Error - mistake made in Coding or Mistake made by a Programmer

Defect - difference between expected and actual in the testing course/ phase

Bug -- accepted the above said defect by developer then called as bug

Fault - an incorrect step, process or data definition in a computer program which causes the program fail to perform action.  
Failure - is an event caused by an error

A **core dump** is a file of a computer’s documented memory of when a program or computer crashed.

Programmers often use a core dump to examine the problem with the use of a debugger. A core dump can include all the system memory or a part of the program that failed. There are several reasons why a computer or program can crash:

* Corrupted data
* A severe user error
* Virus-infected files
* Problems accessing data files
* An outdated operating system
* A segmentation fault or bus error
* A poorly ventilated or dusty computer tower
* A system-detected fault in the software or hardware
* Computer overheating caused by a faulty heat sink or fan

**Цели тестирования:**

* Обнаружение дефектов
* Повышение уверенности в уровне качества
* Предоставление информации для принятия решений
* Предотвращение дефектов

**1.1.5 Когда заканчивать тестирование? (K2)**

Для принятия решения о достаточном объеме тестирования, необходимо

принимать во внимание уровень рисков, включая технические риски, риски

безопасности и бизнес риски, а так же проектные ограничения, такие как время и

бюджет.

**5.2.3 Критерий входа (K2)**

Критерий входа определяет, когда нужно начинать тестирование, например, для

начала уровня тестирования или когда набор тестов готов для исполнения.

Обычно критерии входа могут покрывать:

• Готовность и доступность тестового окружения;

• Готовность средства тестирования в окружении;

• Доступность тестируемого кода;

• Доступность тестовых данных.

**5.2.4 Критерий выхода (K2)**

Критерий выхода определяет, когда нужно прекращать тестирование, например,

по окончании уровня тестирования или когда набор тестов достиг определенной

цели.

Обычно критерии выхода могут покрывать:

• Тщательность оценки, например покрытие кода, функциональности или

рисков;

• Оценку плотности дефектов или измерение надежности;

• Стоимость.

• Остаточные риски, такие как неисправленные дефекты или недостаток

тестового покрытия какой-либо области;

• План, основанный на времени выхода ПО на рынок.

Разные точки зрения в тестировании преследуют разные **цели**. Например, в

тестировании на этапе разработки (таком, как компонентное, интеграционное и

системное тестирование), основная цель может заключаться в том, чтобы вызвать

как можно больше отказов, чтобы дефекты в программном обеспечении были

идентифицированы и могли быть исправлены. В приемочном тестировании

основная цель может состоять в том, чтобы подтвердить, что система работает, как ожидалось и повысить уверенность в том, что она удовлетворяет

требованиям.

В некоторых случаях основная цель тестирования может состоять в

том, чтобы оценить качество программного обеспечения (без намерения

исправлять дефекты) и дать информацию заинтересованным лицам о рисках

выпуска системы в установленный срок.

**Принципы**

Эти принципы тестирования были предложены в последние 40 лет и являются

общим руководством для тестирования в целом.

**Принцип 1 – Тестирование демонстрирует наличие дефектов**

Тестирование может показать, что дефекты присутствуют, но не может доказать,

что их нет. Тестирование снижает вероятность наличия дефектов, находящихся в

программном обеспечении, но, даже если дефекты не были обнаружены, это не

доказывает его корректности.

**Принцип 2 – Исчерпывающее тестирование недостижимо**

Полное тестирование с использованием всех комбинаций вводов и предусловий

физически невыполнимо, за исключением тривиальных случаев. Вместо

исчерпывающего тестирования должны использоваться анализ рисков и

расстановка приоритетов, чтобы более точно сфокусировать усилия по

тестированию.

**Принцип 3 – Раннее тестирование**

Чтобы найти дефекты как можно раньше, активности по тестированию должны

быть начаты как можно раньше в жизненном цикле разработки программного

обеспечения или системы, и должны быть сфокусированы на определенных

целях.

**Принцип 4 – Скопление дефектов**

Усилия тестирования должны быть сосредоточены пропорционально ожидаемой,

а позже реальной плотности дефектов по модулям. Как правило, большая часть

дефектов, обнаруженных при тестировании или повлекших за собой основное

количество сбоев системы, содержится в небольшом количестве модулей.

**Принцип 5 – Парадокс пестицида**

Если одни и те же тесты будут прогоняться много раз, в конечном счете этот

набор тестовых сценариев больше не будет находить новых дефектов. Чтобы

преодолеть этот “парадокс пестицида”, тестовые сценарии должны регулярно

рецензироваться и корректироваться, новые тесты должны быть

разносторонними, чтобы охватить все компоненты программного обеспечения,

или системы, и найти как можно больше дефектов.

**Принцип 6 – Тестирование зависит от контекста**

Тестирование выполняется по-разному в зависимости от контекста. Например,

программное обеспечение, в котором критически важна

**Принцип 7 – Заблуждение об отсутствии ошибок.**

Обнаружение и исправление дефектов не помогут, если созданная система не

подходит пользователю и не удовлетворяет его ожиданиям и потребностям.

**Компонентное тестирование** (также известное как модульное) занимается поиском

дефектов и верификацией функционирования программных модулей, программ,

объектов, классов и т.п., которые можно протестировать изолированно.

В процессе могут быть использованы заглушки, драйвера и эмуляторы.

Обычно, компонентное тестирование производится с доступом к тестируемому

коду и с поддержкой рабочего окружения, такого как фреймворк модульного

тестирования или утилиты отладки. На практике компонентное тестирование

обычно производится разработчиками, которые пишут код. Дефекты обычно

исправляются сразу после того, как становятся известны, без занесения их в базу

дефектов.

**Интеграционное тестирование** проверяет интерфейсы между компонентами,

взаимодействие различных частей системы, таких как операционная системы,

файловая система, аппаратное обеспечение, и интерфейсы между системами.

**Системное тестирование** сконцентрировано на поведении тестового объекта как

целостной системы или продукта. Область тестирования должна быть четко

определена в главном плане тестирования либо плане тестирования для

конкретного уровня тестирования.

**Приемочным тестированием** системы чаще всего занимаются заказчики или

пользователи системы, а также другие заинтересованные лица.

Основная цель приемочного тестирования – проверка работоспособности

системы, частей системы или отдельных нефункциональных характеристик

системы. Поиск дефектов не является главной целью приемочного тестирования.

Приемочное тестирование оценивает готовность системы к развертыванию и

Использованию.

Приемочное тестирование может проводиться в различные моменты ЖЦ

разработки, например:

• Для коробочного продукта приемочное тестирование можно провести при

установке или интеграции

• Приемочное тестирование удобства использования компонента можно

провести во время компонентного тестирования

• Приемочное тестирование новой функциональности можно проводить до

системного тестирования

Типичные виды приемочного тестирования:

**Пользовательское приемочное тестирование(User acceptance testing**

**)**

Обычно проверяет готовность системы для использования в бизнесе.

**Эксплуатационное (приемочное) тестирование (Operational (acceptance) testing)**

Приемочное тестирование, проводимое системными администраторами,

включает:

• Тестирование резервного копирования \ восстановления

• Восстановление после сбоев

• Управление пользователями

• Задачи сопровождения

• Задачи загрузки и миграции данных

• Периодическая проверка уязвимостей системы

**Контрактное и правовое приемочное тестирование**

Контрактное приемочное тестирование выполняется для проверки требований,

предъявляемых контрактом в к разрабатываемому ПО. Критерий приема должен

быть определен непосредственно в контракте. Приемочное тестирование на

соответствие стандартам выполняется для проверки соответствия стандартам

государственным, юридическим или стандартам безопасности.

**Альфа и бета тестирование (или тестирование в условиях эксплуатации)**

Разработчики рыночного, или коробочного, ПО часто хотят получить отзывы от

потенциальных или существующих заказчиков до того, как начнется продажа

продукта. Альфа тестирование выполняется организацией, разрабатывающей

продукт, но не группой разработчиков. Бета тестирование, или тестирование в

условиях эксплуатации, выполняется покупателями или потенциальными

заказчиками на их собственных мощностях.

**Функциональное тестирование -** «что» эта система делает

**Нефункциональное тестирование) -** тестирование того, «как» система работает.

Нефункциональное тестирование может выполняться на всех уровнях

тестирования.

(Нефункциональное тестирование рассматривает внешнее поведение

программного обеспечения и в большинстве случаев использует разработку

тестов методом черного ящика.)

**2.3.3 Тестирование структуры/архитектуры программного**

**обеспечения (Структурное тестирование) (K2)**

Структурное тестирование (тестирование методом белого ящика) может

выполняться на всех уровнях тестирования. Структурные методы тестирования

лучше всего использовать после методов разработки тестов на основе

спецификации, чтобы измерить тщательность тестирования, используя измерения

покрытия структуры программы.

Покрытие – это часть структуры программы, которая была охвачена

тестированием, выраженная в процентах. Если покрытие не равно 100%, то

необходимо разрабатывать дополнительные тесты для покрытия пропущенных

участков программы.

**Регрессионное тестирование** – это повторное тестирование уже

протестированных программ после внесения в них изменений, чтобы обнаружить

дефекты, внесенные или пропущенные в результате этих действий. Эти дефекты

могут быть как в проверяемом компоненте, так и в связанном или несвязанным с

ним. Регрессионное тестирование выполняется, когда в программное

обеспечение или его окружение вносятся изменения. Глубина регрессионного

тестирования оценивается риском пропуска дефектов в программном

обеспечении, которое работало ранее.

Тесты должны быть повторяемыми, если они должны использоваться для

подтверждающего или регрессионного тестирования.

Регрессионное тестирование может выполняться на всех уровнях тестирования и

включает функциональное, нефункциональное и структурное тестирование.

Регрессионные наборы тестов запускаются множество раз и меняются медленно,

поэтому регрессионное тестирование является хорошим кандидатом на

автоматизацию.

Преимущества **статического анализа**:

• Раннее обнаружение дефектов до исполнения тестов

• Раннее предупреждение о подозрительных аспектах в коде или дизайне с

помощью вычисления метрик, таких как коэффициент сложности

• Определение дефектов, которые сложно обнаружить с помощью

динамического тестирования

• Определение зависимостей и нарушений целостности в моделях ПО,

например ссылок

• Улучшение пригодности к сопровождению кода и дизайна

• Предотвращение дефектов путем усвоения уроков, полученных во время

Разработки

Типичные дефекты, которые могут быть найдены при статическом анализе:

• Обращение к переменной, которой не присвоено значение

• Несоответствие интерфейсов между модулями и компонентами

• Переменные, которые не используются или некорректно объявлены

• Невыполняемые ветки кода

• Пропущенная или неверная логика (например, бесконечные циклы)

• Излишне сложные конструкции

• Отклонение от стандартов программирования

• Уязвимость в безопасности

• Нарушение синтаксиса в коде или моделях ПО

Расписание выполнения тестов должно учитывать такие факторы, как

регрессионные тесты, приоритеты и технические и логические зависимости.

**методы создания тестов на основе**

Общие признаки подходов, основанных на спецификациях:

• Для описания задач, которые должны быть решены, программных

продуктов или их компонентов, используются модели - формальные или

неформальные.

• Из этих моделей систематически выводятся тестовые сценарии.

Общие признаки подходов, основанных на структуре:

• тестовые сценарии выводятся на основе информации о том, как

спроектировано программное обеспечение (например, на основе

программного кода и подробного описания проектного решения).

• для программного обеспечения может быть измерена величина покрытия

для имеющихся тестовых сценариев, и последующие тестовые сценарии

могут разрабатываться для систематического увеличения покрытия.

Общие признаки методов на основе опыта:

• для определения тестовых сценариев используются человеческие знания и

опыт.

• знания тестировщиков, разработчиков, пользователей и заинтересованных

лиц о программном продукте, его использовании и окружении, являются

одним из источников информации.

• знания о вероятных дефектах и их распределении являются другим

источником информации

**Методы, основанные на спецификациях**

**4.3.1 Эквивалентное разбиение (К3)**

Входные данные для программного обеспечения или системы разбиваются на

группы, от которых ожидается сходное поведение, то есть они должны

обрабатываться аналогичным образом. Эквивалентные области (или классы)

могут быть определены как для валидных, так и для невалидных данных, то есть

тех значений, которые должны отвергаться.

**4.3.2 Анализ граничных значений (К3)**

Поведение на границах эквивалентных областей имеет наибольшие шансы быть

некорректным, таким образом границы являются потенциальным источником

дефектов. Минимальные и максимальные значения сегмента являются

граничными значениями.

**4.3.3 Тестирование таблицы решений (К3)**

Таблицы решений – хороший метод для сбора системных требований,

содержащих логические условия и документирования внутреннего дизайна

системы. Они могут использоваться для записи сложных бизнес-правил, которые

должна реализовывать система. Анализируются спецификации и определяются

условия и действия системы. Входные условия и действия чаще всего

формулируются таким образом, чтобы они могли принимать логические значения

«истина» или «ложь».

Сильной стороной тестирования таблицы решений является то, что она создает

комбинации условий, которые могли бы быть не проверены в ходе тестирования

иным способом. Этот метод может быть применен ко всем ситуациям, в которых

действие программного продукта зависит от нескольких логических альтернатив.

**4.3.4 Тестирование таблицы переходов (К3)**

Система может показывать различные отклики в зависимости от текущих условий

или предшествовавшей истории состояний.

Тесты могут разрабатываться для покрытия типовой последовательности

состояний, для покрытия каждого состояния, для выполнения каждого перехода,

для выполнения определенных последовательностей переходов или же для

тестирования некорректных переходов.

**4.3.5 Тестирование по сценариям использования (К2)**

Тесты могут базироваться на сценариях использования. Сценарий использования

описывает взаимодействия между участниками (включая пользователей и

систему) приводящие к полезным результатам для заказчика или пользователя

системы.

**4.4 Тестирование на основе структуры, или**

**методы белого ящика (К3)**

Тестирование на основе структуры, или тестирование методом белого ящика,

основывается на конкретной структуре программного продукта или системы, как

рассмотрено в следующих примерах:

• Компонентный уровень: структура компонента программного обеспечения,

т.е. операторы, альтернативы, ветви или определенные пути

• Интеграционный уровень: структура может быть представлена деревом

вызовов (диаграмма, в которой модули вызывают другие модули)

• Системный уровень: структура может представлять собой структуру меню,

бизнес-процессов или же схему веб-страницы.

Три метода проектирования тестов на основе структуры для покрытия кода: основанные на операторах, ветвях и альтернативах.

Покрытие альтернатив более строгое, чем покрытие операторов: 100% покрытие

альтернатив обеспечивает 100% покрытие операторов, но не наоборот.

**4.5 Методы, основанные на опыте (К2)**

Исследовательское тестирование - это параллельная разработка тестов, их

выполнение, протоколирование тестирования и изучение, основанные на

концепции тестирования, включающей в себя цели тестирования, и проводимые в

определенных временных рамках.

Данный подход наиболее полезен при наличии

неполных или неактуальных спецификаций и жестких временных ограничений,

или при усилении или дополнении более формального тестирования.

Типичные задачи руководителя тестирования могут включать:

• Координирование стратегий тестирования и планов с менеджерами

проектов и другими людьми;

• Составление и анализ стратегии тестирования для конкретного проекта и

тестовой политики для организации;

• Согласование перспектив тестирования с другими проектными процессами,

такими как интеграционное планирование;

• Планирование тестов согласно содержанию и пониманию рисков, включая

выбор методов тестирования, оценка времени, трудозатрат, стоимости

тестирования и наличия ресурсов, определение уровней и циклов

тестирования, а также планирование управления инцидентами;

• Инициирование спецификаций тестирования, подготовка, создание и

исполнение тестов, отслеживание и контроль результатов тестирования,

проверка критерия выхода;

• Планирование адаптации тестирования согласно результатам и прогрессу

тестирования (иногда задокументировано в отчетах), а так же принятие

решений и выполнение действий по решению проблем;

• Установка адекватного управления конфигурацией для обеспечения

трассируемости тестирования;

• Принятие решение о том, что, как и до какого уровня должно быть

автоматизировано;

• Выбор средства поддержки тестирования и организация обучения

тестировщиков по его использованию;

**Мониторинг прогресса тестирования**

Обычные тестовые метрики включают в себя:

• Процент проделанной работы по подготовке тестовых сценариев (или

процентное соотношение запланированных и подготовленных сценариев);

• Процент проделанной работы по подготовке тестового окружения;

• Выполнение тестовых сценариев (например, количество

выполненных\невыполненных тестовых сценариев, успешно

пройденных\неудачных тестовых сценариев);

• Информация о дефектах (например, плотность дефектов, количество

найденных и исправленных дефектов, интенсивность отказов и результаты

повторного тестирования);

• Тестовое покрытие требований, рисков или кода;

• Субъективная уверенность тестировщиков в продукте;

• Даты контрольных точек тестирования;

• Стоимость тестирования, включая стоимость по сравнению с выгодой

нахождения следующего дефекта или запуска следующего теста.

**Отчеты о тестировании** предоставляют итоговую информацию о тестировании,

включая:

• Что произошло во время цикла тестирования, например, даты, когда были

достигнуты критерии выхода;

• Проанализированную информацию и метрики для поддержки рекомендаций

и решений о последующих действиях, таких как оценка оставшихся дефектов, кономическое обоснование продолжения тестирования, оставшиеся риски и уровень уверенности в тестируемом ПО.

**Примеры действий по контролю тестирования:**

• Принятие решений на основании данных мониторинга тестирования;

• Повторная расстановка приоритетов при возникновении установленного

риска (например, задержка выпуска ПО);

• Изменение графика тестирования согласно доступности тестового

окружения;

**Риски проекта** – это риски, которые влияют на способность проекта достигнуть его

целей, и включают:

1. Организационные факторы:

* Недостаток квалификации, подготовки и сотрудников;
* Личные проблемы сотрудников;
* Политические проблемы, такие как:
* Тестировщики в недостаточной степени сообщают о своих проблемах и результатах тестирования;
* Неспособность следовать информации, полученной во времятестирования или рецензирования (например, не улучшать практики разработки или тестирования);
* Неверное отношения к тестированию или ложные ожидания (например, не принимать во внимание значение найденных во время тестирования дефектов);

2) Технические проблемы:

* Проблемы в определении верных требований;
* Объем, при котором требования не могут соответствовать заданным
* ограничениям;
* Вовремя не готово тестовое окружение;
* Позднее преобразование данных, планирование миграции и разработки тестовых данных и средств преобразования\миграции тестовых данных;
* Низкое качество проектирования, кода, конфигурационных и тестовых данных и тестов;

1. Проблема поставщика:

* Отказ третьей стороны;
* Проблемы контракта.

**Риски продукта** – это потенциальные области сбоя (неблагоприятные будущие

события или опасность) в ПО или системе, т.к. они подвергают риску качество

продукта, например:

* Поставка потенциально ненадежного ПО;
* Возможность того, что программное\аппаратное обеспечение может

нанести вред человеку или компании;

* Плохие характеристики ПО (например, функциональность, надежность,

удобство использования или производительность);

* Неполнота и низкое качество данных (например, проблемы миграции,

преобразования и перемещения данных, отклонение от стандарта данных);

* ПО, которое не выполняет предполагаемых функций.