Оглавление

[1 Парадигмы программирования 1](#_Toc92917799)

[2 Популярность языков программирования 3](#_Toc92917800)

[3 Модели жизненного цикла программного проекта. Каскадная модель. 4](#_Toc92917801)

[4 Модели жизненного цикла программного проект. V-образная модель. 7](#_Toc92917802)

[5 Модели жизненного цикла программного проект. Спиральная модель. 7](#_Toc92917803)

[6 Модели жизненного цикла программного проект. Итерационная модель. 8](#_Toc92917804)

[7 Жизненный цикл программного проекта в стандарте ГОСТ 19.102-77 (ЕСПД) 9](#_Toc92917805)

[8 Качество. Основные понятия 12](#_Toc92917806)

[9 Измерительные шкалы 16](#_Toc92917807)

[10 Измерение качества ПО 19](#_Toc92917808)

[11 Качество ПО в стандартах 22](#_Toc92917809)

[12 Управление качеством ПО 23](#_Toc92917810)

[13 (14) Метрики оценки качества программ 25](#_Toc92917811)

[15 Модели надежности ПО 29](#_Toc92917812)

[16 Метрическая теория программ. Основные классификации 30](#_Toc92917813)

[17 Метрическая теория программ. Метрики Холстеда 32](#_Toc92917814)

[18 Метрическая теория программ. Метрика МакКейба 38](#_Toc92917815)

[19 Метрическая теория программ. Модели надежности 40](#_Toc92917816)

[Расчет метрики функциональных указателей 44](#_Toc92917817)

[Пример расчета метрик Холстеда 52](#_Toc92917818)

[Расчет метрики Мак-Кейба 57](#_Toc92917819)

[Решение задачи на применение эвристической модели Руднера. 64](#_Toc92917820)

[Решение задачи на применение модели Джилински-Моранды. 64](#_Toc92917821)

[Решение задачи на применение модели Милса. 68](#_Toc92917822)

1 Парадигмы программирования

Парадигм программирования - набор идей и понятий, которые определяют стиль написания программ, подход к программированию. Парадигма означает пример, образец.

Классификация парадигм языков программирования:

1) Директивные, называемые также процедурными или императивными, примерами их являются Algol, Fortran, Basic, Pascal, C;

2) Декларативные разделяются на:

Функциональное программирование - парадигма программирования, основанная на концепции трактовки процесса выполнения программы, как вычисление значений функций в математическом понимании (возвращаемое значение, без побочных эффектов, без изменения данных). Первый язык Lips.

Логические парадигмы. Первый язык Prolog.

3) Структурное программирование (появилось в 60-е годы ХХ века) - это парадигма программирования, основанная на концепции представления структуры любой программной системы в виде 4-х базовых управляющих конструкций:

1) последовательное исполнение - однократное выполнение операций в том порядке, в котором они записаны в тексте программы [последовательность действий]; 2) ветвление — однократное выполнение одной из операций, в зависимости от выполнения некоторого заданного условия [альтернатива – условие]; 3) цикл - многократное исполнение одной операции до тех пор, пока выполняется заданное условие [цикл повторения]; 4) вызов процедур (подпрограмм).

4) ООП

Наиболее популярная парадигма. Разработана в конце 60-х гг. в Норвегии. Основана на концепции представлении предметной области (и/или проблемной области) в виде системы взаимосвязанных абстрактных объектов (классов) и их реализаций (объектов – экземпляров классов).

В ООП основными элементами программы являются не переменные и методы, а объекты и сообщения.

Основные идеи ООП:

• программа представляет собой модель некоторого реального процесса;

• модель описана как совокупность взаимодействующих между собой объектов;

• объект описывается набором атрибутов и набором операций;

• взаимодействие между объектами осуществляется посылкой

специальных сообщений от одного объекта к другому (сообщение

может потребовать выполнения определенных действий);

• объекты, описанные одним и тем же набором атрибутов и

способные выполнять один и тот же набор операций, представляют

собой класс.

Три компонента ООП:

1) Инкапсуляция

2) Наследование

3) Полиморфизм

2 Популярность языков программирования

Популярность языков программирования согласно сайту tiobe.com.

2019 год: java C Python C++ C#

2020 год: C java Python C++ C#

Популярность языков программирования согласно сайту cnews на 27 августа 2021 года

Python 100 балов

Java 95,4 бала

C 94,7 бала

C++ 92,4 бала

JavaScript 88,1 бал

Популярность языков программирования согласно рейтингу TIOBE

2021 год

C 12%

Python 11%

Java 10%

C++ 7%

C# 5%

3 Модели жизненного цикла программного проекта. Каскадная модель.

В управления проектами, в том числе при разработке программного обеспечения, важнейшим является понятие жизненного цикла. Определяется как необходимая последовательность фаз, этапов, реализуемая исходя из потребностей задач управления, а также принципов перехода между ними.

Жизненный цикл проекта часто оформляется в виде модели:

1) Методология Института управления проектами.

Жизненный цикл проекта имеет 5 фаз:

1. Инициация

2. Планирование

3. Выполнение

4. Контроль и мониторинг

5. Завершение.

Основан в 1969 году.

PMI разработал и активно продвигает известный стандарт по управлению проектами (PMI представляет собой некоммерческую организацию, занимающуюся продвижением, пропагандой и развитием проектного менеджмента в разных странах).

2) Моделирование жизненного цикла проекта по принципу «водопада».

Каскадная модель жизненного цикла была предложена в 1970 г. Уинстоном Ройсом.

Она предусматривает последовательное выполнение всех этапов проекта в строго фиксированном порядке. Переход на следующий этап означает полное завершение работ на предыдущем этапе. Каждая стадия завершается выпуском полного комплекта документации, достаточной для того, чтобы разработка могла быть продолжена другой командой разработчиков.

3) Моделирование жизненного цикла проекта по итеративной модели

Итеративный подход - выполнение работ параллельно с непрерывным анализом полученных результатов и корректировкой предыдущих этапов работы. Пример реализации итеративного подхода — методология разработки программного обеспечения, созданная компанией Rational Software (RUP), а также MSF и XP.

4) Моделирование жизненного цикла проекта по спиральной модели.

Разработана в середине 1980-х годов Барри Боэмом. Каждая итерация соответствует созданию фрагмента или версии ПО, на ней уточняются цели и характеристики проекта, оценивается качество полученных результатов и планируются работы следующей итерации.

На каждой итерации оцениваются:

• риск превышения сроков и стоимости проекта;

• необходимость выполнения ещё одной итерации;

• степень полноты и точности понимания требований к системе;

• целесообразность прекращения проекта.

5) Моделирование жизненного цикла проекта инкрементным методом

Инкрементное построение: разбиение большого объема проектно-конструкторских работ на последовательность более малых составляющих частей. Сущность же метода заключается в том, что принятия решений на протяжении всего жизненного цикла разбиваются на множество очень мелких решений. Этот способ хорош в случаях, с большим количеством переменных влияющих на развитие проекта и позволяет делать управленческие ошибки без значительных потерь.

КАСКАДНАЯ МОДЕЛЬ

Каскадная модель жизненного цикла была предложена в 1970 году Уинстоном Райтом. Она предусматривает последовательное выполнение всех этапов проекта в строго

фиксированном порядке. Переход на следующий этап означает полное завершение работ на предыдущем этапе.

Свойства и характеристики, определенные на стадии формирования требований, строго документируются в виде технического задания и фиксируются на все время разработки проекта. Каждая стадия завершается выпуском полного комплекта документации, достаточной для того, чтобы разработка могла быть продолжена другой командой разработчиков.

Этапы проекта в соответствии с каскадной моделью:

1. Формирование требований

2. Проектирование

3. Реализация

4. Тестирование

5. Внедрение

6. Эксплуатация и сопровождение

Практическое использование этой модели выявило ее основной недостаток. Она в большей степени применима к общеинженерным разработкам.

Так же существует

V-образная модель

Была разработана с целью избавиться от недостатков Каскадной модели. Она ориентирована на тестирование и позволяет контролировать результат на соответствие ожиданиям с ранних этапов разработки.

Спиральная модель

Была разработана в середине 1980-х годов Барри Боэмом. Она основана на классическом цикле Деминга PDCA. При использовании этой модели ПО создается в несколько итераций (витков спирали) методом прототипирования.

Каждая итерация соответствует созданию фрагмента или версии ПО, на ней уточняются цели и характеристики проекта, оценивается качество полученных результатов и планируются работы следующей итерации.

Итерационная модель

Естественное развитие каскадной и спиральной моделей привело к их сближению и появлению современного итерационного подхода, который представляет рациональное сочетание этих моделей. Различные варианты итерационного подхода реализованы в большинстве современных технологий и методов (RUP, MSF, XP).

4 Модели жизненного цикла программного проект. V-образная модель.

Начало как в вопросе 4.

V-ОБРАЗНАЯ МОДЕЛЬ

Модель, которая называется V-образной, была разработана с целью избавиться от недостатков Каскадной модели. Она ориентирована на тестирование и позволяет контролировать результат на соответствие ожиданиям с ранних этапов разработки. V-образная модель позволяет гораздо лучше контролировать результат на предмет его соответствия ожиданиям, поскольку сфокусирована на тестировании.

Достоинства такого подхода реализуются за счет выполнения процедур аттестации и верификации на ранних фазах жизненного цикла разработки. На рисунке пунктирные линии указывают на зависимость между этапами планирования и постановки задачи с этапами тестирования и приемки, эксплуатации.

Недостатком V–образной модели является то, что она не позволяет учитывать изменяющиеся требования заказчика при разработках продолжительностью несколько лет.

5 Модели жизненного цикла программного проект. Спиральная модель.

Начало как в вопросе 4.

Спиральная модель была разработана в середине 1980-х годов Барри Боэмом. Она основана на классическом цикле Деминга PDCA. При

использовании этой модели ПО создается в несколько итераций (витков спирали) методом прототипирования.

Каждая итерация соответствует созданию фрагмента или версии ПО, на ней уточняются цели и характеристики проекта, оценивается качество полученных результатов и планируются работы следующей итерации.

На каждой итерации при спиральной модели оцениваются:

• риск превышения сроков и стоимости проекта;

• необходимость выполнения ещё одной итерации;

• степень полноты и точности понимания требований к системе;

• целесообразность прекращения проекта.

Один из примеров реализации спиральной модели — RAD (метод быстрой разработки приложений).

6 Модели жизненного цикла программного проект. Итерационная модель.

Начало как в вопросе 4.

Естественное развитие каскадной и спиральной моделей привело к их сближению и появлению современного итерационного подхода, который представляет рациональное сочетание этих моделей. Различные варианты итерационного подхода реализованы в большинстве современных технологий и методов (RUP, MSF, XP).

Полный жизненный цикл разработки продукта состоит из следующих четырех фаз, каждая из которых включает в себя одну или несколько итераций.

1. Начало;

2. Уточнение;

3. Построение;

4. Внедрение.

Изображение выглядит как стол

Автоматически созданное описание

7 Жизненный цикл программного проекта в стандарте ГОСТ 19.102-77 (ЕСПД)

1. Техническое задание

Этап 1.1. Обоснование необходимости

разработки программы

Содержание.

Постановка задачи.

Сбор исходных материалов.

Выбор и обоснование критериев

эффективности и качества разрабатываемой

программы.

Обоснование необходимости проведения научно-исследовательских работ.

Этап 1.2 Научно-исследовательские работы

Содержание.

Определение структуры входных и выходных данных.

Предварительный выбор методов решения задач.

Обоснование целесообразности применения ранее разработанных программ.

Определение требований к техническим средствам.

Обоснование принципиальной возможности решения поставленной задачи

Этап 1.3 Разработка и утверждение

технического задания

Определение требований к программе.

Разработка технико-экономического

обоснования разработки программы.

Определение стадий, этапов и сроков

разработки программы и документации на

неё.

Выбор языков программирования.

Определение необходимости проведения

научно-исследовательских работ на

последующих стадиях.

Согласование и утверждение технического

задания.

2. Эскизный проект.

Этап 2.1 Разработка эскизного проекта.

Содержание.

Предварительная разработка структуры входных и выходных данных.

Уточнение методов решения задачи.

Разработка общего описания алгоритма решения задачи.

Разработка технико-экономического обоснования.

Этап 2.2 Утверждение эскизного проекта

Содержание.

Разработка пояснительной записки.

Согласование и утверждение эскизного проекта.

3. Технический проект

Этап 3.1 Разработка технического проекта.

Содержание.

Уточнение структуры входных и выходных данных.

Разработка алгоритма решения задачи.

Определение формы представления входных и выходных данных.

Определение семантики и синтаксиса языка.

Разработка структуры программы.

Окончательное определение конфигурации технических средств.

Этап 3.2 Утверждение технического проекта

Содержание.

Разработка плана мероприятий по разработке и внедрению программ.

Разработка пояснительной записки.

Согласование и утверждение технического проекта.

4. Рабочий проект

Этап 4.1 Разработка программы

Содержание.

Программирование и отладка программы.

Этап 4.2 Разработка программной документации

Содержание.

Разработка, согласование и утверждение порядка и методики испытаний.

Проведение предварительных государственных, межведомственных, приёмо-сдаточных и других видов испытаний.

Корректировка программы и программной документации по результатам испытаний.

Этап 4.3 Испытания программы

Содержание.

Разработка, согласование и утверждение порядка и методики испытаний.

Проведение предварительных государственных, межведомственных, приёмо-сдаточных и других видов испытаний.

Корректировка программы и программной документации по результатам испытаний.

5. Внедрение

Этап 5.1 Подготовка и передача программ.

Содержание.

Подготовка и передача программы и программной документации для сопровождения и (или) изготовления.

Оформление и утверждение акта о передаче программы на сопровождение и (или) изготовление.

Передача программ в фонд алгоритмов и Программ

8 Качество. Основные понятия

Определение КАЧЕСТВА

(ISO): Качество — это полнота свойств и характеристик продукта, процесса или услуги, которые обеспечивают способность удовлетворять заявленным или подразумеваемым потребностям.

(IEEE): Качество программного обеспечения - это степень, в которой оно обладает требуемой комбинацией свойств.

До рассмотрения вопроса «Оценка качества ПО» приведем ряд важных определений Программное обеспечение (ПО) может представлять:

программу;

программный комплекс;

программное средство;

программный продукт (изделие).

Приведем их определения из разных стандартов

ГОСТ 19781-90 ОБЕСПЕЧЕНИЕ СИСТЕМ ОБРАБОТКИ ИНФОРМАЦИИ ПРОГРАММНОЕ

Термины и определения

Software of data processing systems. Terms and definitions

Программа (Program) - данные, предназначенные для управления конкретными компонентами системы обработки информации в целях реализации определенного алгоритма.

Программное обеспечение - совокупность программ системы обработки информации и программных документов, необходимых для эксплуатации этих программ.

ГОСТ 28806-90

КАЧЕСТВО ПРОГРАММНЫХ СРЕДСТВ

Термины и определения

Software quality. Terms and definitions

Программа (en program): - см. ГОСТ 19781.

Программный продукт (en software product):

Программное средство, предназначенное для поставки, передачи, продажи пользователю.

ГОСТ 28806-90

КАЧЕСТВО ПРОГРАММНЫХ СРЕДСТВ

Термины и определения

Software quality. Terms and definitions

Программное средство; ПС (en software):

Объект, состоящий из программ, процедур, правил, а также, если предусмотрено, сопутствующих им документации и данных, относящихся к функционированию системы обработки информации. Примечание. Программное средство представляет собой конкретную информацию, объективно существующую как совокупность всех значимых с точки зрения ее представления свойств каждого из материальных объектов, содержащих в фиксированном виде эту информацию.

ИСО 2382

Термины и определения стандарта ISO/IEC 2382-1

Программное обеспечение - интеллектуальный продукт, состоящий из программ, процедур, правил и любой другой связанной с ними документации, относящихся к функционированию системы обработки данных. 01.01.08. Программное обеспечение (Софтвер) Любая часть Программ (01.05.01), Процедур (15), правил и документации для Систем обработки информации (01.01.12).

Прим.: Софтвер является результатом интеллектуальной деятельности, не зависящим от средства, на котором он записан.

Программа - алгоритм, записанный в форме, воспринимаемой вычислительной машиной. [ГОСТ 19.004-80].

Программное обеспечение - совокупность программ и документов на них для реализации целей и задач цифровых электронных вычислительных машин. [ГОСТ 19.004-80].

Программное обеспечение (ПО) – интеллектуальный продукт, состоящий из программ, процедур, правил и любой другой связанной с ними документации, относящихся к функционированию системы обработки данных. [ИСО 2382-1].

Продукция программного обеспечения – полный набор компьютерных программ, процедур и связанный сними документации и информации, предназначенный для поставки пользователю. [ИСО 9000-3-91].

Программная продукция - результат процесса разработки ПО, т.е. ПО, выпускаемое для использования. [ГОСТ Р ИСО/МЭК 9294-93].

Терминов из отечественных и международных стандартов, связанных с программным обеспечением Алгоритм - точное предписание, определяющее вычислительный процесс, ведущий от варьируемых начальных данных к искомому результату. [ГОСТ 19.004-80].

Элемент программного обеспечения - какая-либо идентифицируемая часть программного обеспечения на промежуточном или конечном этапе разработки. [ИСО 9000-3-91].

Алгоритм, записанный в понятном для компьютера виде и предназначенный для запуска ее автором.

ДАДИМ НЕОБХОДИМЫЕ ОПРЕДЕЛЕНИЯ СВЯЗАННЫЕ С КАЧЕСТВОМ ПО

Качество программного обеспечения – это совокупность свойств, характеризующих способность программного обеспечения удовлетворять потребностям пользователя в соответствии с предназначением. Управление качеством – это система организационных, экономических, технологических и правовых мероприятий, осуществляемых для удовлетворения требований к качеству программного обеспечения в течение жизненного цикла.

Свойства программы – это особенности, объективно присущие программе, которые проявляются в ее жизненном цикле (разработке, применении, сопровождении).

Характеристика программы – это понятие, отражающее проявление отдельного измеримого фактора присущего программе свойства. Иначе говоря, характеристика – это проявляемый и измеримый атрибут свойства

Система измерений характеристик программного обеспечения – это совокупность измеряемых характеристик, единиц измерения, измерительных шкал и связей, установленных между ними. Если между измеряемыми характеристиками установлены иерархические связи, систему измерений называют иерархической, в противном случае – одноранговой.

Измерительная шкала устанавливает границы (диапазон) и точность измерений характеристик свойств в установленных единицах.

9 Измерительные шкалы

При оценке качества продукта, товара используются разнообразные ИЗМЕРИТЕЛЬНЫЕ ШКАЛЫ. Они предполагают разнообразные МЕТОДЫ ОЦЕНКИ (формирования) характеристик.

В целом методы оценки характеристик программного обеспечения делят на следующие семь групп:

измерительные,

регистрационные,

органолептические,

расчетные,

экспертные,

социологические,

традиционные.

Измерительные методы основаны на получении информации о характеристиках ПО с использованием специальных инструментальных средств (технических или программных средств, обеспечивающих проведение измерений и их автоматизацию).

Регистрационные методы основаны на получении информации о характеристиках ПО во время испытаний или функционирования путем регистрации и подсчета определенных событий (например, моментов и количества ошибок, времени начала и окончания расчетов и т.д.), регистрируемых извне программы с помощью средств измерений общего назначения.

Органолептические методы основаны на получении информации о характеристиках программного обеспечения путем их восприятия органами чувств – в первую очередь зрения, слуха и осязания.

Расчетные методы основаны на получении информации о характеристиках программного обеспечения за счет использования теоретических или эмпирических зависимостей

Экспертные методы используют опыт экспертов-специалистов, компетентных в оценке характеристик программного обеспечения.

Социологические методы используют обработку специальных анкет-опросников, содержащих качественные оценки характеристик программного обеспечения социальными группами, имеющими отношение к применению программного обеспечения.

Традиционные методы объединяют группу сформировавшихся и традиционно используемых в организациях, на предприятиях и иных учреждениях методов количественной оценки характеристик программного обеспечения.

Применительно к ПО используют (чаще, главным образом) следующие три вида измерительных шкал:

номинальные (категорийные),

порядковые,

интервальные.

Номинальная шкала (nominal scale) — это условная схема маркировки, где числа служат исключительно как ярлыки или метки для определения и классификации объектов. Например, номера, присваиваемые респондентам в процессе исследования, составляют номинальную шкалу. При использовании номинальной шкалы для определения объектов существует строгое соответствие, — один к одному, между номерами и объектами. Пример - номера спортсменов. В шкале наименований нельзя сказать, что один объект больше или меньше другого, на сколько единиц они различаются и во сколько раз.

Википедия

Шкала наименований (номинальная,

классификационная)

Используется для измерения значений качественных признаков. Значением такого признака является наименование класса эквивалентности, к которому принадлежит рассматриваемый объект. Примерами значений качественных признаков являются названия государств, цвета, марки автомобилей и т. п. Такие признаки удовлетворяют аксиомам тождества:

Либо А = В, либо А ≠ В;

Если А = В, то В = А;

Если А = В и В = С, то А = С.

Википедия

При большом числе классов используют иерархические шкалы наименований. Наиболее известными примерами таких шкал являются шкалы, используемые для классификации животных и растений.

С величинами, измеряемыми в шкале наименований, можно выполнять только одну операцию — проверку их совпадения или несовпадения.

Порядковая (ранговая) шкала – отображение отношений порядка. Единственно возможные отношения между объектами измерения в данной шкале – это больше/меньше, лучше/хуже. Простейшим примером являются оценки знаний учащихся.

Порядковая шкала указывает лишь последовательность носителей признака или направление степени выраженности признака.

Википедия

Порядковая шкала (или ранговая)

Строится на отношении тождества и порядка. Субъекты в данной шкале ранжированы. Но не все объекты можно подчинить отношению порядка. Например, нельзя сказать, что больше круг или треугольник, но можно выделить в этих объектах общее свойство-площадь, и таким образом становится легче установить порядковые отношения. Для данной шкалы допустимо монотонное преобразование. Такая шкала груба, потому что не учитывает разность между субъектами шкалы. Пример такой шкалы: балльные оценки успеваемости (неудовлетворительно, удовлетворительно, хорошо, отлично), шкала Мооса.

Интервальная шкала помимо отношений, указанных для шкал наименований и порядка, отображает отношение расстояний (разностей) между парами объектов. Для нее допустимым является положительное линейное преобразование. Шкалы Цельсия и Фаренгейта, с помощью которых измеряется физическая температура, представляют собой примеры интернатных шкал.

Интервальная шкала (она же Шкала разностей)

Здесь происходит сравнение с эталоном. Построение такой шкалы позволяет большую часть свойств существующих числовых систем приписывать числам, полученным на основе субъективных оценок. Например, построение шкалы интервалов для реакций. Для данной шкалы допустимым является линейное преобразование. Это позволяет приводить результаты тестирования к общим шкалам и осуществлять, таким образом сравнение показателей. Пример: шкала Цельсия.

Начало отсчёта произвольно, единица измерения задана. Допустимые преобразования — сдвиги. Пример: измерение времени.

10 Измерение качества ПО

Анализ всех составляющих качества должен проводится с учетом сфер ответственности заинтересованных сторон, как внутренних участников исполняемого процесса, так и пользователей процесса.

Проявляемые (измеряемые) свойства программного обеспечения условно можно разделить на две группы:

1) функциональные (внешние),

2) конструктивные (внутренние).

Пользователя, прежде всего, интересуют функциональные свойства.

Разработчиков интересуют как внешние, так и внутренние, или конструктивные свойства.

Пользователя (заказчика) интересуют свойства, характеризующие полезность ПО.



Полезность ПО:

1. Исходная полезность

1.1 Надежность

1.2 Эффективность

1.3 Корректность

2. Удобство применения

2.1 Понятность

2.2 Простота

2.3 Масштабируемость



11 Качество ПО в стандартах

**Показатель качества** – численное значение критерия качества, определяющее степень, в которой программе присуще определенное критерием свойство.

В соответствии с ГОСТ 15467-79 под **показателем качества** следует понимать количественную характеристику одного или нескольких свойств программной продукции, составляющих ее качество применительно к определенным условиям ее создания и эксплуатации.

**Базовое значение показателя качества** – значение показателя качества продукции, принятое за основу при сравнительной оценке ее качества.

– это реально достижимое значение показателя, отражающее современный уровень развития программного обеспечения.

**Комплексный показатель качества** – показатель качества продукции, характеризующий несколько ее свойств (ГОСТ).

- показатель качества, значение которого получают в результате композиции значений других, в том числе комплексных показателей.

**Характеристики качества** отражают свойства, определяющие качество программного обеспечения. Для количественной оценки характеристик качества ПО используют иерархические системы измерений. Иерархию характеристик качества образуют факторы, критерии, метрики и оценочные элементы.

**Факторы и критерии** отражают функциональные характеристики.

**Метрики и оценочные элементы** отражают конструктивные характеристики

**Фактором качества** будем называть свойство, в той или иной степени обуславливающее качество программного обеспечения. При оценке качества учитывают несколько факторов. Для получения численной оценки фактора качества используют один или несколько критериев качества.

**Критерий качества** – это понятие, признак или численный показатель, характеризующий оцениваемый фактор качества. Он является выражением, составленным из характеристик качества, и имеющим физический смысл значением которого является показатель качества. Для вычисления значения критерия используют одну или несколько метрик.

**Метрика** – мера количественной оценки качества программного обеспечения по заданному критерию, система или способ измерений качества программного обеспечения. Метрика содержит один или несколько оценочных элементов.

**Оценочный элемент –** измеримая характеристика программного обеспечения, имеющая численное значение в избранной измерительной шкале.

12 Управление качеством ПО

**Оценкой качества программного обеспечения** называют совокупность операций, включающих:  
1) выбор номенклатуры (состава) показателей качества;  
2) определения значений этих показателей;   
3) сравнения их с базовыми значениями.

С этой оценкой связаны три процедуры.

1. Сертификация

2. Верификация.

3. Аттестация.

**Сертификацией** называют процесс определения соответствия программного обеспечения действующему стандарту качества.

**Верификацией** называют процесс определения соответствия программного обеспечения предназначению**.**

**Аттестацией** называют процесс подтверждения функциональной пригодности программного обеспечения.

Концепция управления качеством ПО:

1.Требования к уровню качества по каждому фактору определяют базовым значением показателя качества

2.Требуемый уровень качества обеспечивается процессом и в процессе производства.

3.Измерение, оценка и контроль уровня качества производится на всех стадиях жизненного цикла.

4.Управление качеством есть непрерывный, информационный и целенаправленный процесс воздействия на программы и документацию, а также на коллективы разработчиков ПС в целях обеспечения требуемого качества при изменяющихся внешних и внутренних условиях путем принятия управленческих решений.

Сущность управления качеством ПО:

1. Обеспечение качества. Определение множества организационных процедур и стан­дартов в целях создания ПО высокого качества.

2. Планирование качества. Выбор из этого множества соответствующего подмножества процедур и стандартов и адаптация их к данному проекту разработки ПО.

3. Контроль качества. Определение и проведение мероприятий, гарантирующих выполнение нормативных процедур и стандартов качества всеми членами команды разработчиков ПО.

Виды нормативных документов (применительно к ПО)

* стандарты,
* технические условия,
* своды правил,
* регламенты,
* положения

**Стандарт** — это нормативный документ, разработанный на основе консенсуса, утвержденный признанным органом, направленный **на** достижение оптимальной степени упорядочения в определенной области.

В стандарте устанавливаются для всеобщего и многократного использования общие принципы, правила, характеристики рекомендательного характера, касающиеся различных видов деятельности или их результатов.

Роль стандартизации в управлении качеством ПО

Стандартизация представляет собой процесс установления и применения правил с целью упорядочения в данной области на пользу и при участии всех заинтересованных сторон.

В процессе стандартизации вырабатываются нормы, правила, требования, характеристики, касающиеся объекта стандартизации, которые оформляются в виде нормативного документа.

13 (14) Метрики оценки качества программ

**Метрика** – мера количественной оценки качества программного обеспечения по заданному критерию, система или способ измерений качества программного обеспечения. Метрика содержит один или несколько оценочных элементов.

По классу методологий, которые используют для разработки программ, модели определения значений их характеристик можно разбить на две группы:

* процедурно-ориентированные,
* объектно-ориентированные.

1.Процедурно-ориентированные метрики

Первая метрика - количество строк кода (SLOC)

Количество строк кода (SLOC) — это метрика программного обеспечения, используемая для измерения его объёма с помощью подсчёта количества строк в тексте исходного кода.

Использует регистрационно-расчетный метод

Вторая (так же регистрационно-расчетными методами) - метрика дефектов качества



Чем больше DQ, тем ниже уровень качества. При этом предполагается соблюдение нормы 0 ≤ DQ ≤ 1.

Достоинства SLOC и производных метрик

• широко распространены;

• просты и легко вычисляются.

Недостатки расчета SLOC

* зависимы от языка программирования;
* требуют исходных данных, которые трудно получить на начальной стадии проекта;
* не приспособлены к непроцедурным языкам программирования.

Другой метод подсчета метрики дефектов - косвенный, Алан Альбрехт 1997 год.

На первом этапе вычисляется величина, называемая количеством функциональных указателей FP.

Далее вычисляют – Качество (DQ) = Ошибки / FP

Еще вычисляются

Производительность = ФункцУказатель / Затраты (FP/чел.-мес.);

Удельная Стоимость = Стоимость / ФункцУказатель (Тыс.$/FP);

Документированность= СтраницДокумента/ФункцУказатель (Страниц/FP)

Недостаток метода – СУБЪЕКТИВНОСТЬ

Опишем порядок вычисления метрики Альбрехта:

В зависимости от типа программного продукта используются пять или шесть типов информационных характеристик.

1. Количество внешних вводов.

2. Количество внешних выводов.

3. Количество внешних запросов.

4. Количество внутренних логических файлов.

5. Количество внешних интерфейсных файлов.

6. Количество внешних алгоритмов.

На первом этапе заполняются таблицы.



На втором этапе - вычисления количество функциональных указателей состоит в присваивание одного из пяти значений четырнадцати системным параметрам.

На втором этапе - вычисление количество функциональных указателей состоит в присваивание одного из **пяти** значений **четырнадцати** системным параметрам:  
1- случайное;   
2- небольшое;   
3- среднее;   
4 – важное;   
5 – основное.

Для расчета количества функциональных указателей используются следующие оценочные элементы:

* количество внешних вводов (вводов данных пользователем ***f1***),
* количество внешних выводов данных (отчеты, экраны, распечатки, сообщения ***f2***),
* количество внешних запросов (диалоговых вводов-выводов ***f3***),
* количество локальных файлов (наборов базы данных ***f4***),
* количество разделяемых с другими программами глобальных файлов (наборов базы данных ***f5***).

**Количество функциональных указателей** рассчитывают по формуле



где *F*– общее количество функциональных указателей:



2. Объектно-ориентированные метрики

1) развивают процедурно – ориентированный подход оценивания сложных программных систем, основанный на анализе связности и сцепления его компонент;

2) учитывают специфические особенности объектно-ориентированных решений разработки программного обеспечения.

К основным объектно-ориентированным метрикам относят

* метрики связности классов;
* метрики сцепления классов;
* комплексные наборы метрик сложности и качества ПС.

Примером комплексного набор являются метрики Лорентца и Кидда

Их набор состоит из 10 метрик, разделенных на три группы:

1) **метрики классов** (четыре метрики);

2) **характеристик операций** (три метрики);

3) **характеристик процесса разработки** (три метрики)

При этом первая группа **метрик классов** содержит:

* **CS-метрику** (Class Size), или метрику размера класса;
* **NOO-метрику** (Number of Operations Overridden by a Subclass), или количество операций, переопределяемых подклассом;
* **NOA-метрику** (Number of Operation Added by a Subclass), или количество операций, добавленных подклассом;
* **SI-метрику** (Specialization Index), или индекс специализации.

Вторая группа **характеристик операций** содержит:

* OC-метрику (Operation Complexity), или метрику сложности операции (метода);
* AOS-метрику (Average Operation Size), или метрику среднего размера операции (метода);
* ANP-метрику (Average Number of Parameters per Operation), или метрику среднего количества параметров на операцию.

И третья группа **характеристик процесса разработки** содержит:

* NSS-метрику (Number of Scenario Scripts), или метрику количества описаний сценариев;
* NKC-метрику (Number of Key Classes), или метрику количества ключевых классов;
* NSUB-метрику (Number of Subsystem), или метрику количества подсистем.

15 Модели надежности ПО

Сначала приведем общую схему моделей определения значений характеристик программ. Их можно классифицировать на 4 группы, связанные

1. Методология разработки
2. Система измерений
3. Назначения модели
4. Метод определения значений

В третьей группе такие классы Комплексной оценки качества;

* Оценки сложности;
* Оценки корректности;
* Оценки надежности;
* Оценки трудоемкости.

В свою очередь модели надежности делят:

1) от **назначения** модели надежности делят на такие:

* прогнозные,
* оценочные,
* измерительные.

2) по **содержанию** на такие:

* математические
* эмпирические.



16 Метрическая теория программ. Основные классификации

За более чем семидесятилетнюю историю ПРОГРАММИРОВАНИЯ разработано большое количество моделей и метрик, оценивающих отдельные производственные и эксплуатационные свойства ПО.

Метрическую теорию программ представляют различные математические модели определения численных значений характеристик программного обеспечения, в том числе характеристик качества.

Начало теории положил М. Холстедом.

Метрики используют, главным образом, для прогнозирования, оценки сложности и корректности программ.

По типу используемой информации о программах в моделях различают метрики четырех типов

1. Основанные на лексическом анализе программ.
2. Основанные на анализе потока управления.
3. Основанные на анализе внутримодульных и межмодульных связей.
4. Основанные на анализе потока данных.

Исторически первыми появились математические модели, представляющие метрики программ, основанных на анализе лексики (1) и потока управления программ (2), реализующих заданный алгоритм.

Типичными представителями таких метрик являются метрики Холстеда и Маккейба.

По типу получаемой информации о метриках программ эти модели можно разбить на следующие группы:

* оценивающие отклонение от нормы,
* прогнозирующие значения характеристик,
* формирующие принятие решений о соответствии программного обеспечения заданным требованиям

Есть и другие подходы к классификации моделей и метрик оценки программ. Одна их них вводит такие классы моделей.

1. Метрики сложности.

Эта группа моделей наиболее многочисленна и разработана. Поэтому в ней рассматривают свои подгруппы.

Первая подгруппа определяется словарными метриками, основанных на метрических соотношениях Холстеда, цикломатических мерах Мак-Кейба и измерениях Тейера.

Вторая подгруппа ориентирована на метрики связей, отражающих сложность отношений между компонентами системы - это метрики Уина и Винчестера.

Третья подгруппа включает семантические метрики, связанные с архитектурным построением программ и их оформлением.

1. Прогнозные модели.

Этот класс моделей формируется на этапе проектирования, и они позволяют рассчитать характеристики надежности программного средства до начала его отладки

1. Оценочные модели.

Эти модели строятся на основе анализа результатов тестирования программ. Они позволяют на основе полученных значений характеристик надежности принимать решение о необходимости продолжать процедуру тестирования программного средства. К этой групп относятся такие модели: Джелинского—Моранды, Миллса и Простая Интуитивная или Эвристическая модель двух независимых групп тестирования Руднера.

Другая классификация метрической теории программ

**Математические модели:**

По методологии разработки

По назначению

По системе измерений

По числу определяемых значений

**По методологии разработки**

1.Процедурно-ориентированные

2. Объектно-ориентированные

**По системе измерений**

1. Нормативные

2. Экспериментальные

3. Иерархические

**По назначению**

1. Оценка сложности

2. Оценка трудоемкости разработки

3. Оценка надежности

4. По числу определяемых значений

**По числу определяемых значений**

1. Измерительные

2. Расчетные

3. Регистрационные

4. Экспертные

**Рабочий Математические модели**

На основе лексического анализа программ

На основе алгоритмической сложности

Процедурно-ориентированные метрики

Объектно-ориентированные метрики

Метрики надежности программ

Исторически первыми появились математические модели, представляющие метрики программ, основанных на анализе лексики. Типичными представителями таких метрик являются метрики Холстеда.

17 Метрическая теория программ. Метрики Холстеда

Метрики Холстеда

Метрики Холстеда отражают лексический подход к измерению характеристик программного обеспечения, основанный на измеримых свойствах алгоритмов. Они используют такие метрические характеристики (оценочные элементы):

n1 - количество различных операторов программы;

n2 - количество различных операндов программы;

N1 - общее количество операторов программы;

N2 - общее количество операндов программы.

На их основе определяются метрики

словарь программы (в условных единицах)

n = n1+n2,

длина реализации (в условных единицах)

N = N1+N2

длина программы (в условных единицах)

Ñ = (n1 × log2 n1)+(n2×log2 × n2),

объем программы (в битах)

V = (N1+N2) × log2(n1+n2).

потенциальный объем программы

V\* = (n2\*+2) × log2(n2\*+2),

где n2\* - общее число входных и выходных

параметров

Потенциальный объем программы является мерой минимально необходимого объема программы с заданным словарем. При этом потенциальный объем не зависит от языка реализации.

При переводе программы с одного языка на другой потенциальный объем не меняется, но действительный объем V или увеличивается, или уменьшается в зависимости от языка реализации.

Используя выражение для потенциального объема программы, Холстедом получены следующие метрики:

уровень программы (в условных единицах)

L = V\*/ V ≅ (2×n2)/(n1× N2),

уровень языка

λ=L×V\*,

интеллектуальное содержание программы (в

условных единицах)

I=L×V≅(2×n2/n1× N2)×(N1+N2)×log2(n1+n2)

работа по программированию (в условных

единицах)

E = V/L = V2/V\*

время на программирование (в условных

единицах)

T = E/S, если n2\*=2, или Т≅(n1×N2×log2n×

(n1×log2n1+n2×log2n2))/(2×n2×S),

S – число Страуда.

Уровень программы L≤1 характеризует эффективность реализации алгоритма

относительно затрат памяти. Только для наиболее сжатой формы реализации алгоритма (V=V\*) уровень программы имеет значение 1. Всем другим вариантам реализации соответствуют значение L<1.

Уровень языка λ – это коэффициент пропорциональности изменения объема программы при переводе с одного языка на другой так, что обеспечивается постоянство произведения уровня программы на потенциальный объем.

Интеллектуальное содержание характеризует меру «сказанного» в программе, или ее «информативность». Интеллектуальное содержание (уровень) программы сильно коррелирует с потенциальным объемом (L≈V\*) и тоже не зависит от языка реализации

Число Страуда S определяется как число «страудовских моментов» в секунду. «Страудовский момент» - это время, необходимое человеку для выполнения элементарного различения объектов, подобно различению кадров фильма. Страуд обнаружил, что человек способен различать от 5 до 20 объектов в секунду

Работа по программированию (уравнение мысленной работы) характеризует величину умственной работы, связанной с написанием программного кода. Так как сумма квадратов двух величин всегда меньше квадрата их суммы, уравнение работы дает основание для разбиения программы на составные части – модули. Модульность снижает работу по программированию. Исследования возможностей оперативного мышления человека дают основания считать, что наиболее продуктивна ситуация, при которой для получения одного результата используется не более пяти объектов. В прикладном отношении этот результат называют гипотезой о «шести объектах».

Для определения количества модулей M в программе Холстед рекомендует использовать выражение

M= n2\*/6,

где n2\* – общее количество входных и выходных переменных в программе

Холстедом получено следующее уравнение ошибок в программе

B=L×E/E0,

где В – количество ошибок в программе, Е0 – среднее число элементарных отличий между возможными ошибками программирования. Используя преобразованное уравнение работы

Е= (V\*)3/λ2,

значение уровня английского языка (λ=2,16) в качестве аналога языка программирования и гипотезу о «шести объектах» идеальной по затратам памяти программы

(n1=n1\*=2, n2=n2\*=6), Холстед вывел следующее уравнение для прогноза количества ошибок в программе:

В=Е 2/3 /3000,

или

В=V/3000,

где V – объем программы.

Кроме своего прямого назначения в практическом отношении метрики длины программы и длины реализации можно использовать для выявления несовершенств программирования.

Если расчеты длины программы и длины реализации отличаются более чем на десять процентов, то это свидетельствует о возможном наличии в программе следующих шести классов несовершенств:

1. Наличие последовательности дополняющих друг друга операторов к одному и тому же операнду, например, А+C-А.

2. Наличие неоднозначных операндов, например, A=D и A=С.

3. Наличие синонимичных операндов, например, А=В и Т=В.

4. Наличие общих подвыражений, например (А+В)×С+D×(А+В).

5. Ненужное присваивание, например С=А+В, если переменная С используется в программе только один раз.

6. Наличие выражений, которые не представлены в свернутом виде как произведение множителей, например X×X+2×X×Y+Y×Y не представлено как (X+Y)×(X+Y).

Длину реализации N можно использовать для прогноза числа фактических машинных команд P с помощью выражения

N=(8/3)\*P

или более грубо, с помощью неравенства

2×P≤ N ≤ 4×P.

Уравнение работы можно использовать для оценки экономической эффективности использования того или иного языка программирования. Относительное сокращение работы по программированию в зависимости от уровня языка используют как показатель эффективности внедрения языка программирования в производственную практику.

Уровень программы 0<L≤1 можно использовать для оценки сложности вариантов реализации заданного алгоритма D (чем меньше затрат

памяти, тем сложнее вариант программы): Отметим, что уровень программы играет двойственную роль в оценке трудности или легкости ее понимания. Специалист, который хорошо знает язык программирования, поймет программу тем быстрее, чем меньше ее объем, то есть выше уровень. Но для человека, менее разбирающегося в программировании, требуется больший объем и меньший уровень.

Установлено,

1) что для любого алгоритма, описанного разными языками, с увеличением объема программы V уровень программы L уменьшается в той же пропорции. Поэтому произведение уровня программы на объем является постоянной величиной, равной потенциальному объему реализации данного алгоритма

L×V=V\*=const

2) если язык не меняется, а меняется только алгоритм, то для любого языка произведение потенциального объема на уровень программы остается постоянной величиной, равной уровню языка

L×V=λ=const

4.2. О понятиях операторы, операции и операнды.

До того, как начинать описание методики Холстеда на конкретных примерах необходимо разобраться в значениях терминов операторы, операции и операнды.

Исходный текст программы, записываемой на том или ином языке программировании представляет собой набор текстовых строк, которые записываются по специальным правилам и в том числе имеет свои элементы. К настоящему времени разработано огромное количество языков программирования. На странице Википедии, посвященной этому вопросу указано, что их разработано уже несколько тысяч и приводится более десятка их типов. Терминология, используемая для описания отдельных элементов языка (конструкций), разнообразна. Приведем описание части терминов, применяемое в этом разделе к трем языкам программирования (С, С++ и С#). Рассматриваемые далее примеры используют только такие средства.

Операторы и операции предполагают реализацию каких-то действий. Если рассматривать программу как набор операторов, то операции являются их составляющими.

В рассматриваемых языках программирования добавление к набору символов, являющихся выражением, знака «;» превращает его в оператор. Например, набор символов i++ является выражением, а такая конструкция i++; уже является оператором (или statement по терминологии Кернигана и Ритчи).

Перечень операторов и операций, используемых в книгах авторов языков программирования С и С++, приведены далее. Отметим отдельно конструкции, связанные со скобками: круглыми, фигурными и квадратными.

Одним из действий, которые могут быть записаны в исходном тексте программ – это вызов функции. Формат его записи такой

Имя()

В словаре операторов и операций, которые будут формироваться далее в примерах, будем учитывать вызов функции при появлении пары круглых скобок и следующим перед ним именем. Обращаем внимание, что не надо путать пару круглых скобок, используемых со многими языковыми конструкциями, например операторами if, for и т.д. При формировании словаря операторов и операций учитываются еще и фигурные скобки. Они, казалось бы, не вызывают никаких действий, но связаны с ограничением области видимости переменных. Операции, которые связаны с квадратными скобками – это обращение к элементам массива. Действия, связанные с их интерпретацией – это доступ к элементам последнего.

Операторы и операции связаны с объектами, над которыми они выполняются. Таким объектами являются, прежде всего, константы, простые переменные, массивы и структуры. Но подобными объектами (операндами) являются и функции, а так же классы и методы.

18 Метрическая теория программ. Метрика МакКейба

В разделе предшествующем описанию методики Холстеда была приведена классификация моделей программометрики. В группе расчетных моделей вводился класс «По методу используемой информации» одна из подгрупп имела название «Поток управления». Представителем этого класса является модель Мак-Кейба. Она, как и методика Холстеда, исторически появилась одной из первых среди моделей метрической теории программ.

Для вычисления характеристики сложности программы по Мак-Кейбу следует построить граф потока управления. Приведем его определение:

«Граф потока управления (англ. control flow graph, CFG) — в теории компиляции — множество всех возможных путей исполнения программы, представленное в виде графа.

В графе потока управления каждый узел (вершина) графа соответствует базовому блоку — прямолинейному участку кода, не содержащего в себе ни операций передачи управления, ни точек, на которое управление передается из других частей программы. Имеется лишь 2 исключения: точка, на которую выполняется переход, является первой инструкцией в базовом блоке, и базовый блок завершается инструкцией перехода. Направленные дуги используются в графе для представления инструкций перехода. Также, в большинстве реализаций добавлено два специализированных блока: входной блок, через который управление входит в граф и выходной блок, который завершает все пути в данном графе.»

Граф потока управления программы — это ориентированный граф, которой позволяет учесть логику программы, анализируя поток передачи управления между операторами. Этот граф формируется из вершин, которые соответствуют операторам программы, и дуг, задающих переход между этими операторами.

Многие программы можно рассматривать как описание того, как входные значения, подаваемые в начальных операторах программы, преобразуются в выходные значения, которые выводятся перед завершением работы программы. Это можно условно изобразить так.

Программа

Вход

Выход

Рис.5.1. Программа – преобразование входных значений в выходные.

В зависимости от подаваемых на вход значений будут выполняться разные операторы программы. Можно говорить о разных маршрутах выполнения операторов программы.

Программа

Вход

Выход

Рис.5.2. Преобразование входных значений в выходные может осуществляться по разным маршрутам исполнения операторов программы.

Рассмотрим, как строятся графы управления программы для типовых блоков, реализуемых основными операторами языков программирования.

При построении графа управления программы надо учитывать только исполнимые операторы, оставляя не учтенными операторы описания. Если в программе несколько операторов, не меняющих порядок действий, следуют один за другим, их можно объединить в одну вершину графа.

При построении такого графа операторы цикла заменяют несколькими вершинами. Например, для оператора типа for должны быть вершины соответствующие начальному присваиванию счетчика цикла, его изменению и ветвлению, определяющему продолжать или закончить выполнение цикла. Аналогично нескольким вершинами заменяют и другие операторы цикла. Они должны соответствовать действиям предшествующим циклу, определяющим продолжать или прекращать выполнение цикла, а так же группе операторов, составляющих тело цикла – множеству повторяемых действий. Таким образом, в графе должны быть три группы вершин, определяющих три элемента составляющих любой цикл.

19 Метрическая теория программ. Модели надежности

Race condition — ошибка проектирования многопоточной системы или приложения, при которой работа системы или приложения зависит от того, в каком порядке выполняются части кода. Race condition — это нежелательная ситуация, которая возникает, когда устройство или система пытается выполнить две или более операций одновременно, но из-за природы устройства или системы, операции должны выполняться в правильной последовательности, чтобы быть выполненными правильно.

Программометрика -

1) наука о количественном оценивании свойств программного обеспечения.

2) (метрическая теория программ) - научное и прикладное направление информатики в области проектирования и управления разработкой программных средств информационных систем (ПС ИС)

Программная метрика - мера, позволяющая получить численное значение некоторого свойства программного обеспечения или его спецификации.

Меры программного обеспечения многочисленны.

Их анализ позволяют обеспечить КАЧЕСТВО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ В практике их оценивают ХАРАКТЕРИСТИКАМИ

Приведем классификацию Характеристики качества ПО

- Функциональность (Functionality)

- Надежность (Reliability)

- Практичность (Usability)

- Эффективность (Efficiences)

- Сопровождаемость (Maintainability)

- Мобильность (Portability)

Характеристики вычисляются на основе различных моделей.

Приведем классификацию моделей Программометрики.

ОЦЕНКА КАЧЕСТВА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Классификация моделей Программометрики.

Четыре критерия классификации.

1. По методологии разработки.

2. По системе измерений.

3. По назначению.

4. По методу определения численных значений.

1. По методологии разработки.

1.1 Процедурно-ориентированные.

1.2 Объектно-ориентированыые.

2. По системе измерений.

2.1. Нормативные

2.2. Экспериментальные

2.3. Иерархические

3. По назначению.

3.1. Оценка сложности

3.2. Оценка трудоемкости

разработки

3.3. Оценка надежности

3.4. Комплексная оценка качества

4. По методу определения

численных значений.

4.1. Измерительные.

4.2. Расчетный.

4.3. Регистрационный.

4.4. Экспертный.

Вторая классификация моделей программометрики (классификация 2)

Классификация 2

Критерии

А. На основе лексического анализа программ.

Б. На основе алгоритмической сложности.

В. Процедурно-ориентированные метрики.

Г. Объектно-ориентированные метрики.

Д. Метрики надежности программ.

ПРИМЕРЫ

А. На основе лексического анализа программ (Холстед).

Б. На основе алгоритмической сложности (МакКейб).

В. Процедурно-ориентированные метрики (На основе функциональных указателей)

Д. Метрики надежности программ.

Д1. Джилински-Моранды

Д2. Миллса

Д3. Эвристическая модель

Д4. Нельсона

Д5. Муса

Классификация моделей надежности

Модели надежности: аналитические (Статические, Динамические), эмпирические.

Д1. Джилински-Моранды

Модель надежности программного обеспечения Джелински-Моранды основана на методе максимального правдоподобия. Она относиться к классу оценочных моделей и применяется на этапе тестирования.

Д2. Миллса

1) Относиться относится к классу оценочных.

2) По другой классификации эта модель относится к классу аналитических статических моделей. Ее применение предполагает внесение искусственных ошибок перед началом тестирования.

Предполагается равновероятное нахождение как внесенных в программу

ошибок, так и имеющихся в ней ранее можно получить полезное соотношение

N = (S×n)/V, где

N — первоначальное число ошибок в программе,

S — количество искусственно внесенных ошибок,

n — число найденных собственных ошибок,

V — число обнаруженных к моменту оценки искусственных ошибок.

Д3. Эвристическая модель

Простая интуитивная или эвристическая модель двух независимых групп тестирования Руднера относится к типу «разметка ошибок» (искусственное внесение в программное обеспечение известных ошибок). Она относится к группе оценочных, но исключается основной недостаток модели Миллса – предположение о том, собственные ошибки, имеющиеся в программе и искусственно вносимые при тестировании имеют одинаковую вероятность. Для этой модели используются данные в предположении, что тестирование осуществляется двумя независимыми группами.

Расчет метрики функциональных указателей

|  |  |
| --- | --- |
| Формула | Единицы измерения |
| Производительность = Длина / Затраты | (тыс.LOC/чел.-мес.); |
| Качество = Ошибки / Длина | (Единиц/тыс. LOC); |
| Удельная стоимость = Стоимость /Длина | (тыс.$/LOC); |
| Документированность = Страниц\_Документа / Длина | (Страниц/тыс.LOC) |

FP= Общее количество строк\*(0,65+0,01\*Sfi).

Рассчитав эту величину можно вычислить другие оценки

|  |  |
| --- | --- |
| Формула | Единицы измерения |
| Производительность = ФункцУказатель / Затраты | (FP/чел.-мес.); |
| Удельная Стоимость = Стоимость / ФункцУказатель | (Тыс.$/FP); |
| Документированность=СтраницДокумента/ФункцУказатель | (Страниц/FP) |

* + - * 1. Табл 1. Оценка сложности внешних вводов.

|  |  |  |  |
| --- | --- | --- | --- |
| Количество ссылок на файлы | Количество элементов данных | | |
| 1-4 | 5-15 | >15 |
| 0-1 | Низкий | Низкий | Средний |
| 2 | Низкий | Средний | Высокий |
| >2 | Средний | Высокий | Высокий |

* + - * 1. Табл 2. Оценка сложности внешних выводов.

|  |  |  |  |
| --- | --- | --- | --- |
| Ссылки на файлы | Количество элементов данных | | |
| 1-4 | 5-19 | >19 |
| 0-1 | Низкий | Низкий | Средний |
| 2-3 | Низкий | Средний | Высокий |
| >3 | Средний | Высокий | Высокий |

Табл. 3.Оценка сложности внешних запросов.

|  |  |  |  |
| --- | --- | --- | --- |
| Ссылки на файлы | Количество элементов данных | | |
| 1-4 | 5-19 | >19 |
| 0-1 | Низкий | Низкий | Средний |
| 2-3 | Низкий | Средний | Высокий |
| >3 | Средний | Высокий | Высокий |

Табл. 4. Оценка сложности внутренних логических файлов

|  |  |  |  |
| --- | --- | --- | --- |
| Ссылки на файлы | Количество элементов данных | | |
| 1-19 | 20-50 | >50 |
| 0-1 | Низкий | Низкий | Средний |
| 2-5 | Низкий | Средний | Высокий |
| >5 | Средний | Высокий | Высокий |

Табл. 5. Оценка сложности внешних интерфейсных файлов

|  |  |  |  |
| --- | --- | --- | --- |
| Ссылки на файлы | Количество элементов данных | | |
| 1-19 | 20-50 | >50 |
| 0-1 | Низкий | Низкий | Средний |
| 2-5 | Низкий | Средний | Высокий |
| >5 | Средний | Высокий | Высокий |

Подсчитанное для каждой информационной характеристики количество элементов каждого типа заносится в такую общую таблицу.

Табл. 6. Данные для расчета FP-метрики.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя характеристики | Количество с учетом сложности | | | |
| Низкий | Средний | Высокий | Итого |
| Внешние вводы | m\*3=\_ | m\*4=\_ | m\*6=\_ |  |
| Внешние выводы | m\*4=\_ | m\*5=\_ | m\*7=\_ |  |
| Внешние запросы | m\*3=\_ | m\*4=\_ | m\*6=\_ |  |
| Внутренние логические файлы | m\*7=\_ | m\*10=\_ | m\*15=\_ |  |
| Внешние интерфейсные файлы | m\*5=\_ | m\*7=\_ | m\*10=\_ |  |
| Общее количество | | | |  |

В этой таблице m – количество элементов фиксированного значения сложности для каждого информационного элемента. После вычисления значения в каждой клетке умножением m на ранг, суммируются значения каждой строки в ячейки столбца «Итого». И, наконец, просуммировав значения последнего столбца, будем иметь необходимое для расчета FP «Общее количество».

Второй шаг вычисления количество функциональных указателей состоит в присваивание одного из пяти значений четырнадцати системным параметрам. Такие значения, называемые коэффициентов регулирования сложности субъективно расставляются выбором одного из следующих значений [Черников Б.В. Управление качеством программного обеспечения. – М.; ИД «ФОРУМ»: ИНФРА-м, 2012]:

1- случайное;

2- небольшое;

3- среднее;

4 – важное;

5 – основное.

Табл. 7. Коэффициенты регулирования сложности

|  |  |  |
| --- | --- | --- |
| N | Вопрос для системного параметра | Обозна-чение |
| 1 | Какое влияние имеет наличие средств передачи данных? | f1 |
| 2 | Какое влияние имеет распределенная обработка данных? | f2 |
| 3 | Какое влияние имеет распространенность используемой аппаратной платформы? | f3 |
| 4 | Какое влияние имеет критичность к требованиям производительности и ограничению времени ответа? | f4 |
| 5 | Какое влияние имеет частота транзакций? | f5 |
| 6 | Какое влияние имеет ввод данных в режиме реального времени? | f6 |
| 7 | Какое влияние имеет эффективность работы конечного пользователя? | f7 |
| 8 | Какое влияние имеет оперативное обновление локальных файлов в режиме реального времени? | f8 |
| 9 | Какое влияние имеет скорость обработки данных (вычислений)? | f9 |
| 10 | Какое влияние имеет количество и категории пользователей? | f10 |
| 11 | Какое влияние имеет легкость инсталляции? | f11 |
| 12 | Какое влияние имеет легкость эксплуатации? | f12 |
| 13 | Какое влияние имеет разнообразие условия применения? | f13 |
| 14 | Какое влияние имеет простота внесения изменений? | f14 |

И после выполнения двух, описанных выше шагов, можно вычислить количество функциональных указателей FP. И далее другие оценки.

1) Производительность.

2) Удельная Стоимость.

3) Документированность.

Формулы для их вычисления были приведены выше.

Пример

Вторая программа, для которой рассчитана функционально-ориентированная метка – «Вертикальная змейка» по первому алгоритму. Вот ее текст, реализованный на языке программирования C#. Будем называть ее Программа 2.

*using System; 1*

*using System.Collections.Generic; 2*

*using System.Linq; 3*

*using System.Text; 4*

*5*

*namespace ConsoleApplication2 6*

*{ 7*

*class Program 8*

*{ 9*

*static void Main(string[] args) 10*

*{ 11*

*int N=4; int M=4; int N\_M=16; int i, j, k; 12*

*int[,] a = { {11, 12, 13, 14}, {21, 22, 23, 24},*

*{31, 32, 33, 34}, {41, 42, 43, 44} }; 13*

*int[] b; 14*

*b = new int[N\_M]; 15*

*16*

*k = 0; 17*

*for(j=0; j<M; j=j+2) 18*

*{ 19*

*for (i=0; i<N; i++) 20*

*{ 21*

*b[k]=a[i,j]; k=k+1; 22*

*} 23*

*k = k + M; 24*

*} 25*

*k = M; 26*

*for(j=1; j<M; j=j+2) 27*

*{ 28*

*for (i=N-1; i>=0; i--) 29*

*{ 30*

*b[k]=a[i,j]; k=k+1; 31*

*} 32*

*k = k + M; 33*

*} 34*

*for(k=0;k<N\_M; k++) 35*

*{ 36*

*Console.WriteLine("{0:d} ", b[k] ); 37*

*} 38*

*Console.ReadKey(); 39*

*} 40*

*} 41*

*} 42*

***1. Количество внешних вводов***. СРЕДННЕ – 1.

*Console.ReadKey(); 39*

***2. Количество внешних выводов***. СРЕДНЕЕ – 1.

*Console.WriteLine("{0:d} ", b[k] ); 37*

***3. Количество внешних запросов***. 0

НЕТ.

***4. Количество внутренних логических файлов***. 0

НЕТ

***5. Количество внешних интерфейсных файлов.***СРЕДНЕЕ – 1.

*b = new int[N\_M]; 15*

Табл.10. Данные для расчета FP-метрики Программы 2.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя характеристики | Количество с учетом сложности | | | |
| Низкий | Средний | Высокий | Итого |
| Внешние вводы | 1\*3= 3 | 0\*4= 9 | 0\*6= 0 | 3 |
| Внешние выводы | 1\*4= 1 | 0\*5= 0 | 0\*7= 0 | 4 |
| Внешние запросы | 0\*3= 0 | 0\*4= 0 | 0\*6= 0 | 0 |
| Внутренние логические файлы | 0\*7= 0 | 0\*10= 0 | 0\*15 = 0 |  |
| Внешние интерфейсные файлы | 1\*5= 1 | 0\*7= 9 | 0\*10= 0 | 5 |
| Общее количество | | | | 12 |

Коэффициенты регулирования сложности для Программы 2 сведены в такую таблицу.

Табл.11. Коэффициенты регулирования сложности Программы 2.

|  |  |  |
| --- | --- | --- |
| N | Вопрос для системного параметра | Обозна-чение |
| 1 | Какое влияние имеет наличие средств передачи данных? | f1 = 3 |
| 2 | Какое влияние имеет распределенная обработка данных? | f2 = 1 |
| 3 | Какое влияние имеет распространенность используемой аппаратной платформы? | f3 = 1 |
| 4 | Какое влияние имеет критичность к требованиям производительности и ограничению времени ответа? | f4 = 1 |
| 5 | Какое влияние имеет частота транзакций? | f5 = 1 |
| 6 | Какое влияние имеет ввод данных в режиме реального времени? | f6 = 1 |
| 7 | Какое влияние имеет эффективность работы конечного пользователя? | f7 = 5 |
| 8 | Какое влияние имеет оперативное обновление локальных файлов в режиме реального времени? | f8 = 1 |
| 9 | Какое влияние имеет скорость обработки данных (вычислений)? | f9 = 1 |
| 10 | Какое влияние имеет количество и категории пользователей? | f10 = 1 |
| 11 | Какое влияние имеет легкость инсталляции? | f11 = 1 |
| 12 | Какое влияние имеет легкость эксплуатации? | f12 = 1 |
| 13 | Какое влияние имеет разнообразие условия применения? | f13 = 1 |
| 14 | Какое влияние имеет простота внесения изменений? | f14 = 1 |

Поясним расставленные в таблице значения. Отличные от случайного влияния имеют 11 показателей (все кроме имеющих номера 1, 6 и 7).

Для показателя 1 определим влияние как среднее – требуется ввод данных.

Для показателя 6 определим влияние как среднее – требуется ввод данных.

Для показателя 7 определим влияние как среднее – влияние существенное, так как программа требует ввода данных и пока пользователь их не введет, программа ожидает ввода и не продолжает работу

Их общая сумма будет равняться (3+1+1+1+1+3+5+1+1+1+1+1+1+1)=22

И после выполнения двух, описанных выше шагов, можно вычислить количество функциональных указателей по следующей формуле:

FP= Общее количество×(0,65+0,01×Sfi) = 12×(0,65+0,01×22) =10,44.

Можно сделать вывод, что данная программа является даже более простой, чем первая, так как у нее еще более невысокий показатель FP.

Пример расчета метрик Холстеда

Текст программы. Далее приводится реализация программы, решающая такую же задачу, как и программы 1 и 2. Отладка программы выполнялась в среде разработки Microsoft Visual С# Express Edition, 2008. Был выбран проект типа Консольное Приложение (Console Application).

Текст программы ПХ.3.

*using System; 1*

*using System.Collections.Generic; 2*

*using System.Linq; 3*

*using System.Text; 4*

*namespace ConsoleApplication1 5*

*{ 6*

*class Program 7*

*{ 8*

*static void Main(string[] args) 9*

*{ 10*

*int a, c, d, k; int[] r = new int[6]; 11*

*Console.WriteLine(" Input a"); 12*

*a = int.Parse(Console.ReadLine()); 13*

*if ( (a<1000000) && (a>1000) ) 14*

*{ 15*

*c=a; 16*

*k=5; 17*

*while (c>=1) 18*

*{ 19*

*d=c/10; 20*

*r[k]=(c-d\*10); 21*

*c=d; 22*

*k=k-1; 23*

*} 24*

*if (r[0]+r[1]+r[2] == r[3]+r[4]+r[5]) 25*

*Console.WriteLine("Happy"); 26*

*еlse 27*

*Console.WriteLine("No happy "); 28*

*} 29*

*else 30*

*{ 31*

*Console.WriteLine("Bad data"); 32*

*} 33*

*Console.ReadKey(); 34*

*} 35*

*} 36*

*} 37*

Приведенные далее таблицы, содержащие словари операторов, операций и операндов аналогичны тому, что было для программ 1 и 2.

Табл.13. Словарь операторов и операций программы

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Операторы и операции | Номера строк | Количество повторений |
| 1 | { | 6(37); 8(36); 10(35); 15(29); 19(24); 31(33) | 6 |
| 2 | . | 12; 13; 13; 26; 28; 32; 34 | 7 |
| 3 | () | 9; 12; 13; 13; 26; 28; 32; 34 | 8 |
| 4 | ; | 11; 11; 12; 13; 16; 17; 20; 21; 22; 23; 26; 28; 32; 34 | 14 |
| 5 | = | 11; 13; 16; 17; 20; 21; 22; 23 | 8 |
| 6 | if | 14; 25 | 2 |
| 7 | < | 14 | 1 |
| 8 | && | 14 | 1 |
| 9 | > | 14 | 1 |
| 10 | while | 18 | 1 |
| 11 | >= | 18 | 1 |
| 12 | / | 20 | 1 |
| 13 | - | 21, 23 | 2 |
| 14 | \* | 21 | 1 |
| 15 | + | 25, 25, 25, 25 | 4 |
| 16 | == | 25 | 1 |
| 17 | [] | 9; 11; 11; 21; 25; 25; 25; 25; 25; 25 | 10 |
| 18 | static | 9 | 1 |
| 19 | void | 9 | 1 |
| 20 | string | 9 | 1 |
| 21 | int | 9 | 3 |
| 22 | new | 11 | 1 |
| 23 | namespace | 5 | 1 |
| 24 | class | 7 | 1 |
|  |  |  | 78 |

А так будут выглядеть список операндов программы 3, реализованную с использованием языка С#.

Табл.14. Словарь операндов программы

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Операнды | Номера строк | Количество повторений |
| 1 | Console | 12; 13; 26; 28; 32; 34 | 6 |
| 2 | WriteLine | 12; 26; 28; 32 | 4 |
| 3 | " Input a" | 12 | 1 |
| 4 | а | 11; 13; 14; 14; 16 | 5 |
| 5 | int | 13 | 1 |
| 6 | Parse | 13 | 1 |
| 7 | ReadLine | 13 | 1 |
| 8 | 1000000 | 14 | 1 |
| 9 | 1000 | 14 | 1 |
| 10 | c | 11; 16; 18; 20; 21; 22 | 6 |
| 11 | k | 11; 17; 21; 23; 23 | 5 |
| 12 | 5 | 17, 25 | 2 |
| 13 | 0 | 18 | 1 |
| 14 | d | 11; 20; 21; 22 | 4 |
| 15 | 10 | 20, 21 | 2 |
| 16 | r | 11; 21; 25; 25; 25; 25; 25; 25 | 8 |
| 17 | 1 | 18; 23; 25 | 3 |
| 18 | 2 | 25 | 1 |
| 19 | 3 | 25 | 1 |
| 20 | 4 | 25 | 1 |
| 21 | "Happy" | 26 | 1 |
| 22 | "No happy " | 28 | 1 |
| 23 | "Bad data" | 32 | 1 |
| 25 | ConsoleApplication1 | 5 | 1 |
| 26 | Program | 7 | 1 |
| 27 | main | 9 | 1 |
| 28 | args | 9 | 1 |
| 29 | 6 | 11 | 1 |
| 30 | ReadKey | 34 | 1 |
|  |  |  | 64 |

Список входных и выходных параметров. Он такой же, как и для 1 и 2 программы.

Расчет метрик Холстеда. Для третьей программы, характеристики, вычисляемые по методике Холстеда, будут такими.

Табл.15. Значения метрик Холстеда для программы

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование характеристики | Обозначение и формула для вычисления | Значения | |
| Число простых (уникальных) операторов и операций | h1 | 24 | |
| Число простых (уникальных) операндов | h2 | 30 | |
| Общее число всех операторов и операций | N1 | 78 | |
| Общее число всех операндов | N2 | 64 | |
| Число входных и выходных переменных (параметров) | h2\* | 3 | |
| Словарь программы | h=h1+h2 | 54 | |
| Длина реализации программы | N=N1+N2 | 142 |
| Длина программы | Ñ = (h1×log2h1)+(h2×log2h2) | 257,2 | |
| Объем программы (в битах) | V =(N1+N2)×log2(h1+h2) ) | 817,2 | |
| Теоретическая длина реализации программы | Nтеоретич. = 2×h2×log2h2 | 294,4 | |
| Теоретическое число простых (уникальных) операндов | h2теоретич. = h2\*×log2h2\* | 4,75 | |
| Потенциальный объем программы | V\* = (h2+2) ×log2(h2+2) | 11,6 | |
| Уровень реализации программы | L = V\*/V | 0,0142 | |
| Наилучшее количество модулей | kopt=h2\*/log22×h2\* | 1,16 | |
| Работа программирования | E=V/L | 57522 | |
| Уровень реализации языка | l=L2×V\* | 0,1649 | |
| Квалификационное время программирования | T=E/S (S=18) | 3196 | |

Расчет метрики Мак-Кейба

**Программа 2**

Условия задачи и описание алгоритма. Преобразовать двумерный массив в одномерный, используя обход элементов «вертикальной змейкой». Эта программа и алгоритм уже были подробно описаны в разделе посвящено методике Холстеда. В алгоритме программы 4 используются два цикла. Первый заполняет результирующий массив из нечетных столбцов таблицы в естественном, возрастающем порядке номеров строк (цикл записан в строке 18 программы). Второй цикл заполняет результирующий массив из четных столбцов таблицы в обратном, убывающем порядке номеров строк (строка программы с номером содержит такой цикл). Обращаем внимание, что в языке С# номера строк и столбцов начинаются не с 1, а с 0.

Текст программы вертикальная «змейка».

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace ConsoleApplication2

{

class Program

{

static void Main(string[] args)

{

int[,] a = { {11, 12, 13, 14}, {21, 22, 23, 24}, {31, 32, 33, 34}, {41, 42, 43, 44} };

int[] b;

b = new int[16];

int N=4; int M=4; int i, j, k;

k=0; 1

for(i=0; i<N; i=i+2) 2

{

for (j=0; j<M; j++) 3

{

b[k]=a[i,j]; k=k+1; 4

}

k=k+M; 5

}

k = M; 6

for(i=0; i<N; i=i+2) 7

{

for (j=M-1; j>=0; j--) 8

{

b[k]=a[i,j]; k=k+1; 9

}

k=k+M; 10

}

for(k=0;k<N\*M; k++) 11

{

Console.WriteLine("{0:d} ", b[k] ); 12

}

Console.ReadKey(); 13

}

}

}

Приведем текст этой программы с заменой операторов цикла.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace ConsoleApplication2

{

class Program

{

static void Main(string[] args)

{

int N = 4; int M = 4; int N\_M = 16; int i, j, k;

int[,] a = { {11, 12, 13, 14}, {21, 22, 23, 24},

{31, 32, 33, 34}, {41, 42, 43, 44} };

int[] b;

b = new int[N\_M];

k = 0; // 1

j = 0; // 2

one: i = 0; // 3

two: b[k] = a[i, j]; k = k + 1; i = i + 1; // 4

if (i < N) goto two; // 5

k = k + M; j = j + 2; // 6

if (j < M) goto one; // 7

k = M; // 8

j = 1; // 9

one1: i = N - 1; // 10

two1: b[k] = a[i, j]; k = k + 1; i = i - 1; // 11

if (i >=0) goto two1; // 12

k = k + M; j = j + 2; // 13

if (j < M) goto one1; // 14

k = 0; // 15

pr: Console.WriteLine("{0:d} ", b[k]); k = k + 1; // 16

if (k < N\_M) goto pr; // 17

Console.ReadKey(); // 18

}

}

}

Построим граф сложности для программы («Вертикальная змейка 1» С##). Отметим, что циклы в этой программе заменяются как в схеме цикла с постусловием.

154

165

17

18

2

3

1

4

5

6

7

8

10

11

12

9

135

14

Рис.5.13. Граф сложности программы Х4.

Заметим, что в этом графе 18 вершин и 22 ребера, а значит цикломатическое число Маккейба равно Z = m – n + 2 = 22 – 18 + 2 = 6.

Для полученного графа сформируем матрицу смежности вершин.

Табл. 3. Матрица смежности вершин.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 2 | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 3 |  | 1 |  |  |  |  | 1 |  |  |  |  |  |  |  |  |  |  |  |
| 4 |  |  | 1 |  | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 5 |  |  |  | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 6 |  |  |  |  | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 7 |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  |  |  |  |  |
| 8 |  |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  |  |  |  |
| 9 |  |  |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  |  |  |
| 10 |  |  |  |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  |  |
| 11 |  |  |  |  |  |  |  |  |  | 1 |  | 1 |  |  |  |  |  |  |
| 12 |  |  |  |  |  |  |  |  |  |  | 1 |  |  |  |  |  |  |  |
| 13 |  |  |  |  |  |  |  |  |  |  |  | 1 |  |  |  |  |  |  |
| 14 |  |  |  |  |  |  |  |  |  |  |  |  | 1 |  |  |  |  |  |
| 15 |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 |  |  |  |  |
| 16 |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 |  | 1 |  |
| 17 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 |  |  |
| 18 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 |  |

По рисунку построим матрицу достижимости вершин. Она строится из таких соображений. Из вершины 1 доступны все вершины. Но эта вершина не достижима не для какой-то другой вершины графа.

Табл.4. Матрица достижимости вершин.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 2 | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 3 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |  |  |  |  |  |  |  |  |  |  |  |
| 4 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |  |  |  |  |  |  |  |  |  |  |  |
| 5 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |  |  |  |  |  |  |  |  |  |  |  |
| 6 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |  |  |  |  |  |  |  |  |  |  |  |
| 7 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |  |  |  |  |  |  |  |  |  |  |  |
| 8 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |  |  |  |  |  |  |  |  |  |  |  |
| 9 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |  |  |  |  |  |  |  |  |  |  |
| 10 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |  |  |  |  |
| 11 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |  |  |  |  |
| 12 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |  |  |  |  |
| 13 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |  |  |  |  |
| 14 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |  |  |  |  |
| 15 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |  |
| 16 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |  |  |
| 17 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |  |  |
| 18 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |  |

Решение задачи на применение эвристической модели Руднера.

Простая интуитивная или эвристическая модель двух независимых групп тестирования Руднера относится к типу «разметка ошибок» (искусственное внесение в программное обеспечение известных ошибок). Она относится к группе оценочных, но исключается основной недостаток модели Миллса – предположение о том, собственные ошибки, имеющиеся в программе и искусственно вносимые при тестировании имеют одинаковую вероятность.

Для этой модели используются данные в предположении, что тестирование осуществляется двумя независимыми группами.

Рассмотрим пример.

Предположим, что две независимые группы тестировщиков нашли в программе 40 и 30 ошибок соответственно. При этом оказалось, что 10 ошибок общие, их нашли обе группы. Требуется найти общее количество ошибок в программе, до начала тестирования

Имеются такие исходные данные:

-количество ошибок обнаруженных первой независимой группой тестировшиков (N1=40);

-количество ошибок обнаруженных второй независимой группой тестировшиков (N2=20);

-количество ошибок обнаруженных как первой, так и второй группой тестировшиков (N12=10).



Формуле общего числа ошибок (N) в программе N = 80 ошибок.

Из них не обнаружено 80 – 40 – 20 + 10 = 30 ошибок.

Решение задачи на применение модели Джилински-Моранды.

Модель надежности программного обеспечения Джелински-Моранды основана на методе максимального правдоподобия. Она относиться к классу оценочных моделей и применяется на этапе тестирования.

Ме́тод максима́льного правдоподо́бия или метод наибольшего правдоподобия (ММП, ML, MLE — англ. maximum likelihood estimation) в математической статистике — это метод оценивания неизвестного параметра путём максимизации функции правдоподобия[1].

Основан на предположении о том, что вся информация о статистической выборке содержится в функции правдоподобия.

Например, вы интересуетесь таким антропометрическим параметром, как рост жителей России. Предположим, у вас имеются данные о росте некоторого количества людей, а не всего населения. Кроме того, предполагается, что рост является нормально распределённой величиной с неизвестной дисперсией и средним значением. Среднее значение и дисперсия роста в выборке являются максимально правдоподобными к среднему значению и дисперсии всего населения.

В основе модели положены следующие предпосылки:

а) Экспоненциальная зависимость между плотностью вероятности интервалов времени между появлением ошибок.

б) Интенсивность ошибок линейно зависит от количества оставшихся ошибок (на любом случайном интервале).

в) После каждого появления устраняют ошибку и не вносят новую.

г) Каждый тест находит только одну ошибку.

Еще одно условие применимости модели Джелински-Моранды – это соответствие результатов тестирования допущению об уменьшении интенсивности ошибок после устранения очередной ошибки, то есть количество тестов (интервал времени для обнаружения каждой последующей ошибки) увеличивается.

Для принятых допущений интенсивность λ ошибок для интервала i можно вычислить из соотношения

λ(ti) = (N – i + 1)×k.

Отметим, что продолжительность интервалов тестирования измеряют в количестве тестов, а не во временных единицах.

Для плотности вероятности ошибки р(ti) на случайном интервале ti справедливо такое соотношение.

р(ti)=(N–i+1)×k×(ехр(N – i + 1)).

Применяя метод максимального правдоподобия для двух неизвестных этого уравнения N и k, получим следующее соотношение из двух уравнений



Модель Джелински-Моранды позволяет оценить количество ошибок в программе до начала тестирования по результатам тестирования – количеству обнаруженных ошибок (обозначено n). Используются обозначения:

N – общее количество ошибок в программе;

k - коэффициент пропорциональности в соотношении для вычисления интенсивности ошибок,

i – номер теста, выевшего ошибку,

ti - продолжительность интервалов тестирования.

Приведем расчет по модели Джилински-Моранды.

Условия



Проведено 7 тестов, ошибки обнаружены на 3м и 7м тесте.

По условию задачи количество тестов семь, а две ошибки (n=2) обнаружены третьим и седьмым тестом (t1=3, t2=4). Как видим, соблюдаются условия возрастания времени обнаружения ошибок (t2>t1).

Подставим в него значения, Запишем уравнение





Проведено 7 тестов, ошибки обнаружены на 2м и 7м тесте.



Решив эту систему, получим серию соотношений

(2N-1)(7N-4)=14(N2-N),

14N2-7N-8N+4=14N2-14N,

-N+4=0

из которых получим сначала N=4, а затем k=1/12.

Таким образом, всего в программе имеется 4 ошибки, а не обнаруженными осталось две ошибки (N-2=4-2=2).

Решение задачи на применение модели Милса.

1) Относиться относится к классу оценочных.

2) По другой классификации эта модель относится к классу аналитических статических моделей.

Ее применение предполагает внесение искусственных ошибок перед началом тестирования.

Предполагается равновероятное нахождение как внесенных в программу ошибок, так и имеющихся в ней ранее можно получить полезное соотношение

N = (S×n)/V, где

N — первоначальное число ошибок в программе,

S — количество искусственно внесенных ошибок,

n — число найденных собственных ошибок,

V — число обнаруженных к моменту оценки искусственных ошибок.