НАДЁЖНОСТЬ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Лекция 12:
Проведение тестирования.
Анализ данных об отказах для принятия решений

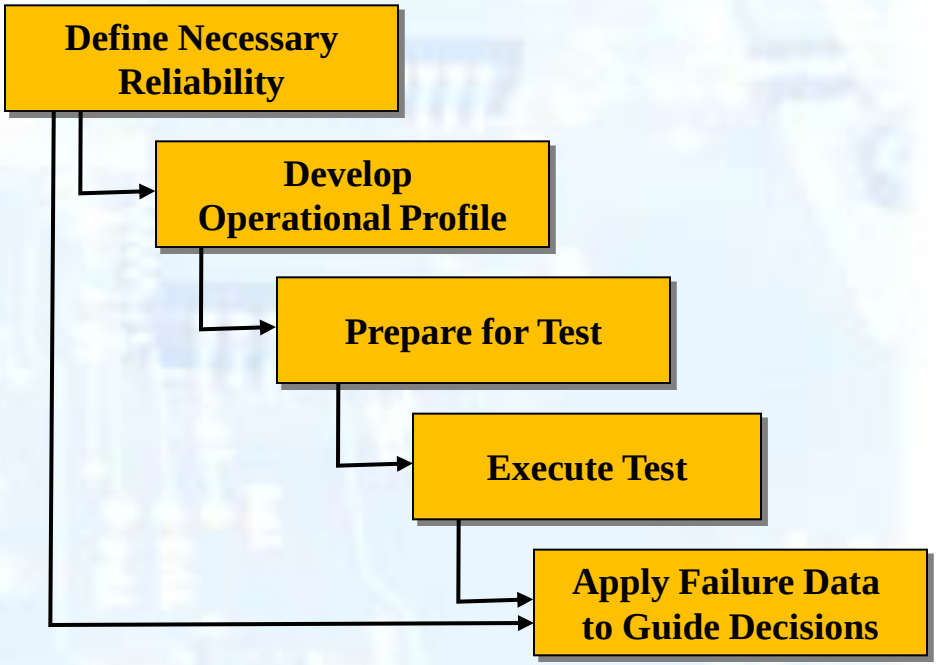
ВМиК МГУ им. М.В. Ломоносова,
Кафедра АСВК, Лаборатория Вычислительных Комплексов
Ассистент Волканов Д.Ю.

План лекции

- Распределение ресурсов при тестировании
- Методы тестирования
- Диаграмма надёжности
- Модель RDC
- Критерии выпуска ПО

SRE: процесс

- 5 шагов в SRE процессе:
  - Определить необходимый уровень надежности
  - Разработать функциональный разрез
  - Подготовить для тестирования
  - Выполнить тестирование
  - Проанализировать данные об ошибках и использовать их для принятия решений



Как распределить время

1) Между тестируемыми системами
2) Между функциональным, регрессионным и нагрузочным тестированием для каждой из подсистем
3) Между режимами функционирования для нагрузочных тестов каждой из подсистем

1. Распределение время тестирования в масштабах системы

- Выделяйте время на тестирование связанных систем в соответствии с рассчитанным риском
- Распределяйте оставшееся время в соответствии с пропорциями, определёнными для тестовых сценариев

2. Распределение времени и рост надёжности

- При росте надёжности для каждой подсистемы и каждого нового тестового сценария выделяйте время на:
  - Функциональное тестирование [первая версия программы]
  - Регрессионное тестирование [последующие версии программы]
- При этом гарантируется проверка всех новых критических операций.
- Оставшееся время выделяется на нагрузочное тестирование.

3. Распределение времени нагрузочного тестирования

- Выделяйте время нагрузочного тестирования между режимами функционирования программы в соответствии с числом операций в каждом из них

- **Пример:**

Режим	Пропорция операций
Пиковая нагрузка	0.1
Инициализация	0.7
Часы простоя	0.2

Запуск тестовых сценариев (1)

- Последовательность тестирования:
  1. **Приобретённые компоненты**
     - Проверочные тесты
  2. **Разрабатываемая система**
     - Сначала функциональные, затем нагрузочные тесты для первой версии программы
     - Сначала функциональные, затем регрессионные тесты для последующих версий программы
  3. **Интерфейсы других систем**
     - Только нагрузочные тесты
- Допустимо изменение порядка или параллельное тестирование

Запуск тестовых сценариев (2)

- При тестах на увеличение надёжности сначала проводите функциональное, а затем нагрузочное тестирование. Проводите регрессионное тестирование после каждого значительного изменения программы
- Выполнение тестов проводится в случайное время
- При функциональном тестировании новые тесты и регрессионные тесты предыдущей версии программы выполняются в случайном порядке
- При нагрузочном тестировании вызывайте сценарии каждого режима функционирования в соответствии с выделенным для него временем
- Число нагрузочных тестов определяется временем, выделенным для данного режима функционирования

Выбор сценариев (1)

- Для нагрузочных тестов, сценарии могут выбираться повторно. Для функциональных и регрессионных тестов это не так
- При функциональном и регрессионном тестировании результаты прогонов скорее всего будут идентичными, так как влияние косвенных переменных сведено к минимуму
- При нагрузочном тестировании результаты нескольких прогонов скорее всего будут отличаться

Выбор сценариев (2)

- Зачем повторять одни и те же тесты?

- Одна операция может содержать в себе несколько ошибок. Если выбор производится без перемещения, то операция может быть вызвана только один раз, и вероятность обнаружения других ошибок снижается

Определение отказов системы

Чтобы определить факт отказа:

1) Найдите отклонения от нормы в выводе программы
2) Определите какие из них вызваны отказами
3) Определите время возникновения отказа
4) Определите класс тяжести отказа
5) Задокументируйте отказ

1. Анализ вывода программы

1. Некоторые отклонения от нормального поведения можно найти с помощью средств отладки (GDB,…)
   - Примеры:
     - Неверное обращение к памяти
     - Взаимная блокировка процессов
     - Зависание
     - etc.

2. Отклонения и ошибки

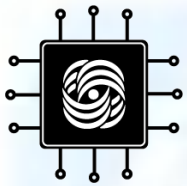
- С помощью ручного анализа отклонения можно сделать вывод, что нарушаются требования к программе, следовательно произошёл отказ
- Отказы наибольшей тяжести можно определить без анализа вывода программы:
  - Аварийное завершение программы
  - Неполное выполнение поставленной задачи

3. Время возникновения отказа

- Для определения времени отказа используйте ту же меру, что и при задании желаемой целевой интенсивности отказов
- Для тестов функциональности измеряйте время, когда произошёл отказ
- Для тестов надёжности системы используйте интервальную оценку.
  - [5 отказов за 1000 операций]

4. Определение класса тяжести отказа

- Выберете один из описанных ранее классов тяжести отказов, который бы соответствовал найденной ошибке
- Задокументируйте отказы и их классы тяжести
- Начинайте исправлять ошибки с отказов, имеющих больший класс тяжести

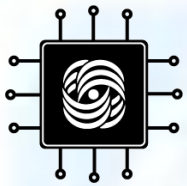


Терминология (1)

- **Статический анализ**
  - Ручная проверка кода
    - Взаимопроверка
    - Формальная проверка
  - Методы нахождения отклонений и ошибок
    - Аудит исходного кода
    - Проверка интерфейсов
    - Поблочное тестирование
    - Анализ потоков данных
  - Структурный анализ

Терминология (2)

- **Динамический анализ**
  - Инструментализация
    - *Структурный анализ*
    - *Измерение производительности: времени выполнения и объёма потребляемых ресурсов*
    - *Измерение производительности: анализ сложности алгоритма работы*
    - *Вставка динамических проверок (assert)*
    - *Интерактивное тестирование и отладка*
  - Случайное тестирование



Терминология (3)

- **Динамический анализ**
  - Функциональное тестирование
    - *На основе спецификации*
    - *Построение причинно-следственных связей*
  - Мутационное тестирование
    - *Мутационный анализ*
    - *Подсев ошибок*
  - Тестирование в реальном времени
  - Формальное тестирование

Простейшие технологии тестирования

- Два широко известных метода:
  - **Метод прозрачного ящика**
    - Обнаруживает ошибки на основе анализа внутренней структуры программы
  - **Метод чёрного ящика**
    - Оценивает насколько хорошо программа выполняет предъявляемые к ней требования

1. Метод прозрачного ящика

- Требует детального знания реализации программы
- Цель – обеспечить полное **покрытие кода**
  - Как много путей выполнения программы проверены на самом деле?
  - Эффективность теста часто измеряется размером покрытого им кода

2. Метод чёрного ящика

- Предполагает, что требования уже прошли валидацию
- Поиск нереализованной или неправильно реализованной функциональности
- Задаёт входные данные, заранее зная какой результат должен быть получен
- Существует много методов:
  - Тестирование производительности
  - Стресс-тестирование
  - Тестирование надёжности
  - Тестирование безопасности
  - etc.

Уровни тестирования

- Тестирование связано с циклом разработки программ:
  - Блочное
  - Интеграционное
  - Пользователями
  - Установки
  - Регрессионное

1. Блочное тестирование

- Тестирование одного модуля программы методом белого ящика в контролируемом окружении и независимо от других модулей
  - Модуль – функция или небольшой их набор, который может быть полностью покрыт тестовыми сценариями

2. Интеграционное тестирование

- Тестируется комбинация модулей
- Фокус на их взаимодействии
- Методы белого и чёрного ящика
- Существует три основных подхода:
  - Сверху-вниз
  - Снизу вверх
  - Большого взрыва

3. Внешнее тестирование

- Метод чёрного ящика
- Проверяет, что система корректно реализует требуемый функционал
- Иногда известно как *альфа* тестирование
- Тестирование имитирует использование системы конечными пользователями

4. Системное тестирование

- Более робастная версия внешнего тестирования
- Различие в тестовой платформе
- Окружение идентично реальному
  - Учитывается «железо», размеры БД, …
- Позволяет лучше оценить нефункциональные требования (производительность, безопасность,…)

5. Тестирование пользователями

- Также известно как бета-тестирование
- Система тестируется конечными пользователями
- Ещё более реалистичное тестирование по сравнению системным
- Валидация на основе пользовательских представлений
- Определяет степень готовности для развёртывания

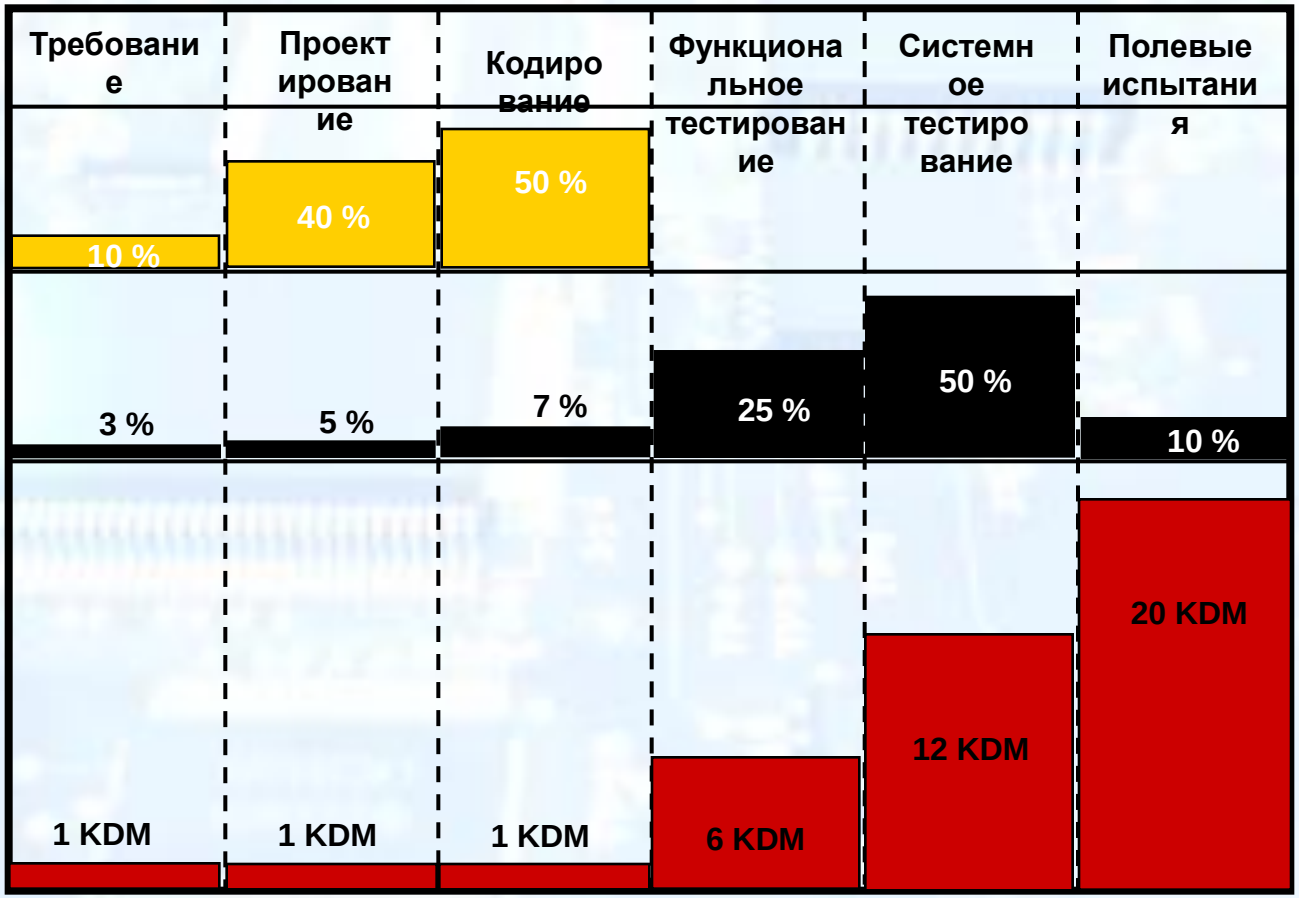
6. Тестирование установки

- Проверка полной, частичной установки/удаления программы и её обновления
- Расплывчато описывается в литературе

7. Регрессионное тестирование

- Тестирование модифицированной программы
- Проверяет, что внесённые изменения не повлияли на работу остальных подсистем
- Для тестирования выбираются сценарии, на работу которых могут повлиять внесённые правки
- Исправление ошибки может добавить новую ошибку, нарушить структуру или корректность интеграции с другими компонентами

Стоимость ошибки



1 KDM = 1,000 Немецких марок

Тестирование на основе ошибок

- Определяет классы ошибок и входные данные, способные вызвать эти ошибки, если они присутствуют
- ***Отправка ошибок -*** сознательное добавление ошибок в программу до её тестирования. На основе числа найденных в процессе тестирования ошибок определяется число реальных ошибок в коде
- ***Мутационное тестирование*** добавляет ошибки в код, чтобы определить оптимальные для тестов данные
- ***Добавление ошибок*** позволяет определить реакцию всей системы на некорректное поведение отдельных её частей

Нефункциональное тестирование

- Нефункциональные требования описывают отдельные функции системы, а скорее задают общие параметры системы (производительность, безопасность, совместимость, удобство использования, …)
- На тестирование нефункциональных требований могут быть затрачены большие усилия
- Задание нефункциональных требований позволяет перейти от приложения которое способно выполнить требуемые задачи к приложению, которое нужно реальным пользователям системы

Документирование нефункциональных требований

- Обычно нефункциональные требования задаются двумя способами одновременно:
  - Создаётся единый документ, в котором описываются требования применительно ко всем случаям использования системы
  - Каждое требование к системе содержит раздел «нефункциональные требования», где описываются требования, отличные от общих спецификаций

Тестирование нефункциональных требований (1)

- **Не откладывайте проверку нефункциональных требований**
- Обычно рекомендуют проверять нефункциональные требования одновременно с соответствующими им функциональными требованиями

Тестирование нефункциональных требований (1)

- **Проводите тестирование производительности в реальном окружении**
- Для определения производительности СУБД нужно провести тесты на БД размером в 1, 100, 500, 1,000, 5,000, 10,000, 50,000, и 100,000 записей
- Такой способ тестирования производительности позволяет определить верхнюю границу, после которой приложении использовать нецелесообразно

Тестирование нефункциональных требований (3)

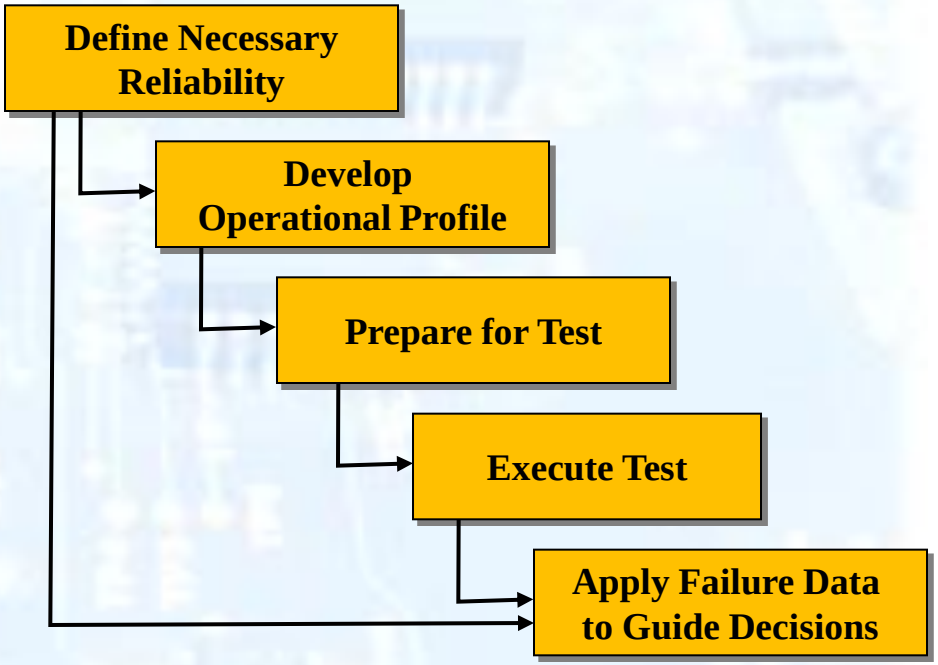
- **Доверяйте тестирование удобства работы с приложением его реальным пользователям**
- Используйте для получения информации сразу несколько методов:
  - Исследуйте существующие средства
  - Наблюдайте за работой пользователя
  - Предлагайте пользователям несколько прототипов интерфейса

Тестирование нефункциональных требований (3)

- **Исследуйте вопросы безопасности на как на уровне отдельного требования, так и на общесистемном уровне**
- Каждая функция системы, вероятно, несёт в себе определённые угрозы безопасности
- На основе грамотно составленных требований к безопасности можно построить набор сценариев для их проверки
- Если угрозы безопасности критичны, то для его тестирования стоит использовать сторонние приложения

SRE: процесс

- 5 шагов в SRE процессе:
  - Определить необходимый уровень надежности
  - Разработать функциональный разрез
  - Подготовить для тестирования
  - Выполнить тестирование
  - Проанализировать данные об ошибках и использовать их для принятия решений



Типы решений

- **Решения, относящиеся к сертификационным испытаниям**
  - Принять/отклонить приобретенный компонент
  - Принять/отклонить подсистему, операционную систему, аппаратное обеспечение и т.п.
- **Решения, относящиеся к испытаниям повышения надежности**
  - Управление процессом разработки ПО
  - Выпуск ПО

Обзор: FIO

- Интенсивность отказов цели (FIO) (λ_F)
  - Интенсивность отказов определяется как отказы в некоторых единицах, например:
    - 3 сигнала за 100 часов функционирования.
    - 5 отказов за 1000 заданий и т.д.
- Средняя наработка до отказа (MTTF)
  - Ожидаемое время до обнаружения следующего отказа.

$$A = \frac{1}{1 + \lambda\ t_m} = \frac{MTTF}{MTTF + t_m} \quad \text{therefore} \quad MTTF = \frac{1}{\lambda}$$

Диаграмма надежности /2

- Эффективный способ проверки выполняется FIO (λ_F) или нет
- Он основан на сборе данных об отказах в конкретные моменты времени
- Вертикальная ось: количество отказов(n)
- Горизонтальная ось: нормализованные данные об отказах (Tn), т.е. время отказа $\times$ λ_F

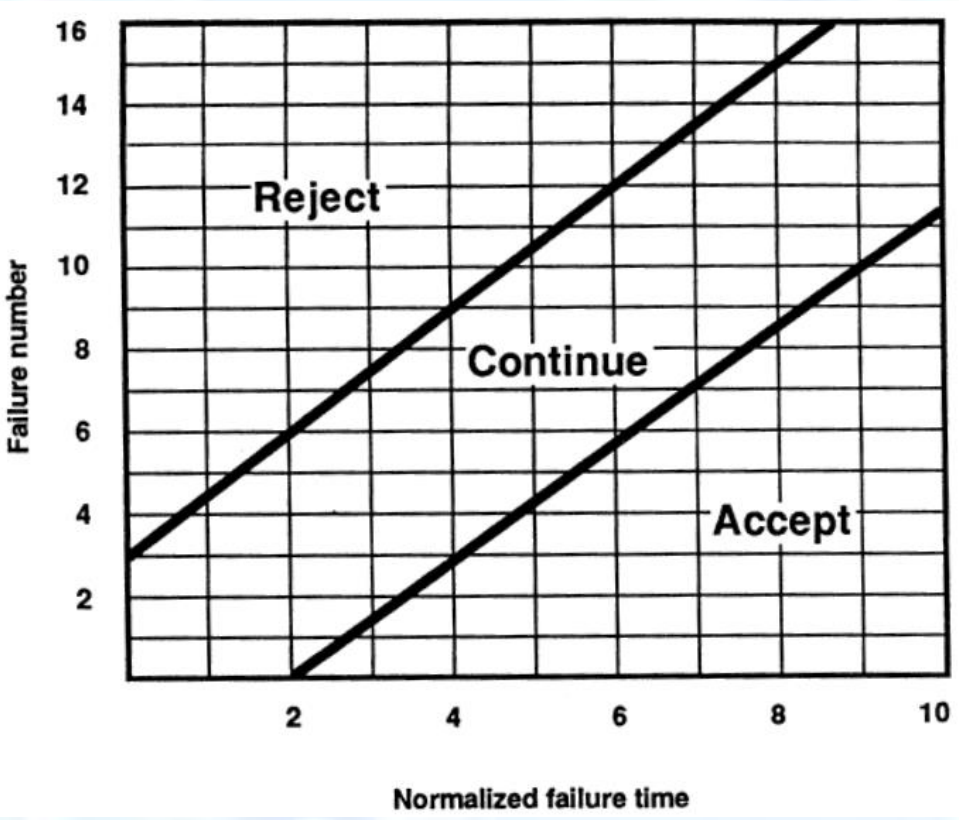


Figure from Musa's Book

Параметры /1

- **Отношение различимости (γ):** Приемлемая ошибка в оценке интенсивности отказов
- **Риск заказчика(β) :** Вероятность того, что разработчик готов принять ложное заявление, что целевая интенсивность отказов выполняется (т.е. принятие), когда это не так
- **Риск разработчика(α) :** Вероятность того, что разработчик готов принять ложно заявив, что целевая интенсивность отказов **не** выполняется (т.е. ***отклонение***), когда это так и есть

Параметры /2

Для α =10% и β= 10% и γ =2

- Существует риск в 10% (β) ошибочного принятия ПО, когда его целевая интенсивность отказов на самом деле равна или больше чем в 2 раза (γ =2) превышает целевую интенсивность отказов
- Существует риск в 10% (α) ошибочного отклонения ПО, когда его целевая интенсивность отказов на самом деле равна или менее чем 2 раза (γ =2) меньше целевой интенсивности

Диаграмма надежности /2

$$A = \ln \frac{\beta}{1-\alpha} \qquad B = \ln \frac{1-\beta}{\alpha}$$

- *A* изменяется быстро с *риском заказчика,* но очень незначительно с *риском разработчика* и это определяет отрезок границы **принятия** с горизонтальной линией *n=0*
- *B* изменяется быстро с *риском разработчика*, но очень незначительно с *риском заказчика* и это определяет отрезок границы ***отклонения*** с вертикальной линией $T_n=0$

Диаграмма надежности /3

- Граница между областями ***принятия*** and и ***продолжения***

$$T_n = \frac{A}{1-\gamma} - \frac{\ln \gamma}{1-\gamma} n$$

- Граница между областями ***отклонения*** и ***продолжения***

$$T_n = \frac{B}{1-\gamma} - \frac{\ln \gamma}{1-\gamma} n$$

Диаграмма надежности /4

Значения отрезков границ с различными горизонтальными и вертикальными осями

	Discrimination ratio γ		
Intercept with	2	1.5	1.1
Horizontal line at $n = 0$	$-A, -B$	$-2A, -2B$	$-10A, -10B$
Horizontal line at $n = 16$	$-A + 11.1, -B + 11.1$	$-2A + 13.0, -2B + 13.0$	$-10A + 15.2, -10B + 15.2$
Vertical line at $T_N = 0$	$1.44\,A, 1.44\,B$	$2.47\,A, 2.47\,B$	$10.5\,A, 10.5\,B$
Vertical line at $T_N = 16$	$\frac{A + 16}{0.693}, \frac{B + 16}{0.693}$	$\frac{A + 8}{0.405}, \frac{B + 8}{0.405}$	$\frac{A + 1.6}{0.0953}, \frac{B + 1.6}{0.0953}$

Диаграмма надежности /5

Значения A и B для различных уровней рисков заказчика и разработчика

Supplier risk	Parameter	Consumer risk 0.1	0.05	0.01	0.001
0.1	*A*	−2.20	−2.89	−4.50	−6.80
	B	2.20	2.25	2.29	2.30
0.05	*A*	−2.25	−2.94	−4.55	−6.86
	B	2.89	2.94	2.99	2.99
0.01	*A*	−2.29	−2.99	−4.60	−6.90
	B	4.50	4.55	4.60	4.60
0.001	*A*	−2.30	−2.99	−4.60	−6.91
	B	6.80	6.86	6.90	6.91

Table from Musa's Book

Диаграмма надежности /6

- Когда уровни риска (α и β) уменьшаются, система потребует проведения больше испытаний прежде чем достигнуть областей принятия или отклонения, *т.е.* необходимо продолжить расширение областей
- Когда отношение различимости (γ) уменьшается, система потребует проведения больше испытаний прежде чем достигнуть областей принятия или отклонения, *т.е.* необходимо продолжить расширение областей

RDC:Когда и как

- Когда использовать RDC?
  - Когда данные об отказах ограничены несколькими отказами и требуется определить тенденцию в надежности системы
- Как использовать RDC?
  - Собрать данные об отказах (количество отказов и моменты времени)
  - Определить MTTF и ожидаемые уровни доверия
  - Нарисовать точки отказов на графике и проанализировать тенденцию надежности

RDC: Пример/1

- Риск потребителя
  $\beta = 5\%$
- Риск поставщика
  $\alpha = 5\%$
- Отношение различимости
  $\gamma = 2$

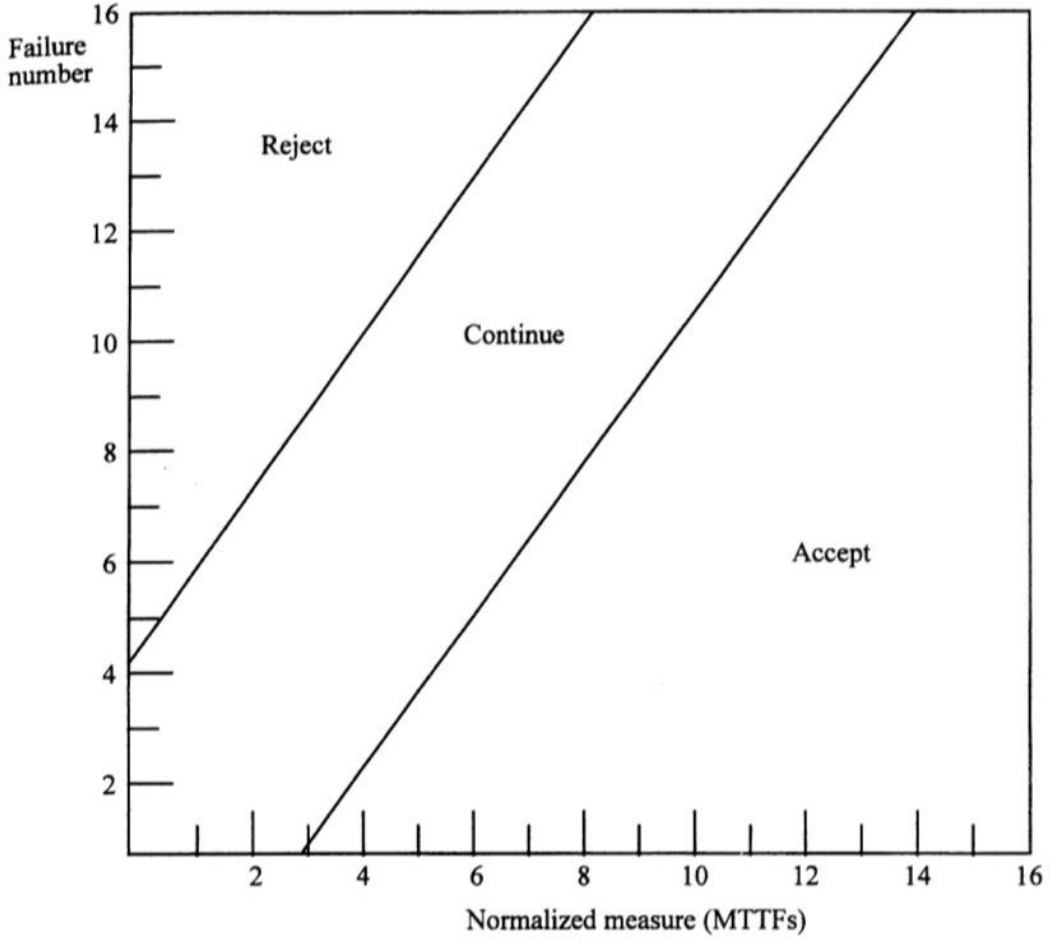


Figure from Musa's Book

RDC: Пример /2

- Риск потребителя
  β = 1%
- Риск поставщика
  α = 1%
- Отношение различимости
  γ = 2

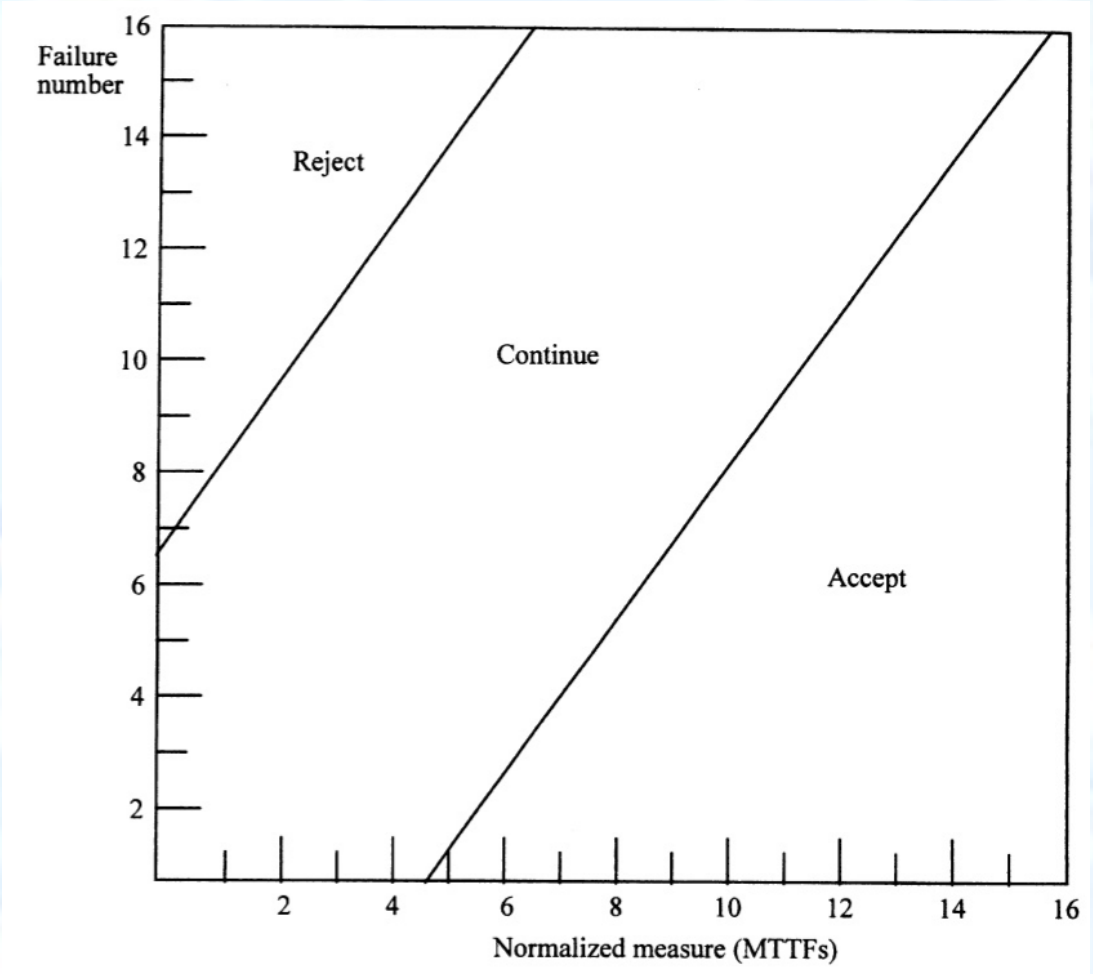


Figure from Musa's Book

RDC: Пример /3

- Риск потребителя
  β = 0,1%
- Риск поставщика
  α = 0,1%
- Отношение различимости
  γ = 2

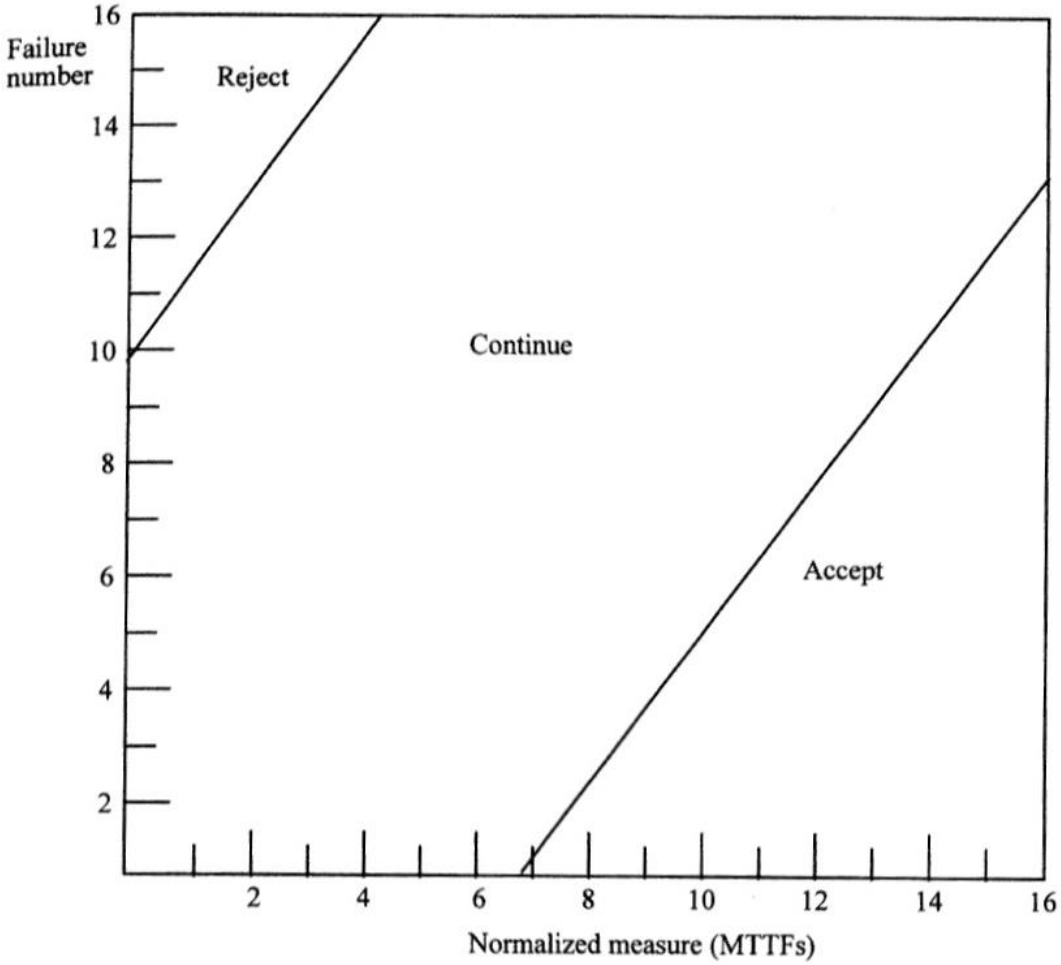


Figure from Musa's Book

RDC: Пример /4

- Риск потребителя
  β = 10%
- Риск поставщика
  α = 10%
- Отношение различимости
  γ = 1.2

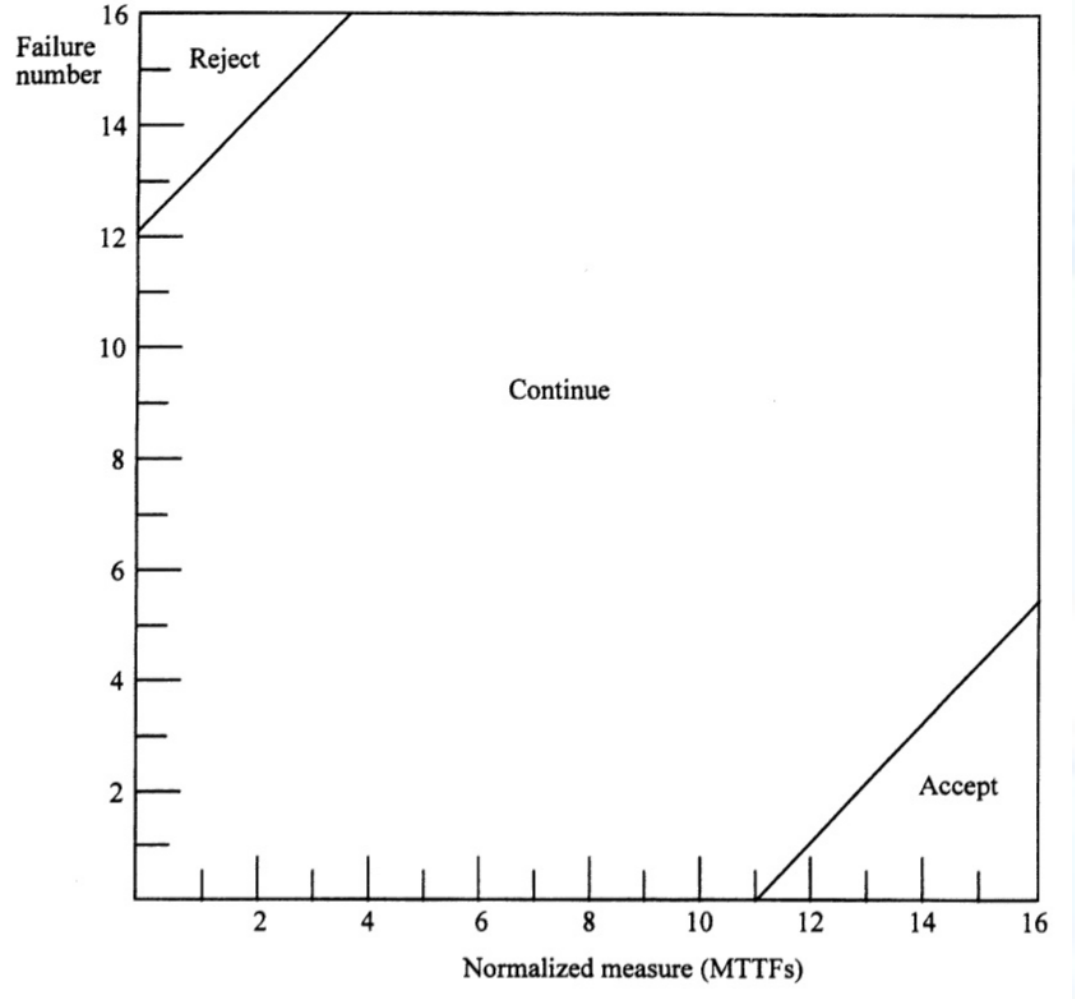


Figure from Musa's Book

Пример 1

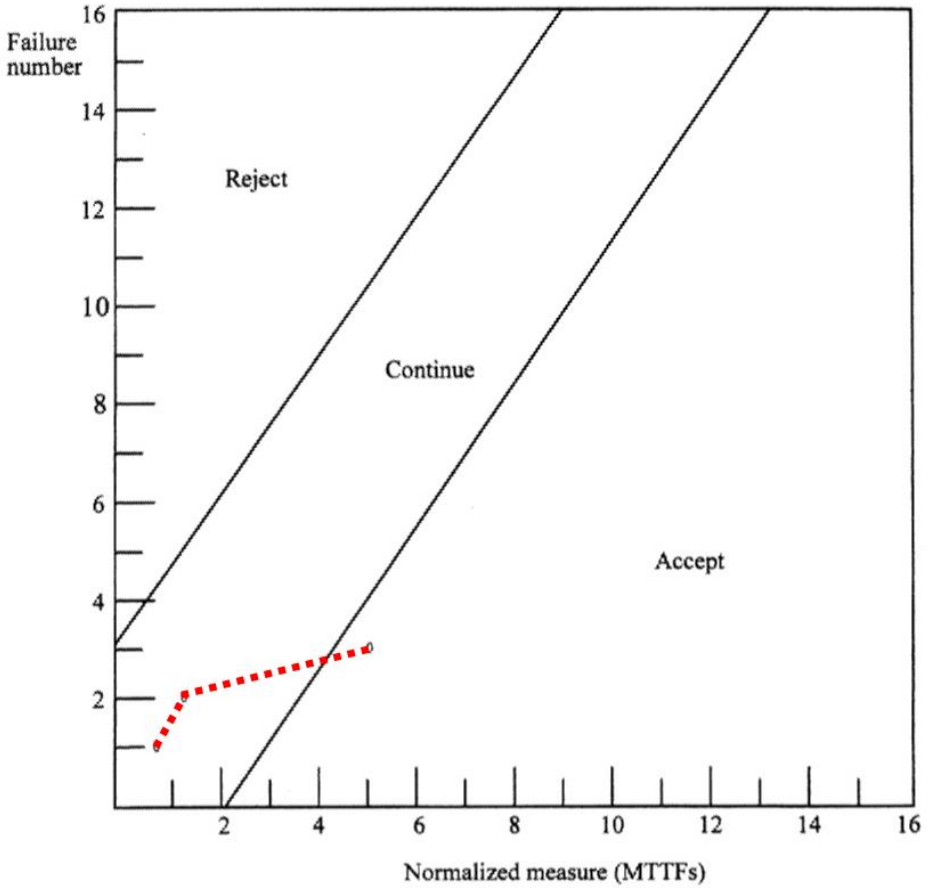
Failure number	*Measure (million transactions)*	*Normalized Measure (MTTF)*
1	0.1875	0.75
2	0.3125	1.25
3	1.25	5

$\lambda_F = 4$ failures/million transactions

$\alpha = \%10$

$\beta = \%10$

$\gamma = 2$



Пример 2

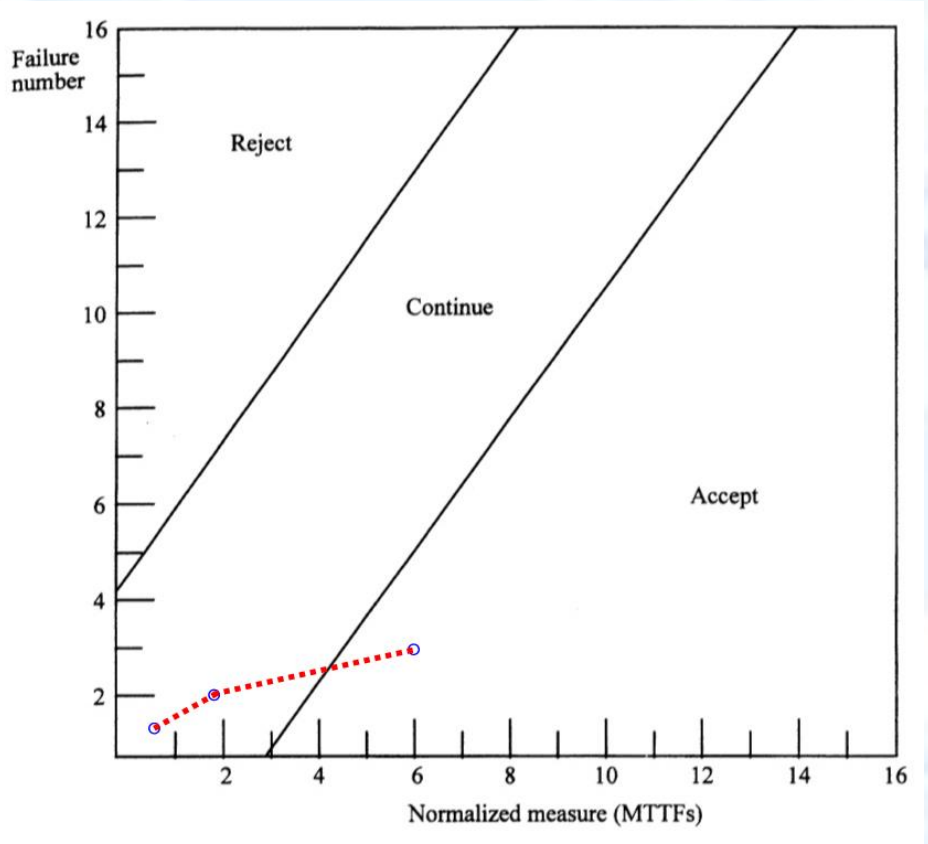
Failure number	*Measure (CPU hour)*	*Normalized Measure (MTTF)*
1	8	0.8
2	19	1.9
3	60	6

$\lambda_F = 0.1$ failures/CPU hour

$\alpha = 0.05$

$\beta = 0.05$

$\gamma = 2$



Пример 3 /1

- Мы собрались купить новый цветной лазерный принтер для лаборатории. Мы имеем принтер для проведения тестовых сертификационных испытаний на нем. Инструкция показывает, что необходимо заменить тонер каждые 10000 страниц. Мы хотим, чтобы система работала безотказно между двумя последовательными изменениями тонера или в худшем случае – один отказ

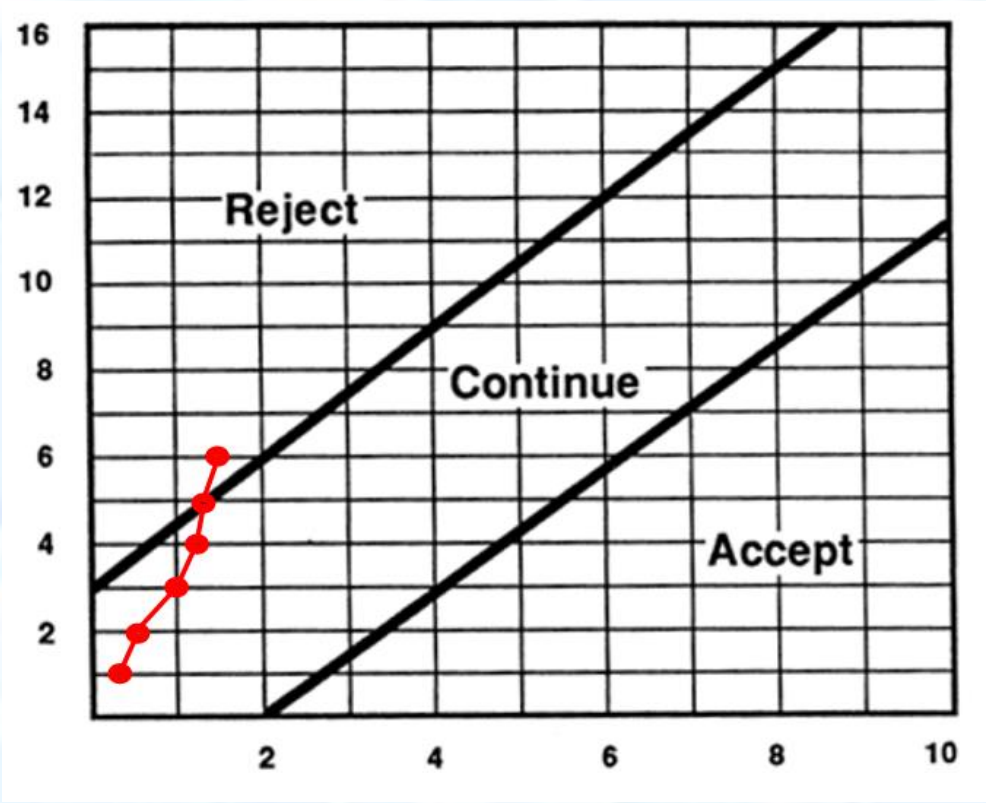
а) Какая должна быть наша целевая интенсивность отказов для системы?

$$\lambda_F = 1/10000 \text{ pages}$$

Пример 3 /2

b) Мы наблюдаем, что отказы происходят на 4000 страниц, 6000 страниц, 10000 страниц, 11000 страниц, 12000 страниц и 15000 страниц. Используя демонстрационные диаграммы надежности, какое заключение можно сделать об этом принтере?

Пример 3 /3



- Из-за провала сертификационных испытаний мы отказываемся от этого принтера

RDC: Достоинства и ограничения

- RDC анализ это очень эффективный с точки зрения времени и стоимости способ анализа надежности системы. Он может использоваться для демонстрации надежности системы при принятии решений
- Недостатком RDC анализа является то, что RDC не позволяет дать количественную оценку надежности (или доступность) изучаемой системы. Он может лишь показать тенденции изменений и их влияние на надежность системы
- Другой недостаток: экспериментировать с различными уровнями достоверности и MTTF возможно, но довольно утомительно

Типы решений

- **Решения, относящиеся к сертификационным испытаниям**
  - Принять/отклонить приобретенный компонент
  - Принять/отклонить подсистему, операционную систему, аппаратное обеспечение и т.п.
- **Решения, относящиеся к испытаниям повышения надежности**
  - Управление процессом разработки ПО
  - Выпуск ПО

Критерий выпуска

- Следующие показатели должны рассматриваться совместно для получения адекватной картины качества продукта:
  - Стабильность, надежность и доступность системы
  - Дефект объёма
  - Выдающиеся критические проблемы
  - Обратная связь с пользователями ранних версий программ
  - Другие атрибуты качества, которые имеют особое значение для конкретного продукта, заказчика и рынка(например, простота использования, производительность, безопасность, мобильность)

From Dr. Kan's Book

Подход:модели

- Использование средств таких как CASRE для построения соотношения $\lambda / \lambda F$ во времени, чтобы проанализировать тенденции
- CASRE использует **базовую экспоненциальную** и **логарифмическую пуассоновскую** и некоторые другие модели
- Базовая экспоненциальная модель предполагает конечное количество отказов за бесконечное время; логарифмическая пуассоновская модель предполагают бесконечное количество отказов
- При оценке соотношения $\lambda / \lambda F$, базовая экспоненциальная модель имеет «оптимистическую» тенденцию; логарифмическая пуассоновская модель имеет «пессимистическую» тенденцию

Критерий выпуска /1

- Оценка нормализованной интенсивности отказов имеет степень неопределенности. Она определяется «доверительным интервалом»
- Например, если доверительный интервал 90%, то можно ожидать, что система достигнет своей целевой интенсивности только с вероятностью 90%
- CASRE 2.0 не позволяет вычислять «доверительные интервалы

Критерий выпуска /2

- Как включить доверительный интервал в оценку?
- Можно использовать следующий приближенный подход:
  - Рассмотрим завершение тестирования для системы, когда нормированная интенсивность отказов снижается до 0,5 или ниже
  - Предположим, что 90% - доверительный интервал для интенсивности отказов вблизи релиза и таким образом интенсивность отказов расширяется от 0.5 до 1.5 раз

Критерий выпуска /3

Рассмотрим вопрос о выпуске продукта

1. Тестирование (функций и нагрузки) прекращено удовлетворительно для продукта
2. Тестирование прекращено удовлетворительно для всех вариантов продукта или нормированные значения отношения $\lambda/\lambda F$ для этих вариантов не превышают 0.5
3. Продукт и его варианты проходит все приемочные испытания, запланированные для них
4. Связанные системы проходят все приемо-сдаточные испытания

Развитие программ /1

- Модели надежности, используемые для оценки интенсивности отказов (λ), предполагают, что программа работает стабильно
- Программа может изменяться из-за:
  - Изменения требований
  - Интеграции новых частей
  - Необходимости адаптироваться к изменяющейся аппаратной и программной среде
  - Необходимости улучшения производительности системы
  - Эволюция как часть процесса разработки

Развитие программ /2

Вероятные сценарии:

- Игнорирование изменений при медленном развитии программы или незначительном по размеру
- Игнорирование старых данных на старой фазе и базы оценок на новой фазе
- Поэтапное развитие
  1. Применение изменений по компонентам
  2. Применение изменений к функциональным группам

а) Игнорирование изменений

- Эффекты развития могут быть проигнорированы для программ развивающихся медленно и незначительно по размеру (например, менее чем 5% от общего количества кода за неделю)
- **Достоинства:** Не требуется никаких дополнительных испытаний и сбора данных
- **Недостатки:** Оценки параметров модели и λ/λF отношения будут отставать от истинного положения и иметь ошибки, но диапазон ошибки может быть приемлемым

b) Игнорирование старых данных

- Установить диапазон изменения для данных об отказах для того, чтобы включить только последние отказы
- Окно последних отказов должно обычно включать последние 40-50 отказов для приемлемой оценки параметров
- Применять только, когда изменения программы ограничены(например, менее 20% от изменений программы)

c) Поэтапная эволюция /1

- Подходы поэтапного развития как правило работают лучше *с малым количеством больших изменений*, каждое из которых – результат добавления независимых элементов (подсистем, пакетов и т.п.)
- ***В компонентном подходе***, добавляются отдельно интенсивности отказов для отдельных компонентов, чтобы получить интенсивность отказов системы на каждом этапе
- ***В подходе с функциональными группами***, добавляются взвешенные интенсивности отказов функциональных групп, чтобы получить интенсивность отказов системы на каждом этапе

c) Поэтапная эволюция /2

- Достоинства и недостатки подхода с поэтапной эволюцией:
- **Достоинства:** Функциональный профиль системы может применяться непосредственно
- **Недостатки:**
  - Требуется дополнительный сбор данных с множества элементов
  - Большая ошибка оценки (или с более поздним достижением заданной степени точности) из-за меньших размеров выборки

Спасибо за внимание!