Лекция 5. Процесс тестирования

5.1. Процесс тестирования: потоки информации

Процесс тестирования и отладки ПО можно представить следующим образом (рис. 5.1).

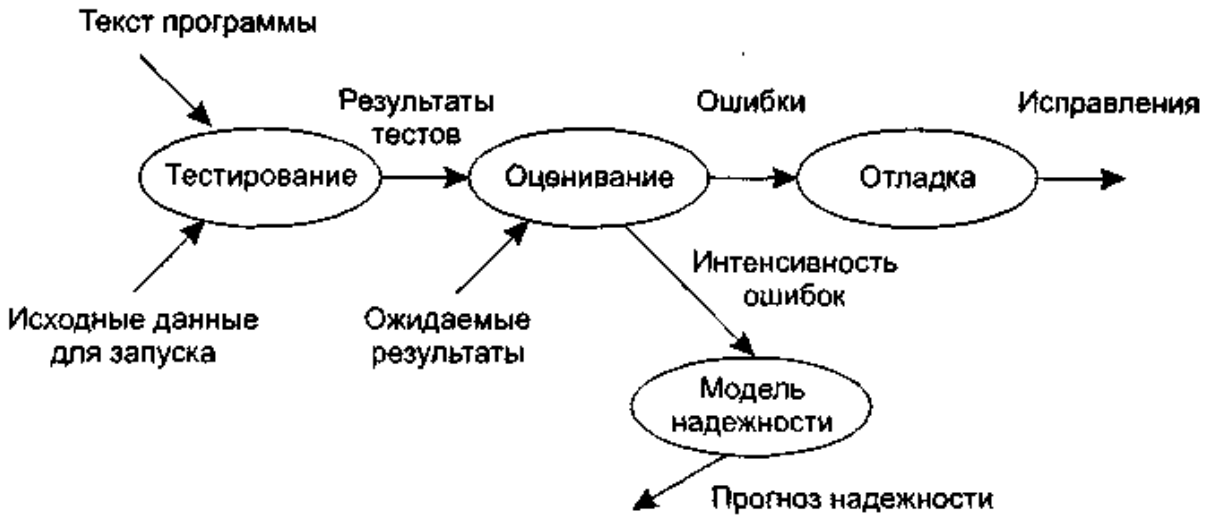


Рисунок 5.1 – Информационные потоки процесса тестирования

Для того, чтобы убедиться, что программа работает, необходим *код* этой программы и *исходные данные* для запуска программы. Но правильность работы нельзя оценить без эталонов – *ожидаемых результатов*. Совокупность исходных данных для запуска программы и соответствующий им набор ожидаемых результатов работы ПП называется **тестом** (тестовым вариантом, тестовым набором).

В случае несовпадения полученных и ожидаемых результатов фиксируется *ошибка.* Описание ошибки заносится в отчет об имеющихся проблемах. Далее следует ее локализация и отладка программы. После внесения исправлений тестирование повторяют. Чтобы этот процесс не оказался бесконечным, ещё до начала тестирования необходимо сформулировать критерии его завершения.

Собранная в ходе тестирования информация об интенсивности ошибок служит базой для построения модели надежности.

5.2. Отчет об имеющихся проблемах (баг-репорт)

«Репорт» переводится как жалоба, отчет, сообщение, доклад, репутация, донесение, рапорт, слух. **Баг-репорт** – это технический документ, отчет об имеющихся проблемах (обнаруженных ошибках в работе ПО). Его задача показать, как воспроизводится проблема, помочь понять ее причину и важность устранения. **Баг-трекер** (система отслеживания ошибок) – программный инструмент, с помощью которого фиксируется информация о выявленных проблемах.

Правила оформления записей в баг-репорте в каждой компании свои. Они зависят от политики компании, технологий разработки, используемого инструмента, типа проекта и т.п. Однако, описание проблемы должно быть простым, понятным, технически грамотным.

Прежде, чем сделать запись, следует убедиться, что *ошибка воспроизводится.* Далее необходимо *минимизировать число шагов* для воспроизведения. Путей воспроизведения может быть несколько. Выбираем самый короткий и простой. Иногда бывает полезно разнести разные пути воспроизведения в разные записи. Возможно, они инициированы разными проблемами в коде. Если ошибка не воспроизводится, вероятно, что-то вы не учли. Может быть, ошибка воспроизводится статистически (т.е. конкретного алгоритма как ее получить – нет, но в течение некоторого времени она проявляется). В некоторых компаниях такого рода ошибки собирают, чтобы спрогнозировать проблемы.

Как правило, в число **обязательных полей** отчета об ошибках входят: короткое описание (Bug Summary), серьезность (Severity), шаги к воспроизведению (Steps to reproduce), результат (Actual Result), ожидаемый результат (Expected Result). Остановимся на них чуть подробнее.

Заголовок (короткое описание). Здесь необходимо в одном предложении коротко и ясно сказать, **где** *(в* каком компоненте архитектуры, месте интерфейса пользователя)**, что** (происходит или не происходит, пишем про наличие или отсутствие проблемы, а не её содержание) **и когда** (в какой момент времени, при каких условиях, событиях) не работает. Формулировка "Где? Что? Когда?" облегчает сортировку и систематизацию дефектов, исключение их дублирования.

Например,
Где: на странице NNN
Что: неправильный расчет данных
Когда: после ввода в поле Y отрицательного значения.

Уточнить детали, которые пришлось опустить в заголовке (версию ПП, в котором проявляется проблема; очевидную причину ошибки и т.д.), можно в поле **«*Подробное описание*»,** если таковое имеется.

Серьезность (Severity) и /или ***приоритет*** (Priority)**.** Наличие этих полей и значения в них отличаются от багтрекера к багтрекеру. Приоритет показывает срочность и важность исправления ошибки. Серьезность – это критичность ошибки с точки зрения её влияния на работоспособность ПП: опечатка в тексте, мелкая проблема, значительная проблема и т.п. В большинстве трекеров представлены следующие в*иды критичности*:

- *блокирующая* ошибка (Immediate, Blocker). Приводит приложение в нерабочее состояние и не позволяет полноценно проводить его тестирование. Обычно выявляется в ходе первичного запуска новой версии ПП;
- *критическая* ошибка (Crit – Urgent) – имеются проблемы с выполнением ключевых функций приложения, но тестирование может быть продолжено;
- *значительная* ошибка (High), но не критичная. Есть проблемы вызова тестируемой функции, однако существуют другие способы ее активизации. И работа с приложением может быть продолжена;
- *незначительная* ошибка (Normal) – обычно очевидная проблема пользовательского интерфейса и локализации. Не затрагивает бизнес-логику тестируемого компонента;
- *тривиальная* ошибка (Low) – малозаметна, плохо воспроизводится, связана с проблемами сторонних библиотек или сервисов.

Если в багтрекере есть оба поля, то тестировщик, как правило, выставляет только Severity, а Priority – старший тестировщик/старший программист/менеджер или любой другой ответственный за это дело человек.

Шаги для воспроизведения. Это **основное поле** для заполнения в баг-репорте. Шаги следует описать с упоминаем всех вводимых пользователем данных и промежуточных результатов. Шагов должно быть достаточно для воспроизведения проблемы. К репорту присоединяют, если требуется, исходные файлы, скриншоты с визуальным отображением проблемы. На скриншотах место с ошибкой лучше указать.

В полях ***Результат*** и ***Ожидаемый результат*** вводят соответственно то, что получили, и то, что должны были получить.

Многие системы баг-трекинга по умолчанию выставляют ***Статус (Status)*** ошибки. Это состояние, в котором находится обнаруженная ошибка: проблема зарегистрирована – подтвержден экспертом факт наличия ошибки – понятна вызвавшая ошибку причина – принято решение о начале работы над исправлением – ошибка исправлена – исправление интегрировано в основное пространство – подтверждено исправление ошибки

Кроме выше перечисленных, в баг-репорт обычно **включены поля**: Проект (Project), Компонент приложения (Component), Номер версии (Version, affect version), Автор (Author) баг репорта, Назначен на (Assigned To, кому исправлять), Версия программно-аппаратного окружения (Environment), в котором проявляется проблема: версия операционной системы, наличие сервис-паков, разрядность, версия браузера, установленные плагины и т.д.; Прикрепленный файл (Attachment), если прикрепленные данные помогут в исправлении бага.

5.3. Критерии завершения тестирования

Процесс тестирования, исходя из принципа «ошибки в программе есть», потенциально бесконечен. Но в соответствии с планом проект должен быть завершен, а ПП передан заказчику. Увеличение длительности процесса тестирования не только срывает планы, но и приводит к удорожанию ПП. Встает вопрос, когда можно завершать тестирование, чтобы выдержать сроки и стоимость проекта и обеспечить требуемый уровень качества ПП.

Если разработка ведётся в рамках каскадного подхода, то период тестирования и тест-план определены изначально. Нередко длительность тестирования определяют по остаточному принципу. В любом случае следует провести минимальное тестирование: проверить покрытие функций, граничных значений; конфигурацию технических средств; устойчивость к ошибкам пользователя.

При других подходах можно воспользоваться критериями завершения тестирования. Выделяют **три группы критериев**[6]**:**

1. основанные ***на методологиях проектирования тестов*** – определенное количество тестов, полученных по методам анализа причинно-следственных связей, анализа граничных значений и предположения об ошибке, перестают выявлять ошибки; признаком возможности окончания тестирования является полнота охвата тестами, пропущенными через ПП, множества различных ситуаций и относительно редкое проявление ошибок на последнем отрезке процесса тестирования.

2. основанные ***на оценке возможного количества ошибок***: тестирование завершают при обнаружении 93-95% ошибок, возможное количество которых оценивают экспертно или по специальным методикам (например, методике Миллса).

3. основанные ***на динамике выявления ошибок***. Тестирование завершают, если количество вновь обнаруживаемых ошибок приблизилось к минимуму (рис.5.2).

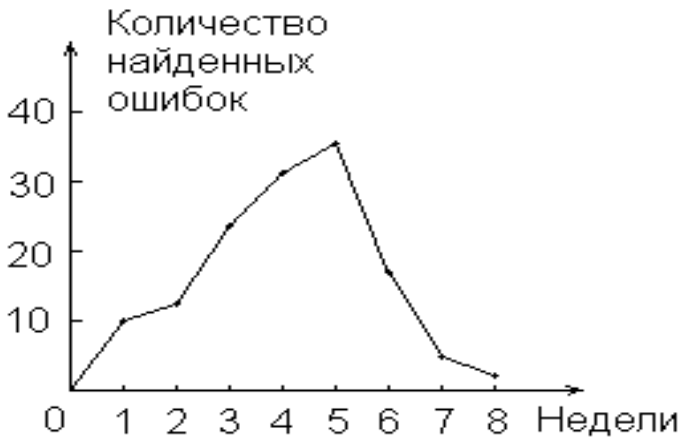


Рисунок 5.2 **–** Динамика выявления ошибок

Критерии можно комбинировать. Допустимое количество ошибок определяется в соответствии с указанной в спецификации качества надежностью ПП.

Во всех случаях следует обратить внимание на ***критичность*** ошибки – это степень относительного влияния, которое ошибка оказывает на работу программного обеспечения. Критичность ошибки может быть высокой (Hight), средней (Medium) или низкой (Low). Ошибка *высокой* критичности приводит к невозможности или некорректной работе важной функции в программе, к повреждению или потере данных. Невозможно использовать обходные методы, нет доступных решений, которые позволят моментально устранить проблему. *Средней* – предполагает частичную потерю функциональности программного обеспечения или некорректное выполнение отдельных операций. Возможно временное использование обходного метода, пока служба поддержки пытается устранить проблему. *Низкая* критичность соответствует незначительным проблемам, таким как ошибки в документации, общие вопросы использования, рекомендации по усовершенствованию или модификации продукта.

Например, в таблице 5.1 и на рисунке 5.3 представлены результаты тестирования реального ПП (проект одной из вологодских компаний). Тестирование выполнялось методом «черного ящика» по критерию покрытия входных/выходных данных. График показывает, что тестирование можно завершать, но так как последняя выявленная ошибка имеет высокий приоритет, то тестирование было продолжено.

Таблица 5.1 – Результаты тестирования ПП по периодам

Критичность ошибки	Период (с июня 2009 по февраль 2010)									Итого
	VI	VII	VIII	IX	X	XI	XII	I	II	
Hight	0	7	8	10	19	16	17	4	1	82
Medium	6	1	5	4	9	6	2	1	0	34
Low	1	1	1	0	3	0	0	0	0	6
Итого	7	9	14	14	31	22	19	5	1	122

Рисунок 5.3 **–** Динамика выявления ошибок **ПП**

5.3. Модели надежности ПП

Модели надежности отображают качество ПП и основные закономерности изменения числа обнаруживаемых в нем ошибок и имеют вероятностный характер. С помощью модели можно приближенно оценить ожидаемую надежность функционирования ПП в процессе испытаний и эксплуатации, число не выявленных ошибок и время, необходимое для их обнаружения. Точность модели растёт с увеличением интенсивности выявления ошибок (т.е. при невысоком качестве ПП).

На рисунке 4.3 приведена классификация моделей надежности ПП.

Выделены две группы моделей: аналитические и эмпирические. ***Эмпирические*** (или *феноменологические)* модели устанавливают связь между характеристиками ПП и его надежностью и базируются на анализе структуры программы: количество точек ветвления, циклов, межмодульных связей и т.д. При этом предполагается, что связь между надежностью и другими параметрами является статической. Н*апример*, с ростом сложности программы уменьшается уровень ее надежности (возрастает вероятность ошибок программистов, трудность их обнаружения и устранения). ***Аналитические*** модели представляют собой формулы, связывающие показатели надежности с результатами тестирования. В *динамических* моделях появление отказов рассматривается с учётом фактора времени. *В статических* моделях момент выявления ошибок в процессе тестирования не учитывается, но могут использоваться зависимости количества ошибок от числа тестовых прогонов или от характеристики входных данных.

Динамические модели делят на *непрерывные* (если фиксируются интервалы времени между каждыми предыдущим и последующим отказами и получается непрерывная картина появления отказов во времени) и *дискретные* (фиксируют только число отказов за произвольный интервал времени, соответственно поведение ПП будет представлено только в дискретных точках).

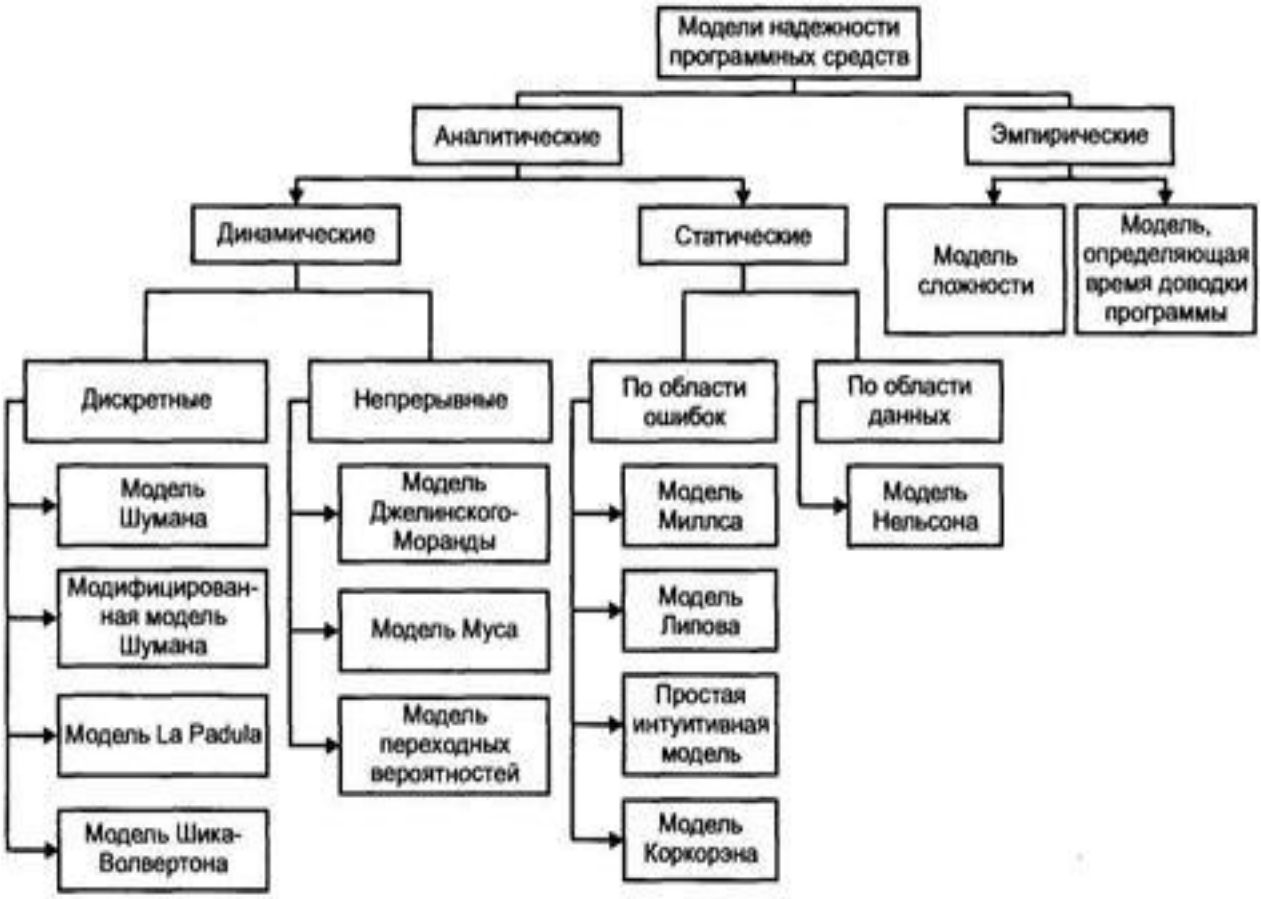


Рисунок 4.3 – Классификационная схема моделей надежности ПП

В качестве примера *дискретной динамической* модели рассмотрим ***модель Шумана*** (экспоненциальную модель). Модель Шумана основана на следующих допущениях:

1. общее число команд в программе на машинном языке постоянно;
2. в начале компоновочных испытаний число ошибок равно некоторой постоянной величине, и по мере исправления ошибок их становится меньше. В ходе испытаний программы новые ошибки не вносятся;
3. ошибки изначально различимы, по суммарному числу исправленных ошибок можно судить об оставшихся;
4. интенсивность отказов программы пропорциональна числу остаточных ошибок.

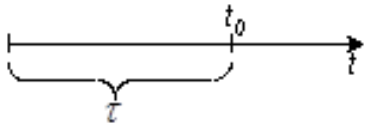
В соответствии с предположением 2 будем считать, что в начальный момент компоновки программных средств системы в них имеются небольшие ошибки (*E* – количество ошибок). С этого времени отсчитывается время отладки τ, которое включает затраты времени на выявление ошибок с помощью тестов, на контрольные проверки и т.д. При этом время исправного функционирования системы не учитывается. В течение времени τ устанавливается $\varepsilon_0(\tau)$ ошибок в расчете на одну команду машинного языка. Т.о. удельное число ошибок на одну машинную команду, остающихся в системе после времени работы τ равно

$$\varepsilon_\tau(\tau) = \frac{E}{I} - \varepsilon_0(\tau),$$

где I – общее число машинных команд.

Исходя из предположения 4, значение функции частоты или интенсивности отказов $z(t)$ пропорционально числу ошибок, оставшихся в программном обеспечении после израсходования на отладку времени τ, то есть

$$z(t) = C\ \varepsilon_\tau(\tau),$$



где *C* – коэффициент пропорциональности.

Тогда, если время работы системы *t* отсчитывается от момента времени t_0, а τ остается фиксированным (τ=const), то функция надежности или вероятность безотказной работы на интервале времени от 0 до *t* есть

$$R(t,\tau) = \exp\left\{-C\left[\frac{E}{I} - \varepsilon_0(\tau)\right]t\right\}$$

Для нахождения *C* и *E* используются принцип максимального правдоподобия (пропорция).

Модель ***Джелинского-Моранда*** – это *непрерывная динамическая* модель с дискретным убыванием интенсивности отказов. Предполагается, что интенсивность ошибок описывается кусочно-постоянной функцией, пропорциональной числу не устраненных ошибок, т.е. интенсивность отказов $\lambda(t)$ постоянна до обнаружения и исправления ошибки, после чего она опять становится постоянной, но с другим, меньшим, значением. При этом предполагается, что между $\lambda(t)$ и числом оставшихся в программе ошибок существует прямая зависимость

$$\lambda(t) = k(M - i) = \lambda_i,$$

где *M* – неизвестное первоначальное число ошибок,

i – число обнаруженных ошибок, зависящих от времени *t,*

k – константа

Частота обнаружения *i*-ой ошибки t_i задается соотношением

$$a_i(t) = \lambda_i e^{-\lambda_i t_i}$$

Значения неизвестных параметров *k* и *M* может быть оценено на основе последовательности наблюдений интервалов между моментами обнаружения ошибок по методу максимального правдоподобия.

Модель Миллса – пример статической модели надежности. Она построена на принципе «посева» ошибок и предположении о том, что все ошибки (как естественные, так и искусственно внесенные) имеют равную вероятность быть найденными в процессе тестирования. Модель Миллса позволяет предсказать количество ошибок в программе на момент начала тестирования и установить доверительный уровень этого прогноза.

Перед началом тестирования в программу вносят некоторое количество дополнительных ошибок. Ошибки вносятся случайным образом и фиксируются в протоколе искусственных ошибок. Специалист, проводящий последующее

тестирование, не знает ни количества, ни характера внесенных ошибок до момента оценки показателей надежности по модели Миллса.

На основе протокола искусственных ошибок все обнаруженные в ходе тестирования ошибки делят на собственные (допущенные программистами и не выявленные ранее) и искусственные (внесенные перед началом тестирования). Обозначим: N – первоначальное количество ошибок в программе (число собственных ошибок), S – количество искусственно внесенных ошибок, n – число найденных собственных ошибок, V – число обнаруженных к моменту оценки искусственных ошибок. Исходя из предположения о равной вероятности обнаружения ошибок, запишем пропорцию n/N =V/S , откуда

$$N = \frac{S \cdot n}{V}.$$

Например, если в программу внесено 20 ошибок и к некоторому моменту тестирования обнаружено 15 собственных и 5 внесенных ошибок, значение N можно оценить в 60.

Вторая часть модели связана с выдвижением и проверкой гипотезы о количестве собственных ошибок в программе. Мерой доверия к модели является величина C (степень отлаженности программы), которая показывает вероятность того, насколько правильно найдено значение N.

Предположим, что в программе имеется *K* собственных ошибок (оставим за N обозначение числа собственных ошибок, вычисляемого по формуле Миллса) и еще в нее внесено *S* искусственных ошибок. Далее возможны два варианта развития событий:

а) Программа тестируется, пока не будут обнаружены все внесенные ошибки, подсчитывается число обнаруженных собственных ошибок. Тогда по формуле Миллса мы предполагаем, что первоначально в программе было *N = n* ошибок. Вероятность C, с которой можно высказать такое предположение, рассчитывают по следующему соотношению:

$$C = 1, \quad \text{если} \quad n > K$$

$$C = \frac{S}{S + K + 1}, \quad \text{если} \quad n <= K$$

Например, утверждается, что в программе нет ошибок (K = 0), вносится 4 ошибки в программу, при тестировании все они обнаруживаются и при этом не находятся собственные ошибки программы, тогда C = 0.8. Чтобы достичь уровня 95 %, необходимо было бы внести в программу 19 ошибок.

б) Не все искусственно рассеянные ошибки обнаружены при тестировании. В этом случае следует использовать более сложное комбинаторное выражение. Величина C рассчитывается по модифицированной формуле

$$C = 1, \quad \text{если} \quad n > K$$

$$C = \frac{\frac{S}{V - 1}}{\left(\frac{S + K + 1}{V + K}\right)}, \text{если} \quad n <= K$$

,

где и числитель, и знаменатель формулы при *n* <= *K* являются биноминальными коэффициентами вида:

$$\frac{a}{b} = \frac{a!}{b!(a - b)!}$$

(В комбинаторике биномиальный коэффициент C_n^k для неотрицательных целых чисел *n* и *k* интерпретируется как количество сочетаний из *n* по *k*, то есть количество всех подмножеств (выборок) размера *k* в *n*-элементном множестве (т.е. количество неупорядоченных наборов).)

Достоинством модели является простота применения математического аппарата, наглядность и возможность использования в процессе тестирования. Легко представить программу внесения ошибок, которая случайным образом выбирает модуль и вносит логические ошибки, изменяя или убирая операторы (природа внесения ошибок должна оставаться в тайне, все внесенные ошибки их следует регистрировать).

Основными **недостатками** модели являются:

1) необходимость внесения искусственных ошибок. Этот процесс плохо формализуется. Исходя из предположения об одинаковой вероятности обнаружения собственных и внесенных ошибок, следует, что внесенные ошибки должны быть типичными, но непонятно какими именно они должны быть;

2) предположение об одинаковой вероятности обнаружения собственных и внесенных ошибок. Это предположение невозможно проверить, особенно на более поздних стадиях разработки программ, когда многие простые ошибки (например, синтаксические) уже исправлены, и только наиболее сложные для обнаружения ошибки ещё не найдены;

3) достаточно вольное допущения величины K, которое основывается исключительно на интуиции и опыте человека, проводящего оценку, т.е. допускается большое влияние субъективного фактора.

Однако по сравнению с проблемами других моделей эта проблема кажется не очень сложной и разрешимой.

Существуют другие классификации моделей, например, предсказывающие, прогнозные, измеряющие и т.д. Некоторые модели, основанные на информации, полученной в ходе тестирования ПО, дают возможность делать прогнозы его поведения в процессе эксплуатации.

Таким образом, на входе процесса тестирования имеем программу (или иной объект тестирования), исходные данные для ее запуска и ожидаемые результаты, а на выходе – выявленные ошибки и вероятностную оценку надёжности ПП.

Встает вопрос, какое количество и каких тестов необходимо, чтобы обеспечить требуемую надежность ПП? Исчерпывающее тестирование гарантирует полную проверку программы. Однако проверка работоспособности программы на всех возможных наборах исходных данных требует очень больших затрат времени и в большинстве случаев невозможна: слишком много тестов надо выполнить.

Решение подсказала системология: для уменьшения количества тестов необходимо все возможные варианты выполнения программы разбить на классы, эквивалентные с точки зрения проверки её правильности. Тогда проверка работоспособности ПП с использованием любого представителя класса даст один и тот же результат. Для повышения качества тестирования классификацию

выполняют по нескольким *разным* основаниям (признакам). Эти признаки называют ***критериями полноты тестирования.*** Обычно критерии формулируют в терминах «покрытие чего-либо», например, «покрытие классов входных данных», «покрытие функций». Например, критерий «покрытие функций» означает, что каждая функция проверяется минимум на одном тесте.