Технологии программирования. Компонентный подход

В. В. Кулямин

Лекция 3. Унифицированный процесс разработки и экстремальное программирование

Аннотация

Рассматриваются в деталях модели разработки ПО, предлагаемые в рамках унифицированного процесса разработки Rational (RUP) и экстремального программирования (XP).

Ключевые слова

«Тяжелые» процессы разработки, «живые» методы разработки, унифицированный процесс Rational, экстремальное программирование, модели ПО.

Текст лекции

В этой лекции мы рассмотрим детально два процесса разработки — *унифицированный процесс Rational (Rational Unified Process, RUP)* и *экстремальное программирование (Extreme Programming, XP)*. Оба они являются примерами итеративных процессов, но построены они на основе различных предположений о природе разработки программного обеспечения, и, соответственно, сами достаточно сильно отличаются.

RUP является примером так называемого *«тяжелого»* процесса, детально описанного и предполагающего поддержку собственно разработки исходного кода ПО большим количеством вспомогательных действий. Примерами таких действий являются разработка планов, технических заданий, многочисленных проектных моделей, проектной документации, и пр. Основная цель такого процесса — отделить успешные практики разработки и сопровождения ПО от конкретных людей, умеющих их применять. Многочисленные вспомогательные действия дают надежду сделать возможным успешное решение задач по конструированию и поддержке сложных систем с помощью имеющихся работников, не обязательно являющихся суперпрофессионалами.

Для достижения этого выполняется иерархическое пошаговое детальное описание предпринимаемых в той или иной ситуации действий, чтобы можно было научить обычного работника действовать аналогичным образом. В ходе проекта создается много промежуточных документов, позволяющих разработчикам последовательно разбивать стоящие перед ними задачи на более простые. Эти же документы служат для проверки правильности решений, принимаемых на каждом шаге, а также отслеживания общего хода работ и уточнения оценок ресурсов, необходимых для получения желаемых результатов.

Экстремальное программирование, наоборот, представляет так называемые *«живые» (agile)* методы разработки, которые делают упор на использовании хороших разработчиков, а не хорошо отлаженных процессов разработки. Живые методы избегают фиксации четких схем действий, чтобы обеспечить б***о***льшую гибкость в каждом конкретном проекте, а также выступают против разработки дополнительных документов, не вносящих непосредственного вклада в получение готовой работающей программы.

Унифицированный процесс Rational

RUP [1,2] является довольно сложной, детально проработанной итеративной моделью жизненного цикла ПО.

Исторически RUP является развитием модели процесса разработки, принятой в компании Ericsson в 70-х-80-х годах XX века. Эта модель была создана Джекобсоном (Ivar Jacobson), впоследствии, в 1987, основавшим собственную компанию Objectory AB именно для развития технологического процесса разработки ПО как отдельного продукта, который можно было бы переносить в другие организации. После вливания Objectory в Rational в 1995 разработки Джекобсона были интегрированы с работами Ройса (Walker Royce, сын автора «классической»

каскадной модели), Крухтена (Philippe Kruchten) и Буча (Grady Booch), а также с развивавшимся параллельно *универсальным языком моделирования (Unified Modeling Language, UML)*.

RUP основан на трех ключевых идеях.

- Весь ход работ направляется итоговыми целями проекта, выраженными в виде ***вариантов использования (use cases)*** — сценариев взаимодействия результирующей программной системы с пользователями или другими системами, при выполнении которых пользователи получают значимые для них результаты и услуги. Разработка начинается с выделения вариантов использования и на каждом шаге контролируется степенью приближения к их реализации.
- Основным решением, принимаемым в ходе проекта, является ***архитектура*** результирующей программной системы. Архитектура устанавливает набор компонентов, из которых будет построено ПО, ответственность каждого из компонентов (т.е. решаемые им подзадачи в рамках общих задач системы), четко определяет интерфейсы, через которые они могут взаимодействовать, а также способы взаимодействия компонентов друг с другом.
  Архитектура является одновременно основой для получения качественного ПО и базой для планирования работ и оценок проекта в терминах времени и ресурсов, необходимых для достижения определенных результатов. Она оформляется в виде набора графических моделей на языке UML.
- Основой процесса разработки являются *планируемые и управляемые итерации*, объем которых (реализуемая в рамках итерации функциональность и набор компонентов) определяется на основе архитектуры.

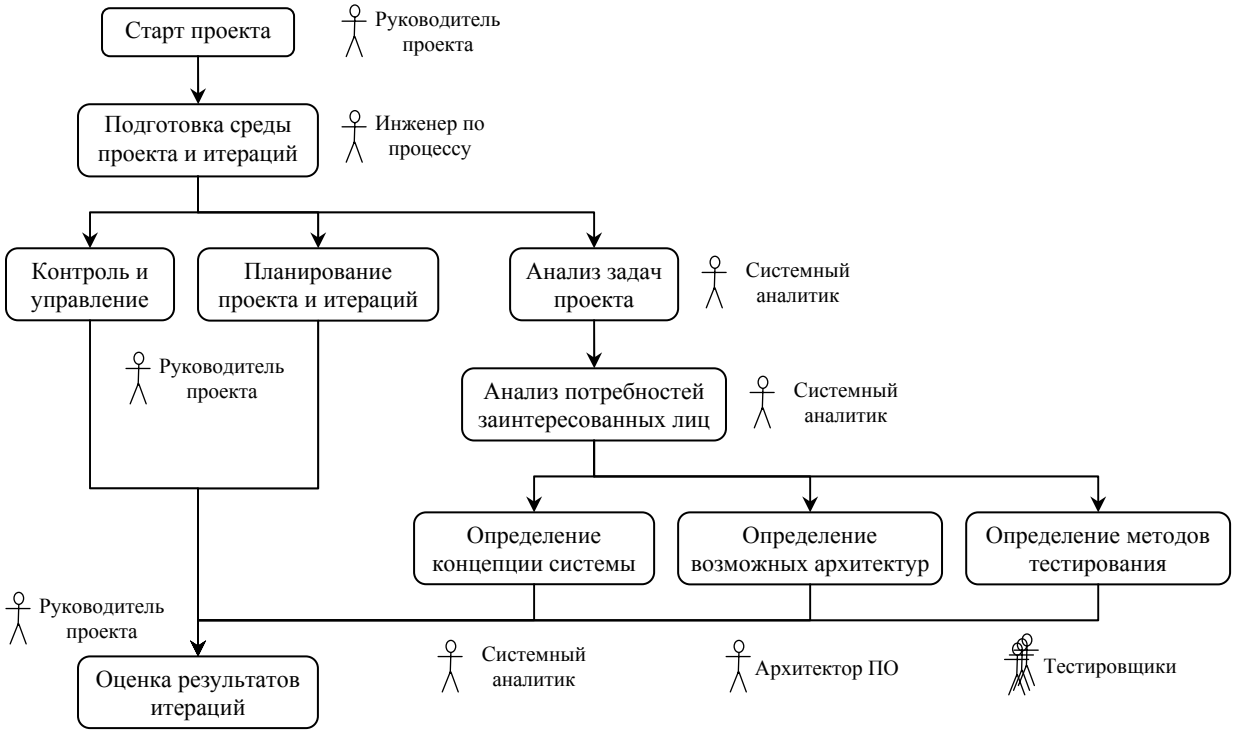

Рисунок 1. Пример хода работ на фазе начала проекта.

RUP выделяет в жизненном цикле на 4 основные фазы, в рамках каждой из которых возможно проведение нескольких итераций. Кроме того, разработка системы может пройти через несколько циклов, включающих все 4 фазы.

1. **Фаза начала проекта (Inception)**
   Основная цель этой фазы — достичь компромисса между всеми заинтересованными лицами относительно задач проекта и выделяемых на него ресурсов.
   На этой фазе определяются основные цели проекта, руководитель проекта и бюджет проекта, основные средства его выполнения — технологии, инструменты, ключевые

исполнители, а также происходит, возможно, апробация выбранных технологий с целью подтверждения возможности достичь целей с их помощью, составляются предварительные планы проекта.
На эту фазу может уходить около 10% времени и 5% трудоемкости одного цикла.
Пример хода работ на этой фазе показан на Рис. 1.

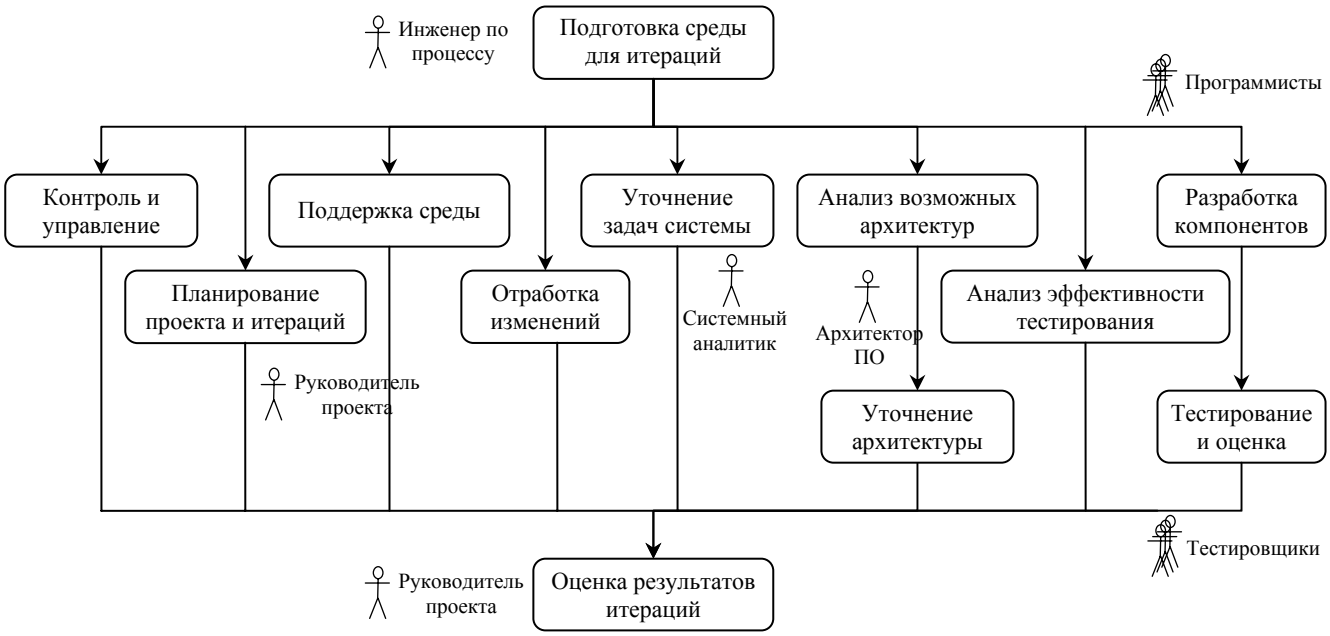

Рисунок 2. Пример хода работ на фазе проектирования.

2. **Фаза проектирования (Elaboration)**
   Основная цель этой фазы — на базе основных, наиболее существенных требований разработать стабильную базовую архитектуру продукта, которая позволяет решать поставленные перед системой задачи и в дальнейшем используется как основа разработки системы.
   На эту фазу может уходить около 30% времени и 20% трудоемкости одного цикла.
   Пример хода работ на этой фазе представлен на Рис. 2.

Рисунок 3. Пример хода работ на фазе построения.

3. **Фаза построения (Construction)**
   Основная цель этой фазы — детальное прояснение требований и разработка системы,

удовлетворяющей им, на основе спроектированной ранее архитектуры. В результате должна получиться система, реализующая все выделенные варианты использования. На эту фазу уходит около 50% времени и 65% трудоемкости одного цикла.
Пример хода работ на этой фазе представлен на Рис. 3.

4. **Фаза внедрения (Transition)**
   Цель этой фазы — сделать систему полностью доступной конечным пользователям. На этой стадии происходит развертывание системы в ее рабочей среде, бета-тестирование, подгонка мелких деталей под нужды пользователей.
   На эту фазу может уходить около 10% времени и 10% трудоемкости одного цикла.
   Пример хода работ на этой фазе представлен на Рис. 4.

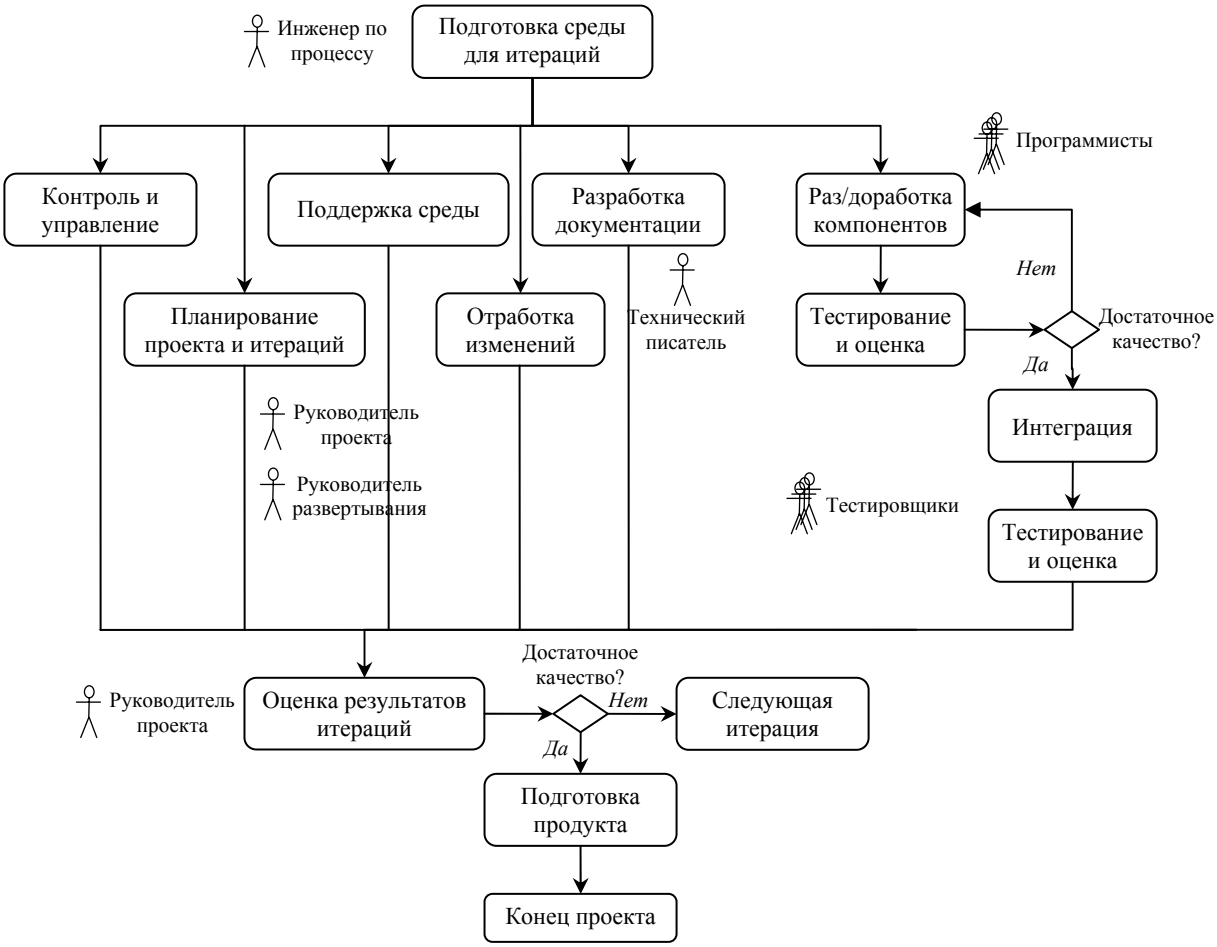

Рисунок 4. Пример хода работ на фазе внедрения.

Артефакты, вырабатываемые в ходе проекта, могут быть представлены в виде баз данных и таблиц с информацией различного типа, разных видов документов, исходного кода и объектных модулей, а также моделей, состоящих из отдельных элементов. Основные артефакты и потоки данных между ними согласно RUP изображены на Рис. 5.

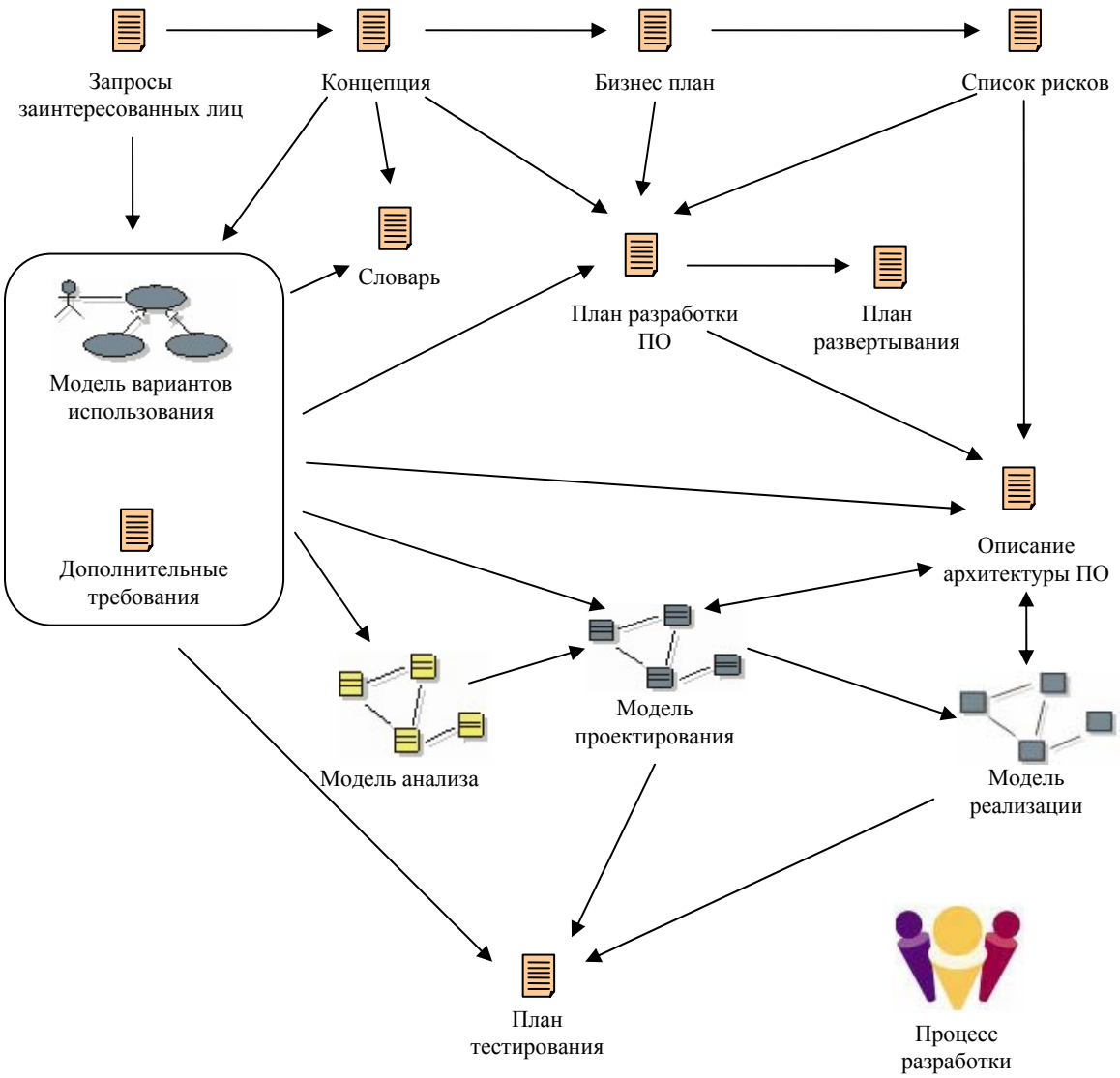

Рисунок 5. Основные артефакты проекта по RUP и потоки данных между ними.

Наиболее важные с точки зрения RUP артефакты проекта — это модели, описывающие различные аспекты будущей системы. Большинство моделей представляют собой наборы диаграмм UML. Основные используемые виды моделей следующие.

- **Модель вариантов использования (Use-Case Model).**
  Эта модель определяет *требования к ПО* — то, что система должна делать — в виде набора вариантов использования. Каждый вариант использования определяет сценарий взаимодействия системы с ***действующими лицами (actors)*** или ролями, дающий в итоге значимый для них результат. Действующими лицами могут быть не только люди, а и другие системы, взаимодействующие с рассматриваемой. Вариант использования определяет основной ход событий, развивающийся в нормальной ситуации, а также может включать несколько альтернативных сценариев, которые начинают работать только при специфических условиях.
  Модель вариантов использования служит основой для проектирования и оценки готовности системы к внедрению.
  Примером варианта использования может служить сценарий действий клиента Интернет-магазина по отношению к сайту этого магазина, в результате которых клиент заказывает товар, например, книги. Такой вариант использования можно назвать «Заказ товара». Если нас интересует сайт магазина только как программная система, результатом можно считать то, что запись о сделанном заказе занесена в базу данных, а оператору заказов отправлено электронное письмо, содержащее всю информацию, необходимую для того, чтобы можно было сформировать заказ. В нее входит

контактная информация покупателя, идентификатор записи о заказе и, например, список заказанных книг с их ISBN, их количество для каждого наименования и номера партий для удобства их поиска на складе. При этом выполнение остальной части варианта использования — это дело других составляющих системы под названием «Интернет-магазин». Эта работа может включать звонок или письмо клиенту и подтверждение, что именно он сделал заказ, вопрос об удобных для него форме, времени и адресе доставки и форме оплаты, формирование заказа, передача его для доставки курьеру, доставка и подтверждение получения заказа и оплаты.
В нашем примере действующими лицами являются клиент, делающий заказ, и оператор заказов.
Альтернативные сценарии в рамках данного варианта могут включиться, если, например, заказанного пользователем товара нет на складе, или сам пользователь находится на плохом счету в магазине, не оплатив какие-то прежние заказы, или наоборот, он является привилегированным клиентом или представителем крупной организации.

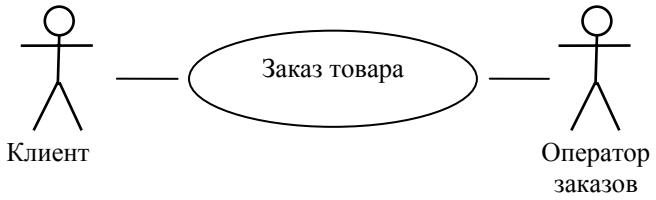

Рисунок 6. Пример варианта использования и действующих лиц.

- **Модель анализа (Analysis Model)**.
  Это модель включает основные классы, необходимые для реализации выделенных вариантов использования, а также возможные связи между классами. Выделяемые классы разбиваются на три разновидности — *интерфейсные*, *управляющие* и *классы данных*. Эти классы представляют собой набор сущностей, в терминах которых работа системы должна представляться пользователям. Они являются понятиями, с помощью которых достаточно удобно объяснять себе и другим происходящее внутри системы, не слишком вдаваясь в детали.
  Интерфейсные классы (boundary classes) соответствуют устройствам или способам обмена данными между системой и ее окружением, в том числе пользователями.
  Классы данных (entity classes) соответствуют наборам данных, описывающих некоторые однотипные сущности внутри системы. Эти сущности являются абстракциями представлений пользователей о данных, с которыми работает система.
  Управляющие классы (control classes) соответствуют алгоритмам, реализующим какие-то значимые преобразования данных в системе и управляющим обменом данными с ее окружением в рамках вариантов использования.
  В нашем примере с Интернет-магазином можно было бы выделить следующие классы в модели анализа: интерфейсный класс, предоставляющий информацию о товаре и возможность сделать заказ, интерфейсный класс, представляющий сообщение оператору, управляющий класс, обрабатывающий введенную пользователем информацию и преобразующий ее в данные о заказе и сообщение оператору, класс данных о заказе. Соответствующая модель приведена на Рис. 7.
  Каждый класс может играть несколько ролей в реализации одного или нескольких вариантов использования. Каждая роль определяет его обязанности и свойства, тоже являющиеся частью модели анализа.
  В рамках других подходов модель анализа часто называется ***концептуальной моделью*** системы. Она состоит из набора классов, совместно реализующих все варианты использования и служащих основой для понимания поведения системы и объяснения его всем заинтересованным лицам.

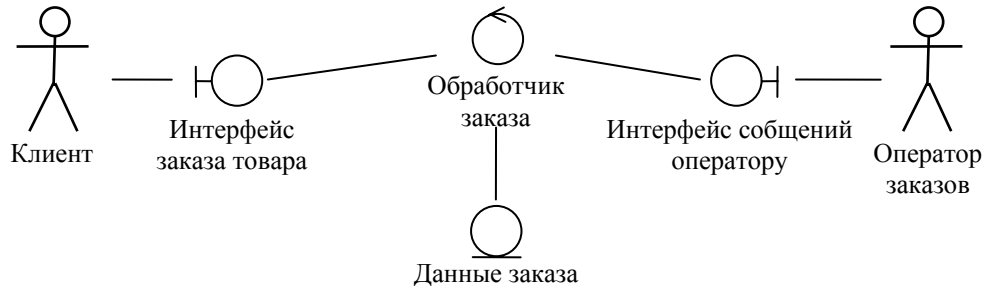

Рисунок 7. Пример модели анализа для одного варианта использования.

- **Модель проектирования (Design Model)**.
  Модель проектирования является детализацией и специализацией модели анализа. Она также состоит из классов, но более четко определенных, с более точным и детальным распределением обязанностей, чем классы модели анализа. Классы модели проектирования должны быть специализированы для конкретной использованной платформы (включающей операционные системы всех вовлеченных машин, используемые языки программирования, интерфейсы и классы конкретных компонентных сред, таких как J2EE, .NET, COM или CORBA, интерфейсы выбранных для использования систем управления базами данных, СУБД, например, Oracle или MS SQL Server, используемые библиотеки разработки пользовательского интерфейса, такие как swing или swt в Java, MFC или gtk, интерфейсы взаимодействующих систем и пр.).
  В нашем примере, прежде всего, необходимо детализировать классы, уточнить их функциональность. Например, для того, чтобы клиенту был удобнее заказывать товар, нужно предоставить ему список имеющихся товаров, какие-то способы навигации и поиска в этом списке, а также детальную информацию о товаре. Это значит, что интерфейс заказа товара реализуется в виде набора классов, представляющих, например, различные страницы сайта магазина. Точно так же данные заказа должны быть детализированы в виде нескольких таблиц в СУБД, включающих, обычно, данные самого заказа (дату, ссылку на данные клиента, строки с количеством отдельных товаров и ссылками на товары), данные товаров, клиента и пр. Кроме того, для реляционной СУБД понадобятся классы-посредники между ее таблицами и объектной структурой остальной программы. Обработчик заказа может быть реализован в виде набора объектов нескольких классов, например, с выделенным отдельно набором часто изменяемых политик (скидки на определенные категории товаров и определенным категориям клиентов, сезонные скидки, рекламные комплекты и пр.) и более постоянным общим алгоритм обработки.
  Далее, приняв, например, решение реализовывать систему с помощью технологий J2EE или .NET, мы тем самым определяем дополнительные ограничения на структуру классов, да и на само их количество. О правилах построения ПО на основе этих технологий рассказывается в следующих лекциях.
- **Модель реализации (Implementation Model)**.
  Под *моделью реализации* в рамках RUP и UML имеют в виду набор *компонентов* результирующей системы и связей между ними. Под компонентом здесь имеется в виду ***компонент сборки*** — минимальный по размерам кусок кода системы, который может участвовать или не участвовать в данной конфигурации, единица конфигурационного управления. Связи между компонентами развертывания представляют собой зависимости между ними. Если компонент зависит от другого компонента, он не может быть поставлен отдельно от него.
  Часто компоненты представляют собой отдельные файлы с исходным кодом. Далее мы познакомимся с компонентами J2EE, часто состоящими из нескольких файлов.
- **Модель развертывания (Deployment Model)**.
  Модель развертывания представляет собой набор ***узлов*** системы, являющихся физически отдельными устройствами, способными обрабатывать информацию —

серверами, рабочими станциями, принтерами, контроллерами датчиков и пр., со *связями* между ними, образованными различного рода сетевыми соединениями. Каждый узел может быть нагружен некоторым множеством компонентов, определенных в модели реализации.
Цель построения модели развертывания — определить физическое положение компонентов распределенной системы, обеспечивающее выполнение ею нужных функций в тех местах, где эти функции будут доступны и удобны для пользователей.
В нашем примере Web-сайта магазина узлами системы являются один или несколько компьютеров, на которых развернуты Web-сервер, пересылающий по запросу пользователя текст нужной странички, набор программных компонентов, отвечающих за генерацию страничек, обработку действий пользователя и взаимодействие с базой данных, и СУБД, в рамках которой работает база данных системы. Кроме того, строго говоря, в систему входят все компьютеры клиентов, на которых работает Web-браузер, делающий возможным просмотр страничек сайта и пересылку кодированных действий пользователя для их обработки.

- **Модель тестирования (Test Model** или **Test Suite)**.
  В рамках этой модели определяются ***тестовые варианты*** или ***тестовые примеры (test cases)*** и *тестовые процедуры (test scripts)*. Первые являются определенными сценариями работы одного или нескольких действующих лиц с системой, разворачивающимися в рамках одного из вариантов использования. Тестовый вариант включает, помимо входных данных на каждом шаге, где они могут быть введены, условия выполнения отдельных шагов и корректные ответы системы для всякого шага, на котором ответ системы можно наблюдать. В отличие от вариантов использования, в тестовых вариантах четко определены входные данные, и, соответственно, тестовый вариант либо вообще не имеет альтернативных сценариев, либо может предусматривать альтернативный порядок действий, если система может вести себя недетерминировано и предоставлять разные результаты в ответ на одни и те же действия. Все другие альтернативы обычно заканчиваются вынесением вердикта о некорректной работе системы.
  Тестовая процедура представляет собой способ выполнения одного или нескольких тестовых вариантов и их составных элементов (отдельных шагов и проверок). Это может быть инструкция по ручному выполнению входящих в тестовый вариант действий или программный компонент, автоматизирующий запуск тестов.
  Для выделенного варианта использования «Заказ товара» можно определить следующие тестовые варианты:
    - заказать один из имеющихся на складе товаров и проверить, что сообщение об этом заказе поступило оператору;
    - заказать большое количество товаров и проверить, что все работает так же;
    - заказать отсутствующий на складе товар и проверить, что в ответ приходит сообщение об его отсутствии;
    - сделать заказ от имени пользователя, помещенного в «черный список», и проверить, что в ответ приходит сообщение о неоплаченных прежних заказах.

RUP также определяет *дисциплины,* включающие различные наборы деятельностей, которые в разных комбинациях и с разной интенсивностью, выполняются на разных фазах. В документации по процессу каждая дисциплина сопровождается довольно большой диаграммой, поясняющей действия, которые нужно выполнить в ходе работ в рамках данной дисциплины, артефакты, с которыми надо иметь дело, и роли вовлеченных в эти действия лиц.

- **Моделирование предметной области (бизнес-моделирование, Business Modeling)**
  Задачи этой деятельности — понять предметную область или бизнес-контекст, в которых должна будет работать система и убедиться, что все заинтересованные лица понимают его одинаково, осознать имеющиеся проблемы, оценить их возможные решения и их последствия для бизнеса организации, в которой будет работать система.
  В результате моделирования предметной области должна появиться ее модель в виде

набора диаграмм классов (объектов предметной области) и деятельностей (представляющих бизнес-операции и бизнес-процессы). Эта модель служит основой модели анализа.

- **Определение требований (Requirements)**
  Задачи — понять, что должна делать система, и убедиться во взаимопонимании по этому поводу между заинтересованными лицами, определить границы системы и основу для планирования проекта и оценок затрат ресурсов в нем.
  Требования принято фиксировать в виде модели вариантов использования.
- **Анализ и проектирование (Analysis and Design)**
  Задачи — выработать архитектуру системы на основе требований, убедиться, что данная архитектура может быть основой работающей системы в контексте ее будущего использования.
  В результате проектирования должна появиться модель проектирования, включающая диаграммы классов системы, диаграммы ее компонентов, диаграммы взаимодействий между объектами в ходе реализации вариантов использования, диаграммы состояний для отдельных объектов, и диаграммы развертывания.
- **Реализация (Implementation)**
  Задачи — определить структуру исходного кода системы, разработать код ее компонентов и протестировать их, интегрировать систему в работающее целое.
- **Тестирование (Test)**
  Задачи — найти и описать дефекты системы (проявления недостатков ее качества), оценить ее качество в целом, оценить выполнены или нет гипотезы, лежащих в основе проектирования, оценить степень соответствия системы требованиям.
- **Развертывание (Deployment)**
  Задачи — установить систему в ее рабочем окружении и оценить ее работоспособность на том месте, где она должна будет работать.
- **Управление конфигурациями и изменениями (Configuration and Change Management)**
  Задачи — определение элементов, подлежащих хранению в репозитории проекта и правил построения из них согласованных конфигураций, поддержание целостности текущего состояния системы, проверка согласованности вносимых изменений.
- **Управление проектом (Project Management)**
  Задачи — планирование, управление персоналом, обеспечение взаимодействия на благо проекта между всеми заинтересованными лицами, управление рисками, отслеживание текущего состояния проекта.
- **Управление средой проекта (Environment)**
  Задачи — подстройка процесса под конкретный проект, выбор и замена технологий и инструментов, используемых в проекте.

Первые пять дисциплин считаются рабочими, остальные — поддерживающими. Распределение объемов работ по дисциплинам в ходе проекта выглядит согласно руководству по RUP примерно так, как показано на Рис. 8.

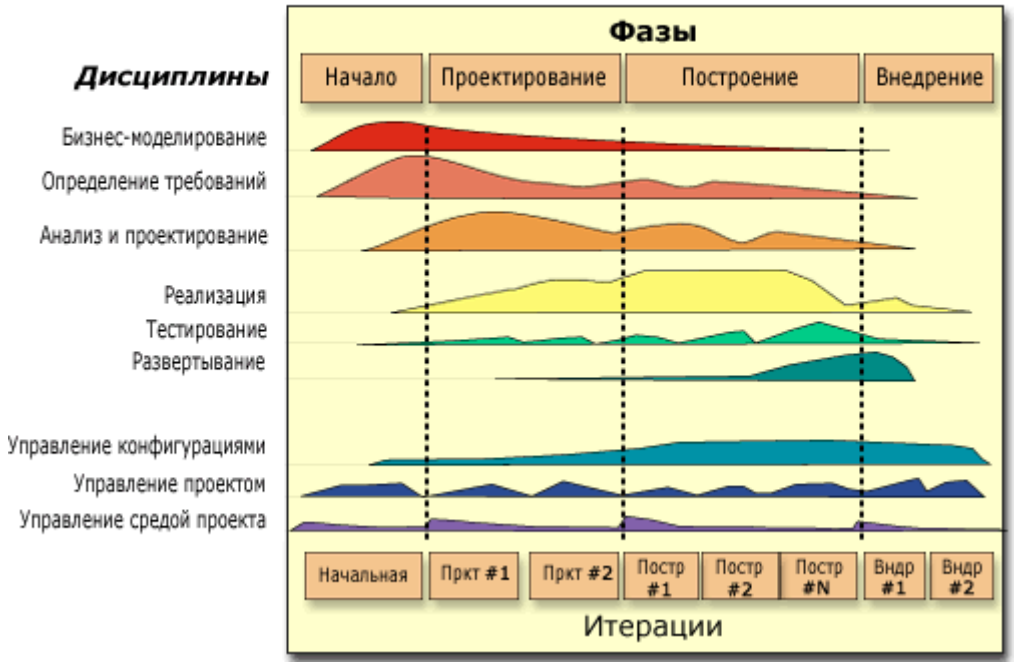

Рисунок 8. Распределение работ между различными дисциплинами в проекте по RUP.

Напоследок перечислим техники, используемые в RUP согласно [3].

- Выработка концепции проекта (project vision) в его начале для четкой постановки задач
- Управление по плану
- Снижение рисков и отслеживание их последствий, как можно более раннее начало работ по преодолению рисков
- Тщательное экономическое обоснование всех действий — делается только то, что нужно заказчику
- Как можно более раннее формирование базовой архитектуры
- Использование компонентной архитектуры
- Прототипирование, инкрементная разработка и тестирование
- Регулярные оценки текущего состояния
- Управление изменениями, постоянная отработка изменений извне проекта
- Нацеленность на создание продукта, работоспособного в реальном окружении
- Нацеленность на качество
- Адаптация процесса под нужды проекта

Экстремальное программирование

Экстремальное программирование (Extreme Programming, XP) [4] возникло как эволюционный метод разработки ПО «снизу-вверх». Этот подход является примером так называемого метода *«живой» разработки (Agile Development Method)*. В группу «живых» методов входят, помимо экстремального программирования, методы SCRUM, DSDM (Dynamic Systems Development Method, метод разработки динамичных систем), Feature-Driven Development (разработка, управляемая характеристиками результата) и др.

Основные принципы «живой» разработки ПО зафиксированы в манифесте «живой» разработки [5], появившемся в 2000 году.

- Люди, участвующие в проекте, и их общение более важны, чем процессы и инструменты
- Работающая программа более важна, чем исчерпывающая документация

- Сотрудничество с заказчиком более важно, чем обсуждение деталей контракта
- Отработка изменений более важна, чем следование планам

«Живые» методы появились как протест против чрезмерной бюрократизации разработки ПО, обилия побочных, не являющихся необходимыми для получения конечного результата документов, которые приходится оформлять при проведении проекта в соответствии с большинством «тяжелых» процессов, дополнительной работы по поддержке фиксированного процесса организации, как это требуется в рамках, например, CMM. Большая часть таких работ и документов не имеет прямого отношения к разработке ПО и обеспечению его качества, а предназначена для соблюдения формальных пунктов контрактов на разработку, получения и подтверждения сертификатов на соответствие различным стандартам.

«Живые» методы позволяют большую часть усилий разработчиков сосредоточить собственно на задачах разработки и удовлетворения реальных потребностей пользователей. Отсутствие кипы документов и необходимости поддерживать их в связном состоянии позволяет более быстро и качественно реагировать на изменения в требованиях и в окружении, в котором придется работать будущей программе.

Тем не менее, XP имеет свою схему процесса разработки (хотя широко используемое понимание «процесса разработки» как достаточно жесткой схемы действий противоречит идее «живости» разработки), приведенную на Рис. 9.

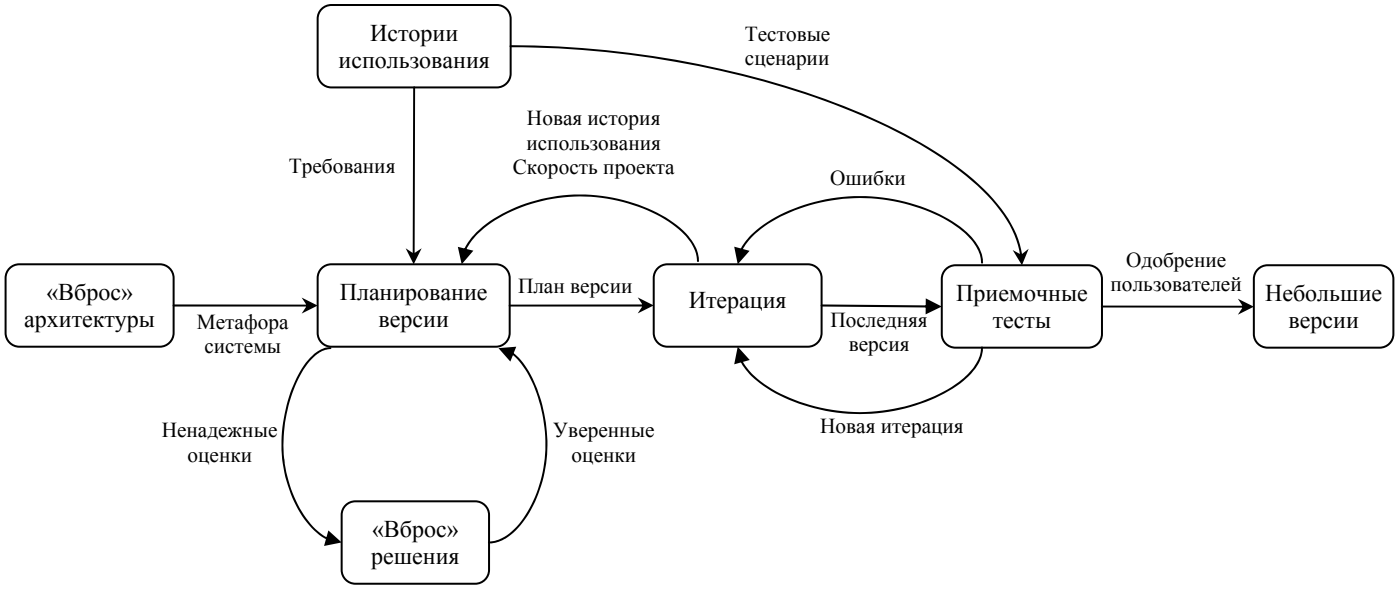

Рисунок 9. Схема потока работ в XP.

По утверждению авторов XP, эта методика представляет собой не столько следование каким-то общим схемам действий, сколько применение комбинации следующих техник. При этом каждая техника важна, и без ее использования разработка считается идущей не по XP, согласно утверждению Кента Бека, Kent Beck [4], одного из авторов этого подхода, наряду с Уордом Каннингемом, Ward Cunningham, и Роном Джефрисом, Ron Jeffries.

- **Живое планирование (planning game)**
  Его задача как можно быстрее определить объем работ, который нужно сделать до следующей версии ПО. Решение принимается, в первую очередь, на основе приоритетов заказчика (т.е. его потребностей, того, что нужно ему от системы для более успешного ведения своего бизнеса) и, во вторую, на основе технических оценок (т.е. оценок трудоемкости разработки, совместимости с остальными элементами системы и пр.). Планы изменяются, как только они начинают расходиться с действительностью или пожеланиями заказчика.
- **Частая смена версий (small releases)**
  Самая первая работающая версия должна появиться как можно быстрее, и тут же должна начать использоваться. Следующие версии подготавливаются через достаточно короткие

промежутки времени (от нескольких часов при небольших изменениях и небольшой программе, до месяца-двух при серьезной переработке крупной системы).

- **Метафора (metaphor) системы**
  Метафора в достаточно простом и понятном команде виде должна описывать основной механизм работы системы. Это понятие напоминает архитектуру, но должно гораздо проще, всего в виде одной-двух фраз описывать основную суть принятых технических решений.
- **Простые проектные решения (simple design)**
  В каждый момент времени система должна быть сконструирована так просто, насколько это возможно. Не надо добавлять функции заранее — только после явной просьбы об этом. Вся лишняя сложность удаляется, как только обнаруживается.
- **Разработка на основе тестирования (test-driven development)**
  Разработчики сначала пишут тесты, потом пытаются реализовать свои модули так, чтобы тесты срабатывали. Заказчики заранее пишут тесты, демонстрирующие основные возможности системы, чтобы можно было увидеть, что система действительно заработала.
- **Постоянная переработка (refactoring)**
  Программисты постоянно перерабатывают систему для устранения излишней сложности, увеличения понятности кода, повышения его гибкости, но без изменений в его поведении, что проверяется прогоном после каждой переделки тестов. При этом предпочтение отдается более элегантным и гибким решениям, по сравнению с просто дающими нужный результат. Неудачно переработанные компоненты должны выявляться при выполнении тестов и откатываться к последнему целостному состоянию (вместе с зависимыми от них компонентами).
- **Программирование парами (pair programming)**
  Кодирование выполняется двумя программистами на одном компьютере. Объединение в пары произвольно и меняется от задачи к задаче. Тот, в чьих руках клавиатура, пытается наилучшим способом решить текущую задачу. Второй программист анализирует работу первого и дает советы, обдумывает последствия тех или иных решений, новые тесты, менее прямые, но более гибкие решения.
- **Коллективное владение кодом (collective ownership)**
  В любой момент любой член команды может изменить любую часть кода. Никто не должен выделять свою собственную область ответственности, вся команда в целом отвечает за весь код.
- **Постоянная интеграция (continuous integration)**
  Система собирается и проходит интеграционное тестирование как можно чаще, по несколько раз в день, каждый раз, когда пара программистов оканчивает реализацию очередной функции.
- **40-часовая рабочая неделя**
  Сверхурочная работа рассматривается как признак больших проблем в проекте. Не допускается сверхурочная работа 2 недели подряд — это истощает программистов и делает их работу значительно менее продуктивной.
- **Включение заказчика в команду (on-site customer)**
  В составе команды разработчиков постоянно находится представитель заказчика, который доступен в течение всего рабочего дня и способен отвечать на все вопросы о системе. Его обязанностью являются достаточно оперативные ответы на вопросы любого типа, касающиеся функций системы, ее интерфейса, требуемой производительности, правильной работы системы в сложных ситуациях, необходимости поддерживать связь с другими приложениями и пр.
- **Использование кода как средства коммуникации**
  Код рассматривается как важнейшее средство общения внутри команды. Ясность кода — один из основных приоритетов. Следование стандартам кодирования, обеспечивающим

такую ясность, обязательно. Такие стандарты, помимо ясности кода, должны обеспечивать минимальность выражений (запрет на дублирование кода и информации) и должны быть приняты всеми членами команды.

- **Открытое рабочее пространство (open workspace)**
  Команда размещается в одном, достаточно просторном помещении, для упрощения коммуникации и возможности проведения коллективных обсуждений при планировании и принятии важных технических решений.
- **Изменение правил по необходимости (just rules)**
  Каждый член команды должен принять перечисленные правила, но при возникновении необходимости команда может поменять их ,если все ее члены пришли к согласию по поводу этого изменения.

Как видно из применяемых техник, XP рассчитано на использование в рамках небольших команд (не более 10 программистов), что подчеркивается и авторами этой методики, больший размер команды разрушает необходимую для успеха простоту коммуникации и делает невозможным применение многих перечисленных приемов.

Достоинствами XP, если его удается применить, является большая гибкость, возможность быстро и аккуратно вносить изменения в ПО в ответ на изменения требований и отдельные пожелания заказчиков, высокое качество получающегося в результате кода и отсутствие необходимости убеждать заказчиков в том, что результат соответствует их ожиданиям.

Недостатками этого подхода являются невыполнимость в таком стиле достаточно больших и сложных проектов, невозможность планировать сроки и трудоемкость проекта на достаточно долгую перспективу и четко предсказать результаты длительного проекта в терминах соотношения качества результата и затрат времени и ресурсов. Также можно отметить неприспособленность XP для тех случаев, в которых возможные решения не находятся сразу на основе ранее полученного опыта, а требуют проведения предварительных исследований

XP как совокупность описанных техник впервые было использовано в ходе работы на проектом C3 (Chrysler Comprehensive Compensation System, разработка системы учета выплат работникам компании Daimler Chrysler). Из 20-ти участников этого проекта 5 (в том числе упомянутые выше 3 основных автора XP) опубликовали еще во время самого проекта и в дальнейшем 3 книги и огромное количество статей, посвященных XP. Этот проект неоднократно упоминается в различных источниках как пример использования этой методики [6,7,8]. Приведенные ниже данные об этом проекте собраны на основе упомянутых статей [9], за вычетом не подтверждающихся сведений, и иллюстрируют проблемы некоторых техник XP при их применении в достаточно сложных проектах.

Проект стартовал в январе 1995 года, с марта 1996 года, после включения в него Кента Бека, он проходил с использованием XP. К этому времени он уже вышел за рамки бюджета и планов поэтапной реализации функций. Команда разработчиков была сокращена и в течении примерно полугода после этого проект развивался довольно успешно. В августе 1998 года появился прототип, который мог обслуживать около 10000 служащих. Первоначально предполагалось, что проект завершится в середине 1999 года, и результирующее ПО будет использоваться для управления выплатами 87000 служащим компании. Он был прерван в феврале 2000 года после 4-х лет работы по XP в связи с полным несоблюдением временных рамок и бюджета. Разработанное ПО ни разу не использовалось для работы с данными о более чем 10000 служащих, хотя было показано, что оно справится с данными 30000 работников компании. Человек, игравший роль включенного в команду заказчика в проекте, уволился через несколько месяцев такой работы, не выдержав нагрузки, и так и не получил адекватной замены до конца проекта.

Литература к Лекции 3

[1] У. Ройс. Управление проектами по созданию программного обеспечения. М.: Лори, 2002.
[2] А. Якобсон, Г. Буч, Дж. Рамбо. Унифицированный процесс разработки программного обеспечения. СПб.: Питер, 2002.
[3] Kroll, The Spirit of the RUP. *www-106.ibm.com/developerworks/rational/library/content/RationalEdge/dec01/ TheSpiritoftheRUPDec01.pdf*

[4] К. Бек. Экстремальное программирование. СПб.: Питер, 2002.
[5] http://www.agilemanifesto.org/
[6] K. Beck, et. al. Chrysler goes to “Extremes”. Distributed Computing, 10/1998.
[7] A. Cockburn. Selecting a Project’s Methodology. IEEE Software, 04/2000.
[8] L. Williams, R. R. Kessler, W. Cunningham, R. Jeffries. Strengthening the Case for Pair Programming. IEEE Software 4/2000.
[9] G. Keefer. Extreme Programming Considered Harmful for Reliable Software Development. AVOCA Technical Report, 2002.
Доступен как http://www.avoca-vsm.com/Dateien-Download/ExtremeProgramming.pdf.

Задания к Лекции 3