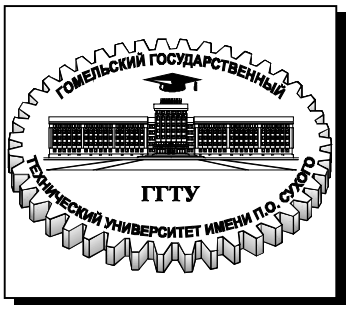


Министерство образования Республики Беларусь

Учреждение образования
«Гомельский государственный технический
университет имени П. О. Сухого»

Кафедра «Информатика»

Д. В. Прокопенко, В. Н. Шибеко

УПРАВЛЕНИЕ РАЗРАБОТКОЙ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ

Рекомендовано учебно-методическим объединением по образованию в области информатики и радиоэлектроники в качестве учебно-методического пособия для студентов учреждений высшего образования, обучающихся по специальности 1-40 04 01 «Информатика и технологии программирования»

Электронный аналог печатного издания

Гомель 2020

УДК 004.9:005.8(075.8)
ББК 32.972я73
П80

Рецензенты: зав. кафедрой информационно-вычислительных систем Белорусского торгово-экономического университета потребительской кооперации д-р техн. наук, профессор *А. Н. Семенюта*;
доц. кафедры «Автоматизированные системы обработки информации» Гомельского государственного университета имени Ф. Скорины канд. техн. наук, доцент *А. В. Воруев*

Прокопенко, Д. В.

П80 Управление разработкой программного обеспечения : учеб.-метод. пособие / Д. В. Прокопенко, В. Н. Шибеко ; М-во образования Респ. Беларусь, Гомел. гос. техн. ун-т им. П. О. Сухого. – Гомель : ГГТУ им. П. О. Сухого, 2020. – 75 с. – Систем. требования: PC не ниже Intel Celeron 300 МГц ; 32 Mb RAM ; свободное место на HDD 16 Mb ; Windows 98 и выше ; Adobe Acrobat Reader. – Режим доступа: https://elib.gstu.by. – Загл. с титул. экрана.

ISBN 978-985-535-447-6.

Изложены основные темы, изучаемые в курсе «Управление разработкой программного обеспечения». Приведены сведения о жизненном цикле программного обеспечения, проектном подходе к разработке, рассмотрены вопросы планирования работ, ресурсов и управления версиями.

Для студентов специальности 1-40 04 01 «Информатика и технологии программирования».

УДК 004.9:005.8(075.8)
ББК 32.972я73

ISBN 978-985-535-447-6

Оглавление

ВВЕДЕНИЕ

Целью учебно-методического пособия является формирование практических навыков в области управления разработкой программного обеспечения. Особенностью современного состояния в данной области является существенное развитие средств поддержки автоматизации разработки: от развитых инструментов программирования и тестирования программ до автоматизированных средств, помогающих руководителю проекта не только планировать, но и контролировать ход разработки. Современные требования бизнеса, направленные на сокращение сроков внедрения программных продуктов в производство, неизбежно влекут и изменение моделей жизненного цикла разработки программного обеспечения, переход от классических водопадных моделей с регламентированным по договорным документам сроком внедрения к итеративным моделям быстрой разработки с короткими циклами производства программного продукта. При классическом подходе тщательно анализируются желания заказчика, формируются требования, готовятся проект и проектная документация, выполняется разработка кода и тестирование и только после этого продукт вводится в эксплуатацию. Положительным моментом является наличие согласованной проектной документации, определенность и понятность шагов, простота применения и управления. Но такой подход приводит к значительным срокам разработки, как следствие, требования морально устаревают и часто не соответствуют более современным целям бизнеса. Как показывает опыт, более 40 % проектов завершились с опозданием или превысили запланированный бюджет, или требуемые функции не были реализованы в полном объеме.

Это и явилось тем самым отправным моментом для поиска новых моделей жизненного цикла, что привело к моделям с коротким циклом разработки. Но такие модели могли возникнуть только при наличии средств автоматизации тестирования непосредственно в этой короткой итерации, разработка программного продукта стала немыслимой без постоянного процесса тестирования. Только в этом случае, в рамках итерации, стало возможным создание продукта приемлемого качества.

Переход от классических моделей разработки к быстрым (адаптивным) моделям привел к изменению объема и качества документации процесса. Отказ от формализованных и согласованных требований, от проектов, конечно, снизил качество документации, но также существенно уменьшил сроки подготовительных этапов, переведя это на уровень «пользовательских историй», которые могут быть опера-

тивно пересмотрены. В рамках короткой итерации пользователи получают готовый продукт и на основе его пересматривают свои истории, их важность, приоритет. Преимуществом данного подхода является постоянное участие представителей бизнеса в подготовке и выпуске очередной реализации (релиза) продукта. Но короткий цикл одновременно является и недостатком, так как сосредоточение только на текущих целях не дает возможности увидеть перспективу («за деревьями не виден лес»). И не зря в современных процессах видна тенденция более долгосрочного планирования, как минимум квартального, и формирование соответствующих целей более высокого уровня (эпиков). В этом случае просматривается двухуровневый контур управления:

– первый – создание эпиков (неформализованных требований в новом понимании);

– второй – преобразование в ясные пользовательские истории, формирование задач итерации, управление итерацией.

Глава 1. ВВЕДЕНИЕ В ПРОЕКТНОЕ УПРАВЛЕНИЕ

1.1. Проект

Разработка программного обеспечения – это разработка и реализация проекта. Результатом проекта является созданное программное обеспечение или программный продукт. Далее будем считать, что понятия «проект», «программный продукт», «программное обеспечение» синонимичны. Как следствие, далее понимаем, что управление разработкой программного обеспечения – это управление программным проектом. Согласно руководству к своду знаний по управлению проектами [1] проект – это временное предприятие, предназначенное для создания уникальных продуктов, услуг или результатов. Временный характер проекта означает, что у любого проекта есть определенное начало и завершение. Завершение наступает, когда возникают следующие моменты принятия решений:

– цели проекта достигнуты;

– признано, что цели проекта не могут быть достигнуты;

– исчезла необходимость в проекте.

Как процесс, проект может рассматриваться как последовательность действий в рамках разработки, при наличии ограничений и применения ресурсов. Ресурсы в программных проектах – это, в первую очередь, специалисты, участвующие в разработке, команда разработки. Ограничения – это возможные ограничения на сроки, ресурсы.

Проект характеризуется:

1) *целью проекта*. Целью является программный продукт, передаваемый пользователям и несущий некоторую для них ценность;

2) *сложностью*. Для достижения цели проекта необходимо выполнить множество работ по проектированию, разработке, тестированию и внедрению у пользователя программного продукта. Выполняемые работы тесно связаны между собой как во времени, так и в пространстве, связи могут быть очень сложными, особенно если в проекте много работ и много исполнителей. И потому современные тенденции предполагают создание небольших команд (5–9 человек) и коротких ясных циклов;

3) *уникальностью*. Уникальность проекта проявляется не только в уникальности конкретных требований, «пользовательских историй», но и в уникальности конкретной команды разработчиков, адаптируемости циклов разработки, наличии и использовании инструментов и сред разработки;

4) *жизненным циклом*. Согласно руководству к своду знаний по управлению проектами [1] любой проект характеризуется последовательностью фаз, задаваемых исходя из потребностей управления проектом. Классический вариант:

– инициализация;

– планирование;

– реализация и мониторинг;

– завершение.

Более подробно об этом изложено ниже, при рассмотрении конкретных моделей. Кроме того, все проекты имеют три важных показателя:

– стоимость проекта;

– содержание (объем работ по проекту);

– сроки реализации.

Эту «тройку» (стоимость, содержание, сроки) называют проектным треугольником ограничений (рис. 1.1), который используется для контроля отклонения факта от данных показателей. Особенность его в том, что изменение одного из показателей влечет изменение других;

Рис. 1.1. Треугольник ограничений проекта

5) *ограниченностью во времени.* Согласно [1] проект ограничен временными рамками начала и завершения. В рамках цикла разработки необходима концентрация ресурсов для выполнения намеченных работ. По мере освобождения ресурсы используются на другие цели. Конец проекта наступает вместе с достижением всех его целей, либо когда нарушены стороны треугольника ограничений (т. е. нарушены сроки, не выполнены объемы, превышены ресурсы).

1.2. Жизненный цикл разработки программного обеспечения

Любой проект проходит этапы, которые в совокупности образуют то, что называют жизненным циклом проекта. Это касается и программного обеспечения. Жизненный цикл программного проекта начинается с принятия решения о необходимости создания и заканчивается при соответствующем решении. Но на этом жизненный цикл программного продукта не завершается, так как после распространения версии продукта требуется обеспечение необходимого его обслуживания, выполняемого на фазе поддержки.

И если для проекта жизненный цикл связан разработкой, внедрением и развертыванием, то фаза обслуживания готового программного продукта связана со сбытом, поддержкой его в актуальном состоянии, соответствующего требованиям клиента и рынка (рис. 1.2). Фаза обслуживания программного продукта завершается снятием его с сопровождения. Отметим, что программные продукты, в отличие от материальных, не «ржавеют», а морально устаревают. Эта особенность и является основной причиной принятия решения о снятии программного продукта с сопровождения. Именно эта особенность влечет падение интересов пользователей к данному продукту и выбор более совершенного, ценного и доступного с их точки зрения.

В рамках данной книги кратко рассмотрим модели жизненного цикла программного проекта.

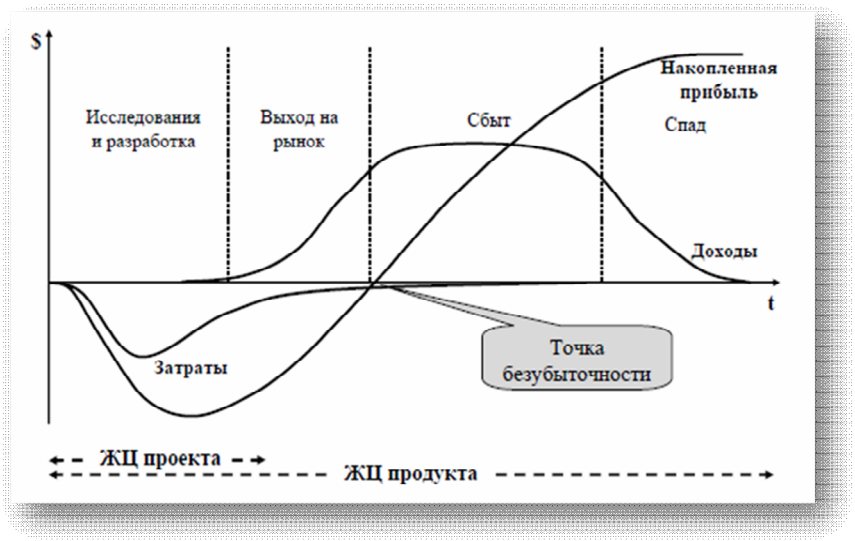


Рис. 1.2. Жизненный цикл продукта и проекта

Наиболее известными являются три: каскадная, инкрементная и спиральная модели жизненного цикла, ставшие классическими моделями разработки.

Каскадная модель. В рамках данной модели жизненный цикл выглядит как поток последовательных работ, проходящий через унифицированные фазы разработки (рис. 1.3):

– анализа требований;
– проектирования;
– реализации;
– тестирования;
– ввода в действие.

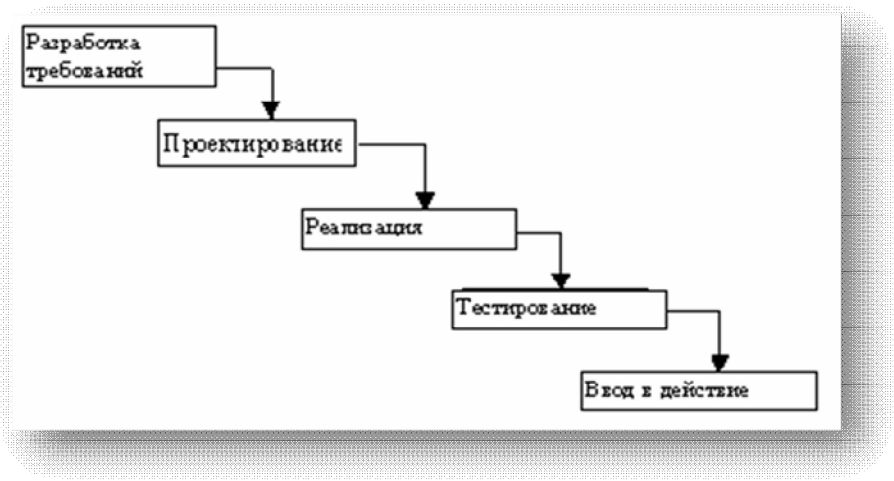


Рис. 1.3. Каскадная модель

Модель предусматривает, что выполнение каждой фазы начинается только после полного завершения предыдущей. На всех фазах выполняются организационные процессы управления проектом, включающие инициацию, оценку качества, верификацию и аттестацию, конфигурацию, разработку документации. Главная особенность модели – в результате завершения фаз формируются продукты, которые не могут изменяться на последующих шагах (требования и спецификация, проектная документация, программный код, эксплуатационная документация).

Достоинства модели:

– определенность и ясность этапов, простота ее применения;

– на каждой фазе формируется законченный набор документации (промежуточный продукт), отвечающий критериям полноты и согласованности. Этим обеспечивается ясность и логичность разработки продуктов последующих фаз;

– логическая последовательность фаз позволяет планировать сроки завершения всех работ и соответствующие ресурсы (денежные, материальные и людские).

Каскадная модель удобна при разработке относительно простых программных продуктов, когда в самом начале разработки достаточно точно и полно сформулированы все требования к продукту. В чистом виде она практически не применяется, так как обладает рядом недостатков:

– сложность полного и четкого формулирования требований при условии невозможности их изменения на протяжении жизненного цикла;

– как следствие, низкая гибкость в управлении проектом, так как невозможен возврат к предыдущим фазам для решения возникающих проблем, что в свою очередь приводит к увеличению сроков и, соответственно, – затрат;

– непригодность промежуточного продукта для использования;

– позднее обнаружение проблем, в результате устранение ошибок более ранних фаз на последней фазе обходится очень дорого;

– пользователь принимает участие только в самом начале, при разработке требований, и в самом конце, при приемке программного продукта. Пользователи могут оценить возможности продукта только после окончания всего процесса разработки. Обучение происходит в конце жизненного цикла, когда продукт уже запущен в эксплуатацию.

Устранение недостатков отсутствия возможности возврата к предыдущим фазам в рамках каскадной модели является целью итерационной модели.

Классическая итерационная модель. Никто не может объять необъятное. Только реализация очень простых и четко определенных задач проходит без возвратов на предыдущие шаги, например, простые задачи печати документов. При разработке больших управляющих систем, например, в области космических исследований, необходимость в возвратах на предыдущие этапы возникает на любом этапе жизненного цикла. Это может быть следствием уточнения внешних условий, требований к эксплуатации системы, изменения списка используемого оборудования, а также ошибок, допущенных на предыдущих фазах.

Классическая итерационная модель во главу угла ставит именно возможность возвратов на предыдущие этапы (рис. 1.4).

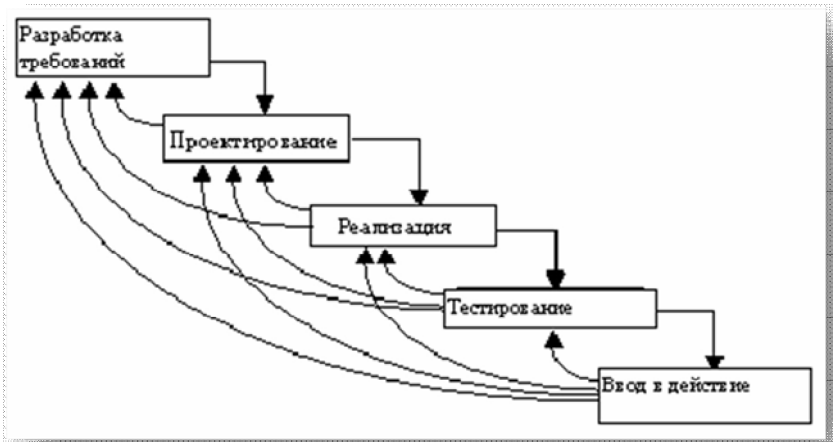


Рис. 1.4. Классическая итерационная модель

Каждый этап завершается проверкой полученных результатов. Цель проверки – устранить как можно большее число проблем, связанных с разработкой программного продукта. Это не обязательно ошибки, это также может быть уточнение условий и результатов предыдущих этапов.

Разработка ведется итерациями с учетом обратной связи между этапами, при этом возможен возврат на любой этап для его пересмотра. При таком подходе время жизни каждого из этапов растягивается на весь период разработки, каждый этап выпускает дополнения, изменения в своем промежуточном продукте.

По завершении этапа проводится его анализ и проверки. Каждая из указанных проверок может отослать разработчиков системы к повторению любого из ранее пройденных этапов. Подобный подход предлагает распределять наращивание функциональности и интерфейсных возможностей по итерациям, позволяет ослабить требование переделки старого при возвратах.

Подобная модель зарекомендовала себя в очень сложных системах, целевых программах, когда требуется не просто разработка, а исследования, макетирование будущей сложной системы, например, разработок в области космоса. *Но у нее остался один существенный недостаток – конечный результат пользователь может увидеть очень и очень нескоро.* Устранить данный недостаток призвана спиральная модель.

Спиральная модель. Цель спиральной модели – на основе цикла (спирали) производится разработка новой версии продукта. В каждой версии выполняются этапы каскадной модели. Отличие – по завершении передача пользователю готовой версии программного продукта.

Далее жизненный цикл повторяется на каждом витке спирали: разрабатываются (уточняются) требования, планируются работы следующего цикла, которые учитывают предложения и замечания пользователей, полученные при эксплуатации предыдущей версии (рис. 1.5).

Рис. 1.5. Спиральная модель

Эта модель характерна для большинства систем. Перед первым циклом у заказчика и разработчика нет конечного продукта, а также (что возможно и реально) нет уверенности в том, что заказчик получит то, что он желает, так видимые им риски могут быть очень велики. А потому принимается решение о разработке продукта по частям, с выпуском версий и возможностью изменения требований, либо принятия решения об отказе от дальнейшего развития.

Достоинства модели:

– уже после первого цикла пользователи системы увидят работоспособный продукт в виде его версии. Благодаря этому пользователи готовят предложения по внесению изменений в требования, уточняя и дополняя их;

– короткие циклы и возможность изменения требований при разработке программного продукта обеспечивают большую гибкость разработки, устранение ошибок и неточностей требований. Это также снижает риски заказчика, так как он может принять решение о завершении цикла, не выходя на очередной, либо о внесении изменений.

Недостатки модели:

– неясны перспективы развития проекта. Этот недостаток вытекает из возможности принятия решения заказчиком об остановке развития;

– затруднены операции планирования работ и ресурсов, так как границы перехода от одного этапа к другому становятся «размытыми». Для решения этой проблемы вводятся временные ограничения на каждый этап жизненного цикла внутри спирали. На основе этого готовятся объемы, сроки реализации работ и стоимость данного цикла. Плановые сроки, стоимость готовятся на основе опыта команды разработчиков и статистических данных, полученных в предыдущих проектах или утвержденных нормативов разработки.

Спиральная модель получила широкое распространение, так как ее применение целесообразно во многих случаях:

– при разработке инновационных проектов;

– при разработке проектов, где возможны существенные изменения или дополнения требований.

При всех своих достоинствах спиральная модель ориентирована на среднесрочные проекты, длительность которых может быть до года.

Но развитие сред проектирования и разработки программного обеспечения, а также требование времени быстрого внедрения программных продуктов в производственный цикл дали стимул к очередному пересмотру моделей жизненного цикла, переходу к моделям быстрой разработки.

RAD (*rapid application development* – быстрая разработка приложений) – модель ЖЦПО, полученная для создания программных продуктов с более высокой скоростью и приемлемым качеством, что было затруднено при классических подходах. Опишем данную модель на основе информации из Википедии (Википедия: RAD-программирование).

Четыре практики RAD:

– небольшая команда разработчиков;

– срок проекта – три-четыре месяца;

– на основе прототипирования и средств визуального моделирования и разработки;

– активное привлечение заказчика.

Последнее условие подразумевает выполнение требований заказчика с учетом их возможных изменений в период разработки системы. Кроме того, *обязательным фактором RAD является качественная документация*, обеспечивающая удобство эксплуатации и будущее сопровождения системы. Это означает, что дополнительные затраты на сопровождение программного продукта (например, силами Заказчика) сразу после поставки будут значительно меньше.

Условия применения:

– *интерфейс пользователя* (*GUI*) *есть главный фактор*. RAD-технология предлагает прототипы интерфейсов, причем они обсуждаются еще на этапе пользовательского проектирования;

– *низкая вычислительная сложность ПО*. Именно распространение персональных компьютеров дало толчок данной модели разработки. Предприятиям требовались простые типовые программные продукты, внедренные в их производственный цикл;

– *необходимо выполнение проекта в сжатые сроки*. Предыдущее условие часто шло «рука об руку» с данным. Типовые решения бухгалтерского, производственного учета позволяли создавать продукты, за короткие сроки и отвечающие требованиям текущего момента. Опыт команд разработчиков играет здесь важную роль, так как знание технологических процессов позволяет им выполнить данное условие. Такая команда понимает, что при больших сроках весьма высока вероятность отклонений в условиях, например, изменения в налоговом законодательстве, учетной политике предприятия. В этих случаях продукт морально стареет еще до завершения его разработки. А чтобы этого не случилось, качественная документация, как обязательное условие RAD-модели, обеспечит удобство эксплуатации и будущее сопровождения системы;

– *нечетко определены требования к ПО*. Это стандартная ситуация, когда заказчик очень слабо видит границы будущего программного продукта, и, как следствие, не может точно сформулировать свое видение. RAD-модель готова к такому повороту;

– *ограниченность бюджета*. Данное условие выполняется за счет RAD-подходов, указанных выше:

– небольшой командой разработчиков;

– в срок три-четыре месяца.

Это обеспечивает минимизацию трудозатрат и, в соответствии с договорными отношениями, нахождение в рамках треугольника ограничений, одной из сторон которого является стоимость;

– *разбиение проекта на функциональные компоненты – типовой случай*. Если предполагаемая система настолько велика, что невозможно уложиться в заданные сроки, то она разбивается на несколько частей. Каждая часть разрабатывается отдельно. Данные части общего продукта могут выпускаться этой же командой, либо возможно создание и привлечение несколько команд разработчиков.

Изменения принципов к разработке внесли изменение в список этапов данной модели жизненного цикла. Однако, несмотря на это,

она сохранила в себе свойство спиральности. Описание фаз модели также приведем на основе данных Википедии.

Фазы разработки (рис. 1.6):

– планирование;

– пользовательское проектирование;

– конструирование;

– переключение.

Рис. 1.6. Фазы разработки

Планирование – этап подготовки требований. На этом этапе пользователи и команда разработчиков обсуждают контуры проекта, его задачи, требования, включая требования к качеству. Это этап сбора и анализа требований. Важный момент – требования должны быть выполнимы в рамках короткого цикла разработки. Работы завершаются согласованием требований, договорными документами и переходом к пользовательскому проектированию.

Пользовательское проектирование – разрабатываются прототипы, которые выполняют требования. Прототипы преобразуются в рабочие модели на основе CASE-инструментов. Пользовательское проектирование хотя и является длительным интерактивным процессом, но дает возможность выбрать пользователям из альтернативных решений одно, отвечающее их требованиям. Например, изменение форм и видов сеансов диалога, форм выходных документов.

Конструирование – этап непосредственно разработки программного продукта. Этап аналогичен стадии «реализация» в классических моделях ЖЦПО. Пользователи также участвуют и могут подавать предложения по изменению в системе. В задачи конструирования входят разработка приложений, написание и тестирование кода.

Переключение – это переход на новую систему. Переход предполагает конверсию данных, необходимое обучение пользователей. В сравнении с традиционными методами разработки, цикл разработки более сжат во времени и программный продукт будет более быстро доставлен до заказчика и установлен на рабочих местах.

Современные условия внедрения разработки и программного обеспечения в технологический цикл предприятия требуют еще более сжатого цикла разработки. Следует отметить, что появлению новых подходов в методологии разработки способствовали следующие условия:

– более жесткие требования бизнеса к срокам, качеству разработки;

– развитие сред разработки, включение инструментов проектирования как составляющие в среды разработки;

– внедрение инструментов тестирования в процесс разработки, появление новых подходов (практик) разработки, где тестирование является неотъемлемой частью разработки;

– классические модели предлагали строго упорядоченные процессы, в которых планируется и прогнозируется весь объем работ. При этом основной участник – разработчик, человек с его недостатками, в расчет не принимался;

– все модели ориентировались на создание качественной документации, подготовку которой в условиях коротких циклов и изменяющихся требований становилось все труднее разработать.

Ответом на это стало появление метрологии гибкой разработки – Agile методологии, которая возникла на заре XXI века. Особенность и привлекательность гибких подходов – поиск компромиссов между строгой дисциплиной поведения и ее отсутствием. Цель – подбор разумного компромисса, чтобы получить эффективную отдачу от команды. Команда и каждый ее член становятся активными участниками планирования и выполнения работ. Другая важная особенность – отказ от документации в ее классическом звучании. Третья – очень короткие циклы разработки – от одной-двух недель до месяца и при этом планирование задач выполняет команда и это в условиях активного взаимодействия с заказчиком при постоянном пересмотре им требований.

Рассмотрим более подробно несколько процессов разработки в рамках методологии гибкой разработки с общим названием «Agile software development».

Agile – это целое семейство в методологии быстрой разработки.

8 февраля 2001 г. 17 специалистов собрались в штате Юта, чтобы обсудить вопросы применения простых и легких методик разра-

ботки. Результатом встречи стал «Манифест быстрой разработки». И его следует привести, так как в нем изложены основные принципы данного подхода. Текст манифеста доступен на более чем 50 языках (в том числе русском) и включает в себя 4 ценности, 12 принципов.

При описании манифеста будем опираться на книгу [2], так как, например, на просторах интернета можно найти и иные описания. Начнем с ценностей: «Ценности:

– люди и взаимодействие важнее процессов и инструментов;

– работающий продукт важнее исчерпывающей документации;

– сотрудничество с заказчиком важнее согласования условий контракта;

– готовность к изменениям важнее следования первоначальному плану».

Из текста видно, что ценности ставят во главу угла не формализацию процессов, а человека и взаимодействие, готовность к изменениям.

Сами принципы приведем непосредственно полностью, так как ее автор Роберт Сесил Мартин (Robert Cecil Martin), также известный как Дядя Боб – не просто консультант и автор в области разработки ПО. Именно он в 2001 г. организует встречу группы, которая создала гибкую методологию разработки со следующими принципами:

• Высшим приоритетом считать удовлетворение пожеланий заказчика посредством поставки полезного программного обеспечения в сжатые сроки с последующим непрерывным обновлением.

• Не игнорировать изменение требований, пусть даже на поздних этапах разработки. Гибкие процессы позволяют учитывать изменения и тем самым обеспечивать заказчику конкурентные преимущества.

• В процессе разработки предоставлять промежуточные версии ПО часто, с интервалом от пары недель до пары месяцев, отдавая предпочтение меньшим срокам.

• Заказчики и разработчики должны работать совместно на протяжении всего проекта.

• Проект должны воплощать в жизнь целеустремленные люди. Создайте им условия, обеспечьте необходимую поддержку и верьте, что они доведут дело до конца.

• Самый эффективный и продуктивный метод передачи информации команде разработчиков и обмена дискуссиями внутри ее – разговор лицом к лицу.

• Работающая программа – основной показатель прогресса в проекте.

• Гибкие процессы способствуют долгосрочной разработке. Заказчики, разработчики и пользователи должны быть в состоянии поддерживать неизменный темп сколь угодно долго.

• Непрестанное внимание к техническому совершенству и качественному проектированию повышает отдачу от гибких технологий.

• Простота – искусство достигать большего, делая меньшее, основа основ.

• Самые лучшие архитектуры, требования и проекты выдают самоорганизующиеся команды.

• Команда должна задумываться над тем, как стать еще более эффективной, а затем соответственно корректировать и подстраивать свое поведение.

Принципы определяют не только гибкость в планировании и разработке, но также выдвигают команду на первый план. Именно команда, ее мотивированность, со стремлением к совершенствованию в условиях неопределенного срока поддержки темпа разработки является стержнем методологии.

Agile методология не включает конкретных практических подходов (практик), но опирается на ценности и принципы. Здесь, как в RAD, выделяется практика работы небольших команд. Упор делается на личное общение. Поэтому желательно, чтобы команда была расположена в одном помещении. Еще одна особенность методологии – включение заказчиков или полномочных представителей (product owner) в состав команды. Именно product owner определяет требования к продукту. Включение заказчика в состав команды дает реальную возможность регулярного и частого общения с представителями бизнеса. Благодаря этому обе стороны (разработчики и бизнес) всегда в курсе событий. С Agile методологиями возникла концепция создания MVP – концепция создания минимального жизнеспособного продукта (minimum viable product). IT-компании от мелких до крупных используют данное понятие как исходную точку для создания успешного продукта. Это не просто быстрое создание версии продукта, но и анализ экономической целесообразности, стремление получить максимум информации о полезности продукта при минимальных усилиях. И здесь очень важен короткий срок разработки версии. Именно Agile методологии ориентированы на модель MVP. Это отказ от дорогостоящих исследований, поиска, разработки полных решений и переход к циклически создаваемым версиям, несущим конкретную ценность для пользователя. А сейчас, зная ценности и принципы, на которые опираются гибкие методологии, рассмотрим несколько известных:

– экстремальное программирование (eXtreme Programming, XP – автор Кент Бек, 1999);

– Scrum;

– Kanban.

Экстремальное программирование. Технология основана на последовательном развитии программного продукта и разработке его короткими итерациями. Основа подхода – короткие циклы (обычно 1–2 недели, в RAD – 2–3 месяца), частый выпуск релизов. Agile-ценности преобразуются здесь в небольшой набор конкретных правил:

– командная работа;

– разработка наиболее простых работающих решений;

– гибкое адаптивное планирование;

– оперативная связь с пользователем.

В соответствии с принципами Agile, XP ориентируется на небольшие команды, но что также важно, и на небольшие программные системы, в которых требования часто меняются.

В отличие от Agile, в XP есть основные практики. Это 12 основных приемов экстремального программирования, которые могут быть объединены в четыре группы [3]:

1) короткий цикл обратной связи (Fine-scale feedback), включающий:

– разработку через тестирование (Test-driven development);

– игру в планирование (Planning game);

– постоянное присутствие заказчика;

– парное программирование (Pair programming);

2) непрерывный, а не пакетный процесс:

– непрерывная интеграция (Continuous integration);

– рефакторинг кода;

– частые небольшие релизы (Small releases);

3) понимание, разделяемое всеми:

– простота проектирования (Simple design);

– метафора системы (Domain Driven Design – проблемно-ориентированное проектирование);

– коллективное владение кодом (Collective code ownership) или выбранными шаблонами проектирования (Collective patterns ownership);

– стандарт оформления кода (Coding standard or Coding conventions);

4) социальная защищенность программиста – 40-часовая рабочая неделя.

Помимо пунктов, которые в том или ином виде можно найти в принципах Agile, здесь мы можем увидеть новые моменты, изложенные в виде практик.

Разработка через тестирование. XP предполагает не прямое написание программного кода, а на основе автоматических тестов. В этом связка классических шагов (этапов) «Разработка–Тестирование» разворачивается и на первое место ставится шаг разработки теста.

Особое внимание уделяется двум видам тестирования:

– модульное(юнит)-тестирование;

– функциональное тестирование.

Важная особенность, ставшая правилом в современных подходах: разработчик кода приложения не может быть уверен в правильности написанного им кода до тех пор, пока не сработают все тесты разрабатываемого им кода. Модульные тесты позволяют разработчикам убедиться в том, что каждый отдельный класс, метод (процедура, функция) по отдельности и автономно работает корректно.

Модульные тесты необходимы всем разработчикам при изучении фрагментов модуля, особенно при сопровождении унаследованного кода. Функциональные тесты нужны для комплексного тестирования логики, образуемой взаимодействием нескольких частей, модулей. Тесты позволяют оценить назначение тестируемого кода, понять, как он функционирует и как используется. Наличие кода модуля в совокупности с кодами тестов для данного модуля является самой актуальной частью документации. При этом команды принимают решение об упрощении списка проектной документации, что снижает трудозатраты на разработку. Наличие актуальных тестов и кода программы упрощают и процесс рефакторинга.

Стандарты оформления кода. Важно отметить, что подход «тесты и код модуля – как основа документации» требует и практики соблюдения стандартов оформления кода. Этим обеспечивается эффективное выполнение остальных практик (например, ясность кода).

Парное программирование. Очень интересной практикой является практика парного программирования. Это когда один программист готовит код, а напарник изучает только что написанный код. Здесь преследуются две цели:

– код знают как минимум два программиста и неожиданное увольнение одного не приводит к стрессам в команде;

– напарник проводит ревизию кода и при этом достигается более качественное написание кода.

Коллективное владение. Если парное программирование распространить на всех членов команды, меняя пары, то возникает еще одно преимущество – так или иначе вся команда знает код приложения.

При этом каждый член команды ответственен за весь код и каждый может вносить изменения в программу. Такой подход создает дублирующие звенья и повышает стабильность процесса разработки, поскольку необходимость внесения изменений при развитии проекта или ошибках может быть выполнена любым членом команды.

Scrum. Scrum опирается на принципы гибкой методологии, но добавляет свои практики. Все вместе взятые принципы и практики Agile позволяют в короткие во времени жесткие циклы предоставлять конечному пользователю работающий программный продукт с новыми возможностями. Эти циклы в Scrum называются спринтами.

Строго фиксированная небольшая длительность спринта придает процессу разработки предсказуемость.

Дальнейший материал изложим, опираясь на термины из книги [4], автором является Джефф Сазерленд, также автор принципов гибкой разработки и автор Scrum.

Спринт-итерация. Это короткий цикл, в ходе которого создается модифицированная улучшенная версия программного продукта. Спринт фиксирован по времени, длительность одного спринта от 1 до 4 недель (Agile-принцип). Важно отметить, что:

– чем короче спринт (цикл), тем более гибким является процесс разработки и обратная связь с заказчиком;

– при более длительных спринтах команда имеет больше времени на решение возникших в процессе разработки вопросов, а заказчик может уменьшить время и ресурсы на совещания.

Поэтому команда подбирает длительность спринта согласно опыту, составу, а также требованиям заказчика.

Оценка объема работ. Для планирования спринта необходима оценка объема работ. И здесь нет жестких норм. Необходимо отметить, что существует проблема в оценке объема задачи:

– заказчик часто не знает точно, чего он хочет, и требования могут измениться. При этом он желает знать, когда и что будет готово;

– оценка занимает много времени и не дает точного результата. Существующие нормативы трудозатрат быстро устаревают в связи с развитием технологий;

– оценки часто делают люди, которые не выполняют задачу;

– оценка принимается за сроки, что не совсем так, это больше некоторая (неполная) характеристика трудозатрат и сложности, которую следовало бы сопоставить с некоторым аналогом.

Поэтому для оценки объема работ используется не абсолютная оценка, а относительная, измеряемая в очках, «попугаях» («38 попугаев»), баллах и т. п. Главный смысл здесь в сопоставимости оценок. На первых спринтах используется предварительная оценка. Данная оценка уточняется по мере продвижения проекта и является критерием определения объемов задач очередного спринта.

Ведение журналов. Выделяют журналы:

– журнал проекта (Project backlog);

– журнал спринта (Sprint backlog).

Журнал проекта – это список требований пользователей к функциональности продукта. Список должен быть упорядочен по степени важности задач, подлежащих реализации.

Элементы этого списка называются пользовательскими историями (user story).

Журнал спринта (Sprint backlog) – содержит список отобранных задач из журнала проекта. Разработчики просматривают объемный список задач из журнала проекта и определяют приоритетность, а затем выбирают те, которые будут выполнены за цикл спринта: от начала и до конца. Новые функции будут включены в выпускной релиз спринта.

История спринта (*Sprint Story*). Требуемую функциональность, которую добавляют в бэклог, часто называют историей. Зачастую история имеет следующую структуру:

Будучи пользователем <тип пользователя>
я хочу сделать
<действие>,
чтобы получить
<результат>.

Такая структура удобна тем, что понятна как разработчикам, так и заказчикам.

Диаграммы сгорания задач (Burndown chart). Согласно Википедии существуют два вида диаграммы (рис. 1.7):

– диаграмма сгорания для спринта – показывает количество сделанной и оставшейся работы в спринте;

– диаграмма накопления задач (Burnup chart) – количество сделанной и оставшейся работы для программного продукта. Диаграмма строится по данным нескольких спринтов.

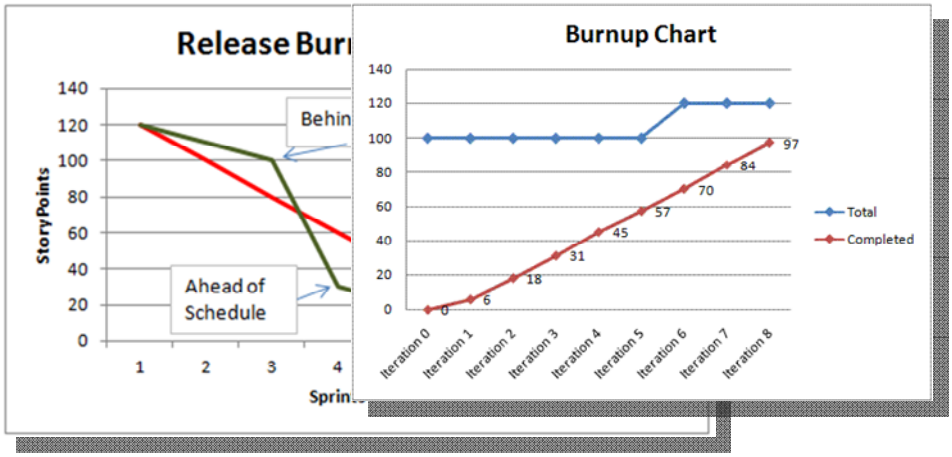


Рис. 1.7. Диаграммы сгорания и накопления задач

Остановка спринта (*Abnormal Termination*). Это аномальное действие по завершении работ. Инициатива может исходить: от команды, если достичь цели спринта невозможно; владельца проекта, если результаты спринта ему не интересны.

Так или иначе, после остановки спринта проводится совещание, обсуждаются причины возникновения исключительной ситуации и принимается решение о возможности продолжения.

Scrum-митинги. Организуются:

– при планировании (Sprint Planning Meeting);

– ежедневные.

Планирование спринта происходит в начале нового цикла:

– из бэклога проекта командой выбираются задачи, назначаются приоритеты с учетом важности задач;

– эти задачи записываются в бэклог спринта. Каждая задача оценивается в баллах, «попугаях»;

– команда определяет исполнителей задач.

Ежедневные митинги (Daily Scrum meeting):

– длятся не более 10–15 минут;

– проводятся в одном и том же месте;

– начинаются точно вовремя.

В течение совещания каждый член команды отвечает на три вопроса:

– Что я делал вчера?

– Что я буду делать сегодня?

– Какие у меня проблемы на пути решения задачи?

Канбан – японский термин, введенный компанией Toyota для организации своего конвейерного производства, дословно переводится

как «карточка», «вывеска». Хорошо подходит для поточного производства, где нет необходимости в планировании, процессы хорошо определены и необходимо объединить их в общий конвейер сборки.

Ключевыми понятиями Канбана являются:

– визуализация процесса;

– лимит задач для каждого этапа;

– выбор задач из списка;

– настройка и адаптация.

Визуализация. Для визуализации используются канбан-доски (рис. 1.8).

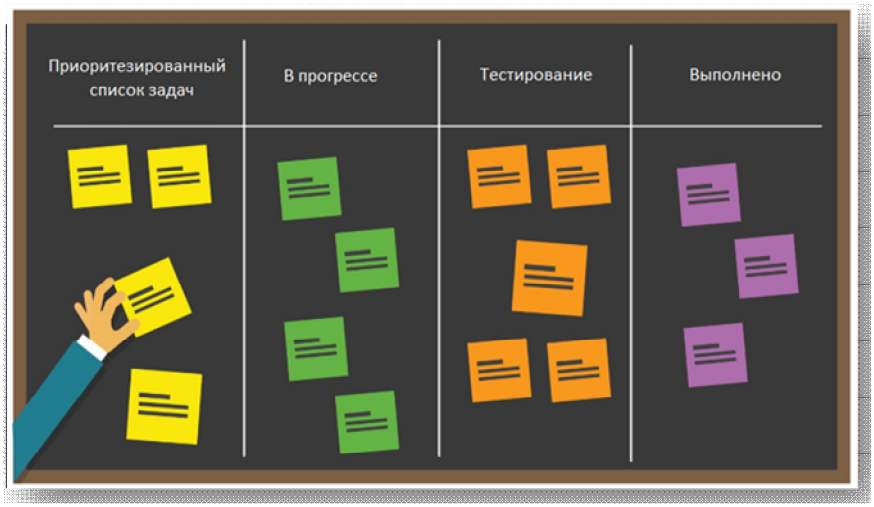


Рис. 1.8. Вид канбан-доски

Лимит задач. Лимит задач ограничен количеством сотрудников на этапе. Точнее, лимит равен количеству сотрудников на этапе (рис. 1.9).

Рис. 1.9. Лимит задач

Выбор задач из списка. Задачи по мере поступления заносятся в буфер задач, являющийся списком заданий для следующего процесса (рис. 1.10).

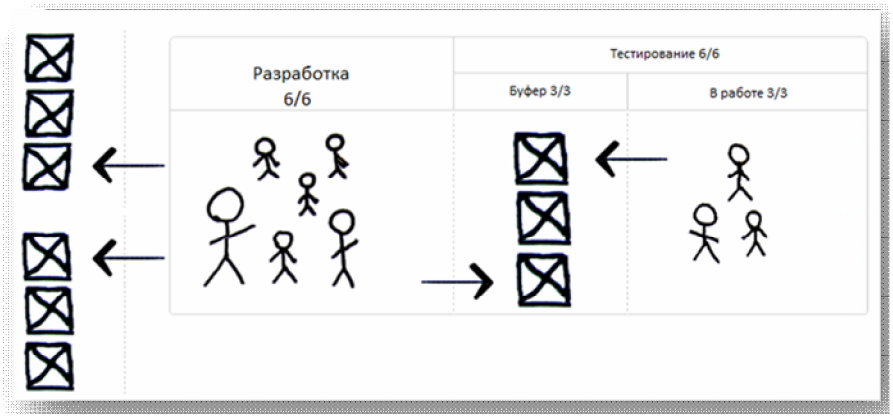


Рис. 1.10. Процесс извлечения задач из списка

Настройка и адаптация. Основная цель – сократить время выполнения задач. Для этого выполняется постоянный мониторинг, анализ и улучшение процесса, устранение препятствий, влияющих на процесс (рис. 1.11).

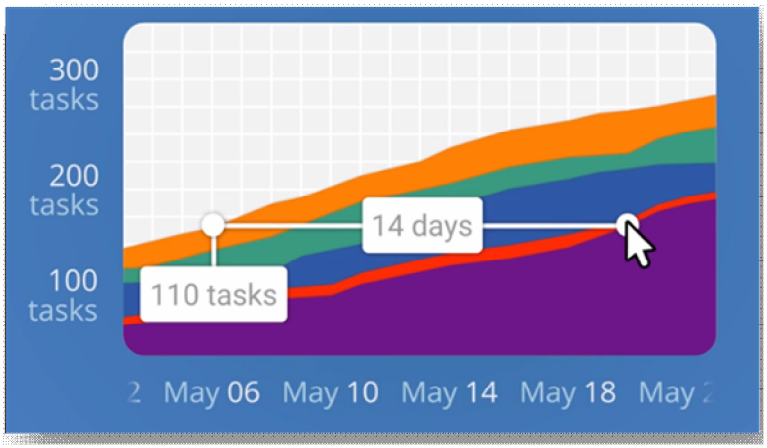


Рис. 1.11. Мониторинг задач

Главное достоинство гибких методологий заключается в стремлении быстрого внедрения программного продукта за счет обеспечения своевременной обратной связи.

Рассмотрим списки этапов классических моделей жизненного цикла. Это:

– разработка требований;
– проектирование;
– реализация;

– тестирование;

– ввод в действие.

Спиральная модель приводит к цикличности этапов. Развитие сред разработки и тестирования программного обеспечения, повышение требований к качеству разработки призывает необходимость объединения этапов реализации и тестирования. Спиральность и требования быстрой разработки, сокращение длительности этапов цикла, оперативная обратная связь с пользователями приводит к вынесению вопросов договорных отношений, пересмотра требований за рамки основных этапов. Кроме того, в рамках гибких методологий за рамки основных этапов выносится и задача проектирования.

В рамках основного производства остаются:

– планирование;

– разработка, включающая тестирование;

– ввод в действие.

В рамках практикума будем рассматривать следующий список:

1. Инициация.
2. Производство.
   2.1. Планирование.
   2.2. Разработка.
   2.3. Ввод в действие.

Пункты «Инициация» и «Производство» – это наши фазы. Подпункты пункта 2 – этапы.

Инициация. Для классических моделей работы этого этапа включают мероприятия по маркетингу, подготовке и участию в тендерах и конкурсах и другие мероприятия преддоговорной работы. На фазе инициации проекта формируется команда, ведется концептуальное планирование и проектирование будущего проекта. Горизонт планирования – срок заключения договора (контракта) на спиральный цикл.

Для гибких методологий на этом этапе готовится список требований, пожеланий заказчика, которые включаются в журнал проекта в виде пользовательских историй.

Если по завершению спирального цикла принимается решение о продолжении работ, то данная фаза повторяется. Но при этом у команды уже есть список пожеланий, собранный во время производственной фазы, в отличие от первой спирали.

Производство–Планирование. Для классических моделей на этом этапе определяются исполнители работ проекта, и формируется базо-

вый план. Горизонт планирования – предполагаемый срок окончания проекта. Этап включает:

1. Составление перечня необходимых работ. Здесь важен не только список работ, но и порядок их исполнения, так как многие из них связаны между собой.

2. Определение ресурсов и их назначение. С любой работой связаны определенные ресурсы: материальные (бумага, электронные носители информации), трудовые (исполнители, оборудование). При включении работы в план необходимо понимать: какие ресурсы и в каком объеме потребуются на выполнение данной работы. Суть назначения ресурса на работу состоит в том, что мы задаем не только ресурс, но и его потребность на данный объем работ. С ресурсами связана стоимость их использования. Следствием этого будет потребность в данном ресурсе не только в натуральном измерении, но и его стоимость в расчете на всю работу. Результатом планирования является подготовленный план проекта. План проекта позволяет получить ответы на вопросы:

– с помощью каких работ будет достигаться результат проекта;

– какие ресурсы (люди, оборудование, материалы) будут необходимы для выполнения работ;

– в какое время эти ресурсы будут востребованы в работах по проекту.

После согласования план становится основой базового плана, в который входят следующие показатели:

– содержание работ (объемы работ);

– сроки выполнения работ;

– стоимость, контролируемые показатели базового плана.

Для гибких методологий (для Scrum) это планирование спринта, описанное ранее.

Производство–Планирование. Процессы разработки и тестирования в основном одинаковы как для классических моделей, так и дли гибких методологий, если они основываются на единой базе инструментов.

Рассмотрим только мониторинг. Для классических моделей – это выполнение и контроль хода работ (мониторинг работ).

Мониторинг проекта начинается с момента фиксации базового плана проекта, заканчивается после выполнения обязательств сторон, участвующих в проекте, одной из которых является команда разработчиков во главе с менеджером.

На основе показателей контролируется ход работ на этапе мониторинга.

Этап мониторинга состоит в систематическом наблюдении и сборе фактических данных и сопоставлении оценок плановых и фактических показателей выполнения работ. Основные плановые показатели – это показатели треугольника ограничений.

Аналогичные показатели могут быть подготовлены не только для проекта в целом, но и по каждой работе, а если проект сложный и разделен на последовательные шаги реализации – фазы, то подобные показатели готовятся для каждого шага.

На основе сопоставления оценок плана и факта:

– выявляются отклонения в реализации проекта;

– прогнозируются последствия сложившейся ситуации;

– обосновывается необходимость принятия корректирующего воздействия.

Для гибких методологий, с длительностью всего 1–2 недели, процесс мониторинга упрощенный, основа – наглядные изобразительные средства, например «Диаграмма сгорания задач».

Производство – ввод в действие. Этап предполагает принятие решения руководством компании о судьбе проекта и продолжении работ.

Для классических моделей выполняются необходимые действия для закрытия работ. Это, в частности, подписание актов приемки/сдачи выполненных работ, подготовка и передача в соответствии с договоренностью проектной документации, лицензионных прав. При продолжении работ переход к фазе инициации.

Для гибких методологий: демонстрация версии заказчику, обсуждение, выпуск релиза. При продолжении работ переход к этапу планирования спринта.

1.3. Основы сетевого планирования

План проекта дает информацию о работах, ресурсах и их назначениях:

– задачи;

– ресурсы;

– назначения.

Задачей называется работа, выполняемая в рамках данного издания для достижения определенного результата. Ресурсы и назначения имеют смысл, указанный выше (§ 1.2). Поэтому далее будем считать,

что понятия «работа» и «задача» синонимичны в рамках пособия. Каждая задача характеризуется свойствами:

– название задачи;

– длительность. Длительность задачи – это промежуток рабочего времени, который нужен для выполнения задачи;

– начало (выполнения);

– окончание (выполнения).

Фазы, длительности. Обычно задачи из-за их многочисленности объединяют в группы, которые называют суммарными задачами или фазами. Полученная фаза может состоять из одной или нескольких задач для достижения основных результатов проекта. Для получения результата одной отдельной задачи выполняется только она одна, а для получения результата фазы выполняется группа задач – это и есть отличие фазы от задачи: суммирование результатов других задач. В фазы могут входить не только отдельные задачи, но и другие фазы. Для удобства проект разбивается на фазы. С целью поиска и исправления ошибок по окончании проектной фазы осуществляется анализ полученных результатов.

Вехи – задачи, при выполнении которых достигаются промежуточные цели проекта. Вехой принято выделять последнюю задачу фазы, в результате которой достигается ее промежуточный результат, при условии отсутствия такой задачи, но достижения фазового результата (одновременное завершение нескольких задач). Если такой задачи нет, а фазовый результат достигается, например, одновременным завершением нескольких задач, то создается фиктивная завершающая задача, длительностью 0 дней, без выделения исполнителей.

Зависимости и связи. В проекте задачи связаны между собой, это называется зависимостью задач, и эти зависимости обозначаются с помощью связей. Связи определяют последовательность выполнения работ в проекте: каким образом время начала или окончания одной задачи влияет на время окончания или начала другой.

Стандартно существует четыре типа связей:

– окончание–начало;

– начало–начало;

– окончание–окончание;

– начало–окончание.

По умолчанию назначается связь типа «*Окончание–Начало*».

Конец–Начало – последовательность, при которой предшествующая задача должна завершиться до начала последующей.

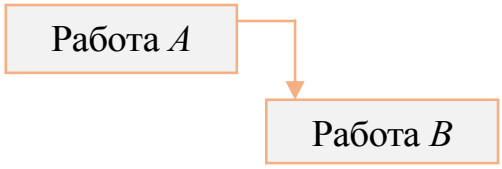


Рис. 1.12. Связь «*Конец–Начало*»

Например, на рис. 1.12 все связи вида «*Конец–Начало*».

Начало–Начало – последовательность, при которой задачи должны выполняться одновременно (не требуется завершения предшествующей работы до начала последующей, а нужно, чтобы предшествующая задача только началась (рис. 1.13)).

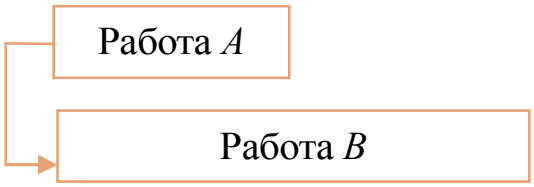


Рис. 1.13. Связь «*Начало–Начало*»

Конец–Конец – последовательность, при которой окончание последующей работы контролируется окончанием работы предшественницы, работы выполняются параллельно (рис. 1.14).

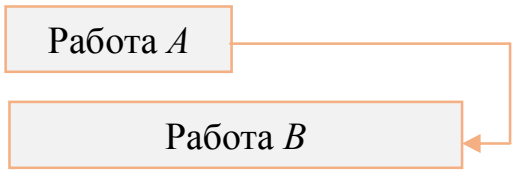


Рис. 1.14. Связь «*Конец–Конец*»

Таким типом связи объединяются те работы, которые выполняются одновременно, и каждая из них не может закончиться, пока не завершится другая.

Начало–Конец – последовательность, при которой требуется задержать окончание работы, при этом ее окончание связывается с началом другой работы (рис. 1.15).

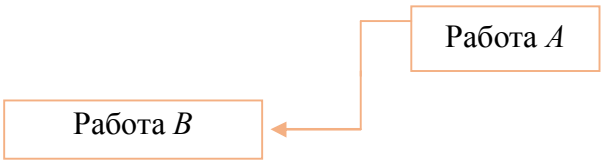


Рис. 1.15. Связь «*Начало–Конец*»

Этот тип связи применим в том случае, когда работа *A* является работой с фиксированной датой начала, а *B* не может закончиться до тех пор, пока не началась *A*.

Ресурсы. Под ресурсами понимаются:

– исполнители (сотрудники);

– материалы;

– оборудование;

– затраты, необходимые для выполнения проектных задач, при этом исполнители и оборудование относятся к трудовым ресурсам.

Ресурс «затраты» определяется тогда, когда использование его не может быть запланировано в задаче в натуральных единицах (времени, количестве и т. п.). Это ресурс не зависит от того, сколько времени задействован в задаче сотрудник или материальный ресурс (например, затраты на телефонные разговоры, почтовые переводы и т. п.), поэтому он планируется и учитывается только в денежном выражении.

Ресурс «материальный» позволяет планировать ресурсы, планируемые в количественном и стоимостном (цена за единицу) выражении. Это могут быть обычные материальные ресурсы (бумага, краска и т. п.). Для них в планировании помимо цены необходимо указать единицу измерения материала, например, для бумаги – пачка, для краски – банка или килограммы.

Трудовые ресурсы – это, в первую очередь в программных проектах, исполнители, а также оборудование.

Исполнители для проектов ИТ – является основной вид ресурсов, так как именно на данный вид ресурса приходятся затраты на выполнение работ. Оборудование может быть представлено по виду и количеству. Планирование и учет его во многом аналогичен ресурсам вида «исполнители».

При составлении списка ресурсов-исполнителей используется ролевое планирование. Каждый сотрудник, участвующий в проекте, получает определенную роль, соответствующую его квалификации. Например, в разработке ИТ-проекта могут участвовать разработчики баз данных, проектировщики, программисты, тестировщики и менеджер команды.

При этом, например, сначала определяется, что для исполнения работ требуются два программиста, разработчик базы данных, проектировщик, тестировщик и один менеджер, а затем, когда план проекта утвержден, подбираются конкретные сотрудники для этих ролей.

Важное свойство всех ресурсов – их стоимость, на основе которой рассчитывается стоимость всего проекта.

Назначения. Назначения – это связь определенной задачи с ресурсами, необходимыми для ее выполнения. На одну задачу может быть назначено несколько ресурсов. Назначения связывают задачи и ресурсы. На рис. 1.16 схематично показана данная связь, где каждое назначение однозначно связано с конкретной работой и конкретным ресурсом. Стрелки указывают на однозначность связи.

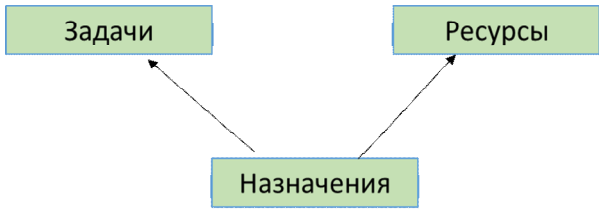


Рис. 1.16. Связь задач, ресурсов и назначений

Благодаря такой однозначности мы можем выбрать все ресурсы для конкретной задачи, либо все задачи, в которых конкретный ресурс используется. Назначения позволяют получить ответы на целый ряд вопросов:

– определить список ответственных за исполнение задачи. Это список ресурсов-исполнителей, связанных через назначения с данной задачей;

– определить список задач, в которых участвует конкретный исполнитель;

– рассчитать общий объем времени, затрачиваемый исполнителем на проект, выделить задачи, в которых он участвует одновременно, определить перегруженность ресурса, о чем сказано будет ниже;

– для любого конкретного ресурса выделить список задач, в которых он участвует, и суммировать время и стоимость его по каждой задаче;

– можно определить общую стоимость проекта. Получив стоимость ресурса по проекту, мы можем просуммировать стоимость всех ресурсов. Эта стоимость определяет третью вершину проектного треугольника (см. рис. 1.1).

Глава 2. РАЗРАБОТКА ПРОЕКТА

Итерации подразделяются на три этапа:
1) инициация;
2) производство:
   – планирование;
   – исполнение;
3) завершение.

2.1. Инициация

2.1.1. Участники и команда проекта

Участники проекта – это все заинтересованные лица и организации, которые активно вовлечены в проект, или чьи интересы могут быть затронуты в результате исполнения и завершения проекта. Среди них можно выделить ключевых участников:

1) заказчик;
2) исполнитель (подрядчик):
   – руководство исполнителя;
   – руководитель, он же менеджер проекта;
   – команда проекта.

Заказчик и исполнитель могут создавать собственные группы. Для классических моделей жизненного цикла важны два момента:

– наличие руководителя проекта;

– наличие куратора (со стороны заказчика).

В случае гибких методологий будем опираться на основных участников в рамках Scrum-подхода:

– скрам-мастер (Scrum Master) – проводит совещания, но не выдает задания, а выявляет и устраняет проблемы;

– владелец продукта (Product Owner) – формирует бэклог проекта, оценивает важность задач, является представителем стороны заказчика;

– команда разработки (Development Team) – является основной рабочей единицей, отвечает за результат.

Различиями в формировании участников являются:

– вовлечение представителя стороны заказчика в команду проекта в виде владельца продукта. В классической модели – изменение роли куратора и перевод его в команду;

– упразднение роли руководителя проекта с заменой ее на роль скрам-мастера.

Но команда проекта – это команда специалистов. Для классических моделей характерно выделение ролей в команде. При определении ролей будем опираться на описание команды в фирме IBM.

Специалисты (*Команда*). Роли и ответственности участников типового проекта разработки ПО условно разделены на пять групп:

1. Анализ. Извлечение, документирование и сопровождение требований к продукту.

2. Управление. Определение и управление производственными процессами.

3. Производство. Проектирование и разработка ПО.

4. Тестирование. Тестирование ПО.

5. Обеспечение. Производство дополнительных продуктов и услуг.

Группа анализа включает в себя следующие роли:

– менеджер продукта от Заказчика (функциональный заказчик). Представляет в проекте интересы пользователей продукта;

– бизнес-аналитик. Построение модели предметной области (онтологии);

– бизнес-архитектор. Разрабатывает бизнес-концепцию системы. Определяет общее видение продукта, его интерфейсы, поведение и ограничения;

– специалист по требованиям. Документирование и сопровождение требований;

– системный аналитик. Отвечает за перевод требований в функциональные требования к ПО.

Группа управления состоит из следующих ролей:

– руководитель проекта. Отвечает за достижение целей проекта при заданных ограничениях (по срокам, бюджету и содержанию), осуществляет управление проектом и выделенными ресурсами;

– системный архитектор. Разработка технической концепции системы. Принятие ключевых проектных решений относительно внутреннего устройства программной системы и ее технических интерфейсов;

– руководитель группы тестирования. Определение целей и стратегии тестирования, управление тестированием;

– ответственный за управление изменениями, конфигурациями, за сборку и поставку программного продукта.

Производственная группа включает в себя следующие роли:

– проектировщик. Проектирование компонентов и подсистем в соответствие с общей архитектурой, разработка архитектурно значимых модулей;

– проектировщик базы данных;

– проектировщик интерфейса пользователя;

– разработчик. Проектирование, реализация и отладка отдельных модулей системы.

Группа тестирования в проекте состоит из следующих ролей:

– проектировщик тестов. Разработка тестовых сценариев;

– разработчик автоматизированных тестов;

– тестировщик. Тестирование продукта. Анализ и документирование результатов.

Группа обеспечения, как правило, не входит в команду проекта. Она выполняет работы в рамках своей процессной деятельности. К группе обеспечения можно отнести следующие проектные роли:

– технический писатель;

– переводчик;

– дизайнер графического интерфейса;

– разработчик учебных курсов, тренер;

– участник рецензирования;

– продажи и маркетинг;

– системный администратор;

– технолог;

– специалист по инструментальным средствам и др.

В зависимости от проекта и условий его выполнения указанные роли могут совмещаться.

Для гибких методологий характерно:

– небольшой состав команды: 2–9 человек;

– взаимозаменяемость.

Эта особенность влечет иерархию команд, где на нижнем уровне – непосредственно команда разработки и тестирования продукта, которая включает и представителя заказчика, а на уровнях выше (особенно при больших проектах) – аналитики, проектировщики, архитекторы, интеграторы.

Для небольших проектов (Startup-проектов) команды во многом аналогичны Scrum-командам. Далее будем ориентироваться на этот вариант формирования.

2.1.2. Требования к проекту

Требования – важная и исходная составляющая в разработке программного обеспечения. Это, по факту:

– с одной стороны – пожелания заказчика, что он хотел бы получить;

– с другой – обязательства исполнителя выполнить эти пожелания.

Требования – это отправная точка проектирования, разработки и тестирования. Если в требованиях что-то «нето»и реализовано будет «не то», т. е. работа множества людей будет выполнена впустую.

Требования – это и основа проверки и передачи программного продукта заказчику. На основе требований, изложенных в техническом задании, формируется календарный план проекта, определяется его стоимость, организуются работы по разработке программного продукта. Требования – это и источник данных, позволяющий предотвращать конфликтные ситуации. Конечно, при этом важна точность формулировок, однозначность толкований. Поэтому в классических моделях жизненного цикла такая важная роль уделяется требованиям и подготовленным на основе требований спецификациям. Свои пожелания высказывают не только пользователи, роли которых могут быть разнообразны, но и их руководители, заказчик.

Поэтому традиционно выделяются:

– бизнес-требования;

– пользовательские требования.

Бизнес-требования – это бизнес-цель, ради которой разрабатывается продукт. Это желание заказчика, как с помощью программы он будет получать бизнес-преимущества.

Такие требования изначально могут выглядеть следующим образом:

– необходим инструмент для автоматического приведения кодировок текстовых документов к одной;

– необходима информационная система ведения каталога достопримечательностей.

Пользовательские требования – это требования к будущим автоматизированным задачам, которые пользователь может выполнять с помощью разрабатываемой системы. В простом виде это реакция системы на действия пользователя, с необходимым для него результатом.

Указанные два уровня требований описывают поведение системы на уровне, понятном пользователю.

Также могут быть подготовлены дополнительные материалы, поясняющие поведение программы:

– диаграммы взаимодействия;

– сценарии использования.

Требования необходимы для оценки:

– объема работ;

– стоимости проекта;

– времени разработки.

На основе требований производится первичное планирование разработки. Для менеджера это основа планирования и определения сторон «треугольника ограничений».

При формулировке и анализе требований выделятся функциональные и нефункциональные требования.

Функциональные требования – описывают поведение системы, которое определил пользователь в своих пожеланиях. Это учет бухгалтерских и производственных данных, проверка, обработка и формирование отчетов.

К нефункциональным требованиям традиционно относятся:

– удобство использования;

– безопасность и надежность;

– модифицируемость и расширяемость;

– масштабируемость и устойчивость, которыми программный продукт должен обладать.

Список нефункциональных требований дополняет функциональные требования. Приведем примеры.

Требования к надежности:

– сохранность данных обеспечивается штатными средствами СУБД, регламентами компании, включая разграничение прав доступа;

– целостность данных обеспечивается штатными механизмами ссылочной целостности СУБД и специализированными хранимыми процедурами;

– корректность вводимых данных обеспечивается через механизмы контроля, реализованные в ФПО системы;

– разграничение доступа к функциональным режимам осуществляется на основе Администратора комплексов и авторизации пользователя при входе в систему.

Требования к программной совместимости. Внедряемые программные комплексы подсистемы должны интегрироваться:

– с программными продуктами массового использования Microsoft Office (MS Word, Excel) через механизмы экспорта; импорт данных из этих продуктов может быть осуществлен при наличии согласованных форматов передаваемых данных либо при наличии модели данных в задокументированном виде;

– с ранее внедренными программными продуктами через механизмы экспорта/импорта; при этом должны быть согласованы форматы передаваемых данных.

Список форматов импорта/экспорта должен быть согласован на этапе опытной эксплуатации.

В классических моделях жизненного цикла выделяется еще один уровень – требования к продукту, или спецификация. Эта формализация первых двух уровней до вида, понятного команде при разработке программы.

Спецификации. Каждая спецификация должна содержать информацию, достаточную для того, чтобы разработчик смог однозначно и полно реализовать требование, не обращаясь за разъяснениями и уточнениями к автору.

Такой подход позволит руководителю проекта, пробежавшись по спецификациям, сформировать из них набор «заготовок» для составления подробного плана-графика проекта.

Подобная формализация отсутствует в гибких моделях разработки. Основа построения требований – пользовательские истории. Для этого, как изложено выше, ведется журнал проекта. В журнал проекта прописываются пожелания пользователей. Они прописываются в виде пользовательских историй в форме:

Будучи пользователем<тип пользователя>
я хочу сделать
<действие>,
чтобы получить
<результат>.

Каждая история связана с ролью пользователя в приложении. В отличие от требований в классических моделях здесь допускаются «вольности» в записи, история может быть очень неопределенной. Но она не будет реализована, если команда не решит, что ее можно реализовать за один спринт и включит ее в журнал спринта. Если история сложна или неопределенна, она пересматривается, уточняется и разбивается на подзадачи. Только после этого она может попасть в журнал спринта. Такой подход означает, что пересмотр истории лежит за рамками работы команды либо команда своими силами выполняет разбиение в рамках одного или нескольких спринтов.

2.1.3. Практическое задание 1

1. Разбиться на группы. Каждая группа – команда проекта.

Команда получает задание в виде цели на разработку диалогового приложения. Роли в команде:

• ***Классический вариант***

Руководитель проекта. Вместе с командой отвечает за планирование сроков и объемов. Отвечает за достижение целей проекта при заданных ограничениях (по срокам и содержанию), осуществляет управление проектом. Совмещает роли:

– системный архитектор.

Менеджер продукта (куратор, владелец продукта) – представляет интересы Заказчика. Совмещает роли:

– функциональный заказчик;

– дизайнер графического интерфейса.

Разработчик. Совмещает роли:

– проектировщики (архитектуры, базы данных, интерфейсов);

– реализация и отладка модулей системы;

– автономное тестирование.

Тестировщик. Тестировщик совмещает роли:

– разработка тестовых сценариев;

– функциональное тестирование;

– анализ и документирование результатов.

Примечание. По согласованию с преподавателем команда может предложить свое распределение ролей.

• ***Гибкая разработка***

Scrum-мастер. Вместе с командой отвечает за планирование сроков и объемов. Отвечает за достижение целей проекта при заданных ограничениях (по срокам и содержанию).

Владелец продукта – представляет интересы заказчика. Совмещает роли:

– функциональный заказчик;

– дизайнер графического интерфейса.

Разработчик. Совмещает роли:

– проектировщики (архитектуры, базы данных, интерфейсов);

– реализация модулей системы;

– автономное и функциональное тестирование.

Примечание. По согласованию с преподавателем команда может предложить свое распределение ролей.

2. Получить задания от преподавателя. Разрешается предложить свой вариант с предварительным согласованием с преподавателем.

3. На основе задания сформулировать несколько пунктов требований (пользовательских историй).

4. Команда выполняет статическую проверку требований (пользовательских историй), формируется журнал проекта.

5. Результатом практического задания являются требования (пользовательские истории), журнал проекта.

Контрольные вопросы

1. Роли в команде при классической модели разработки.
2. Роли в команде при модели гибкой разработки.
3. Техническое задание и его назначение.
4. Журнал проекта, пользовательские истории – назначение, отличие от технического задания.

2.2. Планирование

2.2.1. Планирование задач выпуска

Для планирования задач воспользуемся методологией Scrum-разработки. В рамках данной методологии руководитель проекта выступает как Scrum-мастер. Пункты технического задания (спецификации) определим как журнал пожеланий проекта (Project backlog). Для планирования спринта определим журнал пожеланий спринта (*Sprint backlog*) – содержит функциональность, выбранную менеджером продукта из журнала пожеланий проекта. Длительность одного спринта фиксируем от одной до четырех недель. Команда выбирает из списка требование с учетом заданных приоритетов. Если требование, размещенное в журнале пожеланий, не может быть реализовано за длительность спринта, оно должно быть пересмотрено и детализировано. Данный пункт пересматривается вне рамок задач, размещенных в журнале пожеланий спринта. Если команда, выбрав задачу из журнала пожеланий проекта, не выполняет ее за время спринта, то она возвращается в журнал пожеланий проекта, для дальнейшего уточнения пункта требования, разбиения его на подзадачи, размещения их в журнале пожеланий проекта или снятия данного требования из процесса разработки продукта.

Для выбранных задач из списка журнала пожеланий проекта команда определяет их объемы, выполняет назначение, планирует их выполнение на этапе спринта. Планируются задачи разработки, тестирования, управления рабочими версиями спринта и выпуска релиза. Принятие решения о готовности релиза выполняется по окончании спринта. В релиз могут быть включены только выполненные задачи. Решение о готовности выполняется всеми участниками разработки (команда, пользователи, представители заказчика).

2.2.2. Практическое задание 2

Знакомство с инструментами планирования. Для автоматизированного планирования работ команда использует инструмент планирования ProjectLibre либо предлагает собственные варианты, согласованные с преподавателем.

В главе 3 «Автоматизированное планирование работ» в целях быстрого знакомства с инструментом планирования ProjectLibre и его освоения пошагово описано выполнение двух заданий.

В случае применения собственных вариантов средств (доски Kanban, Scrum) команда выбирает список работ из раздела «Автоматизированное планирование работ» и готовит план работ на основе его.

Контрольные вопросы

1. Инструмент планирования ProjectLibre.
2. Инструменты планирования и учета задач в гибких разработках.

2.2.3. Практическое задание 3

На основе подготовленных пунктов требований:

• Менеджер продукта совместно с преподавателем задают приоритеты реализации требований из журнала проекта.

• Команда:

– выбирает из списка требования с учетом заданных приоритетов;

– формирует список задач итерации на основе выбранных требований;

– определяет их объемы, выполняет назначение исполнителей (из команды), конкретизирует сроки;

– на основе списка задач формируется чек-лист тестирования [5] задач проекта.

Результатом практического задания являются:

– список работ по проекту, введенный в инструмент планирования;

– план работ по реализации задач версии проекта;

– чек-лист тестов для версии проекта.

Контрольные вопросы

1. Планирование работ над версией.
2. Журнал спринта проекта.
3. Чек-лист спринта.

2.3. Разработка релиза

2.3.1. Реализация задач итерации

Команда уточняет состояние проекта и оставшиеся объемы, которые нужно выполнить в рамках итерации для выпуска релиза. Производится:

– разработка кода рабочей версии продукта;

– подготовка тестов, тестирование кода;

– выполняется обновление рабочей версии проекта.

Конечная версия итерации (релиз) также фиксируется в Git для ее оценки и принятия решения о следующей итерации.

При реализации задач используются:

– система контроля версий – Git. Описание системы контроля версий Git приведено в разделе «Управление версиями»;

– выбранный командой инструмент тестирования.

Из известных и доступных пакетов тестирования предлагается использовать:

– MSTest. Пакет юнит-тестирования от компании Microsoft;

– Nunit. Распространенный пакет unit-тестирования, доступный не только в рамках среды Visual Studio, но и для приложений на Java;

– xUnit. Пакет unit-тестирования, учитывающий достоинства и недостатки пакетов MSTest и Nunit.

2.3.2. Практическое задание 4

1. Установка и настройка инструментов:

– создание и настройка Git-репозитория проекта;

– создание приложения с двумя проектами:

– простого приложения с одним методом, например, калькулятор с операцией сложения;

– тест-проекта для простого приложения. Для тест-проекта – установка пакета тестирования (MsUnit, nUnit, xUnit).

2. Реализовать проекты.

3. В завершение разработки простого приложения – создать версию в репозитории Git.

4. Для проверки приложения тестами загрузить версию проекта из репозитория и выполнить его тестирование.

Контрольные вопросы

1. GIT – как инструмент управления версиями.
2. Инструменты тестирования и их возможности.
3. Автоматизированное тестирование версии проекта.

2.3.3. Практическое задание 5

1. Подготовка решения с тремя пустыми проектами приложения (три слоя):

– слой GUI;

– бизнес-слой;

– слой доступа к данным.

2. Также формируется проект для тестирования приложения.

В соответствии с ролями:

– команда реализует задачи;

– владелец продукта – консультирует, уточняет (при необходимости) дизайн пользовательского интерфейса. После окончания спринта готовится отчет о результатах тестирования. Оценивается готовность релиза на основе тестов;

– руководитель проекта – контролирует сроки и объемы. Вместе с командой принимает решение о готовности релиза.

3. Вся команда ответственна за результат. Для контроля хода работ команда использует инструменты планирования и контроля (предлагая собственные варианты, согласованные с преподавателем).

4. По окончании итерации команда:

– предъявляет готовый релиз проекта;

– тестовый проект проверки релиза;

– отчет о выполненных работах и возможностях.

5. Дополнительно, по указанию преподавателя, может быть подготовлен отчет о реализации и покрытии тестами исходных требований. В отчет не включаются требования, удаленные из журнала итерации.

6. После завершения итерации обсуждаются результаты, принимается решение о возможности подготовки и выпуска новой версии.

Контрольные вопросы

1. Управление версиями на основе GIT.

2. Структура трехслойного приложения. Разделение и назначение слоев.

3. Интеграционное тестирование проекта.

4. Анализ готовности релиза.

5. Сбор трудозатрат.

Глава 3. АВТОМАТИЗИРОВАННОЕ ПЛАНИРОВАНИЕ РАБОТ

3.1. Инструмент планирования – ProjectLibre

ProjectLibre – программное средство, которое помогает пользователю создавать и редактировать задачи, а затем назначать каждую из них различным участникам.

Приложение позволяет быстро управлять и контролировать каждый проект и его задачи в интуитивно понятной среде, просматривая даты начала и окончания терминала и сводные элементы проекта. При первом запуске программы пользователь может выбрать статус проекта и интересующий его вид расходов. Также можно просмотреть основную информацию о базовой продолжительности и фактической стоимости проекта.

После добавления задач в программу можно изменить опцию просмотра проекта на Гант, Сеть, WBS или Использование задач. Не зависимо от выбранного пользователем представления для анализа проекта ProjectLibre отображает всю запрашиваемую информацию в организованном порядке (диаграмма Ганта отображает график проекта и позволяет пользователю просматривать отношения между всеми действиями). Все указанные диаграммы используются для отображения текущего состояния расписания с использованием заливки в процентах, исходя из этого при выполнении следующей задачи без выполненной предыдущей начать не получится.

Параметр «Сеть» позволяет просматривать каждую задачу, организованную в отдельной таблице, и отображает такую информацию, как продолжительность, дата начала и окончания.

ProjectLibre позволяет открывать существующие файлы, выполненные в Microsoft или Primavera. Созданный проект можно распечатать в формате PDF или создать подробный отчет.

Несмотря на то, что ProjectLibre не содержит расширенных и запутанных параметров, он гибко реагирует на потребности каждого, когда речь идет о планировании и управлении различными проектами.

3.2. Задание на проектирование

Рассмотрим следующий пример: ставится задача перед руководителем службы ИТ, в подчинении которого есть небольшой штат разра-

ботчиков программного обеспечения: подготовить предложения по разработке приложения для предприятия, позволяющего автоматизировать рутинные действия специалистов-пользователей в оперативном учете. Приложение должно:

– содержать удобный интерфейс взаимодействия специалиста при подготовке данных, формировании отчетности;

– обеспечить хранение данных в базе данных.

Также требуется подготовить предложения по автоматизации, позволяющие получить представления о сроках и объемах; проект необходимо начать с первого января 2019 года.

Менеджер проекта (руководитель ИТ), как специалист, понимающий информационные бизнес-процессы предприятия и имеющий опыт разработки программных проектов, понял, что необходимо:

– иметь хорошую постановку задачи;

– разработать интерфейс взаимодействия с пользователем;

– разработать структуру базы данных;

– из документов на бумажных носителях выбрать и ввести в базу данных начальные данные (остатки некоторых ресурсов, справочные данные и т. п.);

– выполнить отладку разработанного приложения;

– на основе введенных данных в базу данных выполнить тестирование;

– подготовить документацию по проекту.

3.3. Структурное планирование задач проекта

3.3.1. Краткие сведения о работе

В результате должны получить навыки создания проекта, настройки его календаря, ввода перечня работ и задания их параметров.

Ключевыми параметрами планирования проекта являются:

– длительность;

– методика планирования;

– рабочее время.

Проект можно планировать двумя способами: от даты начала проекта или от даты окончания.

Если же у проекта нет жесткой даты окончания, то при планировании применяется первый способ: фиксируется дата, когда необходимо начать выполнение проекта. В этом случае в ходе составления плана

определяется дата его завершения на основании данных о длительностях задач проекта.

Если же у проекта есть жесткая дата окончания, то при планировании применяется второй способ: фиксируется дата, когда необходимо завершить выполнение проекта.

В этом случае в ходе составления плана определяется дата его начала.

Фиксация метода планирования, даты начала или даты конца проекта доступна в окне «Информация о проекте» ProjectLibre (рис. 3.1).

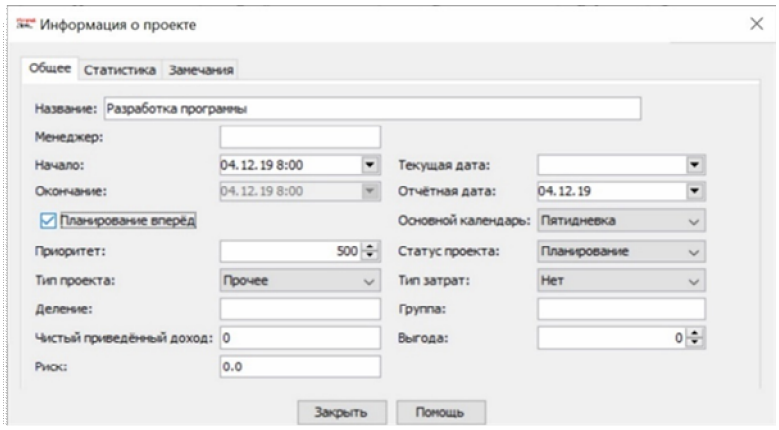


Рис. 3.1. Информация о проекте

Есть возможность переопределить две даты:

– текущую дату;

– дату отчета.

При мониторинге проекта и вводе сводной информации используется дата отчета, а не текущая дата. По умолчанию программа считает, что все выполненные трудозатраты относятся ко времени до даты отчета, а все оставшиеся трудозатраты – ко времени после даты отчета. Если выполнение задачи запланировано после даты отчета, но вносятся данные о фактических трудозатратах по этой задаче, то ProjectLibre изменит выполненную часть задачи так, чтобы она закончилась к дате отсчета, а выполнение оставшихся трудозатрат остается на будущее.

При подготовке проекта используется понятие «рабочее время». Это время, в рамках которого будут выполняться работы, а для этого используется понятие календаря. Календарем в ProjectLibre называется набор параметров, определяющих перечень рабочих и нерабочих дней, а также рабочее время в каждом из рабочих дней. В ProjectLibre доступны три вида календарей:

– пятидневка – расписанию с 8-часовым рабочим днем;
– 24 часа – круглосуточный рабочий день;
– ночная смена – круглосуточный с перерывами.

В раскрывающемся списке Основной календарь (рис. 3.2) можно выбрать один из этих доступных календарей.

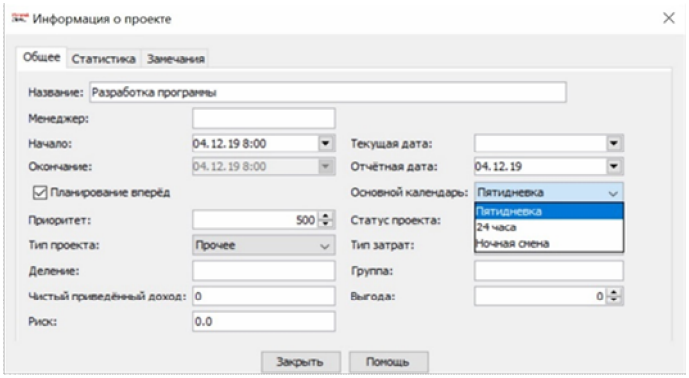


Рис. 3.2. Изменение вида календаря

Иногда возникает необходимость применения календаря, которого нет в списке. В этом случае можно изменить существующий календарь или создать новый. В ProjectLibre существует возможность создавать как групповые, так и личные календари. Поэтому при создании базового календаря в него следует вносить только настройки, общие для всех участников проекта или группы, к которой относится календарь. Специфические настройки заносятся в личный календарь каждого сотрудника.

В поле *Приоритет* указывается число в диапазоне от 0 до 1000, которое используется при выравнивании загрузки ресурсов между разными проектами. Чем больше число, тем выше приоритет проекта. Для одного проекта данная характеристика не принципиальна и в рамках настоящего издания не рассматривается.

Подготовка иерархического списка работ. На основе задания менеджер подготовил список задач для ввода в ProjectLibre:

– постановка задачи;
– разработка интерфейса;
– разработка структуры баз данных;
– разработка модулей обработки;
– заполнение баз данных;
– отладка ПК;
– тестирование ПК;
– составление документации.

Также ему, как опытному ИТ-специалисту, стало ясно, что весь процесс создания необходимо разделить на этапы программирования и отладки (с вводом в действие созданного приложения). Кроме того, подготовленный список хорош, но в нем отсутствуют данные, позволяющие понять, в какие сроки будет выполнена разработка проекта. Этот процесс сложный, зависит от опыта и умения менеджера, а также от списка его команды, участвующей в разработке.

Так как руководитель службы компетентен (будем так считать) и команду определяет он сам, им был подготовлен список задач с ориентировочными длительностями работ, приведенный в табл. 3.1. Длительности необходимы при расчете в ProjectLibre сроков выполнения как отдельных работ, так и всего проекта в целом.

Задачи ProjectLibre объединяются в фазы – группы задач (суммарные задачи), если они состоят из списка подзадач. Информация о длительности фазы суммируется на основе списка подзадач.

Будем считать, что каждая последующая фаза должна начинаться после завершения предыдущей. В этом случае мы должны указать в начале каждой фазы ссылку на последнюю задачу предыдущей фазы, если она однозначно определяет ее завершение.

В нашей таблице определено две фазы. Кроме того, добавим задачу начала проекта и задачу его завершения – для более удобного использования в ProjectLibre. Для устранения неоднозначности завершения задач *Программирование* и *Отладка* введены задачи *Программирование завершено*, *Отладка завершена*.

Важной особенностью проектирования последовательности исполнения является установление связи между задачами. В ProjectLibre связь задачи устанавливается по отношению к предшествующим ей задачам. Поэтому менеджер готовит перечень задач со ссылкой на предшествующие задачи.

Кроме того, если нет конкретной завершающей задачи (их несколько), то возможно добавить задачи с нулевыми длительностями, в нашем случае:

– программирование завершено;

– отладка завершена.

В ProjectLibre задачи нумеруются глобально, т. е. порядковый номер задачи «Программирование» будет 2. Мы это также учтем в табл. 3.1.

Таблица 3.1

Перечень задач проекта с их длительностями и предшественниками

Номер задачи	Название задачи	Предшествен-ники	Длительность, дней
1	Начало реализации проекта	–	–
2	Программирование:	–	–
3	– постановка задачи	1	10
4	– разработка интерфейса	3	5
5	– разработка модулей обработки	4	6
6	– разработка структуры БД	3	7
7	– заполнение БД	6	8
8	– программирование завершено	4; 6	–
9	Отладка:	–	–
10	– отладка ПК	8	5
11	– тестирование ПК	10	10
12	– составление документации	10	5
13	– отладка завершена	11; 12	–
14	Конец проекта	13	–

Из приведенной таблицы видно, что введенная завершающая задача «Программирование завершено» не сможет начаться, пока не завершатся задачи с номерами 5 и 7 («Разработка модулей обработки», «Заполнение БД»).

Подготовленный перечень является основой представления задач в ProjectLibre в виде диаграммы Ганта (рис. 3.3) и информации о сроках задач проекта.

Диаграмма Ганта является представлением ProjectLibre, отображающим свойства задач. В левой части экрана представлена таблица со свойствами задач, в правой части – временная ось с отложенными на них отрезками задач.

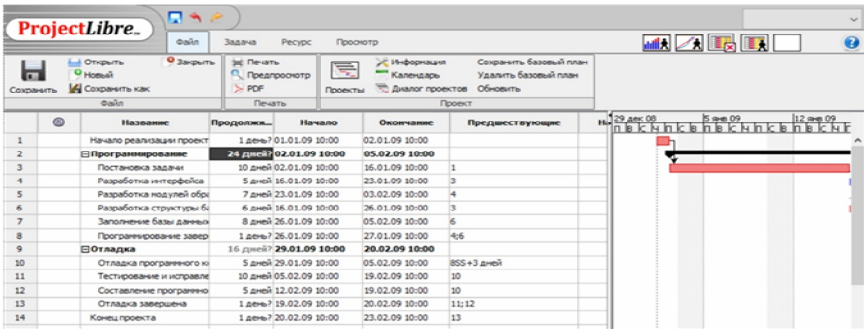


Рис. 3.3. Диаграмма Ганта

На диаграмме Ганта для каждой задачи отображается таблично и графически информация о ее длительности, сроках и последовательности исполнения в рамках перечня задач, составляющего содержание проекта.

Зная начальные сроки, длительности работ и их последовательность, мы можем определить конечные сроки. И наоборот, зная конечные сроки, длительности работ и их последовательность, мы можем определить начальные сроки. Даты начала и конца проекта определяют его длительность.

3.3.2. Порядок выполнения

Задачи планирования:

– освоить функциональные возможности программного средства ProjectLibre по построению сетевой модели проекта;

– применить на практике теоретические положения календарно-сетевого планирования проекта: сформулировать проект и его основные задачи, установить рабочее время проекта, организовать этапы выполнения задач проекта, установить собственные характеристики задач проекта.

Применение планирования работ проекта. Запускаем ProjectLibre. Появляется основное окно программы (рис. 3.4).

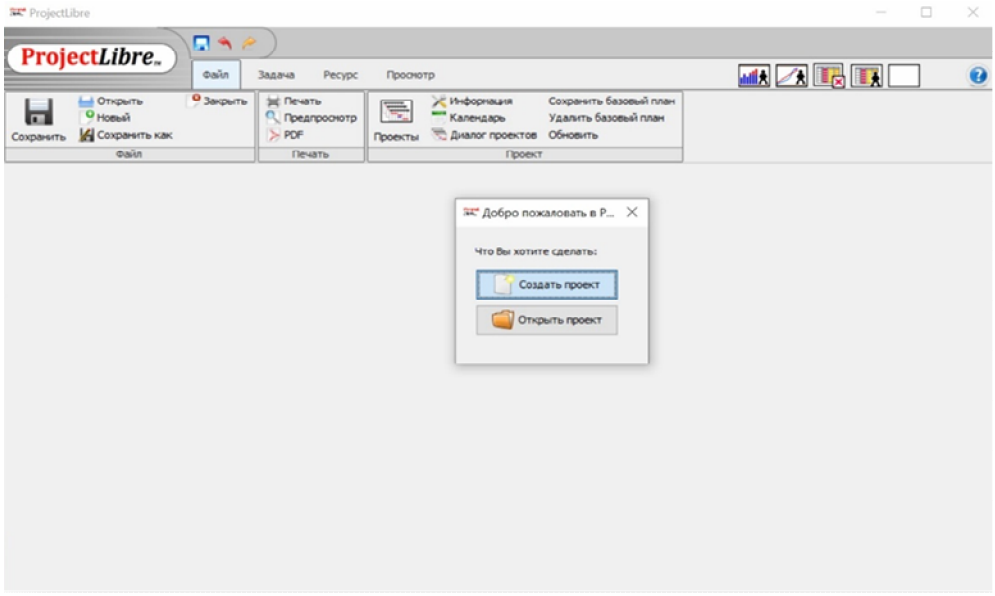


Рис. 3.4. Окно программы ProjectLibre

Создаем новый проект, даем ему название Разработка программы. Указываем дату начала проекта – 01.01.2019 (рис. 3.5).

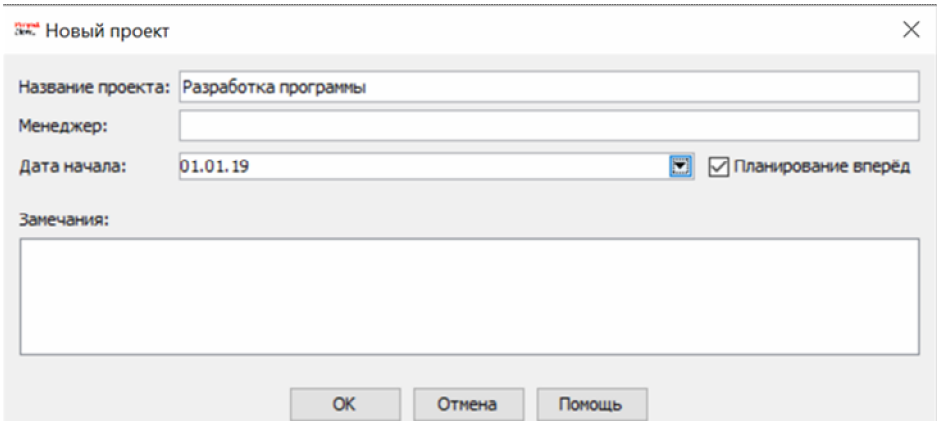


Рис. 3.5. Окно программы ProjectLibre

При всех последующих сохранениях проекта при помощи пункта меню *Файл/сохранить* проект автоматически записывает в уже имеющийся файл без открытия диалога сохранения.

Настройка календаря. Открыть окно календаря – *Задача/Календарь* (рис. 3.6). Задаем нерабочие дни: Новый год, Рождество, 8 марта и т. д. (выбираем день в календаре и нажимаем на панели *Нерабочее время*).

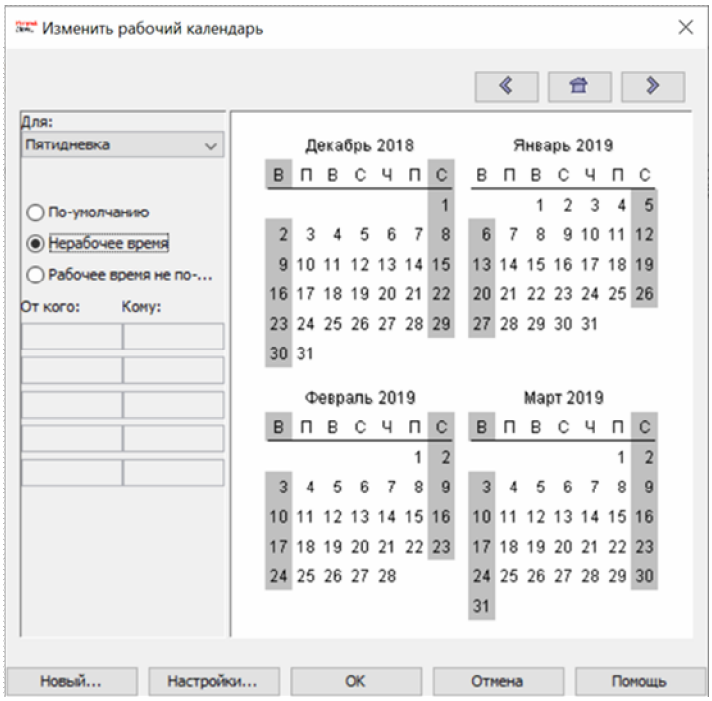


Рис. 3.6. Результат ввода исключений

Ввод перечня задач проекта. Составляем список задач проекта, содержащий вехи, фазы и обычные задачи. Располагаем задачи таким образом, чтобы их порядок соответствовал последовательности выполнения, а после каждой фазы должны быть перечислены входящие в нее вехи и задачи. Для создаваемого проекта *Разработка программы* список задач приведен в табл. 3.1.

В столбец *Название задачи* последовательно вводим названия задач из табл. 3.1. По умолчанию все введенные задачи являются обычными задачами длительностью 1 день. Знак вопроса в столбце *Длительность* означает, что она не была задана пользователем и является предварительной.

В столбце *Длительность* устанавливаем для вех длительность в 0 дней. Результат – на диаграмме Ганта эти задачи изображены ромбиками. Результат ввода задач проекта изображен на рис. 3.7.

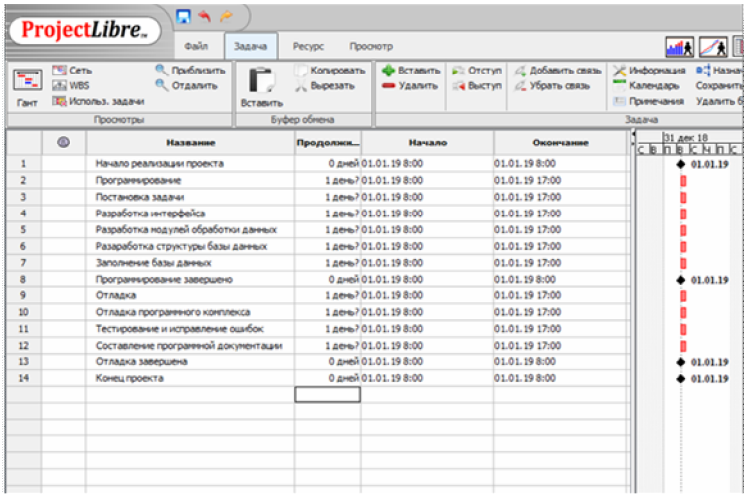


Рис. 3.7. Результат ввода перечня задач

Преобразование задачи в фазу. Для преобразования задачи в фазу все подзадачи этой фазы должны следовать в таблице непосредственно после нее. Удерживая нажатой левую кнопку мыши в области номеров задач, выделяем строки задач с номерами 3–8 и нажимаем кнопку Отступ на панели инструментов *Задача*. Результат – выделенные задачи становятся подзадачами, входящими в *Программирование*, а само *Программирование – фазой*, т. е. составной задачей. На диаграмме Ганта фаза изображается отрезком в виде горизонтальной скобки. Аналогично поступаем с задачами с номерами 10–13. Результат совпадает с изображением на рис. 3.8.

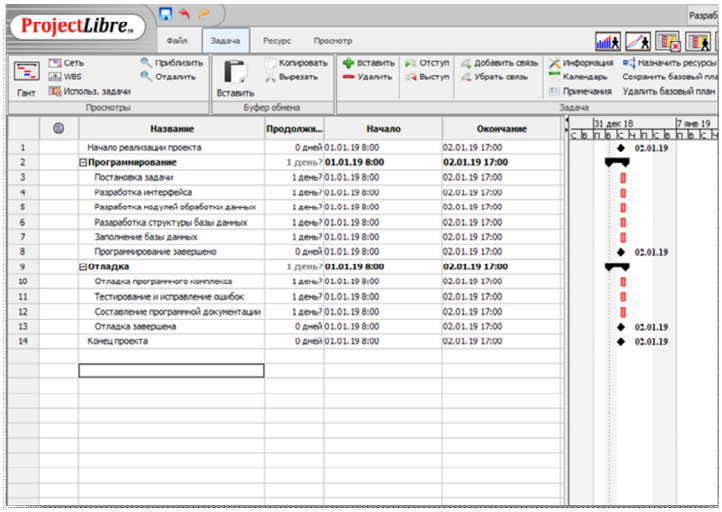


Рис. 3.8. Результат преобразования задачи в фазу

Создание связи. Создавать связи можно различными способами:

– при помощи мыши;

– в окне сведений о задаче;

– при помощи столбца *Предшествующие*.

Рассмотрим наиболее простой из них: создание связи при помощи столбца *Предшествующие*.

В таблице проекта находим столбец *Предшествующие*. В ячейку этого столбца строки задачи *Разработка модулей обработки данных* вводим номер задачи – предшественника 4. В результате установлена связь *Разработка интерфейса* / *Разработка модулей обработки данных*, изображенная на рис. 3.9.

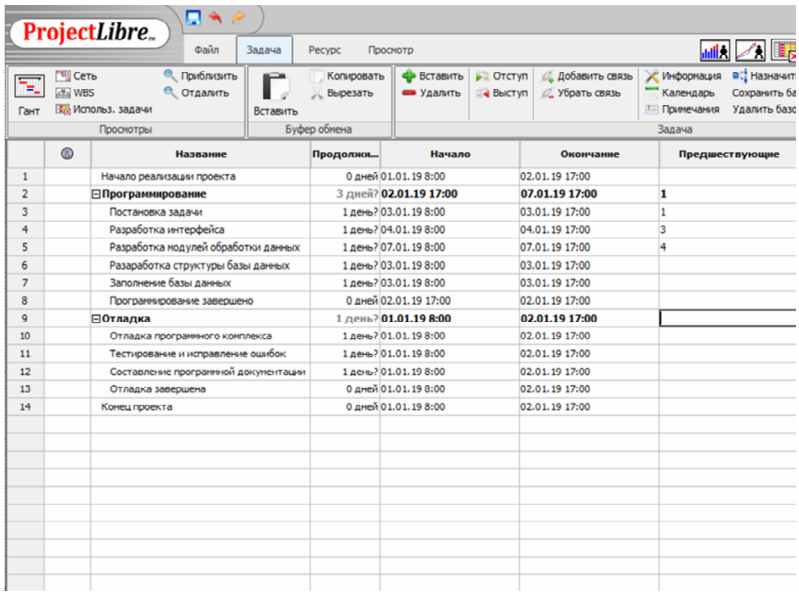


Рис. 3.9. Создание связи через столбец *Предшествующие*

Создаем остальные связи проекта *Разработка Программы* проекта в соответствии с табл. 3.1.

Типы связей, задержки, опережения и ограничения. Двойной щелчок мыши по строке задачи *Тестирование и исправление ошибок* в таблице. В открывшемся окне сведений о задаче выбираем вкладку *Предшествующие*. В строке предшественника *Отладка программного комплекса* изменить значение поля *Тип* на *Начало-начало* (*SS*), а в поле *Задержка* устанавливаем 3 дня.

Двойной щелчок мыши по строке задачи *Составление программной документации*. В открывшемся окне сведений о задаче выбираем вкладку *Дополнительно*. В поле *Тип* выбираем ограничение *Как можно позднее*. Результат преобразований изображен на рис. 3.10.

		Название	Продолжи...	Начало	Окончание	Предшествующие
1		Начало реализации проект	1 день?	01.01.09 10:00	02.01.09 10:00	
2		**Программирование**	**24 дней?**	**02.01.09 10:00**	**05.02.09 10:00**	
3		Постановка задачи	10 дней	02.01.09 10:00	16.01.09 10:00	1
4		Разработка интерфейса	5 дней	16.01.09 10:00	23.01.09 10:00	3
5		Разработка модулей обра	7 дней	23.01.09 10:00	03.02.09 10:00	4
6		Разработка структуры ба	6 дней	16.01.09 10:00	26.01.09 10:00	3
7		Заполнение базы данных	8 дней	26.01.09 10:00	05.02.09 10:00	6
8		Программирование завер	1 день?	26.01.09 10:00	27.01.09 10:00	4;6
9		**Отладка**	16 дней?	**29.01.09 10:00**	**20.02.09 10:00**	
10		Отладка программного к	5 дней	29.01.09 10:00	05.02.09 10:00	8SS+3 дней
11		Тестирование и исправле	10 дней	05.02.09 10:00	19.02.09 10:00	10
12		Составление программно	5 дней	12.02.09 10:00	19.02.09 10:00	10
13		Отладка завершена	1 день?	19.02.09 10:00	20.02.09 10:00	11;12
14		Конец проекта	1 день?	20.02.09 10:00	23.02.09 10:00	13

Рис. 3.10. Результат преобразований

Ввод длительности задач. Ввод длительности задач выполняется в столбце *Длительность* таблицы диаграммы Ганта. Задаем остальные длительности задач проекта *Разработка Программы* в соответствии с табл. 3.1.

3.4. Определение ресурсов и назначений

3.4.1. Краткие сведения о работе

Подготовка листа ресурсов. Менеджером проекта является руководитель службы ИТ и распоряжается он только своими ресурсами. В его распоряжении есть программисты, а также работающий неполный рабочий день постановщик задач. Постановщик работает неполную рабочую неделю, что влияет на сроки проекта. Кроме того, на проект руководитель службы ИТ выделил двух программистов.

Основная специализация первого – интерфейсы взаимодействия с пользователем и обработка данных. Основная специализация второго – базы данных и взаимодействие с ними. Так как разработка ведется в рамках специализированной программной среды, то для решения задач проектирования требуются консультации с разработчиками данной среды.

Менеджер полагает, что необходимо обеспечить сохранность всех версий проекта на любом его этапе. Предположим, что это решается на основе копий текущего состояния на устройства долговременного хранения и это *CD*-матрицы, не позволяющие перезаписать данные.

Учет ресурсов необходим не только для расчета загрузки каждого ресурса в проекте, но и для расчета одного из основных показателей «железного треугольника» – стоимости проекта.

Менеджер располагает ресурсами, представленными в табл. 3.2.

Таблица 3.2

Список ресурсов проекта

Номер ресурса	**Ресурс**	**Тип**	**Примечание**
1	Постановщик задачи	Трудовой	Работает неполную рабочую неделю
2	Программист 1, разработчик интерфейсов	Трудовой	–
3	Программист 2, разработчик баз данных	Трудовой	–
4	*CD*-матрицы	Материальный	–
5	Междугородные переговоры	Затраты	–

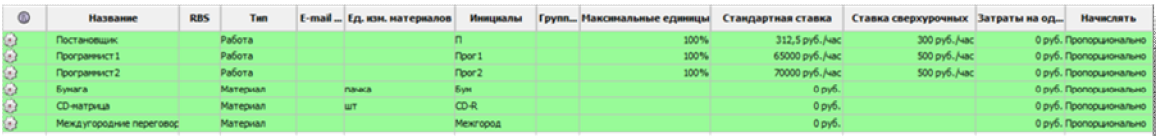
Краткие пояснения к таблице:

1. *CD*-матрицы относятся к материальным ресурсам и затраты на них начисляются на основе стоимости за единицу (цена) и количества использования. Задержка в выполнении многих проектов часто объ-ясняется нехваткой материалов. Если известно, что может возникнуть недостаток наличия материалов и это может сказаться на проекте, они должны быть включены в сетевой план проекта и должен быть составлен график поставок материалов.

2. Междугородные переговоры – это ресурсы типа «Затраты» и списываются суммарно, на основе прогноза.

3. Постановщик задач и программисты относятся к трудовым ресурсам.

В ProjectLibre основные данные о ресурсах описываются в листе ресурсов (рис. 3.11).



	Название	RBS	Тип	E-mail ...	Ед. изм. материалов	Инициалы	Групп...	Максимальные единицы	Стандартная ставка	Ставка сверхурочных	Затраты на од...	Начислять
	Постановщик		Работа			П		100%	312,5 руб./час	300 руб./час	0 руб.	Пропорционально
	Программист1		Работа			Прог1		100%	65000 руб./час	500 руб./час	0 руб.	Пропорционально
	Программист2		Работа			Прог2		100%	70000 руб./час	500 руб./час	0 руб.	Пропорционально
	Бумага		Материал		пачка	Бум			0 руб.		0 руб.	Пропорционально
	CD-матрица		Материал		шт	CD-R			0 руб.		0 руб.	Пропорционально
	Междугородние переговор		Материал			Межгород			0 руб.		0 руб.	Пропорционально

Рис. 3.11. Лист ресурсов

Лист ресурсов является представлением ProjectLibre, отображающим свойства ресурсов. Каждая строка для конкретного ресурса описывает его стандартные свойства.

В нем мы видим, что в проекте используются следующие ресурсы:

– постановщик;

– программист 1;

– программист 2;

– *CD*-матрица;

– бумага;

– межгород.

При этом первые три ресурса относятся к типу *работа*; *бумага*, *межгород* и *CD-матрица* – к материальным, это означает, что затраты на его использование производятся простым указанием суммы списания.

Мы также видим, что для материальных ресурсов в графе *Стандартная ставка* указана стоимость единицы ресурса. При этом в графе *Единицы измерения материалов* указаны:

– для бумаги – пачка;

– для *CD*-матрицы – шт. (штука).

Для ресурсов *работа* должна быть указана его привязка к базовому календарю – ссылка на привязку данного ресурса к стандартному календарю. Ставка указана в сумме оплаты за месяц:

– стандартная ставка позволяет рассчитать затраты на трудовой и материальный ресурсы по этой ставке, при этом для материального ресурса это цена за единицу;

– в графе «*Ставка сверхурочных*» – ставка оплаты сверхурочных при решении об оплате при часах переработки, учитываемых как дополнительное время к стандартному. Ставка указана в сумме к часу (почасовая).

Остальные графы пояснены ниже, в рамках описания сведений о ресурсе.

Перечисленные характеристики ресурса, а также дополнительные отображаются в окне сведений о ресурсе. Окно сведений о ресурсе имеет пять вкладок (рис. 3.12): общее, стоимость, доступность ресурса, задачи, замечания.

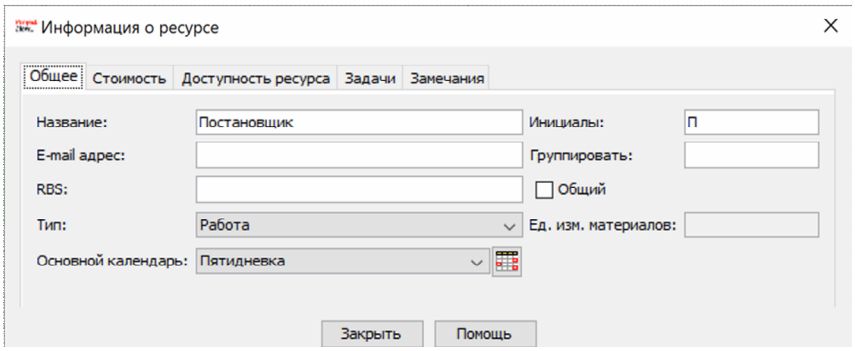


Рис. 3.12. Окно «Информация о ресурсе», вкладка «Общее»

Вкладка «*Общее*» включает информацию:

– название ресурса. Это полное наименование ресурса;

– краткое название ресурса;

– *группировать*. Наименование группировки, необходимой для проведения анализа, например, группа основных средств, группа тестирования;

– тип. Тип ресурса (материальный, работа).

На вкладке *Доступность ресурса* находится таблица для ввода информации по доступности ресурса. Вводится промежуток времени и процент доступности.

Вкладка *Стоимость* позволяет описать для каждого ресурса пять различных видов норм, описываемых через вкладки от «A» до «E». Это означает, что в нашем проекте мы можем использовать до пяти разных норм, при этом мы также можем изменить сроки действия конкретной нормы (рис. 3.13).

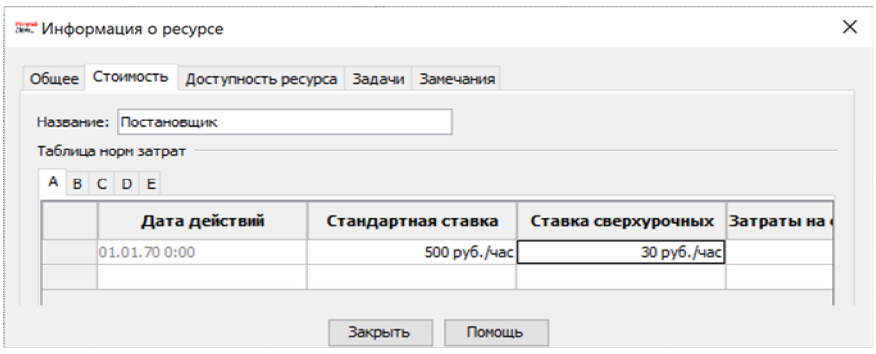


Рис. 3.13. Сведения о ресурсе, вкладка «Затраты»

Любая из данных норм затем может быть использована при назначении ресурса на задачу, но по умолчанию назначается норма «A». Например, трудовой ресурс «Петров» может быть использован как разработчик программы по норме «A» и как тестировщик – по норме «B».

Изменение норм затрат для конкретной задачи выполняется в представлении *Использование задачи* – окно *Сведения о назначении*, где для конкретного ресурса мы выбираем данную норму в свойстве *Таблица норм затрат* (нормы «A», «B», «C», «D», «E»).

Каждый вид нормы задается параметрами:

– стандартная ставка;

– ставка сверхурочных;

– затраты на использование;

– начисление затрат.

Для планирования стоимости ресурсов в ProjectLibre предусмотрено два типа оценки:

– на основе стандартной ставки;

– на основе затрат на использование.

Стандартная ставка выражается в стоимости использования ресурса за единицу времени. Обычно такая ставка используется для учета ресурсов, если известна, например, почасовая или помесячная ставка для исполнителей.

Для ввода основных ставок и ставок сверхурочных трудовых ресурсов эти данные могут быть получены от экономических служб. Предполагаем, что данные предоставлены.

Затраты на использование указываются для трудовых и материальных ресурсов:

– для трудового ресурса (люди и оборудование) значение в поле «Затраты на использование» является величиной затрат, которые начисляются при каждом использовании ресурса. Эта величина увеличивается при каждом назначении единицы ресурса задаче и величина не зависит от количества времени, в течение которого используется ресурс;

– для материального ресурса значение в поле *Затраты* на использование является величиной затрат, которые начисляются единовременно, независимо от числа единиц.

Свойство *Начислять* определяет момент, когда затраты на ресурс начисляются, и придает гибкость в определении момента начисления:

– в начале выполнения задачи;

– в конце ее выполнения;

– пропорционально – в процессе выполнения.

Наиболее применяемый метод – пропорционально. Но для ресурсов типа материальные (например, малоценные и быстроизнашиваемые предметы (МБП)) может быть списание как в начале выполнения работы, так и по ее завершении.

Назначение ресурсов. Мы имеем лист ресурсов. Для расчета стоимости необходим следующий шаг – назначение ресурсов на задачи.

Назначение ресурсов заключается в определении необходимого количества ресурсов для каждой задачи, не относящейся к суммарной. Решение менеджера на данном этапе:

– выбрать задачу из списка, на которую необходимо назначить ресурс;

– выбрать ресурс и выполнить назначение.

Назначение ресурса состоит в указании:

– для трудовых ресурсов – загруженности его на задаче;

– для материальных – количества расхода материального ресурса.

Пусть мы назначаем ресурсы на задачу *Постановка задачи* (рис. 3.14).

1		Начало реализации проект	1 день?	01.01.09 10:00	02.01.09 10:00
2		**⊟Программирование**	**24 дней?**	**02.01.09 10:00**	**05.02.09 10:00**
3		Постановка задачи	10 дней	02.01.09 10:00	16.01.09 10:00

Рис. 3.14. Фрагмент иерархического списка работ со строкой «Постановка задачи»

Менеджеру ясно, что постановкой задачи занимается постановщик. Поэтому мы должны именно его назначить на данную задачу. Если для постановщика мы введем в графе «Единицы» значение 100 %, то это означает, что на данной задаче постановщик будет занят все отведенное ему время.

В графе *Стоимость* отобразится начисление затрат ресурса на данную задачу, автоматически рассчитанных ProjectLibre. Аналогично выполняется назначение всех ресурсов.

Результатом назначения ресурсов будет подготовлена первая версия плана работ. Данный план дает всю необходимую информацию о проекте. Иерархический список работ дает нам объемы работ

(содержание), даты начала и завершения проекта определяют его сроки и длительность, а стоимость его ресурсов определяет стоимость проекта. И если: эти три параметра проекта (объемы, сроки, стоимость) уклады-ваются в треугольник ограничений; заказчик и руководитель нашего предприятия (в данном примере – это одно и то же лицо) не возражают против параметров и не требуют сокращения сроков, стоимости; ресурсы не перегружены (например, никто из людских ресурсов не работает сверх положенного времени), – то данный план может реально стать конечным (базовым), на основе которого будут выполняться работы.

Но это чаще всего лишь черновик, первое приближение к будущему плану работ. Перед менеджером возникает необходимость приложить дополнительные усилия по доработке плана и его оптимизации. Следующие три пункта позволяют без существенной доработки плана получить варианты, возможно конечные.

Назначение профиля ресурса. При назначении ресурса на задачу по умолчанию устанавливается профиль *Пропорциональный*. Это означает, что начисление затрат на выполняемую работу производится равномерно, в соответствии с затраченным временем. Но в планировании возникают ситуации, когда ресурс назначен на две параллельно выполняемые (в какой-то момент) задачи. И при этом возникает перегрузка ресурса. Решить данную проблему можно, изменив профиль ресурса на каждой задаче. Профили, доступные в ProjectLibre:

– плоский;
– загрузка в начале;
– загрузка в конце;
– ранний пик;
– поздний пик;
– двойной пик;
– «колокол»;
– «черепаха».

Большое разнообразие профилей позволяет оптимизировать загрузку конкретного ресурса, не выходя за пределы проектного треугольника. Например, если две работы, в которых участвует ресурс, исполняются параллельно на небольшом временном участке и избежать этого нельзя, не нарушая наш треугольник, то можно загрузку ресурса оптимизировать на данном участке, задав разные профили для ресурса: профиль *Загрузка в начале* для задачи, которая

начинается раньше, и *Загрузка в конце* для задачи, которая начинается позже. При этом мы предполагаем, что ресурс на каждой из задач не перегружен, а назначение профилей снижает общую перегрузку. Изменение профиля загрузки приводит к автоматическому перераспределению трудозатрат. Трудозатраты – это не длительность работы и могут не соответствовать длительности задачи, так как трудозатраты соответствуют времени, затраченному ресурсом на получение результата. Трудозатраты могут быть изменены, а длительность работы при этом может не изменяться. Изменение профиля возможно в рамках представления *Использование задачи*.

Изменение графика трудозатрат. Изменение графика трудозатрат позволяет решить ту же задачу оптимизации, что и выше, но используя «ручное» распределение трудозатрат. Это более тонкий, но также и более трудоемкий инструмент. В этом случае для конкретной задачи и конкретного ресурса можно назначить измененные трудозатраты по любым дням оптимизируемой задачи. Изменение графика трудозатрат возможно в рамках представления *Использование задач*.

Изменение норм затрат в назначении. Изменение норм затрат позволяет решать задачу оптимизации на основе перехода на другие нормы для заданного ресурса. Если мы для ресурса назначаем меньшие по ставке нормы, то тем самым предполагаем, что данная работа может быть выполнена за меньшие затраты (например, требует меньшей квалификации). Если же мы для ресурса назначаем большие по ставке нормы, то соответственно предполагаем, что данная работа требует больших затрат (например, требует большей квалификации). При этом мы не рассматриваем вопросы перегрузки ресурса, так как для нас здесь важно именно привлечение конкретного вида ресурса.

Но что нужно знать: в листе ресурсов должна быть прописана данная норма, на которую необходимо перевести ресурс.

3.4.2. Порядок выполнения

Ввод списка ресурсов. Выбираем пункт меню *Ресурс*/*Ресурсы* (рис. 3.15).

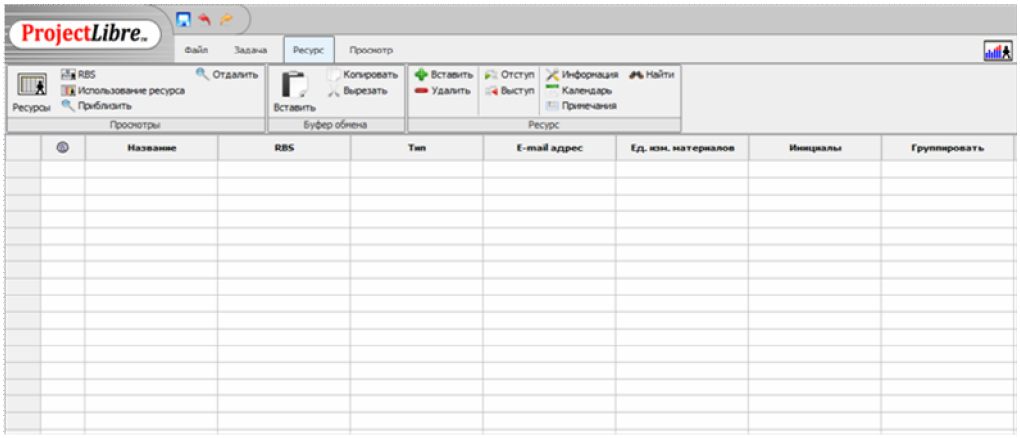


Рис. 3.15. Окно ввода списка ресурсов

Щелкаем мышью по полю *Название* первой пустой строки и вводим название *Постановщик*. Аналогично добавляем в таблицу ресурсы *Программист1* и *Программист2*. Добавляем в таблицу ресурс *Бумага* и *CD-матрица* и выбираем для них тип *Материальный*. Добавляем в таблицу ресурс *Междугородные переговоры* и выбираем для него тип *Затраты*.

Ввод свойств ресурса Постановщик. Дважды щелкаем мышью по строке *Постановщик* таблицы ресурсов. В открывшемся окне выбираем вкладку *Общие* (рис. 3.16). В поле *Инициалы* вводим *П*.

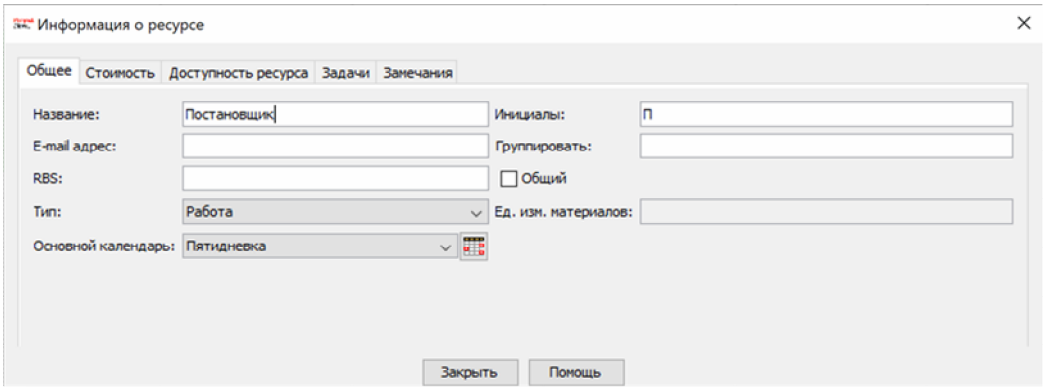


Рис. 3.16. Окно ввода свойств ресурса *Постановщик*

В таблицу доступности ресурса вкладки *Доступность ресурса* вводим три строки:

доступен с – НД; доступен по – 27.01.19; единицы – 100 %; доступен с – 28.01.19; доступен по – 18.02.19; единицы – 100 %; доступен с – 19.02.19; доступен по – НД; единицы – 100 %.

Устанавливаем четырехдневную рабочую неделю. Задаем нерабочие дни: пятница (выбираем день в календаре и нажимаем на панели *Нерабочее время*).

Выбираем вкладку *Стоимость*. В столбец *Стандартная ставка* первой строки вводим 500 руб./мес., а в столбец *Ставка сверхурочных* – 30 руб./час. Предположим, что с 01.02.19 зарплата данного работника должна быть увеличена. Во второй строке задать дату начала действия новой ставки – 01.02.19, стандартная ставка – 600 руб./мес., ставка сверхурочных – 35 руб./час (рис. 3.17).

Выбираем таблицу *норм затрат B* и устанавливаем затраты на использование – 250 руб. Эту норму можно использовать для работ с фиксированной суммой оплаты работника.

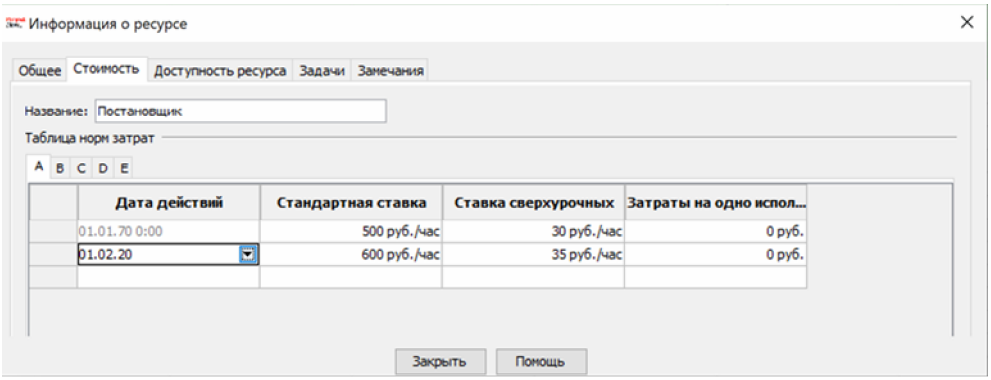


Рис. 3.17. Окно ввода затрат

Ввод свойств ресурса Программист1. В поле *Краткое название* вводим *Прог1*. Выбираем вкладку *Стоимость*. Предположим, что 01.02.19 оплата этого ресурса будет увеличена. Мы должны свести ее строки в таблицу норм затрат A:

– дата действия – стандартная ставка – 650 руб./мес., ставка сверхурочных – 35 руб./час;

– дата действия – 01.02.19, стандартная ставка – 700 руб./мес., ставка сверхурочных – 40 руб./час.

Выбираем таблицу *норм затрат B* и устанавливаем затраты на использование – 350 руб. Эту норму можно использовать для работы фиксированной суммой оплаты работника.

Ввод свойств ресурса Программист2. В поле *Краткое название* вводим *Прог2*. Выбираем вкладку *Стоимость*. В столбец *Стандартная ставка* вводим 700 руб./мес., а в столбец *Ставка сверхурочных* – 40 руб./час. Выбираем таблицу *норм затрат B* и устанавливаем затраты на использование – 350 руб.

Ввод свойств ресурса Бумага. В поле *Краткое название* вводим *Бум*, а в поле *Единицы измерения – пачка*. Выбираем вкладку *Стоимость*. В столбец *Стандартная ставка* вводим 10 руб.

Ввод свойств ресурса CD-матрица. В поле *Краткое название* вводим *CD-R*, а в поле *Единицы измерения* – штука. Выбираем вкладку *Стоимость*. В столбец *Стандартная ставка* вводим 1 руб.

Ввод свойств ресурса Междугородные переговоры. В поле *Краткое название* вводим Межгород.

Ввод назначений для задач проекта. Выбираем пункт меню *Просмотр/Гант*. Двойной щелчок мыши по строке *Постановка задачи*. В открывшемся окне свойств задачи выбираем закладку *Ресурсы*. В таблицу ресурсов добавляем записи: *Постановщик* – 250, *Бумага* – 1, *CD-матрица* – 1 штука/н, *Междугородные переговоры* (рис. 3.18).

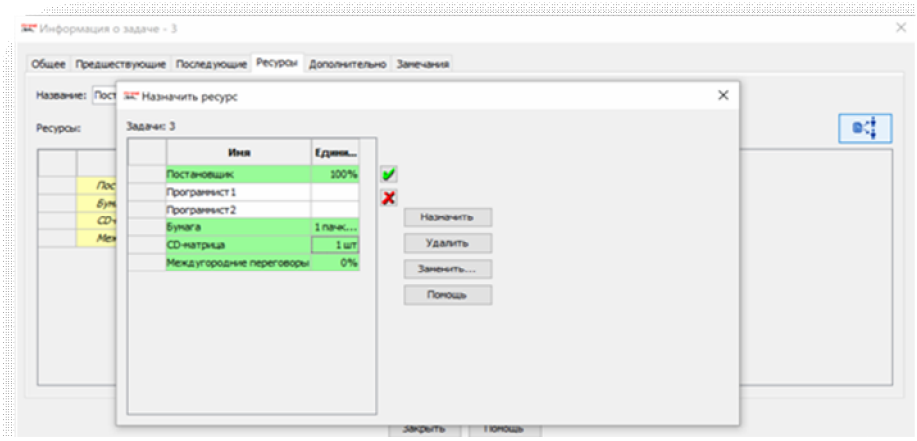


Рис. 3.18. Окно ввода назначений для задач проекта

Остальные ресурсы вводим согласно табл. 3.3.

Таблица 3.3

Исходные данные для ввода ресурсов проекта

Номер задачи	Название задачи	Ресурсы	Единицы, %	Затраты, руб.
1	Разработка интерфейса	Программист1 *CD*-матрица М. переговоры	100 1 штука/д	500
2	Разработка модулей обработки данных	Программист1 *CD*-матрица	100 1 штука/д	
3	Разработка структуры базы данных	Программист2 *CD*-матрица М. переговоры	100 1 штука/д	1500

Окончание табл. 3.3

Номер задачи	Название задачи	Ресурсы	Единицы, %	Затраты, руб.
4	Заполнение базы данных	Программист2 *CD*-матрица	100 1 штука/д	–
5	Отладка программного комплекса	Постановщик Программист1 Программист2 *CD*-матрица	100 100 100 2 штука/д	–
6	Тестирование и исправление ошибок	Постановщик Программист1 Программист2 *CD*-матрица	100 100 100 2 штука/д	–
7	Составление программной документации	Постановщик *CD*-матрица М. переговоры	100 10 штук	2000

Назначение профиля загрузки. Выбираем пункт меню *Задача/Использование задачи*. Двойной щелчок мыши по назначению *Постановщик Постановка задачи*. В открывшемся окне выбрать вкладку *Ресурсы* и в поле *Профиль загрузки* установить значение *Загрузка в конце* (рис. 3.19).

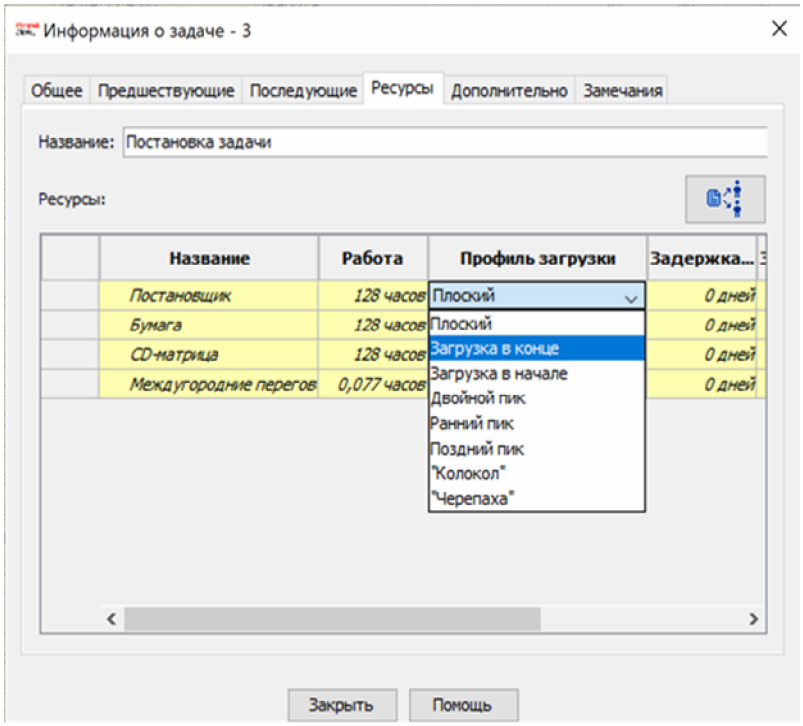


Рис. 3.19. Окно ввода профиля загрузки

Изменение норм затрат в назначении. Выбираем пункт меню *Задача/Использование задачи*. Двойной щелчок мыши по назначению *Постановщик Тестирование и исправление ошибок*. В открывшемся окне выбираем вкладку *Ресурсы* и в поле *Таблица норм затрат* выбираем *Норма В* (рис. 3.20).

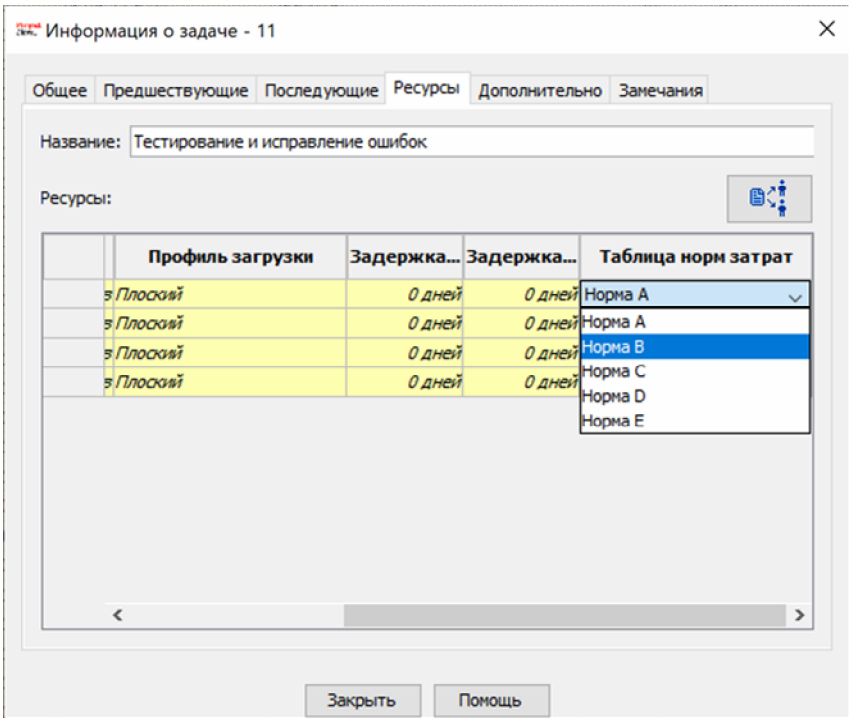


Рис. 3.20. Изменение норм затрат

Аналогично устанавливаем таблицу норм затрат «В» для назначений *Программист1* и *Программист2* этой же задачи. Результат – расчет затрат этих ресурсов для выполнения данной задачи выполняется по таблице норм В.

Глава 4. УПРАВЛЕНИЕ ВЕРСИЯМИ

4.1. Введение в системы контроля версий

Необходимость отслеживания версии документа – задача, встречающаяся постоянно при разработке информационных систем.

Например, текстовые документы Microsoft Office. При сбое компьютера и перезапуске нам будет предложена возможность восстановления версии из последних сохраненных. Еще ситуация – одновременный просмотр и редактирование версии документа. Нам в этом случае будут предложены варианты (версии) частей текста, отредактированные другими пользователями. А если файлов много, то требуется отслеживание их версий, что является обыденной ситуацией при разработке программ. Для этого и необходимы специализированные средства – системы контроля версий.

При отсутствии системы контроля версий приходится проявлять внимательность и изобретательность – самому решать вопросы хранения и восстановления, например, создание файлов и папок с указанием даты. Это помогает для одного пользователя, а если данные файлы (при разработке приложения) правят несколько человек? Здесь без возможного осложнения по различию в содержании файлов не обойтись, во избежание проблем и потребуются данные системы.

Виды систем контроля версий [6]:

– локальные системы контроля версий;

– централизованные системы контроля версий;

– децентрализованные системы контроля версий.

Локальные системы контроля версий (рис. 4.1). Для локальных систем контроля программисты давно разработали системы с простой базой данных, которая хранит записи о всех изменениях в файлах, осуществляя тем самым контроль ревизий.

Недостаток – одному специалисту возможно хранить версии и возвращаться к прежним в случае необходимости. Но если это команда, то простая база, хранящаяся на одном компьютере, не решает эту проблему качественно.

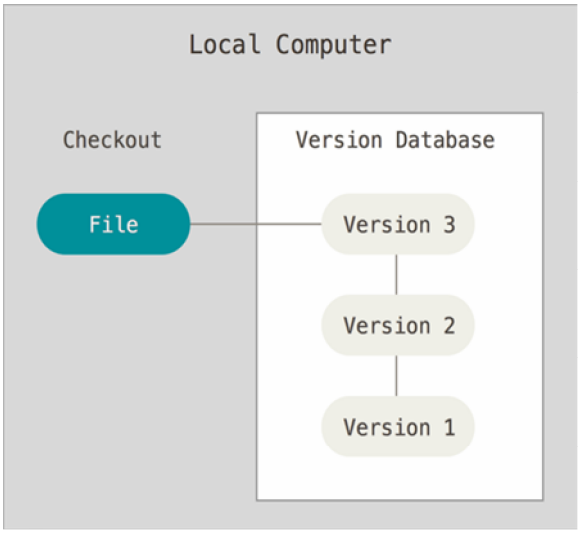


Рис. 4.1. Локальные системы контроля версий

Централизованные системы контроля версий (рис. 4.2). Вынесение базы на общий сервер позволило взаимодействовать команде. Получили распространение системы контроля: CVS, Subversion. Данные системы широко использовались, пока не получила распространение проблема взаимодействия команды, разбросанной по всему миру. Вынесение базы в интернет решает эту проблему. Но возникли конфликтные ситуации, связанные с недостатком хранения версий только в единой базе. Они очевидны: не все имеют постоянный доступ к мировой паутине и там бывают сбои. Как следствие, на локальных машинах возникали собственные версии, конфликтующие с основной.

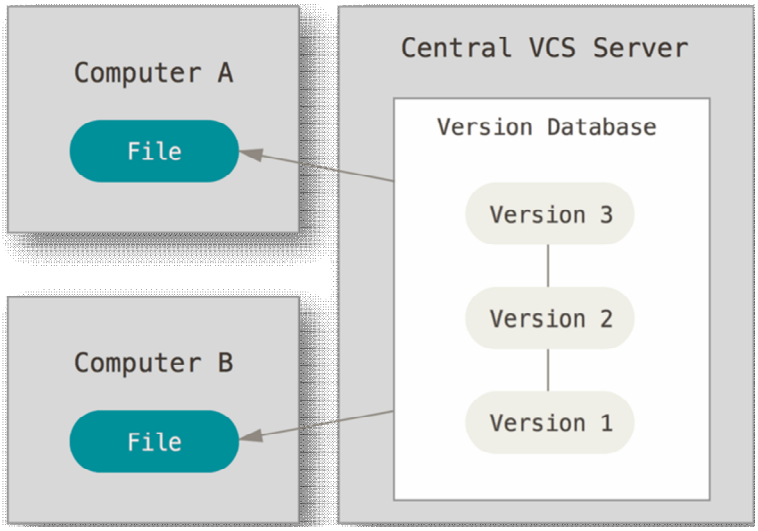


Рис. 4.2. Централизованные системы контроля версий

Децентрализованные системы контроля версий (рис. 4.3). В децентрализованных версиях на каждой локальной машине хранится собственная версия. Но там также хранится информация о полной версии в виде репозитория. В этом случае, даже если централизованный репозиторий откажет, его можно восстановить на основе одной из локальных копий.

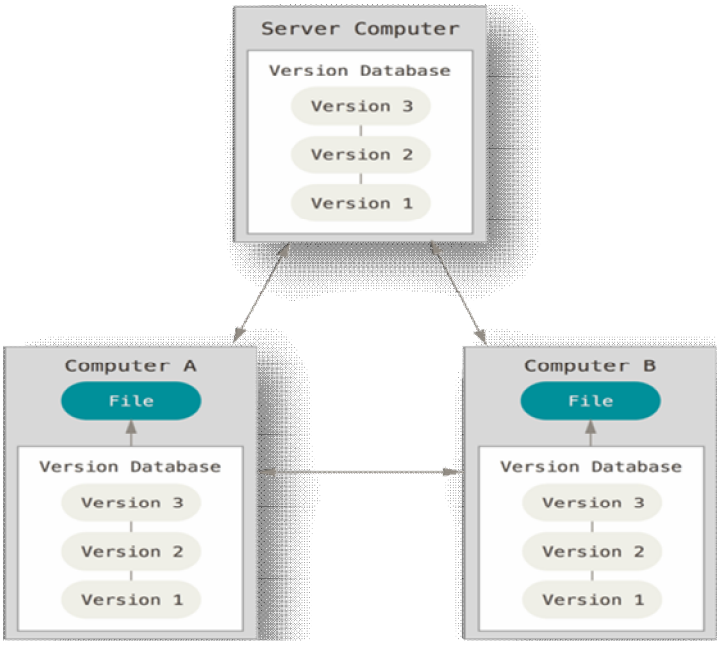


Рис. 4.3. Децентрализованные системы контроля версий

4.2. Схема управления версиями на основе Git и Visual Studio 2019

I. Установка Git и регистрация на GitHub:
– регистрация на Git-хабе;
– создание нового Git-репозитория;
– настройка конфигурации для хранения файлов.
II. Создание решения с проектами:
1) выбор тест-фрейворка;
2) создание решения:
– трехслойного приложения (GUI уровень, уровень бизнес-логики, слой доступа к данным);
– теста приложения;
3) сохранение решения в Git-репозитории.
III. Работа с Git-репозиторием в Visual Studio:

1) сохранение проекта в GIT – смотрите пункт: «Добавление проекта в Git-репозиторий»;

2) клонирование проекта в рамках Visual Studio – смотрите пункт «Добавление существующего проекта из Git-репозитория».

4.3. Установка Git и создание репозитория

Прежде чем использовать Git, вы должны установить его на своем компьютере. Даже если он уже установлен, наверное, это хороший повод, чтобы обновиться до последней версии. Вы можете установить Git из собранного пакета или другого установщика либо скачать исходный код и скомпилировать его самостоятельно.

Установка Git. Для установки Git в Windows имеется несколько способов:

1. Официальная сборка доступна для скачивания на официальном сайте Git. Просто перейдите на страницу *http://git-scm.com/download/win*, и загрузка запустится автоматически. Обратите внимание, что это проект, называемый Git для Windows (другое название msysGit), который отделен от самого Git; для получения дополнительной информации о нем перейдите на *http://msysgit.github.io/*.

2. Другой простой способ установки Git – установить GitHub для Windows. Его установщик включает в себя утилиты командной строки и GUI Git. Он также корректно работает с Powershell, обеспечивает четкое сохранение учетных данных и правильные настройки CRLF. Вы познакомитесь с этими вещами подробнее несколько позже, здесь же отметим, что они будут вам необходимы. Вы можете загрузить GitHub для Windows с сайта: *http://windows.github.com.*

Для создания репозитория на GitHub необходимо:

1) войти в систему, затем в навигационной системе выбрать «+» и «New repository»;

2) далее заполняем поля:

– Repository name;

– Description;

– Репозиторий, находящийся на GitHub, называется главным;

– Базовые операции.

4.4. Создание решения и добавление проектов в Git-репозиторий

Трехслойное решение в рамках Visual Studio создается простым способом:

– создается отдельное решение трехслойного приложения;

– в рамках его выполняем проекты каждого слоя. Все проекты создаются на основе шаблонов Visual Studio;

– создаем тестовый проект. Вид подключаемого пакета тестов – выбор команды.

4.5. Добавление существующего проекта из Git-репозитория

Чтобы добавить проект с Git, необходимо:

1. При открытии Visual Studio выбрать клонирование или извлечение кода (рис. 4.4).

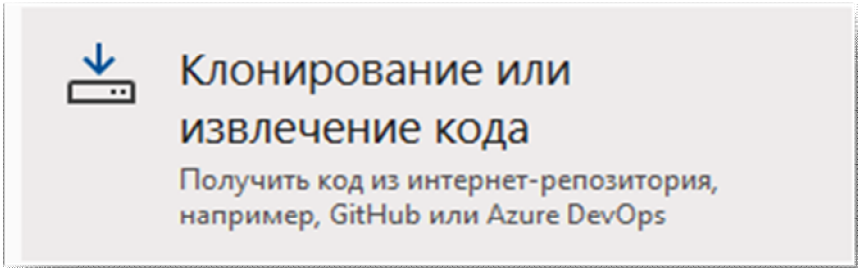


Рис. 4.4. Добавление проекта Git

2. Выбрать папку для размещения проекта (рис. 4.5).

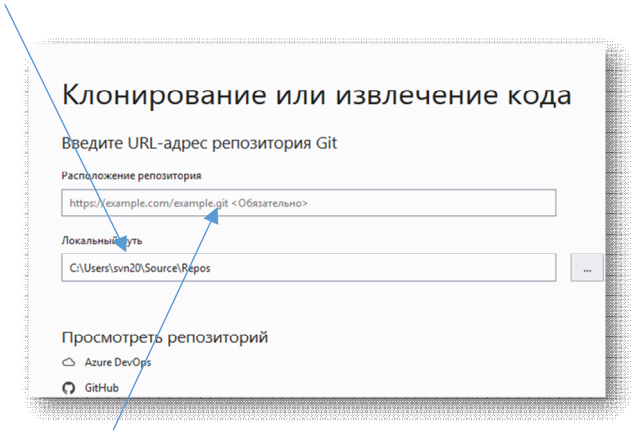


Рис. 4.5. Папка проекта

3. Вставить ссылку на репозиторий Git и нажать кнопку «Клонировать».

4. Получить ссылку на репозиторий можно следующим способом:

– по поиску по GitHub (рис. 4.6);

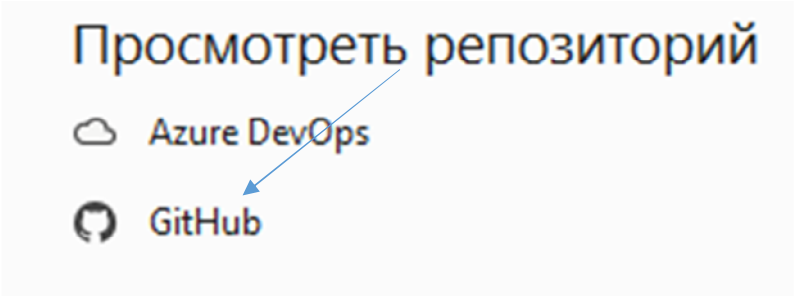


Рис. 4.6. Поиск по Git

– либо копированием ссылки из Git и вставки ее в поле расположения репозитория (рис. 4.7).

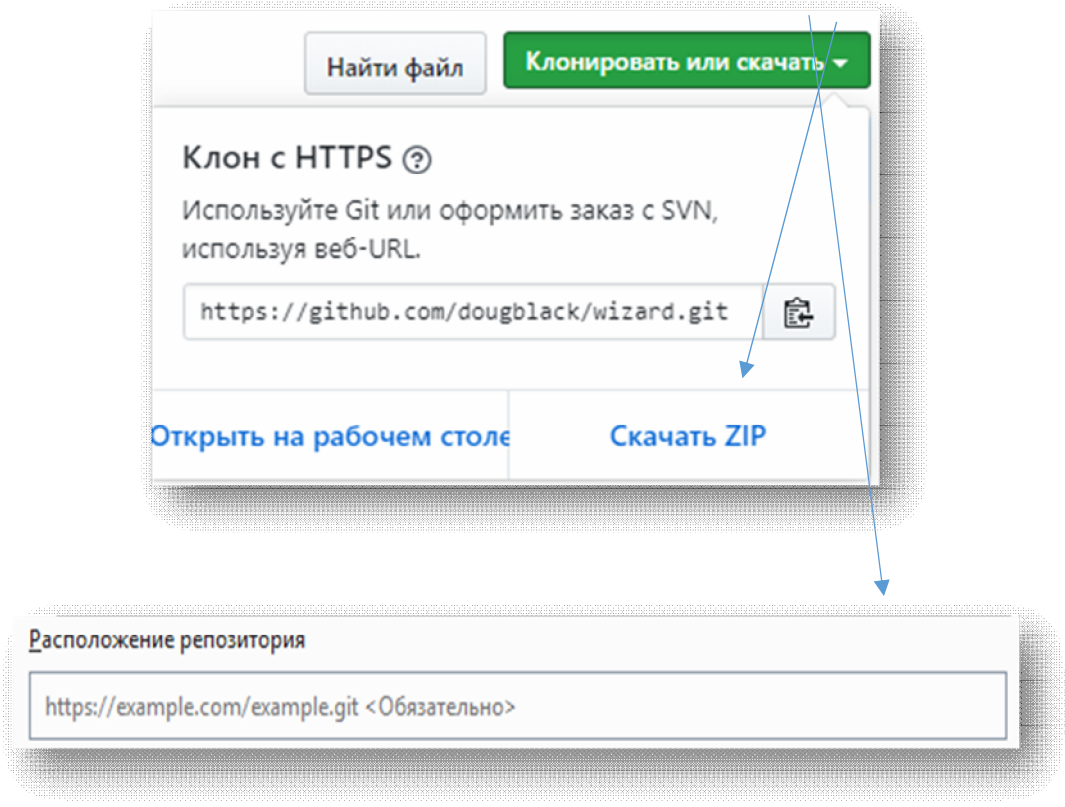


Рис. 4.7. Копирование ссылки

5. Двойной клик по проекту, и получаем окно управления (рис. 4.8):

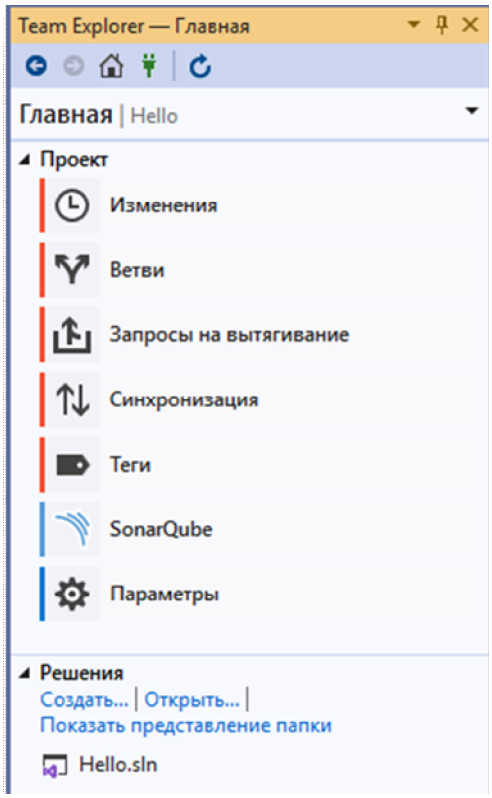


Рис. 4.8. Окно управления

6. Переключаемся на обозреватель решения и работаем с проектом (рис. 4.9).

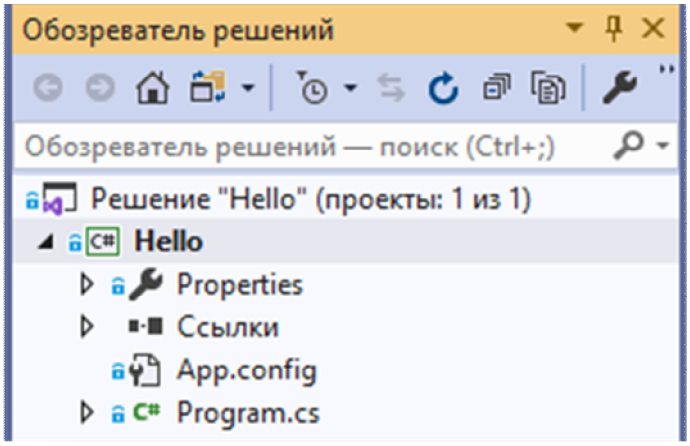


Рис. 4.9. Обозреватель решений

Литература

1. Руководство к своду знаний по управлению проектами (Руководство PMBOK). – 6-е изд. – ANSI/PMI, 2019. – 792 с.

2. Мартин, Р. С. Принципы, паттерны и методики гибкой разработки на языке C# / Р. С. Мартин, М. Мартин. – М. : Символ-Плюс, 2011. – 768 с.

3. Бек, К. Экстремальное программирование: разработка через тестирование / К. Бек. – СПб. : Питер, 2003. – 224 с.

4. Сазерленд, Д. Scrum. Революционный метод управления проектами / Д. Сазерленд. – М. : Манн, Иванов и Фербер, 2016. – 288 с.

5. Куликов, С. С. Тестирование программного обеспечения. Базовый курс / С. С. Куликов. – Минск : Четыре четверти, 2017. – 312 с.

6. Чакон, С. Git для профессионального программиста / С. Чакон, Б. Штрауб. – СПб. : Питер, 2016. – 496 с.

Учебное электронное издание комбинированного распространения

Учебное издание

Прокопенко Дмитрий Викторович
Шибеко Виктор Николаевич

УПРАВЛЕНИЕ РАЗРАБОТКОЙ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Учебно-методическое пособие

Электронный аналог печатного издания

Редактор *Н. В. Гладкова*
Компьютерная верстка *И. П. Минина*

Подписано в печать 09.09.20.
Формат 60x84/16. Бумага офсетная. Гарнитура «Таймс».
Ризография. Усл. печ. л. 4,42. Уч.-изд. л. 4,45.
Изд. № 5.
http://www.gstu.by

Издатель и полиграфическое исполнение
Гомельский государственный
технический университет имени П. О. Сухого.
Свидетельство о гос. регистрации в качестве издателя
печатных изданий за № 1/273 от 04.04.2014 г.
пр. Октября, 48, 246746, г. Гомель