МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

(ДГТУ)

Авиационно-технологический колледж

УТВЕРЖДАЮ
Директор колледжа
_____ В.А. Зибров
личная подпись инициалы, фамилия
«___»_____2023 г.

Рег. № _____

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ

по организации и выполнению учебной практики в рамках профессионального модуля ПМ.02 «Осуществление интеграции программных модулей»

специальность **09.02.07**
Информационные системы и программирование

Ростов-на-Дону,
2023 г.

СОДЕРЖАНИЕ

1.ЗАДАЧИ УЧЕБНОЙ ПРАКТИКИ

Учебная практика по профессиональному модулю «Осуществление интеграции программных модулей» является частью всего комплекса учебных практик специальности 09.02.07 Информационные системы и программирование. Сутью практики является получение необходимых умений и навыков в овладении основ программной инженерии, умение не только кодировать уже разработанные алгоритмы, но и в полной мере уметь грамотно исследовать предметную область, выполнить анализ требований к будущей программной системе, провести проектирование, тестирование, уметь оценить результаты проектирования. Выполнение практики требует соблюдения междисциплинарных связей, а именно - подготавливается дисциплинами по основам алгоритмизации и программированию, базам данных, операционным системам и средам.

Основой практики является коллективная разработка программной системы в пределах всего жизненного цикла – от системного анализа до тестирования и отладки. Учебная группа делится на подгруппы (бригады) разработчиков – но не более 5-6 человек в бригаде. Среди разработчиков осуществляется распределение ролей в зависимости от выбранной стратегии и модели конструирования (XP-модель, модель RAD и пр.) . Выбор модели конструирования определяется характером поставленной задачи, уровнем подготовки и профессионализма разработчиков в пределах бригады. Особая роль отводится процессу анализа требований и проектированию ПП, что осуществляется с применением инструментария программной инженерии, в частности, языка моделирования UML, на котором должен быть представлен некоторый набор диаграмм – диаграммы Use Case, компонентные диаграммы и др. Выбор инструментария кодирования определяется группой разработчиков самостоятельно исходя из особенностей поставленной задачи; это может быть несколько инструментариев, включая, кроме инструментария программирования также СУБД, Access, Excel и др. Обязательное требование к языковой среде – объектно-ориентированные среды разработки.

Важное место в проведении практики занимает организация тестирования создаваемого программного обеспечения; предлагается вести учет и анализ выявленных ошибок – ошибки проектировании, ошибки кодирования (в том числе и синтаксические), особое значение

придается умению спроектировать тестовые варианты, что само по себе является зачастую не тривиальной задачей.

В процессе работы над проектом студентам предлагается проводить мониторинг с ведением дневника конструирования, в котором отмечать затраты времени на каждый этап жизненного цикла ПП, особенности этапа, фиксировать ошибки, нестандартные ситуации и пр.

По окончании конструирования ПП необходимо рассчитать некоторые метрики, оценивающие ПП и процесс его конструирования, что может быть сделано благодаря дневнику, который велся каждой группой разработчиков.

По результатам конструирования составляется отчет – один на группу разработчиков – в соответствии с предлагаемой четкой структурой.

Подведение результатов практики осуществляется в 3 этапа:

- публичная защита (доклад) своей работы каждой группой разработчиков;
- демонстрация созданного ПП на компьютере;
- проверка отчета по практике.

2.ТЕМАТИЧЕСКИЙ ПЛАН

Наименование МДК профессионального модуля	Тема учебной практики	Содержание учебного материала	Объем часов
МДК 02.01. Технология разработки программного обеспечения МДК 02.02. Инструментальные средства разработки программного обеспечения	Анализ требований к ПС	Цели практики, инструктаж. Формулировка индивидуальных заданий; Исследование и описание педметной области на естественном языке; Унифицированный язык моделирования (UML); Выявление актеров, вариантов использования (элементов Use Case) и отношений между ними (UML); Составление диаграмм вариантов использования Use Case на UML	**10**
МДК 02.01. МДК 02.02.	Проектирование ПС	Структурирование системы с определением модели управления; Проектирование архитектуры, структуры будущего проекта ПС на уровне модулей; Проектирование интерфейса; Информационное моделирование. Составление ER- модели, даталогической модели	**16**
МДК 02.01.	Кодирование ПС	Кодирование программ	**20**
МДК 02.01.	Тестирование ПП. Отладка ПП	Тестирование элементов методом «белого ящика». Отладка; Тестирование методом «черного ящика». Отладка; Использование инструментальных средств тестирования и отладки	**10**
МДК 01.01.	Расчет метрик.	Расчет функционально-ориентированных метрик созданного ПС	**4**
МДК.02.02	Составление документации	Составление и оформление технической документации	**2**

	Оформление отчета	Оформление проекта и составление отчета по практике	**4**
	Защита	Демонстрация проекта на компьютере: Публичный доклад по созданному проекту; Ответы на вопросы	**6**
	Дифференцированный зачет		
	Итого часов учебной практики УП.02		**72**

3.СОДЕРЖАНИЕ УЧЕБНОЙ ПРАКТИКИ

Введение

Цели практики. Форма организации практики. Деление на группы разработчиков. Распределение ролей в группе Выдача заданий. Инструктаж.

1. Анализ требований к ПС

Требования к знаниям: студенты должны знать модели анализа требований, язык визуального моделирования UML.

Требования к умениям: студенты должны уметь анализировать предметную область с использованием инструментария программной инженерии - составлять на UML диаграммы Use Case.

Практические работы

1. Исследование и описание предметной области на естественном языке.
2. Выявление актеров, вариантов использования (элементов Use Case) и отношений между ними.
3. Составление диаграмм вариантов использования Use Case на UML.

2. Проектирование ПС

Требования к знаниям: студенты должны знать стратегии и модели конструирования – модель RAD, модель XP-процесс, спиральная модель, модель COM.

Требования к умениям: студенты должны уметь выполнить структурирование системы на подсистемы (модель хранилище данных, клиент-серверная модель, трехуровневая модель и пр.) , выбрать модель управления между подсистемами (централизованного управления, модель менеджера, по прерываниям и пр.), выполнить модульную или объектную декомпозицию.

Практические работы

1. Структурирование системы с определением модели управления ;
2. Проектирование структуры будущего проекта ПС на уровне модулей.

3.Кодирование ПС

Требования к знаниям: студенты должны знать объектно-ориентированную среду разработки (например, Delphi), основы проектирования баз данных, дополнительные технологические возможности, такие как использование DLL, технологии OLE, Active-X.

Требования к умениям: студенты должны уметь составлять программы с осознанным использованием разнообразного инструментария.

Практические работы

1-6 . Кодирование программ.

PS. Процесс кодирования определяется тематикой проекта и выбранным инструментарием, поэтому может детализироваться индивидуально.

4.Тестирование ПС

Требования к знаниям: студенты должны знать типы тестирования и способы тестирования, типы отладки и стратегию отладки.

Требования к умениям: студенты должны уметь составлять тестовые варианты, выполнять полное комплексное тестирование проекта, выполнять отладку как аналитическими методами, так и экспериментальными.

Практические работы

1. Тестирование элементов методами «белого ящика». Отладка
2. Тестирование методами «черного ящика». Отладка.
3. Системное тестирование – стрессовое, восстановления.

5.Расчет метрик

Требования к знаниям : студенты должны знать размерно- и функционально-ориентированные метрики, метрики объектно-ориентированных систем.

Требования к умениям: студенты должны уметь рассчитывать производительность, качество ПС, произвести анализ процесса конструирования.

Практические работы

1. Расчет метрик ПС.

6. Оформление отчета

Отчет по практике оформляется один на группу разработчиков по строгой структуре и содержит описание процесса конструирования ПС, результатов, расчет метрик, выводы. Оформление осуществляется в соответствии с межгосударственными стандартами оформления текстовой информации.

Практические работы

1. Составления отчета по практике.

7.Защита

Защита практики проводится в 3 этапа:

- Публичный доклад по созданному проекту
- Демонстрация проекта на компьютере
- Проверка отчета по практике

4.ВЫПОЛНЕНИЕ РАБОТЫ

1. Сформировать группу из 5-6 человек и распределить обязанности.
2. Разработать ПП в соответствии с выбранной технологией.
3. Оформить отчет по всем этапам разработки программного продукта.

В полной мере объем работы можно проследить по структуре отчета по практике (см. далее п.8).

5.ЗАДАНИЯ УЧЕБНОЙ ПРАКТИКИ

ВАРИАНТЫ ЗАДАНИЙ*:*

1. Составление расписания занятий в учебном заведении.
2. Автоматизировать работу риэлтерской фирмы.
3. Автоматизация учета лекарственных средств в аптеке.
4. Компьютеризировать регистрацию пассажиров в аэропорту.
5. Автоматизация учета движения документов в фирме.
6. Автоматизировать работу администратора гостиницы.
7. Автоматизировать работу регистратуры поликлиники.
8. Автоматизировать работу диспетчера по управлению городским транспортом.
9. Автоматизировать работу по приему абитуриентов в учебное заведение.
10. Автоматизировать работу по обслуживанию пассажиров водным транспортом.
11. Библиотека - работа с абонентами, книжный фонд.
12. Приемная комиссия колледжа.
13. Склад готовой продукции. Учет реализации, поставок. Работа с поставщиками.
14. Учет контингента студентов колледжа.
15. ***Оптимизация грузоперевозок транспортной фирмы.

16. ***Реализация железнодорожных билетов с учетом транзитных станций.
17. ***Игра в шахматы. Самоучитель.
18. Работа туристического бюро. Реализация путевок. Анализ продаж.
19. Магазин сотовой связи.
20. Коммунальные платежи (на примере одного дома).
21. Темы, предложенные студентами

6.ОРИЕНТИРОВОЧНЫЙ ГРАФИК

Исследование предметной области 10 ч
Проектирование 16 ч
Написание функций на языке программирования 16ч
Сбор функций в единую программу 4 ч.
Тестирование 6 ч.
Исправление ошибок и неточностей 4 ч.
Расчет характеристик программного продукта 4 ч
Оформление отчета о проделанной работе 6 ч.
Защита 6 ч.

Примечание! График работ полностью зависит от выбранной технологии и модели разработки ПП.

7.ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ

7.1.БЫСТРАЯ РАЗРАБОТКА ПРИЛОЖЕНИЙ RAD

Модель RAD обеспечивает экстремально короткие сроки проектирования за счет параллельной реализации бизнес-функций, а также за счет компонентно-ориентированного конструирования.
RAD основано на следующих этапах:

1. Бизнес-моделирование – моделируется информационный поток между бизнес–файлами. Ставятся вопросы:
   1) Какая информация руководит процессом?
   2) Какая информация генерируется?
   3) Кто ее генерирует?
   4) Где информация применяется?
   5) Кто обрабатывает ее?
2. Моделирование данных - информационный поток, определенный на 1 –м этапе, отображается набор объектов данных. Определяются свойства и атрибуты каждого объекта и отношения между объектами.
3. Моделирование обработки – определяются преобразования объектов данных, которые обеспечивают основные бизнес-функции.
4. Генерация приложений – используются методы, ориентированные на языки 4-го поколения.
5. Тестирование и объединение в одну систему.

Для каждой бизнес функции создается своя группа разработчиков. На заключительном этапе результаты работы всех групп объединяются в один проект.
Ограничения и недостатки RAD:

1) Для больших проектов требуется большое количество групп, а, следовательно, большое количество людских ресурсов.
2) RAD используется только в системах с декомпозицией на модули, причем производительность их не является критической величиной.
3) RAD не используется в условиях высоких технических рисков, то есть с использованием новых технологий.

7.2. XP – ПРОЦЕСС

XP – процесс относится к адаптивным процессам, использует группы до 10 разработчиков ***равной, высокой квалификации*** (в условиях учебной практики это требование является определяющим при выборе модели конструирования). Используется в условиях часто меняющихся требований. Все разработчики находятся в одном помещении, здесь же представитель заказчика. К базовым действиям относятся: кодирование, тестирование, выслушивание заказчика.
Основные характеристики, обеспечивающие динамизм процесса:

1) непрерывная связь с заказчиком;
2) выбор самого простого решения;
3) быстрая обратная связь;
4) смелость в профилактике сложных проблем.

XP – процесс базируется на 12 методах:

1. Игра планирования – быстрое определение области действия будущего ПО путем объединения деловых приоритетов и технических оценок.
2. Частая смена версий – быстрый запуск в производство простой системы, очень короткие итерации.
3. Метафора – вся разработка проводится на основе простой общедоступной истории о том, как работает вся система, то есть обеспечивается глобальное видение системы.
4. Простое проектирование – настолько просто, на сколько это возможно.
5. Тестирование – непрерывное тестирование модулей.
6. Реорганизация – система реструктурируется с целью устранения дублирования, упрощения и др.
7. Парное программирование – код пишется двумя программистами, работающими на одном компьютере. Это повышает качество.
8. Коллективное владение кодом – любой разработчик может улучшать любой код системы в любое время.
9. Непрерывная интеграция – система интегрируется и строится по много раз в день по мере завершения каждой программы.
10. Отсутствие сверхурочных работ.
11. Вокальный заказчик в группе – все время представитель заказчика, который отвечает на все вопросы.
12. Стандарты кодирования – должны соблюдаться определенные правила для всех.

1.3 СОСТАВЛЕНИЕ ДИАГРАММ USE CASE

Диаграмма Use Case определяет поведение системы с точки зрения пользователя. Диаграмма Use Case рассматривается как главное средство для первичного моделирования динамики системы, используется для выяснения требований к разрабатываемой системе, фиксации этих требований в форме, которая позволит проводить дальнейшую разработку.

Вариант использования представляет собой последовательность действий (транзакций), выполняемых системой в ответ на событие, инициируемое некоторым внешним объектом (действующим лицом). В простейшем случае вариант использования определяется в процессе обсуждения с пользователем тех функций, которые он хотел бы реализовать.

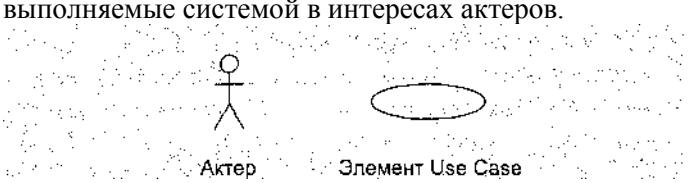
В состав диаграмм Use Case прежде всего входят:

- элементы Use Case ,
- актеры,
- отношения - зависимости, обобщения и ассоциации.

Как и другие диаграммы, диаграммы Use Case могут включать примечания и ограничения. Кроме того, диаграммы Use Case могут содержать пакеты, используемые для группировки элементов модели в крупные фрагменты.

Актеры и элементы Use Case

Вершинами в диаграмме Use Case являются *актеры и элементы Use Case* . Актеры представляют внешний мир, нуждающийся в работе системы. Элементы Use Case представляют действия, выполняемые системой в интересах актеров.



Актер — это роль объекта вне системы, который прямо взаимодействует с ее частью — конкретным элементом (элементом Use Case). Различают актеров и пользователей. Пользователь — это физический объект, который использует систему. Он может играть несколько ролей и поэтому может моделироваться несколькими актерами. Справедливо и обратное — актером могут быть разные пользователи.

Элемент Use Case — это описание последовательности действий (или нескольких последовательностей), которые выполняются системой и производят для отдельного актера видимый результат.

Один актер может использовать несколько элементов Use Case и наоборот, один элемент Use Case может иметь несколько актеров, использующих его. Каждый элемент Use Case задает определенный путь использования системы. Набор всех элементов Use Case определяет полные функциональные возможности системы.

Варианты использования являются необходимым средством на стадии формирования требований к ПО. *Каждый вариант использования — это потенциальное требование к системе, и пока оно не выявлено, невозможно запланировать его реализацию.*

Хорошим источником для идентификации вариантов использования служат внешние события. Следует начать с перечисления всех событий, происходящих во внешнем мире, на которые система должна каким-то образом реагировать. Какое-либо конкретное событие может повлечь за собой реакцию системы, не требующую вмешательства пользователей, или, наоборот, вызвать чисто пользовательскую реакцию. Идентификация событий, на которые необходимо реагироватъ, помогает выделить варианты использования.

Различные разработчики подходят к описанию вариантов использования с разной степенью детализации. Следует предпочитать небольшие и детализированные варианты использования, поскольку они облегчают составление и реализацию согласованного плана проекта.

Отношения в диаграммах Use Case

Между актером и элементом Use Case возможен только один вид отношения — *ассоциация*, отображающая их взаимодействие . Как и любая другая ассоциация, она может быть помечена именем, ролями, мощностью.

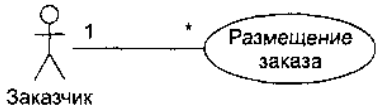


Рис. 12.28. Отношение ассоциации

Между актерами допустимо отношение *обобщения*, означающее, что экземпляр потомка может взаимодействовать с такими же разновидностями экземпляров элементов Use Case , что и экземпляр родителя.

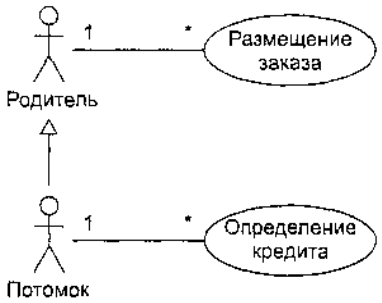


Рис. 12.29. Отношение обобщения между актерами

Между элементами Use Case определены отношение *обобщения* и две разновидности отношения *зависимости — включения и расширения*.

Отношение обобщения фиксирует, что потомок наследует поведение родителя. Кроме того, потомок может дополнить или переопределить поведение родителя. Элемент Use Case , являющийся потомком, может замещать элемент Use Case , являющийся родителем, в любом месте диаграммы.

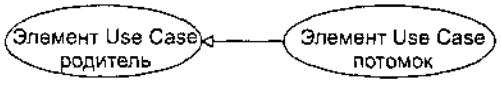


Рис. 12.30. Отношение обобщения между элементами Use Case

Отношение *включения* между элементами Use Case означает, что базовый элемент Use Case явно_ включает поведение другого элемента Use Case в точке, которая определена в базе. Включаемый элемент Use Case никогда не используется самостоятельно — его конкретизация может быть только частью другого, большего элемента Use Case. Отношение включения является примером отношения делегации. При этом в отдельное место (включаемый элемент Use Case) помещается определенный набор обязанностей системы. Далее остальные части системы могут агрегировать в себя эти обязанности (при необходимости).

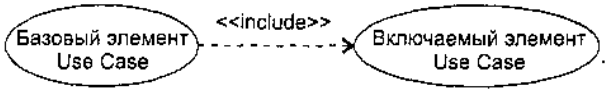


Рис. 12.31. Отношение включения между элементами Use Case

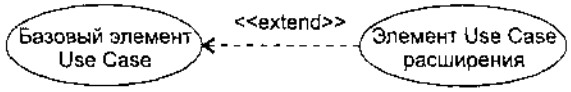


Рис. 12.32. Отношение расширения между элементами Use Case

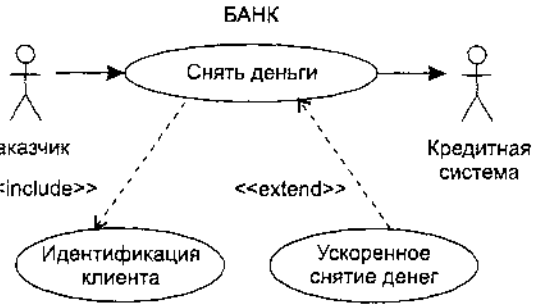


Рис. 12.33. Простейшая диаграмма Use Case для банка

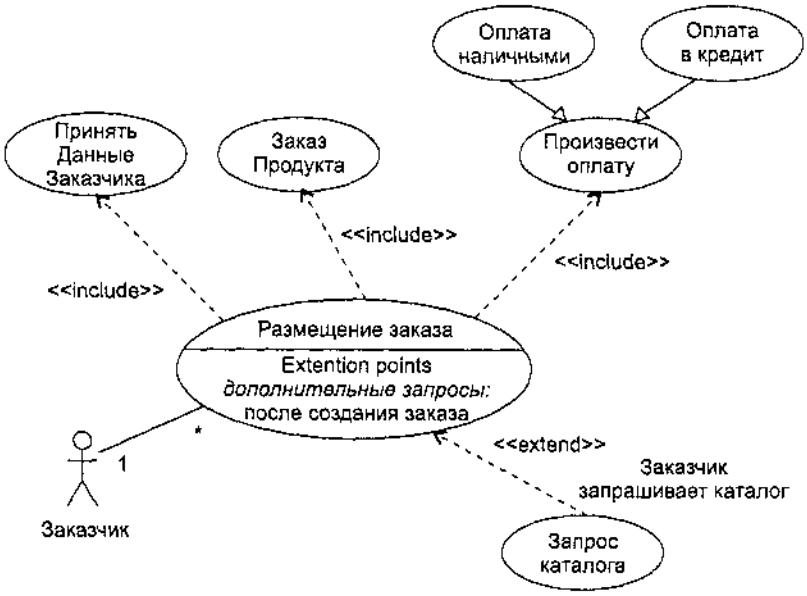


Рис. 12.34. Диаграмма Use Case для обслуживания заказчика

Отношение *расширения* между элементами Use Case означает, что базовый элемент Use Case неявно включает поведение другого элемента Use Case в точке, которая определяется косвенно расширяющим элементом Use Case. Базовый элемент Use Case может быть автономен, но при определенных условиях его поведение может расширяться поведением из другого элемента Use Case. Базовый элемент Use Case может расширяться только в определенных точках — точках расширения. Отношение расширения применяется для моделирования выбираемого поведения системы. Таким способом можно отделить обязательное поведение от необязательного поведения. Например, можно использовать отношение расширения для отдельного подпотока, который выполняется только при определенных условиях, находящихся вне поля зрения базового элемента Use Case . Наконец, можно моделировать отдельные потоки, вставка которых в определенную точку управляется актером.

Пример 1 простейшей диаграммы Use Case , в которой использованы отношения включения и расширения, приведен на рис. 12.33.

Как показано на рис. 12.34, внутри элемента Use Case может быть дополнительная секция с заголовком Extension points. В этой области перечисляются точки расширения. В указанную здесь точку дополнительные запросы вставляется последовательность действий от расширяющего элемента Use Case «Запрос каталога». Для справки отмечено, что точка расширения размещена после действий, обеспечивающих создание заказа. На этом же рисунке отображены отношения наследования между

элементами Use Case . Видно, что элементы Use Case «Оплата наличными2 и «Оплата в кредит» наследуют поведение элемента Use Case «Произвести оплату» и являются его специализациями.

7.4 ДИАГРАММЫ ПРИЧИННО-СЛЕДСТВЕНЫХ СВЯЗЕЙ

Тестирование — выполнение программы с целью обнаружения ошибок. Тесты демонстрируют, как выполняются функции, как принимаются исходные данные, как вырабатываются результаты. Каждый тест определяет:

1 набор входных данных и условий для запуска программы
2 набор ожидаемых результатов

Существуют 2 принципа тестирования программных продуктов: функциональное и структурное. *Структурное тестирование* - известна структура программы, исследуются внутренние элементы структуры и связи между ними, проверяются все независимые маршруты, ветви и выполнение всех циклов. *Функциональное тестирование* - известны функции программы, исследуется работа каждой функции на всей области определения.

Диаграммы причинно-следственных связей относятся к функциональному тестированию и определяют способ проектирования тестовых вариантов, который обеспечивает формальную запись логических условий и соответствующих действий . Используется автоматный подход к решению задачи.

Шаги способа:

1) для каждого модуля перечисляются причины (условия ввода или классы эквивалентности условий ввода) и следствия (действия или условия ввода). Каждой причине и следствию присваивается свой идентификатор;
2) разрабатывается граф причинно-следственных связей4
3) граф преобразуется в таблицу решений;
4) столбцы таблицы решений преобразуются в тестовые варианты.

Изобразим базовые символы для записи графов причин и следствий (cause-effect graphs).

Обозначения. Сделаем предварительные *замечания*:

1) причины будем обозначать символами c_i , а следствия – символами e_i;
2) каждый узел графа может находиться в состоянии 0 или 1 (0 – состояние отсутствует, 1 – состояние присутствует).

Функция тождество (рис. 1) устанавливает, что если значение c_i есть 1, то и значение e_i есть 1; в противном случае значение e_i есть 0.

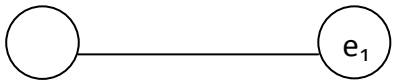


Рис. 1**.** Функция *тождество*

Функция *не* (рис. 2) устанавливает, что если значение c_1 есть 1, то значение e_1 есть 0, в противном случае значение e_1 есть 1.

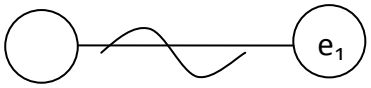


Рис. 2**.** Функция *не*

Функция *или* (рис. 3) устанавливает, что если c_1 или c_2 есть 1, то e_1 есть 1, в противном случае e_1 есть 0.

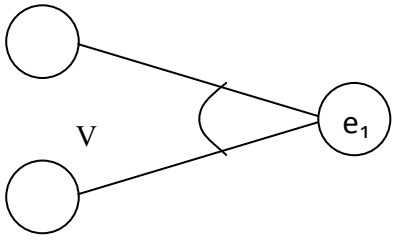


Рис. 3. Функция *или*

Функция *и* (рис.4) устанавливает, что если c_1 и c_2 есть 1, то e_1 есть 1, в противном случае e_1 есть 0.

Часто определённые комбинации причин невозможны из-за синтаксических или внешних ограничений. Используются перечисленные ниже обозначения ограничений.

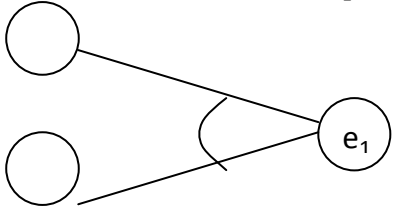


Рис. 4**.** Функция *и*

Ограничение E (*исключает*, Exclusive, рис. 5) устанавливает, что E должно быть истинным, если хотя бы одна из причин – *a* или *b* – принимает значение 1 (*a* и *b* не могут принимать значение 1 одновременно).

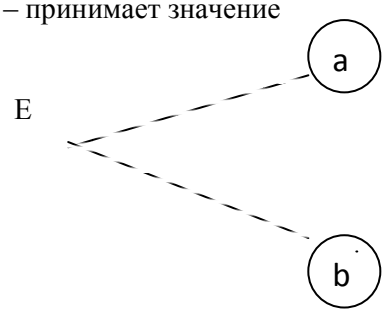


Рис. 5**.** Ограничение E (исключает, Exclusive)

Ограничение I (*включает*, Inclusive, рис. 6) устанавливает, что по крайней мере одна из величин *a*, *b* или *c*, всегда должна быть равной 1(*a*, *b* и *c* не могут принимать значение 0 одновременно).

I

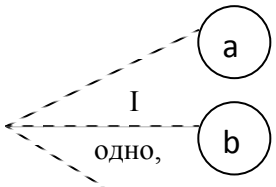


Рис. 6**.** Ограничение I (включает, Inclusive)

Ограничение O (одно и только одно, Only one, рис. 7) устанавливает, что одна и только одна из величин *a* или *b* должна быть равна 1.

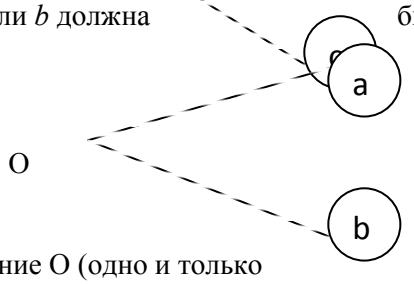


Рис. 7. Ограничение O (одно и только одно, Only one)

Ограничение R (требует, Requires, рис. 8) устанавливает, что если *a* принимает значение 1, то и *b* должна принимать значение 1 (нельзя, чтобы *a* было равно 1, а б – 0).

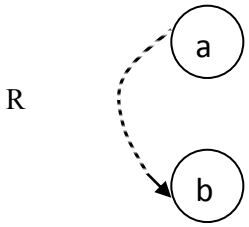


Рис. 8. Ограничение R (требует, Requires)

Часто возникает необходимость в ограничениях для следствий.

Ограничение M (скрывает, Masks, рис. 9) устанавливает, что если следствие a имеет значение 1, то следствие *b* должно принять значение 0.

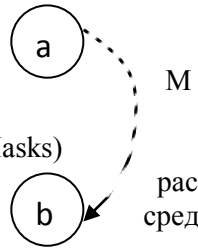


Рис. 9**.** Ограничение M (скрывает, Masks)

Для иллюстрации использования способа рассмотрим пример, когда программа выполняет расчёт оплаты за электричество по среднему или переменному тарифу.

При расчёте по среднему тарифу:

- при месячном потреблении энергии меньшем, чем 100 кВт/ч, выставляется фиксированная сумма;
- при потреблении энергии большем или равном 100 кВт/ч применяется процедура А планирования расчёта;

При расчёте по переменному тарифу:

- при месячном потреблении энергии меньшем, чем 100 кВт/ч, применяется процедура А планирования расчёта;
- при потреблении энергии большем или равном 100 кВт/ч применяется процедура В планирования расчёта;

Шаг 1. *Причинами* являются:

1) расчёт по среднему тарифу;
2) расчёт по переменному тарифу;
3) месячное потребление электроэнергии меньшее, чем 100 кВт/ч;
4) месячное потребление электроэнергии большее или равное 100 кВт/ч.

На основании различных комбинаций причин можно перечислить следующие *следствия*:

- 101 – минимальная месячная стоимость;
- 102 – процедура А планирования расчёта;
- 103 – процедура В планирования расчёта;

Шаг 2. Разработка графа причинно-следственных связей (рис.13).

Узлы причин перечислим по вертикали у левого края рисунка, а узлы следствий – у правого края рисунка. Для следствия 102 возникает необходимость введения вторичных причин – 11 и 12, – их размещаем в центральной части рисунка.

Шаг 3. Генерация таблицы решений (табл.1). При генерации причины рассматриваются как условия, а следствия – как действия.

1. Выбирается некоторое следствие, которое должно быть в состоянии «1».
2. Находятся все комбинации причин (с учётом ограничений), которые устанавливают это следствие в состояние «1». Для этого из следствия прокладывается обратная трасса через граф.
3. Для каждой комбинации причин, приводящих следствие в состояние «1»,

строится один столбец.

4. Для каждой комбинации причин доопределяются состояния всех других следствий. Они помещаются в тот же столбец таблицы решений.
5. Действия 1-4 повторяются для всех следствий графа.

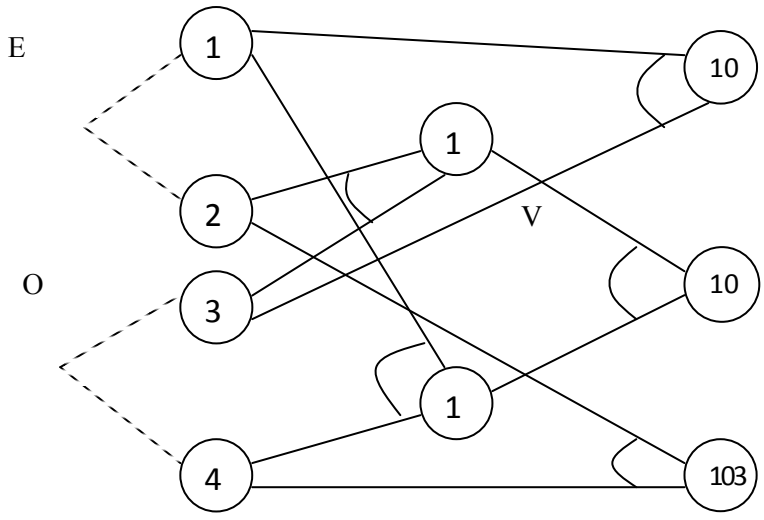


Рис.10. Граф причинно-следственных связей

Шаг 4. Преобразование каждого столбца таблицы в тестовый вариант. В нашем примере таких вариантов четыре.

Тестовый вариант 1 (столбец 1) **ТВ1:**

ИД: расчёт по среднему тарифу; месячное потребление электроэнергии 75 кВт/ч.

ОЖ.РЕЗ.: минимальная месячная стоимость.

Тестовый вариант 2 (столбец 2) **ТВ2:**

ИД: расчёт по переменному тарифу; месячное потребление электроэнергии 90 кВт/ч.

ОЖ.РЕЗ.: процедура А планирования расчёта.

Тестовый вариант 3 (столбец 3) **ТВ3:**

ИД: расчёт по среднему тарифу; месячное потребление электроэнергии 100 кВт/ч.

ОЖ.РЕЗ.: процедура В планирования расчёта.

Тестовый вариант 4 (столбец 4) **ТВ4:**

ИД: расчёт по переменному тарифу; месячное потребление электроэнергии 100 кВт/ч.

ОЖ.РЕЗ.: процедура В планирования расчёта.

Табл. 1. Таблица решений для расчёта оплаты за электричество

Номера столбцов→			1	2	3	4
Условия	Причины	1	1	0	1	0
		2	0	1	0	1
		3	1	1	0	0
		4	0	0	1	1
	Вторичные причины	11	0	0	1	0
		12	0	1	0	0
Действия	Следствия	101	1	0	0	0
		102	0	1	1	0
		103	0	0	0	1

7.5 ФУНКЦИОНАЛЬНО-ОРИЕНТИРОВАННЫЕ МЕТРИКИ

Функционально-ориентированные метрики косвенно измеряют программный продукт и процесс его разработки. Используется функциональность или полезность. Используется 5 информационных характеристик:

1) Количество внешних вводов – подсчитываются все вводы поля, по которым поступают различные прикладные данные.
2) Количество внешних выводов – отчеты, экраны, распечатки, сообщения об ошибках.
3) Количество внешних запросов. Запрос – это диалоговый ввод, который приводит к немедленному запросу.
4) Количество внутренних логических файлов, которые могут быть частью базы данных или отдельными файлами.
5) Количество внешних интерфейсных файлов, то есть логических файлов из других приложений, на которые ссылаются данные приложения.

Тем самым определяются функциональные показатели или точки, то есть 5 видов (FP – Function Points)
Для задач инженерных, научных, системных, реального времени используется другое понятие функциональности – количество и сложность алгоритма. В общем виде количество функциональных точек FP = общее количество * (0.65 + 0.01 * Σ Fi),
где Fi – коэффициент регулировки сложности.

Существует специальная таблица из 14 коэффициентов, каждый из которых имеет значение, отражающее его влияние на программный продукт.

На основании данного FP рассчитываются 4 характеристики:

$$\text{Производительность} = \frac{FP}{\text{Затраты}}$$

$$\text{Качество} = \frac{\text{ошибки}}{FP}$$

$$\text{Удельная стоимость} = \frac{\text{стоимость}}{FP}$$

Количество страниц документа

$$\text{Документированность} = \frac{14}{FP}$$

Для оценки программного продукта (размера программного продукта, трудоемкости, продолжительности, стоимости) используются 2 различных подхода:

1) по аналогии с предыдущем;
2) с использованием различных моделей, в частности, широко используется модель COCOMO.

8.ОФОРМЛЕНИЕ ОТЧЕТА ПО ПРАКТИКЕ

Отчет по практике (пример титульного листа приведен в приложении) должен иметь следующую структуру:

Введение
1. Общая часть
   1.1. Постановка задачи
   1.2. Описание модели конструирования
   1.3. Состав группы разработчиков
   1.4. Распределение ролей и работ в группе разработчиков
   1.5. Инструментальные средства разработки
2. Специальная часть
   2.1. Анализ требований к ПО. Диаграмма Use Case.
   2.2. Кодирование. Структура ПО.
   2.3. Тестирование, отладка
      2.3.1.Планирование тестовых вариантов по методу диаграмм причин-следствий.
      2.3.2.Анализ ошибок. (Таблица 2)
3. Оценка проекта. Метрики – расчет FP, производительности, качества (таблицы 3, 4, 5)
4. Выводы. Достоинства и недостатки проекта.
5. Приложения
   5.1. Исходные тексты программных модулей.
   5.2. Дневник процесса конструирования (Таблица 1).

Все необходимые расчеты необходимо реализовать с помощью следующих таблиц:

Таблица 1 - Дневник конструирования

Дата	Кол-во чел-часов	Этап конструирования	Работа	Ошибки	
				Наименов	Время на исправлен

Таблица 2 - Анализ ошибок

№ п/п	Этап конструирования (системный анализ, анализ требований, кодирование, тестирование, отладка)	Характер ошибки	Время на исправление (час)

Таблица 3. - Расчет количества информационных характеристик

№ п/п	Модуль	Информационные характеристики	

	ПО	Внешн. вводы	Внешн. выводы	Запросы	Файлы БД	Интерф. файлы	Кол-во методов

Таблица 4. Исходные данные для расчета указателя свойств

№п/п	Характеристика	Количество	Сложность	Итого
1	Вводы	□	x 4	=□
2	Выводы	□	x 5	=□
3	Запросы	□	x 4	=□
4	Логические файлы	□	x 7	=□
5	Интерфейсные файлы	□	x 7	=□
6	Количество методов	□	x 3	=□
Общее количество				=□

В таблице прямоугольниками отмечены места подстановки значений количественных характеристик каждого вида по уровням сложности. Полученные в каждой строке значения суммируются, давая полное значение для данной информационной характеристики. Это полные значения затем суммируются по вертикали, формируя ***общее количество***.

Затем расчет количества FP осуществляется по формуле:

FP = *Общее количество* x (0,65 + 0,01 x ΣFi) ,

где Fi – коэффициенты регулировки сложности, характеризующие системные параметры приложения. Значения выбираются эмпирически в результате ответа на 14 вопросов, которые характеризуют системные параметры приложения. Каждый коэффициент может принимать следующие значения : 0 – нет влияния, 1 – случайное, 2 – небольшое, 3 – среднее, 4 – важное, 5 – основное (таблица 5).

Таблица 5 - Определение системных параметров приложения

№ п/п	Системный параметр приложения	Коэффициенты влияния					
		0	1	2	3	4	5
1	Передачи данных						
2	Распред.обр.данных						
3	Производительность						
4	Распростран.конфигурации						
5	Скорость транзакций						
6	Оперативный ввод данных						
7	Эфф. работы пользователя						
8	Оперативное обновление						
9	Сложность обработки						
10	Повторная используемость						
11	Легкость инсталляции						
12	Легкость эксплуатации						
13	Разн. условия размещения						
14	Простота изменений						
	итого	Σ					

9. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Достоинства модели RAD.
2. Почему компонентный подход к созданию программ считается очень перспективным?
3. На каких этапах основано RAD?
4. Ограничения и недостатки RAD.
5. Автор и время создания XP – процесса.
6. На каких методах базируется XP – процесс?
7. Самые спорные методы XP-процесса и почему?
8. Какие информационные характеристики используют функционально – ориентированные метрики?
9. В чем особенности расчета метрик для задач высокой алгоритмической сложности?
10. На каком этапе жизненного цикла используются диаграммы Use Case?
11. В чем принципиальная разница между функционально-ориентированными метриками и метриками объектно-ориентированных систем?

10. ПЕРЕЧЕНЬ ЛИТЕРАТУРЫ И СРЕДСТВ ОБУЧЕНИЯ

Федорова, Г. Н.	Разработка модулей программного обеспечения для компьютерных систем: учебник для студентов учреждений сред. проф. образования	М.: Академия, 2018
Федорова, Г. Н.	Осуществление интеграции программных модулей: учебник для студентов учреждений сред. проф. образования	М.: Академия, 2018

1.

СРЕДСТВА ОБУЧЕНИЯ

1. Персональный компьютер класса Pentium
2. Описание языка UML
3. Инструментальные средства разработки объектно-ориентированных систем
4. Средства CASE-технологий (по выбору).
5. УМК по дисциплине
6. Электронный учебник (лекции по дисциплине)
7. Сборник практических работ по дисциплине
8. Методические рекомендации к проведению учебной практики

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Авиационно-технологический колледж ДГТУ

ОТЧЕТ ПО УЧЕБНОЙ ПРАТИКЕ УП.02

по профессиональному модулю ПМ.02. «Осуществление интеграции программных модулей»

Специальности 09.02.07 Информационные системы и программирование

Тема _____
_____
_____

Выполнили _____
_____
_____
_____
_____
_____

Проверил _____/_ _____

Оценка _____

Ростов-на-Дону, 20 г.