всех участников проекта.
3. Внесите в проект все задачи, приведенные в календарном плане, полученном в ходе выполнения лабораторной работы № 1.
4. Назначьте задачи исполнителям от лица менеджера проекта.
5. Внесите от лица участников - кодеров несколько отчетов о ходе выполнения задач.
6. Внесите от лица участников проекта – тестеров несколько отчетов о найденных ошибках.
7. Внесите от лица участников проекта - кодеров отчеты об исправлении ошибок, обнаруженных тестерами.
8. Сформируйте отчет о ходе выполнения проекта.

4. Лабораторная работа № 4. Использование систем контроля версий исходного кода программ

Количество аудиторных часов: 4.

Цели занятия: получение первоначальных навыков использования систем контроля версий исходного кода программ, получение первоначальных навыков организации коллективной разработки программного обеспечения.

Цель работы: создать в системе контроля версий репозиторий для нового проекта и выполнить все основные действия с исходным кодом программы, связанные с контролем версий.

Средства реализации: ограничений не накладывается, рекомендуется использование Subversion и Sun Netbeans 6.5.

Software Configuration Management или **Конфигурационное управление** подразумевает под собой комплекс методов, направленных на то, чтобы систематизировать изменения, вносимые разработчиками в программный продукт в процессе его разработки и сопровождения, сохранить целостность системы после изменений, предотвратить нежелательные и непредсказуемые эффекты, а также сделать процесс внесения изменений более формальным. Изначально управление конфигурацией применялось не в программировании, но в связи с высокой динамичностью сферы разработки ПО, в ней она особенно полезна. К процедурам можно отнести создание резервных копий, контроль исходного кода, требований проекта, документации и т. д. Степень формальности выполнения данных процедур зависит от размеров проекта, и при правильной ее оценке данная концепция может быть очень полезна. Конфигурационное управление требует выполнения множества трудоемких рутинных операций. На практике, в большинстве случаев, для конфигурационного управления применяются специальные системы контроля версий исходного кода программ. В качестве примера такой системы рассмотрим самую распространенную на сегодняшний день – Subversion.

Subversion — это бесплатная система управления версиями с открытым исходным кодом. Subversion позволяет управ-

лять файлами и каталогами, а так же сделанными в них изменениями во времени. Это позволяет восстановить более ранние версии данных, даёт возможность изучить историю всех изменений. Благодаря этому многие считают систему управления версиями своего рода «машиной времени».

Subversion может работать через сеть, что позволяет использовать её на разных компьютерах. В какой то степени, возможность большого количества людей не зависимо от их местоположения совместно работать над единым комплектом данных поощряет сотрудничество. Когда нет того ответственного звена цепи, того контролирующего элемента, который утверждает все изменения, работа становится более эффективной. При этом не нужно опасаться, что отказ от контролирующего элемента повлияет на качество, ведь благодаря сохранению истории изменений, даже если при изменении данных будут допущены ошибки, всегда можно сделать откат изменений к прежнему состоянию.

Некоторые системы управления версиями выступают также в качестве систем управления конфигурацией программного обеспечения. Такие системы специально созданы для управления деревьями исходного кода и имеют множество особенностей, непосредственно относящихся к разработке программ: они понимают языки программирования и предоставляют инструменты для сборки программ. Subversion не является такой системой, она представляет собой систему общего назначения, которую можно использовать для управления *любым* набором файлов. Для Вас это будут исходники Ваших программ, а для кого-то другого это будет список продуктов или сведённое цифровое видео.

Subversion предоставляет следующие возможности:

Контроль изменений каталогов

Subversion использует «виртуальную» файловую систему с возможностями управления версиями, которая способна отслеживать изменения во времени целых структур каталогов. Под управление версиями попадают и файлы, и каталоги.

Настоящая история версий

Subversion делает возможным добавление, удаление, копирование и переименование как файлов, так и каталогов. При

этом каждый вновь добавленный файл начинает жизнь с чистого листа, сохраняя собственную историю изменений.

Атомарная фиксация изменений

Каждый набор изменений либо попадает в хранилище целиком, либо не попадает туда вовсе. Это позволяет разработчикам создавать и фиксировать изменения логически оправданными кусками, предотвращая тем самым проблемы, которые могут возникать в тех случаях, когда только часть необходимых изменений помещается в хранилище успешно.

Метаданные с версиями

Каждый файл и каталог имеет собственный набор свойств, представленных в виде названия и значения. Вы можете создавать и сохранять любые необходимые пары названий свойств и их значений. Свойства файлов точно так же находятся под управлением версиями, как и их содержимое.

Выбор средств доступа к хранилищу по сети

В Subversion используется абстракция доступа к хранилищу, что позволяет реализовывать самые разные сетевые механизмы доступа. Subversion может быть подключена к серверу HTTP Apache в виде модуля, что даёт ей огромное преимущество с точки зрения устойчивости работы и способности к взаимодействию, а также предоставляет прямой доступ к существующим возможностям этого сервера, включая установление личности, проверку прав доступа и сжатие информации при передаче. Кроме того, имеется лёгкий самостоятельный сервер Subversion, который использует собственный протокол взаимодействия с клиентами и может легко туннелировать данные через SSH.

Единый способ работы с данными

Subversion обнаруживает различия между файлами с помощью специального бинарного алгоритма, который одинаково работает как с текстовыми, так и с бинарными файлами. Файлы записываются в хранилище в сжатом виде независимо от их типа, а различия между отдельными версиями могут передаваться по сети в обоих направлениях.

Эффективные ветки и метки

Временные затраты за использование веток и меток не должны быть пропорциональны размеру проекта. Subversion создаёт ветки и метки путём простого копирования проекта, ис-

пользуя механизм, похожий на жёсткие ссылки в файловых системах. Благодаря этому, операции по созданию веток и меток занимают немного времени.

Дружелюбность по отношению к разработчикам

Subversion не имеет исторического багажа. Она реализована в виде набора динамических библиотек на языке C, API которых хорошо известен. Это делает Subversion чрезвычайно удобной для сопровождения системой, пригодной для взаимодействия с другими приложениями и языками программирования.

Установленная Subversion имеет несколько компонентов. Ниже приводится краткий обзор того, что вы получаете. Не тревожьтесь, если краткие описания заставляют вас чесать затылок, в этой книге есть еще много страниц, посвященных облегчению вашего замешательства.

svn

Клиент с интерфейсом командной строки.

svnversion

Программа, показывающая состояние (в пределах ревизий существующих элементов) рабочей копии.

svnlook

Инструмент прямого управления хранилищем Subversion.

svnadmin

Инструмент для создания, настройки или восстановления хранилища Subversion.

svndumpfilter

Программа для фильтрации дамповых потоков хранилища Subversion.

mod_dav_svn

Подключаемый модуль для HTTP-сервера Apache, использующийся для предоставления сетевого доступа к вашему хранилищу.

svnserve

Собственный отдельный сервер, запускаемый как процесс-демон и доступный посредством SSH; еще один способ для предоставления сетевого доступа к хранилищу.

svnsync

Программа для последовательного зеркалирования одного хранилища в другое через сеть.

Рабочая копия Subversion представляет собой обычное дерево каталогов на вашем компьютере, содержащее набор файлов. Вы можете по своему усмотрению редактировать эти файлы и, если это исходные коды, вы можете обычным способом скомпилировать из них программу. Ваша рабочая копия — это ваше личное рабочее пространство. Subversion как не смешивает с вашими изменения, вносимые другими, так и не делает доступными для других изменения, сделанные вами, пока вы сами не прикажете сделать это. Вы даже можете иметь несколько рабочих копий одного и того же проекта.

После того, как вы внесли изменения в файлы вашей рабочей копии и убедились в том, что они корректно работают, Subversion предлагает вам команды «публикации» (записи в хранилище) ваших изменений, в результате чего они станут доступными для всех участников проекта. Если другие участники проекта опубликовали свои изменения, Subversion предлагает вам команды для объединения (путем чтения информации из хранилища) этих изменений с вашей рабочей копией.

Рабочая копия содержит несколько дополнительных файлов, созданных и обслуживаемых Subversion, которые помогают ей при выполнении этих команд. В частности, каждый каталог в вашей рабочей копии содержит подкаталог с именем .svn который называется *служебным каталогом* рабочей копии. Файлы в служебном каталоге помогают Subversion определить какие файлы рабочей копии содержат неопубликованные изменения, и какие файлы устарели по отношению к файлам других участников.

Как правило, хранилище Subversion содержит файлы (или исходный код) нескольких проектов; обычно каждый проект представляется в виде подкаталога файловой системы хранилища. При таком подходе, пользовательская рабочая копия обычно соответствует отдельному подкаталогу хранилища.

Для того чтобы создать рабочую копию, вам нужно получить какой-либо из подкаталогов хранилища. Например, если вы получите /calc, у вас будет рабочая копия наподобие этой:

```
$ svn checkout http://svn.example.com/repos/calc
A    calc/Makefile
A    calc/integer.c
```

```
A    calc/button.c
Checked out revision 56.

$ ls -A calc
Makefile  integer.c  button.c  .svn/
```

Буквы A говорят о том, что Subversion добавил этот элемент в вашу рабочую копию. Теперь у вас есть личная копия каталога /calc хранилища, с одним небольшим добавлением — каталогом .svn, содержащим, как было указано выше, дополнительную информацию, необходимую Subversion.

Предположим, вы внесли изменения в button.c. Так как каталог .svn помнит дату изменения файла и его оригинальное содержимое, Subversion может сказать о том, что вы изменили файл. Subversion не публикует ваших изменений, пока вы не прикажете это сделать. Публикация ваших изменений более известна как *фиксация* (или *checking in*) изменений в хранилище.

Для того, чтобы опубликовать ваши изменения, вы можете воспользоваться командой **commit**.

```
$ svn commit button.c -m "Fixed a typo in but-
ton.c."
Sending        button.c
Transmitting file data .
Committed revision 57.
```

Теперь ваши изменения в button.c, вместе с примечанием, описывающим эти изменения (а именно: исправление опечатки), зафиксированы в хранилище; если другой пользователь создаст рабочую копию /calc, он увидит ваши изменения в последней версии файла.

Предположим, у вас есть партнер, Светлана, которая создала рабочую копию /calc одновременно с вами. Когда вы зафиксировали изменения в button.c, рабочая копия Светланы осталась неизмененной, так как Subversion модифицирует рабочие копии только по запросу пользователей.

Для приведения рабочей копии в актуальное состояние Светлана может попросить Subversion *обновить* её рабочую копию, используя команду Subversion **update**. Это включит ваши изменения в ее рабочую копию, так же как и все другие изменения, зафиксированные после того, как она создавала рабочую копию.

```
$ pwd
/home/svetik/calc

$ ls -A
.svn/ Makefile integer.c button.c

$ svn update
U    button.c
Updated to revision 57.
```

Вывод команды **svn update** говорит, что Subversion обновила содержимое button.c. Обратите внимание, что Светлана не должна указывать, какой файл обновить; для определения файлов, которые необходимо привести в актуальное состояние, Subversion использует информацию в каталоге .svn, а также информацию из хранилища.

Операция **svn commit** публикует изменения любого количества файлов и каталогов за одну атомарную операцию. В своей рабочей копии вы можете менять содержимое файлов, создавать, удалять, переименовывать и копировать файлы и каталоги, а затем зафиксировать все изменения за одну атомарную транзакцию.

Под «атомарной транзакцией» понимается следующее: либо в хранилище вносятся все изменения полностью, либо они не вносятся вообще. Subversion ведёт себя так, принимая в расчет возможные программные сбои, системные сбои, проблемы с сетью, а также неверные действия пользователя.

Каждый раз, когда происходит фиксация, создаётся новое состояние файловой системы, которое называется *правкой (revision)*. Каждая правка получает уникальный номер, на единицу больший номера предыдущей правки. Начальная правка только что созданного хранилища получает номер 0 и не содержит ничего, кроме пустого корневого каталога.

Чтобы импортировать новый проект в хранилище Subversion, воспользуйтесь командой **svn import**.

Хотя Subversion имеет множество возможностей, опций и украшательств, в каждодневной практике используются только некоторые из них.

Типичный рабочий цикл выглядит примерно так:

Обновление рабочей копии

svn update
Внесение изменений
svn add
svn delete
svn copy
svn move
Анализ изменений
svn status
svn diff
svn revert
Слияние изменений, выполненных другими, с вашей рабочей копией
svn update
svn resolved
Фиксация изменений
svn commit

После внесения изменений вы должны зафиксировать их в хранилище, но перед этим было бы неплохо посмотреть, что же, собственно, вы изменили. Проанализировав перед фиксацией свои изменения, вы сможете составить более аккуратное лог-сообщение. Кроме того, вы можете обнаружить, что изменили файл непреднамеренно, что позволит еще до фиксации вернуть файл к предыдущему состоянию. К тому же, это хорошая возможность пересмотреть и проверить изменения перед их публикацией. Чтобы увидеть все сделанные изменения, вы можете воспользоваться **svn status**, **svn diff** и **svn revert**. Первые две команды вы можете использовать для того, чтобы найти измененные файлы рабочей копии, а затем, при помощи третьей, отменить некоторые (или все) изменения.

svn status печатает пять колонок букв, затем несколько пробелов, затем имя файла или каталога. Первая колонка показывает статус файла или каталога и/или ее содержимого. Вторая колонка показывает статус свойств файлов и каталогов. Если во второй колонке показывается M, свойства были изменены. Если в этой колонке показывается C, то это означает, что свойства файла находятся в состоянии конфликта, который должен быть разрешен до фиксации изменений в хранилище. Во всех других случаях будет выведен пробел. Третья колонка может содержать

только пробел или L, это значит, что у каталога заблокирована рабочая область .svn. Четвертая колонка может содержать только пробел или +, это означает, что элемент был запланирован для «добавления с историей». Пятая колонка может содержать только пробел или S. Это означает, что файл или каталог был переключен с пути остальной рабочей копии на ветку (используя **svn switch**). Шестая колонка показывает информацию о блокировках.

svn diff - еще один механизм для анализа изменений. Запустив **svn diff** без аргументов, можно увидеть, *какие именно* изменения вы внесли, в результате будут выведены изменения файлов в едином формате представления различий. Команда **svn diff** формирует свой вывод, сравнивая ваши рабочие файлы с кэшированными «нетронутыми» копиями из .svn. Весь текст запланированных для добавления файлов показывается как добавленный, а весь текст запланированных для удаления файлов показывается как удаленный.

svn revert - возвращает файл в состояние, предшествующее модификации, путем замены файла его кэшированной «первоначальной» копией из .svn-области. Кроме того, обратите внимание, что **svn revert** может отменить *любые* запланированные операции — например, вы можете прийти к решению всё-таки не добавлять новый файл

svn log - показывает вам развернутую информацию: лог-сообщения, присоединенные к правкам, с указанием даты изменений и их авторов, а также изменения путей к файлам в каждой правке.

svn cat - эта команда используется для получения отдельного файла в том виде, в каком он был в конкретной ревизии и вывода его на экран.

svn list - показывает список файлов в каталоге для любой указанной правки.

svn list - показывает содержимое каталога в хранилище, при этом не закачивая его на локальную машину.

svn cleanup - ищет в рабочей копии и выполняет незавершенные лог-файлы, удаляя по ходу выполнения блокировки в рабочей копии. Если Subversion когда-нибудь говорила вам о

том, что часть рабочей копии «заблокирована», то вам нужно запустить эту команду.

svn import — это быстрый способ скопировать неверсионированное дерево файлов в хранилище, создавая при необходимости подкаталоги. Обратите внимание на то, что после завершения импорта оригинальное дерево файлов *не* конвертируется в рабочую копию. Чтобы начать работать, вам необходимо создать новую рабочую копию (**svn checkout**) дерева файлов.

Порядок выполнения лабораторной работы

1. Создайте новый проект.
2. Экспортируете созданный проект в репозиторий системы контроля версий.
3. Удалите созданный проект на своем компьютере и обновите проект из репозитория.
4. Внесите изменения в файлах с исходными кодами и сохраните изменения в репозитории. Обновите файлы с исходными кодами из репозитория.
5. Внесите изменения в файлах с исходными кодами таким образом, чтобы у двух участников проекта изменения были в одном и том же файле. Попытайтесь сохранить изменения в репозитории. Устраните обнаруженные конфликты версий. Повторно сохраните изменения в репозитории.
6. Создайте отдельную ветку проекта. Внесите изменения в файлы с исходными кодами. Сохраните изменения в репозитории.
7. Объедините созданную на предыдущем шаге ветку с основной веткой проекта.

5. Лабораторная работа № 5. Использование средств автоматизации тестирования программного обеспечения

Количество аудиторных часов: 8.

Цели занятия: получение первоначальных навыков использования средств автоматизации тестирования программного обеспечения.

Цель работы: разработать тесты для приведенных классов, провести регрессионное тестирование, выполнить профилирование программы.

Средства реализации: ограничений не накладывается, рекомендуется использование Sun Netbeans 6.5.

Тестирование — процесс выполнения программы с целью обнаружения ошибок. Шаги процесса задаются тестами. Каждый тест определяет:

- свой набор исходных данных и условий для запуска программы.
- набор ожидаемых результатов работы программы.

Тестирование обеспечивает:

- обнаружение ошибок;
- демонстрацию соответствия функций программы ее назначению;
- демонстрацию реализации требований к характеристикам программы;
- отображение надежности как индикатора качества программы.

Тестирование по принципу «черного ящика»

При тестировании отдельных модулей как «черных ящиков» известны функции программы, и при этом исследуется работа каждой функции (рис. 5.1).

Структура программы или модуля (в зависимости от того, что является объектом тестирования) неизвестна. Тесты представляют собой набор входящих параметров с имеющимися значениями и набор выходящих параметров с ожидаемыми значениями. Если фактически полученные параметры соответствуют ожидаемым, тест пройден успешно.

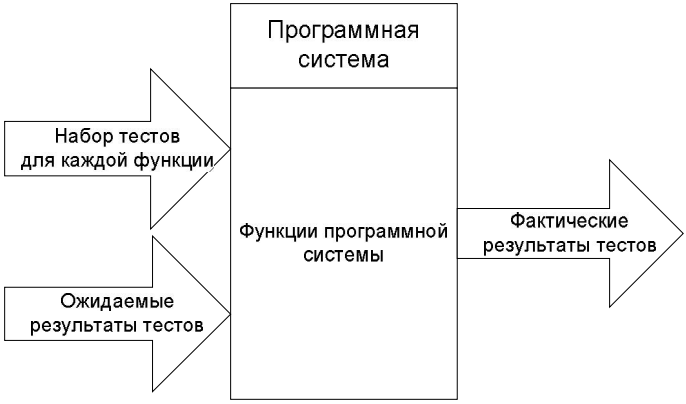

Рис. 5.1. Тестирование по принципу «черного ящика»

Тестирование по принципу «черного ящика» обеспечивает поиск следующих категорий ошибок:

- некорректные или отсутствующие функции;
- ошибки интерфейса;
- ошибки во внешних структурах данных или в доступе к внешней базе данных;
- ошибки характеристик (требования к аппаратному обеспечению и т. п.);
- ошибки инициализации или завершения.

Тестирование по принципу «белого ящика»

При тестировании отдельных модулей как «белых ящиков» известны внутренние состав и структура модулей, а исследуются взаимосвязи между элементами программы (рис. 5.2). Проверяется корректность построения всех элементов программы и правильность их взаимодействия друг с другом. Обычно анализируются управляющие связи элементов, реже — информационные связи. Тестирование по принципу «белого ящика» характеризуется степенью соответствия выполняемых тестов формируемой логике программы. Исчерпывающее тестирование, как и в случае тестирования по принципу «черного ящика», затруднительно. Если все элементы (операторы) программы и переходы между этими элементами (условные, безусловные, циклы и т.

п.) представить графом, то программа считается полностью протестированной, если проверена правильность ее работы на всех путях на рассматриваемом графе.

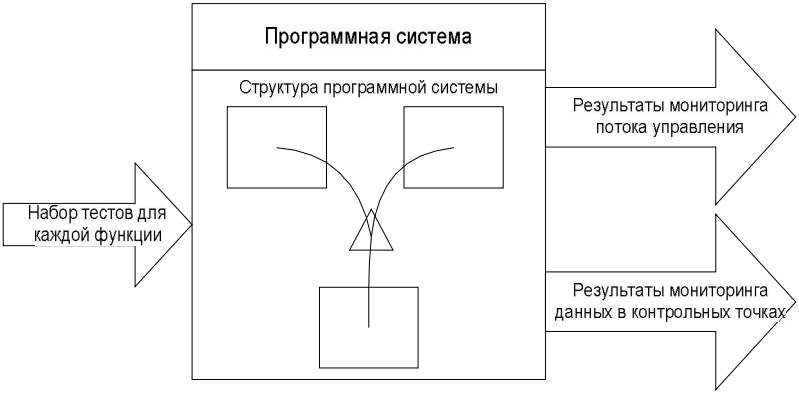

Рис. 5.2. Тестирование по принципу «белого ящика»

Таким образом, тесты представляют собой эталонный граф переходов между операторами с контрольными точками для проверки правильности обработки данных. Если в результате выполнения теста фактические переходы и данные в контрольных точках совпадают с эталонными, тест считается успешно пройденным.

Тестирование по принципу «белого ящика» позволяет быстро выяснить большую часть ошибок в «центре» программы, то есть в наиболее часто выполняющихся участках кода. Проверка потока управления программы на нескольких наборах исходных данных позволяет говорить о работоспособности программы с большей вероятностью, чем при тестировании по принципу «черного ящика» на тех же наборах данных.

В то же время количество независимых потоков управления может быть очень велико. Если при разработке программы изначально не применялись высокоформализованные методы кодирования и не строились высокодетализированные модели программного продукта, получить эти маршруты (и, следовательно, исходные тесты) весьма затруднительно. Как минимум, нужно провести полный реинжиниринг исходного кода (то есть

получить требуемые высокодетализированные модели программного продукта), на что после написания программы решится далеко не каждый разработчик.

Тестирование по принципу «белого ящика», как правило, используют при разработке программ с повышенными требованиями к надежности (т. е. когда сбой в программе может привести к смерти человека или какой-либо катастрофе). Кроме того, тестирование «белого ящика» используют при тестировании наиболее критических элементов системы, от работоспособности которых зависит правильность работы всех остальных элементов, и для локализации ошибок, выявленных при тестировании по принципу «черного ящика». В остальных случаях исчерпывающее тестирование «белого ящика» проводят редко.

Тестирование — один из важнейших этапов контроля качества разработанного ПО. Автоматическое тестирование является его составной частью. Оно использует программное обеспечение для проверки выполнения проводимых тестов, что помогает сократить время тестирования и упростить его процесс.

В наши дни создается большое количество инструментов с графическим интерфейсом (GUI), использование которых помогает облегчить работу программистов и повышает их производительность. Эти процессы увеличили требования к тестировщикам. Сейчас они должны обрабатывать большое количество информации за короткий срок. Потому использование автоматических тестов является необходимым условием для экономии средств и времени тестировщиков.

Современные средства разработки создают довольно сложные приложения, и их ручное тестирование является очень трудоемким процессом. Недостаток ручного тестирования также в том, что результаты выполнения тестов не сохраняются и их трудно повторить заново. Автоматические тесты позволяют упростить процесс ручного тестирования, сделать его наиболее удобным и точным.

Для автоматизации тестирования существует большое количество приложений. Наиболее популярные из них по итогам 2007 года:

* Mercury LoadRunner, QTP
* Segue SilkPerformer

* Rational FunctionalTester, PerformaneTester, TestStudio
* AutomatedQA TestComplete

Использование этих инструментов помогает тестировщикам автоматизировать следующие задачи:

* установка продукта
* создание тестовых данных
* GUI взаимодействие
* определение проблемы

Однако автоматические тесты не могут полностью заменить ручное тестирование. Автоматизация всех испытаний — очень дорогой процесс, и потому автоматическое тестирование является лишь дополнением ручного тестирования. Наилучший вариант использования автоматических тестов — регрессионное тестирование.

Регрессио́нное тести́рование (англ. regression testing, от лат. regressio — движение назад) — собирательное название для всех видов тестирования программного обеспечения, направленных на обнаружение ошибок в уже протестированных участках исходного кода. Такие ошибки — когда после внесения изменений в программу перестает работать то, что должно было продолжать работать, — называют регрессионными ошибками (англ. regression bugs).

Обычно используемые методы регрессионного тестирования включают повторные прогоны предыдущих тестов, а также проверки, не попали ли регрессионные ошибки в очередную версию в результате слияния кода.

Из опыта разработки ПО известно, что повторное появление одних и тех же ошибок — случай достаточно частый. Иногда это происходит из-за слабой техники управления версиями или по причине человеческой ошибки при работе с системой управления версиями. Но настолько же часто решение проблемы бывает «недолго живущим»: после следующего изменения в программе решение перестаёт работать. И наконец, при переписывании какой-либо части кода часто всплывают те же ошибки, что были в предыдущей реализации.

Поэтому считается хорошей практикой при исправлении ошибки создать тест на неё и регулярно прогонять его при последующих изменениях программы. Хотя регрессионное тести-

рование может быть выполнено и вручную, но чаще всего это делается с помощью специализированных программ, позволяющих выполнять все регрессионные тесты автоматически.

В некоторых проектах даже используются инструменты для автоматического прогона регрессионных тестов через заданный интервал времени. Обычно это выполняется после каждой удачной компиляции (в небольших проектах) либо каждую ночь или каждую неделю.

Регрессионное тестирование является неотъемлемой частью экстремального программирования. В этой методологии проектная документация заменяется на расширяемое, повторяемое и автоматизированное тестирование всего программного пакета на каждой стадии цикла разработки программного обеспечения.

Регрессионное тестирование может быть использовано не только для проверки корректности программы, часто оно также используется для оценки качества полученного результата. Так, при разработке компилятора, при прогоне регрессионных тестов рассматривается размер получаемого кода, скорость его выполнения и время компиляции каждого из тестовых примеров.

Порядок выполнения лабораторной работы

1. Создайте исходный код для классов на приведенных выше диаграммах.
2. Напишите автоматические тесты для проверки работоспособности программы в области вставки, обновления, удаления, поиска записей в базе данных.
3. Выполните тесты.
4. Внесите искусственно ошибку в программу.
5. Выполните повторно тесты. Сформируйте отчет о найденных ошибках. Исправьте ошибки. Выполните повторно тесты.
6. Выполните профилирование программы. Проконтролируйте использование оперативной памяти, процессора.
7. Определите узкие места программы, сформулируйте рекомендации по увеличению производительности программы за счет оптимизации узких мест.

6. Организация самостоятельной работы студента

Самостоятельная работа студентов по дисциплине «Программная инженерия» включает:

1) проработку теоретического материала, изложенного на лекциях;

2) подготовку к лабораторным работам и оформление отчетов по работе;

3) изучение тем (вопросов) теоретической части курса, отводимых на самостоятельную проработку.

Самостоятельному изучению подлежат следующие вопросы основных разделов дисциплины:

1) введение:

- современные методологии разработки ПО;
- сопровождение ПО;
- методы и средства программной инженерии;

2) инженерия требований:

- совершенствование процессов работы с требованиями;
- методы анализа требований;
- многокритериальный выбор;

3) управление рисками в IT:

- управление рисками в инжиниринговой компании;
- разработка программы управления рисками в IT;

4) управление персоналом в проектах по разработке ПО:

- методы тестирования при наборе персонала;
- психологический портрет личности;
- групповые тренинги по формированию команды разработчиков;

5) управление проектами:

- графические среды представления проектной деятельности;
- история концепций управления проектами;

6) управление качеством, результативность IT:

- цели и техники тестирования;
- оценка программ в результате тестирования;
- оценка выполненных тестов;

7) управление стоимостью проекта:

- научная организация труда;
- кодекс законов о труде;
- основы разработки бизнес-планов.

Качество самостоятельной работы студентов оценивается по результатам устного опроса студентов на лекциях, защиты отчетов по лабораторным работам и проверки конспектов по указанным темам.

Рекомендуемая литература

1. Орлов С.А. Технологии разработки программного обеспечения: Учебник. – СПб.: Питер, 2002. – 464 с (наличие в библ. ТУСУР – 26 экз.).

2. Вендров А.М. Проектирование программного обеспечения экономических информационных систем: учебник. – М.: Финансы и статистика, 2002. (наличие в библ ТУСУР – 36 экз.).

3. Ларман К. Применение UML и шаблонов проектирования: Введение в объектно-ориентированный анализ и проектирование: Учебное пособие: Пер. с англ. - М.: Вильямс, 2001. - 496 с.

4. Роберт Т. Фатрелл, Дональд Ф. Шафер, Линда И. Шафер. Управление программными проектами. Достижение оптимального качества при минимуме затрат.: Персона.—М.: Издательский дом «Вильямс», 2004г. – 1136 с

5. Ф. Брукс. Мифический человеко-месяц, или как создаются программные системы – Символ-Плюс, 2006 – 304с.

6. Орлик С. Введение в программную инженерию и управление жизненным циклом программного обеспечения / Пер. с англ. [Электронный ресурс]. — URL: http://sorlik.ru/swebok-ru/

7. Системная и программная инженерия. Словарь-справочник: учебное пособие / Батоврин В.К. - М.: ДМК Пресс, 2010. - 280 с. Гриф УМО вузов по университетскому политехническому образованию [Электронный ресурс]. — URL: http://e.lanbook.com/books/pdf.php?book_id=1097&p_id=25&bookid=1279

8. Мытник С.А. Технологии программирования: учеб. пособие. - Томск: ТУСУР, 2007. – 196 с. : (*наличие в библ. ТУСУРа – 51 экз.*).

9. Липаев В.В. Программная инженерия: методологические основы: учебник для вузов. — М.: ТЕИС, 2006. – 605 с. В библиотеке всего экз.: 1 (счз1 (1).

10. Попов Ю.И., Яковенко О.В. Управление проектами : учеб. пособие. - М.: Инфра-М, 2007. – 207 с. Экземпляры всего: 13 анл (3), счз1 (1), счз5 (1), аул (8)

11. Ехлаков Ю.П. Введение в программную инженерию [Электронный ресурс]: методические указания к выполнению

практических и самостоятельных работ; Томский государственный университет систем управления и радиоэлектроники. – 14 с. – URL: http://edu.tusur.ru/training/publications/996

12. Информационные технологии и программные продукты: рынок, экономика, нормативно-правовое регулирование: учебное пособие / Ю. П. Ехлаков. — Томск: Томский государственный университет систем управления и радиоэлектроники, 2007. — 176 с [Электронный ресурс]. — URL: http://edu.tusur.ru/training/publications/26

13. Введение в программную инженерию: учебное пособие / Ю. П. Ехлаков. — Томск: Эль Контент, 2011. — 148 с .[Электронный ресурс]. — URL: http://edu.tusur.ru/training/publications/141

14. Бабенко Л.П., Лаврищева Е.М. Основы программной инженерии. Учебник. – Киев: Знання, 2008. –269 с.

15. Соммервилл Иан. Инженерия программного обеспечения, 6-е издание.: Пер. с англ. -М.: Издательский дом "Вильямс", 2007. - 624 с

16. Вигерс К.И. Разработка требований к ПО. Москва, 2004.– Русская редакция Microsoft.–575с.

17. Шафер Д, Фатрел Р, Шафер Л. Управление программными проектами: достижение оптимального качества при минимуме затрат.: Пер. с англ. - М.: Вильямс., 2003. - 1136с.

18. Андон Ф.И., Коваль Г.И., Коротун Т.М., Суслов В.Ю. Основы инженерии качества программных систем.–К: Академпериодика, 2006.–502с.

19. Руководство к своду знаний по управлению проектами (Руководство PMBOK). Издательства: Институт Управления Проектами, Project Management Institute, 2004 г.

20. Салливан Эд. Время – деньги. Создание команды разработчиков программного обеспечения/ Пер.с англ. – М.: Русская редакция, 2002. – 364с. Константин Л. Человеческий фактор в программировании. - Пер. с англ. - СПб:Символ-Плюс, 2004. - 384 с.

21. Международные стандарты ISO серии 9000 [Электронный ресурс]. – В открытом доступе. – URL: http://www.m2bc.ru/qs_iso-scheme