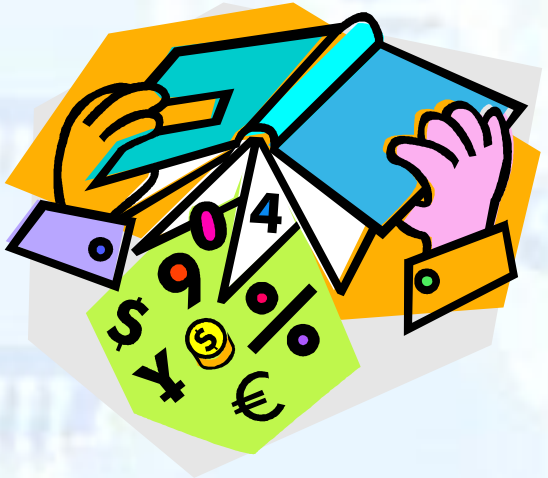
НАДЁЖНОСТЬ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Лекция 9:
Модели надёжности ПО
Оптимизация надёжности ВС

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

План лекции

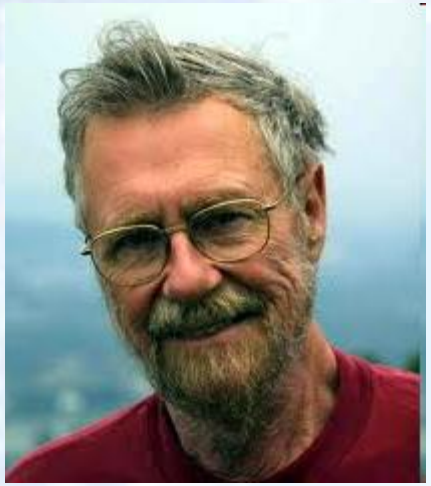


- Основные свойства МНПО
- Пуассоновские модели
- Марковские модели
- Байесовские модели
- Пример выбор модели оценки
- Задача оптимизации надёжности ВС
- Методы обеспечения отказоустойчивости

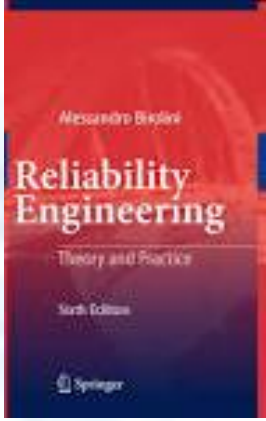
Эпиграф

- *Если отладка — процесс удаления ошибок, то программирование должно быть процессом их внесения.*

Эдсгер Дейкстра



Литература

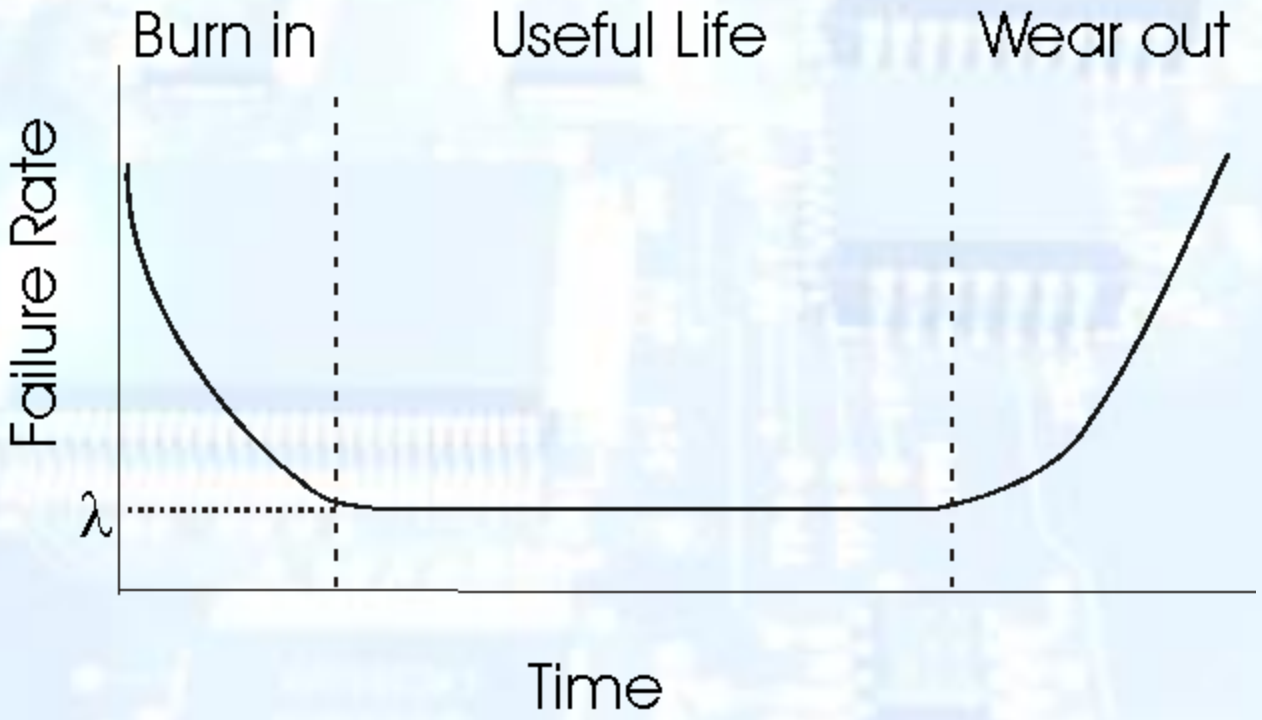


- Birolini A. Reliability engineering. – Berlin : Springer, 2007. – Т. 5. (Глава 2 – Reliability Analysis During the Design Phase)
- Lyu M. R. et al. Handbook of software reliability engineering. – CA : IEEE computer society press, 1996. – Т. 222.(Глава 3 - Software Reliability Modeling Survey) (http://www.cse.cuhk.edu.hk/~lyu/book/reliability/pdf/Chap_3.pdf)
- A.Wood Software Reliability Growth Models. Technical Report, 1996 (http://www.hpl.hp.com/techreports/tandem/TR-96.1.pdf)

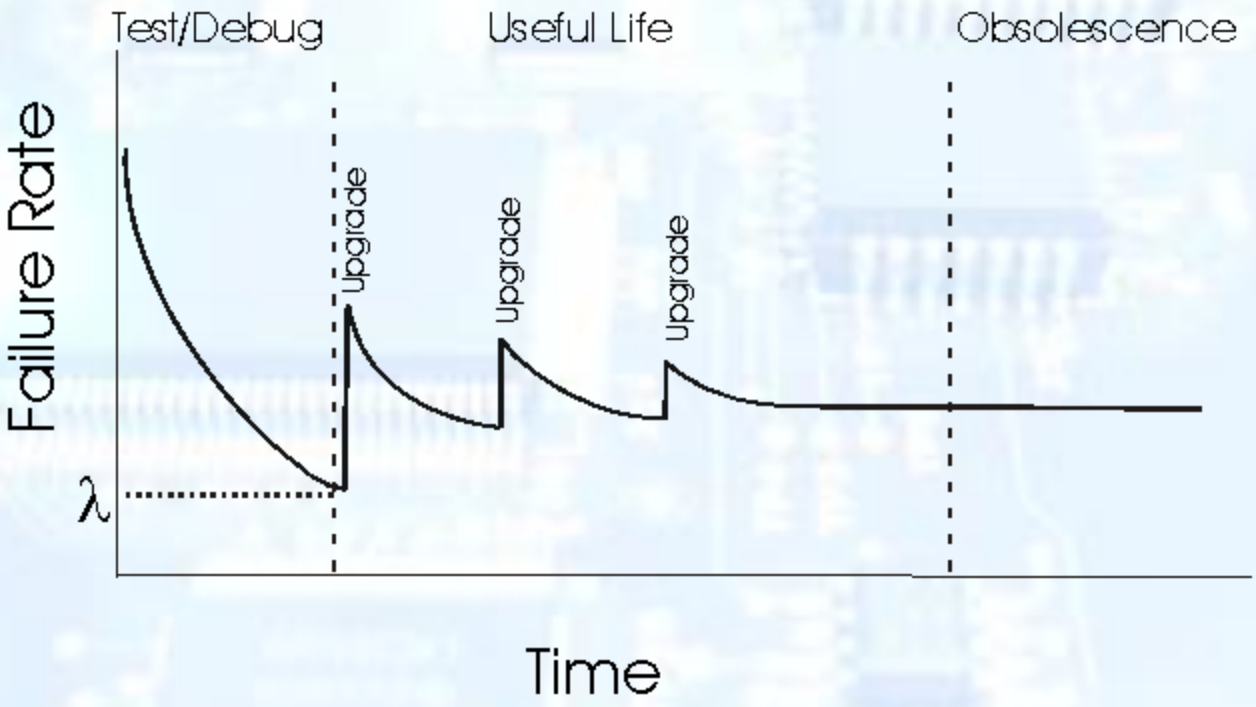
Литература (2)

- Kuo W., Wan R. Recent advances in optimal reliability allocation //Computational Intelligence in Reliability Engineering. – Springer Berlin Heidelberg, 2007. – С. 1-36. (http://content.schweitzer-online.de/static/content/catalog/newbooks/978/354/037/9783540373674/9783540373674_Excerpt_001.pdf)
- Pullum L. L. Software fault tolerance techniques and implementation. – Artech House, 2001.

Надёжность аппаратуры



Надёжность ПО



Особенности оценки надёжности ПО

- Оценка надёжности сложная проблема, так как не понятна природа ПО
- Непонятно как измерять размер ПО

Оценка числа ошибок в программе

- Задачи:
  - оценить число оставшихся дефектов в текущей версии программы.
  - оценить вероятность отказа компонента или системы
  - оценить среднее время до следующего отказа
- Возможны два пути:
  - Статический анализ кода
  - Анализ статистики тестирования

Пример статистики тестирования

Неделя	Проект 1	Проект 2	Проект 3	Проект 4
1	16	13	6	1
2	27	26	13	8
3	41	40	28	11
4	54	61	48	19
5	69	84	57	27
6	81	95	60	32
7	90	104		36
8	96	112		39
9	99	117		41
10	100			42

Наша задача – описать процесс обнаружения ошибок математической моделью

Пример статистики тестирования (2)

- Данные:
  - Время отказа
  - Интервал между отказами

Failure no.	Failure times (hours)	Failure interval (hours)
1	10	10
2	19	9
3	32	13
4	43	11
5	58	15
6	70	12
7	88	18
8	103	15
9	125	22
10	150	25
11	169	19
12	199	30
13	231	32
14	256	25
15	296	40

Пример статистики тестирования (3)

- Данные:
  - Зависимые отказы к заданному времени
  - Отказы во временной интервал

Time(s)	Cumulative Failures	Failures in interval
30	2	2
60	5	3
90	7	2
120	8	1
150	10	2
180	11	1
210	12	1
240	13	1
270	14	1

Модели надёжности

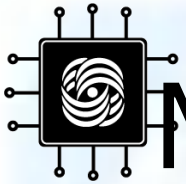
- Конечная цель – оценить надежность программы, основываясь на данных, полученных при тестировании.
- Суть моделей надёжности - подобрать адекватную модель, описывающую количество обнаруженных ошибок с помощью известных вероятностных распределений.

Принцип применения моделей надёжности

- Собрать данные об отказах
- Проверить данные (выбрать вид распределения)
- Выбрать модель
- Оценить параметры модели
- Настроить модель с выбранными параметрами
- Проверка критерия согласия
- Оценить требуемые параметры

Модели надёжности ПО

- С 1970х годов предложено более 200 моделей но как определить надёжность ПО вопрос открытый
- Одну модель нельзя использовать во всех ситуациях
- Нет полной модели или даже более менее репрезентативной



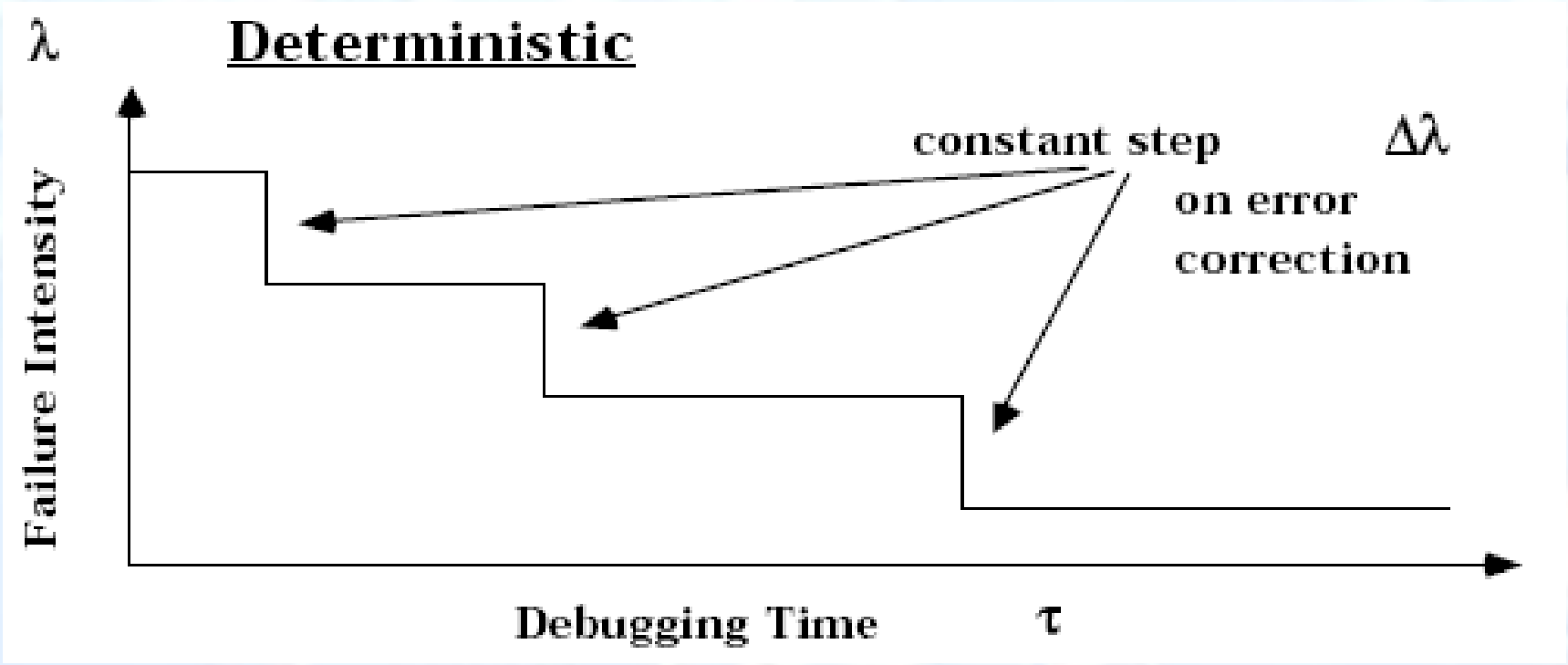
Модели надёжности ПО (2)

- Предположения
- Факторы
- Математическая функция

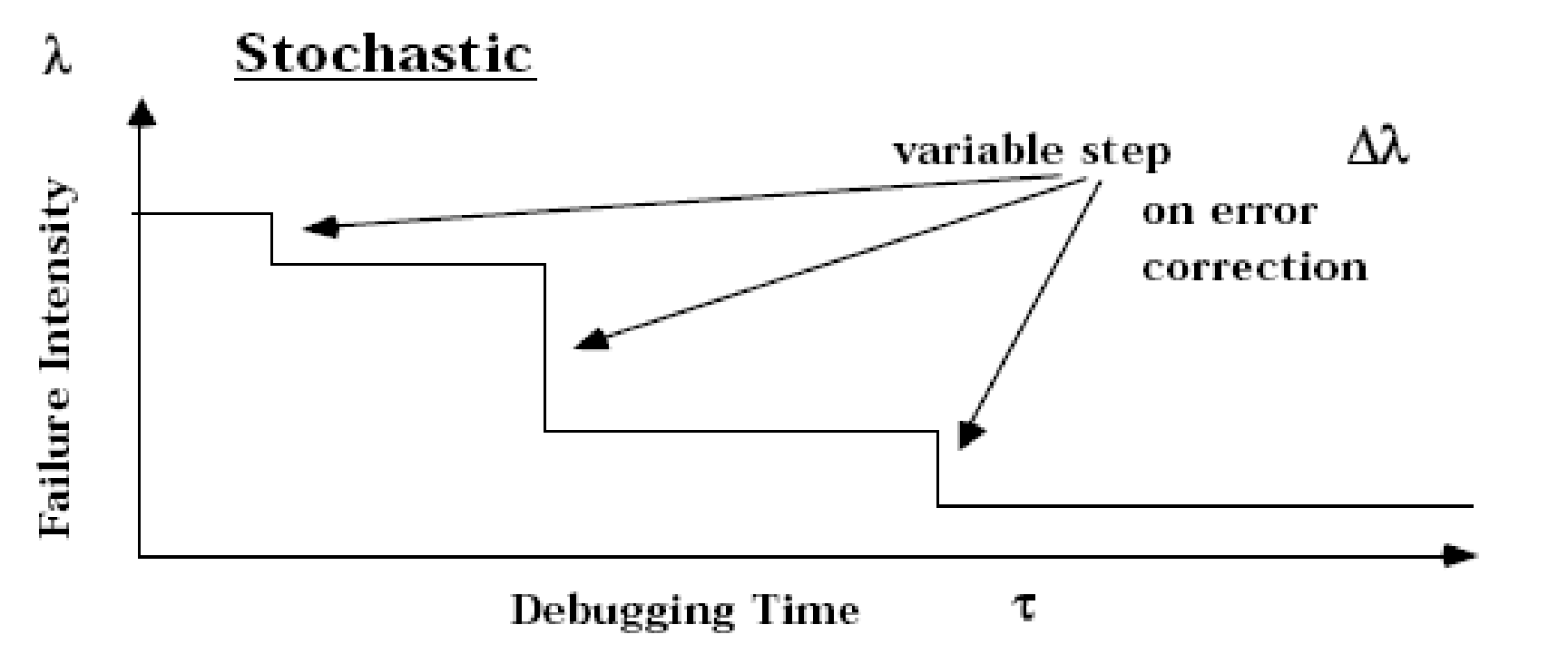
Виды моделей

- Пуассоновские
- Марковские
- Байесовские
- Комбинированные

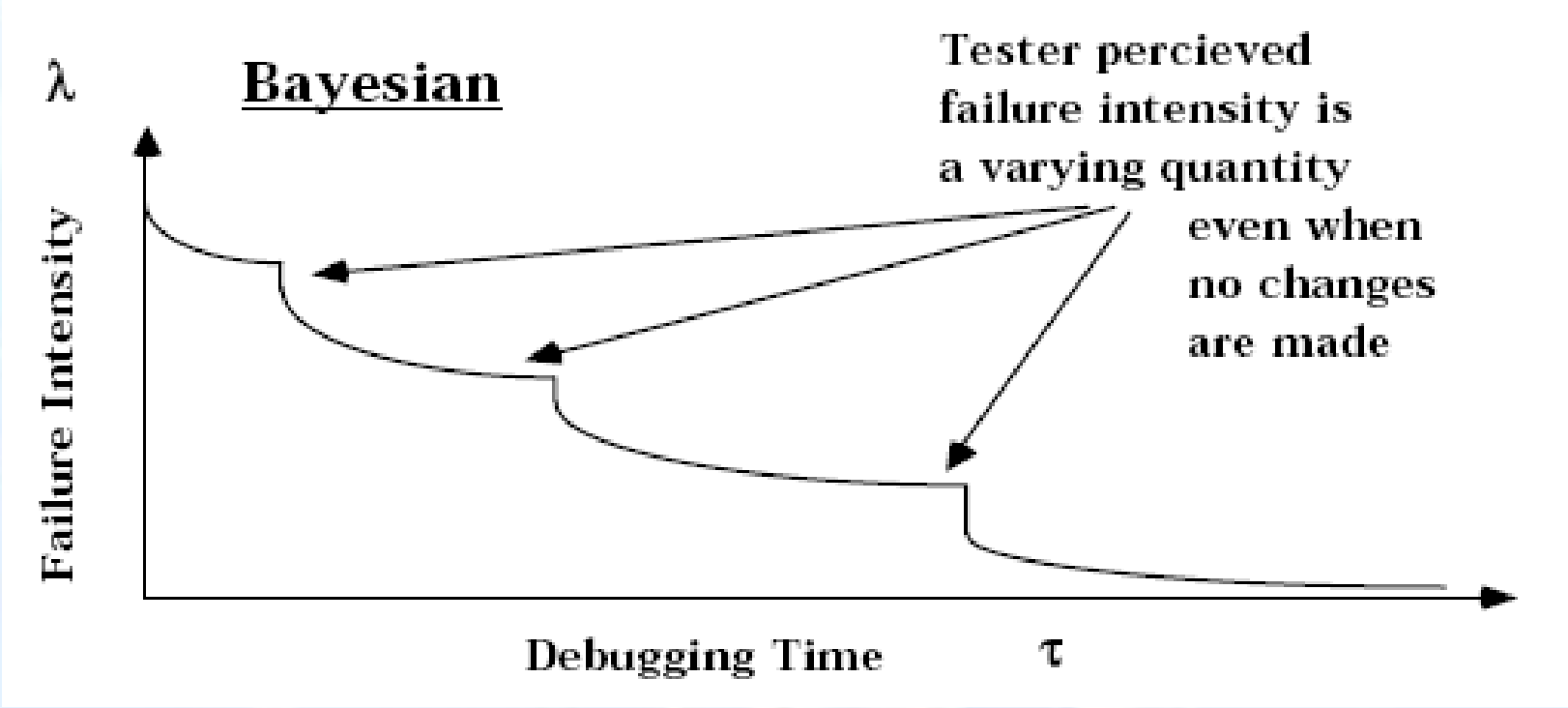
Модели надёжности ПО (3)



Модели надёжности ПО (4)



Модели надёжности ПО (5)



Основная теорема о пуассоновском процессе

- В каждый момент времени

$$P(\xi_t = k) = e^{-\lambda} \frac{\lambda^k}{k!}$$

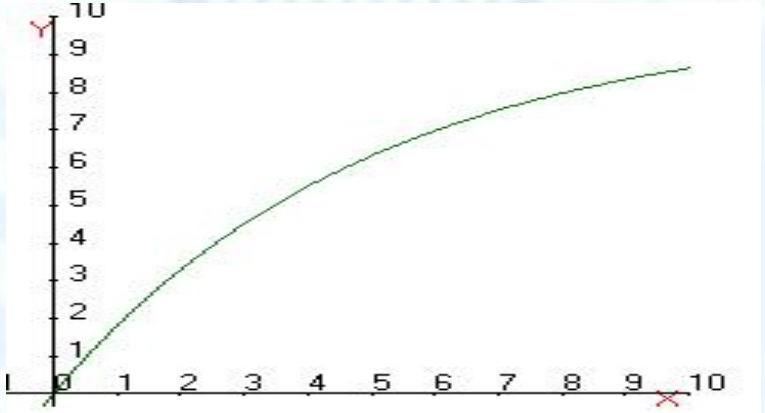
то есть имеет распределение Пуассона.

- Λ называется интенсивностью потока. В рассматриваемом случае интенсивность есть количество ошибок, найденных к данному моменту.

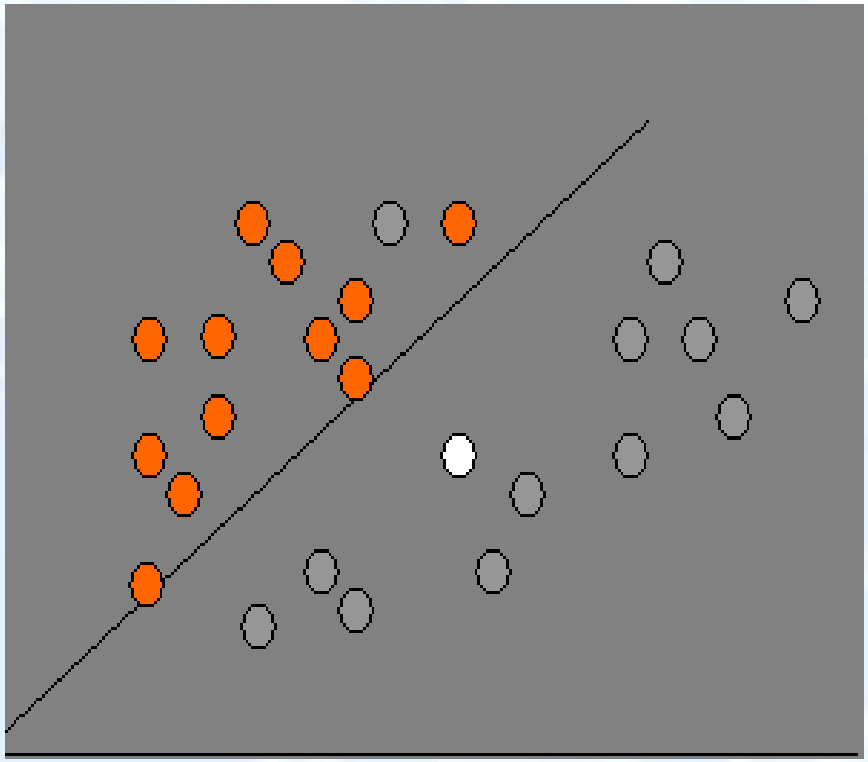
Экспоненциальная модель

- Наиболее простая из выпуклых моделей

$$m(t) = a(1 - e^{-bt})$$



Оценка параметров: линейная регрессия



- Происходит минимизация суммы квадратов расстояний до некой прямой.

Ограничения для SRGM

- Ошибки исправляются сразу после обнаружения
- При исправлении новых ошибок не появляется
- В период тестирования не появляется нового кода
- Тестирование централизовано, то есть в каждый момент одна версия программы тестируется одной группой тестеров.
- Ошибки не зависят друг от друга.

Примеры усовершенствованных моделей

- Данная модель позволяет избавиться от ограничения на зависимость ошибок друг от друга

$$m(t) = a * \frac{(1 - e^{-bt})}{1 + \Psi(r) e^{-bt}}$$

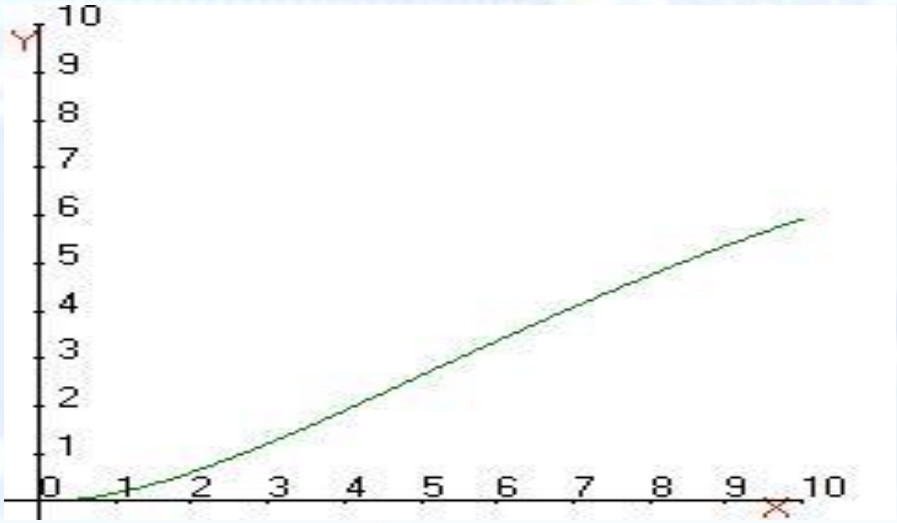
$$\Psi(r) = \frac{1 - r}{r}$$

Параметр r есть отношение количества ошибок, которые можно обнаружить в текущий момент, к общему количеству ошибок. При r = 1 получаем стандартную экспоненциальную модель.

Модель Yamada

- Кривая становится S-изогнутой, и происходящее можно рассматривать как модель для тестирования, при котором умения тестера постепенно увеличиваются.

$$m(t) = a(1-(1+bt)e^{-bt})$$



Обзор остальных моделей

- Модель Вейбулла – учитывает качество тестирования
- Модель Парето – учитывает, что более серьезные ошибки исправляются раньше
- …

Особенности пуассоновских моделей

- По пуассоновским моделям много работ, их применяли в разных проектах множество раз, и получали адекватные результаты.
- Возможность исследовать несколько параллельных независимых источников ошибок
- Изначально, пуассоновские процессы использовались для моделирования отказов оборудования.

Модель Jelinski-Moranda

- Рассматриваем процесс тестирования как марковскую цепь с неизвестным начальным состоянием. Полагая, что надежность линейно зависит от количества ошибок, можно считать время между обнаружением двух ошибок случайной величиной с экспоненциальным распределением.

Модель Jelinski-Moranda

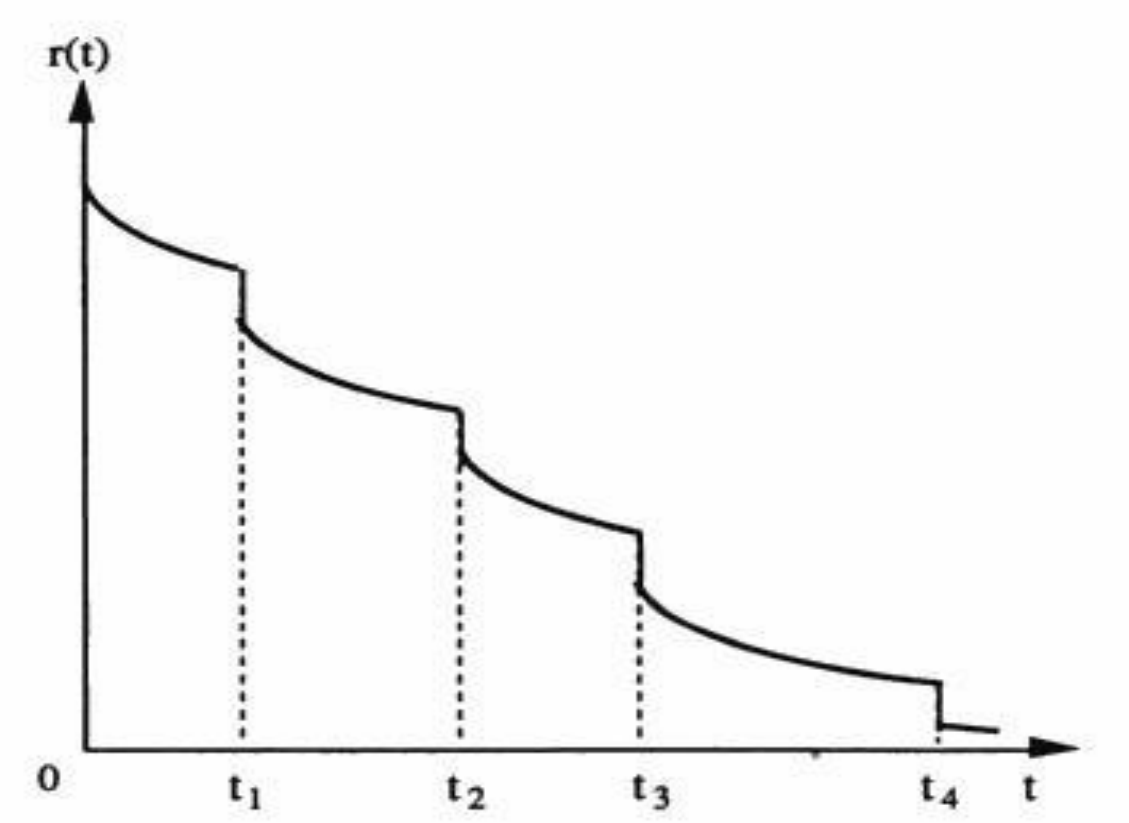
- Пусть в программе a ошибок, в начальный момент интенсивность равна a*b, где b – положительный параметр. Ti – время между появлением i-й и i-1 ошибок имеет показательное распределение с интенсивностью (a – i + 1)*b

Байесовский подход

- Байесовскими называют различные модели, учитывающие новые данные для корректировки существующих гипотез. Если в описанных выше моделях используются некоторые предположения о характере распределения a priori, то при использовании байесовских методов эти параметры меняются в зависимости от новых измерений.

Частота в LV-модели

Частота – скорость изменения интенсивности, т.е. m'(t).



Открытый вопрос

- Не решена проблема выбора между существующими SRGM. Критерии применимости той или иной SRGM к конкретному проекту до конца не сформулированы

Пример входных данных

Cum. Failures	Cum. Time	Intensity	Time
5	342	0.014620	171.00
10	571	0.021834	456.50
15	968	0.012594	769.50
20	1984	0.004921	1476.00
25	3098	0.004488	2541.00
30	5049	0.002563	4073.50
35	5324	0.018182	5186.50
40	6380	0.004735	5852.00
45	7644	0.003956	7012.00
50	10089	0.002045	8866.50
55	10982	0.005599	10535.50
60	12559	0.003171	11770.50
65	14708	0.002327	13633.50
70	16185	0.003385	15446.50
75	17758	0.003179	16971.50
80	20567	0.001780	19162.50

Интенсивность

$$\text{Intensity (failures/CPU sec)} = \frac{\Delta f}{\Delta t} = \frac{5}{\text{Cum.T}_2\text{-Cum.T}_1}$$

$$\text{Time (average)} = \text{Cum.T}_1 + \frac{\Delta t}{2}$$

- Две модели "basic execution time model‟ и "logarithmic Poisson execution time model" (e.g. Musa et al. 1987).

Basic Execution Time Model

- **Basic Execution Time Model**
  - Интенсивность отказов λ(т) за время т:

$$\lambda(\tau) = \lambda_0 e^{-\frac{\lambda_0}{\nu_0}\tau}$$

  - где λ0 начальная интенсивность и v0 ожидаемоеколичество отказов.

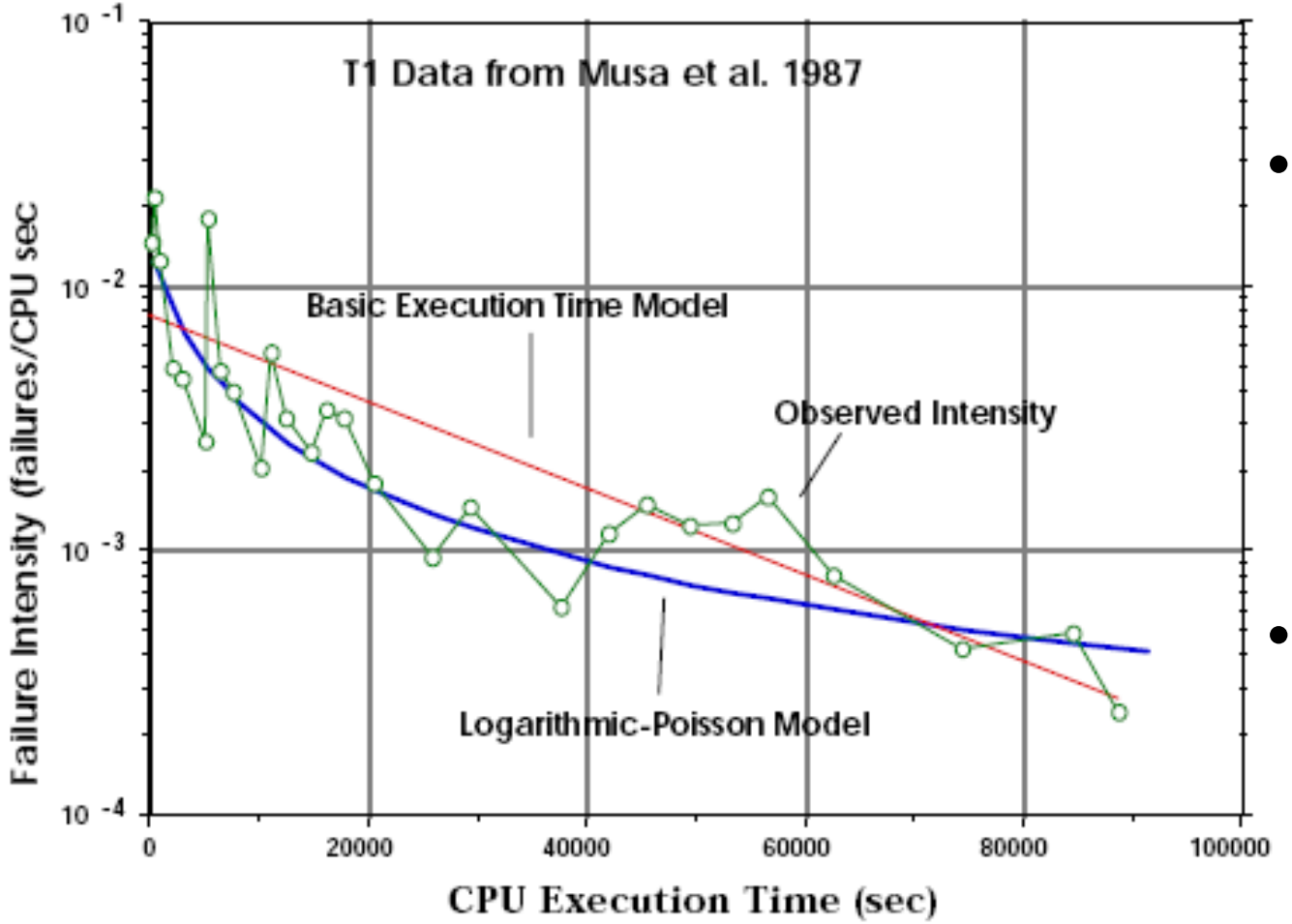
Logarithmic Poisson execution time model

$$\lambda(\tau) = \lambda(\mu) = \lambda_0 \left(1 - \frac{\mu}{\nu_0}\right)$$

- где μ(т) среднее время между откзами к времени т.

$$\mu(\tau) = \nu_0\left(1 - e^{-\frac{\lambda_0}{\nu_0}\tau}\right)$$

Пример (2)

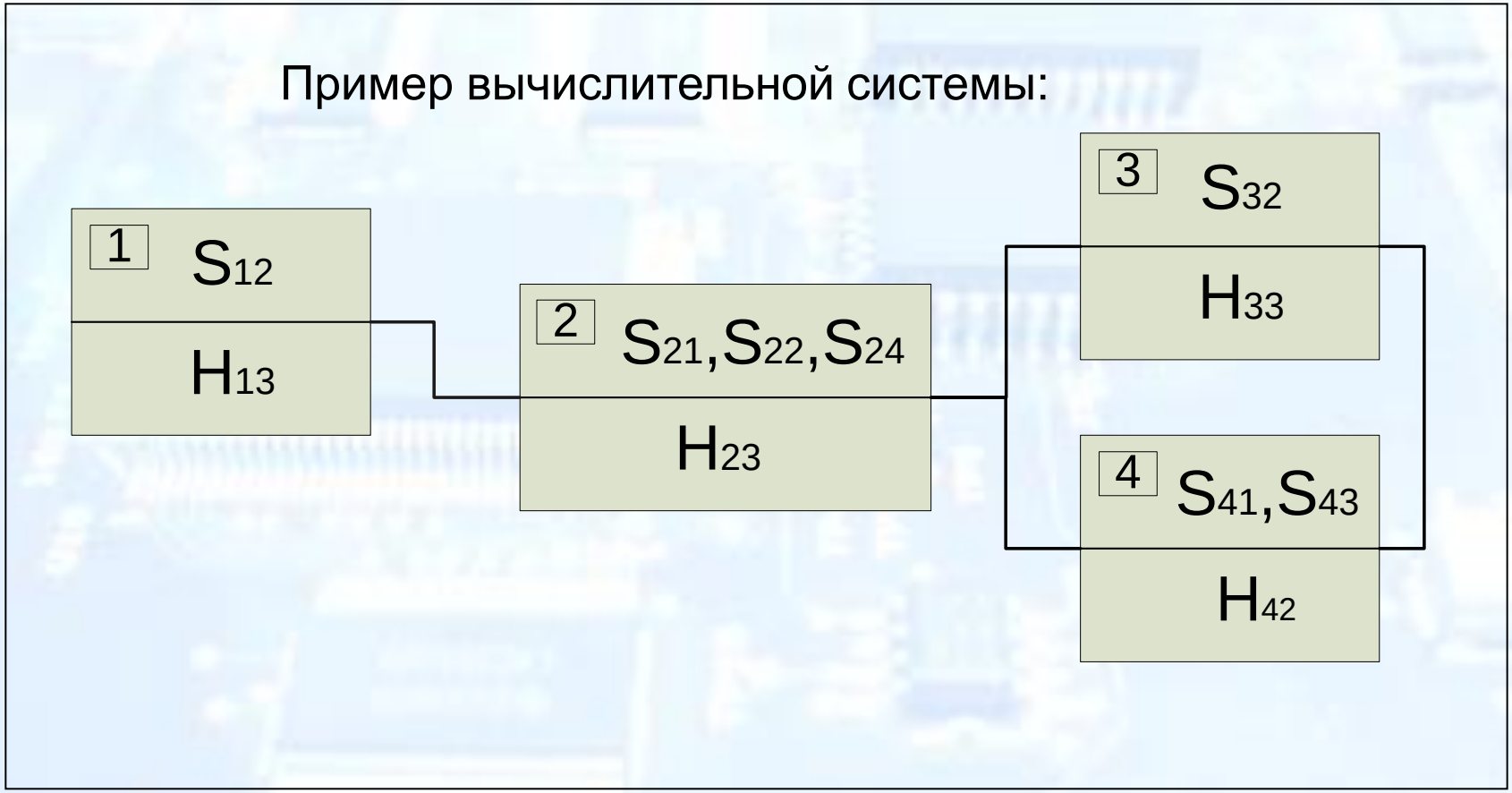


- В этом случае Logarithmic-Poisson Model лучше приближает the Basic Execution Time Model.
- В некоторых других проектах BE model приближает лучше чем LP модель.

Выводы

- Нет универсальной модели
- Измерения очень важны для нахождения корректной модели
- Результаты тестирования необходимо использовать и не один раз
- Модели надёжности ПО не так просты как описаны здесь ☺

Неформальная постановка задачи оптимизации надёжности ВС



Классическая постановка задачи оптимизации надёжности ВС

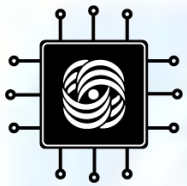
- *Дано:*
  - N – количество подсистем
  - Ch_{ij} , Cs_{ij} – стоимость использования аппаратного/программного компонента j для подсистемы i
  - Rh_{ij} , Rs_{ij} – надёжность аппаратного/программного компонента j для подсистемы i
  - Prv, Pd, Pall – вероятности отказа нескольких версий программных компонентов, вероятность отказа схемы принятия решений, вероятность отказа сразу всех версий программного компонента
- *Необходимо найти:*
  - Оптимальный набор компонентов и МОО, на котором $R_{RTES} \rightarrow max$
- *Ограничения:*
  - $C_{RTES} < Cost$

Принятые ограничения

- Все компоненты являются неремонтируемыми
- Моменты появления отказов для аппаратуры статистически независимы
- Все аппаратные компоненты ВС являются активными
- Количество доступных версий для всех компонентов фиксировано
- Интенсивность отказов постоянно

Особенности задачи

- NP-трудная
- Аргументы и значения функции надежности являются дискретными
- Многоэкстремальная область допустимых решений
- Область допустимых решений несвязна
- Функция надежности нелинейна
- Функция стоимости линейна
- Каждый модуль вычислительной системы имеет свой доступный набор механизмов обеспечения отказоустойчивости



Методы решения задачи

- Динамическое программирование
- Точные методы
- Алгоритмы муравьиных колоний
- Генетические алгоритмы
- Имитация отжига
- Иммунные алгоритмы
- Прочие эвристики
- Поиск с отсечением

Цели

- Отказоустойчивая компьютерная система должна предоставлять сервисы в случае ошибок
- Отказы могут возникать, так как дефекты присутствуют в компонентах системы
- Система с отказоустойчивостью надёжней системы без отказоустойчивости, но больше затрачивается ресурсов на обеспечение отказоустойчивости

Проблемы …

- Традиционные подходы к отказоустойчивости в аппаратных системах основываются на копировании с использованием особенностей отказов физических компонентов.
- Большинство аппаратных методов отказоустойчивости не могут быть применены напрямую в ПО, где почти все ошибки разработчика.

Пример

$$\mathbf{x} \cdot \mathbf{y} = \sum_{i=1}^{6} x_i \cdot y_i$$

$$\mathbf{x} = \left(10^{20}, 1223, 10^{24}, 10^{18}, 3, -10^{21}\right)$$

$$\mathbf{y} = \left(10^{30}, 2, -10^{26}, 10^{22}, 2111, 10^{19}\right)$$

- Правильный ответ должен быть 8779
- Но при обычной реализации возвращается 0

История ...

- **Защитное программирование:**
  - ad-hoc методы минимизирующие ущерб который может возникнуть из-за ошибок
- **Метод двух версий:**
  - Создание двух независимых версий ПО и запуск их. По любому расхождению в версиях включается триггер

Методы обеспечения отказоустойчивости

- Обнаружение ошибок
- Диагностическое тестирование
- Изоляция ошибок
- Маскировка ошибок
- Корректирование ошибок
- Устранение ошибок

Требования к компонентам

Организация резервирования

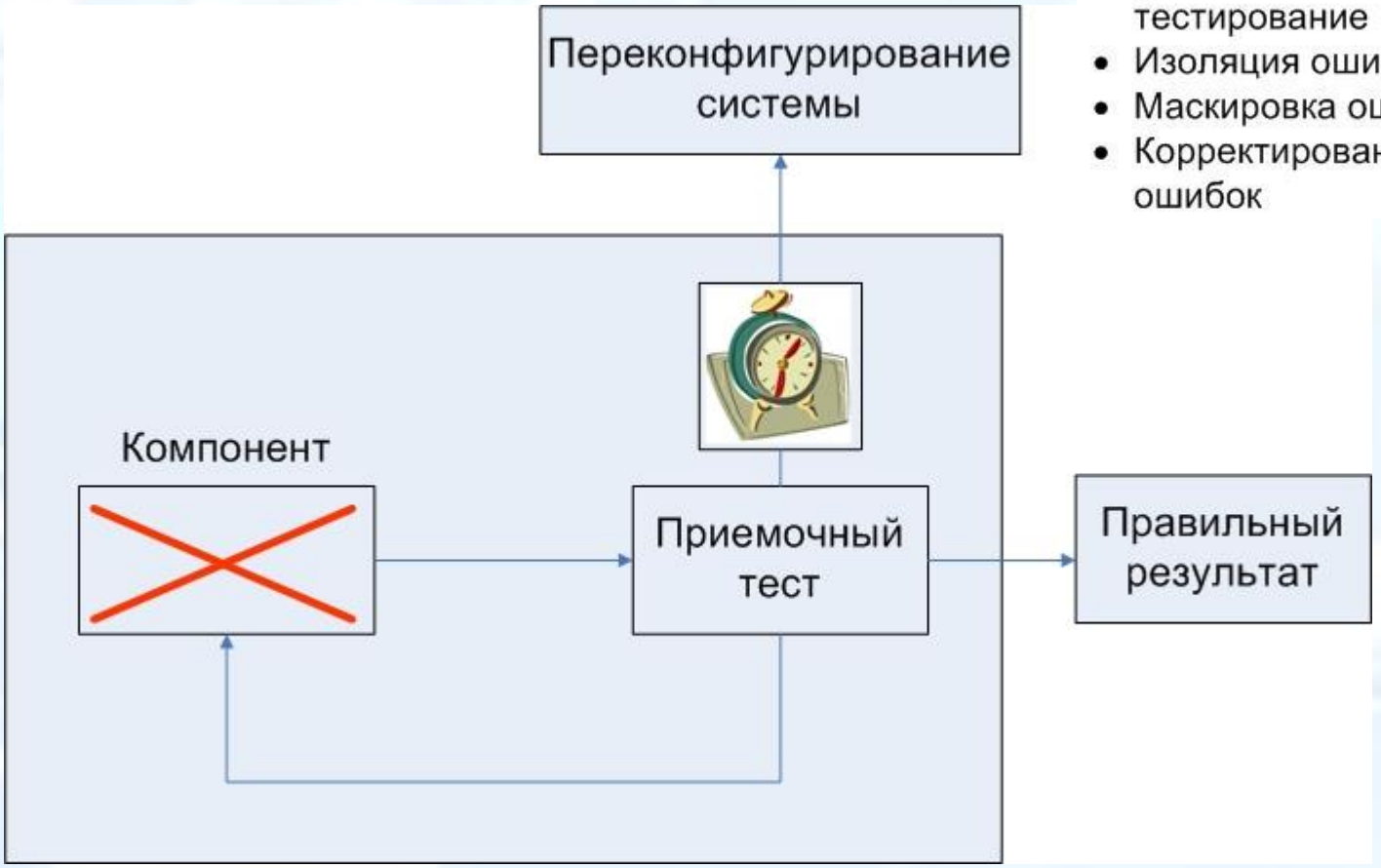
- Общие подходы
  - Организация глобального времени
  - Выделение изолированных регионов
  - Отказы в разделяемых компонентах
- Программное резервирование
  - Пространственное резервирование
  - Временное резервирование
    - Функциональный сдвиг во времени
    - Информационный сдвиг во времени
- Аппаратное резервирование
  - Пространственное резервирование
  - Кодирование

Механизмы обнаружения ошибок

- Программные
  - Приемочные тесты
  - Отказоустойчивые алгоритмы
  - Проверки
    - Временные
    - Кодовые
    - Реверсные
    - Семантические
    - Структурные
- Аппаратные
  - Диагностическое тестирование

Приемочные тесты

- Диагностическое тестирование
- Изоляция ошибок
- Маскировка ошибок
- Корректирование ошибок



Отказоустойчивые алгоритмы. Попарное тестирование

- Диагностическое тестирование
- Изоляция ошибок
- Маскировка ошибок
- Корректирование ошибок

Отказоустойчивые алгоритмы. Голосование

- Диагностическое тестирование
- Изоляция ошибок
- Маскировка ошибок
- Корректирование ошибок

Механизмы устранения ошибок

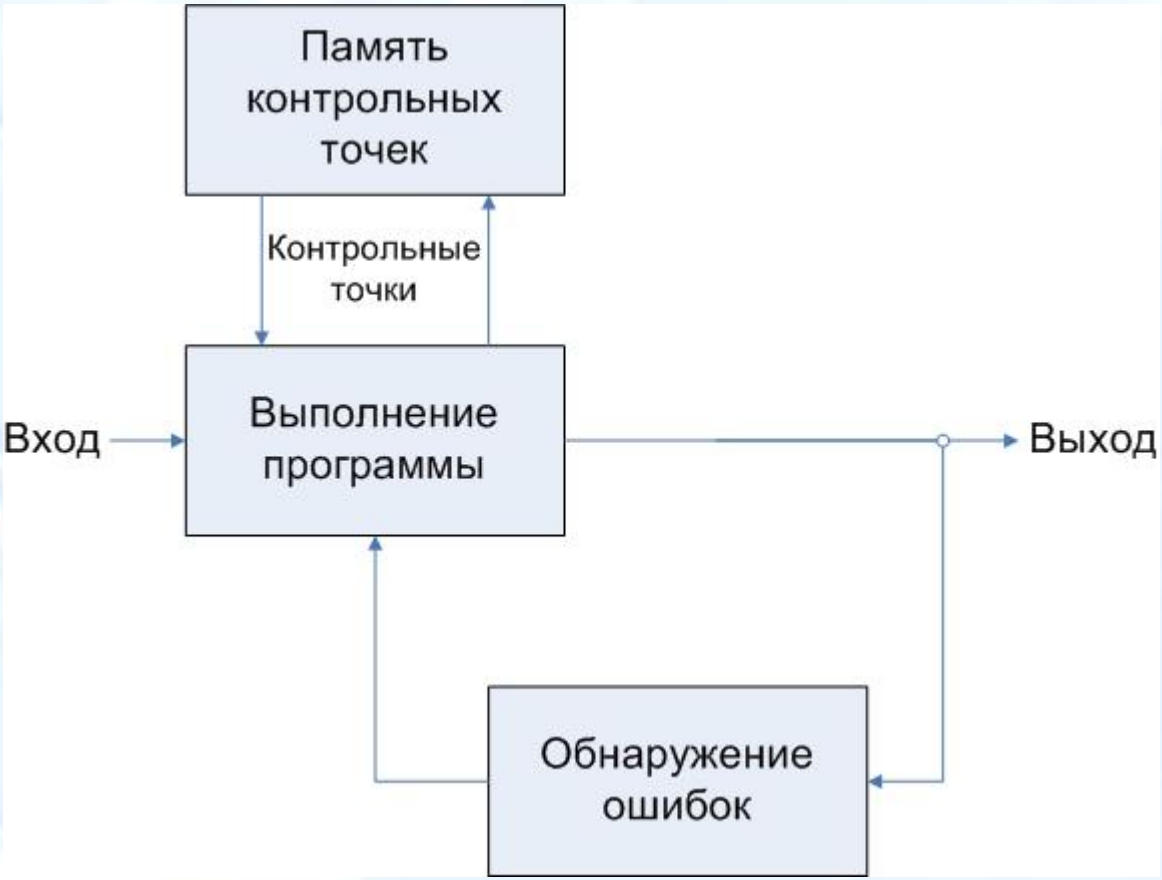
- Программные
  - Одноверсионное программирование
    - Контрольная точка и перезапуск
    - Парные прогоны
  - Многоверсионное программирование
    - Восстановление блоками
    - N-версионное программирование
    - N-самотестируемое программирование
- Аппаратные
  - Резервирование компонент
  - Переконфигурирование системы

Схемы голосования

голосователь сравнивает результаты двух и более функционально эквивалентных компонентов ПО и определяет корректныйрезультат

- Схемы голосования:
  - Большинство
  - Консенсус
  - 2-из-N

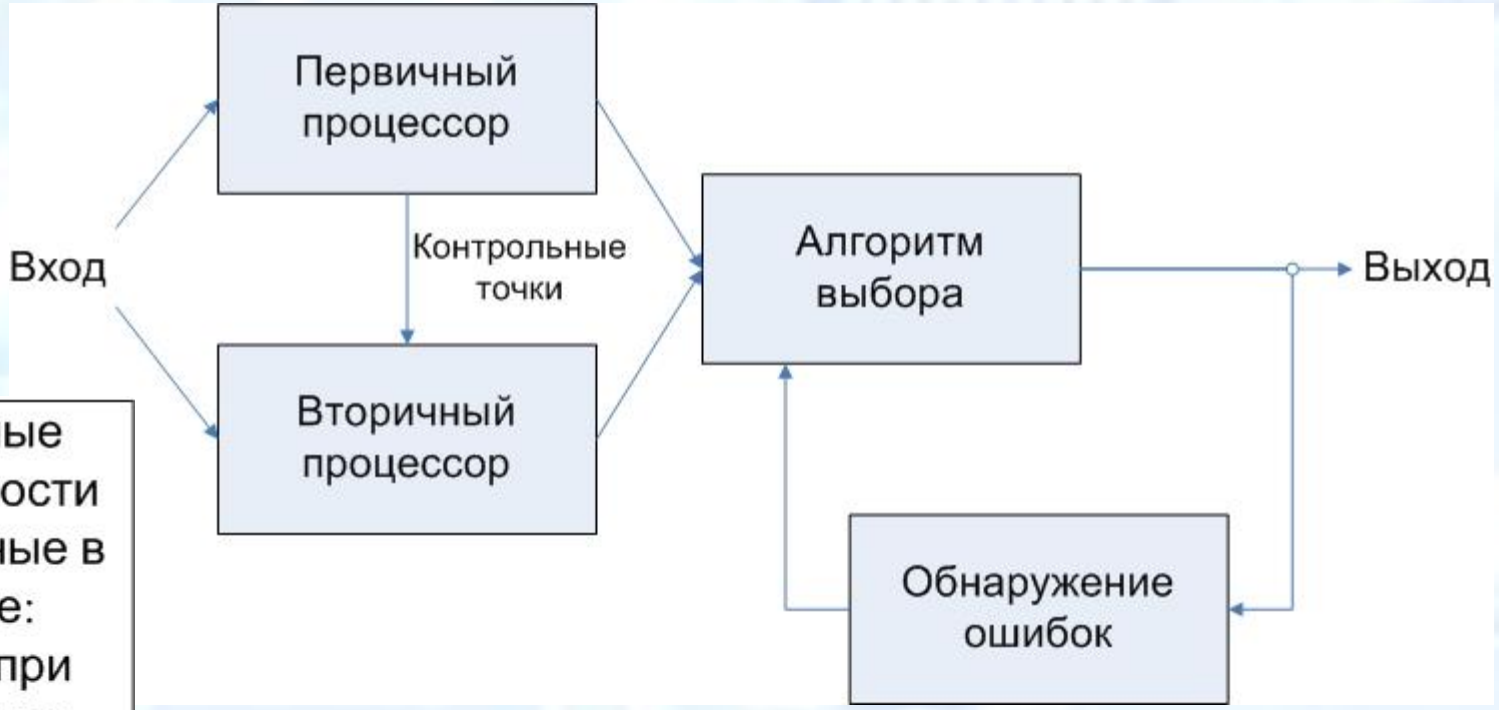
Контрольная точка и перезапуск



∀ временные неисправности:

- врем. э/м помеха
- приработка устройства

Парные прогоны



∀ временные неисправности + постоянные в аппаратуре:

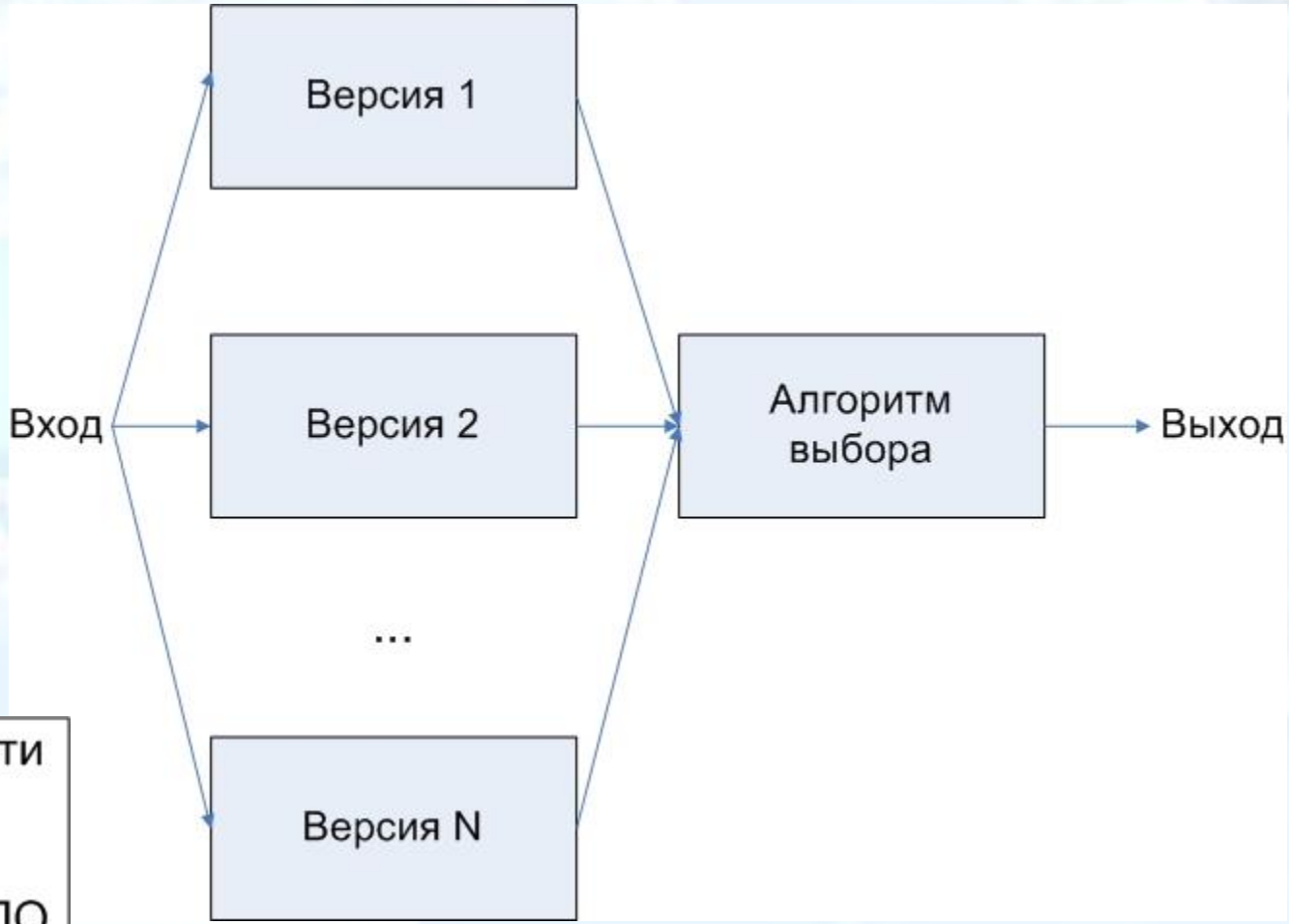
- ошибка при разработке аппаратуры
- ошибка ввода

Восстановление блоками

∀ временные неисправности + постоянные в ПО:

- ошибка при разработке ПО
- врем. э/м помеха

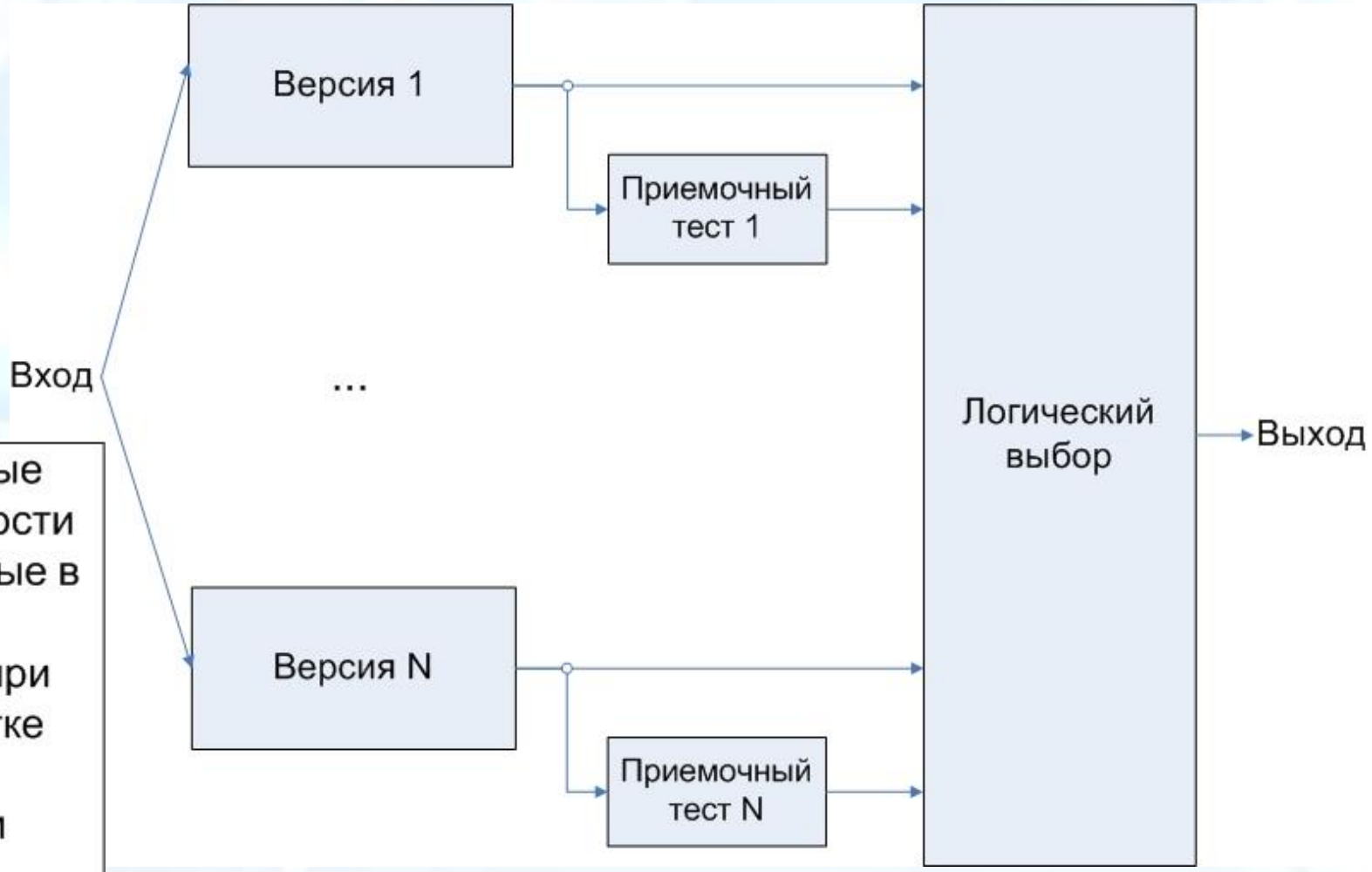
N-версионное программирование



∀ неисправности в ПО:

- ошибка при разработке ПО
- ошибка ввода

N-самотестируемое программирование



∀ временные неисправности + постоянные в ПО:

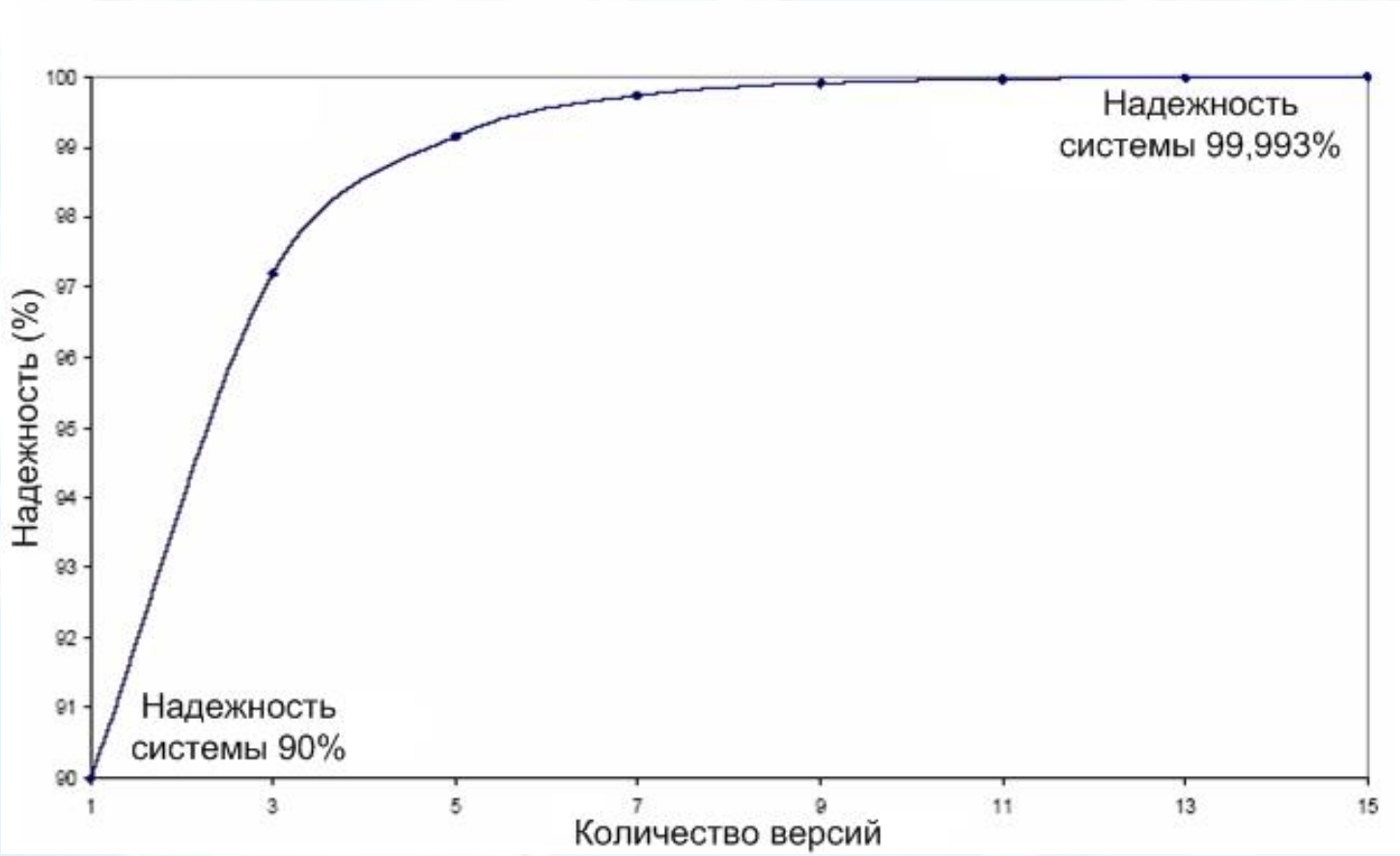
- ошибка при разработке ПО
- врем. э/м помеха

N-версионное программирование. Примеры

Год	Проект и спонсор	Кол-во версий	Требуемое разнообразие	ЯП
1975-76	Текстовый редактор Software Engineering	27	Разработчики	PL/1
1977-78	Решение ДУ Software Engineering	16	Разработчики + 3 алгоритма	PL/1
1979-83	Аэропортовый планировщик NSF	18	Разработчики + 3 спецификации	PL/1
1984-86	Отказоустойчивый датчик NASA	5	Разработчики	Pascal
1986-88	Беспилотная посадка самолета Sperry Flight Systems	6	Разработчики + 6 языков	Pascal, Ada, Modula-2, C, Prolog, T

Космический шаттл

Зависимость надежности от количества версий



Заключение

- Необходимо корректное покрытие приёмочными тестами для обнаружения ошибок
- Часто невозможно быстро проверить корректность процедуры (например, для алгоритма “сортировки” это также сложно как сам алгоритм “сортировки”).
- Проверки часто могут быть очень ресурсоёмкими.
- ***Возможность проявления ненайденных ошибок не должно быть определяющим***

Спасибо за внимание!