Научно исследовательская работа студента на тему:

«Автоматизированная система разработки тестов программного обеспечения по критерию C1»

Автор: Ерёмичев В.В.

ВВЕДЕНИЕ

Тестирование программного обеспечения - это процесс исследования ПО с целью получения информации о качестве продукта.

Существующие на сегодняшний день методы тестирования ПО не позволяют однозначно и полностью выявить все дефекты и установить корректность функционирования анализируемой программы, поэтому все существующие методы тестирования действуют в рамках формального процесса проверки исследуемого или разрабатываемого ПО.

Такой процесс формальной проверки или верификации может доказать, что дефекты отсутствуют с точки зрения используемого метода. (То есть нет никакой возможности точно установить или гарантировать отсутствие дефектов в программном продукте с учётом человеческого фактора, присутствующего на всех этапах жизненного цикла ПО).

Существует множество подходов к решению задачи тестирования и верификации ПО, но эффективное тестирование сложных программных продуктов — это процесс в высшей степени творческий, не сводящийся к следованию строгим и чётким процедурам или созданию таковых.

1. СТАНДАРТИЗАЦИЯ ПРОЦЕССА ТЕСТИРОВАНИЯ

В сфере информационных технологий существует единый стандарт в области тестирования ISO 9126 в Украине ДСТУ ISO/IEC 14598-6:2005- «Информационная технология. Оценка программного продукта. Характеристики качества и руководство по их применению».

ISO 9126 это международный стандарт, определяющий оценочные характеристики качества программного обеспечения (далее ПО). Стандарт разделяется на 4 части, описывающие следующие вопросы: модель качества; внешние метрики качества; внутренние метрики качества; метрики качества в использовании

Модель качества, установленная в первой части стандарта , классифицирует качество ПО в 6-ти структурных наборах характеристик, которые в свою очередь детализированы под характеристиками (субхарактеристиками), такими как:

- Функциональность — *Набор атрибутов характеризующий, соответствие функциональных возможностей ПО набору требуемой пользователем функциональности. Детализируется следующими подхарактеристиками:*
  - Пригодностью для применения
  - Корректностью (правильностью, точностью)
  - Способностью к взаимодействию (в частности сетевому)
  - Защищенностью
- Надёжность — *Набор атрибутов, относящихся к способности ПО сохранять свой уровень качества функционирования в установленных*

условиях за определенный период времени. Детализируется следующими подхарактеристиками:

- Уровнем завершенности (отсутствия ошибок)
- Устойчивостью к дефектам
- Восстанавливаемостью
- Доступностью
- Готовностью

- Практичность **(**применимость**)** — *Набор атрибутов, относящихся к объему работ, требуемых для исполнения и индивидуальной оценки такого исполнения определенным или предполагаемым кругом пользователей. Детализируется следующими подхарактеристиками:*
  - Понятностью
  - Простотой использования
  - Изучаемостью
  - Привлекательностью
- Эффективность — *Набор атрибутов, относящихся к соотношению между уровнем качества функционирования ПО и объемом используемых ресурсов при установленных условиях. Детализируется следующими подхарактеристиками (субхарактеристиками):*
  - Временной эффективностью
  - Используемостью ресурсов

- Сопровождаемость — *Набор атрибутов, относящихся к объему работ, требуемых для проведения конкретных изменений (модификаций). Детализируется следующими подхарактеристиками :*
  - Удобством для анализа;
  - Изменяемостью
  - Стабильностью
  - Тестируемостью
- Мобильность — *Набор атрибутов, относящихся к способности ПО быть перенесенным из одного окружения в другое. Детализируется следующими подхарактеристиками (субхарактеристиками):*
  - Адаптируемостью
  - Простотой установки (инсталляции)
  - Сосуществованием (соответствием)
  - Замещаемостью

В стандарте выделена модель характеристик качества в использовании. Основными характеристиками качества программных средств (далее ПС) в использовании рекомендуются:

- **Системная эффективность** — Применения программного продукта по назначению.
- **Продуктивность** — Производительность при решении основных задач ПС, достигаемая при реально ограниченных ресурсах в конкретной внешней среде применения.

- **Безопасность** — Надежность функционирования комплекса программ и возможный риск от его применения для людей, бизнеса и внешней среды
- **Удовлетворение требований и затрат пользователей в соответствии с целями применения ПС**

Вторая и третья части стандарта ISO 9126-2,3 посвящены формализации соответственно внешних и внутренних метрик характеристик качества сложных ПС. В ней изложены содержание и общие рекомендации по использованию соответствующих метрик и взаимосвязей между типами метрик.

Четвертая часть стандарта ISO 9126-4 предназначена для покупателей, поставщиков, разработчиков, сопровождающих, пользователей и менеджеров качества ПС. В ней повторена концепция трех типов метрик, а также аннотированы рекомендуемые виды измерений характеристик ПС.

С технической точки зрения тестирование заключается в выполнении приложения на некотором множестве исходных данных и сверке получаемых результатов с заранее известными (эталонными) с целью установить соответствие различных свойств и характеристик приложения заказанным свойствам. Как одна из основных фаз процесса разработки программного продукта (Дизайн приложения - Разработка кода - Тестирование), тестирование характеризуется достаточно большим вкладом в суммарную трудоемкость разработки продукта. Широко известна оценка распределения трудоемкости между фазами создания программного продукта: 40%-20%-40%.

2. ОБЗОР СОВРЕМЕННЫХ МЕТОДОВ И ПРИЕМОВ ТЕСТИРОВАНИЯ

За всю историю существования тестирования как области знаний в информационных технологиях, были разработаны множество способов выявления программных дефектов, которые применяются в по отдельности и в комбинации друг с другом для достижения наилучшего результата и получения так называемого оптимального тестирования, когда находится баланс между затраченными на тестирование программного продукта временем и усилиями и степенью оттестированности системы.

2.1 Понятие об управляющем графе программы

Управляющий граф программы (УГП) - граф G(V,A), где V(V1,… Vm) – множество вершин (операторов), A(A1,… An) – множество дуг (управлений), соединяющих операторы-вершины.

Путь – последовательность вершин и дуг УГП, в которой любая дуга выходит из вершины Vi и приходит в вершину Vj.

Ветвь – путь (V1, V2, … Vk), где V1 - либо первый, либо условный оператор программы, Vk - либо условный оператор, либо оператор выхода из программы, а все остальные операторы – безусловные, например: (3,4) (4,5,6,4) (4,7). Пути, различающиеся хотя бы числом прохождений цикла – разные пути, поэтому число путей в программе может быть не ограничено. Ветви - линейные участки программы, их конечное число.

Существуют реализуемые и нереализуемые пути в программе, в нереализуемые пути в обычных условиях попасть нельзя.

Пример:

Дана функция на языке C# которая возводит число x в неотрицательную степень n. Построим на примере данной программы УГП (см Рис 1.)

```
1  double Power(double x, int n){
2  double z=1; int i;
3  for (i=1;
4  n>=i;
5  i++)
6  {z = z*x;} /* Возврат в п.4 */
7  return z;}
```

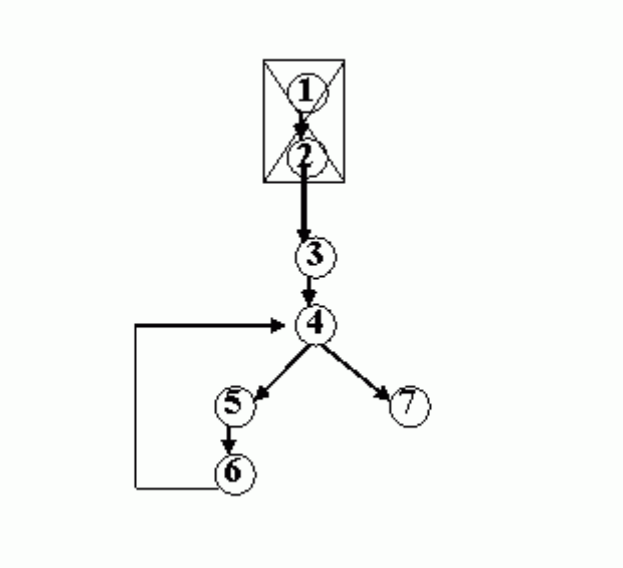

Рис. 1 «Управляющий граф программы»

2.2 Классификация критериев тестирования.

Не существует в природе идеального критерия тестирования, который бы подходил для всех типов программного обеспечения по структуре и ложности. Идеальный критерий должен обладать следующими свойствами:

1. **Критерий должен быть достаточным**, т.е. показывать, когда некоторое конечное множество тестов достаточно для тестирования данной программы.
2. **Критерий должен быть полным**, т.е. в случае ошибки должен существовать тест из множества тестов, удовлетворяющих критерию, который раскрывает ошибку.

3. **Критерий должен быть надежным**, т.е. любые два множества тестов, удовлетворяющих ему, одновременно должны раскрывать или не раскрывать ошибки программы
4. **Критерий должен быть легко проверяемым**, например вычисляемым на тестах

Для нетривиальных классов программ в общем случае не существует полного и надежного критерия, зависящего от программ или спецификаций. Поэтому мы стремимся к идеальному общему критерию через реальные частные.

1. **Структурные критерии** используют информацию о структуре программы (критерии так называемого "белого ящика"), что предполагает знание исходного текста программы или спецификации программы в виде потокового графа управления. Структурные критерии базируются на основных элементах графа управления - операторах, ветвях и путях.

Управляющий граф программы

- Условие критерия тестирования команд (критерий C0) - набор тестов в совокупности должен обеспечить прохождение каждой команды не менее одного раза.
- Условие критерия тестирования ветвей (критерий C1) - набор тестов в совокупности должен обеспечить прохождение каждой ветви не менее одного раза.
- Условие критерия тестирования путей (критерий C2) - набор тестов в совокупности должен обеспечить прохождение каждого пути не менее 1 раз.

Недостатком структурных критериев является тот факт, что структурные критерии не проверяют соответствие спецификации, если оно не отражено в структуре программы. Поэтому при успешном тестировании программы по критерию C1 мы можем не заметить ошибку, связанную с невыполнением некоторых условий спецификации требований.

2. **Функциональные критерии** формулируются в описании требований к программному изделию (критерии так называемого "черного ящика") Они обеспечивают, прежде всего, контроль степени выполнения требований заказчика в программном продукте. Поскольку требования формулируются к продукту в целом, они отражают взаимодействие тестируемого приложения с окружением. Проблема функционального тестирования - это прежде всего трудоемкость; дело в том, что документы, фиксирующие требования к программному изделию, как правило, достаточно объемны, тем не менее соответствующая проверка должна быть всеобъемлющей.
    a. Выделяют следующие частные виды функциональных критериев:
    b. тестирование пунктов спецификации;
    c. тестирование классов входных данных;
    d. тестирование правил - набор тестов в совокупности должен обеспечить проверку каждого правила, если входные и выходные значения описываются набором правил некоторой грамматики;
    e. тестирование классов выходных данных;
    f. тестирование функций;
    g. комбинированные критерии для программ и спецификаций.

3. **Критерии стохастического тестирования** формулируются в терминах проверки наличия заданных свойств у тестируемого приложения, средствами проверки некоторой статистической гипотезы. Применяется при тестировании сложных программных комплексов - когда набор детерминированных тестов (X, Y) имеет громадную мощность.

4. **Мутационные критерии** ориентированы на проверку свойств программного изделия на основе подхода Монте-Карло.

   Метод мутационного тестирования состоит в том, что в разрабатываемую программу P вносят мутации (мелкие ошибки), т.е. искусственно создают программы-мутанты P1, P2... . Затем программа P и ее мутанты тестируются на одном и том же наборе тестов (X, Y).

   Если на наборе (X, Y) подтверждается правильность программы P и, кроме того, выявляются все внесенные в программы-мутанты ошибки, то набор тестов (X, Y) соответствует мутационному критерию, а тестируемая программа объявляется правильной. Если некоторые мутанты не выявили всех мутаций, то надо расширять набор тестов (X, Y) и продолжать тестирование.

Фазы тестирования

При тестировании как правило выделяют три фазы: модульное, интеграционное и системное тестирование.

Модульное тестирование - это тестирование программы на уровне отдельно взятых модулей, функций или классов. Цель модульного тестирования состоит в выявлении локализованных в модуле ошибок в реализации алгоритмов, а также в определении степени готовности системы к переходу на следующий уровень разработки и тестирования. Модульное тестирование проводится по принципу "белого ящика", то есть основывается

на знании внутренней структуры программы, и часто включает те или иные методы анализа покрытия кода.

Интеграционное тестирование - это тестирование части системы, состоящей из двух и более модулей. Основная задача интеграционного тестирования - поиск дефектов, связанных с ошибками в реализации и интерпретации интерфейсного взаимодействия между модулями. Основная разница между модульным и интеграционным тестированиями состоит в целях, то есть в типах обнаруживаемых дефектов, которые, в свою очередь, определяют стратегию выбора входных данных и методов анализа.

Системное тестирование качественно отличается от интеграционного и модульного уровней. Оно рассматривает тестируемую систему в целом и оперирует на уровне пользовательских интерфейсов. Основная задача системного тестирования состоит в выявлении дефектов, связанных с работой системы в целом, таких как неверное использование ресурсов системы, непредусмотренные комбинации данных пользовательского уровня, несовместимость с окружением, непредусмотренные сценарии использования, отсутствующая или неверная функциональность, неудобство в применении и тому подобное.

Системное тестирование производится над проектом в целом с помощью метода "черного ящика". Структура программы не имеет никакого значения, для проверки доступны только входы и выходы, видимые пользователю. Тестированию подлежат коды и пользовательская документация.

Кроме того, выделяют **регрессионное тестирование** - цикл тестирования, который производится при внесении изменений на фазе системного тестирования или сопровождения продукта. Главная проблема регрессионного тестирования - выбор между полным и частичным перетестированием и пополнением тестовых наборов. При частичном

перетестировании контролируются только те части проекта, которые связаны с измененными компонентами.

Этапы тестирования

Каждая фаза тестирования включает в себя следующие этапы:

1. **Определение целей** (требований к тестированию), включающее следующую конкретизацию: какие части системы будут тестироваться, какие аспекты их работы будут выбраны для проверки, каково желаемое качество и т. п.

2. **Планирование**: создание графика (расписания) разработки тестов для каждой тестируемой подсистемы; оценка необходимых человеческих, программных и аппаратных ресурсов; разработка расписания тестовых циклов. Важно отметить, что расписание тестирования обязательно должно быть согласовано с расписанием разработки создаваемой системы.

3. **Разработка тестов** (тестового кода для тестируемой системы).

4. **Выполнение тестов**: реализация тестовых циклов.

2.3 Анализ результатов.

Тестовый цикл - это цикл исполнения тестов, включающий фазы 4 и 5 тестового процесса. Тестовый цикл заключается в прогоне разработанных тестов на некотором однозначно определяемом срезе системы (состоянии кода разрабатываемой системы). Обычно такой срез системы называют **build**.

Тестовый план - это документ, или набор документов, который содержит тестовые ресурсы, перечень функций и подсистем, подлежащих тестированию, тестовую стратегию, расписание тестовых циклов, фиксацию тестовой конфигурации (состава и конкретных параметров аппаратуры и

программного окружения), определение списка тестовых метрик, которые на тестовом цикле необходимо собрать и проанализировать (например метрик, оценивающих степень покрытия тестами набора требований).

Тесты разрабатывают на основе спецификаций как вручную, так и с помощью автоматизирующих средств. Помимо собственно кода, в понятие "тест" включается его общее описание и подробное описание шагов, выполняемых в данном тесте.

Для оценки качества тестов используют различные метрики, связанные с количеством найденных дефектов, покрытием кода, функциональных требований, множества сценариев.

Вся информация об обнаруженных в процессе тестирования дефектах (тип, условия обнаружения, причина, условия исправления, время, затраченное на исправление) заносятся в базу дефектов.

Информация о тестовом плане, тестах и дефектах используется в конце каждого цикла тестирования для генерации тестового отчета и корректирования системы тестов для следующей итерации.

2.4 Принципы создания тестов

Создание на основе анализа потока управления . В этом случае элементы, которые должны быть покрыты при прохождении тестов, определяются на основе структурных критериев тестирования C0,C1,C2. К ним относятся вершины, дуги, пути управляющего графа, и критерий покрытия вызовов(каждый вызов каждой функции программы должен быть осуществлен хотя бы один раз)

На основе анализа потока данных (элементы, которые должны быть покрыты определяются на основе информационного графа программы). Этот вид направлен на выявление ссылок на неинициализированные переменные и избыточные присваивания тестирования всех взаимосвязей, включающих в себя использование и определение переменной. Недостаток стратегии в том, что она не гарантирует покрытия решений.

Построение набора тестов.

1. Конструирование УГП
2. Выбор тестовых путей
    a. Статические методы - построение каждого пути посредством постепенного его удлинения за счет добавления дуг, пока не будет достигнута выходная вершина.
    Недостатки – не учитывается возможная нереализуемость построенных путей тестирования (непредсказуемый процент брака).
    - Трудоемкость (переход от покрывающего множества путей к полной системе тестов осуществляется вручную)
    Достоинство - сравнительно небольшое количество необходимых ресурсов
    b. Динамические методы - построение полной системы тестов, удовлетворяющих заданному критерию, путем одновременного решения задачи построения покрывающего множества путей и тестовых данных. При этом можно автоматически учитывать реализуемость или нереализуемость ранее рассмотренных путей или их частей.
    Достоинство - некоторый качественный уровень - реализуемость путей.

c. Методы реализуемых путей- выделение из множества путей подмножества всех реализуемых путей, из которых строится покрывающее множество путей.

3. Генерация тестов, соответствующих тестовым путям.

3 . МАТЕМАТИЧЕСКАЯ ПОСТАНОВКА ЗАДАЧИ

Метрики - характеристики кода, которые дают возможность произвести оценку понятности, гибкости и правильности кода, выраженные в числовом эквиваленте.

1. CCN (Cyclomatic Complexity Number) - цикломатическая сложность кода, данная метрика показывает количество возможных условных вариантов выполнения кода или количества различных путей выполнения (прохождения) метода. Формула вычисления сложности выглядит следующим образом

   CC = E - N + P, где E- число ребер графа, N число вершин а P - число связанных компонент

   Для кода это число вычисляется так : при встрече каждого условного оператора(if,for,while,case...) прибавляется единица, также единица прибавляется для EntryPoint метода. Например, сложность следующего кода равна 3

```
public int getValue(int param1) {
  int value = 0;
  if (param1 == 0)  {
    value = 4;
  } else {
    value = 0;
  }
  return value;
}
```

2. Code Coverage - покрытие кода разнообразными тестами, данная метрика определяет отношение между количеством вызванных тестом строк к общему количеству строк метода/класса/пакета. Также помимо просто построчного покрытия есть поблочное/веточное(branch) - которое учитывает блоки или ветки разделенными условными/циклическими операторами. Средств оценки данной метрик очень много, но все они работают примерно одинаково - сначала в исходный код добавляются специальные операторы логгирования далее прогоняются необходимые тесты и собирается статистика зафиксированная логгерами. Это один из видов метрик который меняет (в некоторых языках) исходный(промежуточный) код (ну есть еще Mutation testing :)). Средства оценки - Cobertura,Emma,NCover,RCOV. Обычно данная метрика измеряется в процентах, максимальное значение естественно 100%, но это не означает что ошибок нет, это всего лишь значит, что весь исходный код вызывался из тестов.

Виды покрыти кода.
1) (Statement) C1 - линейное покрытие кода, считается каждый выполненный оператор (строчка кода или блок).Данный вид покрытия самый простой и не требует изменения исходного кода.

2)Оценка покрытия кода с учетом условных операторов(Decision - C2) - в каждом условном операторе (if,while,switch...) проверяется вызывался ли условный оператор со значением условия true и false, данный способ оценки более адекватный, но не учитывает тот случай если условие содержит несколько выражений - например if(sample||sample2), при этом вариант sample2 не проверяется/вызывается (если это функция).

3)Оценка покрытия кода с учетом всех условий(Condition) - проверяется покрытие кода на предмет того вызывается ли условный оператор со всеми возможными значениями проверяемых переменных. Данный способ более

адекватен чем предыдущий и позволяет получить объективную оценку покрытия

4)Оценка покрытия с учетом путей выполнения(Path Coverage) - в данном случае считаются все пути, которые выполняются в процессе работы (путь - уникальная последовательность выполнения операторов, с учетом условных операторов)

5)Оценка покрытия с учетом вызовов функций(Function Coverage) - учитываются только уникальные вызовы функций - скажем так очень не объективная оценка покрытия (также ее называют BullsEye coverage)

6)Оценка покрытия с учетом выполнения циклов (Loop Coverage)- данная метрика оценивает выполнялись ли циклы имеющиеся в коде один или более раз или не выполнялись.

7)Оценка покрытия с учетом одновременного выполнения (Race Coverage)- показывает выполняется ли какой либо код одновременно - помогает выявить возможные проблемы в частности Race condition

8)Оценка покрытия с учетом операторов сравнения - выявляет встречаются ли в операторах сравнения (>,<) непредусмотренные значения (например 1>1).

9)Оценка покрытия с учетом выполнения таблицы конечных автоматов - определяет выполнились ли все возможные ветки в таблицы конечного автомата

4. ПРИНЦИП ПРОЕКТИРОВАНИЯ АВТОМАТИЗИРОВАННЫХ СИСТЕМ ТЕСТИРОВАНИЯ

1. Определить требования к системе автоматизированного тестирования. Начало всех начал – выяснение того, что придется делать, иными словами – определение скоупа системы. Для успешного решения этой задачи лучше всего использовать технику мозгового штурма.
2. Определение компонентов системы.
3. Определение компонентов для разработки. На самом деле в реальности ни одна разработка в наше время не делается полностью «с нуля», так как глупо не использовать уже готовые и достаточно стабильные компоненты и библиотеки. Скорее такая разработка сводится в выбору наиболее подходящих для задачи компонентов и написания интеграционного кода для них.
4. Оценка объемов работы
5. Разработка.

5. ОБЗОР СУЩЕСТВУЮЩИХ СИСТЕМ ТЕСТИРОВАНИЯ

UniTESK (Unified Testing & specification toolKit) — технология тестирования программного и аппаратного обеспечения на основе формальных спецификаций, разработанная в Институте системного программирования РАН. Технология представляет собой сочетание хорошо зарекомендовавших себя техник, которые могут применяться в различных комбинациях, взаимно сочетаясь и усиливая друг друга. Это делает технологию гибкой и настраиваемой под существующие процессы разработки на всех этапах жизненного цикла разработки программного обеспечения от сбора и анализа требований до сопровождения.

Основу для вынесения вердиктов о правильности поведения тестируемой системы составляют контрактные спецификации в форме пред- и постусловий, написанные на расширениях традиционных языков программирования, таких как C, Java, и позволяющие выносить вердикт полностью автоматически. Спецификации являются представлением

функциональных требований к системе. Форма спецификаций и основанные на них критерии покрытия обеспечивают прослеживаемость требований.

Успешно использующиеся на практике, техники построения тестов на основе обхода графов состояний позволяют существенно минимизировать количество создаваемого вручную программного кода, вместе с тем обеспечивая разнообразие и массивность тестового набора.

Техники абстракции данных и критерии покрытия, основанные на требованиях, позволяют гибко управлять размером тестового набора и направлять генерацию на покрытие определенных требований, минимизируя тем самым время выполнения тестового набора.

Специальный промежуточный слой, имеющийся в технологии, позволяет быстро настраивать тестовый набор на различные реализации с той же функциональностью.

IBM® Rational® Functional Tester - средство для автоматизированного регрессионного тестирования приложений написанных на Java,.NET, Web- приложений.

HP Quality Center представляет собой законченную интегрированную систему для обеспечения управления процессами контроля качества на всех этапах разработки ПО.

В HP Quality Center включены средства организации и проведения тестирования (ручного). HP Quality Center представляет собой трёхзвенную систему, состоящую из базы данных, сервера приложений и толстый клиент — надстройку для браузера. Из этого можно сделать вывод, что система достаточно громоздка и сложна в обращении.

TestComplite - Один из самых мощных коммерчески продуктов, предназначенных для автоматизации тестирования программного

обеспечения. Это инструмент для автоматизации тестирования, позволяющий создавать и выполнять тесты для любых Windows и Web приложений, как простых, так и обладающих богатой функциональностью. Используя TestComplete, вы с легкостью сможете создавать и автоматизировать тесты для вашего приложения. Автоматизированные тесты работают быстрее, охватывают больше областей тестируемого приложения и снижают затраты на тестирование.

6. СУЩЕСТВУЮЩИЕ ПРОБЛЕМЫ В ТЕСТИРОВАНИИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Главной проблемой тестирования ПО является его трудоемкость. Для анализа программного кода и создания тестового набора с достаточно хорошей степенью покрытия кода тратиться львиная доля всего рабочего времени тестировщика. На сегодняшний момент не существует качественных систем, которые бы составляли тесты в автоматическом режиме по исходному коду. Основная часть программ для тестирования занимается тестированием по уже заданным тестовым наборам. Тестирование должно максимально эффективно охватывать программный код с целью выявления в нем максимального количества ошибок.

Актуальной остается задача создания инструментальных средств, позволяющих :

1) накапливать информации о покрытых и непокрытых ветвях для всех использованных тестов ;

2) выделять разработчику еще не покрытые при тестировании участки программы, облегчая выбор следующих тестов ;

3) поддерживать более мощные критерии полноты структурного тестирования.

7. ОБОБЩЕНИЕ РЕЗУЛЬТАТОВ НАУЧНОГО ПОИСКА И АНАЛИЗА.

Целью магистерской работы является предложить эффективный способ автоматизированной разработки тестовых наборов по критерию C1 – критерию тестирования ветвей программного обеспечения.

Для того, чтобы достигнуть цели следует решить ряд задач, приведенных в списке ниже:

1. Задача синтаксического анализа исходного кода программы с целью выделения в ней ветвей, а так же определения возможных значений параметров условий.

2. Построение управляющего графа программы на основе данных, полученных в предыдущем пункте, для удобства отслеживания степени протестированности найденных ветвей.

3. Разработка тестовых наборов для тестирования ветвей УГП.

Данные задачи не относятся к разряду тривиальных и требуют к себе тщательного внимания и досконального изучения, из за отсутствия подобных систем, разрабатываемый проект можно отнести к проекту с высокой степенью новизны

7.1 Предложения по структуре магистерской работы

Содержание магистерской работы будет состоять из описания работы алгоритма разбора исходного кода на ветви и генерации тестовых наборов, которые смогут покрыть эти ветви, а так же контроля за процессом выполнения задачи.

В работе приводится описание задач, связанных с процессом тестирования программного обеспечения. Для того, чтобы показать связь этого процесса и возникающих в нем задач, отметим его основные этапы:

1. Подготовка тестирования.

2. Проектирование тестов.

3. Разработка тестов.

В работе выделены не все задачи тестирования ПО, а только те из них, которые специфичны для этого процесса, и могут быть интересны с точки зрения научных исследований, а также требуют специальных подходов к решению:

1. Генерация и выбор тестовых данных.
2. Создание модели тестирования.
3. Задача тестового оракула.
4. Выбор критериев тестового покрытия и оценка тестового покрытия.
5. Генерация тестовых сценариев.

Задачи тестирования основаны на следующих постулатах тестирования ПО:

- полное тестирование невозможно;
- стопроцентная автоматизация тестирования невозможна;
- отсутствие общей теории тестирования.

Исследования по генерации тестовых данных ведутся повсеместно, а большое число публикаций приводит к тому, что разрабатываемые подходы достаточно сложно классифицировать. Перечислим наиболее известные подходы:

Список ссылок и литературы:

1. http://www.intuit.ru/department/se/techdevmsnet/14/ «Современные технологии тестирования»
2. http://www.protesting.ru/automation/practice/automation_from_scratch.html «Пишем систему автоматизированных тестов»
3. Г. Мейерс. *Искусство тестирования*. М.: Финансы и статистика, 1982.
4. Кулямин В. В. Критерии тестового покрытия, основанные на структуре контрактных спецификаций // Труды ИСП РАН. Подход UniTESK
5. «Тестирование программного обеспечения. Wikipedia.» http://ru.wikipedia.org/wiki/Тестирование_программного_обеспечения.
6. Дастин Э., Рэшка Д., Пол Д. Автоматизированное тестирование программного обеспечения. Внедрение, управление и эксплуатация. М.: Лори, 2003.
7. Wegener J., Buhr K., Pohlheim H., Automatic test data generation for structural testing of embedded software systems by evolutionary testing /In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2002). NY: Morgan Kauf-mann. 2002, pp. 1233–1240.
8. Pargas R. P., Harrold M. J., Peck R. R. Test-data generation using genetic algorithms
9. Баранцев А.В., Бурдонов И.Б., Демаков А.В. и др. Подход UniTesK к разработке тестов: достижения и перспективы. М.: Труды Института системного программирования РАН.
10. Tracey N., Clark J., Mander K., McDermid J. An automated framework for structural test-data generation /In Proceedings 13th IEEE International Conference Automated Software Engineering. 1998.
11. Черноножкин С.К. Задача автоматического построения тестов и статический анализ//Программирование, 2001, № 2 -с.47-59.
12. Липаев В.В. Тестирование программ. М., Радио и связь, 1986.
13. Корнеев Г.А., Станкевич А.С. Методы тестирования решения задач на соревнованиях по программированию// Вестник II межвузовской конференции молодых ученых. Сборник научных трудов /Под ред. В.Л. Ткалич. — СПб: СПбГУИТМО(ТУ), 2005. том 1.-е. 36-40.
14. Кауфман А.В., Черноножкин С.К.Критерии тестирования и система оценки полноты набора тестов//Программирование, 1998, № 6.-е. 44-59
15. Касьянов В.Н., Евстигнеев В.А. Графы в программировании: обработка, визуализация и применение. СПб.:БХВ-Петербург, 2003.

16. Bueno P.M.S., Jino M. Automatic test data generation for program paths using genetic algorithms//International Journal of Software Engineering and Knowledge Engineering. Vol. 12, No. 6 (2002) pp. 691-709.