***Лекция 2 . Виды и методы тестирования . Критерии выбора тестов и оценки качества ПО.***

**ВИДЫ ТЕСТИРОВАНИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

Все виды тестирования программного обеспечения, в зависимости от преследуемых целей, можно условно разделить на следующие группы:

* Функциональные
* Нефункциональные
* Связанные с изменениями

***Функциональные виды тестирования***

Функциональные тесты базируются на функциях и особенностях, а также взаимодействии с другими системами, и могут быть представлены на всех уровнях тестирования: *компонентном или модульном, интеграционном, системном и приемочном*. Функциональные виды тестирования рассматривают внешнее поведение системы. Далее перечислены одни из самых распространенных видов функциональных тестов:

Функциональное тестирование (Functional testing)

Тестирование безопасности (Security and Access Control Testing)

Тестирование взаимодействия (Interoperability Testing)

***Нефункциональные виды тестирования***

Нефункциональное тестирование описывает тесты, необходимые для определения характеристик программного обеспечения, которые могут быть измерены различными величинами. В целом, это тестирование того, "Как" система работает. Далее перечислены основные виды нефункциональных тестов:

* Все виды тестирования производительности:
  + нагрузочное тестирование (Performance and Load Testing)
  + стрессовое тестирование (Stress Testing)
  + тестирование стабильности или надежности (Stability / Reliability Testing)
  + объемное тестирование (Volume Testing)
* Тестирование установки (Installation testing)
* Тестирование удобства пользования (Usability Testing)
* Тестирование на отказ и восстановление (Failover and Recovery Testing)
* Конфигурационное тестирование (Configuration Testing)

***Связанные с изменениями виды тестирования***

После проведения необходимых изменений, таких как исправление бага/дефекта, программное обеспечение должно быть пере тестировано для подтверждения того факта, что проблема была действительно решена. Ниже перечислены виды тестирования, которые необходимо проводить после установки программного обеспечения, для подтверждения работоспособности приложения или правильности осуществленного исправления дефекта:

* Дымовое тестирование (Smoke Testing)
* Регрессионное тестирование (Regression Testing)
* Тестирование сборки (Build Verification Test)

Санитарное тестирование или проверка согласованности/исправности (Sanity Testing)

**КРИТЕРИИ ВЫБОРА ТЕСТОВ И ОЦЕНКИ КАЧЕСТВА ПО**

# Требования к идеальному критерию тестирования

Требования к идеальному критерию:

* *Критерий должен быть достаточным*, т.е. показывать, когда некоторое конечное множество тестов достаточно для тестирования данной программы.
* *Критерий должен быть полным*, т.е. в случае ошибки должен существовать тест из множества тестов, удовлетворяющих критерию, который раскрывает ошибку.
* *Критерий должен быть надежным*, т.е. любые два множества тестов, удовлетворяющих ему, одновременно должны раскрывать или не раскрывать ошибки программы.
* *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.

# КЛАССЫ КРИТЕРИЕВ

1. Структурные критерии используют информацию о структуре программы (критерии так называемого «белого ящика»).
2. Функциональные критерии формулируются в описании требований к программному изделию (критерии так называемого «черного ящика»),
3. Критерии стохастического тестирования формулируются в терминах проверки наличия заданных свойств у тестируемого приложения, средствами проверки некоторой статистической гипотезы.
4. Мутационные критерии ориентированы на проверку свойств программного изделия на основе подхода Монте-Карло.

## Структурные критерии (класс I)

Структурные критерии используют модель программы в виде «белого ящика», что предполагает знание исходного текста программы или спецификации программы в виде потокового графа управления. Структурная информация понятна и доступна разработчикам подсистем и модулей приложения, поэтому данный класс критериев часто используется на этапах модульного и интеграционного тестирования (Unit testing, Integration testing).

Структурные критерии базируются на основных элементах УГП (управляющий граф программы), операторах, ветвях и путях:

* Условие критерия *тестирования команд* (критерий С0) – набор тестов в совокупности должен обеспечить прохождение каждой команды не менее одного раза. Это слабый критерий, он, как правило, используется в больших программных системах, где другие критерии применить невозможно.
* Условие критерия *тестирования ветвей* (критерий С1) – набор тестов в совокупности должен обеспечить прохождение каждой ветви не менее одного раза. Это достаточно сильный и при этом экономичный критерий, поскольку множество ветвей в тестируемом приложении конечно и не так уж велико. Данный критерий часто используется в системах автоматизации тестирования.
* Условие критерия *тестирования путей* (критерий С2) – набор тестов в совокупности должен обеспечить прохождение каждого пути не менее одного раза. Если программа содержит цикл (в особенности с неявно заданным числом итераций), то число итераций ограничивается константой (часто – 2, или числом классов выходных путей).

***Пример.*** Рассмотрим условия тестирования простой программы в соответствии со структурными критериями.

**1** public void Method (ref int x) {

**2** if (x>17)

**3** x = 17-x;

**4** if (x==-13)

**5** x = 0; }

Тестовый набор из одного теста, удовлетворяет критерию команд (С0):

(X,Y)={(xBX=30, хвых=0)} покрывает все операторы трассы 1-2-3-4-5-6.

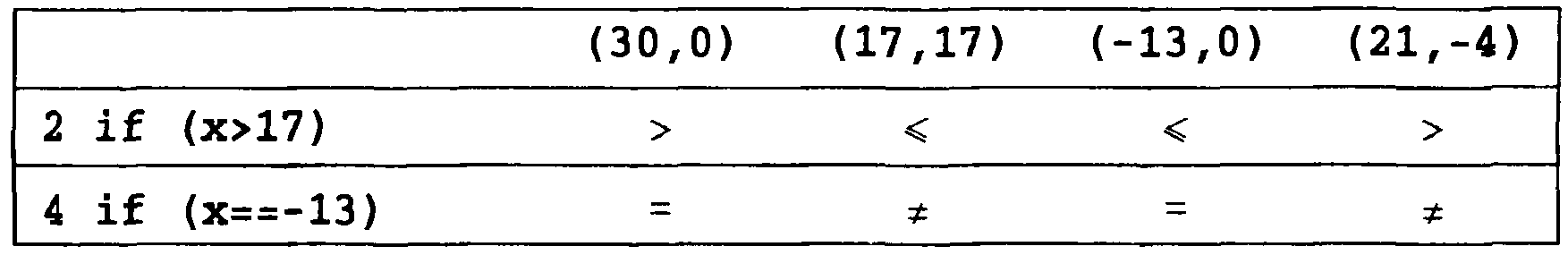
Тестовый набор из двух тестов, удовлетворяет критерию ветвей (С1):

(X,Y)={(30,0), (17,17)} добавляет 1 тест к множеству тестов для С0 и трассу 1-2-4-6. Трасса 1-2-3-4-5-6 проходит через все ветви достижимые в операторах if при условии true, а трасса 1-2-4-6 – через все ветви, достижимые в операторах if при условии false.

Тестовый набор из четырех тестов, удовлетворяет критерию путей (С2):

(X,Y)=<(30,0), (17,17), (-13,0), (21,-4)}.

Набор условий для двух операторов if с метками 2 и 4 приведен в следующей таблице



Критерий ветвей С2 проверяет программу более тщательно, чем критерии – С1, однако даже в случае, если он удовлетворен, нет оснований утверждать, что программа реализована в соответствии со спецификацией.

Например, в случае, если спецификация задает условие, что , невыполнимость которого можно подтвердить на тесте (-177,-177). Действительно, операторы 3 и 4 на тесте (-177,-177) не изменят величину и результат не будет соответствовать спецификации. Структурные критерии не проверяют соответствие спецификации, если оно не отражено в структуре программы. Поэтому при успешном тестировании программы по критерию С2 мы можем не заметить ошибку, связанную с невыполнением некоторых условий спецификации требований.

## Функциональные критерии (класс II)

Функциональный критерий — важнейший для программной индустрии критерий тестирования. Он обеспечивает, прежде всего, контроль степени выполнения требований заказчика в программном продукте. Поскольку требования формулируются к продукту в целом, они отражают взаимодействие тестируемого приложения с окружением. При функциональном тестировании преимущественно используется модель «черного ящика». Проблема функционального тестирования – это, прежде всего, трудоемкость; дело в том, что документы, фиксирующие требования к программному изделию (Software requirement specification, Functional specification и т.п.), как правило, достаточно объемны, тем не менее, соответствующая проверка должна быть всеобъемлющей.

Ниже приведены частные виды функциональных критериев.

* *Тестирование пунктов спецификации* – набор тестов в совокупности должен обеспечить проверку каждого тестируемого пункта не менее одного раза. *Спецификация требований* может содержать сотни и тысячи пунктов требований к программному продукту, и каждое из этих требований при тестировании должно быть проверено в соответствии с критерием не менее чем одним тестом.
* *Тестирование классов входных данных* – набор тестов в совокупности должен обеспечить проверку представителя каждого класса входных данных не менее одного раза. При создании тестов классы входных данных сопоставляются с режимами использования тестируемого компонента или подсистемы приложения, что заметно сокращает варианты перебора, учитываемые при разработке тестовых наборов. Следует заметить, что, перебирая в соответствии с критерием величины входных переменных (например, различные файлы - источники входных данных), мы вынуждены применять мощные тестовые наборы. Действительно, наряду с ограничениями на величины входных данных, существуют ограничения на величины входных данных во всевозможных комбинациях, в том числе проверка реакций системы на появление ошибок в значениях или структурах входных данных. Учет этого многообразия – процесс трудоемкий, что создает сложности для применения критерия
* *Тестирование правил — набор тестов в совокупности должен обес*печить проверку каждого правила, если входные и выходные значения описываются набором правил некоторой грамматики. Следует заметить, что грамматика должна быть достаточно простой, чтобы трудоемкость разработки соответствующего набора тестов была реальной (вписывалась в сроки и штат специалистов, выделенных для реализации фазы тестирования).
* *Тестирование классов выходных данных* – набор тестов в совокупности должен обеспечить проверку представителя каждого выходного класса, при условии, что выходные результаты заранее расклассифицированы, причем отдельные классы результатов учитывают, в том числе, ограничения на ресурсы или на время (time out). При создании тестов классы выходных данных сопоставляются с режимами использования тестируемого компонента или подсистемы, что заметно сокращает варианты перебора, учитываемые при разработке тестовых наборов.
* *Тестирование функций* – набор тестов в совокупности должен обеспечить проверку каждого действия, реализуемого тестируемым модулем, не менее одного раза. Очень популярный на практике критерий, который, однако, не обеспечивает покрытия части функциональности тестируемого компонента, связанной со структурными и поведенческими свойствами, описание которых не сосредоточено в отдельных функциях (т.е. описание рассредоточено по компоненту). Критерий тестирования функций отчасти объединяет особенности структурных и функциональных критериев. Он базируется на модели «полупрозрачного ящика», где явно указаны не только входы и выходы тестируемого компонента, но также состав и структура используемых методов (функций, процедур) и классов.
* *Комбинированные критерии для программ и спецификаций* – набор тестов в совокупности должен обеспечить проверку всех комбинаций непротиворечивых условий программ и спецификаций не менее одного раза.

При этом все комбинации непротиворечивых условий надо подтвердить, а условия противоречий следует обнаружить и ликвидировать.

## Стохастические критерии (класс III)

**Стохастическое тестирование** применяется при тестировании сложных программных комплексов – когда набор детерминированных тестов (X,Y) имеет громадную мощность. В случаях, когда подобный набор невозможно разработать и исполнить на фазе тестирования, можно применить следующую методику:

* Разработать программы – имитаторы случайных последовательностей входных сигналов {х}.
* Вычислить независимым способом значения {у} для соответствующих входных сигналов {х} и получить тестовый набор (X,Y).
* Протестировать приложение на тестовом наборе (X,Y), используя два способа контроля результатов:
  + *Детерминированный контроль* – проверка соответствия вычисленного значения {ув} значению у, полученному в результате прогона теста на наборе {х} — случайной последовательности входных сигналов, сгенерированной имитатором.
  + Стохастический контроль – проверка соответствия множества значений {ув}, полученного в результате прогона тестов на наборе входных значений {х}, заранее известному распределению результатов F(Y). В этом случае множество Y неизвестно (его вычисление невозможно), но известен закон распределения данного множества. (Критерии стохастического тестирования [3, c. 36]

## Мутационный критерий (класс IV)

Постулируется, что профессиональные программисты пишут сразу почти правильные программы, отличающиеся от правильных мелкими ошибками или описками типа – перестановка местами максимальных значений индексов в описании массивов, ошибки в знаках арифметических операций, занижение или завышение границы цикла на 1 и т.п. Предлагается подход, позволяющий на основе мелких ошибок оценить общее число ошибок, оставшихся в программе.

Подход базируется на следующих понятиях:

*Мутации* – мелкие ошибки в программе.

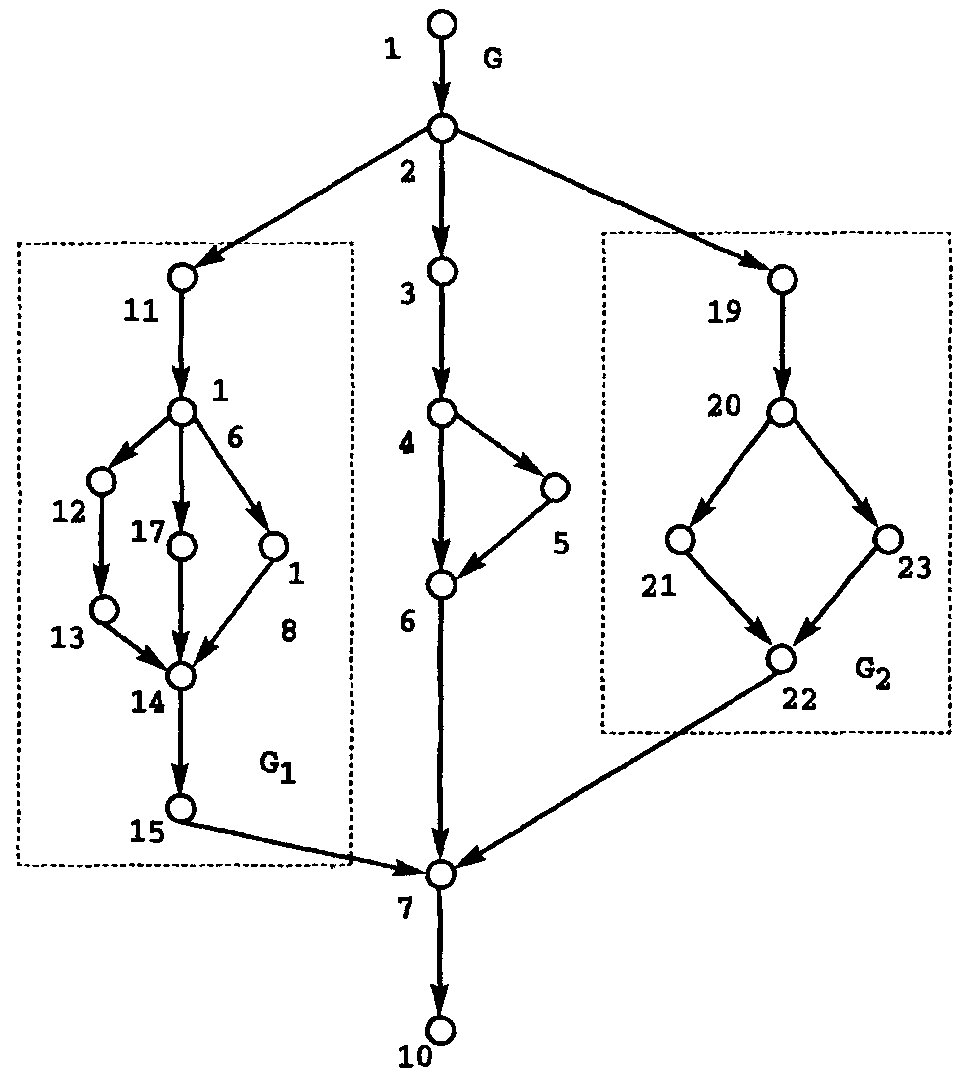
*Мутанты –* программы, отличающиеся друг от друга мутациями.

*Метод мутационного тестирования* – в разрабатываемую программу Р вносят мутации, т.е. искусственно создают программы-мутанты Р1,Р2... Затем программа Р и ее мутанты тестируются на одном и том же наборе тестов (Х,У).

Если на наборе (X,Y) подтверждается правильность программы Р и, кроме того, выявляются все внесенные в программы-мутанты ошибки, то *набор тестов (XY) соответствует мутационному критерию*, а тестируемая программа объявляется правильной.

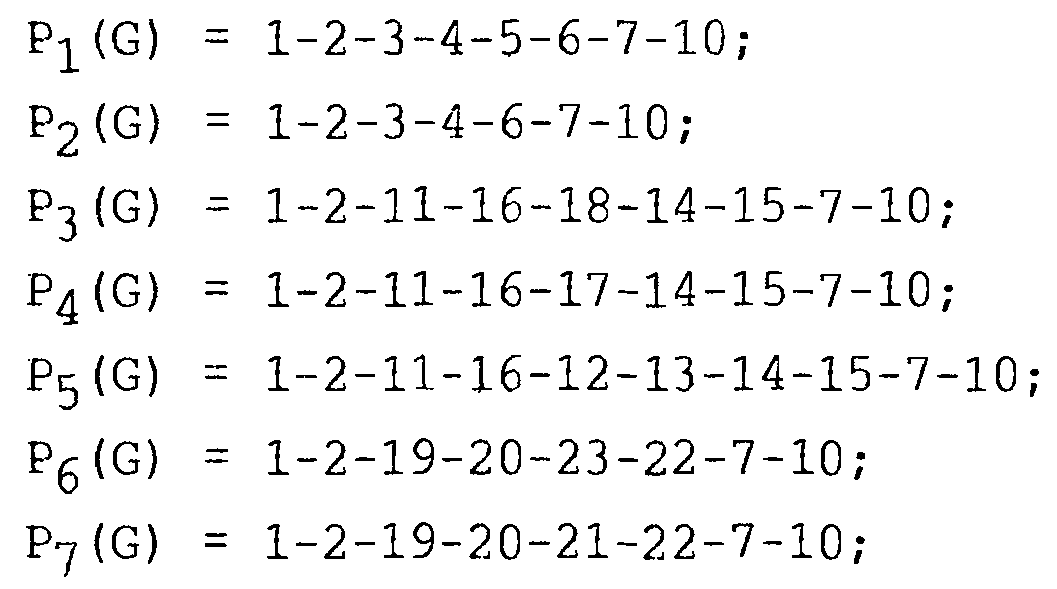
Если некоторые мутанты не выявили всех мутаций, то надо расширять набор тестов (Х,У) и продолжать тестирование.

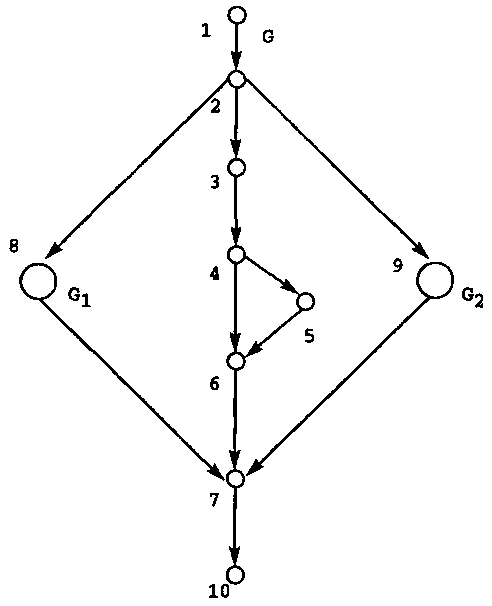
# ОЦЕНКА КАЧЕСТВА ПО

Рассмотрим две модели ПО, используемые при оценке оттестированности.

Для оценки степени оттестированности часто используется УГП – управляющий граф программы. УГП многокомпонентного объекта G, который содержит внутри себя два компонента G1 и G2, УГП которых раскрыты.

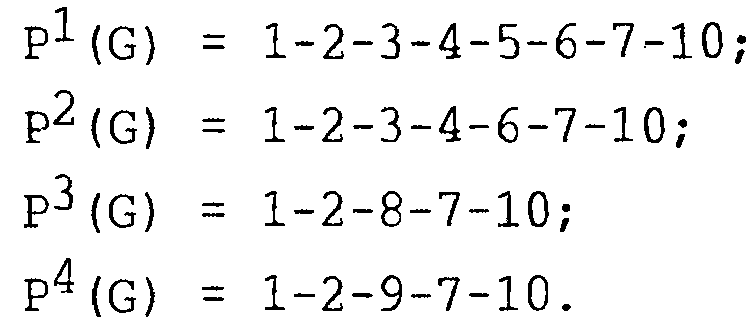
В результате УГП компонента G имеет такой вид, как если бы компоненты G1 и G2 в его структуре специально не выделялись, а УГП компонентов G1 и G2 были вставлены в УГП G. Для тестирования компонента G в соответствии с критерием путей потребуется прогнать тестовый набор, покрывающий следующий набор трасс графа G:



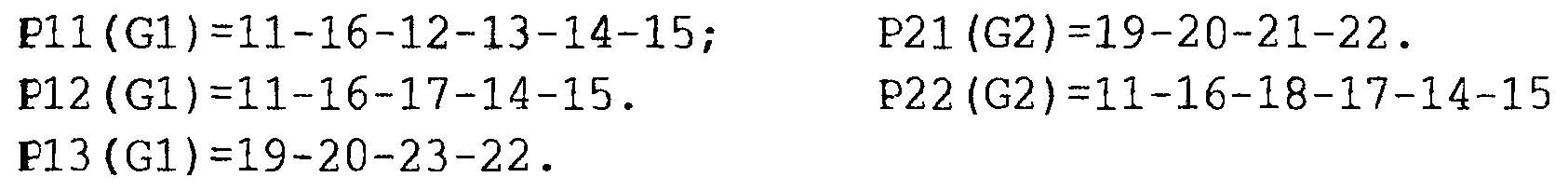
УГП компонента G, представленный в виде иерархической модели.

В иерархическом УГП G входящие в его состав компоненты представлены ссылками на свои УГП G1 и G2

Для исчерпывающего тестирования иерархической модели компонента G в соответствии с критерием путей требуется прогнать следующий набор трасс:



Приведенный набор трасс достаточен при условии, что компоненты G1 и G2 в свою очередь исчерпывающе протестированы. Чтобы обеспечить выполнение этого условия в соответствии с критерием путей, надо прогнать все трассы:



## Методика интегральной оценки тестированности

* Выбор критерия С и приемочной оценки тестированности программного проекта – L.
* Построение дерева классов проекта и построение УГП для каждого модуля.
* Модульное тестирование и оценка TV на модульном уровне.
* Построение УГП, интегрирующего модули в единую иерархическую (классовую) модель проекта.
* Выбор тестовых путей для проведения интеграционного или системного тестирования.
* Генерация тестов, покрывающих тестовые пути шага 5.
* Интегральная оценка тестированности проекта с учетом оценок тестированности модулей-компонентов.
* Повторение шагов 5-7 до достижения заданного уровня тестированности L