РОССИЙСКАЯ АКАДЕМИЯ НАУК
СИБИРСКОЕ ОТДЕЛЕНИЕ
ИНСТИТУТ СИСТЕМ ИНФОРМАТИКИ
им. А. П. Ершова

На правах рукописи

Черноножкин Сергей Константинович

Методы и инструменты метрической поддержки разработки качественных программ

Специальность 05.13.11 – математическое и программное обеспечение вычислительных машин, комплексов, систем и сетей

Автореферат

диссертации на соискание учёной степени
кандидата физико-математических наук

Новосибирск 1998

Работа выполнена в Институте ситем информатики имени А.П. Ершова Сибирского отделения Российской академии наук (ИСИ СО РАН).

Научный руководитель: *И.В.Поттосин*,
доктор физико-математических наук,
профессор

Официальные оппоненты: *В.А. Евстигнеев*,
доктор физико-математических наук,
профессор,
В.И. Гололобов,
кандидат технических наук

Ведущая организация: Институт технической кибернетики
АН Республики Беларусь

Защита состоится 12 февраля 1999 г. в 14 час. 30 мин. на заседании диссертационного совета К0003.93.01 в Институте систем информатики Сибирского отделения РАН по адресу:

630090, Новосибирск, пр. Акад. Лаврентьева, 6.

С диссертацией можно ознакомиться в читальном зале библиотеки ВЦ СО РАН (пр. Акад. Лаврентьева, 6).

Автореферат разослан

Учёный секретарь
диссертационного совета
К0003.93.01
к.ф.-м.н. М. А. Бульонков

Актуальность темы. Проблема создания качественного программного обеспечения появилась одновременно с рождением самого программирования. Бурное развитие вычислительной техники и ее внедрение практически во все сферы нашей деятельности только обострили данную проблему. Конечно, наука не стояла на месте и много сделано в области разработки методов создания качественных программ. Однако наша повседневняя жизнь показывает — данная проблема не решена и необходимы дальнейшие исследования в данной области.

В рамках данной работы предлагается (развивается) подход по метрическому контролю за качеством создаваемого продукта, что особенно важно, так как при определенных условиях можно обеспечить контроль и запрет на использование некачественных программ.

Цель диссертационной работы заключается в разработке моделей и на их основе инструментов для метрического контроля за качеством создаваемого программного продукта, причем рассматриваются не все показатели качества, а только два из них — надежность и эффективность. Для обеспечения надежности предлагается модель сложности и комплексный критерий тестирования.

Научная новизна работы состоит в следующем:

- разработана модель мер сложности для Модула-2-программ, которая включает меры для процедуры, модуля определений и модуля реализаций, и позволяет оценить сложность данного модуля и принять необходимые решения по дальнейшему его использованию, разработана методика применения получаемых результатов;
- разработана концепция средств профилирования (сбора динамических характеристик программ) для ассемблерных и Модула-2-программ, основным ее отличием от существующих является понятие частичного профиля программы;
- предложен способ формального описания критериев тестирования с помощью понятий: множество требуемых элементов и покрытия данного множества, разработан комплексный критерий тестирования для Модула-2/Оберон-2-программ, включающий семь критериев структурного тестирования, три из которых предложены и сформулированы впервые; разработан простой язык описания тестовых условий для их описания и задания во вновь сформулированных критериях;
- изучена проблема построения минимального дугового покрытия для ассемблерных и Модула-2-программ, разработан алгоритм его построения.

Практическая ценность работы состоит в разработке ряда программных систем: метрических компиляторов для ассемблера ЕС ЭВМ и Модула-2, профилировщиков для них, TGS — системы поддержки структурного тестирования и компонента оценки тестированности для нее, ОСТ — системы контроля тестированности Модула-2-программ. Все системы разрабатывались в рамках договора с НПО ПМ и переданы заказчику.

Апробация работы и публикации. Результаты работы неоднократно докладывались и обсуждались на объединенном семинаре ИСИ СО РАН и НГУ "Системное программирование" с участием ведущих специалистов.

Методы оценки тестированности программ были представлены на Третьем Сибирском конгрессе по прикладной и индустриальной математике (ИНПРИМ-98) в секции "Теория и методология создания систем информатики". Работа поддерживалась грантами РФФИ 94-01-01327 и 97-01-00724. По теме диссертации опубликовано восемь печатных работ.

Структура и объем работы. Диссертационная работа состоит из введения, трех глав, заключения, списка литературы из 111 наименований и четырех приложений. Объем основной части работы — 121 страница, объем приложений — 19 страниц. Работа включает 5 таблиц и 3 рисунка.

Во **введении** кратко сказано о цели данной работы, о двух рассматриваемых показателях качества программ: надежности и эффективности. Из принципов и методов обеспечения надежности выделяются два — контроль за сложностью и эффективный поиск ошибок.

Первая глава посвящена мерам сложности программ. Создание высококачественного надежного программного обеспечения является одной из основных задач современного программирования. Для успешного ее решения необходимо при проектировании программы произвести "правильное"ее разбиение на модули, т. е. выбрать представление такое, которое способствовало бы более простому написанию, отладке, тестированию и сопровождению. Следовательно, нужны некоторые математические средства, позволяющие классифицировать программный модуль в качестве сложного или простого.

Первая трудность на выбранном пути — понять, что такое сложность программы. Очевидно, сложность программы зависит от человека, создающего программу и разбирающегося в ней, от имеющейся документации о программе и, конечно, самой программы.

В свою очередь, сложность самой программы зависит от: размера программы; ее структур данных; структур управления, определяемых логикой программы; правильного ее разбиения на модули; внутренних связей каждого модуля; межмодульных связей в программе.

Таким образом, сложность программы является, по терминологии теории измерений, *латентной*, или *скрытой*, переменной в отличие от переменных индикаторов, которые можно непосредственно измерять. Функцию, устанавливающую значение латентной переменной того или иного свойства программы исходя из значений индикаторов называют *оценочной функцией*, или *мерой*; область ее значений — *оценочным множеством*; совокупность индикаторов-аргументов — *метрикой* измерения.

Поэтому разработка метрик и мер, способных ранжировать программы по некоторому критерию качества, и создание программных инструментов, обеспечивающих их вычисление — важные проблемы метрологии качества программ.

В метрической теории программ выделяют семь направлений, нас будут интересовать только два из них:

— методический анализ структурной сложности программных модулей и их взаимосвязей для решения проблемы сложности, тестирования и поддержки модулей в работоспособном состоянии, а также минимизация информационных потоков между модулями;

— получение обобщенных характеристик качества программ по ряду критериев.

Метрики сложности программ можно разделить на два больших класса: статические и динамические.

Статические метрики, в свою очередь, можно разделить на четыре типа: объемные, топологические, потока данных и комплексные.

Объемные меры оценивают сложность программ исходя из ее объемных характеристик. Топологические меры основаны на анализе структуры управления программы, представленной в виде управляющего графа или какой-либо другой графовой модели. Потоковые меры основаны на оценке потока данных в программе между ее различными частями. Комплексные меры пытаются устранить недостатки мер, основанных на одном каком-то индикаторе, путем учета разных аспектов сложности, измеряемых разными индикаторами.

Раздел 1.2 содержит описание объемных метрик, которые представлены простейшими объемными мерами и мерами Холстеда.

Простейшие объемные метрики. К ним можно отнести: L — число строк в программе, включая комментарии; S — число исполняемых операторов; U — число программных модулей: подпрограмм, функций и т. д.; S/U — средний размер программного модуля.

Меры Холстеда основаны на следующих индикаторах: n_1,n_2 — число типов операторов и различных операндов; N_1, N_2 — число всех операторов и операндов.

В качестве наиболее известных мер Холстеда можно привести следующие: словарь программы — $n = n_1 + n_2$; длина программы — $N = N_1 + N_2$; оценочная длина программы — $\hat{N} = n_1 \times \log_2 n_1 + n_2 \times \log_2 n_2$; объем программы — $V = N \times \log_2 n$; мера трудности создания программы — $E = V/L$.

Ясно, что оценка сложности программы по Холстеду не учитывает всех характеристик сложности программы. Программы со сложной графовой структурой и линейная, имеющие равный объем, обладают одинаковой сложностью, что в общем-то неверно.

Раздел 1.3 посвящен обзору топологических мер сложности, которые появились в результате неспособности мер Холстеда и аналогичных им учитывать сложность структур управления.

П.1.3.1 содержит определение наиболее известной и популярной меры — Мак-Кейба. *Цикломатической сложностью* программы с управляющим графом G (n вершин, m дуг и одна компонента связности) называется величина $v(G) = m-n+2$. Цикломатическая сложность программы, содержащей вызовы подпрограмм, определяется как сумма цикломатических сложностей самой программы и вызываемых подпрограмм.

Критика по адресу меры Мак-Кейба и желание охватить другие характеристики программ привели к созданию новых мер, часть из которых является вариантами цикломатической сложности. Например, **п.1.3.2 - п.1.3.4** содержат определения мер сложности Хансена, интервальной и топологической. Узловая мера (KNOTS), описание которой дано в **п.1.3.7**, определена только для неструктурированных программ с расположением одного оператора в строке.

Раздел 1.4 содержит описание трех мер, основанных на учете потока данных: мер Хенри и Кафуры, меры Овиедо и меры Тая.

Меры сложности, основанные на концепции информационных потоков между компонентами системы, описаны С. Хенри и Д. Кафурой и содержат информационную сложность процедуры, модуля и информационную сложность модуля относительно структуры данных.

Мера Овиедо определяет сложность программы только через межлучевой поток данных (управляющий граф состоит из непересекающихся участков операторов — лучей). Мера Тая исследует влияние DU-пар (буквой D обозначается определение данных, а U — их использование) на выбор той ветви программы, которая исполняется.

В комплексных мерах, приведеных в **разделе 1.5,** делается попытка объединения различных мер сложности в одну, которая учитывала бы разные

аспекты сложности и давала более аккуратные результаты.

П. Кокол ввел следующую модель комплексной меры:
$H_M = (M + R_1 \times M(M_1) + ... + R_n \times M(M_n)/(1 + R_1 + ... + R_n)$, где: M — базовая метрика; M_i — другие интересные для нас меры; R_i — корректно подобранные коэффициенты; $M(M_i)$ — функции. Функции $M(M_i)$ и коэффициенты R_i вычисляются с использованием регрессивного анализа.

Раздел 1.6 содержит описание трех мер для измерения сложности плана (дизайна) системы. Мера Мак-Клура предназначена для управления сложностью структурированных программ в процессе их проектирования. Она применяется к иерархическим схемам разбиения программ на модули; позволяет выбрать схему разбиения с меньшей сложностью еще до написания программы; исходит из предпосылки, что сложность программы зависит от трех факторов: числа возможных путей исполнения программы, количества управляющих конструкций и набора переменных, управляющих выбором пути.

Мера стабильности плана модуля, введенная Дж. Колоффелло и С. Йо, может быть определена как число предположений-изменений, которые надо сделать в других модулях, но касающихся модуля, стабильность которого измеряем.

Т. Мак-Кейб и Ч. Батлер расширили понятие цикломатической сложности на архитектурный план системы. Архитектурный план системы есть блок-схема системы, или дерево иерархий. По аналогии с управляющим графом программы можно определить цикломатическую сложность этого графа. *Сложность плана модуля* с графом G есть цикломатическая сложность его редуцированного графа. Преобразование (редуцирование) применяется для устранения из модуля сложности, которая не относится к взаимодействию между модулями.

В **разделе 1.7** приведены метрики для объектно-ориентированных (ОО) языков программирования. Метрики Кемерера и Чидамбера, Шарбле и Кохен, Деметера измеряют различные аспекты сложности ОО программ, которая может быть передана как: сложность классов, сложность взаимодействия классов как связь отправитель-получатель, сложность связи классов, задаваемая иерархией наследования.

Раздел 1.8 содержит описание набора мер для распределенного софтвера. Метрики включают четыре группы: первая оценивает размер графа параллельности (concurrency graph) (**п.1.8.1**), вторая — недетерминизм (**п.1.8.2**), третья базируется на цикломатической сложности (**п.1.8.3**) и последняя — на информационных потоках (**п.1.8.4**).

Раздел 1.9 — основной — содержит описание модели мер сложности для Модула-2-программ, предложенной автором. Вводятся метрики для процедуры, модуля определений и модуля реализации.

Метрики для процедуры:

1) СлП — сложность тела процедуры. Возможно применение различных ранее упоминавшихся мер: Мак-Кейба, Холстеда и т. д.;

2) СлКП (сложность коммутативная для процедуры) равняется числу процедур, которые непосредственно вызываются из данной;

3) СлККП (сложность коммутативная комплексная для процедуры) равна

общему числу процедур, которое может быть выполнено, при вызове процедуры;

4) СлГМП (сложность через глобалы модуля для процедуры) равна числу глобалов модуля, использующихся в процедуре;

5) СлГП (сложность через глобалы других модулей для процедуры) равна числу используемых в процедуре переменных, являющихся глобалами других модулей.

6) СлТП (сложность как типы в процедуре) равняется числу типов, используемых в процедуре и определенных в данном модуле;

7) СлТИП (сложность как импортируемые типы в процедуре) равна числу используемых в процедуре типов, импортируемых из других модулей.

8) СлСВ (сложность взаимодействия с модулями программы) равна числу модулей, объекты из которых используются в процедуре.

Метрики для модуля определений:

1) СлМО (сложность модуля определений) — это тройка, состоящая из следующих позиций: числа процедур, числа переменных, числа типов, естественно определенных в данном модуле;

2) СлИМО (сложность импорта модуля определений) равняется числу непосредственно импортируемых модулей;

3) СлВМО (сложность как связность модуля определений) равна общему числу импортируемых модулей (с учетом импорта из импортированных).

Метрики для модуля реализации:

1) СлМ (сложность модуля) — суммарная сложность содержащихся в нем процедур. Если число процедур n, то $M = \sum_{i=0}^{n} i$, при этом тело модуля можно считать нулевой процедурой;

2) СлВ (сложность как связность модуля) равняется общему числу импортируемых модулей;

3) СлИМ (сложность как ширина импорта модуля) равна числу импортируемых модулей;

4) СлTV (сложность как число определенных в модуле сущностей) равняется числу определенных в модуле типов и переменных, т.е. $TV = (\sum Type, \sum Var)$;

5) СлДМ (сложность дизайна модуля) равна числу модулей, которые используют: а) типы, определенные в исследуемом; б) переменные данного модуля; в) процедуры данного модуля;

$= (MT, MV, M)$.

6) СлИС (сложность как импорт сущностей) равна числу импортируемых сущностей: типов, констант, переменных, процедур.

Раздел 1.10 содержит описание двух реализаций метрических компиляторов, выполненных автором. В рамках проекта СОКРАТ был реализован метрический компилятор с языка ассемблера ЕС ЭВМ. Данный компилятор обеспечивал вычисление мер сложности, которые включали: объемные, топологические (Мак-Кейба, KNOTS) и меры для оценки документированности программы. Для вычисления мер был разработан процедурный интерфейс с кросс-транслятором с ассемблера, обеспечивающий сбор необходимой информации о программе

и всех ее объектах. Вся собираемая информация и значения мер сохранялись в текстовом файле, который в дальнейшем использовался разработчиком для оценки сложности программы. Основная трудность данной работы состояла в выработке методики применения результатов метрического компилятора.

При выработке рекомендаций по использованию набора мер сложности автором были предложены пороговые значения мер, которые определялись как из общепризнанных пороговых значений для наиболее популярных мер сложности, так и на основе статистического материала заказчика об ошибках и доработках в программах. Рекомендации сводились к классификации модулей на основе значений набора мер для них. Модуль может быть простой, сложный и очень сложный. В зависимости от этого возможны различные действия пользователя: никаких, особое внимание при отладке и тестировании, перепроектирование и перепрограммирование.

Модель мер Модула-2 реализована для XDS компилятора. Используется front-end компилятора XDS, который обеспечивает построение внутреннего представления программы, обрабатываемое инструментами метрического анализатора, для проведения статического анализа программы и сбора о ней и ее объектах информации необходимой, для вычисления набора мер.

Вычисляемый набор мер включает: объемные, потоковые, структурные и меры для оценки документированности. Данный набор позволяет оценить различные аспекты сложности объекта, выявить его взаимосвязи с другими объектами и провести его классификацию: простой, нормальной сложности, сложный, очень сложный. И на этой основе принять решение о дальнейших шагах по разработке данного модуля, а именно: особого внимания и затрат на тестирование не требует, достаточно стандарных действий по тестированию; при тестировании и отладке уделить чуть больше внимания и составить "хороший"набор тестов; уделить особое внимание при отладке и тестировании; возможно перепроектировать и переписать те объекты, которые внесли наибольшую сложность, в крайнем случае подвергнуть тщательному тестированию.

Раздел 1.11 — заключение — подводит итоги данной главы и содержит *методологию оценки качества программ с помощью метрик* . Это систематический подход, основанный на задании требований качества, выборе характеристик, определяющих качество, подборе мер, либо измеряющих данные характеристики, либо предсказывающих их значения, проведении измерений, выполнении анализа и оценки результата, их использовании для программных систем. Он охватывает полный жизненный цикл программы и включает пять шагов, которые могут применяться с повторением, так как понимание, полученное от применения определенного шага, может привести к необходимости переоценки результатов предыдущих шагов.

С точки зрения автора, необходим еще один шаг — создание системы, которая на основании полученных значений мер дает рекомендации о дальнейшей судьбе модуля или его части, т.е. проводит классификацию модулей и на ее основе — рекомендации, т.к. для большинства пользователей как сами меры, так и их конкретные значения мало о чем говорят и без классификации модулей метрические компиляторы очень часто оказываются нежизнеспособными.

В главе 2 рассматриваются методы и инструменты измерения динамических характеристик программ. Совокупность средств исследования динамических

характеристик программ получила название *профилировщик*. Основной задачей данных инструментов является построение профиля программы — набора определенных характеристик, полученного в процессе исполнения программы. Этот набор не постоянен и может меняться от задачи к задаче в зависимости от того, какие характеристики необходимо получить о ходе выполнения программы. В данной работе профиль содержит информацию о динамике (частота и время) выполнения операторов, строк, блоков, процедур и характере изменения значений глобальных переменных программы. Главное назначение профилировщика состоит в подсчете числа исполнений определенных, заданных пользователем объектов программы, а также числа обращений и/или изменений ее переменных (тоже не всех, а заданных), а для исполняемых объектов — времени их исполнения. Объектами, для которых производится подсчет их числа исполнений, являются строки, операторы, процедуры и блоки — подряд расположенные строки программы. Время исполнения подсчитывается для блоков и процедур, а в случае ассемблерных программ — и для каждого выделенного оператора.

Для реализации перечисленных функций необходимо:

— обеспечить выделение пользователем следующих объектов программы: строк, операторов, блоков, процедур, переменных и в случае сложных переменных их простых составляющих;

— обеспечить во время исполнения программы порождение определенного события при достижении выделенных объектов;

— написать подпрограммы обработки порождаемых событий.

Среда отладчика, в которую встроен профилировщик, содержит механизмы порождения событий, необходимо только добавить типы вновь создаваемых событий и реализовать методы их обработки. Создаваемые методы с идеологической точки зрения очень просты — надо для каждого исследуемого объекта завести счетчик или счетчики и при необходимости увеличивать их значения. Основная задача — обеспечение эффективного доступа к исследуемым объектам.

В отличие от других подобных систем в данной работе было введено и реализовано понятие *частичного* профиля, получаемого с помощью средств, предоставляемых рабочей средой профиля — диалогово-пакетным кросс-отладчиком. Обычно профилировщики строят профиль программы для всего времени ее исполнения, так как элементы профилирования задаются до исполнения и не могут быть изменены до завершения исполнения и нет возможности посмотреть результаты во время исполнения программы. Профиль создается после исполнения путем обработки собранных данных. Разработанная система позволяет расширить возможности профилирования, вводя *частичный* профиль, который характеризует работу программы лишь на некотором ее временно́м участке. Другими словами, пользователь задает точки останова, используя средства отладчика, и в ходе исполнения программы после остановки на одной из ранее поставленных точек останова задает набор отслеживаемых характеристик для профилирования и, возможно, новые точки останова. При дальнейшем исполнении, в момент остановки программы, существует возможность посмотреть желаемые характеристики и сохранить их в файле, а если необходимо, изменить набор характеристик, и т. д. на протяжении всего сеанса исполнения программы.

В **п.2.1** приведен обзор применения профилей и методов их построения. Остановимся только на двух способах использования профилей. Результаты профилирования можно использовать на этапе улучшения качества программы, а именно: подвергать улучшению те участки программы, на которых она затрачивает больше всего времени исполнения. Особый интерес представляет применение профилирования на этапе тестирования программ. Получаемые динамические характеристики помогают выделить интенсивно используемые компоненты программ, которые целесообразно подвергать наиболее тщательной проверке. Вместе с этим находятся компоненты, с малой или нулевой частотой проверки исполнения, подлежащие дополнительному тестированию или исследованию на достижимость. Анализ статистики условных переходов и итераций циклов позволяет обнаружить некоторые типы ошибок.

Третья глава посвящена методам тестирования программ и инструментам, обеспечивающим помощь пользователю как в построении хорошего набора тестов, так и в оценке полноты имеющегося набора тестов. Тестирование является одним из основных методов достижения надежности программного обеспечения, поэтому, чтобы добиться высокого качества программы, необходимо уметь строить для нее хороший набор тестов. Хороший набор тестов должен, в первую очередь, иметь высокую вероятность нахождения всех возможных ошибок в программе, причем из соображений эффективности желательно, чтобы эта цель достигалась с минимальными затратами времени на построение набора тестов, его реализацию, исполнение и проверку результатов.

Проблему получения хорошего набора тестов в общем случае можно сформулировать следующим образом:

Постановка задачи. *Дана программа P и набор тестов T. Достаточно ли этого набора тестов T для "полного"тестирования программы P, и если нет, то какие тесты нужно добавить?*

Полное тестирование программы означает, что набор тестов может обнаружить любую ошибку в программе, и если результаты исполнения программы на этом наборе тестов совпадают с ожидаемыми, то правильность программы доказана "от противного". Однако задача построения полного набор тестов является алгоритмически неразрешимой, поэтому возможно лишь частичное решение этой проблемы, т.е. для данного набора тестов иногда можно сказать, какие тесты необходимо добавить еще.

Для этого были разработаны методы (критерии) тестирования, которые требуют построения набора тестов с определенными свойствами.

Далее во введении проводится обзор и дается классификация систем поддержки пользователя для построения хорошего набора тестов.

Формулируется **комплексный критерий системы OCT** (**O**beron-2/Modula-2 **C**overage **T**ool), включающий следующие критерии: C1, покрытия условий, покрытия циклов, покрытия отношений, покрытия рекурсивных процедур, критерий интеграционного тестирования — покрытие вызовов процедур, а также критерий покрытия входных параметров и результатов процедур. Три последних критерия предложены впервые и очень важны по двум причинам, во-первых, они фактически объединяют методы белого и черного ящиков и, во-вторых, тестируют такой важный объект как процедура, что

для сильно запроцедуренного языка программирования просто необходимо.

В разделе 3.1 предложен механизм формального описания критериев тестирования и с его помощью даны описания наиболее популярных критериев тестирования.

Пусть $\mathcal{P}$ — множество всех программ, и пусть P — программа из $\mathcal{P}$ с множеством входных данных In и множеством выходных данных Out. Набор тестов T для программы P — это некоторое подмножество входных данных: $T \subseteq In$.

Критерий тестирования C — это такой предикат, что

$$C(P,T) \iff T \text{ достаточно для тестирования } P \text{ согласно данному критерию.}$$

Такое определение не очень полезно для целей построения набора тестов, удовлетворяющего критерию, поэтому определим C с помощью множества требуемых элементов и понятия покрытия.

Для критерия C зададим множество требуемых элементов $REQ_C(P)$ (функция, определенная на $\mathcal{P}$) и предикат COV_C, определенный на множестве $\mathcal{P} \times In \times REQ_C$ с неформальной семантикой:

$$COV_C(P,x,r) \iff \text{при исполнении программы } P \text{ на входных данных } x \text{ требуемый элемент } r \text{ покрывается.}$$

Сам предикат C можно определить следующим образом:

$$C(P,T) \iff \forall\, r \in REQ_C(P)\ \&\ (\exists\, x \in In : COV_C(P,x,r)) \\ \exists\, t \in T : COV_C(P,t,r).$$

Таким образом, для определения критерия надо задать множество требуемых элементов и определить предикат покрытия конкретного элемента. Именно так далее в данном разделе определяются критерии тестирования.

Например, критерий all-defs определяется следующим образом:

Критерий all-defs (покрытие всех определений)

$$\begin{aligned} REQ_d(P) &= \{(x,v) \mid x \in VAR,\ v \in Def(x)\} \\ COV_d(P,x,(z,v)) &\iff \exists\, q \in PT(run(P,x))\ \ \exists\, u \in Use(x) \mid \\ &\quad clear_reach(q,v,u,z), \end{aligned}$$

где VAR — множество всех переменных программы, $Def(x)$ — множество вершин из V (вершины УГ), в которых переменная x из VAR определяется, $Use(x)$ — множество вершин, в которых x используется, $PT(run(P,x))$ — множество путей исполнения P на x.

Предикат $clear_reach(q,v,u,x)$, означающий, что участок пути $[v,u] \subset q$ $(v,u \in V)$ не содержит присваиваний переменной $x \in VAR$, определяется так:

$$clear_reach(q,v,u,x) \iff \exists\, i,j\ :\ i<j\ :\ q_i = v\ \&\ q_j = u\ \& \\ \forall\, s\ :\ i<s<j\ :\ q_s \notin Def(x).$$

В разделе 3.2 приведен обзор существующих систем поддержки структурного тестирования программ, инструментов проектирования тестов из текста программы, а также дано описание и реализация системы `TGS (Test Generation System)` проекта СОКРАТ. Последняя для каждой процедуры находит множество путей, покрывающих все ее дуги, циклы и определения переменных (для

покрытия дуг строится минимальное множество путей), и затем вычисляет для них условия реализуемости. После этого пользователь должен найти тесты, для которых эти условия реализуемости выполняются, и получить в результате набор тестов, удовлетворяющих критериям C1, all-defs и покрытия циклов.

Задача поддержки структурного тестирования программ, по существу, состоит из двух частей. Во-первых, необходимо помочь пользователю в построении набора тестов, покрывающих логику программы (эта часть решается системой TGS), а во-вторых, при тестировании оценить набор тестов на полноту для выбранного критерия тестирования и в каждый текущий момент показать степень тестированности для данного критерия.

Разработанная система обеспечивает поддержку тестирования программ, написанных на двух языках: Модуле-2 и языке ассемблера ЕС ЭВМ, и содержит следующие компоненты: компонент построения управляющего графа программы и сохранения его в архиве; компонент построения множества путей, покрывающих все требуемые элементы комплексного критерия, для критерия C1 строится минимальное множество путей; компонент построения формулы реализуемости для заданного пути, реализует символическое исполнение программы по данному пути; компонент пропуска пакета тестов с контролем критерия тестирования и оценкой степени тестированности.

Далее в разделе изложены методы реализации компонентов системы.

В разделе 3.3 описана система ОСТ, которая реализована методом *инструментации* исходной программы, т.е. в исходный текст программы вставляется некоторый код, проверяющий выполнение критериев при исполнении программы и сохраняющий результаты после ее окончания. Затем эти результаты (полученные после исполнения набора тестов) обрабатываются генератором отчетов, который сообщает пользователю степень тестированности инструментированных модулей и при необходимости показывает непокрытые тестовые условия и, таким образом, помогает построить дополнительные тесты.

Далее в разделе дано подробное описание механизма работы системы, инструментации исходной программы для всех критериев комплексного критерия системы ОСТ и языка описания тестовых условий.

Для критериев покрытия входных параметров и результатов процедур и покрытия вызовов процедур тестовые условия задаются с помощью специально разработанного языка описания тестовых условий.

Пользователь системы создает текстовый файл, в котором описывает тестовые условия типа и тестовые условия вызова процедур, естественно, только в том случае, если он хочет воспользоваться этими критериями.

Каждое тестовое условие состоит из выражения типа BOOLEAN и связанного с ним сообщения. Выражение инструментатор вставит с некоторой модификацией в текст тестируемого модуля, а сообщение будет присутствовать в итоговом полном отчете модуля, если данное тестовое условие не будет покрыто.

В заключении данного раздела (**п.3.3.4**) подчеркивается,что разработанная система контроля тестированности программ, написанных на языке Модула-2/Оберон-2, позволяет значительно повысить эффективность тестирования и найти многие ошибки, которые иначе могли бы остаться в программе и

обнаружиться только при ее эксплуатации. Это обеспечивается использованием разработанного комплексного критерия тестирования, благодаря которому возможно создание хорошего набора тестов. При этом удачный выбор критериев избавляет пользователя от построения малоэффективных тестов или таких, на построение которых тратилось бы слишком много времени.

При разработке и реализации системы ОСТ впервые предложены и реализованы критерии покрытия рекурсии, входных параметров и результатов процедур, а также язык описания тестовых условий для типов и вызовов процедур и его применение в инструменте подобного типа.

В рамках данной работы выполнены реализации следующих программных систем: метрические компиляторы с ассемблера ЕС ЭВМ и Модулы-2, профилировщики для них, система TGS и компонент оценки тестированности ассемблерных программ для критерия C1, система ОСТ.

В приложении А приведена методика использования средств измерения (данный набор средств состоит из метрического компилятора, профилировщика системы TGS и компонента оценки степени тестированности) для ассемблерных программ, включающая пороговые значения мер сложности, и на их основе классификация модулей.

Приложение Б содержит примеры работы профилировщиков асемблерных и Модула-2-программ.

В приложении В приведены основные алгоритмы построения минимального дугового покрытия.

Приложение Г содержит пример работы системы ОСТ и возможный анализ полученных результатов для пополнения набора тестов и обнаружения имеющихся ошибок в программе.

В **Заключении** перечисляются основные результаты работы.

1. На основе изучения и анализа имеющихся мер сложности программ разработана модель мер сложности для Модула-2-программ. Данная модель включает меры для процедуры, модуля определений и модуля реализаций. Она в совокупности позволяет оценить сложность данного модуля и принять необходимые решения по дальнейшим шагам относительно его использования. Реализован набор инструментов, вычисляющий набор мер данной модели, и компонент вычисляющий набор мер сложности для программ, написанных на ассемблере ЕС ЭВМ. Разработана методика применения получаемых результатов.

2. Разработана концепция средств профилирования (сбора динамических характеристик программ) для ассемблерных и Модула-2-программ. Основным ее отличием от существующих является понятие частичного профиля программы. Проведена реализация данной концепции — созданы профилировщики ассемблерных и Модула-2-программ.

3. Предложен способ формального описания критериев тестирования с помощью понятий: множество требуемых элементов и покрытия данного множества. Разработан комплексный критерий тестирования для Модула-2/Оберон-2-программ, включающий семь критериев структурного тестирования, три из которых предложены и сформулированы впервые. Для задания тестовых условий и их описания во вновь сформулированных критериях разработан простой язык описания тестовых условий.

4. Исследована проблема построения минимального дугового покрытия для ассемблерных и Модула-2-программ. Предложен алгоритм построения минимального дугового покрытия с учетом тех замечаний и неточностей, которые присутствовали в алгоритме, предложенным К.А. Иыуду и М.М. Ариповым с уточнениями В.А. Шимарова.

   В рамках проекта СОКРАТ реализована система TGS, строящая множество путей, покрывающих тестовые условия, задаваемые комплексным критерием, который состоит из следующих трех критериев структурного тестирования: C1, покрытия всех существенных определений переменных (all-defs) и покрытия циклов (all-cycles). Для критерия C1 строится минимальное дуговое покрытие, которое пополняется необходимым набором путей для выполнения двух оставшихся критериев. Для всех построенных путей формируется формула их реализуемости.

   Реализован компонент контроля тестированности ассемблерных программ для критерия C1, обеспечивающий показ как степени тестированности программы, так и тех ее ветвей, которые еще не исполнялись на текущий момент тестирования, если, конечно, они есть.

5. Реализована система ОСТ — система контроля тестированности Модула-2/Оберон-2 программ (система оценки полноты набора тестов), которая обеспечивает показ как степени тестированности программ по комплексному критерию, так и непокрытых, на текущий момент, тестовых элементов, что, в свою очередь, или помогает построить новые тесты для их покрытия, или простой анализ полученных результатов, связанных с непокрытостью определенных тестовых условий, приводит к выявлению ошибок в программе.

1. **Черноножкин С.К.** Меры сложности программ (Обзор) //Системная информатика. Вып. 5: Архитектурные, формальные и программные модели. — Новосибирск: Наука, 1997. — С. 188—227.

2. **Черноножкин С.К.** Меры сложности программ (Обзор). — Новосибирск, 1994. — 36с. — (Препр. /СО РАН. ИСИ; № 21).

3. **Черноножкин С.К.** Средства профилирования программ в системе СОКРАТ. — Новосибирск, 1998. — 20с. — (Препр. /СО РАН. ИСИ; № 48).

4. **Кауфман А.В., Черноножкин С.К.** Инструменты поддержки структурного тестирования в системе СОКРАТ // Средства и инструменты окружений программирования. — Новосибирск, 1995. — С. 30—45.

5. **Кауфман А.В., Черноножкин С.К.** Структурное тестирование в системе СОКРАТ // Программные системы. — Новосибирск, 1996. — С. 135—148.

6. **Кауфман А.В., Черноножкин С.К.** ОСТ: система контроля тестированности Модула-2-программ. — Новосибирск, 1997. — 46 с. — (Препр. /СО РАН. ИСИ; №38).

7. **Кауфман А.В., Черноножкин С.К.** Критерии тестирования и система оценки полноты набора тестов // Программирование. — 1998. — №6.

8. **Черноножкин С.К.** Методы оценки тестированности программ // Третий Сибирский конгресс по прикладной и индустриальной математике (ИНПРИМ-98): Тез. докл. Ч.5 — Новосибирск: Изд. ИМ СО РАН, 1998. — С.52.

Подписано в печать 22.12.1998 Объем 1.1 уч.-изд.л.
Формат бумаги 60×90 1/16 Тираж 100 экз.

Отпечатано на ризографе "AL-Group".
630090, Новосибирск-90, пр. Акад. Лаврентьева, 6.