МИНИСТЕРСТВО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ИРКУТСКИЙ ГОСУДАРСТВЕННЫЙ АГРАРНЫЙ УНИВЕРСИТЕТ им. А.А. ЕЖЕВСКОГО

Кафедра «Информатика и математическое моделирование»

Учебно-методические указания
для самостоятельной работы
по курсу
«Современные технологии разработки программного обеспечения»
для студентов направления:
09.04.03 Прикладная информатика

Молодежный, 2020

Печатается по решению методической комиссии института экономики, управления и прикладной информатики Иркутского государственного аграрного университета им. А.А. Ежевского.

Протокол №3 от 26 ноября 2020 г.

Рецензенты: к.т.н., доцент, директор института экономики, управления и прикладной информатики Федурина Н.И.; доцент кафедры информатики и математического моделирования Белякова А.Ю.

Бендик Н.В. Учебно-методические указания для самостоятельной работы по курсу «Современные технологии разработки программного обеспечения» для студентов направления подготовки 09.04.03 «Прикладная информатика», [Текст] / Н.В. Бендик – Иркутск: Изд-во Иркутского ГАУ, 2020. – 39 с.

Данные методические указания разработано для поддержки компьютерных лабораторных занятий и самостоятельной работы по курсу «Современные технологии разработки программного обеспечения» для студентов и магистрантов, обучающихся по направлениям «Бизнес-информатика», «Прикладная информатика».

История развития технологий разработки ПО

1. Технологии

2. Этапы развития

3. Методы проектирования

4. Этапы и элементы процесса разработки

5. Инструментарий технологии программирования

Существует множество различных процессов для создания ПО. Тем не менее, технологий, рассматривающих полный жизненный цикл

проекта разработки ПО, сочетающих в себе научный подход, серьезную базу исследований и имеющих историю реального использования и адаптации, относительно немного.

За несколько десятилетий эволюции аппаратное обеспечение значительно усовершенствовалось. Вычислительные мощности, которые еще десять-пятнадцать лет назад могли себе позволить лишь немногие научные учреждения и обслуживание которых требовало целого штата специалистов, сегодня доступны практически каждому инженеру.

Однако эти мощности требуют соответствующего программного обеспечения. И именно в этой области, несмотря на то что аппаратные ресурсы стали значительно более доступны, наблюдаются значительные проблемы [25].

В СССР достижения в области производства ПО были значительно лучшими. Тому способствовали следующие объективные предпосылки:

 плановая организация производства оптимально сочеталась с каскадной моделью разработки ПО;

 контроль успешности проекта был ориентирован не на удовлетворение требований заказчика, а на удовлетворение изначально согласованного ТЗ;

 разработкой ПО занимались, как правило, высококвалифицированные специалисты в специализированных институтах;

 поскольку проекты в основном ориентировались на ВПК, бюджеты были фактически не ограниченными (по сегодняшним меркам).

Но по ряду причин советская школа разработки ПО прекратила свое развитие и многие достижения были утрачены. В рыночных условиях (быстро меняющиеся требования, ограниченные бюджеты, ориентация на результат, острая конкуренция за высококвалифицированный персонал) использование старых наработок советской школы оказалось ограничено очень узкими областями.

1. Технологии

Термин «технология» – он подчеркивает аналогию между созданием программного продукта и промышленным производством. Он отражает современную тенденцию к вводу дисциплины, организации и инструментирования в такой творческий процесс, как программирование. Слово фиксирует ту точку зрения, что программирование, несмотря на интеллектуальность и творческий характер этой деятельности, нуждается в организации и регламентировании, наборе соглашений и правил, не говоря уже об инструментальном обеспечении. Сейчас это кажется тривиальным, но в 60-е годы такую точку приходилось отстаивать. Да и сейчас порой возникают трения на почве регламентирования деятельности разработчиков.

Сам русский термин «технология программирования» был введен русским академиком Андреем Петровичем Ершовым. Он трактовал термин

«программирование» в обобщенном виде и подразумевал все виды деятельности, выполняемые в ходе создания программных систем. На западе для определения этой деятельности использовался термин «engineering». Сейчас обобщённый термин, применимый к созданию программных средств, обозначают как «разработка» или «конструирование».

Справедлива формула:

разработка = анализ + проектирование + программирование (кодирование) + тестирование + отладка

Иногда сюда также включают “сопровождение”. Чтобы подчеркнуть промышленно-производственный аспект, говорят о “технологии разработки” или “технологии конструирования”.

2. Этапы развития

Лишь в начале 90-х Британское сообщество вычислительной техники (British Computer Society) начало присваивать разработчикам программ звание инженера. В США только в 1998 году стало возможным хоть где-то зарегистрироваться в качестве профессионального инженера программного обеспечения. Но по-прежнему, даже в начале нынешнего века, общепризнанным остается тот факт, что разработке программного обеспечения не достает достаточно развитой научной базы. По некоторым оценкам, 75% организаций, занимающихся разработкой программ, делают это на примитивном уровне.

С момента зарождения технология разработки программ испытала несколько подъемов в своем развитии. Один из них связан с публикацией письма Эдстера Дийкстры (Edsger Dijkstra) в Ассоциацию вычислительной техники, озаглавленного так: «О вреде использования операторов GOTO» (GOTO statement considered harmful, 1968). В те времена программы писались

с активным использованием операторов безусловного перехода. Обращая внимание на недостатки таких программ, Дийкстра предложил свою концепцию структурного программирования, позволяющую избежать использования таких операторов. Концепция Дийкстры основывалась на том наблюдении Бема и Якоби (Flow Diagrams, Turning Machines and languages with only two formation rules, 1966), что для записи любой программы в принципе достаточно только трех конструкций управления – последовательного выполнения, ветвления и цикла. То есть теоретически необходимость в использовании операторов перехода отсутствует [24, 25].

Следующий шаг в развитии структурного программирования связан с введением аппарата функций, позволяющих разбивать структурную программу на обозримые по своим размерам части. При таком подходе программа пишется в терминах вызова функций верхнего уровня, которые реализуются при помощи функций более низкого уровня. Нисходящее программирование еще так же называли модульным программированием.

Структурным программам недоставало одного важного свойства – в их структуре непосредственно не отображалась сущность предметной области. Из-за этого было трудно их модифицировать в условиях изменяющихся требований. Позднее возникла парадигма объектной ориентированности. Она основана на использовании объектов, объединяющих в себе данные и функциональность. На ОО парадигме основаны многие современные языки и системы программирования.

3. Методы проектирования

Говорят, что Генри Форд совершил революцию в производстве автомобилей, когда заметил, что узлы автомобиля можно стандартизировать, так что при сборке автомобилей данной модели можно будет использовать любой экземпляр требуемого узла.

Столь же важным в настоящее время признается возможность при разработке одних приложений заимствовать идеи, архитектуру, проект и исходный код других приложений. Если приложения проектируются таким образом, что различные их части могут быть использованы многократно, то в конечном итоге это приводит к уменьшению стоимости разработки приложений.

Однако, чтобы это было возможным, приложения должны быть модульными. Модульность приложения, собственно, и означает, что оно состоит из легко идентифицируемых и заменяемых частей. По этой причине при правильном проектировании программного продукта особое внимание должно уделяться модульности, особенно на стадии разработки архитектуры.

К формальным методам проектирования относятся те методы, которые основаны на математике. Формальные методы помогают решить задачи обеспечения надежности программ. Они могут быть применены как при анализе требований для обеспечения точности формулировки требований, так и в процессе реализации для обеспечения соответствия кода программы сформулированным требованиям. Как правило формальные методы используют математику в ее логическом аспекте. В вычислительном же аспекте математика задействована в связи с использованием метрик, которые мы будем рассматривать далее.

Сегодня существует огромное количество различных процессов для создания ПО. Тем не менее, именно технологий, рассматривающих полный жизненный цикл проекта разработки ПО, сочетающих в себе научный подход, серьезную базу исследований и имеющих историю реального использования и адаптации, относительно немного.

Из методологий и технологий, получивших определенное признание на данный момент, можно назвать следующие: Datarun, CMM, Microsoft Solution

Framework (MSF), Oracle Method, Rational Unified Process (RUP), SADT (IDEFx).

Особое место в этом списке занимает технология компании Rational Software. В ее методологии применен наиболее современный процессно-ориентированный подход: так как разработка ПО является производством, то, как и на всяком производстве, при выявлении проблем в продукции (симптомов) необходимо корректировать процесс (устранять причины). Особенностью этой технологии является то, что в ее создании участвуют ведущие методисты в области разработки ПО, такие как Г. Буч (ООАП), Дж. Рамбо (OMT), А. Джекобсон (Objectory), внесшие весомый вклад в теорию и практику разработки современного ПО. Кроме того, следует заметить, что эта технология развивалась и проходила проверку с участием военного ведомства США [2, 24].

4. Этапы и элементы процесса разработки

В 80-е и 90-е в области разработки ПО преобладали две тенденции. Одна – это быстрый рост приложений, в том числе создаваемых для Web. Другая – расцвет инструментальных средств и парадигм (подходов к проектированию).

Несмотря на появление новых тенденций, основные этапы разработки ПО остались неизменными:

- Определение процесса разработки ПО;
- Управление проектом разработки;
- Описание целевого программного продукта;
- Проектирование продукта;
- Разработка продукта;
- Тестирование частей;

 Интеграция частей и тестирование продукта в целом;

 Сопровождение продукта.

Разработчики меняют последовательность проработки этих направлений. В реальности разработка ПО обычно определяется требуемым набором функций или сроком сдачи проекта. В результате, только хорошо организованные группы инженеров, владеющих методами разработки ПО, способны правильно построить работу. В противном случае разработчиков обычно ожидает хаос.

Система разработки ПО включает в себя 4 “П” (Персонал, процесс, проект, продукт)[15, 24].

Персонал – те, кем это делается. Команда разработчиков наилучшим образом работает, если каждый участник знает, что он должен делать, и имеет определенные обязанности. Другая сторона аспекта персонала – это заинтересованные в проекте лица: заказчиками, пользователи и инвесторы. В любом производстве результаты определяются используемой технологией. В силу специфичности производства ПО (практически нулевая стоимость тиражирования, очень быстрое моральное старение и т.д.) технология его создания сильно зависит от качества команды разработчиков, поэтому должна включать в себя организационный и управленческий аспекты.

Процесс – способ, которым это делается. Выделяют: водопадный процесс, итеративный процесс, XP. Индивидуальный процесс разработки (Personal Software Process), командный процесс разработки (Team Software Process). Модель зрелости возможностей (Capability Maturity Model) для оценки возможностей команды разработчиков.

Проект – совокупность действий, необходимая для создания артефакта. Проект включает контакт с заказчиком, написание документации, проектирование, написание кода и тестирование продукта.

Продукт – это не только программное обеспечение, но и все составляющие его артефакты. Под артефактами понимается объектные модули, исходный код, документация, результаты тестов и измерений продуктивности.

Качество – *приложения должны удовлетворять заранее определенному уровню качества.*

Для достижения требуемого уровня качества применяются следующие методы:

 инспектирование (процесс проверки качества, ориентированный на команды разработчиков. Он применяется на всех этапах разработки);

 формальные методы (доказательство правильности – математическое или логическое);

 тестирование;

 методы управления проектом [3, 24].

5. Инструментарий технологии программирования

Инструментарий технологии программирования – совокупность программ и программных комплексов, обеспечивающих технологию

разработки, отладки и внедрения создаваемых программных продуктов.

Группы программных продуктов

5.1. Средства для создания приложений

Средства для создания приложений – локальные средства, обеспечивающие выполнение отдельных видов работ по созданию программ, делятся на:

- *языки и системы программирования;*
- *инструментальная среда пользователя.*

Язык программирования – формализованный язык для описания алгоритма решения задачи на компьютере.

Они делятся на классы:

- *машинные языки – языки программирования, воспринимаемые аппаратной частью компьютера (машинные коды);*
- *машинно-ориентированные языки – языки программирования, которые отражают структуру конкретного типа компьютера (ассемблеры);*
- *алгоритмические языки – не зависящие от архитектуры компьютера языки программирования для отражения структуры алгоритма (Паскаль, бейсик, Фортран и др.);*

 процедурно–ориентированные языки – языки программирования, где имеется возможность описания программы как совокупности процедур (подпрограмм).

 проблемно–ориентированные языки – предназначены для решения задач определенного класса (Lisp);

Другой классификацией языков является их деление на языки, ориентированные па реализацию основ структурного программирования, основанного на модульной структуре программного продукта и типовых управляющих структурах алгоритмов обработки данных различных программных модулей, и объектно-ориентированные языки, поддерживающие понятие объектов, их свойств и методов обработки.

Системы программирования включают:

 компилятор (транслятор);

 интегрированную среду разработки программ (не всегда);

 отладчик;

 средства оптимизации кода программ;

 набор библиотек;

 редактор связей;

 сервисные средства (утилиты) (для работы с библиотеками, текстовыми и двоичными файлами);

 справочные системы;

 систему поддержки и управления продуктами программного комплекса.

Компилятор транслирует всю программу без ее выполнения.

Трансляторы (интерпретаторы) выполняют пооперационную обработку и выполнение программы.

Отладчики – специальные программы, предназначенные для трассировки и анализа выполнения других программ.

Трассировка – это обеспечение выполнения в пооператорном варианте.

Инструментальная среда пользователя – это специальные средства, встроенные в пакеты прикладных программ, такие, как:

- библиотека функций, процедур, объектов и методов обработки;
- макрокоманды;
- клавишные макросы;
- языковые макросы;
- конструкторы экранных форм и объектов;
- генераторы приложений;
- языки запросов высокого уровня;
- конструкторы меню и др.

Интегрированные среды разработки программ объединяют набор средств для их комплексного применения на технологических этапах создания программы.

5.2. Средства для создания информационных систем (Case–технология)

CASE-технология (CASE – Computer-Aided System Engineering) – программный комплекс, автоматизирующий весь технологический процесс анализа, проектирования, разработки и сопровождения сложных программных систем. Средства CASE-технологий делятся на:

- *встроенные в систему реализации – все решения по проектированию и реализации привязки к выбранной СУБД;*
- *независимые от системы реализации – все решения по проектированию ориентированы на унификацию (определение) начальных*

этапов жизненного цикла программы и средств их доку-ментирования, обеспечивают большую гибкость в выборе средств реализации.

Основное достоинство CASE-технологии – это поддержка коллективной работы над проектом за счет возможности работы в локальной сети разработчиков, экспорта (импорта) любых фрагментов проекта, организованного управления проектами. В некоторых CASE-системах поддерживается создание каркаса программ и создание полного продукта.

Технология разработки программного обеспечения

Поговорим о жизненном цикле программ, организационных и вспомогательных моментах при разработке ПО, пройдёмся по основным этапам создания программных продуктов, а также коснёмся некоторых моделей жизненного цикла программ.

Технология разработки программного обеспечения (ПО) – это комплекс мер по созданию программных продуктов (ПП). Данная деятельность включает в себя несколько этапов, с которыми так или иначе придётся столкнуться при разработке достаточно крупного ПО.

Перечислим основные **этапы жизненного цикла программы** и дадим краткую характеристику каждому из этапов. Всякая разработка включает в себя:

- **Процесс приобретения**. Данный процесс представляет собой действия заказчика разработки ПО, и обычно включает в себя такие мероприятия, как: формирование требований и ограничений к программному продукту (ограничения могут быть связаны с выбором программной

архитектуры, а также с приемлемым быстродействием системы и т.д.); заключение договора на разработку; анализ и аудит работы исполнителя. В конце данного процесса заказчик осуществляет приёмку готового программного продукта.

- **Процесс поставки** включает в себя мероприятия, проводимые исполнителем по поставке ПО. Исполнитель анализирует требования заказчика, выполняет проектирование и анализ работ, решает, как будет происходить процесс конструирования (программирования): своими силами, либо же с привлечением сторонних команд разработки (подрядчика), также осуществляет оценку и контроль качества готового программного продукта и выполняет непосредственно поставку продукта и сопутствующие завершающие мероприятия.
- **Процесс разработки**. Его мы подробно рассмотрим в разделе “Этапы создания программных продуктов”.
- **Процесс эксплуатации**. После того, как программное обеспечение будет готово, начинается процесс его эксплуатации организацией-заказчиком и её операторами.
- **Процесс сопровождения**. Фирма-разработчик осуществляет поддержку пользователей программного продукта в случае возникновения у них каких-либо вопросов или проблем. Если в процессе эксплуатации будет обнаружена ошибка в ПП, разработчики должны её устранить. Процесс эксплуатации и процесс сопровождения идут параллельно.

Вспомогательные процессы

Технология разработки программ в рамках жизненного цикла программного обеспечения включает в себя ряд вспомогательных процессов. Рассмотрим их.

- **Процесс документирования**. В процессе разработки и далее исполнитель пишет документацию и руководства пользователя к

разрабатываемому программному продукту. Данные документы помогут разработчикам [вспомнить/разобраться] структуру и код ПО (ибо со временем всё забывается, особенно в больших проектах), а пользователям освоить работу с программой.

- **Процесс управления конфигурацией**. Данный процесс включается в себя работы по управлению наборами разрабатываемых компонентов ПО и по управлению версиями ПП.
- **Процесс обеспечения качества**. Он отвечает за то, чтобы разрабатываемый программный продукт соответствовал предварительным требованиям к разработке, а также стандартам организаций исполнителя и заказчика.
- **Процесс верификации**. Нужен для того, чтобы выявить ошибки внесённые в ПО во время конструирования, а также выявить несоответствия разрабатываемого ПО выработанной архитектуре.
- **Процесс аттестации**. Процесс направлен на подтверждение соответствия получаемых величин эталонным. То есть, выходные данные должны иметь погрешность, удовлетворяющую требованиям и установленным стандартам.
- **Процесс совместной оценки**. Нужен для контроля и проверки состояния персонала и разрабатываемого программного продукта. Выполняется обеими сторонами (заказчиком и исполнителем) на протяжении времени всех работ по проекту.
- **Процесс аудита**. Аудит направлен на независимую оценку текущих положений, состояния проекта, документации и отчетов. При аудите выполняется сравнение с договором и документами, определяющими стандарты. Может выполняться также обеими сторонами.
- **Процесс разрешения проблем**. Реализует устранение недочётов, выявленных во время всех процессов связанных к контролем и оценкой.

Организационные процессы жизненного цикла программного продукта

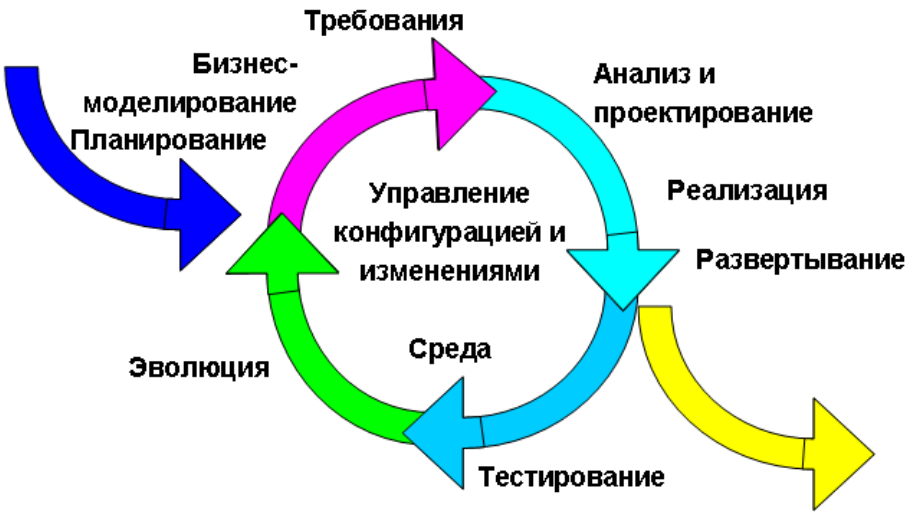
Существует и проводится ряд мер, направленных на повышение организации и качества разработки программного обеспечения. Они называются **организационными процессами жизненного цикла**. Обычно их выделяют четыре вида, и мы рассмотрим каждый.

Организационные процессы жизненного цикла программного обеспечения включают:

- **Процесс управления**, который направлен на грамотное и эффективное управлением персоналом компании-исполнителя. За это отвечают люди, находящиеся на руководящих постах, а также специальный отдел в фирме.
- **Процесс создания инфраструктуры**. Разработка программных продуктов требует наличия огромного количества инфраструктурных компонентов: компьютеров, серверов, специальных программ для разработки и т.д. Кроме того, готовый продукт требует наличия определённых единиц для его работы. Данный процесс необходим для подготовки оборудования и ПО для разработчиков, а также для успешного функционирования готового ПП у заказчика.
- **Процесс усовершенствования**. Направлен на усовершенствование *всех остальных* процессов жизненного цикла программного обеспечения. Усовершенствование может повысить производительность разработчиков и добиться большей выгоды от выполнения заказа на производство программы.
- **Процесс обучения**. Постоянное обучение сотрудников и повышение их квалификации – это залог производства качественных продуктов и программ. Процесс обучения направлен на организацию

мероприятий для повышения уровня и получения новых навыков сотрудниками компании-разработчика.

Этапы создания программных продуктов



Приведём все основные этапы создания программного продукта. Всего их пять. Они так или иначе характерны для любой методологии разработки ПО: будь то классическая водопадная, либо современные гибкие методологии (Agile software development) – во всех из них разработчики проходят через следующие этапы создания программного обеспечения:

1. **Составление требований заказчика**. На данном эта производится работа с заказчиком и документирование его видения и его требований к программе. В подавляющем большинстве случаев данный этап проходит трудно. Поскольку, слабо разбираясь в особенностях разработки ПО, заказчик плохо представляет себе, что нужно знать разработчикам и (самое главное!), что им нужно сообщить о продукте. *Выработка требований чрезвычайно важное мероприятие*. Убедитесь, что все требования полностью понятны вам и вашей команде.

2. **Проектирование программного продукта**. Разобравшись в предметной области, разработчики приступают к проектированию. На данном этапе создания программного продукта разрабатывается архитектура компонентов ПО, выбираются нужные шаблоны проектирования (паттерны) и составляется схема информационной базы данных системы.

3. **Разработка**. Когда требования сформулированы и архитектура готова – команда начинает разработку ПП. На этапе разработки также выполняется *документирование системы*.

4. **Тестирование**. После разработки необходимо произвести тестирование системы в целом, тем самым подтвердить её соответствие требованиям заказчика.
Здесь стоит сказать, что модульные тесты (unit-тесты; т.е. тесты отдельных частей программы) обычно выполняются на этапе разработки программистом, разрабатывавшем конкретный модуль. Когда все тесты пройдены, программное обеспечение готово к выпуску.

5. **Сопровождение ПП**. После выпуска фирма-разработчик отвечает за поддержку программного продукта и выпуска новых версий, которые исправляют ошибки и привносят новый функционал. Также необходимо осуществлять поддержку пользователей разработанного ПО.

Примечание 1: Следует как можно тщательнее подходить к формированию предварительных требований и проектированию, поскольку стоимость исправления ошибок после выпуска ПО, допущенных на этих этапах, обычно в 2-10 (!) раз выше, чем стоимость исправления ошибок сделанных на этапе программирования (Стив Макконнелл “Совершенный код”).

Примечание 2: Очень часто случается, что заказчик уже после составления требований к ПО (т.е. во время проектирования и разработки) объявляется и радостно сообщает исполнителю свои новые идеи или

рассказывает о какой-нибудь “классной” функции, которую нужно добавить в приложение… Бывают случаи, когда это труднореализуемо и сопряжено с пересмотром архитектуры. В данной ситуации можно посоветовать сказать разработчику примерно следующее: “Отлично придумано! Мне нравится! Тогда я пересмотрю свою смету и сроки работы и потом сообщу Вам!”. Практически всегда это срабатывает и гасит пыл заказчика, и он отказывается от новых идей и изменений в проекте.

Модели жизненного цикла

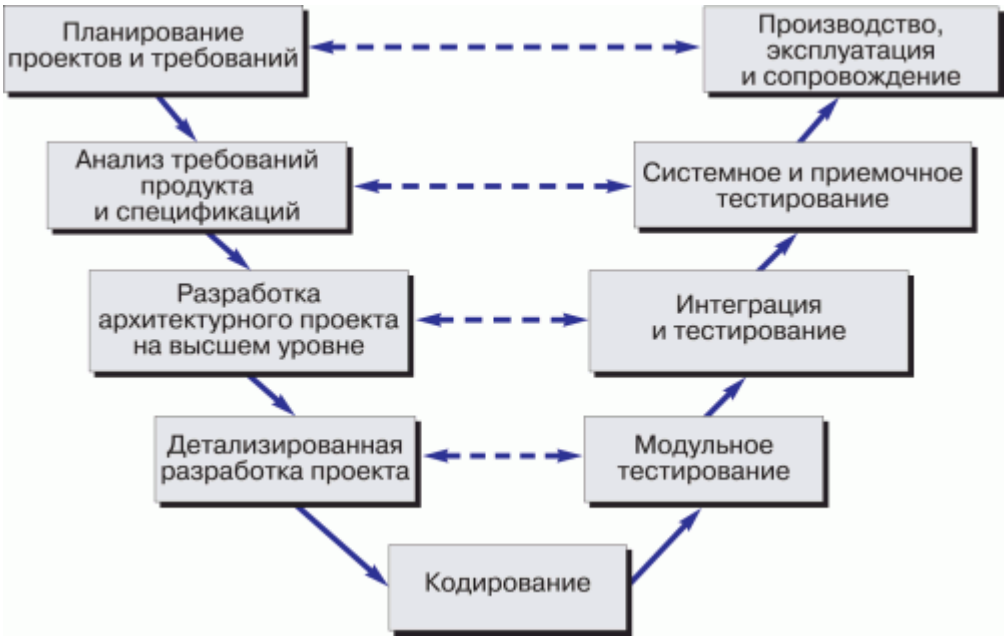
Модель жизненного цикла программного обеспечения характеризует подход команды к разработке ПП. Она отражает акценты и приоритеты во всём процессе изготовления программы, а самое главное, порядок следования этапов создания программных продуктов.

На сегодняшний день существует множество моделей жизненного цикла разработки программного продукта. Мы кратко рассмотрим основные из них и выделим их ключевые особенности.

Каскадная (водопадная) модель

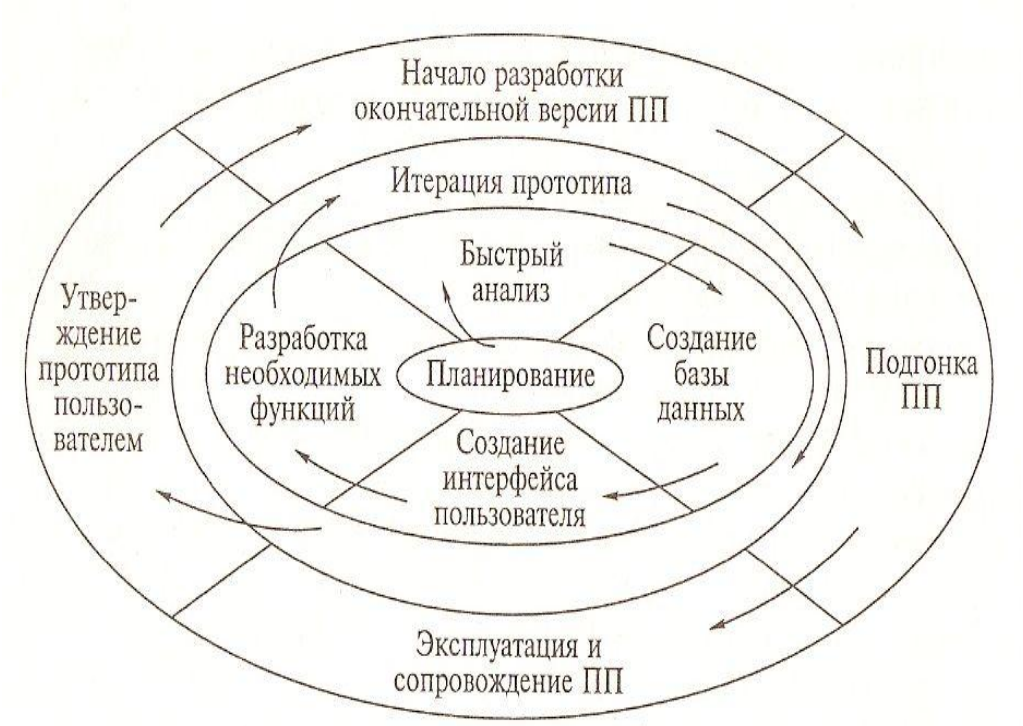
Каскадная (водопадная) модель строго следует последовательности всех этапов разработки ПО и не предполагает возвращения с текущего этапа на предыдущий. Сейчас данная модель практически не используется, разве что в очень малых проектах.

V-образная модель разработки



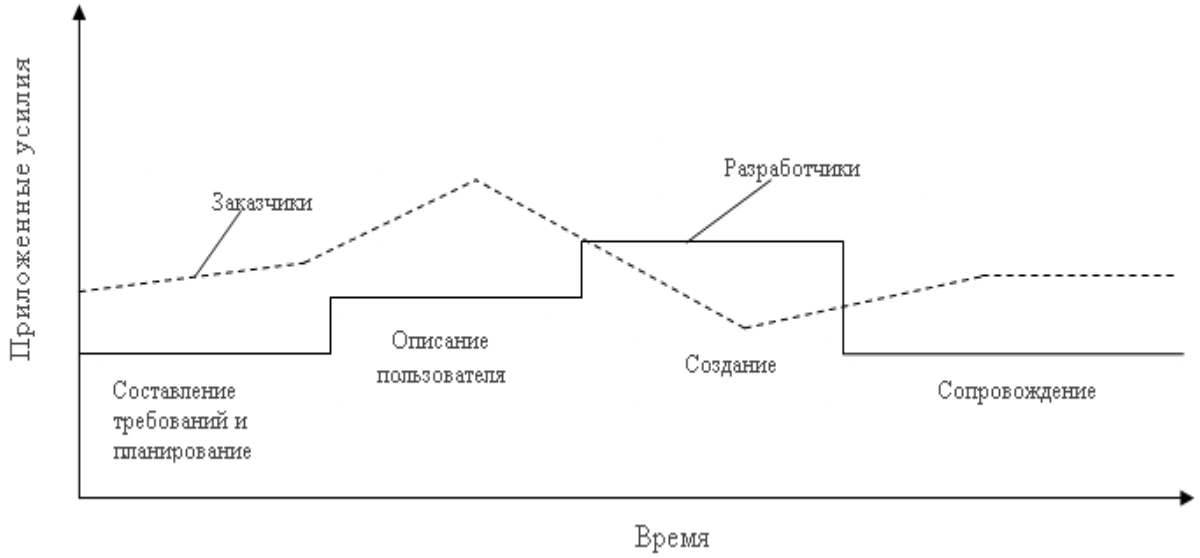
По рисунку можно проследить, что в **V-образной модели** имеется возможность вернуться на некоторые этапы разработки и уточнить нужные требования.

Модель прототипирования



Прототипирование предполагает создание на протяжении всего процесса разработки несколько рабочих версий программы (прототипов) с неполным функционалом. В первом прототипе может быть реализован исключительно один интерфейс приложения.

Модель быстрой разработки (RAD-модель)

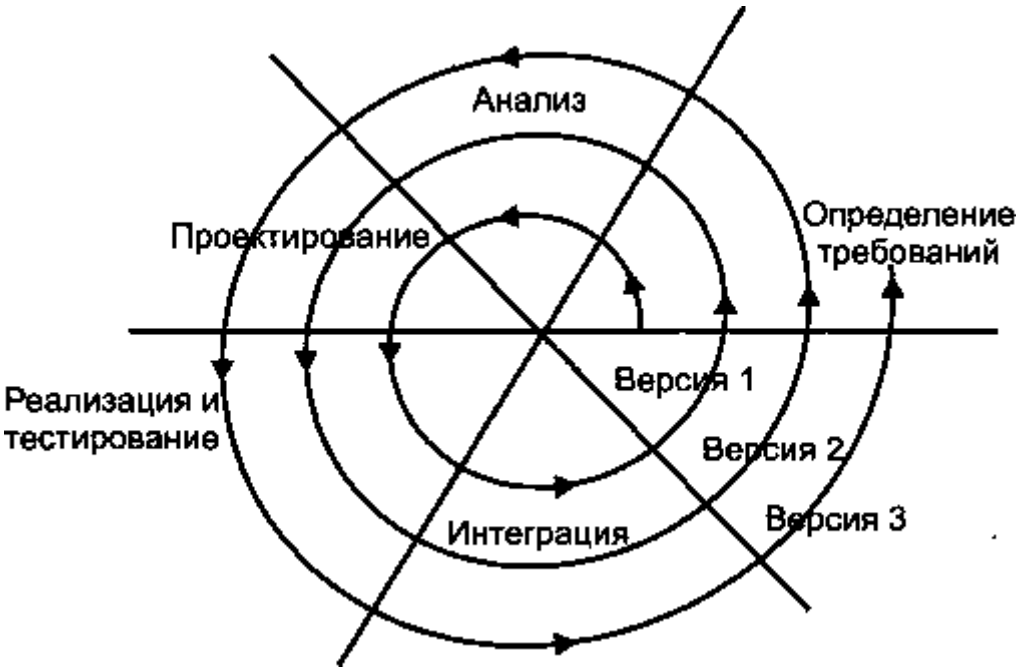


RAD-модель (rapid application development — **быстрая разработка приложений**) ориентирована в первую очередь на быстроту и удобство программирования. Команда делает акцент именно на разработке, а большая часть работы по составлению требований и описанию пользователей возлагается на заказчика.

Итерационная модель

В **итерационной модели** всегда имеется возможность вернуться на любой предыдущий этап разработки ПО для уточнений требований и исправления компонентов. Здесь главное вовремя остановиться, ведь итерации не могут продолжаться бесконечно.

Спиральная модель



В **спиральной модели** все этапы разработки последовательно повторяются по кругу до тех пор, пока текущая версия программы не станет полностью соответствовать требованиям. Здесь также нужно иметь предел и вовремя остановиться.

Гибкие методологии

Гибкие методологии (Agile) олицетворяют современные подходы к разработке ПО. Они используются обычно в небольших командах разработчиков. Среди них такие модели жизненного цикла программного продукта, как Scrum, DSDM, XP, FDD и другие.

Практическая работа №1

1. Теоретические основы разработки программного обеспечения: модели жизненного цикла разработки ПО.

Задание: хорошо разобраться с теорией! Сделать сравнительную таблицу:

	Каскадная (водопадная) модель	**V-образная модель разработки**	**Модель прототипирования**	**RAD-модель**	**Итерационная модель**	**Спиральная модель**	**Гибкие методологии**
Определение (сущность понятия)	1. 2. 3. 4. 5.	1. 2. 3. 4. 5.	1. 2. 3. 4. 5.	1. 2. 3. 4. 5.	1. 2. 3. 4. 5.	1. 2. 3. 4. 5.	1. 2. 3. 4. 5.
Отличительные черты (особенное)	1. 2. 3.	1. 2. 3.	1. 2. 3.	1. 2. 3.	1. 2. 3.	1. 2. 3.	1. 2. 3.
Общие черты (что эти формы объединяет?)							

Лабораторная работа №2

Построение диаграмм вариантов использования и диаграмм классов

Задание 1 Создайте действующих лиц в среде Rational Rose.

Действующие лица:

- Student (Студент) – записывается на курсы;
- Professor (Профессор) – выбирает курсы для преподавания;

- Registrar (Регистратор) – формирует учебный план и каталог курсов, ведет все данные о курсах, профессорах и студентах;
- Billing Systems (Расчетная система) – получает от данной системы информацию об оплате курсов;
- Course Catalog (Каталог курсов) – передает в систему информацию из каталога курсов, предлагаемых университетом.

Задание 2 Создайте варианты использования в среде Rational Rose.

Варианты использования

- Login (Войти в систему);
- Register for Courses (Зарегистрироваться на курсы);
- View Report Card (Посмотреть табель успеваемости);
- Select Courses to Teach (Выбрать курсы для преподавания);
- Submit Grades (Проставить оценки);
- Maintain Professor Information (Ввести информацию о профессорах);
- Maintain Student Information (Ввести информацию о студентах);
- Close Registration (Закрыть регистрацию).

Задание 3 Постройте диаграмму вариантов использования

Готовая диаграмма вариантов использования изображена на рисунке 1

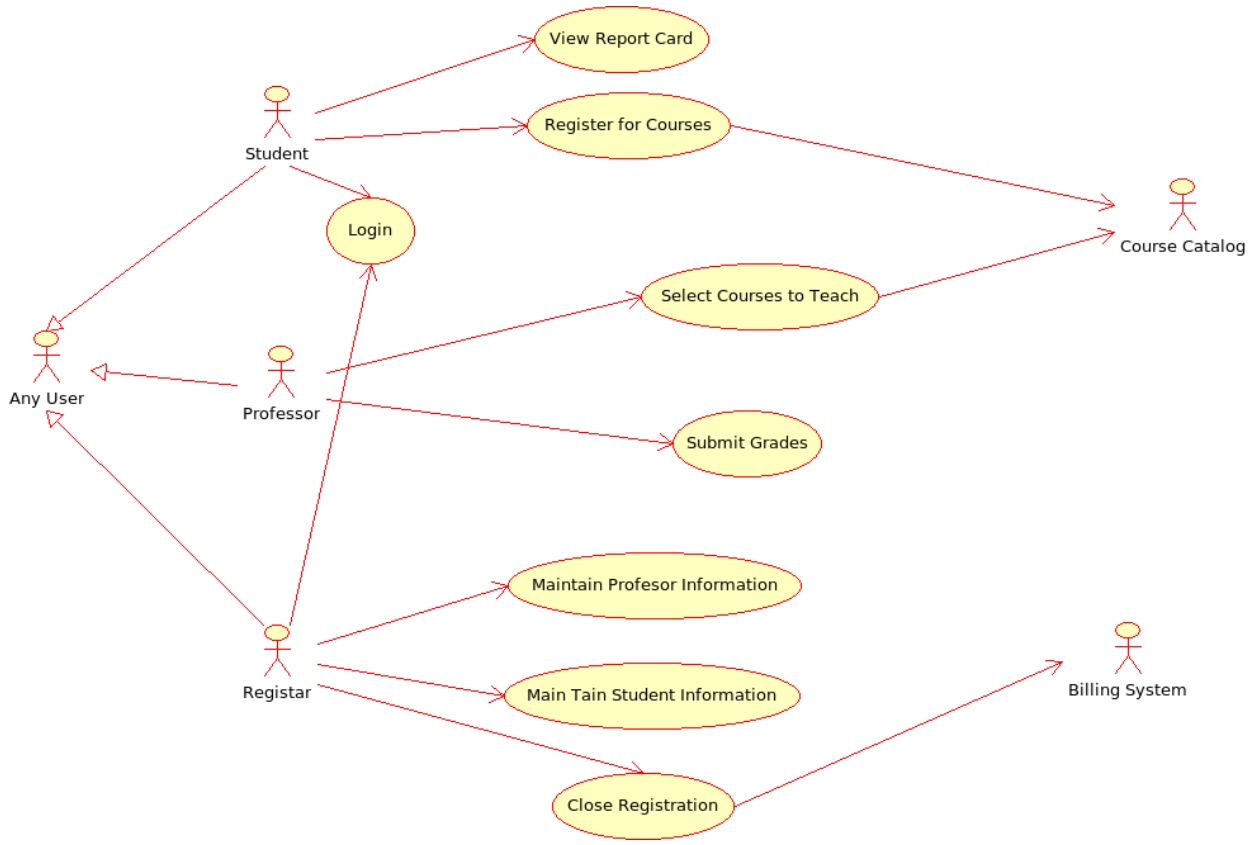


Рисунок 1 - Диаграмма вариантов использования для системы регистрации

Задание 4 Добавьте описание к вариантам использования

- Выделите в дереве вариант использования Register for Courses.
- В окне документации введите следующее описание к этому варианту использования: «This use case allows a student to register for courses in current semester»

Контрольные вопросы

1. По диаграмме вариантов использования сформулируйте постановку задачи.
2. Создайте с помощью MS Word три текстовых файла с описанием вариантов использования Login, Register for Courses, Close Registration.

Пример

Вариант использования Login

Краткое описание. *Данный вариант использования описывает вход пользователя в систему регистрации курсов.*

Основной поток событий *Данный вариант использования начинает выполнятся, когда пользователь хочет зайти в систему,*

1 Система запрашивает имя пользователя и пароль

2 Пользователь вводит имя и пароль

3 Система проверяет доступ и пароль после чего открывается доступ в систему.

Альтернативные потоки *Неправильное имя\пароль. Если во время выполнения основного потока обнаружится, что пользователь ввел неправильное имя или пароль, система выводит сообщение об ошибке. Пользователь может вернуться к началу основного потока или отказаться от входа в систему, при этом выполнение варианта использования завершается.*

Предусловия: Отсутствуют.

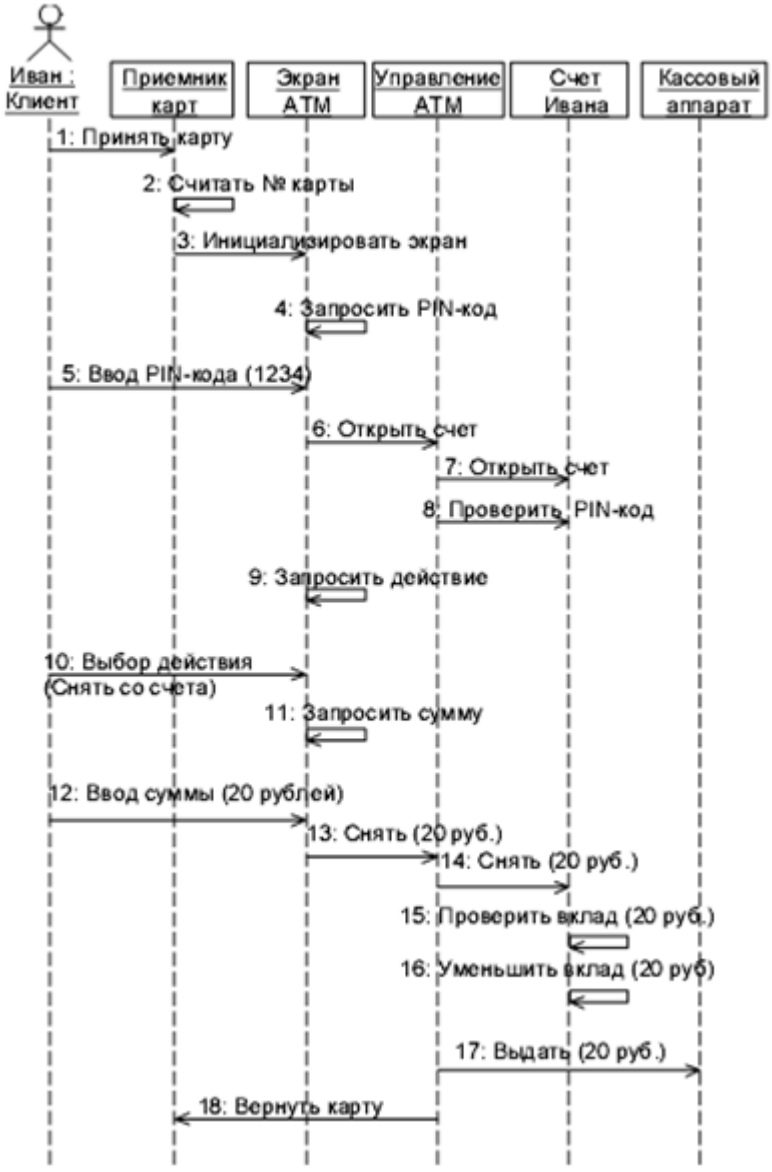
Постусловия: Если вариант использования выполнен успешно, пользователь входит в систему. В противном случае состояние системы не изменяется.

3. Обоснуйте наличие действующего лица Any User.

4. Архив, содержащий модель вариантов использования и ответы на контрольные вопросы, разместить в ЭИОС.

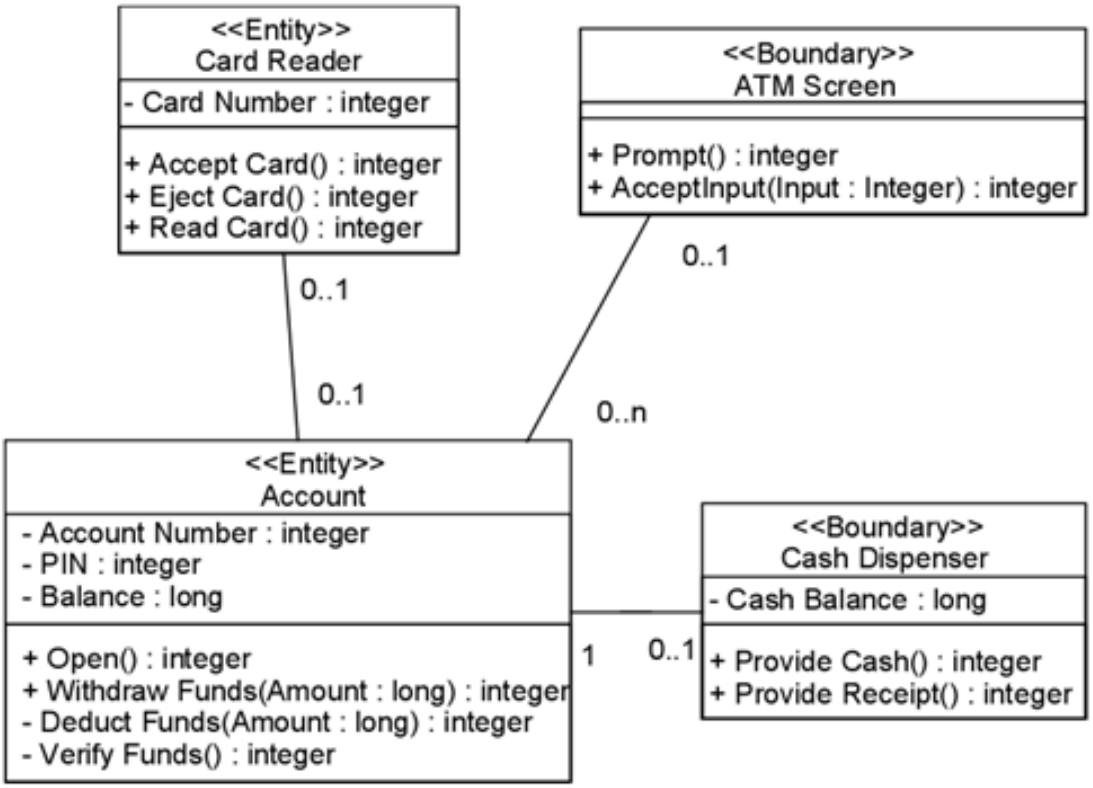
Варианты заданий

Вариант №1. Диаграмма последовательности

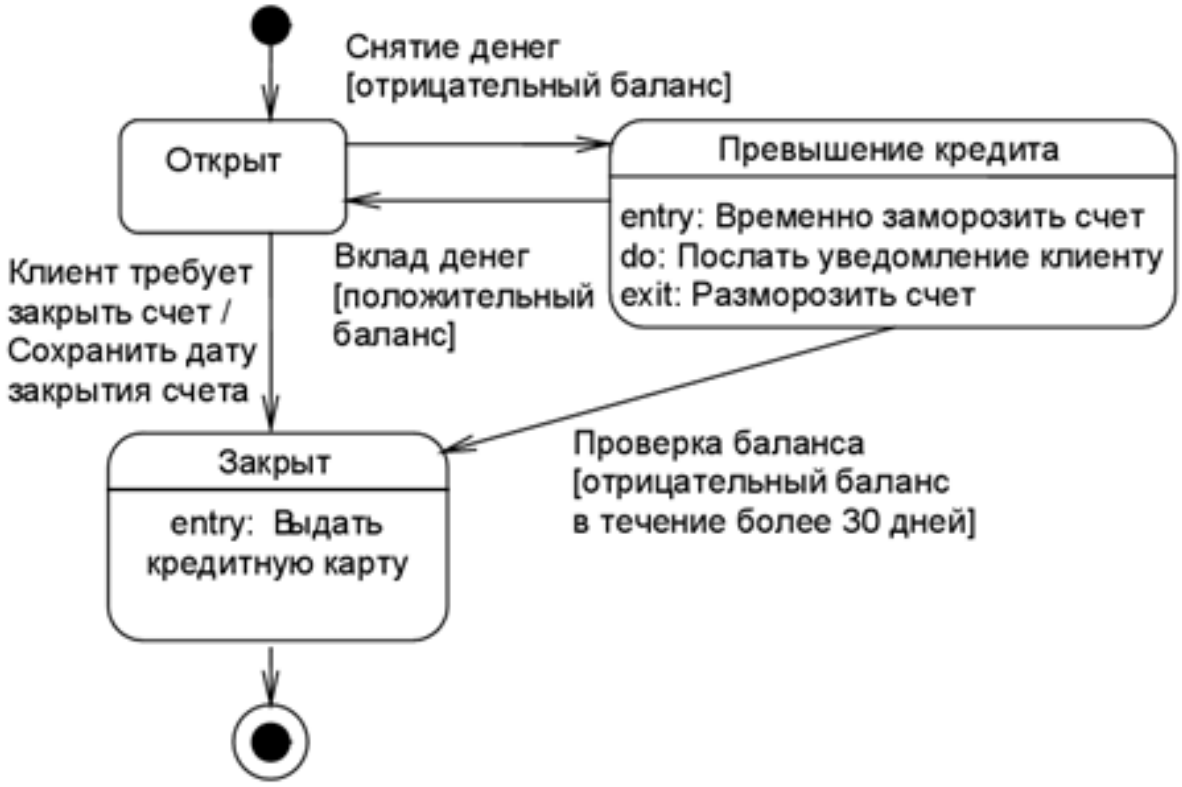


Вариант №2. Диаграммы кооперации

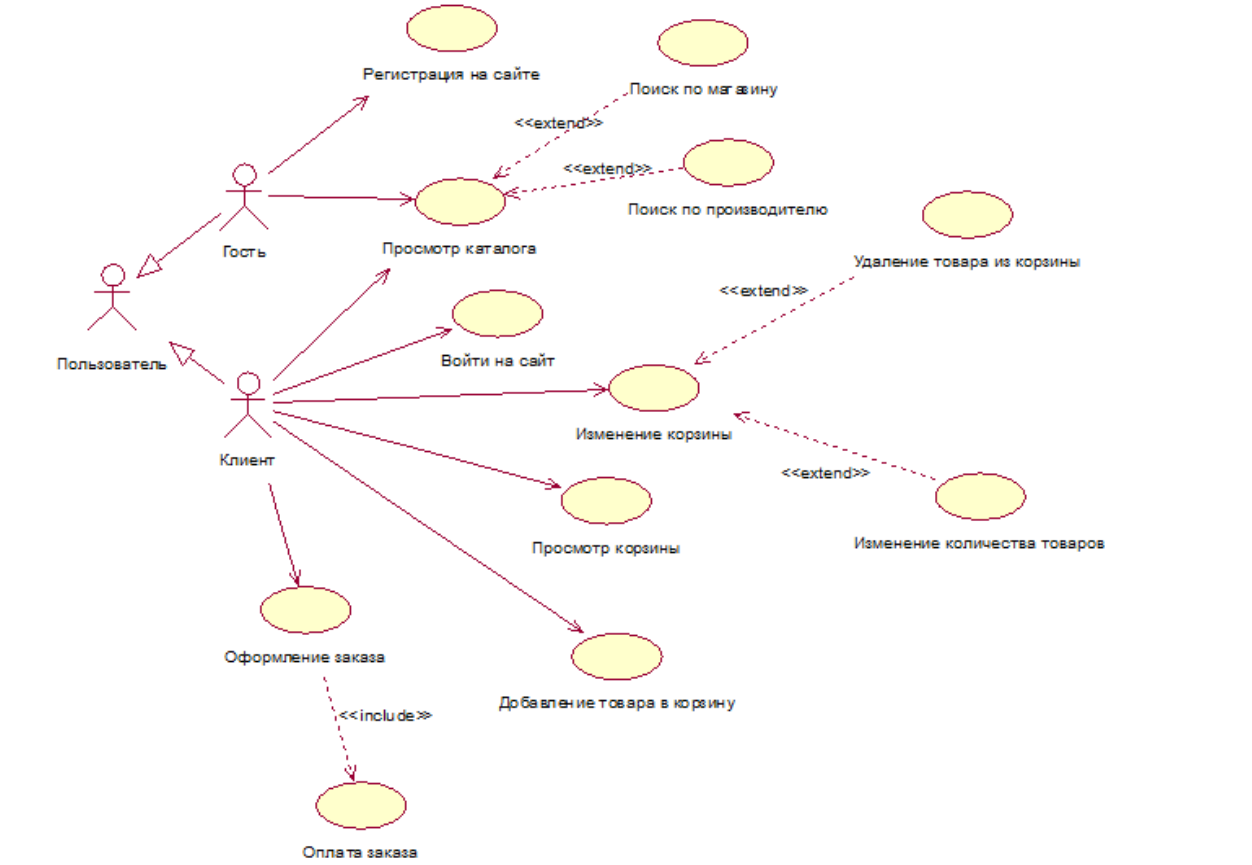
Вариант №3. Диаграмма классов



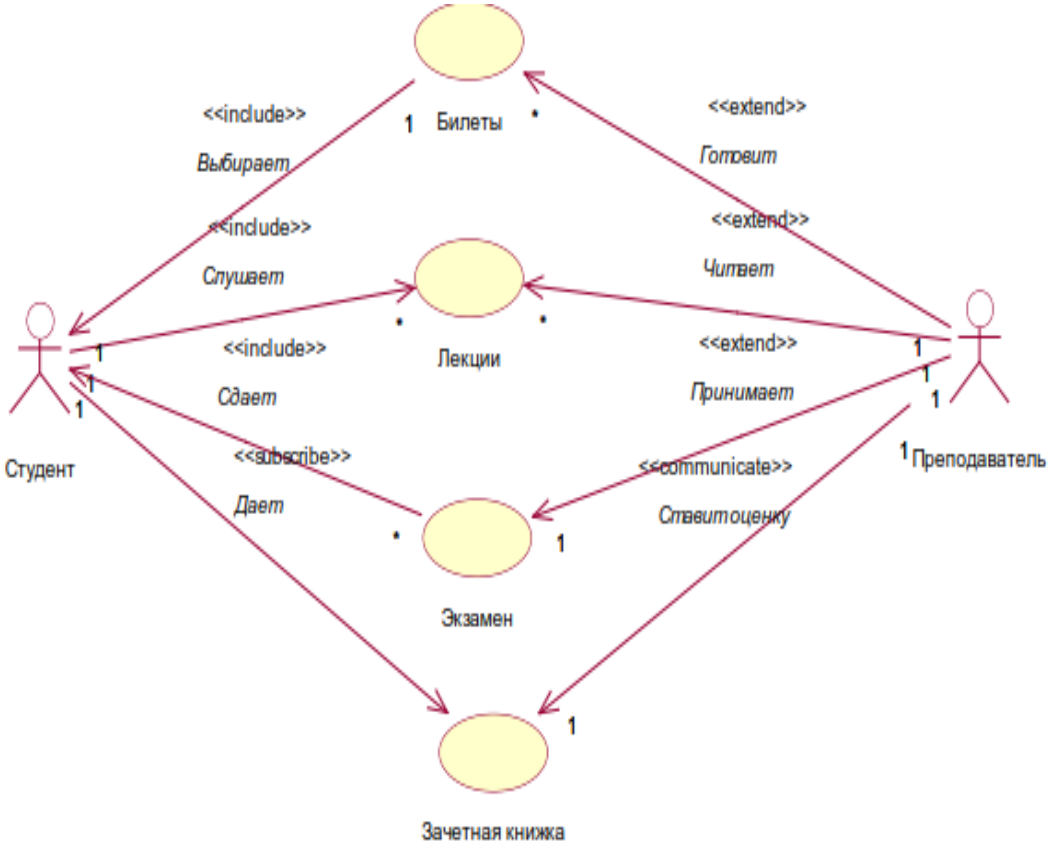
Вариант №4. Диаграмма состояний



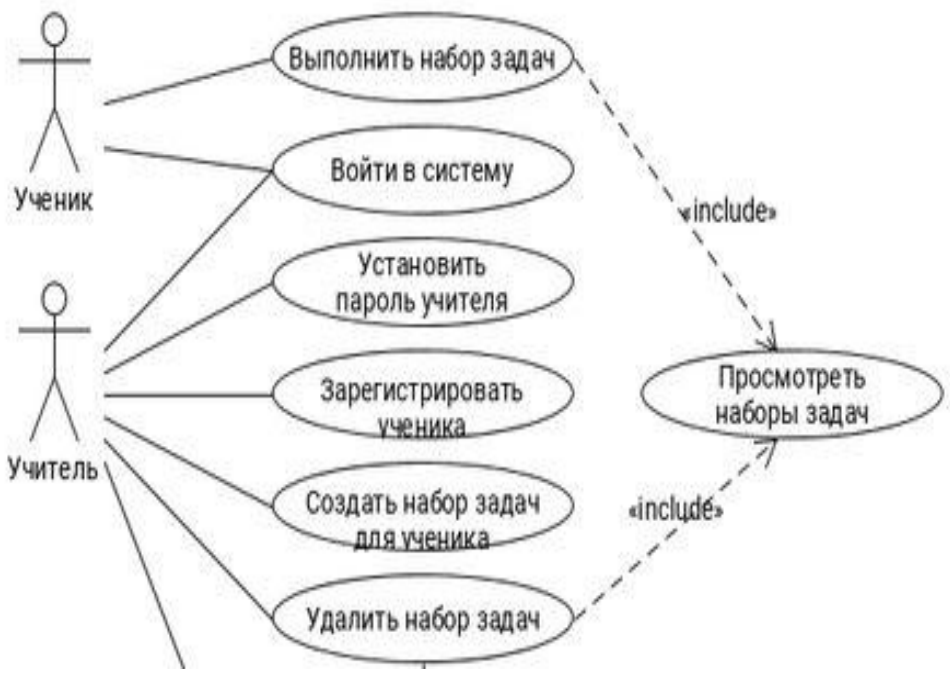
Вариант №5. Диаграмма вариантов использования



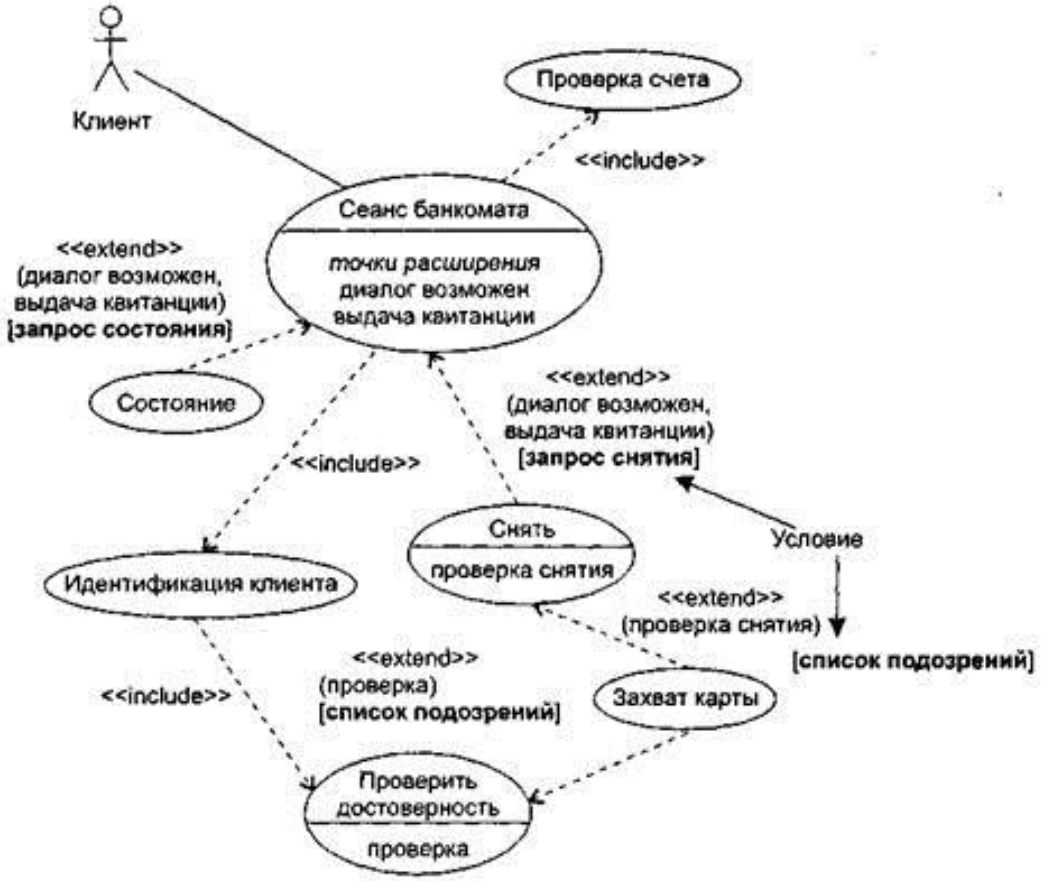
Вариант №6. Диаграмма вариантов использования



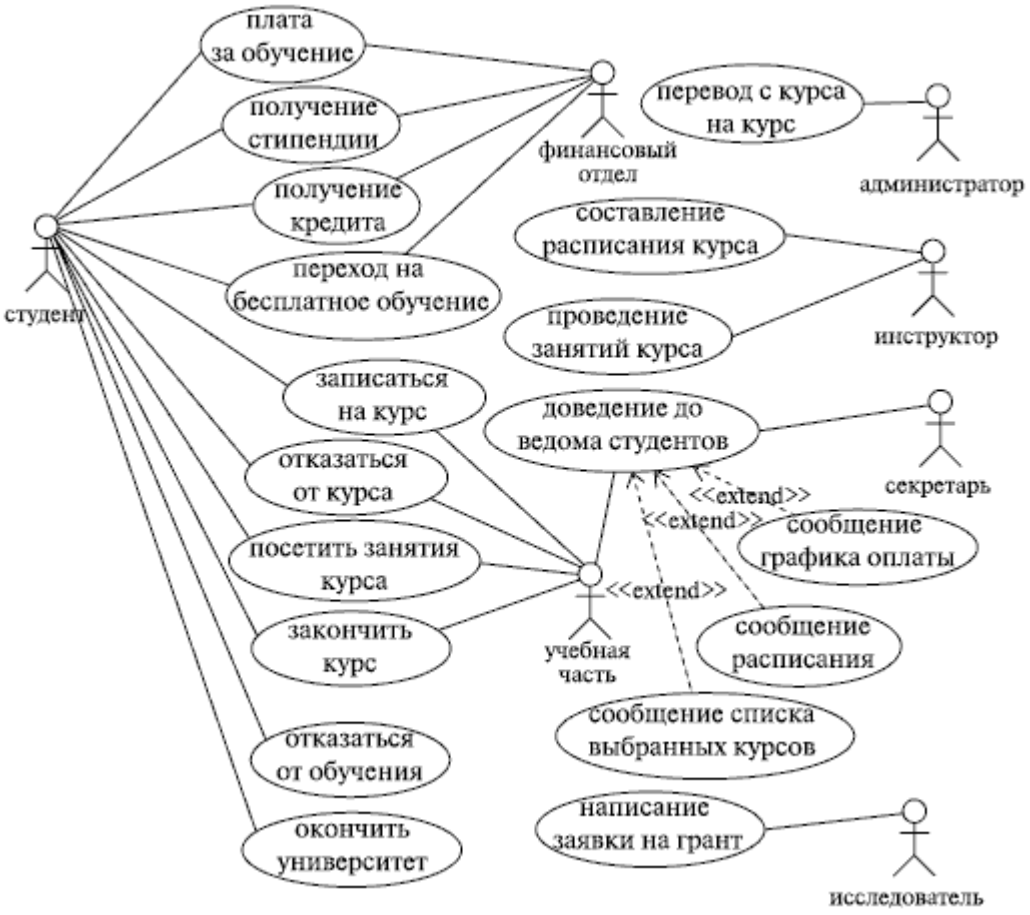
Вариант №7. Диаграмма вариантов использования



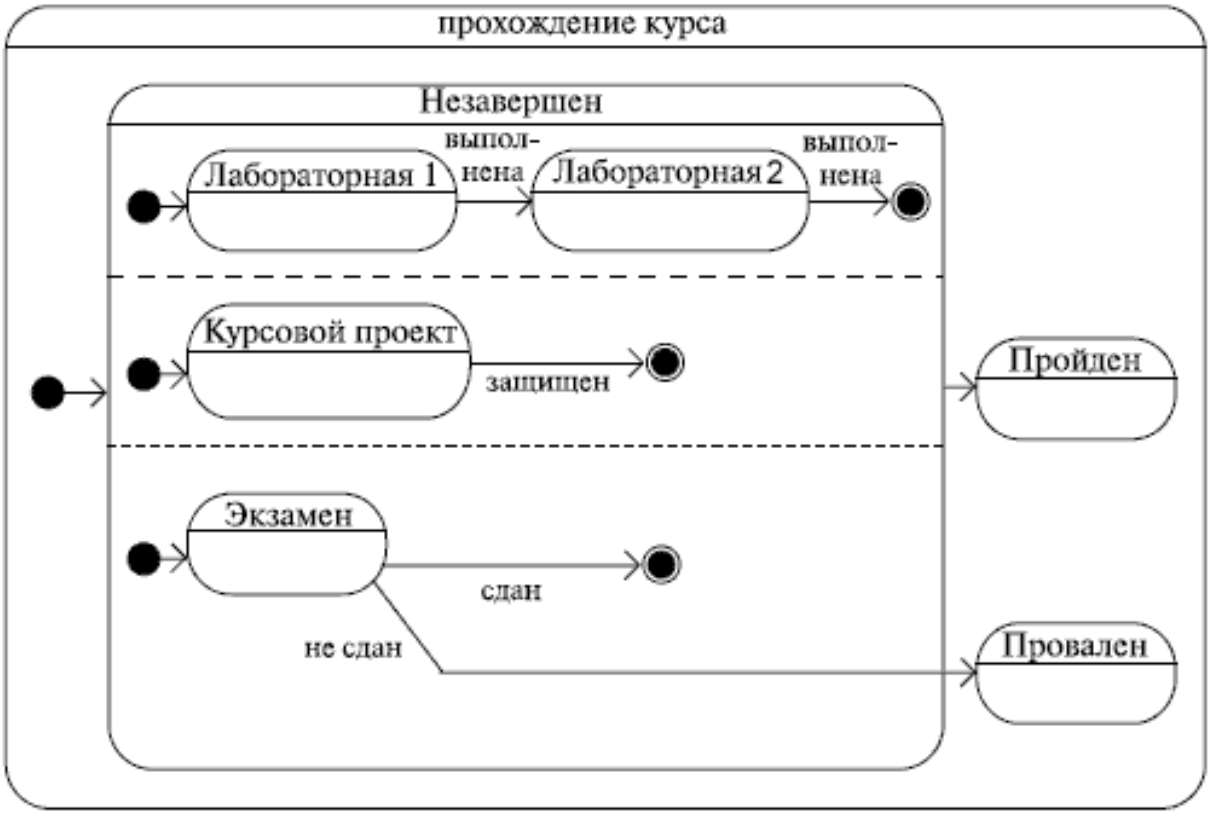
Вариант №8. Диаграмма вариантов использования



Вариант №9. Диаграмма вариантов использования



Вариант №10. Диаграмма состояний



Темы докладов по дисциплине «Современные технологии разработки программного обеспечения»

1. Анализ стилей программирования: процедурный стиль, структурный стиль, объектно-ориентированный стиль.
2. Методы выявления классов и критерии проверки правильности построения класса.
3. Особенности и объекты тестирования. Автономное и комплексное тестирование.
4. Сравнительный анализ моделей ЖЦПО с точки зрения их применимости в методологиях RUP и XP.

5. Анализ опыта применения языка UML как инструмента бизнес-моделирования
6. Сравнительный анализ жестких и гибких методологий разработки программ – XP.
7. Scrum, RAD, RUP - с точки зрения автоматизации менеджмента проекта.
8. Теория и практика сопровождения ПО – анализ основных проблем.
9. Сравнительный анализ и особенности применения моделей качества ISO, TQM, CMM, SPICE.
10. Метод бригады главного программиста и ролевое разделение работ в проекции на типовые методологии разработки программ.
11. Методы обеспечения надежности в распределенных системах.
12. Безопасность информации в среде Интернет.
13. Методы и средства разработки систем офисной автоматизации.
14. Архитектурные решения и реинжиниринг в корпоративных системах.
15. Методы и инструменты визуализации информации при построении информационных систем.