ГОУВПО

ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ И ЗАДАНИЯ

к лабораторным работам по дисциплине

«Конструирование программного обеспечения»

(для студентов направления подготовки 09.03.04 “Программная инженерия”)

ДОНЕЦК-ДОННТУ-2016

ГОУВПО

ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ И ЗАДАНИЯ
к лабораторным работам по дисциплине

«Конструирование программного обеспечения»

(для студентов направления подготовки 09.03.04 “Программная инженерия”)

Рассмотрено на заседании кафедры

программной инженерии

Протокол № 1 от 30.08.2016

Утверждено на заседании

учебно-издательского Совета ДонНТУ

протокол № от

ДОНЕЦК –2016

УДК 681.3

Методические указания и задания к лабораторным работам по дисциплине «Конструирование программного обеспечения» для студентов направления подготовки 09.03.04 «Программная инженерия», Сост.: Чернышова А.В., Донецк, ДонНТУ, 2016 - 127 стр.

Приведены методические указания и задания к выполнению лабораторных работ по дисциплине «Конструирование программного обеспечения» для студентов направления подготовки 09.03.04 «Программная инженерия». Излагаются вопросы, связанные этапами разработки программного обеспечения, таких как составление технического задания к ПО, составление эскизного проекта ПО, оценка LOC и FP метрик программных проектов, анализ чувствительности программного проекта на основе модели COCOMO II, метрики объектно-ориентированных программных систем, организация таблиц в трансляторах и работа с ними.

Методические указания предназначены для усвоения теоретических основ и формирования практических навыков по курсу «Конструирование программного обеспечения».

Составители: ст.преп. каф. ПИ Чернышова А.В.

Лабораторная работа № 1

Тема: Этапы разработки программного обеспечения при структурном подходе к программированию. Стадия ее Техническое задание».

Цель работы: ознакомиться с правилами написания технического задания.

Методические указания к лабораторной работе

Техническое задание представляет собой документ, в котором сформулированы основные цели разработки, требования к программному продукту, определены сроки и этапы разработки и регламентирован процесс приемо-сдаточных испытаний. В разработке технического задания участвуют как представители заказчика, так и представители исполнителя. В основе этого документа лежат исходные требования заказчика, анализ передовых достижений техники, результаты выполнения научно-исследовательских работ, предпроектных исследований, научного прогнозирования и т. п.

Порядок разработки технического задания

Разработка технического задания выполняется в следующей последовательности. Прежде всего, устанавливают набор выполняемых функций, а также перечень и характеристики исходных данных. Затем определяют перечень результатов, их характеристики и способы представления. Далее уточняют среду функционирования программного обеспечения: конкретную комплектацию и параметры технических средств, версию используемой операционной системы и, возможно, версии и параметры другого установленного программного обеспечения, с которым предстоит взаимодействовать будущему программному продукту.

В случаях, когда разрабатываемое программное обеспечение собирает и хранит некоторую информацию или включается в управление каким-либо

техническим процессом, необходимо также четко регламентировать действия программы в случае сбоев оборудования и энергоснабжения.

1. Общие положения

1.1. Техническое задание оформляют в соответствии с ГОСТ 19.106—78 на листах формата А4 и А3 по ГОСТ 2.301—68, как правило, без заполнения полей листа. Номера листов (страниц) проставляют в верхней части листа над текстом.

1.2. Лист утверждения и титульный лист оформляют в соответствии с ГОСТ 19.104—78. Информационную часть (аннотацию и содержание), лист регистрации изменений допускается в документ не включать.

1.3. Для внесения изменений и дополнений в техническое задние на последующих стадиях разработки программы или программного изделия выпускают дополнение к нему. Согласование и утверждение дополнения к техническому заданию проводят в том же порядке, который установлен для технического задания.

1.4. Техническое задание должно содержать следующие разделы:

• введение;

• наименование и область применения;

• основание для разработки;

• назначение разработки;

• технические требования к программе или программному изделию;

• технико-экономические показатели;

• стадии и этапы разработки;

• порядок контроля и приемки;

• приложения.

В зависимости от особенностей программы или программного изделия допускается уточнять содержание разделов, вводить новые разделы или объединять отдельные из них. При необходимости допускается в техническое задание включать приложения.

2. Содержание разделов

2.1. Введение должно включать краткую характеристику области применения программы или программного продукта, а также объекта (например, системы), в котором предполагается их использовать. Основное назначение введения — продемонстрировать актуальность данной разработки и показать, какое место эта разработка занимает в ряду подобных.

2.2. В разделе «Наименование и область применения» указывают наименование, краткую характеристику области применения программы или программного изделия и объекта, в котором используют программу или программное изделие.

2.3. В разделе «Основание для разработки» должны быть указаны:

• документ (документы), на основании которых ведется разработка. Таким документом может служить план, приказ, договор и т. п.

• организация, утвердившая этот документ, и дата его утверждения;

• наименование и (или) условное обозначение темы разработки.

2.4. В разделе «Назначение разработки» должно быть указано функциональное и эксплуатационное назначение программы или программного изделия.

2.5. Раздел «Технические требования к программе или программному изделию» должен содержать следующие подразделы:

• требования к функциональным характеристикам;

• требования к надежности;

• условия эксплуатации;

• требования к составу и параметрам технических средств;

• требования к информационной и программной совместимости;

• требования к маркировке и упаковке;

• требования к транспортированию и хранению;

• специальные требования.

2.5.1. В подразделе «Требования к функциональным характеристикам» должны быть указаны требования к составу выполняемых функций,

организации входных и выходных данных, временным характеристикам и т. п.

2.5.2. В подразделе «Требования к надежности» должны быть указаны требования к обеспечению надежного функционирования (обеспечение устойчивого функционирования, контроль входной и выходной информации, время восстановления после отказа и т. п.).

2.5.3. В подразделе «Условия эксплуатации» должны быть указаны условия эксплуатации (температура окружающего воздуха, относительная влажность и т. п. для выбранных типов носителей данных), при которых должны обеспечиваться заданные характеристики, а также вид обслуживания, необходимое количество и квалификация персонала.

2.5.4. В подразделе «Требования к составу и параметрам технических средств» указывают необходимый состав технических средств с указанием их технических характеристик.

2.5.5. В подразделе «Требования к информационной и программной совместимости» должны быть указаны требования к информационным структурам на входе и выходе и методам решения, исходным кодам, языкам программирования. При необходимости должна обеспечиваться защита информации и программ.

2.5.6. В подразделе «Требования к маркировке и упаковке» в общем случае указывают требования к маркировке программного изделия, варианты и способы упаковки.

2.5.7. В подразделе «Требования к транспортированию и хранению» должны быть указаны для программного изделия условия транспортирования, места хранения, условия хранения, условия складирования, сроки хранения в различных условиях.

2.5.8. В разделе «Технико-экономические показатели» должны быть указаны: ориентировочная экономическая эффективность, предполагаемая годовая потребность, экономические преимущества разработки по

сравнению с лучшими отечественными и зарубежными образцами или аналогами.

2.6. В разделе «Стадии и этапы разработки» устанавливают необходимые стадии разработки, этапы и содержание работ (перечень программных документов, которые должны быть разработаны, согласованы и утверждены), а также, как правило, сроки разработки и определяют исполнителей.

2.7. В разделе «Порядок контроля и приемки» должны быть указаны виды испытаний и общие требования к приемке работы.

2.8. В приложениях к техническому заданию при необходимости приводят:

• перечень научно-исследовательских и других работ, обосновывающих разработку;

• схемы алгоритмов, таблицы, описания, обоснования, расчеты и другие документы, которые могут быть использованы при разработке;

• другие источники разработки.

В случаях, если какие-либо требования, предусмотренные техническим заданием, заказчик не предъявляет, следует в соответствующем месте указать «Требования не предъявляются».

Пример 1.1. Разработать техническое задание на программный продукт, предназначенный для наглядной демонстрации школьникам графиков функций одного аргумента y =/(x).

Разрабатываемая программа должна рассчитывать таблицу значений и строить график функций на заданном отрезке по заданной формуле и менять шаг аргумента и границы отрезка. Кроме этого, программа должна запоминать введенные формулы.

Техническое задание к данному примеру смотри в приложении 2 [2].

Пример 1.2. Разработать техническое задание на разработку «Модуля автоматизированной системы оперативно-диспетчерского управления

теплоснабжением корпусов Московского института».. Техническое задание к данному примеру смотри в приложении 3 [2].

Задание к лабораторной работе

1. Разработать техническое задание на программный продукт (программный продукт выбрать самостоятельно по желанию).

2. Оформить работу в соответствии с ГОСТ 19.106—78.

3. Сдать и защитить работу.

Требование к отчету по лабораторной работе

Отчет по лабораторной работе должен содержать:

1. Постановки задачи.

2. Технического задания на программный продукт.

Защита отчета по лабораторной работе заключается в предъявлении преподавателю полученных результатов, ответы на вопросы преподавателя.

Контрольные вопросы

1. Приведите этапы разработки программного обеспечения.
2. Что включает в себя постановка задачи и предпроектные исследования?
3. Перечислите функциональные и эксплуатационные требования к программному продукту.
4. Перечислите правила разработки технического задания.
5. Назовите основные разделы технического задания.

Лабораторная работа № 2

Тема: Структурный подход к программированию. Стадия «Эскизный проект». Сетевой график выполнения работ.

Цель работы: научиться создавать формальные модели и на их основе определять спецификации разрабатываемого программного обеспечения. Научиться использовать сетевой график выполнения работ для оценки времени выполнения проекта.

Методические указания к лабораторной работе

1. Ознакомиться с лекционным материалом по теме «Этапы разработки программного обеспечения. Анализ требований и определение спецификаций программного обеспечения»

2. Ознакомиться с разд. 3.5 см. [2].

Теоретическая часть. Разработка спецификаций

Разработка программного обеспечения начинается с анализа требований к нему. В результате анализа получают спецификации разрабатываемого программного обеспечения, строят общую модель его взаимодействия с пользователем или другими программами и конкретизируют его основные функции.

При структурном подходе к программированию на этапе анализа и определения спецификаций разрабатывают три типа моделей: модели функций, модели данных и модели потоков данных. Поскольку разные модели описывают проектируемое программное обеспечение с разных сторон, рекомендуется использовать сразу несколько моделей,

разрабатываемых в виде диаграмм, и пояснять их текстовыми описаниями, словарями и т. п.

Структурный анализ предполагает использование следующих видов моделей:

• диаграмм потоков данных (DFD — Data Flow Diagrams), описывающих взаимодействие источников и потребителей информации через процессы, которые должны быть реализованы в системе;

• диаграмм «сущность—связь» (ERD — Entity-Relationship Diagrams), описывающих базы данных разрабатываемой системы;

• диаграмм переходов состояний (STD — State Transition Diagrams), характеризующих поведение системы во времени;

• функциональных диаграмм (методика SADT);

• спецификаций процессов;

• словаря терминов.

Спецификации процессов

Спецификации процессов обычно представляют в виде краткого текстового описания, схем алгоритмов, псевдокодов, Flow-форм или диаграмм Насси — Шнейдермана (см. разд. 3.5.1 [2]).

Словарь терминов

Словарь терминов представляет собой краткое описание основных понятий, используемых при составлении спецификаций. Он должен включать определение основных понятий предметной области, описание структур элементов данных, их типов и форматов, а также всех сокращений и условных обозначений (см. разд. 3.5.2 [2]).

Диаграммы переходов состояний

С помощью диаграмм переходов состояний можно моделировать последующее функционирование системы на основе ее предыдущего и текущего функционирования. Моделируемая система в любой заданный момент времени находится точно в одном из конечного множества состояний. С течением времени она может изменить свое состояние, при

этом переходы между состояниями должны быть точно определены (см. разд. 3.5.3 [2]).

Функциональные диаграммы

Функциональные диаграммы отражают взаимосвязи функций разрабатываемого программного обеспечения. Они создаются на ранних этапах проектирования систем, для того чтобы помочь проектировщику выявить основные функции и составные части проектируемой системы и, по возможности, обнаружить и устранить существенные ошибки. Для создания функциональных диаграмм предлагается использовать методологию SADT (см. разд. 3.5.4 [2]).

Диаграммы потоков данных

Для описания потоков информации в системе применяются диаграммы потоков данных (DFD — Data flow diagrams). DFD позволяет описать требуемое поведение системы в виде совокупности процессов, взаимодействующих посредством связывающих их потоков данных. DFD показывает, как каждый из процессов преобразует свои входные потоки данных в выходные потоки данных и как процессы взаимодействуют между собой (см. разд. 3.5.5 [2]).

Диаграммы «сущность—связь»

Диаграмма сущность—связь — инструмент разработки моделей данных, обеспечивающий стандартный способ определения данных и отношений между ними. Она включает сущности и взаимосвязи, отражающие основные бизнес-правила предметной области. Такая диаграмма не слишком детализирована, в нее включаются основные сущности и связи между ними, которые удовлетворяют требованиям, предъявляемым к ИС (см. разд. 3.5.6 [2]).

Сетевой график

Сетевой график – это инструмент качества, предназначенный для планирования и управления работами. Изначально, в составе семи новых

инструментов управления качеством применялась стрелочная диаграмма, но для практического применения более часто используют сетевой график или диаграмму Ганта.

Все это разные инструменты (хотя часто в литературе и Интернете можно встретить, что это один и тот же инструмент с разными названиями), у каждого из которых есть свои достоинства и недостатки, однако все они служат одной цели – планированию и управлению работами.

Наиболее часто сетевой график применяется для проектов или различных работ, которые составляют набор взаимосвязанных действий. Его применение позволяет определить сроки завершения проекта и выявить возможные варианты сокращения сроков работ. Т.к. работы в сетевом графике взаимоувязаны по времени, то это дает возможность осуществлять контроль хода работ.

По своей сути сетевой график является графом, вершины которого представляют события (состояние работы или объекта в некоторый момент времени), а соединяющие вершины ребра (дуги графа) отображают работы.

Сетевой график, представленный в таком виде, является частью метода PERT (Program Evaluation and Review Technique), разработанного в 1957-58 годах одной из американских компаний для планирования и оценки хода работ проекта.

Существует и другой вариант представления сетевого графика, когда вершинами графа являются работы, а дуги показывают только взаимосвязь между ними.

Такой вариант используется чаще. Он является частью метода CPM (Critical Path Method – метод критического пути).

Порядок построения сетевого графика по методу критического пути следующий:

1. Определяется основная цель планирования – результат, который должен быть получен по завершении работ. Это дает возможность определить границы проекта и примерные сроки завершения работ.

2. Выявляются ограничения, влияющие на сетевой график и планируемые действия. Такими ограничениями обычно являются *какие-либо внешние условия, время и стоимость.*

3. Определяется состав задач (действий) необходимых для достижения поставленной цели. Состав задач можно выявить с помощью древовидной диаграммы (в этом случае сетевой график будет представлять только задачи верхних уровней древовидной диаграммы). Задачи указываются на отдельных карточках или стикерах.

4. На карточках, для каждой задачи отмечается длительность ее выполнения. *Можно указать ресурсы, инструменты и ответственных за выполнение задачи.* Длительность необходимо указывать в одних и тех же единицах измерения для всех задач (например, в минутах, часах, днях и т.д.). В противном случае составить сетевой график будет проблематично. Длительность задач должна быть величиной одного порядка. Например, если большинство задач выполняется за несколько часов, а одна за две-три недели, то это означает, что такая задача должна быть детализирована на составляющие.

5. Рассматриваются все задачи, и определяется, какая из них должна быть выполнена первой. Карточка с этой задачей располагается на сетевом графике слева, либо сверху. Если таких задач больше чем одна, то карточки располагаются одна над другой (одна рядом с другой).

6. Определяется задача, которая должна быть выполнена сразу же после первой. Карточка с этой задачей располагается справа от первой карточки (либо снизу, если выбран вертикальный вариант расположения задач). Если должны начинаться две и более задач, то карточки располагаются одна над другой (одна рядом с другой). Далее определяется задача, которая должна начинаться сразу же после второй, и так далее, пока все карточки с задачами не окажутся расположенными в цепочку.

Если задача должна начинаться до завершения предыдущей задачи, то предыдущую задачу необходимо разделить на составляющие. Задачи могут

выполняться параллельно, но при условии, что связь задач точно определена. Начало выполнения параллельных задач должно быть строго привязано к завершению предыдущей задачи (задач).

7. Отображаются связи между задачами – обычно в виде стрелок, которые показывают последовательность выполнения задач. Направление стрелок устанавливается слева направо (сверху вниз). Для того чтобы между карточками с задачами можно было прорисовать связи, карточки необходимо закрепить на какой-либо ровной поверхности.

Сейчас сетевой график, конечно, наиболее удобно строить, используя программные средства, например MS Visio.

8. Определяется раннее начало и раннее окончание каждой задачи. Для этого сетевой график просматривают в прямом направлении - начинают с первой задачи и далее по очереди двигаются к последней. При этом необходимо соблюдать правило - последующая задача не может быть начата, пока не завершены все предшествующие задачи. Раннее начало последующей задачи будет совпадать с ранним завершением предшествующей. Если предшествующих задач несколько, то ранним началом последующей задачи будет наибольшее из значений раннего окончания одной из предшествующих задач. Ранее окончание каждой из задач определяется как раннее начало плюс длительность задачи.

9. Определяется позднее начало и позднее окончание каждой задачи. Для этого сетевой график просматривают в обратном направлении - начинают с последней задачи и далее по очереди двигаются к первой. При этом необходимо соблюдать правило – предшествующая задача должна быть завершена до того, как начнется каждая из последующих задач. Позднее окончание задачи будет совпадать с поздним началом последующей задачи. Если последующих задач несколько, то поздним окончанием задачи будет наименьшее из значений позднего начала последующих задач. Позднее начало каждой задачи определяется как позднее окончание минус длительность задачи.

10. Определяется резерв времени для каждой задачи. Резерв времени вычисляется как разница между поздним и ранним началом или поздним и ранним окончанием задачи.

11. Определяется путь, где резерв времени для каждой задачи равен нулю. Этот путь называется критическим путем.

Если необходимо оптимизировать сетевой график, то в первую очередь проводится анализ задач, лежащих на критическом пути.

Древовидная диаграмма

Древовидная диаграмма – инструмент, предназначенный для систематизации причин рассматриваемой проблемы за счет их детализации на различных уровнях. Визуально диаграмма выглядит в виде «дерева» - в основании диаграммы находится исследуемая проблема, от которой «ответвляются» две или более причины, каждая из которых далее «разветвляется» еще на две или более причины и так далее.

Применяется древовидная диаграмма когда необходимо определить и упорядочить все потенциальные причины рассматриваемой проблемы, систематизировать результаты мозгового штурма в виде иерархически выстроенного логического списка, провести анализ причин проблемы, оценить применимость результатов различных решений проблемы, выстроить иерархическую взаимосвязь между элементами диаграммы сродства и пр.

Древовидная диаграмма строится следующим образом:

1. Определяется исследуемая проблема. Эта проблема будет являться основанием «ветвей» древовидной диаграммы. Проблему необходимо формулировать ясно и четко, таким образом, чтобы не возникало двоякого толкования формулировки. Если берется формулировка из другого инструмента качества (например, диаграммы сродства), то она должна совпадать с этой формулировкой.

2. Устанавливаются причины, которые приводят к возникновению рассматриваемой проблемы. Для этой цели может применяться метод

мозгового штурма. Если ранее применялась диаграмма сродства или диаграмма связей, то причины берутся из этих диаграмм. Причины размещаются на одном уровне диаграммы. Связь между исследуемой проблемой и причинами первого уровня отображается в виде линий. При выполнении данного шага необходимо проверять обоснованность размещения причин на первом уровне.

3. Каждая из причин первого уровня разбивается на более простые составляющие. Эти элементы будут являться вторым уровнем причин. Далее процесс повторяется до тех пор, пока каждая из причин более высокого уровня может быть детализирована как минимум на две составляющие.

4. Проводится проверка обоснованности размещения причин на соответствующих уровнях детализации для всей диаграммы целиком. Если все причины размещены правильно и обоснованно, то на этом построение древовидной диаграммы завершается.

Рисунок 1- Пример древовидной диаграммы

Преимущества древовидной диаграммы связаны с наглядностью и простотой ее применения и понимания. Кроме того, древовидная диаграмма может легко сочетаться с другими инструментами качества, дополняя их.

К недостаткам данного инструмента можно отнести субъективность расположения элементов на том или ином уровне детализации (особенно если выполняется индивидуальная работа).

В качестве примера сетевого графика рассматривается написание статьи на сайт. Нумерация действий в примере соответствует порядку шагов построения сетевого графика, указанному выше.

1. Определяем цель планирования - подготовить информационную статью для сайта. Требуемый результат - статья размещена на сайте.

2. Определяем ограничения, влияющие на сетевой график: срок подготовки статьи - не более 15 дней.

3. Определяем состав задач: предполагаем, что весь проект состоит из 5-ти задач.

Рисунок 2 – 5 задач планирования

4. Определяем длительность задач: указываем на карточках в графе "длительность" продолжительность выполнения каждой задачи. Длительность задач может определяться нормативами, либо на основании экспертной оценки.

Рисунок 3 – Определение длительности задач

5. Определяем первую задачу: первая задача, с которой начинается проект – «Разработать структуру статьи». Помещаем эту задачу сверху.

Рисунок 4 – Определение первой задачи

6. Определяем порядок следования оставшихся задач: Располагаем задачи в логической последовательности по порядку их исполнения. При этом проверяем условие, при котором каждая последующая задача не может начаться пока не завершены все предыдущие задачи.

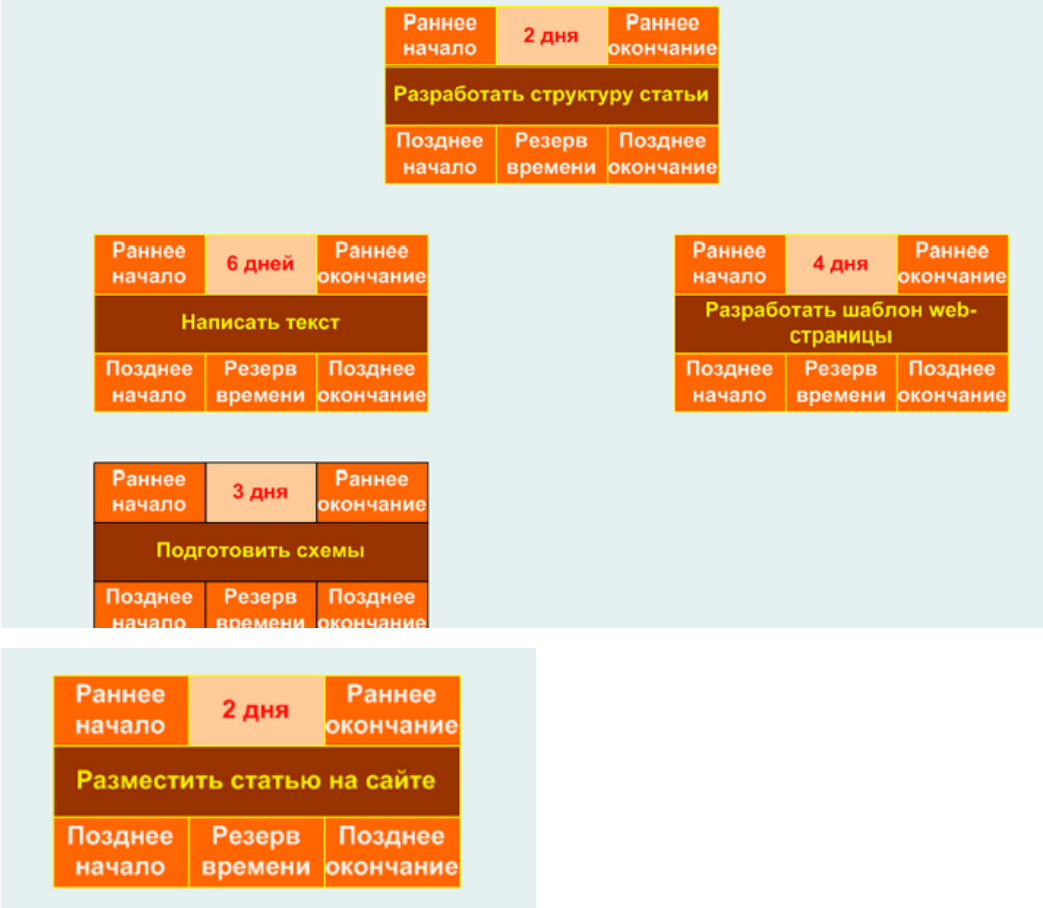

Рисунок 5 – Определение порядка следования задач

7. Устанавливаем связи между задачами.

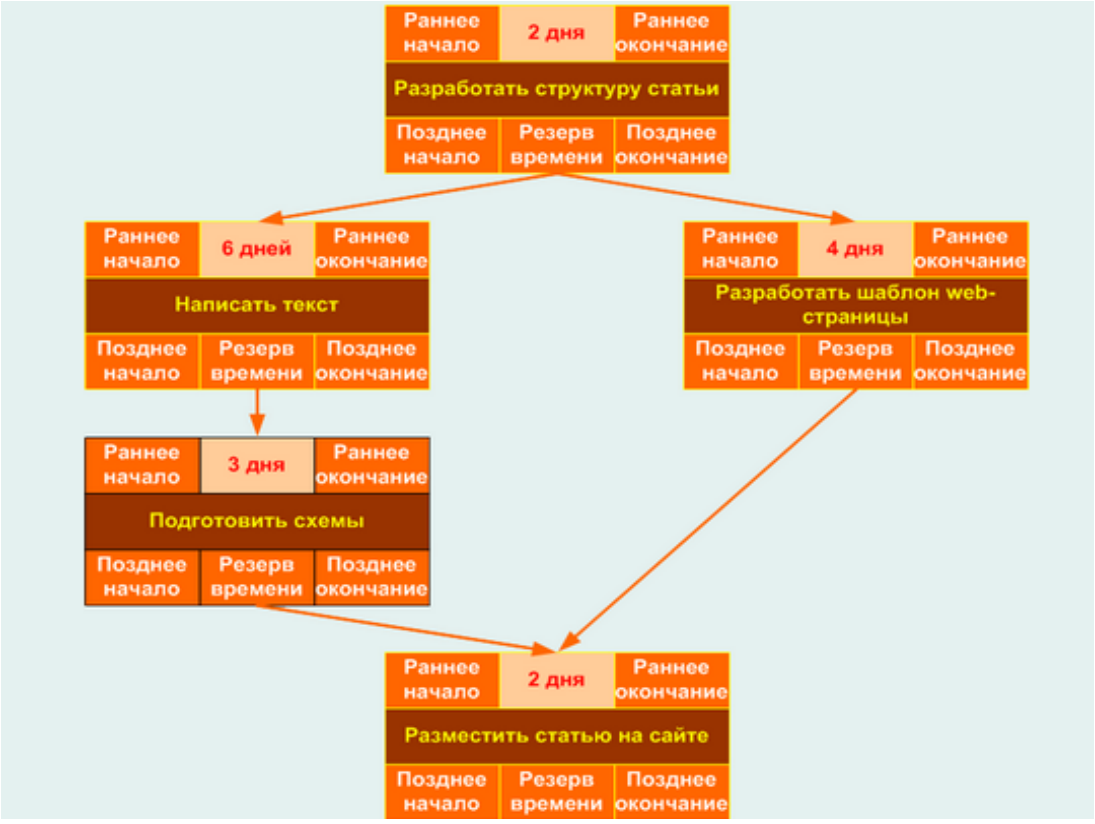

Рисунок 6 – Установление связей между задачами

8. Определяем раннее начало и раннее окончание каждой задачи:

Устанавливаем раннее начало задачи «Разработать структуру статьи» равным нулю, т.к. это первая задача. Раннее окончание рассчитывается как раннее начало плюс длительность задачи, т.е. 0+2=2.

Раннее начало задач «Написать текст» и «Разработать шаблон web - страницы» будет равно 2-м, т.к. эти задачи имеют одну предшествующую задачу.

Рассчитываем раннее окончание задач «Написать текст» и «Разработать шаблон web -страницы» - получаем 8 и 6 дней соответственно.

По аналогии, рассчитываем раннее начало и раннее окончание задачи «Подготовить схемы». Получаем ранее начало - 8 дней, раннее окончание - 11 дней.

Задача «Разместить статью на сайте» имеет две предшествующие задачи – «Подготовить схемы» (раннее окончание 11 дней) и «Разработать шаблон web -страницы» (раннее окончание 6 дней). Раннее начало задачи «Разместить статью на сайте» будет совпадать с наибольшим значением раннего окончания предшествующей задачи, т.е. 11 дней.

Раннее окончание задачи «Разместить статью на сайте» будет равняться 13 дням. Следовательно, длительность всего проекта составит 13 дней.

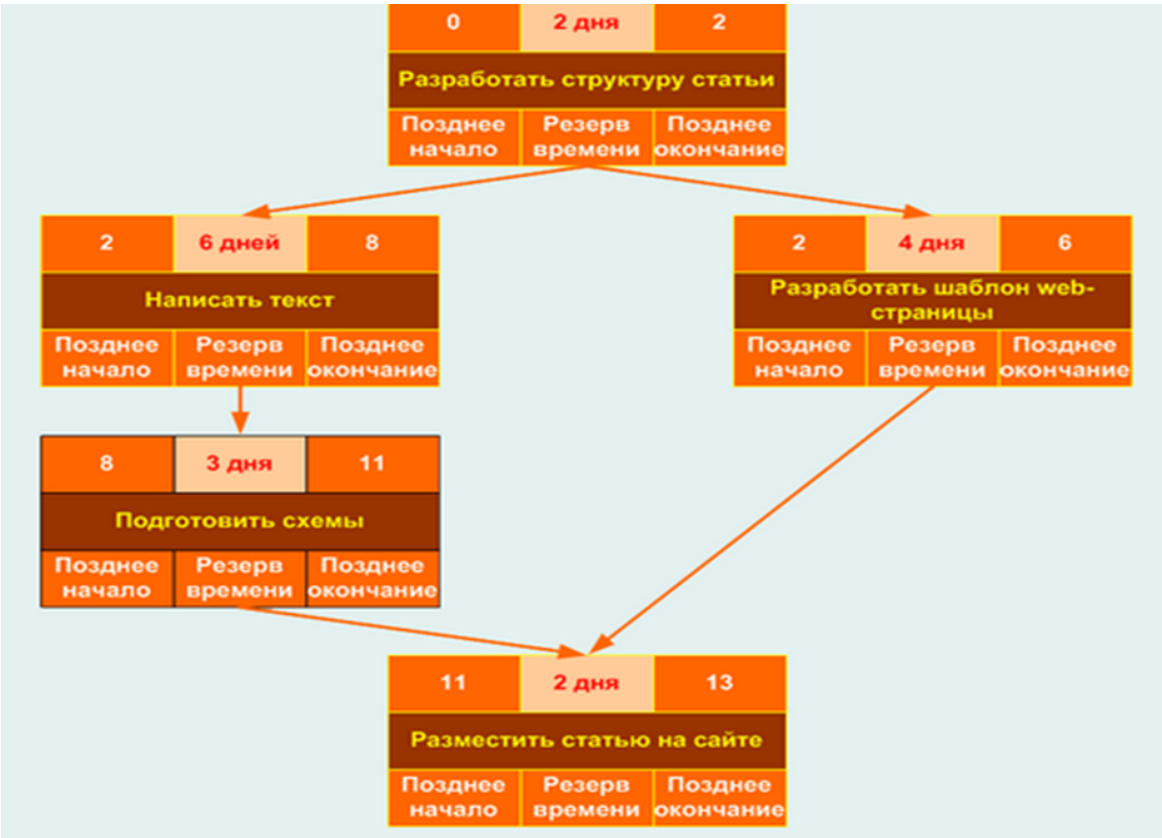

Рисунок 7 – Оценка времени выполнения проекта
(раннее окончание проекта)

9. Определяется позднее начало и позднее окончание каждой задачи.

Позднее окончание задачи «Разместить статью на сайте» совпадает с ранним окончанием этой задачи и равняется 13 дням. Позднее начало задачи рассчитывается как позднее окончание минус длительность, и равняется 11 дней.

Позднее окончание задач «Подготовить схемы» и «Разработать шаблон web -страницы» будет совпадать с поздним началом задачи «Разместить статью на сайте». Позднее начало этих задач будет равняться 8 дней (для задачи «Подготовить схемы») и 7 дней (для задачи «Разработать шаблон web -страницы»).

Рассчитываем позднее окончание и позднее начало задачи «Написать текст». Они составят 8 дней и 2 дня соответственно.

Задача «Разработать структуру статьи» имеет две последующие задачи, позднее начало которых составляет 2 дня (задача «Написать текст») и 7 дней

(задача «Разработать шаблон web -страницы»). Поэтому позднее окончание задачи «Разработать структуру статьи» составит наименьшее из этих значений, т.е. 2 дня. Позднее начало задачи равняется позднее окончание минус длительность, т.е. 0.

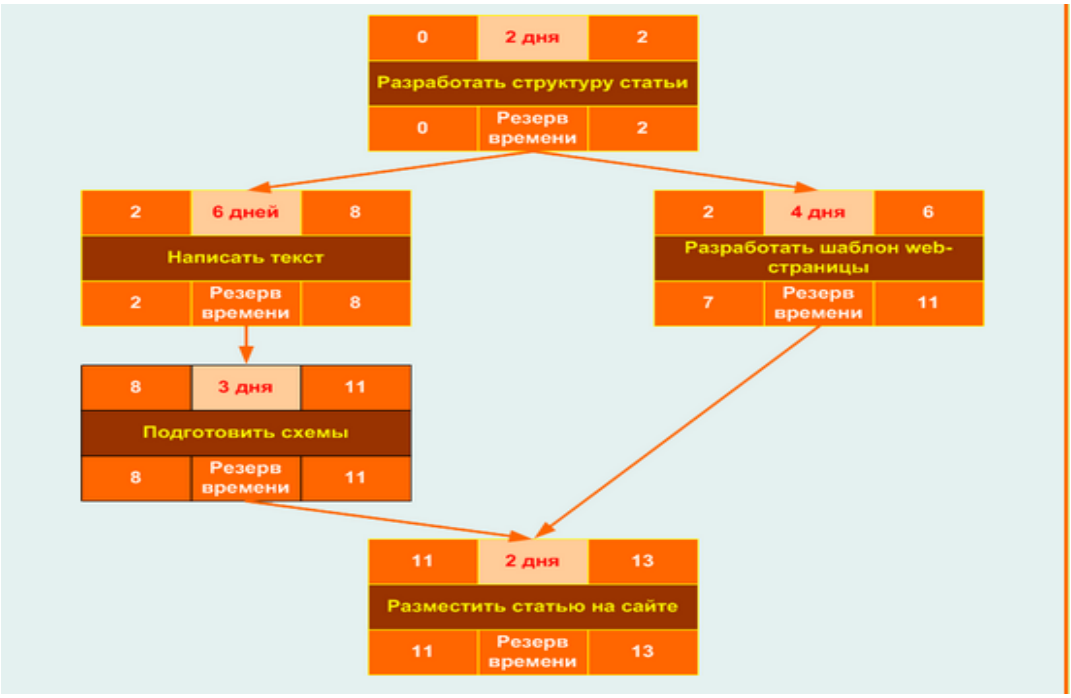

Рисунок 8 – Оценка времени выполнения проекта
(позднее окончание проекта)

10. Определяем резерв времени для каждой задачи: резерв времени возможен только для задачи «Разработать шаблон web -страницы». Для всех остальных задач резерв времени равен нулю.

11. Определяем критический путь: Критический путь выделен красным цветом. Чтобы сократить длительность проекта необходимо оптимизировать задачи, лежащие на критическом пути.

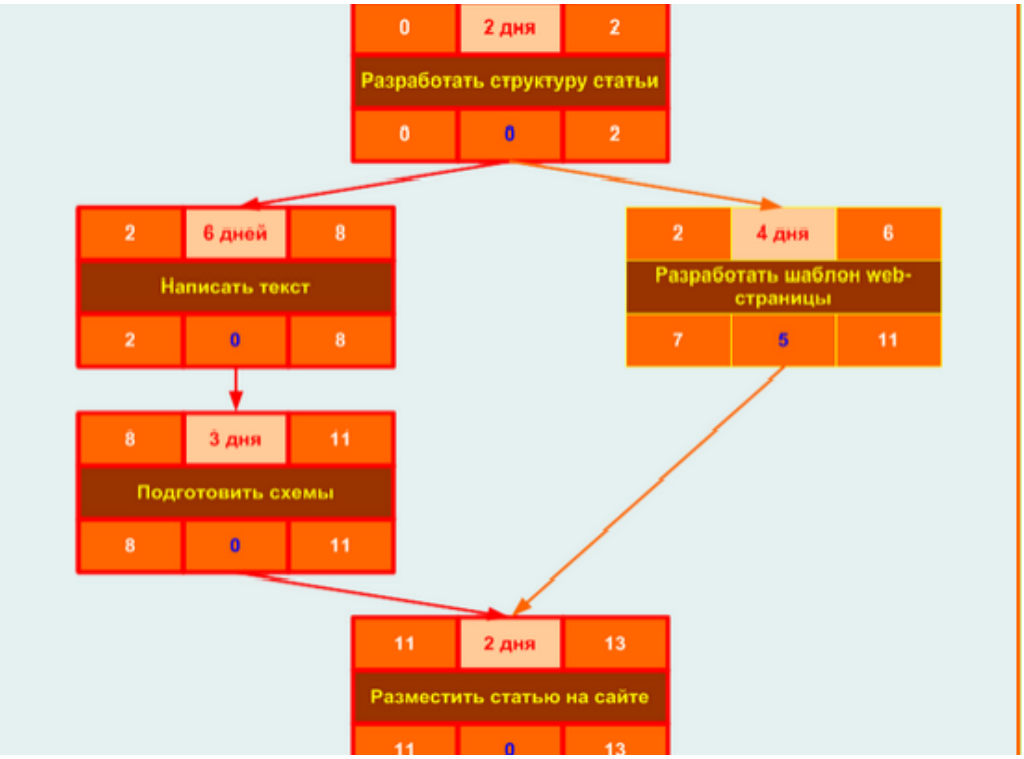

Рисунок 9 – Определение критического пути

Сетевой график позволяет наглядно представить все задачи проекта, увидеть их взаимосвязь и выявить критические аспекты планируемых работ. Кроме того, важным достоинством сетевого графика является возможность представить влияние той или иной задачи на ход выполнения последующих задач, что важно для контроля работ проекта.

Более подробную информацию о сетевых графиках можно посмотреть на http://pstu.ru/files/file/CTF/sp/vopr_i_otv/razd06.html

Порядок выполнения работы

1. На основе технического задания из лабораторной работы № 1 выполнить анализ функциональных и эксплуатационных требований к программному продукту.

2. Определить основные технические решения (выбор языка программирования, структура программного продукта, состав функций ПП, режимы функционирования) и занести результаты в документ, называемый «Эскизным проектом» (см. приложение 4 [2]).

3. Определить диаграммы потоков данных для решаемой

задачи.

4. Определить диаграммы «сущность—связь», если программный продукт содержит базу данных.

5. Определить функциональные диаграммы.

6. Определить диаграммы переходов состояний.

7. Определить спецификации процессов.

8. Добавить словарь терминов.

9. Представить сетевой график выполнения работ.

10. Оформить результаты, используя MS Office или MS Visio в виде эскизного проекта.

Требования к отчету по лабораторной работе:

1. Постановки задачи.

2. Документа «Эскизный проект», содержащего:

• выбор метода решения и языка программирования;

• спецификации процессов;

• все полученные диаграммы;

• словарь терминов.

2. Сетевой график.

Защита отчета по лабораторной работе заключается в предъявлении преподавателю полученных результатов и ответов на вопросы преподавателя.

Контрольные вопросы

1. Назовите этапы разработки программного обеспечения.

2. Что такое жизненный цикл программного обеспечения?

3. В чем заключается постановка задачи и предпроектные исследования?

4. Назовите функциональные и эксплуатационные требования к программному продукту.

5. Перечислите составляющие эскизного проекта.

6. Охарактеризуйте спецификации и модели.

7. Для чего необходимо использовать сетевой график?

Лабораторная работа №3

Тема: Выполнение оценки проекта на основе loc- и fp-метрик.

Цель: оценка границ времени выполнения проекта, измерить программный продукт и процесс его разработки с использованием размерно-ориентированных метрик и функционально-ориентированнхе метрик.

Методические указания к выполнению лабораторной работы

Основной рычаг в планирующих методах — вычисление ***границ времени выполнения задачи.***

Обычно используют следующие оценки:

1. Раннее время начала решения задачи $T_{\min}^{in}$ (при условии, что все предыдущие задачи решены в кратчайшее время).

2. Позднее время начала решения задачи $T_{\max}^{in}$ (еще не вызывает общую задержку проекта).

3. Раннее время конца решения задачи $T_{\min}^{out}$.

$$T_{min}^{out} = T_{min}^{in} + T_{реш}.$$

4. Позднее время конца решения задачи $T_{\max}^{out}$.

$$T_{max}^{out} = T_{max}^{in} + T_{реш}.$$

5. *Общий резерв* — количество избытков и потерь планирования задач во времени, *не* приводящих к увеличению длительности критического пути $T_{к.п}$.

Все эти значения позволяют руководителю (планировщику) количественно оценить успех в планировании, выполнении задач.

Рекомендуемое правило распределения затрат проекта — 40-20-40:

- на анализ и проектирование приходится 40% затрат (из них на

планирование и системный анализ — 5%);

- на кодирование — 20%;
- на тестирование и отладку — 40%.

Размерно-ориентированные метрики **прямо измеряют программный продукт и процесс его разработки.**

Основываются на *LOC*-оценках (*Lines Of Code). LOC-оценка* — это количество строк в программном продукте.

Исходные данные для расчета этих метрик сводятся в таблицу (табл. 2.1).

Таблица 2.1. Исходные данные для расчета LOC-метрик

Проект	Затраты, чел.-мес	Стоимость, тыс. $	KLOC, тыс. LOC	Прогр. док-ты, страниц	Ошибки	Люди
aaa01	24	168	12,1	365	29	3
bbb02	62	440	27,2	1224	86	5
ccc03	43	314	20,2	1050	64	6

На основе таблицы вычисляются размерно-ориентированные метрики *производительности и качества* (для каждого проекта):

$$\text{Производительность} = \frac{\text{Длина}}{\text{Затраты}}\left[\frac{\text{тыс.LOC}}{\text{чел.-мес}}\right];$$

$$\text{Качество} = \frac{\text{Ошибки}}{\text{Длина}}\left[\frac{\text{Единиц}}{\text{тыс.LOC}}\right];$$

$$\text{Удельная стоимость} = \frac{\text{Стоимость}}{\text{Длина}}\left[\frac{\text{Тыс\$}}{\text{LOC}}\right];$$

$$\text{Документированность} = \frac{\text{Страниц Документа}}{\text{Длина}}\left[\frac{\text{Страниц}}{\text{тыс.LOC}}\right].$$

Функционально-ориентированные метрики косвенно измеряют программный продукт и процесс его разработки. Вместо подсчета LOC-

оценки рассматривается не размер, а *функциональность или полезность продукта*.

Используется 5 информационных характеристик.

1. *Количество внешних вводов.* Подсчитываются все вводы пользователя, по которым поступают разные прикладные данные. Вводы должны быть отделены от запросов, которые подсчитываются отдельно.
2. *Количество внешних выводов.* Подсчитываются все выводы, по которым к пользователю поступают результаты, вычисленные программным приложением. В этом контексте выводы означают отчеты, экраны, распечатки, сообщения об ошибках. Индивидуальные единицы данных внутри отчета отдельно не подсчитываются.
3. *Количество внешних запросов.* Под запросом понимается диалоговый ввод, который приводит к немедленному программному ответу в форме диалогового вывода. При этом диалоговый ввод в приложении не сохраняется, а диалоговый вывод не требует выполнения вычислений. Подсчитываются все запросы — каждый учитывается отдельно.
4. *Количество внутренних логических файлов.* Подсчитываются все логические файлы (то есть логические группы данных, которые могут быть частью базы данных или отдельным файлом).
5. *Количество внешних интерфейсных файлов.* Подсчитываются все логические файлы из других приложений, на которые ссылается данное приложение.

Вводы, выводы и запросы относят к категории *транзакция.*

Транзакция — это элементарный процесс, различаемый пользователем и перемещающий данные между внешней средой и программным приложением.

В своей работе транзакции используют внутренние и внешние файлы. Приняты следующие определения.

Внешний ввод — элементарный процесс, перемещающий данные из внешней среды в приложение. Данные могут поступать с экрана ввода или из

другого приложения. Данные могут использоваться для обновления внутренних логических файлов. Данные могут содержать как управляющую, так и деловую информацию. Управляющие данные не должны модифицировать внутренний логический файл.

Внешний вывод — элементарный процесс, перемещающий данные, вычисленные в приложении, во внешнюю среду. Кроме того, в этом процессе могут обновляться внутренние логические файлы. Данные создают отчеты или выходные файлы, посылаемые другим приложениям. Отчеты и файлы создаются на основе внутренних логических файлов и внешних интерфейсных файлов. Дополнительно этот процесс может использовать вводимые данные, их образуют критерии поиска и параметры, не поддерживаемые внутренними логическими файлами. Вводимые данные поступают извне, но носят временный характер и не сохраняются во внутреннем логическом файле.

Внешний запрос — элементарный процесс, работающий как с вводимыми, так и с выводимыми данными. Его результат — данные, возвращаемые из внутренних логических файлов и внешних интерфейсных файлов. Входная часть процесса не модифицирует внутренние логические файлы, а выходная часть не несет данных, вычисляемых приложением (в этом и состоит отличие запроса от вывода).

Внутренний логический файл — распознаваемая пользователем группа логически связанных данных, которая размещена внутри приложения и обслуживается через внешние вводы.

Внешний интерфейсный файл — распознаваемая пользователем группа логически связанных данных, которая размещена внутри другого приложения и поддерживается им. Внешний файл данного приложения является внутренним логическим файлом в другом приложении.

Каждой из выявленных характеристик ставится в соответствие сложность. Для этого характеристике назначается *низкий, средний или высокий ранг*, а затем формируется числовая оценка ранга.

В этой таблице 10 элементов данных: День, Хиты, % от Сумма хитов, Сеансы пользователя, Сумма хитов (по рабочим дням), % от Сумма хитов (по рабочим дням), Сумма сеансов пользователя (по рабочим дням), Сумма хитов (по выходным дням), % от Сумма хитов (по выходным дням), Сумма сеансов пользователя (по выходным дням). Отметим, что поля День, Хиты, % от Сумма хитов, Сеансы пользователя имеют рекурсивные данные, которые в расчете не учитываются.

Таблица 2.2. Пример для расчета элементов данных

Уровень активности дня недели			
День	Хиты	% от Сумма хитов	Сеансы пользователя
Понедельник	1887	16,41	201
Вторник	1547	13,45	177
Среда	1975	17,17	195
Четверг	1591	13,83	191
Пятница	2209	19,21	200
Суббота	1286	11,18	121
Воскресенье	1004	8,73	111
Сумма по рабочим дням	9209	80,08	964
Сумма по выходным дням	2290	19,91	232

Таблица 2.3. Примеры элементов данных

Информационная характеристика	**Элементы данных**
Внешние Вводы	Поля ввода данных, сообщения об ошибках, вычисляемые значения, кнопки
Внешние Выводы	*Поля данных в отчетах, вычисляемые значения, сообщения об ошибках, заголовки столбцов, которые читаются из внутреннего файла*
	Вводимые элементы: поле,

	используемое для поиска, щелчок мыши. Выводимые элементы — отображаемые на экране поля
Внешние Запросы	

Таблица 2.4 Правила учета элементов данных из графического интерфейса пользователя

Элемент данных	Правило учета
Группа радиокнопок	Т.к. в группе пользователь выбирает только одну радиокнопку, все радиокнопки группы считаются одним элементом данных
Группа флажков (переключателей)	Так как в группе пользователь может выбрать несколько флажков, каждый флажок считают элементом данных
Командные кнопки	Командная кнопка может определять действие добавления, изменения, запроса. Кнопка ОК может вызывать транзакции (различных типов). Кнопка Next может быть входным элементом запроса или вызывать другую транзакцию. Каждая кнопка считается отдельным элементом данных
Списки	Список может быть внешним запросом, но результат запроса может быть элементом данных внешнего ввода

Таблица 2.5. Ранг и оценка сложности внешних вводов

Ссылки на файлы	**Элементы данных**		
	1-4	**5-15**	**>15**

0-1	Низкий (3)	Низкий (3)	Средний (4)
2	Низкий (3)	Средний (4)	Высокий (6)
>2	Средний (4)	Высокий (6)	Высокий (6)

Таблица 2.6. Ранг и оценка сложности внешних выводов

Ссылки на файлы	**Элементы данных**		
	1-4	**5-19**	**>19**
0-1	Низкий (4)	Низкий (4)	Средний (5)
2-3	Низкий (4)	Средний (5)	Высокий (7)
>3	Средний (5)	Высокий (7)	Высокий (7)

Таблица 2.7. Ранг и оценка сложности внешних запросов

Ссылки на файлы	**Элементы данных**		
	1-4	**5-19**	**>19**
0-1	Низкий (3)	Низкий (3)	Средний (4)
2-3	Низкий (3)	Средний (4)	Высокий (6)
>3	Средний (4)	Высокий (6)	Высокий (6)

Таблица 2.8. Ранг и оценка сложности внутренних логических файлов

Типы элементов-записей	**Элементы данных**		
	1-19	**20-50**	**>50**
1	Низкий (7)	Низкий (7)	Средний (10)
2-5	Низкий (7)	Средний (10)	Высокий (15)
>5	Средни (10)	Высокий (15)	Высокий (15)

Таблица 2.9. Ранг и оценка сложности внешних интерфейсных файлов

Типы элементов-записей	**Элементы данных**		
	1-19	**20-50**	**>50**
1	Низкий (5)	Низкий (5)	Средний (7)
2-5	Низкий (5)	Средний (7)	Высокий (10)
>5	Средний (7)	Высокий (10)	Высокий (10)

если во внешнем запросе ссылка на файл используется как на этапе ввода, так и на этапе вывода, она учитывается только один раз. Такое же правило распространяется и на элемент данных (однократный учет).

После сбора всей необходимой информации приступают к расчету метрики — *количества функциональных указателей FP* (Function Points). Автором этой метрики является А. Албрехт (1979) [7].

Исходные данные для расчета сводятся в табл. 2.10.

Таблица 2.10. Исходные данные для расчета FP-метрик

Имя характеристики	**Ранг, сложность, количество**			
	Низкий	**Средний**	**Высокий**	**Итого**
Внешние вводы	0x3 = __	0x4 = __	0x6 = __	= 0
Внешние выводы	0x4 = __	0x5 = __	0x7 = __	= 0
Внешние запросы	0x3 = __	0x4 = __	0x6 = __	= 0
Внутренние логические файлы	0x7 = __	0x 10= __	0x15 = __	= 0
Внешние интерфейсные файлы	0x5 = __	0x7 = __	0x10 = __	= 0
Общее количество				= 0

В таблицу заносится количественное значение характеристики каждого

вида (по всем уровням сложности). Места подстановки значений отмечены прямоугольниками (прямоугольник играет роль метки-заполнителя). Количественные значения характеристик умножаются на числовые оценки сложности. Полученные в каждой строке значения суммируются, давая полное значение для данной характеристики. Эти полные значения затем суммируются по вертикали, формируя общее количество.

Количество функциональных указателей вычисляется по формуле

$$\mathbf{FP = Общее\ количество\ x\ (0{,}65 + 0{,}01\ x} \sum_{i=1}^{14} F_i \mathbf{)},$$

где F_i — коэффициенты регулировки сложности.

Каждый коэффициент может принимать следующие значения: 0 — нет влияния, 1 — случайное, 2 — небольшое, 3 — среднее, 4 — важное, 5 — основное.

Значения выбираются эмпирически в результате ответа на 14 вопросов, которые характеризуют системные параметры приложения (табл. 2.11).

Таблица 2.11. Определение системных параметров приложения

№	Системный параметр	Описание
1	Передачи данных	Сколько средств связи требуется для передачи или обмена информацией с приложением или системой?
2	Распределенная обработка данных	Как обрабатываются распределенные данные и функции обработки?
3	Производительность	Нуждается ли пользователь в фиксации времени ответа или производительности?.
4	Распространенность используемой конфигурации	Насколько распространена текущая аппаратная платформа, на которой будет выполняться приложение?
5	Скорость транзакций	Как часто выполняются транзакции? (каждый день, каждую неделю, каждый месяц)

6	Оперативный ввод данных	Какой процент информации надо вводить в режиме онлайн?
7	Эффективность работы конечного пользователя	Приложение проектировалось для обеспечения эффективной работы конечного пользователя?
8	Оперативное обновление	Как много внутренних файлов обновляется в онлайновой транзакции?
9	Сложность обработки	Выполняет ли приложение интенсивную логическую или математическую обработку?
10	Повторная используемость	Приложение разрабатывалось для удовлетворения требований одного или многих пользователей?
11	Легкость инсталляции	Насколько трудны преобразование и инсталляция приложения?
12	Легкость эксплуатации	Насколько эффективны и/или автоматизированы процедуры запуска, резервирования и восстановления?
13	Разнообразные условия размещения	Была ли спроектирована, разработана и поддержана возможность инсталляции приложения в разных местах для различных организаций?
14	Простота изменений	Была ли спроектирована, разработана и поддержана в приложении простота изменений?

После вычисления FP на его основе формируются метрики производительности, качества и т. д.:

$$\text{Производитель} = \frac{\text{ФункцУказатель}}{\text{Затраты}}\left[\frac{\text{FP}}{\text{чел.}-\text{мес}}\right];$$

$$\text{Качество} = \frac{\text{Ошибки}}{\text{ФункцУказатель}}\left[\frac{\text{Единиц}}{\text{FP}}\right];$$

$$\text{Удельная стоимость} = \frac{\text{Стоимость}}{\text{ФункцУказатель}}\left[\frac{\text{Тыс.\$}}{\text{FP}}\right];$$

$$\text{Документированность} = \frac{\text{СтраницДокумента}}{\text{ФункцУказатель}}\left[\frac{\text{Страниц}}{\text{FP}}\right].$$

Область применения метода функциональных указателей — коммерческие информационные системы.

Для продуктов с высокой алгоритмической сложностью используются метрики *указателей свойств* (Features Points). Они применимы к системному и инженерному ПО, ПО реального времени и встроенному ПО.

Для вычисления указателя свойств добавляется одна характеристика - *количество алгоритмов.* Алгоритм здесь определяется как ограниченная подпрограмма вычислений, которая включается в общую компьютерную программу.

Примеры алгоритмов: обработка прерываний, инвертирование матрицы, расшифровка битовой строки. Для формирования указателя свойств составляется табл. 2.12.

Таблица 2.12. Исходные данные для расчета указателя свойств

№	Характеристика	Кол-во	Сложность	Итого
1	Вводы	0	x4	= 0
2	Выводы	0	x5	= 0
3	Запросы	0	x4	= 0
4	Логические файлы	0	x7	= 0
5	Интерфейсные файлы	0	x7	= 0
6	Количество алгоритмов	0	x3	= 0
	Общее количество			= 0

После заполнения таблицы по формуле (2.1) вычисляется значение указателя свойств.

Для сложных систем реального времени это значение на 25-30% больше значения, вычисляемого по таблице для количества функциональных указателей.

Как показано в табл. 2.13, результаты пересчета зависят от языка программирования, используемого для реализации ПО.

Таблица 2.13. Пересчет FP-оценок в LOC-оценки

Язык программирования	**Количество операторов на один FP**
Ассемблер	320
C	128
Кобол	106
Фортран	106
Паскаль	90
C++	64
Java	53
Ada 95	49
Visual Basic	32
Visual C++	34
Delphi Pascal	29
Smalltalk	22
Perl	21
HTML3	15
LISP	64
Prolog	64
Miranda	40
Haskell	38

Выполнение оценки в ходе руководства проектом

Процесс руководства программным проектом начинается с множества действий, объединяемых общим названием *планирование проекта.* Первое из этих действий — выполнение оценки. Оно закладывает фундамент для

других действий по планированию проекта. При оценке проекта чрезвычайно высока цена ошибок. Очень важно провести оценку с минимальным риском.

Выполнение оценки проекта на основе LOC- и FP-метрик

Цель этой деятельности — сформировать предварительные оценки, которые позволят:

- предъявить заказчику корректные требования по стоимости и затратам на разработку программного продукта;
- составить план программного проекта.

При выполнении оценки возможны два варианта использования LOC- и FP-данных:

- в качестве оценочных переменных, определяющих размер каждого элемента продукта;
- в качестве метрик, собранных за прошлые проекты и входящих в метрический базис фирмы.

Обсудим шаги процесса оценки.

- *Шаг 1.* Область назначения проектируемого продукта разбивается на ряд функций, каждую из которых можно оценить индивидуально:

$$f_1, f_2, \ldots, f_n.$$

- *Шаг 2.* Для каждой функции f_i, планировщик формирует лучшую $LOC_{лучшi}$ ($FP_{лучшi}$), худшую $LOC_{худшi}$ ($FP_{худшi}$) и вероятную оценку $LOC_{вероятнi}$ ($FP_{вероятнi}$). Используются опытные данные (из метрического базиса) или интуиция. Диапазон значения оценок соответствует степени предусмотренной неопределенности.
- *Шаг 3.* Для каждой функции/ в соответствии с β-распределением вычисляется ожидаемое значение LOC- (или FP-) оценки:

$$LOC_{ожi} = (LOC_{лучшi} + LOC_{худшi} + 4x\ LOC_{вероятнi})/ 6.$$

- *Шаг 4.* Определяется значение LOC- или FP-производительности разработки функции.

Используется один из трех подходов:

1) для всех функций принимается одна и та же метрика средней производительности $ПРОИЗВ_{ср}$, взятая из метрического базиса;

2) для *i*-й функции на основе метрики средней производительности вычисляется настраиваемая величина производительности:

$$ПРОИЗВ_i = ПРОИЗВ_{ср} x (LOC_{ср} / LOC_{ожi}),$$

где $LOC_{ср}$ — средняя LOC-оценка, взятая из метрического базиса (соответствует средней производительности);

3) для *i*-й функции настраиваемая величина производительности вычисляется по аналогу, взятому из метрического базиса:

$$ПРОИЗВ_i = ПРОИЗВ_{анi} x (LOC_{анi} / LOC_{ожi}).$$

Первый подход обеспечивает минимальную точность (при максимальной простоте вычислений), а третий подход — максимальную точность (при максимальной сложности вычислений).

- *Шаг* 5. Вычисляется общая оценка затрат на проект: для первого подхода

$$ЗАТРАТЫ = (\sum_{i=1}^{n} LOC_{ожi}) / ПРОИЗВ_{ср} [чел.-мес];$$

для второго и третьего подходов

$$ЗАТРАТЫ = \sum_{i=1}^{n} (LOC_{ожi} / ПРОИЗВ_i) [чел.-мес].$$

- *Шаг 6.* Вычисляется общая оценка стоимости проекта: для первого и второго подходов

$$СТОИМОСТЬ = (\sum_{i=1}^{n} LOC_{ожi}) \times УД\_СТОИМОСТЬ_{ср},$$

где УД_СТОИМОСТЬ$_{ср}$ — метрика средней стоимости одной строки, взятая из метрического базиса.

для третьего подхода

$$\text{СТОИМОСТЬ} = \sum_{i=1}^{n} (\text{LOC}_{\text{ож}i} \times \text{УД\_СТОИМОСТЬ}_{\text{ан}i}),$$

где УД_СТОИМОСТЬ$_{ани}$ — метрика стоимости одной строки аналога, взятая из метрического базиса. Пример применения данного процесса оценки приведем ниже.

Пример выполнения оценки проекта на основе LOC- и FP-метрик

Предположим, что поступил заказ от концерна «СУПЕРАВТО». Необходимо создать ПО для рабочей станции дизайнера автомобиля (РДА). Заказчик определил проблемную область проекта в своей спецификации:

- ПО РДА должно формировать 2- и 3-мерные изображения для дизайнера;
- дизайнер должен вести диалог с РДА и управлять им с помощью стандартизованного графического пользовательского интерфейса;
- геометрические данные и прикладные данные должны содержаться в базе данных РДА;
- модули проектного анализа рабочей станции должны формировать данные для широкого класса дисплеев SVGA;
- ПО РДА должно управлять и вести диалог со следующими периферийными устройствами: мышь, дигитайзер (графический планшет для ручного ввода), плоттер (графопостроитель), сканер, струйный и лазерный принтеры.

Прежде всего

-детализировать проблемную область.

-выделить базовые функции ПО

-очертить количественные границы.

-определить, что такое «стандартизованный графический пользовательский интерфейс», какими должны быть размер и другие характеристики базы данных РДА и т. д.

Будем считать, что эта работа проделана и что идентифицированы следующие основные функции ПО:

1. Средства управления пользовательским интерфейсом СУПИ.

2. Анализ двухмерной графики А2Г.

3. Анализ трехмерной графики А3Г.

4. Управление базой данных УБД.

5. Средства компьютерной дисплейной графики КДГ.

6. Управление периферией УП.

7. Модули проектного анализа МПА.

Теперь нужно оценить каждую из функций количественно, с помощью LOC-оценки.

По каждой функции эксперты предоставляют лучшее, худшее и вероятное значения. Ожидаемую LOC-оценку реализации функции определяем по формуле

$$\mathbf{LOC_{ожі} = (LOC_{лучші} + LOC_{худші} + 4 \times LOC_{вероятні}) / 6,}$$

результаты расчетов заносим в табл. 2.22.

Таблица 2.22. Начальная таблица оценки проекта

Функция	**Лучш. [LOC]**	**Вероят. [LOC]**	**Худш. [LOC]**	**Ожид. [LOC]**	**Уд. стоимость [$/LOC]**	**Стоимость[$]**	**Произв. [LOC/ [чел-мес]**	**Затраты [чел-мес]**

СУПИ	1800	2400	2650	2340			
А2Г	4100	5200	7400	5380			
АЗГ	4600	6900	8600	6800			
УБД	2950	3400	3600	3350			
КДГ	4050	4900	6200	4950			
УП	2000	2100	2450	2140			
МПА	6600	8500	9800	8400			
Итого				33360			

Для определения удельной стоимости и производительности обратимся в архив фирмы, где хранятся данные метрического базиса, собранные по уже выполненным проектам. Предположим, что из метрического базиса извлечены данные по функциям-аналогам, представленные в табл. 2.23.

Видно, что наибольшую удельную стоимость имеет строка функции управления периферией (требуются специфические и конкретные знания по разнообразным периферийным устройствам), наименьшую удельную стоимость — строка функции управления пользовательским интерфейсом (применяются широко известные решения).

Таблица 2.23. Данные из метрического базиса фирмы

Функция	**$LOC_{ані}$**	**$УД\_СТОИМОСТЬ_{ані}$[$/ LOC]**	**$ПРОИЗВ_{ані}$[LOC/чел-мес]**
СУПИ	585	14	1260
А_Г	3000	20	440
УБД	1117	18	720
КДГ	2475	22	400
УП	214	28	1400
МПА	1400	18	1800

Считается, что удельная стоимость строки является константой и не изменяется от реализации к реализации. Следовательно, стоимость разработки каждой функции рассчитываем по формуле

$$\text{СТОИМОСТЬ}_i = \text{LOC}_{\text{ожi}} \times \text{УД\_СТОИМОСТЬ}_{\text{ані}}.$$

Для вычисления производительности разработки каждой функции выберем самый точный подход — подход настраиваемой производительности:

$$\text{ПРОИЗВ}_i = \text{ПРОИЗВ}_{\text{ані}} \times (\text{LOC}_{\text{ані}} / \text{LOC}_{\text{ожi}}).$$

Соответственно, затраты на разработку каждой функции будем определять по выражению

$$\text{ЗАТРАТЫ}_i = (\text{LOC}_{\text{ожi}} / \text{ПРОИЗВ}_i)[\text{чел.-мес}].$$

Теперь мы имеем все необходимые данные для завершения расчетов. Заполним до конца таблицу оценки нашего проекта (табл. 2.24).

Таблица 2.24. Конечная таблица оценки проекта

Функция	Лучш.	Вероят.	Худш.	Ожид. [LOC]	Уд. стоимость [S/LOC]	Стоимость [$]	Произв. [LOC/ чел.-мес]	Затраты [чел-мес]
СУПИ	1800	2400	2650	2340	14	32760	315	7,4
А2Г	4100	5200	7400	5380	20	107600	245	21,9
А3Г	4600	6900	8600	6800	20	136000	194	35,0
УБД	2950	3400	3600	3350	18	60300	240	13,9
КДГ	4050	4900	6200	4950	22	108900	200	24,7
УП	2000	2100	2450	2140	28	59920	140	15,2
МПА	6600	8500	9800	8400	18	151200	300	28,0
Итого				33360		656680		146

Учитывая важность полученных результатов, проверим расчеты с помощью FP-указателей. На данном этапе оценивания разумно допустить, что все информационные характеристики имеют средний уровень сложности. В этом случае результаты экспертной оценки принимают вид, представленный в табл. 2.25, 2.26.

Таблица 2.25. Оценка информационных характеристик проекта

Характеристика	***Луч ш.***	***Веро ят***	***Худ ш.***	***Ож ид.***	***Сложн ость***	***Колич ество***
Вводы	20	24	30	24	x 4	96
Выводы	12	15	22	16	x 5	80
Запросы	16	22	28	22	x 4	88
Логические файлы	4	4	5	4	x 10	40
Интерфейсные файлы	2	2	3	2	x 7	14
Общее количество						318

Таблица *2.26.* Оценка системных параметров проекта

Коэффициент регулировки сложности		**Оценка**
F_1	Передачи данных	2
F_2	Распределенная обработка данных	0
F_3	Производительность	4
F_4	Распространенность используемой конфигурации	3
F_5	Скорость транзакций	4
F_6	Оперативный ввод данных	5
F_7	Эффективность работы конечного пользователя	5
F_8	Оперативное обновление	3
F_9	Сложность обработки	5
F_{10}	Повторная используемость	4
F_{11}	Легкость инсталляции	3

F_{12}	Легкость эксплуатации	4
F_{13}	Разнообразные условия размещения	5
F_{14}	Простота изменений	5

Таким образом, получаем:

$$\textbf{FP = Общее количество x (0,65+ 0,01 x}\sum_{i=1}^{14} F_i\textbf{) = 318 x 1,17 = 372.}$$

Используя значение производительности, взятое в метрическом базисе фирмы,

Производительность = 2,55 [FP / чел.-мес],

вычисляем значения затрат и стоимости:

Затраты = FP / Производительность = 145,9 [чел.-мес],

Стоимость = Затраты x $4500 = $656500.

Итак, результаты проверки показали хорошую достоверность результатов.

Но мы не будем останавливаться на достигнутом и организуем еще одну проверку, с помощью модели COCOMO II.

Примем, что все масштабные факторы и факторы затрат имеют номинальные значения. В силу этого показатель степени B = 1,16, а множитель поправки M_p= 1. Кроме того, будем считать, что автоматическая генерация кода и повторное использование компонентов не предусматриваются. Следовательно, мы вправе применить формулу

$$\text{ЗАТРАТЫ} = A \times \text{РАЗМЕР}^{B}[\text{чел.-мес}]$$

и получаем:

$$\text{ЗАТРАТЫ} = 2{,}5(33{,}3)^{1{,}16} = 145{,}87\ [\text{чел.-мес}].$$

Соответственно, номинальная длительность проекта равна

$$\text{Длительность} = [3{,}0 \times (\text{ЗАТРАТЫ})^{(0{,}33+0{,}2(B-1{,}01))}] = 3(145{,}87)^{0{,}36} = 18[\text{мес}].$$

Подведем итоги. Выполнена предварительная оценка программного проекта. Для минимизации риска оценивания использованы три методики, доказавшие корректность полученных результатов.

Задание к лабораторной работе.

Для своего программного проекта (для которого написали техническое задание) сделать оценку границ времени выполнения проекта, измерить программный продукт и процесс его разработки с использованием размерно-ориентированных и функционально-ориентированных метрик.

Порядок выполнения работы

1. Вычислить границы времени выполнения задачи: раннее и позднее времена начала решения задачи, раннее и позднее времена конца решения задачи, резерв.
2. Измерить программный продукт и процесс его разработки с использованием размерно-ориентированных метрик.
3. Измерить программный продукт и процесс его разработки с использованием функционально-ориентированных метрик.

Требование к отчету по лабораторной работе

1. Свой программный проект (для которого написали техническое задание)
2. Краткие теоретические сведения.
3. Расчеты LOC и FP .
4. Выводы.

Контрольные вопросы

1. Охарактеризуйте рекомендуемое правило распределения затрат проекта.

2. Какие размерно-ориентированные метрики вы знаете?

3. Для чего используют размерно-ориентированные метрики?

4. Определите достоинства и недостатки размерно-ориентированных метрик.

5. Что такое функциональный указатель?

6. От каких информационных характеристик зависит функциональный указатель?

Лабораторная работа №4

Тема: Анализ чувствительности программного проекта на основе модели COCOMO II

Цель: применение модели COCOMO II к анализу чувствительности програмного проекта при изменении русловий

Методические указания к лабораторной работе

В 1995 году Боэм ввел более совершенную модель COCOMO II, ориентированную на применение ***в программной инженерии XXI*** века [21].

В состав COCOMO II входят:

- **модель композиции приложения;**
- **модель раннего этапа проектирования;**
- **модель этапа пост-архитектуры.**

Для описания моделей COCOMO II требуется информация о размере программного продукта. Возможно использование LOC-оценок, объектных указателей, функциональных указателей.

1 Модель композиции приложения

Модель композиции используется ***на ранней стадии конструирования ПО***, когда:

- рассматривается макетирование пользовательских интерфейсов;
- обсуждается взаимодействие ПО и компьютерной системы;
- оценивается производительность;
- определяется степень зрелости технологии.

Модель композиции приложения ориентирована на применение объектных указателей.

Объектный указатель — средство косвенного измерения ПО, для его

расчета определяется количество экранов (как элементов пользовательского интерфейса), отчетов и компонентов, требуемых для построения приложения.

Как показано в табл. 2.15, каждый объектный экземпляр (экран, отчет) относят к одному из трех уровней сложности.

В свою очередь, сложность является функцией от параметров клиентских и серверных таблиц данных (см. табл. 2.16 и 2.17), которые требуются для генерации экрана и отчета, а также от количества представлений и секций, входящих в экран или отчет.

Таблица 2.15. Оценка количества объектных указателей

Тип объекта	Количество	Вес			Итого
		Простой	Средний	Сложный	
Экран	0	x1	x2	x3	= 0
Отчет	0	x2	x5	x8	= 0
3GL компонент	0			x10	= 0
Объектные указатели					= 0

Таблица 2.16. Оценка сложности экрана

Экраны	**Количество серверных (срв) и клиентских (клт) таблиц данных**		
Количество представлений	Всего < 4 (< 2 срв, <3клт)	Всего < 8 (2-3 срв, 3-5 клт)	Всего > 8 (>3срв, >5клт)
<3	Простой	Простой	Средний
3-7	Простой	Средний	Сложный
>8	Средний	Сложный	Сложный

Таблица 2.17. Оценка сложности отчета

Отчеты	**Количество серверных (срв) и клиентских (клт) таблиц данных**		
Количество представлений	Всего < 4 (< 2 срв, < 3 клт)	Всего < 8 (2-3 срв, 3-5 клт)	Всего > 8 (>3срв, > 5 клт)
0 или 1	Простой	Простой	Средний
2 или 3	Простой	Средний	Сложный
>4	Средний	Сложный	Сложный

Для учета реальных условий разработки вычисляется **процент повторного использования программных компонентов %REUSE** и определяется количество новых объектных указателей **NOP:**

NOP = (Объектные указатели) x [(100 - %REUSE) /100].

Для оценки затрат, основанной на величине NOP, надо знать скорость разработки продукта PROD. Эту скорость определяют по табл. 2.18, учитывающей уровень опытности разработчиков и зрелость среды разработки.

Проектные затраты оцениваются по формуле

ЗАТРАТЫ = NOP /PROD [чел.-мес],

где PROD — производительность разработки, выраженная в терминах объектных указателей.

Таблица 2.18. Оценка скорости разработки

Опытность возможности разработчика /	**Зрелость возможности среды разработки** /	**PROD**
Очень низкая	Очень низкая	**4**
Низкая	Низкая	**7**
Номинальная	Номинальная	**13**
Высокая	Высокая	**25**
Очень высокая	Очень высокая	**50**

В более развитых моделях дополнительно учитывается множество масштабных факторов, формирователей затрат, процедур поправок.

2 Модель раннего этапа проектирования

Модель раннего этапа проектирования используется в период, когда стабилизируются требования и определяется базисная программная архитектура.

Основное уравнение этой модели имеет следующий вид:

$$\textbf{ЗАТРАТЫ} = A \times \textbf{РАЗМЕР}^{B} \times M_e + \textbf{ЗАТРАТЫ}_{auto}[\text{чел.-мес}],$$

где:

-масштабный коэффициент ***A = 2,5;***

-показатель ***B*** отражает нелинейную зависимость затрат от размера проекта (размер системы РАЗМЕР выражается в тысячах LOC);

-множитель поправки M_e зависит от 7 формирователей затрат, характеризующих продукт, процесс и персонал;

Масштабный фактор (W_i)	**Пояснение**
Предсказуемость PREC	***Отражает предыдущий опыт организации в реализации проектов этого типа.*** Очень низкий означает отсутствие опыта. Сверхвысокий означает, что организация полностью знакома с этой прикладной областью
Гибкость разработки FLEX	***Отражает степень гибкости процесса разработки.*** Очень низкий означает, что используется заданный процесс. Сверхвысокий означает, что клиент установил только общие цели
Разрешение архитектуры /риска RESL	***Отражает степень выполняемого анализа риска***. Очень низкий означает малый анализ. Сверхвысокий означает полный и сквозной анализ риска
Связность группы	***Отражает, насколько хорошо разработчики группы знают друг друга и насколько удачно они совместно работают.*** Очень низкий означает очень трудные взаимодействия.

TEAM	Сверхвысокий, означает интегрированную группу, без проблем взаимодействия
Зрелость процесса PMAT	***Означает зрелость процесса в организации.*** Вычисление этого фактора может выполняться по вопроснику CMM

-слагаемое $\mathbf{ЗАТРАТЫ_{auto}}$ отражает затраты на автоматически генерируемый программный код.

Значение показателя степени ***B*** изменяется в диапазоне **1,01... 1,26**, зависит от пяти масштабных факторов W_i и вычисляется по формуле

$$B = 1{,}01 + 0{,}01\sum_{i=1}^{5} W_i .$$

Общая характеристика масштабных факторов приведена в табл. 2.19, а табл. 2.20 позволяет определить оценки этих факторов. Оценки принимают **6** значений: от очень низкой **(5)** до сверхвысокой **(0).**

Таблица 2.19. Характеристика масштабных факторов

Пример рассмотрим компанию, которая берет проект в малознакомой проблемной области.

Положим, что заказчик не определил используемый процесс разработки и не допускает выделения времени на всесторонний анализ риска.

Для реализации этой программной системы нужно создать новую группу разработчиков. Компания имеет возможности, соответствующие ***2-му уровню зрелости*** согласно модели CMM.

Возможны следующие значения масштабных факторов:

- предсказуемость. Это новый проект для компании — значение Низкий (4);
- гибкость разработки. Заказчик требует некоторого согласования — значение Очень высокий (1);
- разрешение архитектуры/риска. Не выполняется анализ риска, как

следствие, малое разрешение риска — значение Очень низкий (5);

- связность группы. Новая группа, нет информации — значение Номинальный (3);
- зрелость процесса. Имеет место некоторое управление процессом — значение Номинальный (3).

Таблица 2.20. Оценка масштабных факторов

Масштабный фактор (W_i)	**Очень низкий 5**	**Низкий 4**
PREC	Полностью непредсказуемый проект	Главным образом, в значительной степени непредсказуемый
FLEX	Точный, строгий процесс разработки	Редкое расслабление в работе
RESL	Малое разрешение риска (20%)	Некоторое (40%)
TEAM	Очень трудное взаимодействие	Достаточно трудное взаимодействие
PREC	Полностью непредсказуемый проект	В значительной степени непредсказуемый
PMAT	Взвешенное среднее значение от количества ответов «Yes» на вопросник CMM Maturity	

Номинальный 3	**Высокий 2**	**Очень высокий 1**	**Сверхвысокий 0**
Отчасти непредсказуемый	Большей частью накомый	В значительной степени знакомый	Полностью знакомый
Некоторое расслабление в работе	Большей частью согласованный процесс	Некоторое согласование процесса	Заказчик определил только общие цели

Частое (60%)	Большей частью (75%)	Почти всегда (90%)	Полное (100%)
Среднее взаимодействие	Главным образом кооперативность	Высокая кооперативность	Безукоризненное взаимодействие
Отчасти непредсказуемый	Большей частью знакомый	В значительной степени знакомый	Полностью знакомый

Взвешенное среднее значение от количества ответов «Yes» на вопросник CMM Maturity

Сумма этих значений равна **16**, поэтому конечное значение степени B**= 1,17.**

Вернемся к основному уравнению модели раннего этапа проектирования.

Множитель поправки M_e зависит от набора формирователей затрат, перечисленных в табл. 2.21. Для каждого формирователя затрат определяется оценка (по **6**-балльной шкале), где **1** соответствует очень низкому значению, а **6** — сверхвысокому значению.

Таблица 2.21. Формирователи затрат для раннего этапа проектирования

Обозначение	Название
PERS	Возможности персонала (Personnel Capability)
RCPX	Надежность и сложность продукта (Product Reliability and Complexity)
RUSE	Требуемое повторное использование (Required Reuse)
PDIF	Трудность платформы (Platform Difficulty)
PREX	Опытность персонала (Personnel Experience)
FCIL	Средства поддержки (Facilities)
SCED	График (Schedule)

На основе оценки для каждого формирователя по таблице Боэма определяется множитель затрат EM_i Перемножение всех множителей затрат формирует множитель поправки:

$$M_e = \prod_{i=1}^{7} EM_i .$$

Слагаемое **ЗАТРАТЫ$_{auto}$** используется, если некоторый процент программного кода генерируется автоматически. Поскольку производительность такой работы значительно выше, чем при ручной разработке кода, требуемые затраты вычисляются отдельно, по следующей формуле:

ЗАТРАТЫ$_{auto}$=(KALOCx(AT/100))/ATPROD,

где:

KALOC — количество строк автоматически генерируемого кода (в тысячах строк);

AT — процент автоматически генерируемого кода (от всего кода системы);

ATPROD — производительность автоматической генерации кода.

Далее затраты на автоматическую генерацию добавляются к затратам, вычисленным для кода, разработанного вручную.

Модель этапа постархитектуры

Модель этапа постархитектуры используется в период, когда уже сформирована архитектура и выполняется дальнейшая разработка программного продукта.

Основное уравнение постархитектурной модели является развитием уравнения предыдущей модели и имеет следующий вид:

ЗАТРАТЫ = A x $K_{\sim req}$ x РАЗМЕР^B x M_p +ЗАТРАТЫ$_{auto}$ [чел.-мес],

где

-коэффициент $K_{\sim req}$ учитывает возможные изменения в требованиях;

-показатель ***B*** отражает нелинейную зависимость затрат от размера проекта (размер выражается в KLOC), вычисляется так же, как и в предыдущей модели;

-в размере проекта различают две составляющие — новый код и повторно используемый код;

-множитель поправки M_p зависит от 17 факторов затрат, характеризующих продукт, аппаратуру, персонал и проект.

Изменчивость требований приводит к повторной работе, требуемой для учета предлагаемых изменений, оценка их влияния выполняется по формуле

$$K_{\sim req} = 1 + (BRAK/100),$$

где **BRAK** — процент кода, отброшенного (модифицированного) из-за изменения требований.

Размер проекта и продукта определяют по выражению

$$РАЗМЕР = РАЗМЕР_{new} + РАЗМЕР_{reuse} [KLOC],$$

где

- $РАЗМЕР_{new}$ — размер нового (создаваемого) программного кода;
- $РАЗМЕР_{reuse}$ — размер повторно используемого программного кода.

Формула для расчета размера повторно используемого кода записывается следующим образом:

$$РАЗМЕР_{reuse} = KASLOC \times ((100 - AT)/ 100) \times (AA + SU + 0{,}4\, DM + 0{,}3\, CM + 0{,}3\, IM) /100,$$

где

- KASLOC- кол-во строк повторно используемого кода, который д.б. модифицирован (в тыс.строк);
- AT - процент автоматически генерируемого кода;
- DM - процент модифицируемых проектных моделей;

- CM - процент модифицируемого программного кода;
- IM - процент затрат на интеграцию, требуемых для подключения повторно используемого ПО;
- SU - фактор, основанный на стоимости понимания добавляемого ПО; изменяется от 50 (для сложного неструктурированного кода) до 10 (для хорошо написанного объектно-ориентированного кода);
- AA - фактор, который отражает стоимость решения о том, может ли ПО быть повторно используемым; зависит от размера требуемого тестирования и оценивания (величина изменяется от 0 до 8).

Правила выбора этих параметров приведены в руководстве по COCOMO II.

Для определения множителя поправки M_p основного уравнения используют **17** факторов затрат, которые могут быть разбиты на 4 категории. Перечислим факторы затрат, сгруппировав их по категориям.

Факторы продукта:

1) требуемая надежность ПО — RELY;

2) размер базы данных — DATA;

3) сложность продукта — CPLX;

4) требуемая повторная используемость — RUSE;

5) документирование требований жизненного цикла — DOCU.

Факторы платформы (виртуальной машины):

6) ограничения времени выполнения — TIME;

7) ограничения оперативной памяти — STOR;

8) изменчивость платформы — PVOL.

Факторы персонала:

9) возможности аналитика — ACAP;

10) возможности программиста — PCAP;

11) опыт работы с приложением — AEXP;

12) опыт работы с платформой — PEXP;

13) опыт работы с языком и утилитами — LTEX;

14) непрерывность персонала — PCON.

Факторы проекта:

15) использование программных утилит — TOOL;

16) мультисетевая разработка — SITE;

17) требуемый график разработки — SCED.

Для каждого фактора определяется оценка (по **6**-балльной шкале). На основе оценки для каждого фактора по таблице Боэма определяется множитель затрат EM_i. Перемножение всех множителей затрат дает множитель поправки пост-архитектурной модели:

$$M_p = \prod_{i=1}^{17} EM_i .$$

Значение M_p отражает реальные условия выполнения программного проекта и позволяет троекратно увеличить (уменьшить) начальную оценку затрат.

От оценки затрат легко перейти к стоимости проекта. Переход выполняют по формуле:

СТОИМОСТЬ = ЗАТРАТЫ x РАБ_ КОЭФ,

где среднее значение рабочего коэффициента составляет $15 000 за человеко-месяц.

После определения затрат и стоимости можно оценить длительность разработки. Модель COCOMO II содержит уравнение для ***оценки***

календарного времени TDEV, требуемого для выполнения проекта. Для моделей всех уровней справедливо:

$$\textbf{Длительность (TDEV)} = [3{,}0 \times (\textbf{ЗАТРАТЫ})^{(0{,}33+0{,}2(B-1{,}01))}] \times \textbf{SCEDPercentage}/100 \ [\textbf{мес}],$$

где

-*B* — ранее рассчитанный показатель степени;

-SCEDPercentage — процент увеличения (уменьшения) номинального графика.

Если нужно определить номинальный график, то принимается **SCEDPercentage =100** и правый сомножитель в уравнении обращается в единицу. Следует отметить, что COCOMO II ограничивает диапазон уплотнения/растягивания графика (**от 75 до 160%**). Причина проста — если планируемый график существенно отличается от номинального, это означает внесение в проект высокого риска.

пример. Положим, что затраты на проект равны 20 человеко-месяцев. Примем, что все масштабные факторы номинальны (имеют значения 3), поэтому, в соответствии с табл. 2.20, показатель степени 5=1,16. Отсюда следует, что номинальная длительность проекта равна

$$\text{TDEV} = 3{,}0 \times (20)^{0{,}36} = 8{,}8 \text{ мес.}$$

Очень часто увеличение количества разработчиков приводит к возрастанию затрат. причина:

- увеличивается время на взаимодействие и обучение сотрудников, согласование совместных решений;
- возрастает время на определение интерфейсов между частями программной системы.

Удвоение разработчиков не приводит к двукратному сокращению длительности проекта. Модель COCOMO II явно утверждает, что

длительность проекта является функцией требуемых затрат, прямой зависимости от количества сотрудников нет. Другими словами, она устраняет миф нерадивых менеджеров в том, что добавление людей поможет ликвидировать отставание в проекте.

COCOMO II предостерегает от определения потребного количества сотрудников путем деления затрат на длительность проекта. Такой упрощенный подход часто приводит к срыву работ. Реальная картина имеет другой характер. Количество людей, требуемых на этапе планирования и формирования требований, достаточно мало. На этапах проектирования и кодирования потребность в увеличении команды возрастает, после окончания кодирования и тестирования численность необходимых сотрудников достигает минимума.

Анализ чувствительности программного проекта

Рассмотрим возможности этой модели в задачах анализа чувствительности — чувствительности программного проекта к изменению условий разработки.

Будем считать, что корпорация «СверхМобильныеСвязи» заказала разработку ПО для встроенной космической системы обработки сообщений. Ожидаемый размер ПО — 10 KLOC, используется серийный микропроцессор. Примем, что масштабные факторы имеют номинальные значения (показатель степени $B = 1{,}16$) и что автоматическая генерация кода не предусматривается. К проведению разработки привлекаются главный аналитик и главный программист высокой квалификации, поэтому средняя зарплата в команде составит $ 6000 в месяц. Команда имеет годовой опыт работы с этой проблемной областью и полгода работает с нужной аппаратной платформой.

В терминах COCOMO II проблемную область (область применения продукта) классифицируют как «операции с приборами» со следующим описанием: встроенная система для высокоскоростного

мультиприоритетного обслуживания удаленных линий связи, обеспечивающая возможности диагностики.

Оценку пост-архитектурных факторов затрат для проекта сведем в табл. 2.27.

Из таблицы следует, что увеличение затрат в 1,3 раза из-за очень высокой сложности продукта уравновешивается их уменьшением вследствие высокой квалификации аналитика и программиста, а также активного использования программных утилит.

Таблица 2.27. Оценка пост-архитектурных факторов затрат

Фактор	**Описание**	**Оценка**	**Множит**
RELY	Требуемая надежность ПО	Номинал.	1
DATA	Размер базы данных — 20 Кбайт	Низкая	0,93
CPLX	Сложность продукта	Очень высок.	1,3
RUSE	Требуемая повторная используемость	Номинал.	1
DOCU	Документирование жизненного цикла	Номинал.	1
TIME	Ограничения времени выполнения (70%)	Высокая	1,11
STOR	Ограничения оперативной памяти (45 из 64 Кбайт, 70%)	Высокая	1,06
PVOL	Изменчивость платформы (каждые 6 месяцев)	Номинал.	1
ACAP	Возможности аналитика (75%)	Высокая	0,83
PCAP	Возможности программиста (75%)	Высокая	0,87
AEXP	Опыт работы с приложением (1 год)	Номинал.	1
PEXP	Опыт работы с платформой (6 месяцев)	Низкая	1,12
LTEX	Опыт работы с языком и утилитами (1 год)	Номинал.	1
PCON	Непрерывность персонала (1 2% в год)	Номинал.	1
TOOL	Активное использование программных утилит	Высокая	0,86
SITE	Мультисетевая разработка (телефоны)	Низкая	1,1
SCED	Требуемый график разработки	Номинал.	1
Множитель поправки M_p			1,088

Рассчитаем затраты и стоимость проекта:

$$ЗАТРАТЫ = АхРАЗМЕР^{B}xM_{p}=2,5(10)^{1,16}x1,088=36x1,088= 39[чел.-мес],$$

$$СТОИМОСТЬ = ЗАТРАТЫ \text{ x } \$6000 = \$234\ 000.$$

Таковы стартовые условия программного проекта. А теперь обсудим несколько сценариев возможного развития событий.

Сценарий понижения зарплаты

Положим, что заказчик решил сэкономить на зарплате разработчиков. Рычаг — понижение квалификации аналитика и программиста. Соответственно, зарплата сотрудников снижается до **$5000**. Оценки их возможностей становятся номинальными, а соответствующие множители затрат принимают единичные значения:

$$\boldsymbol{EM_{ACAP}=EM_{PCAP}=1.}$$

Следствием такого решения является возрастание множителя поправки $\mathbf{M_p= 1,507,}$ а также затрат и стоимости:

$$ЗАТРАТЫ = 36x\ 1,507 = 54\ [чел.-мес],$$

$$СТОИМОСТЬ = ЗАТРАТЫ \text{ x}\$5000 = \$270\ 000,$$

$$Проигрыш\_ в\_стоимости = \$36\ 000.$$

Сценарий наращивания памяти

Положим, что разработчик предложил нарастить память —

купить за $1000 чип ОЗУ емкостью 96 Кбайт (вместо 64 Кбайт).

Это меняет ограничение памяти (используется не 70%, а 47%), после чего фактор STOR снижается до номинального:

$$\boldsymbol{EM_{STOR}=1 \text{-> } M_p =1,026,}$$

$$ЗАТРАТЫ = 36x1,026 = 37\ [чел.-мес],$$

$$СТОИМОСТЬ = ЗАТРАТЫ \text{ x } \$6000 = \$222000,$$

$$Выигрыш\_ в\_стоимости = \$\ 12\ 000.$$

Сценарий использования нового микропроцессора

Положим, что заказчик предложил использовать новый, более дешевый МП (дешевле на $1000).

Опыт работы с его языком и утилитами понижается от номинального до очень низкого и $EM_{LTEX} = 1,22$, а разработанные для него утилиты (компиляторы, ассемблеры и отладчики) примитивны и ненадежны (в результате фактор **TOOL** понижается от высокого до очень низкого и $EM_{TOOL} = 1,24$):

$$M_p = (1,088 / 0,86) \times 1,22 \times 1,24 = 1,914,$$

ЗАТРАТЫ = 36х1,914 = 69[чел.-мес],

СТОИМОСТЬ = ЗАТРАТЫ х $6000 = $414000,

Проигрыш_в_стоимости = **$180000.**

Сценарий уменьшения средств на завершение проекта

Положим, что к разработке принят сценарий с наращиванием памяти:

ЗАТРАТЫ = 36 х 1,026 = 37 [чел.-мес],

СТОИМОСТЬ = ЗАТРАТЫ х $6000 = $222000.

Кроме того, предположим, что завершился этап анализа требований, на который было израсходовано $22 000 (10% от бюджета). После этого на завершение проекта осталось $200 000.

Допустим, что в этот момент «коварный» заказчик сообщает об отсутствии у него достаточных денежных средств и о предоставлении на завершение разработки только $170 000 (**15%-ное** уменьшение оплаты).

Для решения этой проблемы надо установить возможные изменения факторов затрат, позволяющие уменьшить оценку затрат на 15%.

Первое решение: уменьшение размера продукта (за счет исключения некоторых функций). Нам надо определить размер минимизированного продукта.

Будем исходить из того, что затраты должны уменьшиться с 37 до 31,45 чел.-мес. Решим уравнение:

$$2{,}5\ (\text{НовыйРазмер})^{1{,}16} = 31{,}45\ [\text{чел.-мес}].$$

Очевидно, что

$$(\text{НовыйРазмер})^{1{,}16} = 12{,}58,$$

$$(\text{НовыйРазмер})^{1{,}16} = 12{,}58^{1/1{,}16} = 8{,}872\ [\text{KLOC}].$$

Другие решения:

- уменьшить требуемую надежность с номинальной до низкой.

Это сокращает стоимость проекта на **12%** (EM_{RELY} изменяется с 1 до 0,88). Такое решение приведет к увеличению затрат и трудностей при применении и сопровождении;

- повысить требования к квалификации аналитиков и программистов (с высоких до очень высоких).

При этом стоимость проекта уменьшается на **15-19%.** Благодаря программисту стоимость может уменьшиться на (1 - 0,74/0,87) x 100% = 15%.

Благодаря аналитику стоимость может понизиться на (1 - 0,67/0,83) x 100% = 19%. Основная трудность — поиск специалистов такого класса (готовых работать за те же деньги);

- повысить требования к опыту работы с приложением (с номинальных до очень высоких) или требования к опыту работы с платформой (с низких до высоких). Повышение опыта работы с приложением сокращает стоимость проекта на (1- 0,81) x 100% = 19%;

повышение опыта работы с платформой сокращает стоимость проекта на (1 - 0,88/1,12) x 100% = 21,4%. Основная трудность — поиск экспертов (специалистов такого класса);

- повысить уровень мультисетевой разработки с низкого до высокого. При этом стоимость проекта уменьшается на (1 - 0,92/1,1) x 100% = 16,4%;
- ослабить требования к режиму работы в реальном времени. Предположим, что 70%-ное ограничение по времени выполнения связано с желанием заказчика обеспечить обработку одного сообщения за 2 мс. Если же заказчик согласится на увеличение среднего времени обработки с 2 до 3 мс, то ограничение по времени станет равно (2 мс/3 мс) x 70% = 47%, в результате чего фактор TIME уменьшится с высокого до номинального, что приведет к экономии затрат на (1- 1/1,11) x 100%= 10%;
- учет других факторов затрат не имеет смысла. Некоторые факторы (размер базы данных, ограничения оперативной памяти, требуемый график разработки) уже имеют минимальные значения, для других трудно ожидать быстрого улучшения (использование программных утилит, опыт работы с языком и утилитами), третьи имеют оптимальные значения (требуемая повторная используемость, документирование требований жизненного цикла). На некоторые разработчик почти не может повлиять (сложность продукта, изменчивость платформы). Наконец, житейские неожиданности едва ли позволят улучшить принятое значение фактора «непрерывность персонала».

Задание к лабораторной работе

Для своего программного проекта (для которого написали техническое задание) построить модель COCOMO II (подмодель см. №варианта) и оценить чувствительность в зависимости от сценария (см. .№варианта)

Порядок выполнения работы

1. Построить модель COCOMO II (подмодель см.№варианта)
2. Оценить чувствительность в зависимости от сценария

3. Варианты заданий:

№вар	**Модель COCOMO II**	**Сценарий**
1	модель раннего этапа проектирования;	понижения зарплаты
2	модель этапа пост-архитектуры.	наращивания памяти
3	модель раннего этапа проектирования;	использование нового микропроцессора
4	модель этапа пост-архитектуры.	уменьшение средств на завершение проекта
5	модель раннего этапа проектирования;	понижения зарплаты
6	модель этапа пост-архитектуры.	наращивания памяти
7	модель раннего этапа проектирования;	использование нового микропроцессора
8	модель этапа пост-архитектуры.	уменьшение средств на завершение проекта
9	модель раннего этапа проектирования;	понижения зарплаты
10	модель этапа пост-архитектуры.	наращивания памяти
11	модель раннего этапа проектирования;	использование нового микропроцессора
12	модель этапа пост-архитектуры.	уменьшение средств на завершение проекта
13	модель раннего этапа проектирования;	понижения зарплаты
14	модель этапа пост-архитектуры.	наращивания памяти
15	модель раннего этапа проектирования;	использование нового микропроцессора
16	модель этапа пост-	уменьшение средств на завершение

	архитектуры.	проекта
17	модель раннего этапа проектирования;	понижения зарплаты
18	модель этапа пост-архитектуры.	наращивания памяти
19	модель раннего этапа проектирования;	использование нового микропроцессора
20	модель этапа пост-архитектуры.	уменьшение средств на завершение проекта
21	модель раннего этапа проектирования;	понижения зарплаты
22	модель этапа пост-архитектуры.	наращивания памяти
23	модель раннего этапа проектирования;	использование нового микропроцессора
24	модель этапа пост-архитектуры.	уменьшение средств на завершение проекта

Требование к отчету по лабораторной работе:

1. Свой программный проект (для которого написали техническое задание)
2. Краткие теоретические сведения.
3. Расчеты.
4. Выводы.

Контрольные вопросы

1. В чем состоит назначение модели композиции? На каких оценках она базируется?
2. В чем состоит назначение модели раннего этапа проектирования?
3. Охарактеризуйте основное уравнение модели раннего этапа проектирования.

4. Охарактеризуйте масштабные факторы модели COCOMO II.
5. Как оцениваются масштабные факторы?
6. В чем состоит назначение модели этапа пост-архитектуры COCOMO II?
7. Чем отличается основное уравнение модели этапа пост-архитектуры от аналогичного уравнения модели раннего этапа проектирования?
8. Что такое факторы затрат модели этапа пост-архитектуры и как они вычисляются?
9. Как определяется длительность разработки в модели COCOMO II?
10. Что такое анализ чувствительности программного проекта?
11. Как применить модель COCOMO II к анализу чувствительности?

Лабораторная работа №5

Тема: Метрики объектно-ориентированных программных систем

Цель: Оценить качество визуальных моделей наиболее известными объектно-ориентированными метриками.

Методические указания к лабораторной работе

Метрические особенности объектно-ориентированных программных систем

Объектно-ориентированные метрики вводятся с целью:

- улучшить понимание качества продукта;
- оценить эффективность процесса конструирования;
- улучшить качество работы на этапе проектирования.

Все эти цели важны, но для программного инженера главная цель — повышение качества продукта. Возникает вопрос — как измерить качество объектно-ориентированной системы?

Для любого инженерного продукта метрики должны ориентироваться на его уникальные характеристики. Например, для электропоезда вряд ли полезна метрика «расход угля на километр пробега». С точки зрения метрик выделяют пять характеристик объектно-ориентированных систем: локализацию, инкапсуляцию, информационную закрытость, наследование и способы абстрагирования объектов. Эти характеристики оказывают максимальное влияние на объектно-ориентированные метрики.

Локализация

Локализация фиксирует способ группировки информации в программе. В классических методах, где используется функциональная декомпозиция, информация локализуется вокруг функций. Функции в них реализуются как процедурные модули. В методах, управляемых данными, информация

группируется вокруг структур данных. В объектно-ориентированной среде информация группируется внутри классов или объектов (инкапсуляцией как данных, так и процессов).

Поскольку в классических методах основной механизм локализации — функция, программные метрики ориентированы на внутреннюю структуру или сложность функций (длина модуля, связность, цикломатическая сложность) или на способ, которым функции связываются друг с другом (сцепление модулей).

Так как в объектно-ориентированной системе базовым элементом является класс, то локализация здесь основывается на объектах. Поэтому метрики должны применяться к классу (объекту) как к комплексной сущности. Кроме того, между операциями (функциями) и классами могут быть отношения не только «один-к-одному». Поэтому метрики, отображающие способы взаимодействия классов, должны быть приспособлены к отношениям «один-ко-многим», «многие-ко-многим».

Инкапсуляция

Вспомним, что инкапсуляция — упаковка (связывание) совокупности элементов. Для классических ПС примерами низкоуровневой инкапсуляции являются записи и массивы. Механизмом инкапсуляции среднего уровня являются подпрограммы (процедуры, функции).

В объектно-ориентированных системах инкапсулируются обязанности класса, представляемые его свойствами (а для агрегатов — и свойствами других классов), операциями и состояниями.

Для метрик учет инкапсуляции приводит к смещению фокуса измерений с одного модуля на группу свойств и обрабатывающих модулей (операций). Кроме того, инкапсуляция переводит измерения на более высокий уровень абстракции (пример — метрика «количество операций на класс»). Напротив, классические метрики ориентированы на низкий уровень — количество

булевых условий (цикломатическая сложность) и количество строк программы.

Информационная закрытость

Информационная закрытость делает невидимыми операционные детали программного компонента. Другим компонентам доступна только необходимая информация.

Качественные объектно-ориентированные системы поддерживают высокий уровень информационной закрытости. Таким образом, метрики, измеряющие степень достигнутой закрытости, тем самым отображают качество объектно-ориентированного проекта.

Наследование

Наследование — механизм, обеспечивающий тиражирование обязанностей одного класса в другие классы. Наследование распространяется через все уровни иерархии классов. Стандартные ПС не поддерживают эту характеристику.

Поскольку наследование — основная характеристика объектно-ориентированных систем, на ней фокусируются многие объектно-ориентированные метрики (количество детей — потомков класса, количество родителей, высота класса в иерархии наследования).

Абстракция

Абстракция — это механизм, который позволяет проектировщику выделять главное в программном компоненте (как свойства, так и операции) без учета второстепенных деталей. По мере перемещения на более высокие уровни абстракции мы игнорируем все большее количество деталей, обеспечивая все более общее представление понятия или элемента. По мере перемещения на более низкие уровни абстракции мы вводим все большее

количество деталей, обеспечивая более удачное представление понятия или элемента.

Класс — это абстракция, которая может быть представлена на различных уровнях детализации и различными способами (например, как список операций, последовательность состояний, последовательности взаимодействий). Поэтому объектно-ориентированные метрики должны представлять абстракции в терминах измерений класса. Примеры: количество экземпляров класса в приложении, количество родовых классов на приложение, отношение количества родовых к количеству неродовых классов.

Эволюция мер связи для объектно-ориентированных программных систем

В разделах «Связность модуля» и «Сцепление модулей» главы 4 было показано, что классической мерой сложности внутренних связей модуля является связность, а классической мерой сложности внешних связей — сцепление. Рассмотрим развитие этих мер применительно к объектно-ориентированным системам.

Связность объектов

В классическом методе Л. Констентайна и Э. Йордана определены семь типов связности.

1. ***Связность по совпадению.*** В модуле отсутствуют явно выраженные внутренние связи.
2. ***Логическая связность.*** Части модуля объединены по принципу функционального подобия.
3. ***Временная связность.*** Части модуля не связаны, но необходимы в один и тот же период работы системы.
4. ***Процедурная связность.*** Части модуля связаны порядком выполняемых ими действий, реализующих некоторый сценарий поведения.

5. ***Коммуникативная связность.*** Части модуля связаны по данным (работают с одной и той же структурой данных).
6. ***Информационная (последовательная) связность.*** Выходные данные одной части используются как входные данные в другой части модуля.
7. ***Функциональная связность.*** Части модуля вместе реализуют одну функцию.

Этот метод функционален по своей природе, поэтому наибольшей связностью здесь объявлена функциональная связность. Вместе с тем одним из принципиальных преимуществ объектно-ориентированного подхода является естественная связанность объектов.

Максимально связанным является объект, в котором представляется единая сущность и в который включены все операции над этой сущностью. Например, максимально связанным является объект, представляющий таблицу символов компилятора, если в него включены все функции, такие как «Добавить символ», «Поиск в таблице» и т. д.

Следовательно, восьмой тип связности можно определить так:

8. ***Объектная связность.*** Каждая операция обеспечивает функциональность, которая предусматривает, что все свойства объекта будут модифицироваться, отображаться и использоваться как базис для предоставления услуг.

Высокая связность — желательная характеристика, так как она означает, что объект представляет единую часть в проблемной области, существует в едином пространстве. При изменении системы все действия над частью инкапсулируются в едином компоненте. Поэтому для производства изменения нет нужды модифицировать много компонентов.

Если функциональность в объектно-ориентированной системе обеспечивается наследованием от суперклассов, то связность объекта, который наследует свойства и операции, уменьшается. В этом случае нельзя рассматривать объект как отдельный модуль — должны учитываться все его суперклассы. Системные средства просмотра содействуют такому учету.

Однако понимание элемента, который наследует свойства от нескольких суперклассов, резко усложняется.

Обсудим конкретные метрики для вычисления связности классов.

Метрики связности по данным

Л. Отт и Б. Мехра разработали модель секционирования класса [55]. Секционирование основывается на экземплярных переменных класса. Для каждого метода класса получают ряд секций, а затем производят объединение всех секций класса. Измерение связности основывается на количестве лексем данных (data tokens), которые появляются в нескольких секциях и «склеивают» секции в модуль. Под лексемами данных здесь понимают определения констант и переменных или ссылки на константы и переменные.

Базовым понятием методики является секция данных. Она составляется для каждого выходного параметра метода. *Секция данных* — это последовательность лексем данных в операторах, которые требуются для вычисления этого параметра.

Например, на рис. 14.1 представлен программный текст метода SumAndProduct. Все лексемы, входящие в секцию переменной SumN, выделены рамками. Сама секция для SumN записывается как следующая последовательность лексем:

$N_1 \bullet SumN_1 \bullet I_1 \bullet SumN_2 \bullet O_1 \bullet I_2 \bullet 1_2 \bullet N_2 \bullet SumN_3\ SumN_4 \bullet I_3.$

```
procedure SumAndProduct
( N : integer;
 var SumN, ProdN : integer );
 var
    I  : integer;
begin
    SumN  := 0 ;
    ProdN := 1;
    for I := 1 to N do begin
                          SumN := SumN + I ;
                          ProdN := ProdN * I
                       end
end;
```

Рис. 14.1. Секция данных для переменной SumN

Заметим, что индекс в «1_2» указывает на второе вхождение лексемы «1» в текст метода. Аналогичным образом определяется секция для переменной ProdN:

$N_1 \bullet ProdN_1 \bullet I_1 \bullet ProdN_2 \bullet 1_1 \bullet I_2 \bullet 1_2 \bullet N_2 \bullet ProdN_3 \bullet ProdN_4 \bullet I_4$

Для определения отношений между секциями данных можно показать профиль секций данных в методе. Для нашего примера профиль секций данных приведен в табл. 14.1.

Таблица 14.1. Профиль секций данных для метода SumAndProduct

SumN	ProdN	Оператор
		procedure SumAndProduct
1	1	(Niinteger;
1	1	varSumN, ProdNiinteger)
		var
1	1	l:integer;
		begin
2		SumN:=0
	2	ProdN:=1
3	3	for l:=1 to N do begin
3		SumN:=SumN+l
	3	ProdN:=ProdN*l
		end
		end;

Видно, что в столбце переменной для каждой секции указывается количество лексем из *i*-й строки метода, которые включаются в секцию.

Еще одно базовое понятие методики — секционированная абстракция. *Секционированная абстракция* — это объединение всех секций данных метода. Например, секционированная абстракция метода SumAndProduct имеет вид

$$SA(SumAndProduct) = \{N_1 \cdot SumN_1 \cdot I_1 \cdot SumN_2 \cdot 0_1 \cdot I_2 \cdot I_2 \cdot N_2 \cdot SumN_3 \cdot SumN_4 \cdot I_3,$$
$$N_1 \cdot ProdN_1 \cdot I_1 \cdot ProdN_2 \cdot I_1 \cdot I_2 \cdot I_2 \cdot N_2 \cdot ProdN_3 \cdot ProdN_4 \cdot I_4\}.$$

Введем главные определения.

Секционированной абстракцией класса (Class Slice Abstraction) CSA(C) называют объединение секций всех экземплярных переменных класса. Полный набор секций составляют путем обработки всех методов класса.

Склеенными лексемами называют те лексемы данных, которые являются элементами более чем одной секции данных.

Сильно склеенными лексемами называют те склеенные лексемы, которые являются элементами всех секций данных.

Сильная связность по данным (StrongData Cohesion) — это метрика, основанная на количестве лексем данных, входящих во все секции данных для класса. Иначе говоря, сильная связность по данным учитывает количество сильно склеенных лексем в классе *C,* она вычисляется по формуле:

$$\mathrm{SDC}(C) = \frac{|SG(\mathrm{CSA}(C))|}{|\text{лексемы}(C)|},$$

где *SG*(CSA(*C*)) — объединение сильно склеенных лексем каждого из методов класса *C,* лексемы*(C)* — множество всех лексем данных класса *C.*

Таким образом, класс без сильно склеенных лексем имеет нулевую сильную связность по данным.

Слабая связность по данным (Weak Data Cohesion) — метрика, которая оценивает связность, базируясь на склеенных лексемах. Склеенные лексемы не требуют связывания всех секций данных, поэтому данная метрика определяет более слабый тип связности. Слабая связность по данным вычисляется по формуле:

$$\mathrm{WDC}(C) = \frac{|G(\mathrm{CSA}(C))|}{|\text{лексемы}(C)|},$$

где *G*(CSA(C)) — объединение склеенных лексем каждого из методов класса. Класс без склеенных лексем не имеет слабой связности по данным. Наиболее точной метрикой связности между секциями данных является *клейкость данных (Data Adhesiveness).* Клейкость данных определяется как отношение

суммы из количеств секций, содержащих каждую склеенную лексему, к произведению количества лексем данных в классе на количество секций данных. Метрика вычисляется по формуле:

$$\mathrm{DA}(C) = \frac{\sum d \in G(\mathrm{CSA}(C)"d \in \text{Секции}}{|\text{лексемы}(C)| \times |\mathrm{CSA}(C)|}.$$

Приведем пример. Применим метрики к классу, профиль секций которого показан в табл. 14.2.

Таблица 14.2. Профиль секций данных для класса Stack

array top size	**Класс Stack**
	class Stack {int *array, top, size;
	public:
	Stack (int *s*) {
2 2	size=s;
2 2	array=new int [size];
2	top=0;}
	int IsEmpty () {
2	return top==0};
	int Size (){
2	return size};
	intVtop(){
3 3	return array [top-1]; }
	void Push (int item) {
2 2 2	if (top= =size)
	printf ("Empty stack. \n");
	else
3 3 3	array [top++]=item;}
	int Pop () {
1	if (IsEmpty ())
	printf ("Full stack. \n");
	else
1	--top;}

	};

Очевидно, что CSA(Stack) включает три секции с 19 лексемами, имеет 5 сильно склеенных лексем и 12 склеенных лексем.

Расчеты по рассмотренным метрикам дают следующие значения:

$$SDC(CSA(Stack)) = 5/19 = 0,26$$

$$WDC(CSA(Stack)) = 12/19 = 0,63$$

$$DA(CSA(Stack)) =(7*2 + 5*3)/(19*3) = 0,51$$

Метрики связности по методам

Д. Биемен и Б. Кенг предложили метрики связности класса, которые основаны на прямых и косвенных соединениях между парами методов [15]. Если существуют общие экземплярные переменные (одна или несколько), используемые в паре методов, то говорят, что эти методы соединены прямо. Пара методов может быть соединена косвенно, через другие прямо соединенные методы.

На рис. 14.2 представлены отношения между элементами класса Stack. Прямоугольниками обозначены методы класса, а овалами — экземплярные переменные. Связи показывают отношения использования между методами и переменными.

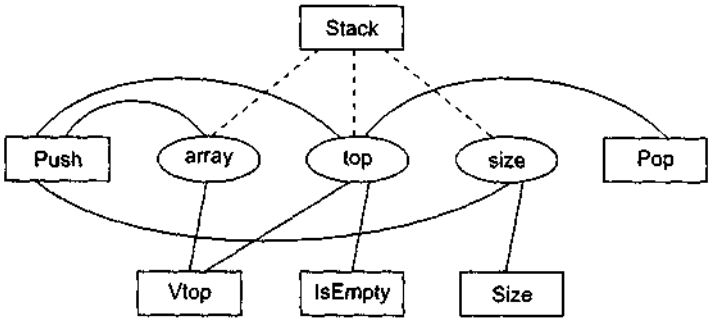

Рис. 14. 2. Отношения между элементами класса Stack

Из рисунка видно, что экземплярная переменная top используется методами Stack, Push, Pop, Vtop и IsEmpty. Таким образом, все эти методы попарно прямо соединены. Напротив, методы Size и Pop соединены

косвенно: Size соединен прямо с Push, который, в свою очередь, прямо соединен с Pop. Метод Stack является конструктором класса, то есть функцией инициализации. Обычно конструктору доступны все экземплярные переменные класса, он использует эти переменные совместно со всеми другими методами. Следовательно, конструкторы создают соединения и между такими методами, которые никак не связаны друг с другом. Поэтому ни конструкторы, ни деструкторы здесь не учитываются. Связи между конструктором и экземплярными переменными на рис. 14.2 показаны пунктирными линиями.

Для формализации модели вводятся понятия абстрактного метода и абстрактного класса.

Абстрактный метод AM(M) — это представление реального метода *M* в виде множества экземплярных переменных, которые прямо или косвенно используются методом.

Экземплярная переменная прямо используется методом *M,* если она появляется в методе как лексема данных. Экземплярная переменная может быть определена в том же классе, что и *M,* или же в родительском классе этого класса. Множество экземплярных переменных, прямо используемых методом *M,* обозначим как *DU(M).*

Экземплярная переменная косвенно используется методом *M,* если: 1) экземплярная переменная прямо используется другим методом *M'*, который вызывается (прямо или косвенно) из метода *M;* 2) экземплярная переменная, прямо используемая методом *M',* находится в том же объекте, что и *M.*
Множество экземплярных переменных, косвенно используемых методом *M,* обозначим как *IU(M).*

Количественно абстрактный метод формируется по выражению:

$$\text{AM}\ (M) = \text{DU}\ (M) \cup \text{IU}\ (M).$$

Абстрактный класс AC(C) — это представление реального класса *C* в виде совокупности абстрактных методов, причем каждый абстрактный метод

соответствует видимому методу класса *C*. Количественно абстрактный класс формируется по выражению:

$$AC\ (C) = [[AM\ (M) \mid M \in V\ (C)]],$$

где *V(C)* — множество всех видимых методов в классе *C* и в классах — предках для *C*.

Отметим, что AM-представления различных методов могут совпадать, поэтому в AC могут быть дублированные элементы. В силу этого AC записывается в форме мультимножества (двойные квадратные скобки рассматриваются как его обозначение).

Локальный абстрактный класс LAC(C) — это совокупность абстрактных методов, где каждый абстрактный метод соответствует видимому методу, определенному только внутри класса *C*. Количественно абстрактный класс формируется по выражению:

$$LAC(C)=[[AM(M)|M \in LV(C)]],$$

где LV*(C)* — множество всех видимых методов, определенных в классе *C*.

Абстрактный класс для стека, приведенного в табл. 14.2, имеет вид:

AC (Stack) = [[{top}, {size}, {array, top}, {array, top, size}, {pop}]].

Поскольку класс Stack не имеет суперкласса, то справедливо:

AC (Stack) = LAC (Stack)

Пусть NP*(C)* — общее количество пар абстрактных методов в AC*(C)*. NP определяет максимально возможное количество прямых или косвенных соединений в классе. Если в классе *C* имеются *N* методов, тогда NP*(C)* = *N*(N*- l)/2. Обозначим:

- NDC*(C)* — количество прямых соединений AC(Q;
- NIC*(C)* — количество косвенных соединений в AC(C).

Тогда метрики связности класса можно представить в следующем виде:

- *сильная связность класса (Tight Class Cohesion (TCC))* определяется относительным количеством прямо соединенных методов:

$$TCC\ (C) = NDC\ (C) / NP\ (C);$$

- *слабая связность класса (Loose Class Cohesion (LCC))* определяется

относительным количеством прямо или косвенно соединенных методов:

$$LCC\ (C) = (NDC\ (C) + NIC\ (C)) / NP\ (C).$$

Очевидно, что всегда справедливо следующее неравенство:

$$LCC(C) >= TCC(C).$$

Для класса Stack метрики связности имеют следующие значения:

$$TCC(Stack)=7/10=0,7$$

$$LCC(Stack)=10/10=l$$

Метрика TCC показывает, что 70% видимых методов класса Stack соединены прямо, а метрика LCC показывает, что все видимые методы класса Stack соединены прямо или косвенно.

Метрики TCC и LCC индицируют степень связанности между видимыми методами класса. Видимые методы либо определены в классе, либо унаследованы им. Конечно, очень полезны метрики связности для видимых методов, которые определены только внутри класса — ведь здесь исключается влияние связности суперкласса. Очевидно, что метрики локальной связности класса определяются на основе локального абстрактного класса. Отметим, что для локальной связности экземплярные переменные и вызываемые методы могут включать унаследованные переменные.

Сцепление объектов

В классическом методе Л. Констентайна и Э. Йордана определены шесть типов сцепления, которые ориентированы на процедурное проектирование [77].

Принципиальное преимущество объектно-ориентированного проектирования в том, что природа объектов приводит к созданию слабо сцепленных систем. Фундаментальное свойство объектно-ориентированного проектирования заключается в скрытости содержания объекта. Как правило, содержание объекта невидимо внешним элементам. Степень автономности

объекта достаточно высока. Любой объект может быть замещен другим объектом с таким же интерфейсом.

Тем не менее наследование в объектно-ориентированных системах приводит к другой форме сцепления. Объекты, которые наследуют свойства и операции, сцеплены с их суперклассами. Изменения в суперклассах должны проводиться осторожно, так как эти изменения распространяются во все классы, которые наследуют их характеристики.

Таким образом, сами по себе объектно-ориентированные механизмы не гарантируют минимального сцепления. Конечно, классы — мощное средство абстракции данных. Их введение уменьшило поток данных между модулями и, следовательно, снизило общее сцепление внутри системы. Однако количество типов зависимостей между модулями выросло. Появились отношения наследования, делегирования, реализации и т. д. Более разнообразным стал состав модулей в системе (классы, объекты, свободные функции и процедуры, пакеты). Отсюда вывод: необходимость измерения и регулирования сцепления в объектно-ориентированных системах обострилась.

Рассмотрим объектно-ориентированные метрики сцепления, предложенные М. Хитцем и Б. Монтазери [38].

Зависимость изменения между классами

Зависимость изменения между классами CDBC (Change Dependency Between Classes) определяет потенциальный объем изменений, необходимых после модификации класса-сервера SC (server class) на этапе сопровождения. До тех пор, пока реальное количество необходимых изменений класса-клиента CC (client class) неизвестно, CDBC указывает количество методов, на которые влияет изменение SC.

CDBC зависит от:

- области видимости изменяемого класса-сервера внутри класса-клиента (определяется типом отношения между CS и CC);

- вида доступа CC к CS (интерфейсный доступ или доступ реализации).

Возможные типы отношений приведены в табл. 14.3, где *n* — количество методов класса CC, ъ— количество методов CC, потенциально затрагиваемых изменением.

Таблица 14.3. Вклад отношений между клиентом и сервером в зависимость изменения

Тип отношения	ъ
SC не используется классом CC	0
SC — класс экземплярной переменной в классе CC	n
Локальные переменные типа SC используются внутри /-методов класса CC	j
SC является суперклассом CC	n
SC является типом параметра для/-методов класса CC	j
CC имеет доступ к глобальной переменной класса SC	n

Конечно, здесь предполагается, что те элементы класса-сервера SC, которые доступны классу-клиенту CC, являются предметом изменений. Авторы исходят из следующей точки зрения: если класс SC является «зрелой» абстракцией, то предполагается, что его интерфейс более стабилен, чем его реализация. Таким образом, многие изменения в реализации SC могут выполняться без влияния на его интерфейс. Поэтому вводится фактор стабильности интерфейса для класса-сервера, он обозначается как к ($0 < k < 1$). Вклад доступа к интерфейсу в зависимость изменения можно учесть умножением на (1 - *k).*

Метрика для вычисления степени CDBC имеет вид:

$$A = \sum_{\substack{Доступ\ i \\ к\ реализации}} \alpha_i + (1-k) \times \sum_{\substack{Доступ\ i \\ к\ интерфейсу}} \alpha_i \quad ;$$

CDBC(CC, SC) = min(*n, A).*

Пути минимизации CDBC:

1) ограничение доступа к интерфейсу класса-сервера;

2) ограничение видимости классов-серверов (спецификаторами доступа public, protected, private).

Локальность данных

Локальность данных LD (Locality of Data) — метрика, отражающая качество абстракции, реализуемой классом. Чем выше локальность данных, тем выше самодостаточность класса. Эта характеристика оказывает сильное влияние на такие внешние характеристики, как повторная используемость и тестируемость класса.

Метрика LD представляется как отношение количества локальных данных в классе к общему количеству данных, используемых этим классом. Будем использовать терминологию языка C++. Обозначим как $M_i(1 \le i \le n)$ методы класса. В их число не будем включать методы чтения/записи экземплярных переменных. Тогда формулу для вычисления локальности данных можно записать в виде:

$$LD = \frac{\sum_{i=1}^{a} |L_i|}{\sum_{i=1}^{a} |T_i|},$$

где:

- $L_i(1 \le i \le n)$ — множество локальных переменных, к которым имеют доступ методы M_i (прямо или с помощью методов чтения/записи). Такими переменными являются: непубличные экземплярныё переменные класса; унаследованные защищенные экземплярныё переменные их суперклассов; статические переменные, локально определенные в M_i ;
- $T_i(1 \le i \le n)$ — множество всех переменных, используемых в M_i, кроме динамических локальных переменных, определенных в M_i.

Для обеспечения надежности оценки здесь исключены все вспомогательные переменные, определенные в M_i , — они не играют важной роли в проектировании.

Защищенная экземплярная переменная, которая унаследована классом *C,* является локальной переменной для его экземпляра (и следовательно, является элементом L_i), даже если она не объявлена в классе *C.* Использование такой переменной методами класса не вредит локальности данных, однако нежелательно, если мы заинтересованы уменьшить значение CDBC.

Набор метрик Чидамбера и Кемерера

В 1994 году С. Чидамбер и К. Кемерер (Chidamber и Кетегег) предложили шесть проектных метрик, ориентированных на классы [24]. Класс — фундаментальный элемент объектно-ориентированной (ОО) системы. Поэтому измерения и метрики для отдельного класса, иерархии классов и сотрудничества классов бесценны для программного инженера, который должен оценить качество проекта.

Набор Чидамбера-Кемерера наиболее часто цитируется в программной индустрии и научных исследованиях. Рассмотрим каждую из метрик набора.

Метрика 1: Взвешенные методы на класс WMC (Weighted Methods Per Class)

Допустим, что в классе *C* определены *n* методов со сложностью $c_1...,c_2,..., c_n$. Для оценки сложности может быть выбрана любая метрика сложности (например, цикломатическая сложность). Главное — нормализовать эту метрику так, чтобы номинальная сложность для метода принимала значение 1. В этом случае

$$WMC = \sum_{i=1}^{n} C_i$$

Количество методов и их сложность являются индикатором затрат на реализацию и тестирование классов. Кроме того, чем больше методов, тем сложнее дерево наследования (все подклассы наследуют методы их родителей). С ростом количества методов в классе его применение

становится все более специфическим, тем самым ограничивается возможность многократного использования. По этим причинам метрика WMC должна иметь разумно низкое значение.

Очень часто применяют упрощенную версию метрики. При этом полагают $C_i = 1$, и тогда WMC — количество методов в классе.

Оказывается, что подсчитывать количество методов в классе достаточно сложно. Возможны два противоположных варианта учета.

1. Подсчитываются только методы текущего класса. Унаследованные методы игнорируются. Обоснование — унаследованные методы уже подсчитаны в тех классах, где они определялись. Таким образом, инкрементность класса — лучший показатель его функциональных возможностей, который отражает его право на существование. Наиболее важным источником информации для понимания того, что делает класс, являются его собственные операции. Если класс не может отреагировать на сообщение (например, в нем отсутствует собственный метод), тогда он пошлет сообщение родителю.
2. Подсчитываются методы, определенные в текущем классе, и все унаследованные методы. Этот подход подчеркивает важность пространства состояний в понимании класса (а не инкрементности класса).

Существует ряд промежуточных вариантов. Например, подсчитываются текущие методы и методы, прямо унаследованные от родителей. Аргумент в пользу данного подхода — на поведение дочернего класса наиболее сильно влияет специализация родительских классов.

На практике приемлем любой из описанных вариантов. Главное — не менять вариант учета от проекта к проекту. Только в этом случае обеспечивается корректный сбор метрических данных.

Метрика WMC дает относительную меру сложности класса. Если считать, что все методы имеют одинаковую сложность, то это будет просто количество методов в классе. Существуют рекомендации по сложности

методов. Например, М. Лоренц считает, что средняя длина метода должна ограничиваться 8 строками для Smalltalk и 24 строками для C++ [45]. Вообще, класс, имеющий максимальное количество методов среди классов одного с ним уровня, является наиболее сложным; скорее всего, он специфичен для данного приложения и содержит наибольшее количество ошибок.

Метрика 2: Высота дерева наследования DIT (Depth of Inheritance Tree)

DIT определяется как максимальная длина пути от листа до корня дерева наследования классов. Для показанной на рис. 14.3 иерархии классов метрика DIT равна 3.

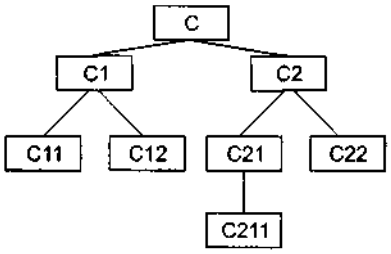

Рис. 14.3. Дерево наследования классов

Соответственно, для отдельного класса DIT, это длина максимального пути от данного класса до корневого класса в иерархии классов.

По мере роста DIT вероятно, что классы нижнего уровня будут наследовать много методов. Это приводит к трудностям в предсказании поведения класса. Высокая иерархия классов (большое значение DIT) приводит к большей сложности проекта, так как означает привлечение большего количества методов и классов.

Вместе с тем, большое значение DIT подразумевает, что многие методы могут использоваться многократно.

Метрика 3: Количество детей NOC (Number of children)

Подклассы, которые непосредственно подчинены суперклассу, называются его детьми. Значение NOC равно количеству детей, то есть количеству непосредственных наследников класса в иерархии классов. На рис. 14.3 класс *C*2 имеет двух детей — подклассы *C*21 и *C*22.

С увеличением NOC возрастает многократность использования, так как наследование — это форма повторного использования.

Однако при возрастании NOC ослабляется абстракция родительского класса. Это означает, что в действительности некоторые из детей уже не являются членами родительского класса и могут быть неправильно использованы.

Кроме того, количество детей характеризует потенциальное влияние класса на проект. По мере роста NOC возрастает количество тестов, необходимых для проверки каждого ребенка.

Метрики DIT и NOC — количественные характеристики формы и размера структуры классов. Хорошо структурированная объектно-ориентированная система чаще бывает организована как лес классов, чем как сверхвысокое дерево. По мнению Г. Буча, следует строить сбалансированные по высоте и ширине структуры наследования: обычно не выше, чем 7 ± 2 уровня, и не шире, чем 7 + 2 ветви [22].

Метрика 4: Сцепление между классами объектов CBO (Coupling between object classes)

CBO — это количество сотрудничеств, предусмотренных для класса, то есть количество классов, с которыми он соединен. Соединение означает, что методы данного класса используют методы или экземплярные переменные другого класса.

Другое определение метрики имеет следующий вид: CBO равно количеству сцеплений класса; сцепление образует вызов метода или свойства в другом классе.

Данная метрика характеризует статическую составляющую внешних связей классов.

С ростом CBO многократность использования класса, вероятно, уменьшается. Очевидно, что чем больше независимость класса, тем легче его повторно использовать в другом приложении.

Высокое значение CBO усложняет модификацию и тестирование, которое следует за выполнением модификации. Понятно, что, чем больше количество сцеплений, тем выше чувствительность всего проекта к изменениям в отдельных его частях. Минимизация межобъектных сцеплений улучшает модульность и содействует инкапсуляции проекта.

CBO для каждого класса должно иметь разумно низкое значение. Это согласуется с рекомендациями по уменьшению сцепления стандартного программного обеспечения.

Метрика 5: Отклик для класса RFC (Response For a Class)

Введем вспомогательное определение. Множество отклика класса RS — это множество методов, которые могут выполняться в ответ на прибытие сообщений в объект этого класса. Формула для определения RS имеет вид

$$RS = \{M\} U_{all\_i} \{R_i\},$$

где $\{R_i\}$ — множество методов, вызываемых методом г, $\{M\}$ — множество всех методов в классе.

Метрика RFC равна количеству методов во множестве отклика, то есть равна мощности этого множества:

$$\text{RFC} - \text{card}\{\text{RS}\}.$$

Приведем другое определение метрики: RFC — это количество методов класса плюс количество методов других классов, вызываемых из данного класса.

Метрика RFC является мерой потенциального взаимодействия данного класса с другими классами, позволяет судить о динамике поведения

соответствующего объекта в системе. Данная метрика характеризует динамическую составляющую внешних связей классов.

Если в ответ на сообщение может быть вызвано большое количество методов, то усложняются тестирование и отладка класса, так как от разработчика тестов требуется больший уровень понимания класса, растет длина тестовой последовательности.

С ростом RFC увеличивается сложность класса. Наихудшая величина отклика может использоваться при определении времени тестирования.

Метрика 6: Недостаток связности в методах LCOM (Lack of Cohesion in Methods)

Каждый метод внутри класса обращается к одному или нескольким свойствам (экземплярным переменным). Метрика *LCOM* показывает, насколько методы не связаны друг с другом через свойства (переменные). Если все методы обращаются к одинаковым свойствам, то LCOM = 0.

Введем обозначения:

- НЕ СВЯЗАНЫ — количество пар методов без общих экземплярных переменных;
- СВЯЗАНЫ — количество пар методов с общими экземплярными переменными.
- I_j— набор экземплярных переменных, используемых методом M_j

Очевидно, что

$$\text{НЕ СВЯЗАНЫ} = \text{card}\,\{I_{ij} \mid I_i \cap I_j = 0\},$$

$$\text{СВЯЗАНЫ} = \text{card}\,\{I_{ij} \mid I_i \cap I_j \neq 0\}.$$

Тогда формула для вычисления недостатка связности в методах примет вид

$$LCOM = \begin{cases} \text{НЕ СВЯЗАНЫ} - \text{СВЯЗАНЫ, если (НЕСВЯЗАНЫ} > \text{СВЯЗАНЫ)}; \\ 0 \text{ в противном случае.} \end{cases}$$

Можно определить метрику по-другому: LCOM — это количество пар методов, не связанных по свойствам класса, минус количество пар методов, имеющих такую связь.

Рассмотрим примеры применения метрики LCOM.

Пример 1: В классе имеются методы: *M1, M2, M3, M4.* Каждый метод работает со своим набором экземплярных переменных:

$$I_1=\{a, b\};\ I_2=\{a, c\};\ I_3=\{x, y\};\ I_4=\{m, n\}.$$

В этом случае

$$\text{НЕ СВЯЗАНЫ} = \text{card}\ (I_{13}, I_{14}, I_{23}, I_{24}, I_{34}) = 5;\ \text{СВЯЗАНЫ} = \text{card}\ (I_{12}) = 1.$$

$$\text{LCOM} = 5\text{-}1=4.$$

Пример 2: В классе используются методы: *M1, M2, M3.* Для каждого метода задан свой набор экземплярных переменных:

$$I_1 = \{a,b\};I_2=\{a,c\};I_3=\{x,y\},$$

$$\text{НЕ СВЯЗАНЫ} = \text{card}\ (I_{13}, I_{23}) = 2;\ \text{СВЯЗАНЫ} = \text{card}\ (I_{12}) = 1,$$

$$\text{LCOM} = 2\text{-}\ 1 = 1.$$

Связность методов внутри класса должна быть высокой, так как это содействует инкапсуляции. Если LCOM имеет высокое значение, то методы слабо связаны друг с другом через свойства. Это увеличивает сложность, в связи с чем возрастает вероятность ошибок при проектировании.

Высокие значения LCOM означают, что класс, вероятно, надо спроектировать лучше (разбиением на два или более отдельных класса). Любое вычисление LCOM помогает определить недостатки в проектировании классов, так как эта метрика характеризует качество упаковки данных и методов в оболочку класса.

Вывод: связность в классе желательно сохранять высокой, то есть следует добиваться низкого значения LCOM.

Набор метрик Чидамбера-Кемерера — одна из пионерских работ по комплексной оценке качества ОО-проектирования. Известны многочисленные предложения по усовершенствованию, развитию данного набора. Рассмотрим некоторые из них.

Недостатком метрики WMC является зависимость от реализации. Приведем пример. Рассмотрим класс, предлагающий операцию интегрирования. Возможны две реализации:

1) несколько простых операций:

Set_interval (min. max)

Setjnethod (method)

Set_precision (precision)

Set_function_to_integrate (function)

Integrate;

2) одна сложная операция:

Integrate (function, min, max. method, precision)

Для обеспечения независимости от этих реализаций можно определить метрику WMC2:

$$WMC2 = \sum_{i=1}^{n} (\text{Количество параметров } i-\text{го метода}).$$

Для нашего примера WMC2 = 5 и для первой, и для второй реализации. Заметим, для первой реализации WMC = 5, а для второй реализации WMC = 1.

Дополнительно можно определить метрику *Среднее число аргументов метода ANAM* (Average Number of Arguments per Method):

$$ANAM = WMC2/WMC.$$

Полезность метрики ANAM объяснить еще легче. Она ориентирована на принятые в ОО-проектировании решения — применять простые операции с малым количеством аргументов, а несложные операции — с многочисленными аргументами.

Еще одно предложение — ввести метрику, симметричную метрике LCOM. В то время как формула метрики LCOM имеет вид:

$$LCOM = \max (0, \text{НЕ СВЯЗАНЫ} - \text{СВЯЗАНЫ}),$$

симметричная ей метрика *Нормализованная NLCOM* вычисляется по формуле:

NLCOM = СВЯЗАНЫ/(НЕ СВЯЗАНЫ + СВЯЗАНЫ).

Диапазон значений этой метрики: $0 \leq$ NLCOM ≤ 1, причем чем ближе NLCOM к 1, тем выше связанность класса.

В наборе Чидамбера-Кемерера отсутствует метрика для прямого измерения информационной закрытости класса. В силу этого была предложена метрика *Поведенческая закрытость информации BIH* (Behaviourial Information Hiding):

BIH - (WEOC/WIEOC),

где WEOC — *взвешенные внешние операции на класс* (фактически это WMC);

WIEOC — *взвешенные внутренние и внешние операции на класс.*

WIEOC вычисляется так же, как и WMC, но учитывает полный набор операций, реализуемых классом. Если BIH = 1, класс показывает другим классам все свои возможности. Чем меньше BIM, тем меньше видимо поведение класса. BIH может рассматриваться и как мера сложности. Сложные классы, вероятно, будут иметь малые значения BIH, а простые классы — значения, близкие к 1. Если класс с высокой WMC имеет значение BIH, близкое к 1, следует выяснить, почему он настолько видим извне.

Использование метрик Чидамбера-Кемерера

Поскольку основу логического представления ПО образует структура классов, для оценки ее качества удобно использовать метрики Чидамбера-Кемерера. Пример расчета метрик для структуры, показанной на рис. 14.4, представлен в табл. 14.4.

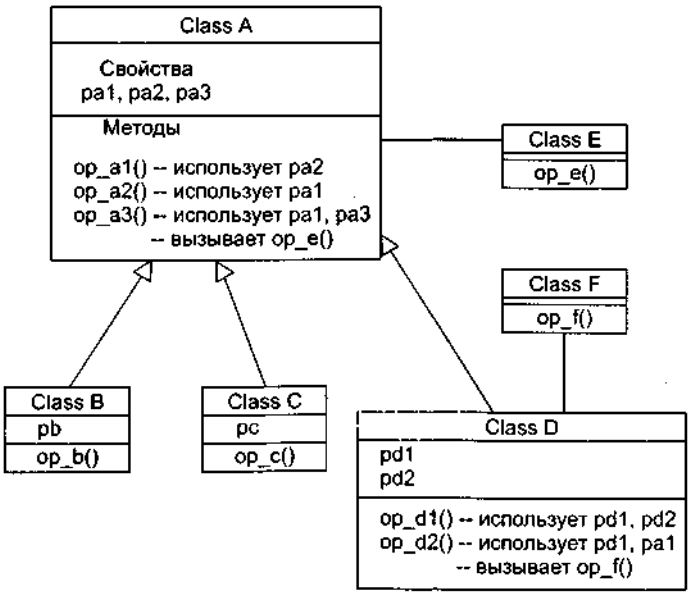

Рис. 14.4. Структура классов для расчета метрик Чидамбера-Кемерера

Прокомментируем результаты расчета. Класс Class A имеет три метода (op_al(), op_a2(), op_a3()), трех детей (Class B, Class C, Class D) и является корневым классом. Поэтому метрики WMC, NOC и DIT имеют, соответственно, значения 3, 3 и 0.

Метрика CBO для класса Class A равна 1, так как он использует один метод из другого класса (метод op_e() из класса Class E, он вызывается из метода op_a3()). Метрика RFC для класса Class A равна 4, так как в ответ на прибытие в этот класс сообщений возможно выполнение четырех методов (три объявлены в этом классе, а четвертый метод op_e() вызывается из op_a3()).

Таблица 14.4. Пример расчета метрик Чидамбера-Кемерера

Имя класса	WMC	DIT	NOC	CBO	RFC	LCOM
Class A	3	0	3	1	4	1
Class B	1	1	0	0	1	0
Class C	1	1	0	0	1	0
Class D	2	1	0	2	3	0

Для вычисления метрики LCOM надо определить количество пар методов класса. Оно рассчитывается по формуле

$$C_m^2 = m!/(2(m-2)!),$$

где m — количество методов класса.

Поскольку в классе три метода, возможны три пары: op_al( )&op_a2(), op_al( )&op_a3() и op_a2( )&op_a3(). Первая и вторая пары не имеют общих свойств, третья пара имеет общее свойство (pal). Таким образом, количество несвязанных пар равно 2, количество связанных пар равно 1, и LCOM = 2-1 = 1.

Отметим также, что для класса Class D метрика CBO равна 2, так как здесь используются свойство pal и метод op_f() из других классов. Метрика LCOM в этом классе равна 0, поскольку методы op_dl() и op_d2() связаны по свойству pdl, а отрицательное значение запрещено.

Метрики Лоренца и Кидда

Коллекция метрик Лоренца и Кидда — результат практического, промышленного подхода к оценке ОО-проектов [45].

Метрики, ориентированные на классы

М. Лоренц и Д. Кидд подразделяют метрики, ориентированные на классы, на четыре категории: метрики размера, метрики наследования, внутренние и внешние метрики.

Размерно-ориентированные метрики основаны на подсчете свойств и операций для отдельных классов, а также их средних значений для всей ОО-системы. Метрики наследования акцентируют внимание на способе повторного использования операций в иерархии классов. Внутренние метрики классов рассматривают вопросы связности и кодирования. Внешние метрики исследуют сцепление и повторное использование.

Метрика 1: Размер класса CS (Class Size)

Общий размер класса определяется с помощью следующих измерений:

- общее количество операций (вместе с приватными и наследуемыми экземплярными операциями), которые инкапсулируются внутри класса;
- количество свойств (вместе с приватными и наследуемыми экземплярными свойствами), которые инкапсулируются классом.

Метрика WMC Чидамбера и Кемерера также является взвешенной метрикой размера класса.

Большие значения CS указывают, что класс имеет слишком много обязанностей. Они уменьшают возможность повторного использования класса, усложняют его реализацию и тестирование.

При определении размера класса унаследованным (публичным) операциям и свойствам придают больший удельный вес. Причина — приватные операции и свойства обеспечивают специализацию и более локализованы в проекте.

Могут вычисляться средние количества свойств и операций класса. Чем меньше среднее значение размера, тем больше вероятность повторного использования класса.

Рекомендуемое значение CS ≤ 20 методов.

Метрика 2: Количество операций, переопределяемых подклассом, NOO (Number of Operations Overridden by a Subclass)

Переопределением называют случай, когда подкласс замещает операцию, унаследованную от суперкласса, своей собственной версией.

Большие значения NOO обычно указывают на проблемы проектирования. Ясно, что подкласс должен расширять операции суперкласса. Расширение проявляется в виде новых имен операций. Если же NOO велико, то разработчик нарушает абстракцию суперкласса. Это

ослабляет иерархию классов, усложняет тестирование и модификацию программного обеспечения.

Рекомендуемое значение NOO ≤ 3 методов.

Метрика 3: Количество операций, добавленных подклассом, NOA (Number of Operations Added by a Subclass)

Подклассы специализируются добавлением приватных операций и свойств. С ростом NOA подкласс удаляется от абстракции суперкласса. Обычно при увеличении высоты иерархии классов (увеличении DIT) должно уменьшаться значение NOA на нижних уровнях иерархии.

Для рекомендуемых значений CS = 20 и DIT = 6 рекомендуемое значение NOA ≤ 4 методов (для класса-листа).

Метрика 4: Индекс специализации SI (Specialization Index)

Обеспечивает грубую оценку степени специализации каждого подкласса. Специализация достигается добавлением, удалением или переопределением операций:

$$SI = (NOO \times уровень) / M_{общ},$$

где *уровень* — номер уровня в иерархии, на котором находится подкласс, $M_{общ}$ — общее количество методов класса.

Пример расчета индексов специализации приведен на рис. 14.5.

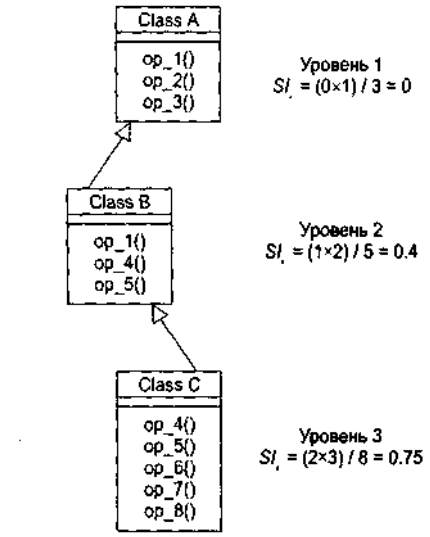

Рис. 14_05

Рис. 14.5. Расчет индексов специализации классов

Чем выше значение SI, тем больше вероятность того, что в иерархии классов есть классы, нарушающие абстракцию суперкласса.

Рекомендуемое значение SI $\leq 0{,}15$.

Операционно-ориентированные метрики

Эта группа метрик ориентирована на оценку операций в классах. Обычно методы имеют тенденцию быть небольшими как по размеру, так и по логической сложности. Тем не менее реальные характеристики операций могут быть полезны для глубокого понимания системы.

Метрика 5: Средний размер операции OS_{AVG} (Average Operation Size)

В качестве индикатора размера может использоваться количество строк программы, однако LOC-оценки приводят к известным проблемам. Альтернативный вариант — *«количество сообщений, посланных операцией».*

Рост значения метрики означает, что обязанности размещены в классе не очень удачно. Рекомендуемое значение $OS_{AVG} \leq 9$.

Метрика 6: Сложность операции OC (Operation Complexity

Сложность операции может вычисляться с помощью стандартных метрик сложности, то есть с помощью LOC- или FP-оценок, метрики цикломатической сложности, метрики Холстеда.

М. Лоренц и Д. Кидд предлагают вычислять ОС суммированием оценок с весовыми коэффициентами, приведенными в табл. 14.5.

Таблица 14.5. Весовые коэффициенты для метрики ОС

Параметр	**Вес**
Вызовы функций API	5,0
Присваивания	0,5
Арифметические операции	2,0
Сообщения с параметрами	3,0
Вложенные выражения	0,5
Параметры	0,3
Простые вызовы	7,0
Временные переменные	0,5
Сообщения без параметров	1,0

Поскольку операция должна быть ограничена конкретной обязанностью, желательно уменьшать ОС.

Рекомендуемое значение ОС ≤ 65 (для предложенного суммирования).

Метрика 7: Среднее количество параметров на операцию NP_{AVG} (Average Number of Parameters per operation)

Чем больше параметров у операции, тем сложнее сотрудничество между объектами. Поэтому значение NP_{AVG} должно быть как можно меньшим.

Рекомендуемое значение $NP_{AVG} = 0,7$.

Метрики для ОО-проектов

Основными задачами менеджера проекта являются планирование, координация, отслеживание работ и управление программным проектом.

Одним из ключевых вопросов планирования является оценка размера программного продукта. Прогноз размера продукта обеспечивают следующие ОО-метрики.

Метрика 8: Количество описаний сценариев NSS (Number of Scenario Scripts)

Это количество прямо пропорционально количеству классов, требуемых для реализации требований, количеству состояний для каждого класса, а также количеству методов, свойств и сотрудничеств. Метрика NSS — эффективный индикатор размера программы.

Рекомендуемое значение NSS — не менее одного сценария на публичный протокол подсистемы, отражающий основные функциональные требования к подсистеме.

Метрика 9: Количество ключевых классов NKC (Number of Key Classes)

Ключевой класс прямо связан с коммерческой проблемной областью, для которой предназначена система. Маловероятно, что ключевой класс может появиться в результате повторного использования существующего класса. Поэтому значение NKC достоверно отражает предстоящий объем разработки. М. Лоренц и Д. Кидд предполагают, что в типовой ОО-системе на долю ключевых классов приходится 20-40% от общего количества

классов. Как правило, оставшиеся классы реализуют общую инфраструктуру (GUI, коммуникации, базы данных).

Рекомендуемое значение: если NKC < 0,2 от общего количества классов системы, следует углубить исследование проблемной области (для обнаружения важнейших абстракций, которые нужно реализовать).

Метрика 10: Количество подсистем NSUB (NumberofSUBsystem)

Количество подсистем обеспечивает понимание следующих вопросов: размещение ресурсов, планирование (с акцентом на параллельную разработку), общие затраты на интеграцию.

Рекомендуемое значение: NSUB > 3.

Значения метрик NSS, NKC, NSUB полезно накапливать как результат каждого выполненного ОО-проекта. Так формируется метрический базис фирмы, в который также включаются метрические значения по классами и операциям. Эти исторические данные могут использоваться для вычисления метрик производительности (среднее количество классов на разработчика или среднее количество методов на человеко-месяц). Совместное применение метрик позволяет оценивать затраты, продолжительность, персонал и другие характеристики текущего проекта.

Набор метрик Фернандо Абреу

Набор метрик *MOOD* (Metrics for Object Oriented Design), предложенный Ф. Абреу в 1994 году, — другой пример академического подхода к оценке качества ОО-проектирования [6]. Основными целями MOOD-набора являются:

1) покрытие базовых механизмов объектно-ориентированной парадигмы, таких как инкапсуляция, наследование, полиморфизм, посылка сообщений;
2) формальное определение метрик, позволяющее избежать

субъективности измерения;

3) независимость от размера оцениваемого программного продукта;
4) независимость от языка программирования, на котором написан оцениваемый продукт.

Набор MOOD включает в себя следующие метрики:

1) фактор закрытости метода (MHF);
2) фактор закрытости свойства (AHF);
3) фактор наследования метода (MIF);
4) фактор наследования свойства (AIF);
5) фактор полиморфизма (POF);
6) фактор сцепления (COF).

Каждая из этих метрик относится к основному механизму объектно-ориентированной парадигмы: инкапсуляции (MHF и AHF), наследованию (MIF и AIF), полиморфизму (POF) и посылке сообщений (COF). В определениях MOOD не используются специфические конструкции языков программирования.

Метрика 1: Фактор закрытости метода MHF (Method Hiding Factor)

Введем обозначения:

- $M_v(C_i)$ — количество видимых методов в классе C_i (интерфейс класса);
- $M_h(C_i)$ — количество скрытых методов в классе C_i (реализация класса);
- $M_d(C_i) = M_v(C_i) + M_h(C_i)$ — общее количество методов, определенных в классе C, (унаследованные методы не учитываются).

Тогда формула метрики MHF примет вид:

$$MHF = \frac{\sum_{i=1}^{TC} M_h(C_i)}{\sum_{i=1}^{TC} M_d(C_i)},$$

где TC — количество классов в системе.

Если видимость *m-го* метода *i*-го класса из *j*-го класса вычислять по выражению:

$$is\_visible(M_{mi}, C_j) = \begin{cases} 1, \text{if} \begin{cases} j \neq 1 \\ C_j \text{ может вызвать } M_{mi} \end{cases} \\ 0, \text{else} \end{cases},$$

а процентное количество классов, которые видят *m*-й метод *i*-го класса, определять по соотношению:

$$V(M_{mi}) = \frac{\sum_{i=1}^{TC} is\_visible(M_{mi}, C_j)}{TC - 1}$$

то формулу метрики MHF можно представить в виде:

$$MHF = \frac{\sum_{i=1}^{TC} \sum_{m=1}^{M_d(C_i)} (1 - V(M_{mi}))}{\sum_{i=1}^{TC} M_d(C_i)}.$$

В числителе этой формулы MHF — сумма закрытости всех методов во всех классах. Закрытость метода — процентное количество классов, из которых данный метод невидим. Знаменатель MHF — общее количество методов, определенных в рассматриваемой системе.

С увеличением MHF уменьшаются плотность дефектов в системе и затраты на их устранение. Обычно разработка класса представляет собой пошаговый процесс, при котором к классу добавляется все больше и больше деталей (скрытых методов). Такая схема разработки способствует возрастанию как значения MHF, так и качества класса.

Метрика 2: Фактор закрытости свойства AHF (Attribute Hiding Factor)

Введем обозначения:

- $A_v(C_i)$ — количество видимых свойств в классе C_i (интерфейс класса);
- $A_h(C_i)$ — количество скрытых свойств в классе C_i (реализация класса);
- $A_d(C_i) = A_v(C_i) + A_h(C_i)$ — общее количество свойств, определенных в классе C_i (унаследованные свойства не учитываются).

Тогда формула метрики *AHF* примет вид:

$$AHF = \frac{\sum_{i=1}^{TC} A_h(C_i)}{\sum_{i=1}^{TC} A_d(C_i)},$$

где TC — количество классов в системе.

Если видимость *m*-го свойства *i*-го класса из *j*-го класса вычислять по выражению:

$$is\_visible(A_{mi}, C_j) = \begin{cases} 1, \text{if} \begin{cases} j \neq 1 \\ C_j \text{ может вызвать } A_{mi} \end{cases} \\ 0, \text{else} \end{cases},$$

а процентное количество классов, которые видят *m*-е свойство *i*-го класса, определять по соотношению:

$$V(A_{mi}) = \frac{\sum_{i=1}^{TC} is\_visible(A_{mi}, C_j)}{TC - 1},$$

то формулу метрики *AHF*можно представить в виде:

$$AHF = \frac{\sum_{i=1}^{TC} \sum_{m=1}^{A_d(C_i)} (1 - V(A_{mi}))}{\sum_{i=1}^{TC} A_d(C_i)}.$$

В числителе этой формулы AHF — сумма закрытости всех свойств во всех классах. Закрытость свойства — процентное количество классов, из которых данное свойство невидимо. Знаменатель AHF — общее количество свойств, определенных в рассматриваемой системе.

В идеальном случае все свойства должны быть скрыты и доступны только для методов соответствующего класса (AHF = 100%).

Метрика 3: Фактор наследования метода MIF (Method Inheritance Factor)

Введем обозначения:

- $M_i(C_i)$ — количество унаследованных и не переопределенных методов в классе C_i;
- $M_0(C_i)$ — количество унаследованных и переопределенных методов в

классе C_i;

- $M_n(C_i)$ — количество новых (не унаследованных и переопределенных) методов в классе C_i;
- $M_d(C_i) = M_n(C_i) + M_0(C_i)$ — количество методов, определенных в классе C_i;
- $M_a(C_i) = M_d(C_i) + M_i(C_i)$ — общее количество методов, доступных в классе C_i.

Тогда формула метрики MIF примет вид:

$$MIF = \frac{\sum_{i=1}^{TC} M_i(C_i)}{\sum_{i=1}^{TC} M_i(C_i)}.$$

Числителем MIF является сумма унаследованных (и не переопределенных) методов во всех классах рассматриваемой системы. Знаменатель MIF — это общее количество доступных методов (локально определенных и унаследованных) для всех классов.

Значение MIF = 0 указывает, что в системе отсутствует эффективное наследование, например, все унаследованные методы переопределены.

С увеличением MIF уменьшаются плотность дефектов и затраты на исправление ошибок. Очень большие значения MIF (70-80%) приводят к обратному эффекту, но этот факт нуждается в дополнительной экспериментальной проверке. Сформулируем «осторожный» вывод: умеренное использование наследования — подходящее средство для снижения плотности дефектов и затрат на доработку.

Метрика 4: Фактор наследования свойства AIF (Attribute Inheritance Factor)

Введем обозначения:

- $A_i(C_i)$ — количество унаследованных и не переопределенных свойств в классе C_i;
- $A_0(C_i)$ — количество унаследованных и переопределенных свойств в

классе C_i;

- $A_n(C_i)$ — количество новых (не унаследованных и переопределенных) свойств в классе C_i;
- $A_d(C_i) = A_n(C_i) + A_0(C_i)$ — количество свойств, определенных в классе C_i;
- $A_a(C_i) = A_d(C_i) + A_i(C_i)$ — общее количество свойств, доступных в классе C_i.

Тогда формула метрики AIF примет вид:

$$AIF = \frac{\sum_{i=1}^{TC} A_i(C_i)}{\sum_{i=1}^{TC} A_a(C_i)}.$$

Числителем AIF является сумма унаследованных (и не переопределенных) свойств во всех классах рассматриваемой системы. Знаменатель AIF — это общее количество доступных свойств (локально определенных и унаследованных) для всех классов.

Метрика 5: Фактор полиморфизма POF (Polymorphism Factor)

Введем обозначения:

- $M_0(C_i)$ — количество унаследованных и переопределенных методов в классе C_i;
- $M_n(C_i)$ — количество новых (не унаследованных и переопределенных) методов в классе C_i;
- $DC(C_i)$ — количество потомков класса C_i;
- $M_d(C_i) = M_n(C_i) + M_0(C_i)$ — количество методов, определенных в классе C_i.

Тогда формула метрики POF примет вид:

$$POF = \frac{\sum_{i=1}^{TC} M_0(C_i)}{\sum_{i=1}^{TC} [M_n(C_i) \times DC(C_i)]}.$$

Числитель POF фиксирует реальное количество возможных полиморфных ситуаций. Очевидно, что сообщение, посланное в класс C_i связывается (статически или динамически) с реализацией именуемого метода. Этот метод, в свою очередь, может или представляться несколькими «формами», или переопределяться (в потомках C_i).

Знаменатель POF представляет максимальное количество возможных полиморфных ситуаций для класса C_i. Имеется в виду случай, когда все новые методы, определенные в C_i, переопределяются во всех его потомках.

Умеренное использование полиморфизма уменьшает как плотность дефектов, так и затраты на доработку. Однако при POF > 10% возможен обратный эффект.

Метрика 6: Фактор сцепления COF (Coupling Factor)

В данном наборе сцепление фиксирует наличие между классами отношения «клиент-поставщик» (client-supplier). Отношение «клиент-поставщик» (C_c =>C_s) здесь означает, что класс-клиент содержит но меньшей мере одну не унаследованную ссылку на свойство или метод класса-поставщика.

$$is\_client(C_c, C_s) = \begin{cases} 1, \text{if } C_c => C_s \cap C_c \neq C_s, \\ 0, \text{else}, \end{cases}$$

Если наличие отношения «клиент-поставщик» определять по выражению:

$$COF = \frac{\sum_{i=1}^{TC}[\sum_{j=1}^{TC} is\_client(C_i, C_j)]}{TC^2 - TC}.$$

то формула для вычисления метрики COF примет вид:

Знаменатель COF соответствует максимально возможному количеству сцеплений в системе с TC-классами (потенциально каждый класс может быть поставщиком для других классов). Из рассмотрения исключены рефлексивные отношения — когда класс является собственным

поставщиком. Числитель COF фиксирует реальное количество сцеплений, не относящихся к наследованию.

С увеличением сцепления классов плотности дефектов и затрат на доработку также возрастают. Сцепления отрицательно влияют на качество ПО, их нужно сводить к минимуму. Практическое применение этой метрики доказывает, что сцепление увеличивает сложность, уменьшает инкапсуляцию и возможности повторного использования, затрудняет понимание и усложняет сопровождение ПО.

Метрики для объектно-ориентированного тестирования

Рассмотрим проектные метрики, которые, по мнению Р. Байндера (Binder), прямо влияют на тестируемость ОО-систем [17]. Р. Байндер сгруппировал эти метрики в три категории, отражающие важнейшие проектные характеристики.

Метрики инкапсуляции

К метрикам инкапсуляции относятся: «Недостаток связности в методах LCOM», «Процент публичных и защищенных PAP (Percent Public and Protected)» и «Публичный доступ к компонентным данным PAD (Public Access to Data members)».

Метрика 1: Недостаток связности в методах LCOM

Чем выше значение LCOM, тем больше состояний надо тестировать, чтобы гарантировать отсутствие побочных эффектов при работе методов.

Метрика 2: Процент публичных и защищенных PAP (Percent Public and Protected)

Публичные свойства наследуются от других классов и поэтому видимы для этих классов. Защищенные свойства являются специализацией и приватны для определенного подкласса. Эта метрика показывает процент публичных свойств класса. Высокие значения *PAP* увеличивают вероятность побочных эффектов в классах. Тесты должны гарантировать обнаружение побочных эффектов.

Метрика 3: Публичный доступ к компонентным данным PAD (Public Access to Data members)

Метрика показывает количество классов (или методов), которые имеют доступ к свойствам других классов, то есть нарушают их инкапсуляцию. Высокие значения приводят к возникновению побочных эффектов в классах. Тесты должны гарантировать обнаружение таких побочных эффектов.

Метрики наследования

К метрикам наследования относятся «Количество корневых классов NOR (Number Of Root classes)», «Коэффициент объединения по входу FIN», «Количество детей NOC» и «Высота дерева наследования DIT».

Метрика 4: Количество корневых классов NOR (Number Of Root classes)

Эта метрика подсчитывает количество деревьев наследования в проектной модели. Для каждого корневого класса и дерева наследования должен разрабатываться набор тестов. С увеличением NOR возрастают затраты на тестирование.

Метрика 5: Коэффициент объединения по входу FIN

В контексте О-О-смистем FIN фиксирует множественное наследование. Значение FIN > 1 указывает, что класс наследует свои свойства и операции от нескольких корневых классов. Следует избегать FIN > 1 везде, где это возможно.

Метрика 6: Количество детей NOC

Название говорит само за себя. Метрика заимствована из набора Чидамбера-Кемерера.

Метрика 7: Высота дерева наследования DIT

Метрика заимствована из набора Чидамбера-Кемерера. Методы суперкласса должны повторно тестироваться для каждого подкласса.

В дополнение к перечисленным метрикам Р. Байндер выделил метрики сложности класса (это метрики Чидамбера-Кемерера — WMC, CBO, RFC и метрики для подсчета количества методов), а также метрики полиморфизма.

Метрики полиморфизма

Рассмотрим следующие метрики полиморфизма: «Процентное количество не переопределенных запросов OVR», «Процентное количество динамических запросов DYN», «Скачок класса Bounce-C» и «Скачок системы Bounce-S».

Метрика 8: Процентное количество не переопределенных запросов OVR

Процентное количество от всех запросов в тестируемой системе, которые не приводили к перекрытию модулей. Перекрытие может приводить к непредусмотренному связыванию. Высокое значение OVR увеличивает возможности возникновения ошибок.

Метрика 9: Процентное количество динамических запросов DYN

Процентное количество от всех сообщений в тестируемой системе, чьи приемники определяются в период выполнения. Динамическое связывание может приводить к непредусмотренному связыванию. Высокое значение DYN означает, что для проверки всех вариантов связывания метода потребуется много тестов.

Метрика 10: Скачок класса Bounce-C

Количество скачущих маршрутов, видимых тестируемому классу. Скачущий маршрут — это маршрут, который в ходе динамического связывания пересекает несколько иерархий классов-поставщиков. Скачок может приводить к непредусмотренному связыванию. Высокое значение Bounce-C увеличивает возможности возникновения ошибок.

Метрика 11: Скачок системы Bounce-S

Количество скачущих маршрутов в тестируемой системе. В этой метрике суммируется количество скачущих маршрутов по каждому классу системы. Высокое значение Bounce-S увеличивает возможности возникновения ошибок.

Задание к лабораторной работе

Для своего программного проекта (для которого написали техническое задание) сделать оценку с использованием приведенных метрик.(Набор метрик Чидамбера и Кемерера, или Метрики Лоренца и Кидда, или Набор метрик Фернандо Абреу)

Варианты заданий:

Набор метрик Чидамбера и Кемерера	1, 4, 7, 10, 13, 16, 19, 22, 25,
Метрики Лоренца и Кидда	2, 5, 8, 11, 14,17, 20, 23, 26,
Набор метрик Фернандо Абреу	3, 6, 9, 12, 15, 18, 21, 24, 27

Порядок выполнения работы.

Измерить программный продукт и процесс его разработки с использованием приведенных метрик по варианту, сделать выводы и привести рекомендации по улучшению разрабатываемого программного обеспечения.

Требование к отчету по лабораторной работе:

1. Свой программный проект (для которого написали техническое задание)
2. Краткие теоретические сведения.
3. Расчеты в соответствии с вариантом.
4. Выводы.

Контрольные вопросы

1. Какие факторы объектно-ориентированных систем влияют на метрики для их оценки и как проявляется это влияние?
2. Какое влияние оказывает наследование на связность классов?
3. Охарактеризуйте метрики связности классов по данным.

4. Охарактеризуйте метрики связности классов по методам.
5. Какие характеристики объектно-ориентированных систем ухудшают сцепление классов?
6. Объясните, как определить сцепление классов с помощью метрики «зависимость изменения между классами».
7. Поясните смысл метрики локальности данных.
8. Какие метрики входят в набор Чидамбера и Кемерера? Какие задачи они решают?
9. Как можно подсчитывать количество методов в классе?
10. Какие метрики Чидамбера и Кемерера оценивают сцепление классов? Поясните их смысл.
11. Какая метрика Чидамбера и Кемерера оценивает связность класса? Поясните ее смысл.
12. Как добиться независимости метрики WMC от реализации?
13. Как можно оценить информационную закрытость класса?
14. Сравните наборы Чидамбера-Кемерера и Лоренца-Кидда. Чем они похожи? В чем различие?
15. На какие цели ориентирован набор метрик Фернандо Абреу?
16. Охарактеризуйте состав набора метрик Фернандо Абреу.
17. Сравните наборы Чидамбера-Кемерера и Фернандо Абреу. Чем они похожи? В чем различие?
18. Сравните наборы Лоренца-Кидда и Фернандо Абреу. Чем они похожи? В чем различие?
19. Дайте характеристику метрик для объектно-ориентированного тестирования.

Лабораторная работа №6

Тема: Организация таблиц в трансляторах и работа с ними.

Цель работы: Получить практические навыки проектирования и реализации таблиц в трансляторах и операций с таблицами.

Методические указания к лабораторной работе

Для выполнения работы необходимо знать способы организации таблиц и методы поиска информации в таблицах.

При выполнении лабораторной работы построить и реализовать конкретную таблицу для транслятора с одного из предлагаемых языков программирования: СИ, ПАСКАЛЬ, C#, и т.п.

Необходимые теоретические сведения для выполнения работы изложены в книгах [см. список литературы к методическим указаниям].

Теоретические сведения

Таблица - это множество записей, каждая из которых представляет набор поименованных полей. Одно поле записи является ключом, по значению которого осуществляется поиск записи в таблице и добавление новой записи в таблицу.

Таблицы являются наиболее употребительной структурой данных в трансляторах. При трансляции описательной части программы заполняются соответствующие таблицы транслятора. Трансляция исполнительной части программы состоит в генерации команд выходного языка. При генерации осуществляются необходимые семантические проверки на основе информации, содержащейся в таблицах. Значительная часть времени трансляции затрачивается на поиск в таблицах. Поэтому эффективной организации поиска информации в таблицах уделяется особое внимание. Кроме того, каждая запись таблицы должна занимать как можно меньше

памяти для того, чтобы хватило памяти при трансляции больших программ, и трансляция была быстрой.

Основной таблицей любого транслятора является "ТИ"-таблица идентификаторов (имен, символов), в которой хранятся все атрибуты (характеристики) идентификаторов, необходимые для проверки семантики программы и для генерации машинного кода, ключом для "ТИ" является сам идентификатор, а его атрибутами могут быть тип значения, адрес в памяти, признак инициализации значения переменной, обозначаемой идентификатором, и т.д. атрибуты идентификатора выясняются из описания и контекста использования идентификатора в программе.

В языках, имеющих конечное число типов (АЛГОЛ-60, ФОРТРАН, PL/I), тип можно представить целым числом. В языках, имеющих потенциально бесконечное число типов (АЛГОЛ-68, ПАСКАЛЬ, АДА, СИ), информация о типе представляется указателем на запись в специальной таблице типов. В таблице типов хранят имя типа, объем памяти, занимаемой одним

Значением данного типа, структурные характеристики: для массивов - количество измерений и границы изменения индексов по каждому измерению; для структур (записей) - количество элементов (полей), для каждого элемента (поля) необходимо помнить имя и тип значения,

В зависимости от реализуемого языка программирования и методов трансляции в трансляторе могут также использоваться таблицы меток, процедур, операций и т.д.

В таблице меток хранят саму метку, признак определенности метки, вид метки (метка оператора, метка процедуры и т.п.), адрес команды, с которой связывается метка.

В таблице процедур содержится следующая информация: имя процедуры, вид процедуры(процедура или функция), количество

параметров, их имена и тип, адрес первой команды тела процедуры, для функций - тип результата.

В таблице операций необходимо иметь обозначение операции, ее приоритет и ассоциативность, связь с процедурой генерации машинных команд для данной операции. Информация о допустимых типах значения операндов операции может храниться в таблице или находиться в соответствующей процедуре генерации машинных команд.

Все таблицы транслятора можно разделить на статические и динамические.

В статических таблицах независимо от транслируемой программы имеется постоянное количество неизменяемых записей. Примерами статических таблиц могут быть таблица операций, определенных в языке, таблица встроенных в язык примитивных функций. Количество и содержание таких таблиц определяется языком программирования, для которого конструируется транслятор.

В любом трансляторе имеются динамические таблицы, для которых количество и/или содержимое записей таблицы изменяется в ходе трансляции. Примеры динамических таблиц "ТИ", таблица типов, таблица констант и т. п. В динамических таблицах может быть статическая часть. Например, в "ТИ" могут находиться ключевые слова языка, в таблице типов - встроенные в язык типы, в таблице процедур - встроенные в язык примитивные функции, а в таблице констант - константы, определяемые языком программирования и его реализацией. Статические части таблиц долины быть заполнены (проинициализированы) до начала трансляции программ. Для динамических таблиц возможно оформление статической части в виде отдельной таблицы с более эффективной реализацией операции поиска информации.

Суммарный объем памяти, необходимой для статических таблиц, не зависит от величины и содержимого транслируемой программы, поэтому

основное внимание следует обратить на память для динамических таблиц транслятора. Требование минимального объема памяти для одной записи таблицы выполняется, если все поля записи минимального размера. Однако некоторые поля могут представлять информацию, которую нельзя непосредственно закодировать, используя поле фиксированного размера. Например, длина идентификатора во многих языках программирования не ограничивается. В этом случае размер поля берут по максимуму, или в поле помещают указатель на дополнительную структуру данных (см. рис. 1).

В "ТИ" количество информации, которое нужно хранить для идентификатора, зависит от объекта, связываемого с идентификатором - простой переменной, массивом, процедурой и т.п. таким образом, количество необходимых полей одной

Записи таблицы может зависеть от значений атрибутов записи, в данном случае возможно несколько способов структурной организации одной записи таблицы транслятора.

1) альтернативные поля выносят в дополнительные структуры данных, а в исходной таблице хранят указатель. Для рассмотренного примера вид объекта будет определять дополнительную структуру данных (в частном случае таблицу), на которую ссылается указатель.

2) альтернативные поля "перекрываются" по памяти. Для рассматриваемого примера каждая запись будет иметь поля, содержащие или атрибуты массива или атрибуты процедуры. Смысл содержимого "перекрытых" полей определяется видом объекта, связываемого с идентификатором.

3) запись таблицы содержит переменное количество полей, в зависимости от объекта. При этом структура и длина записей в одной таблице будут переменными, поэтому каждая.

Запись должна содержать информацию о своей структуре и длине. Например, для транслятора с языка PL/I возможные связи таблиц-идентификаторов (ТИ), меток (ТМ) и процедур (ТП) - показаны на рис. 2 для оператора описания процедуры-функции, вычисляющей максимальное значение функции F на интервале (A,B).

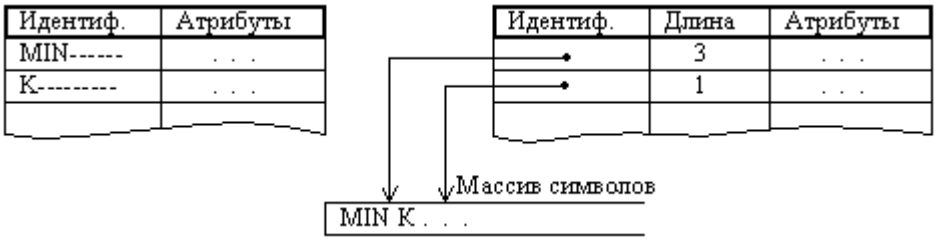

Рисунок 1 – Примеры структуры "Таблица идентификаторов" для языка АДА

MAXF: PROCEDURE (F, A, B) RETURNS (REAL (B));

Выбор структурной организации записи таблицы и способов кодирования информации в ней, связей с другими таблицами в каждом конкретном случае осуществляется компромиссом между противоречивыми требованиями минимальности памяти и быстрого выполнения операций,

Для таблиц в трансляторах используют следующие операции:

1) поиск записи;

2) добавление записи;

3-4) чтение/занесение значения атрибута заданной записи;

5) исключение записи;

6) реорганизация таблицы.

Во время трансляции для статических таблиц определены операции 1 и 3, а для динамических таблиц могут применяться все перечисленные операции.

Поиск записи в таблице наиболее употребительная операция, она реализуется по-разному для статических и динамических таблиц, для статических таблиц желателен прямой метод поиска: по значению ключа сразу определяется место соответствующей записи в таблице. Последовательный поиск допустим для статических таблиц с небольшим количеством записей (например, поиск в таблице операций). Для ускорения поиска в динамических таблицах используется хеш-адресация: адрес записи в таблице получается "хешированием" ключа - выполнением простых арифметических или логических операций над ключом. По методу организаций поиска выделяют несколько методов организации таблиц: неупорядоченная (простая иди древовидная), упорядоченная, с вычисляемыми входами и т.д.

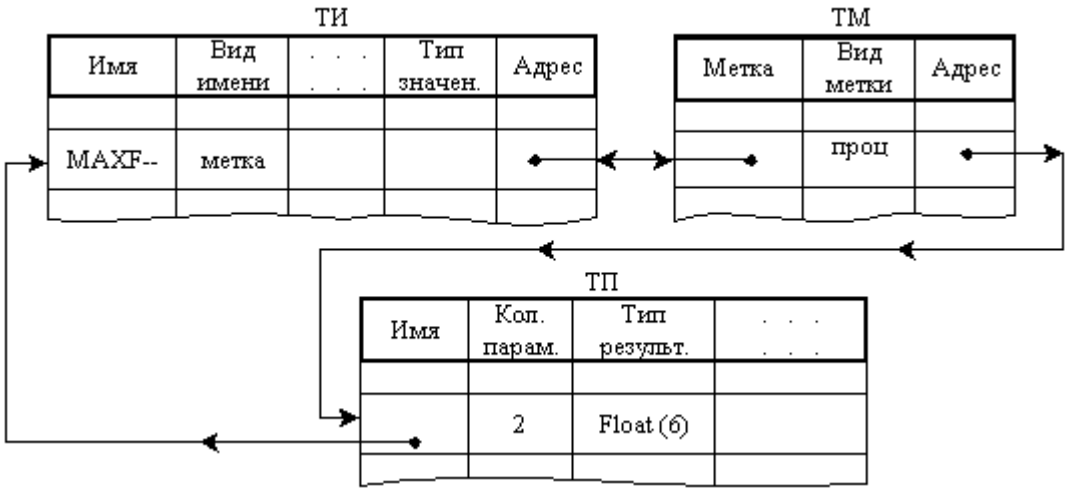

Рисунок 2 – Связи таблиц в трансляторе с языка PL/1

Как правило, добавлению записи в таблицу предшествует поиск. Если

поиск не обнаружил соответствующей записи в таблице, то, возможно, эту запись необходимо добавить в таблицу.

При семантических проверках нужно знать атрибуты записей. Для этого используется операция "чтения значения" атрибута заданной записи.

Например, для конструкций "вызов процедуры" необходимо проверить соответствие фактических и формальных параметров по количеству и типу значений. Рекомендуется реализацию операция "чтения значения" атрибута записи выполнять в виде макроопределения или процедуры-функции.

В ряде языков программирования для некоторых объектов (например, меток) использующее вхождение объекта (оператор GOTO) может в тексте программы предшествовать определяющему вхождению объекта (оператора, помеченного меткой). В этом и в ряде других случаев доопределенные значения атрибутов заносятся в таблицу с помочью операции занесения значения атрибута.

Для языков программирования, имеющих блочную структуру программы (СИ, АДА, ПАСКАЛЬ и др.), допускается локализация объектов внутри блока (меток, переменных, процедур и т.п.), при завершении трансляции блока информация о его локальных объектах должна быть удалена из таблиц операцией исключения записей.

Реорганизация таблиц может использоваться, например, для упорядочивания записей в таблице с целью ускорения поиска или для уплотнения записей в таблице.

Задание к лабораторной работе

Для выбранного языка программирования уяснить семантику понятий, для которых конструируется таблица. Язык программирования выбрать из предложенных, или тех, которых в списке нет (по желанию студента).

1. Определить структуру записи таблицы и ее связи с другими таблицами, а также операции над записями таблицы.

2. Выбрать и обосновать метод организации таблицы с учетом ее связей.

3. Реализовать таблицу и операции над ней с помощью инструментального языка программирования. Реализация работы с таблицей должна быть выполнена в виде нескольких процедур, соответствующих операциям над записями таблицы: поиск записи, добавление новой записи, чтение/занесение значения некоторого атрибута записи, исключение записи, реорганизация таблицы.

Для инициализации таблицы идентификаторов выбрать наиболее употребительные ключевые слова (15-25) языка, среди которых обязательно должны быть ключевые слова, связанные с описаниями (см. Варианты задании лабораторной работы).

Для инициализации таблицы процедур (подпрограмм) взять 5-8 функций.

Таблица операций должна содержать информацию о всех операциях языка программирования.

Таблицу типов необходимо проинициализировать всеми встроенными в язык типами.

Варианты заданий

1.2.1. Таблицы для транслятора с языка СИ

1) Таблица идентификаторов и ее инициализация.

2) Таблица типов и ее инициализация.

3) Таблица констант и ее инициализация.

4) Таблица операций и ее инициализация.

5) Таблица меток.

6) Таблица процедур и ее инициализация.

1.2.2. Таблицы для транслятора с языка ПАСКАЛЬ

7) Таблица идентификаторов и ее инициализация.

8) Таблица типов и ее инициализация.

9) Таблица констант и ее инициализация.

10) Таблица операций и ее инициализация.

11) Таблица меток,

12) Таблица процедур и ее инициализация.

Требование к содержанию отчета

1) Название лабораторной работы и задание.

2) Перечень конструкций выбранного языка программирования, определяющих и использующих понятие или объект, для которых конструируется таблица.

3) Структура записи таблицы и связи с другими таблицами, операции над записями таблицы.

4) Метод организации таблицы: неупорядоченная (простая или древовидная), упорядоченная, перемешанная (с открытым доступом или с цепочками переполнения).

5) Текст программ на инструментальном языке программирования, списывающих таблицу и реализующих операции над записями таблицы.

6) Набор тестовых данных и ожидаемых результатов контроля правильности программ.

Контрольные вопросы

1. Каким требованиям должны удовлетворять организация таблиц транслятора и реализация операций работы с таблицами?

2. Приведите примеры статических и динамических таблиц транслятора.

3. Какие методы ускорения поиска можно эффективно использовать для статических таблиц транслятора?

4. Какими методами можно ускорить поиск в динамических таблицах транслятора?

5. Предложите несколько хеш-функций для таблицы имен.

6. Сформулируйте достоинства и недостатки различных структур таблиц транслятора.

Литература

1. С. А. Орлов. "Технологии разработки программного обеспечения. Разработка сложных программных систем"., — СПб.: Питер, 2002. — 464 с.

2. Л.Г. Гагарина, Е.В. Кокорева, Б.Д. Виснадул, "Технология разработки программного обеспечения"., М.:, ИД "Форум", - 2008 г., 400 стр.

3. А. Ахо, Дж. Ульман., "Теория синтаксического анализа, перевода и компиляции", М.: Книга по Требованию, 2012. – 613 с.

4. Ахо А., Ренди С., Ульман Дж. Компиляция. – М.:Мир, 2002. – 834 с.

5. Орлов С.А., Программная инженерия. Учебник для вузов. 5-е издание обновленное и дополненное. Стандарт третьего поколения. Санкт-Петербург: Питер, 2016 г. , 640 с. Режим доступа: Электронный ресурс https://books.google.com.ua/books?id=_9ZLDAAAQBAJ&pg=PA2&lpg=PA2&dq=%D0%BA%D0%BD%D0%B8%D0%B3%D0%B0+%D0%BE%D1%80%D0%BB%D0%BE%D0%B2+%D1%81.%D0%B0.+%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%B0%D1%8F+%D0%B8%D0%BD%D0%B6%D0%B5%D0%BD%D0%B5%D1%80%D0%B8%D1%8F&source=bl&ots=PAdR1-r6SM&sig=SZM7wlc5QROeMwGoykWllnA6E2w&hl=ru&sa=X&ved=0ahUKEwjLpNDa7bPSAhXK6RQKHdauC0oQ6AEITzAI#v=onepage&q&f=false

Содержание

МЕТОДИЧЕСКИЕ УКАЗАНИЯ И ЗАДАНИЯ
к лабораторным работам по дисциплине

«Конструирование программного обеспечения»

(для студентов направления подготовки 09.03.04 “Программная инженерия”)

Составители: Алла Викторовна Чернышова