Министерство образования и науки Российской Федерации

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Ю.М. Морозов

НАДЕЖНОСТЬ АППАРАТНО-ПРОГРАММНЫХ КОМПЛЕКСОВ

Санкт-Петербург

Кандидат технических наук, доцент СПбГПУ В.Г. Токмаков.
Кандидат технических наук, начальник отдела надежности «Океанприбор» А.И. Муравьев.

Морозов Ю.М. **Надежность аппаратно-программных комплексов**: учеб. пособие / **Ю.М. Морозов**. – 2011. – 136 с.

Учебное пособие развивает основные положения теории надежности применительно к сложным аппаратно-программным комплексам, содержит последовательное изложение методов оценки показателей надежности двухполюсных, иерархических и сетевых структур технических средств. Рассматриваются вопросы влияния резервирования на показатели надежности систем. Большое внимание уделено рассмотрению математических моделей надежности программного обеспечения для этапа отладки, методам оценки параметров математических моделей по статистическим результатам детерминированного тестирования.

Учебное пособие предназначено для обучения студентов высших учебных заведений по направлению подготовки магистров «Информатика и вычислительная техника», «Программная инженерия», «Системный анализ и управление».

ВВЕДЕНИЕ

С отказами, их проявлениями и последствиями человечество сталкивалось практически всегда. Пещера, выбранная питекантропом для проживания, могла оказаться ненадежной. Обвал потолка пещеры мог привести к жертвам. К этому же мог привести неправильный выбор палки для копья. Каменный топор, состоящий из рубила, палки и лианы, был уже системой, эта система отказывала чаще чем любой элемент, входящий в ее состав.

В течение многих веков лучшие умы человечества пытались создать устройства. Которые бы освободили их от рутинного занятия – проведения вычислений. Но эффект был всегда один – устройства, сконструированные идеально, практически не работали. Уровень надежности элементной базы не позволял строить системы определенной сложности. Так было и с цифровой вычислительной машиной Чарльза Беббиджа и с машиной «Марк-1» , созданной по проекту Говарда Айкена.

Теория надежности относительно молодая наука. Возникла она тогда , когда стали строить сложные системы, состоящие из большого числа ненадежных элементов. Первые публикации посвященные проблеме построения надежных систем из ненадежных элементов появились в середине 50-ых годов прошлого столетия. ЭВМ – сложная система, переход от одного поколения к другому был связан, как правило, с тем, что транзистор по показателям надежности оказался на порядок лучше, чем электронная лампа , а микросхема малой, средней а затем и большой степени интеграции лучше чем транзистор.

В ЭВМ третьего поколения слабым звеном, с точки зрения надежности, оказалось программное обеспечение. Поэтому с 80-тых годов двадцатого

столетия вопросам надежности программного обеспечения уделяется особое внимание при создании любых аппаратно-программных комплексов

Программное обеспечение современных информационных и управляющих систем представляет собой чрезвычайно сложный комплекс, в создании которого принимают участие сотни различных специалистов. При эксплуатации таких комплексов возникают сбои и отказы, обусловленные искажениями программ и данных. Высокая ответственность систем управления и обработки информации обусловила необходимость изучения причин программных отказов и методов борьбы с ними. Недостаточная надежность программных комплексов может причинить ущерб, значительно превышающий их собственную стоимость. Высокая эффективность таких систем может быть достигнута при соответствующей надежности функционирования. Даже малое количество не установленных в программе ошибок может значительно снижать надежность программного обеспечения вычислительных систем.

Высокое качество программ может достигаться безошибочным проектированием ("пассивными методами") и выявлением и устранением ошибок ("активными методами"). Однако методы безошибочного проектирования, способствуя значительному повышению качества программ, не могут гарантировать удовлетворение всех заданных требований к программному продукту, а главное, не полностью предотвращают ошибки.

Активные методы поиска и устранения ошибок дополняют пассивные в процессе достижения заданного качества и позволяют оценивать ряд показателей качества. Основным активным методом является тестирование, которое состоит в проверке программ на соответствие заданным правилам построения и конкретным результатам их исполнения.

Современная практика тестирования программ базируется, в основном, на квалификации и интуиции специалистов, что приводит к различию

трудоемкости создания программ и достигаемого их качества. Из-за "оптимизма" многих программистов и пренебрежения планомерным, систематическим тестированием в программах остаются ошибки, выявляемые впоследствии при завершении отладки или в процессе эксплуатации. Для эффективного тестирования программ необходима методологическая и инструментальная (средства автоматизации) база.

Основным методом обнаружения ошибок при отладке программ является их тестирование. При этом затраты на тестирование для обнаружения ошибок являются наибольшими, достигают 30-40% общих затрат на разработку программ и в значительной степени определяют качество созданного программного продукта. Высокая доля затрат на тестирование приводит к необходимости создания методов и средств, позволяющих достигать максимального качества программ при реальных ограничениях на длительность тестирования и на связанные с этим затраты. Эффективность тестирования является важнейшим фактором, определяющим стоимость и длительность разработки сложных комплексов программ с заданным качеством.

Основной целью тестирования является выявление всех отклонений результатов функционирования реальной программы от заданных эталонных значений. При этом задача состоит в обнаружении максимального числа ошибок, в качестве которых принимается любое отклонение от эталонов. Наиболее трудоемкими и детализирующими являются методы детерминированного тестирования. Основной задачей детерминированного тестирования является установление факта работоспособности программ и соответствия их требованиям технического задания, выявление и устранение ошибок и доведение характеристик программ до уровня требований, заданных заказчиком. При детерминированном тестировании контролируется каждая комбинация исходных эталонных данных и соответствующая ей комбинация результатов функционирования программы. Это позволяет выявлять отклонение результатов от эталона с конкретным фиксированием

всех значений исходных и результирующих данных, при которых это отклонение было обнаружено.

Проблема обеспечения необходимой надежности программных комплексов пока мало изучена, однако в настоящее время наблюдается устойчивая тенденция к расширению масштабов исследований в этой области. Однако, пока не существует единственного подхода к решению проблемы достижения качественного и надежного программного обеспечения. Тем не менее даже приближенные методы позволяют получать ряд важных оценок, необходимых для выбора наиболее целесообразных решений при проектировании программных комплексов.

ГЛАВА 1. ОСНОВНЫЕ ПОНЯТИЯ И КОЛИЧЕСТВЕННЫЕ ХАРАКТЕРИСТИКИ НАДЕЖНОСТИ ТЕХНИЧЕСКИХ СИСТЕМ

1.1 Основные понятия теории надежности

Надежность – [ГОСТ 27.002-83] – это свойство любого изделия сохранять в допустимых пределах основные параметры в течение заданного интервала времени при заданных условиях эксплуатации.

Изделие – это некий объект, ориентированный на некое функциональное использование (реализующий некий набор функций).

Этот объект может быть либо элементом, либо системой.

Элемент – это некий объект, который далее не может быть разделен на составные части.

Система – это некая совокупность элементов, которые в процессе реализации заданных функций взаимодействуют между собой.

Более 3-х элементов -> система сложная.

В рамках функционального наполнения надежность разделяется:

- **безотказность** – свойство изделия безотказно выполнять все свои функции в течение заданного интервала времени
- **долговечность** – свойство выполнять свои функции в течение заданного интервала времени с перерывами на требуемый ремонт и профилактическое обслуживание

- **ремонтопригодность** – свойство объекта, связанное с его приспособленностью для проведения профилактических и ремонтных работ
- **сохраняемость** – свойство изделия сохранять свои основные параметры в процессе транспортировки и хранения

Живучесть – это свойство объекта – ... – только в случае воздействия целенаправленных поражающих факторов на объект. (Но это понятие не из теории надежности)

Для любого объекта в теории надежности существует понятие работоспособного и неработоспособного состояния.

Работоспособное состояние – когда все важнейшие параметры объекта или системы находятся в пределах, задаваемых спецификациями.

Неработоспособное состояние – когда хотя бы один из важнейших параметров ушел за пределы допустимого уровня.

Пример: утюг – если перегорает спираль, то все параметры остаются прежними, а не нагревает -> неработоспособное сост. А если сгорела лампочка, то значения не имеет, т.к. это не важнейший параметр.

Переход из работоспособного состояния в не работоспособное связан с **отказом** системы.

Отказы в системах бывают 2-х основных типов:

- внезапные
- постепенные

внезапный отказ – это скачкообразное изменение состояния системы, связанное с неблагоприятным сочетанием внешних и внутренних параметров

постепенный отказ – связан с медленным движением неких параметров по допустимой области, на границу и выход в недопустимую область

Существует понятие частичного и полного отказа:

- **частичный отказ** – не исполняется некая совокупность функций
- **полный отказ** – ситуация в изделии, когда не исполняются все функции изделия

Отказы бывают:

1) конструктивные
2) производственные
3) эксплуатационные

1-е связаны с недоработками при конструировании (в процессе проектирования)
2-е – с нарушениями технологии производства
3-е – с неправильной эксплуатацией изделия

Перемежающий отказ – периодический отказ одной и той же природы.
Сбой (самоустраняющийся отказ) – они идут на несколько порядков чаще, чем отказы.

Для ПО характерно то, что все его отказы связаны с ошибками, которые остались в нем после его отладки.
Все вопросы о надежности связаны с большими программными продуктами длительного срока эксплуатации. Также ПО не подвержено старению.

Под **программным отказом** понимается состояние, когда в условиях, удовлетворяющих спецификациям, ПО всетаки выдает результаты, не соответствующие спецификации, документации или ожиданию пользователя.

(В таком определении есть элемент субъективизма, он должен быть по возможности убран)

ПО отказы (программные отказы):

1) не обесценивающие
2) частично обесценивающие
3) полностью обесценивающие

Не обесценивающие отказы – незаметны, влияние на результат незначительно

Частично обесценивающие – обесценивают часть наработки ПО к моменту обнаружения отказа (например, точки рестарта, когда после последней точки рестарта)

Полностью обесценивающие отказы – приводят к полному обесцениванию результата работы программного обеспечения и для получения верного результата необходим полный пересчет.

Наработка – календарное время эксплуатации изделия от момента выпуска до момента предельного его состояния.

Ресурс – время работы изделия от момента выпуска до предельного состояния.

Существует более 40 показателей надежности в ГОСТе (под разные изделия – свои показатели).

Все изделия делятся на **восстанавливаемые** и **не восстанавливаемые.**

Восстанавливаемое – это изделие, у которого есть восстанавливающий орган, который обеспечивает локализацию причины отказа и замену

элемента системы, и ЗИП. (Восстанавливающий орган – ремонтная бригада. ЗИП – запчасти и принадлежности.)

Не восстанавливаемое – это изделие которое после отказа снимается с эксплуатации и заменяется другим (например, лампочка).

В теории надежности существует понятие:

Безизбыточная система – любая система, для которой отказ любого элемента есть отказ всей системы.

Уровень развития техники не позволяет строить безизбыточные системы с нужной надежностью -> для повышения надежности используется **избыточность** (в систему вводятся дополнительные элементы)

Избыточность – некие дополнительные ресурсы, вводимые в систему.

В вычислительных системах используются **4 вида избыточности**:

1) Структурная
2) Информационная
3) Функциональная
4) Временная

Структурная может быть на уровне одного элемента, группы элементов или всей системы в целом -> выделяют **поэлементное** и **групповое резервирование**.

При этом выделяют **основной** элемент (выполняет основные функции) и **резервный** (ничего не исполняет).

Если резерв использует ресурсы, то говорят о **горячем** резерве, в противном случае имеем **холодный** резерв. **Облегченный** – часть использует, а часть нет.

Скользящий резерв – используется, когда в системе есть несколько одинаковых элементов, резерв может заменять любой.

Информационная избыточность – введение дополнительной информации на уровне исходных данных, промежуточных данных, которые позволяют в случае отказа пересчитывать основные данные за счет дополнительных.

Функциональная избыточность – в системе реализуется избыточное число функций, которое позволяет исправить возникшие неисправности (или их часть)

Информационная, функциональная и структурная избыточности **не** возможны без **временной** избыточности.

Трн + Δt = Трр, где Трн – время решения

Δt – время на проверки (избыточность)

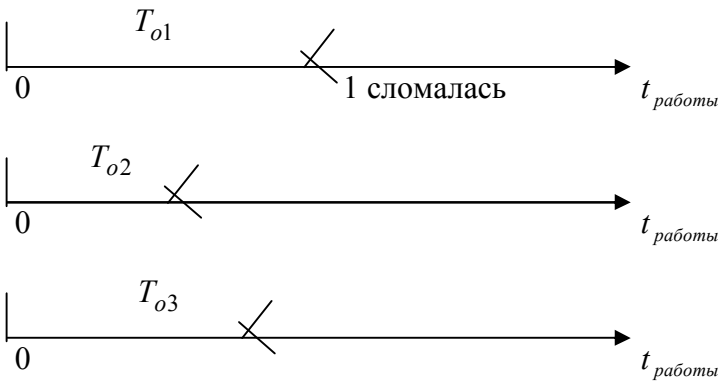
Трр – реальное время решения

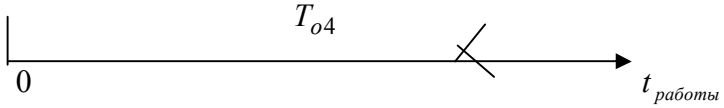
Если Δt = Трн, то это – двойное решение. Обычно Δt = 30%Трн.

Запас времени необходим для введения резерва в действие.

1.2 Показатели наджности невосстанавливаемых элементов

Время работы 4-х изделий (сбор статистики):

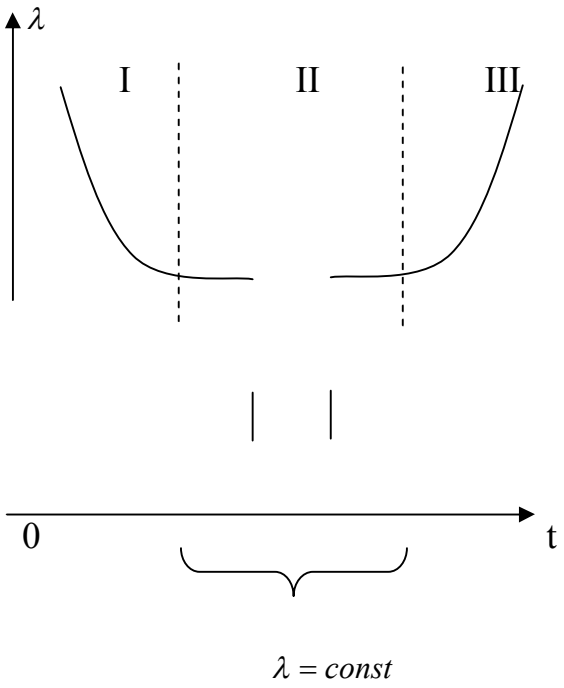




$T_o = \frac{\sum_{i=1}^{N} T_{oi}}{N}$ - среднее время безотказного функционирования или наработка на отказ (есть справочники для всех изделий)

$\lambda = \frac{1}{T_o}$ - интенсивность отказа, в общем случае $\lambda = \lambda(t)$ - функция от времени жизни изделия

Для всех изделий функция λ одинакова:



I – область приработки – проявляются конструктивные и производственные отказы. Этот этап должен быть коротким и обычно проходит на заводах при испытаниях.

II – нормальное функционирование. λ является практически константой. Здесь проявляются в основном случайные (внезапные) отказы. (Соответствует формуле $\lambda = \frac{1}{T_o} = const$).

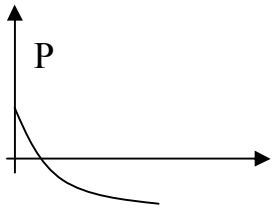
III – проявляются постепенные отказы, связанные с физическим износом изделия. Как правило, на этом этапе экономически невыгодно эксплуатировать.

I и II интервалы могут быть рассмотрены как наработка.

Для всех радиоэлектронных изделий T_{oi} распределено по экспоненциальному закону.

$P_{безотказ.функционирования}(t) = e^{-\lambda t}$ - вероятность безотказной работы на интервале [0;t]

Рассматриваем II зону, где $\lambda = const$:



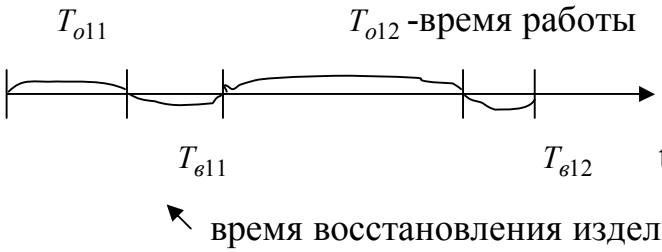
$Q(t) = 1 - P(t) = 1 - e^{-\lambda t}$ - вероятность отказа на интервале [0;t]

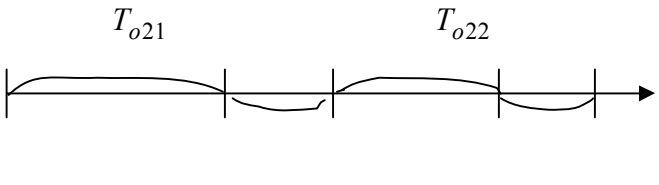
$\exists$ сбои, поэтому вводят $T_{0сбоя}$ – среднее время между сбоями – и $\lambda_{сбоя}$ – интенсивность сбоев. $T_{0сбоя}$ на приблизительно 2 порядка меньше T_o.

P(t) не зависит от предыстории, т.е. от того, сколько до момента 0 изделие проработало без отказа.

1.3 Показатели надежности восстанавливаемых изделий

Работа 2-х одинаковых изделий:

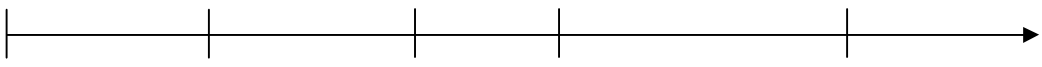




$\overline{T}_o = \frac{\sum_{i=1}^{l}\sum_{j=1}^{k} T_{oij}}{M}$ - среднее время между отказами (M – общее число интервалов функционирования). Это верно при $\overline{T}_в << \overline{T}_o$, где $\overline{T}_в = \frac{\sum_{i=1}^{j}\sum_{j=1}^{k} T_{вij}}{M}$

Можно найти среднее время восстановления ($\overline{T}_в$), но только если восстанавливаем в одном месте (один орган).

обнаружение отказа — время локализации — время ремонта — время информац. восстан.



отказ

$T_в = T_{обн} + T_{лок} + T_{рем} + T_{инф.восст.}$

$T_{обн}$ - случайная величина, зависящая от средств обнаружения контроля

$T_{лок}$ - случайная величина, зависящая от квалификации ремонтника

$T_{рем}$ - случайная величина, зависящая от квалификации ремонтника и полноты ЗИПа

$T_{инф.восст.}$ - тоже случайная величина

$\lambda = \frac{1}{\overline{T}_o}$ - интенсивность отказа, тоже $\lambda = \lambda(t)$

$\mu = \frac{1}{\overline{T}_в}$ - параметр восстановления, $\overline{T}_в$ тоже распределено по экспоненциальному закону.

$P(t) = e^{-\lambda t}$ - вероятность исправной работы на интервале [0;t]

$Q(t) = 1 - e^{-\lambda t}$ - вероятность отказа на интервале [0;t]

$P_в(t) = e^{-\mu \cdot t}$ - вероятность восстановления на интервале [0;t]

$Q_в(t) = 1 - e^{-\mu \cdot t}$ - вероятность невосстановления на интервале [0;t]

Комплексные параметры:

а) $k_г$ **- коэффициент готовности** – это вероятость того, что в произвольный момент времени изделие будет работоспособным (точечное определение)

$k_г$ - это какую долю единичного интервала времени изделие будет в работоспособном состоянии (интервальное определение)

$k_г = \frac{\overline{T}_o}{\overline{T}_o + \overline{T}_в} = \frac{\mu}{\mu + \lambda}$ - интервальная оценка стационарного коэф-та готовности

б) $k_{пр} = 1 - k_г = \frac{\lambda}{\mu + \lambda}$ **- коэффициент простоя**

в) $k_{ог} = k_г \cdot P(t)$ **- коэффициент оперативной готовности** – это вероятность того, что в произвольный момент времени изделие работает и проработает без отказа на интервале от 0 до t.

Таблица $k_г$:

утюг - $k_г$ = 0,999999

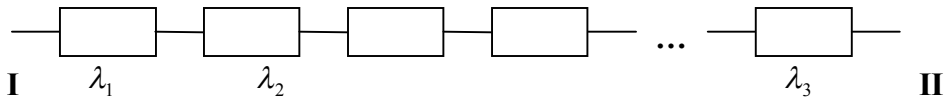
АТС - $k_г$ = 0,9

Цифровые телефонные станции- $k_г = 0{,}98$

1.4 Способы соединения элементов в теории надежности

Существует понятие структурной схемы, из нее необходимо сделать структурно-надежную схему, которая определяет надежность функционирования во времени.

Последовательное соединение (безизбыточная система):



I,II – Полюса

Характеризуется тем, что отказ для любого элемента приводит к отказу системы.

λ_i - интенсивность отказов i-шош элемента.

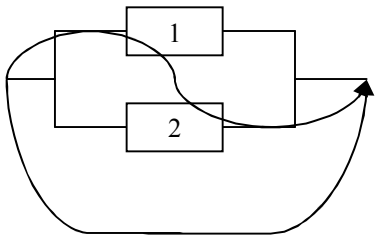
$\Lambda_c = \sum_{i=1}^{m} \lambda_i$ - интенсивность отказа системы

$T_0 = \frac{1}{\Lambda_c}$ - наработка на отказ системы

$P_c = \prod_{i=1}^{n} P_i$ - вероятность безотказного функционирования системы, если элементы независимы

$K_r = \prod_{i=1}^{n} K_i$ - если число восстановительных органов достаточно, т.е. как только отказ появляется, за него сразу «берутся».

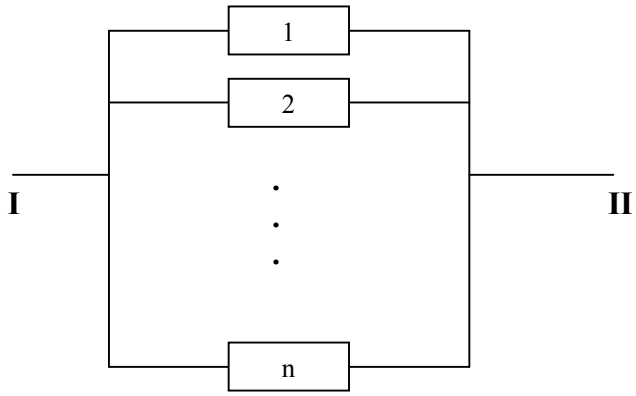
Параллельное соединение (избыточная система):



Горячий резерв.

Дублированная пара

В общем случае:



Для дублированной пары:

Q = Q1*Q2 – вероятность отказа

P = 1-Q = 1-(1-P1)(1-P2) – вероятность безотказного функционирования

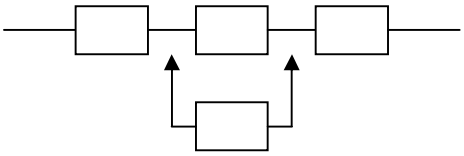
Kr = (1-(1-Kr1)(1-Kr2)) – коэффициент готовности в случае неограниченных возможностей по восстановлению.

Про λ и T нельзя ничего сказать.

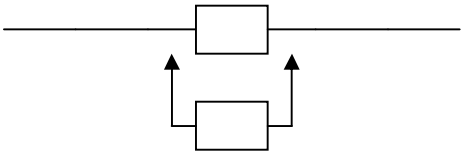
$$P_c = 1 - \prod_{i=1}^{n} (1 - P_i)$$

Говорим о кратности резервирования. Если n элементов, то (n-1) кратное резервирование.

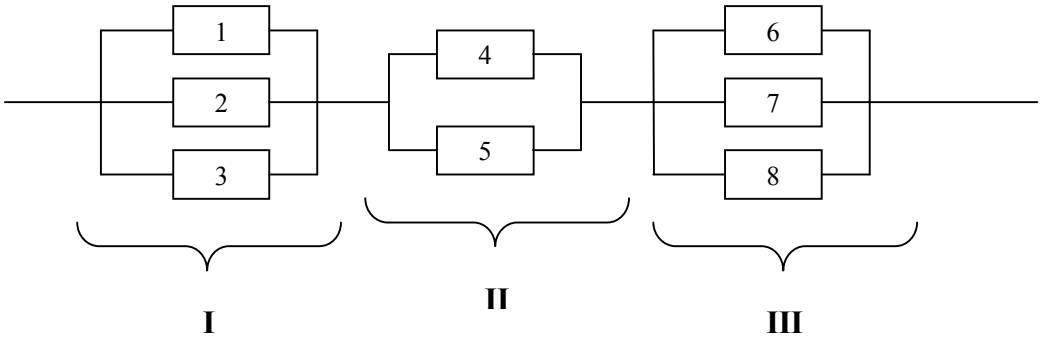
Скользящий резерв:



Холодный резерв:



Параллельно-Последовательное соединение:



I,**II**,**III** – фрагменты с чисто параллельными элементами

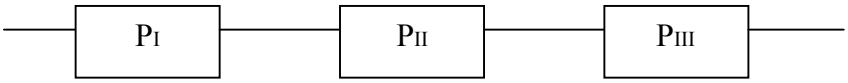
$P_I = 1 - \prod_{i=1}^{n} Q_i$ -безотказное функционирование фрагмента **I**

Qi = 1 – Pi, i = 1,2,3 – Вероятность отказа i-ого звена

$P_{II} = 1 - \prod_{i=4}^{5} Q_i$ i=4,5

$P_{III} = 1 - \prod_{i=6}^{8} Q_i$ i=6,7,8

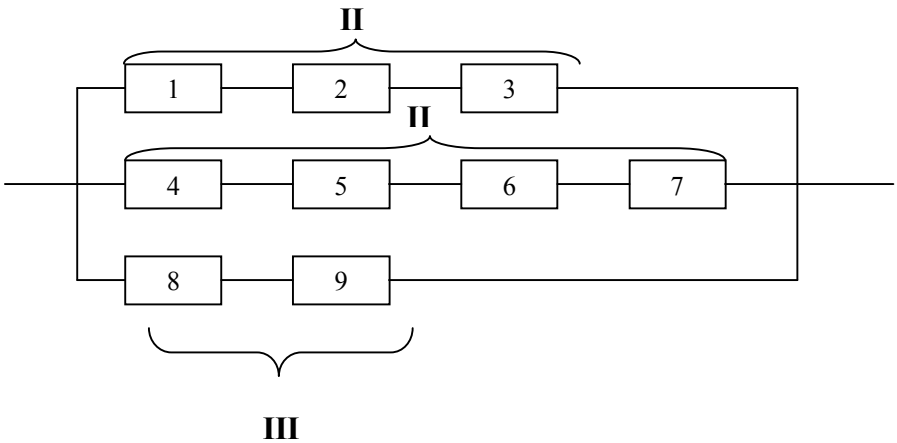
Переходим к структуре:



Pc = PI PII PIII – Вероятность безотказного функционирования системы

$$= (1 - \prod_{i=1}^{3}(1 - P_i))(1 - \prod_{i=4}^{5}(1 - P_i))(1 - \prod_{i=6}^{8}(1 - P_i))$$

Последовательно-параллельная структура

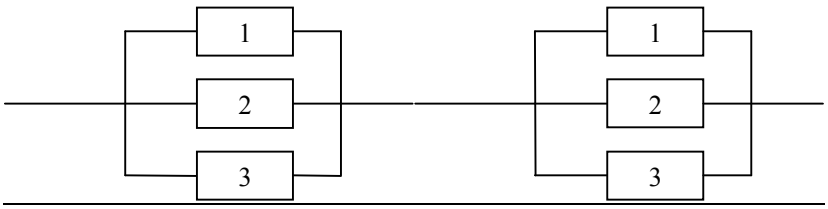


$$P_I = P_1 \cdot P_2 \cdot P_3 = \prod_{i=1}^{3} P_i$$

$$P_{II} = P_4 \cdot P_5 \cdot P_6 \cdot P_7 = \prod_{i=4}^{7} P_i$$

$$P_{III} = \prod_{i=8}^{9} P_i$$

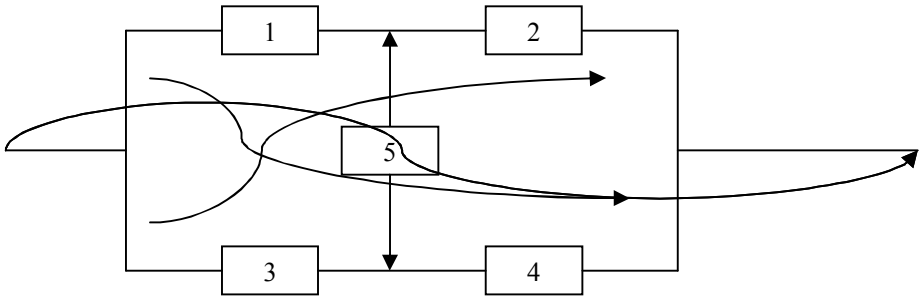
Перейдем к структуре:



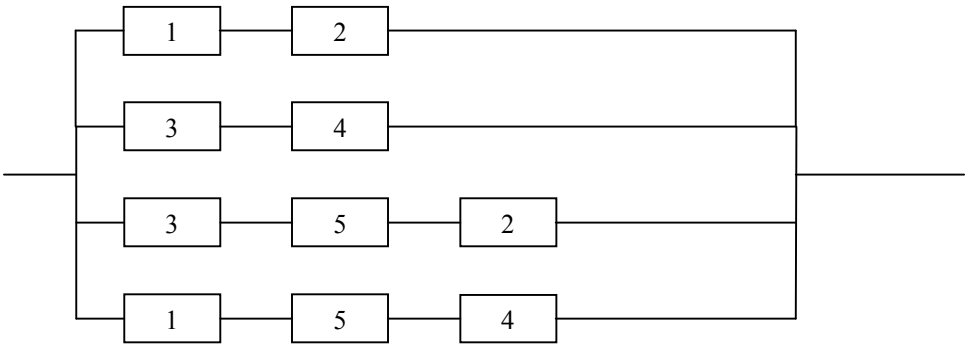
$P_c = 1 - (1 - \prod_{i=1}^{3} P_i)(1 - \prod_{i=4}^{7} P_i)(1 - \prod_{i=8}^{9} P_i)$ - вероятность безотказной работы всей системы

Пример схемы, не сводящейся к 4ем классическим соединениями.

Схема «мостик» -структурно-сложная схема.



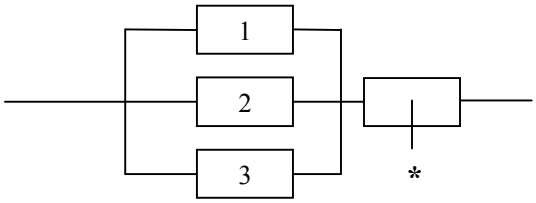
Попытаемся перейти к последовательно-параллельной схеме:



Однако здесь присутствуют зависимые элементы!!!

Это не классическое паралелльное соединение. В классике все элементы независимы.

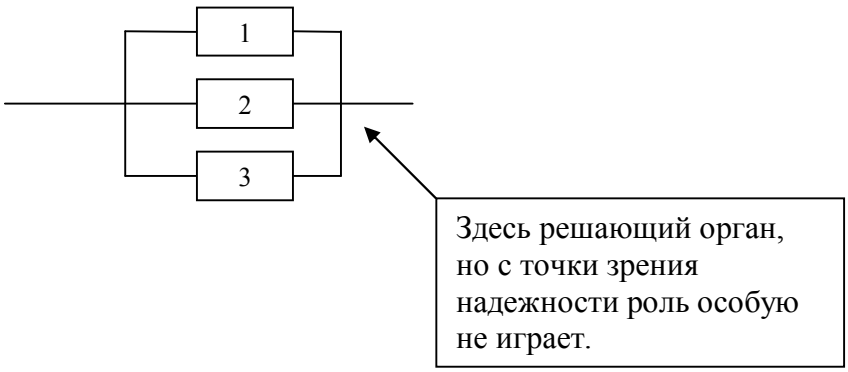
Мажоритарная схема:



* - решающий орган для цветов. Его можно убрать.

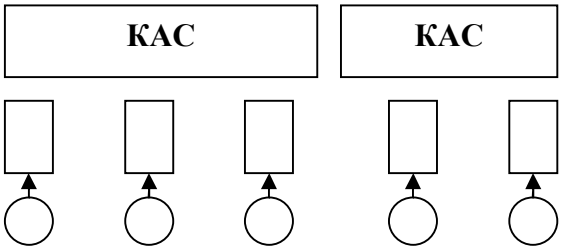
Мажоритарная система:

! Но на выходе – сигнал по принципу 2 из 3х.(результат совпадает)



Это тоже не сводится к классическому параллельному соединению.

Нужны свои методы расчета.

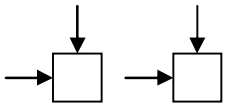


1.5 Структуры аппаратно-программных комплексов (АПК)

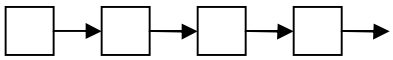
Будем оценивать надежность по структурному критерию.

Классические структуры:

1) Возьмем МКМД (Много команд много данных). Несколько вариантов машин работают автономно.

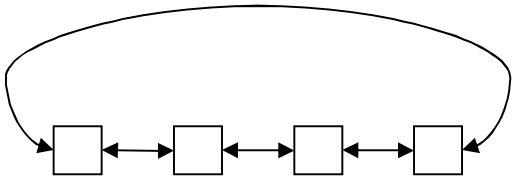


2) Далее идет конвейерный принцип.

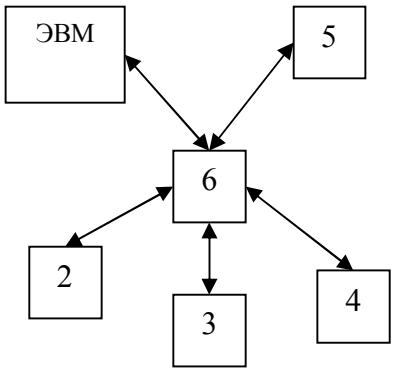
 - Конвейерная структура многопрофильной системы

3) Векторная структура

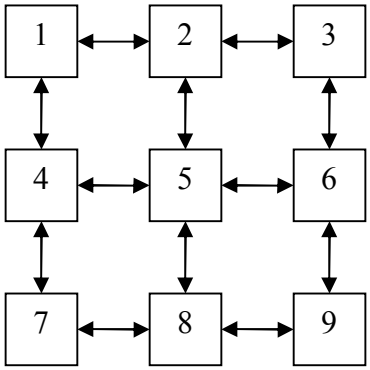
4) Кольцо



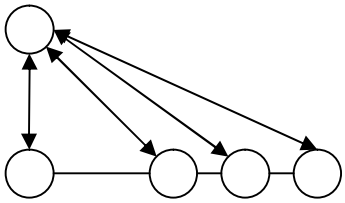
5) Звезда

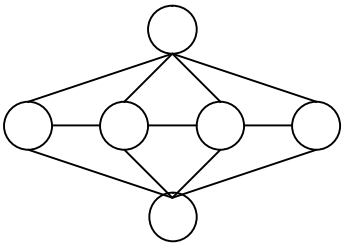


6) Двумерная решетка



7) Рекуррентные структура (по Ушакову)

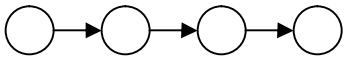




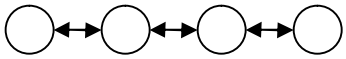
В надежности в качестве модели исследуется графовая модель.

Существует понятие ориентированных и не ориентированных графов

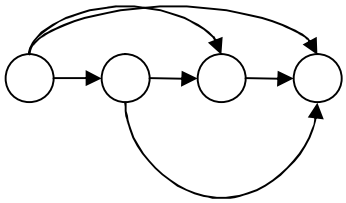
1) Односвязный ориентированный граф = конвейер



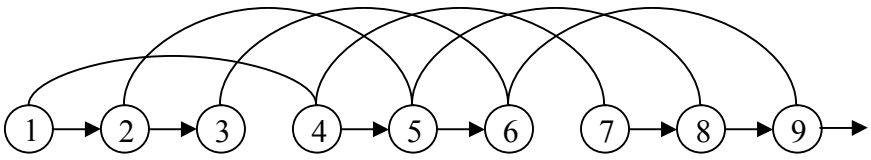
2) Неориентированный граф = вектор



3) Полносвязный ориентированный граф (каждый с каждым)



Нарисуем двумерную решетку в графовой модели:



Это многосвязный граф. Т.о. теория графов (графовые модели) является заменой структурных представлений.

Если рассмотреть сети, то их нельзя просто взять графовой моделью!

Любую структуру можно представить графом.

Рассмотрим матрицу связности:

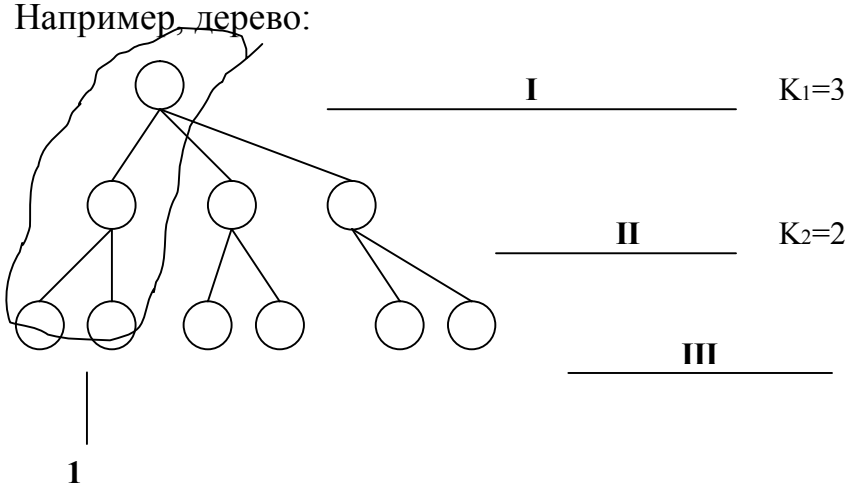
$$\begin{pmatrix} P_{11} & P_{12} & P_{13} \\ P_{21} & P_{22} & P_{23} \\ P_{31} & P_{32} & P_{33} \end{pmatrix}$$

Т.е. сети нельзя оценить показателем надежности – используют 2х мерную матрицу связности.

Для 2х точек сети можно построить графовую модель, а для всей сети – сложно.

Для иерархической среды также будем рассматривать графовую модель:

Например, дерево:



Здесь существуют определенные уровни иерархии. Уровни иерархии бывают изотропные и не изотропные.

Чтобы система была изотропная, показатели надежности должны быть одинаковые у элементов одного уровня.

У каждого уровня существует коэффициент ветвления.

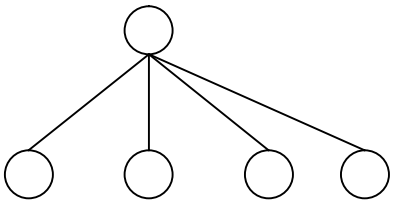
Чтобы система была изотропная, коэффициенты ветвления должны быть одинаковы на каждом уровне (у элементов).

Система 1 представляется двухполюсным графом. Структура может быть от односвязного до полносвязного.

Если такая структура односвязная, то говорят, что это дерево.

Чтобы система была изотропная, структура всех ветвей должна быть одинаковая. Во всех остальных ситуациях система не изотропная.

Звезда так представляется (частный случай изотропной иерархической структуры).



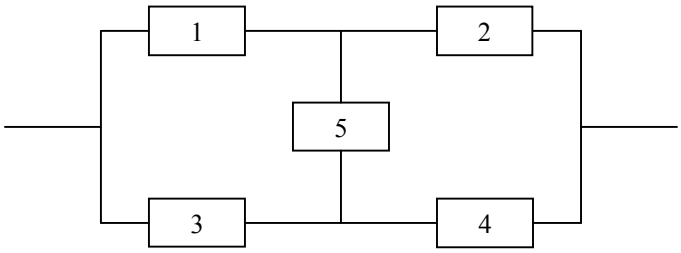
Это не классическая структура. Иерархические структуры в случае введения в них информационной избыточности к двухполюсникам не сводятся.

Глава 2. Методы оценки показателей надежности.

2.1 Метод исключения особого элемента (используется для оценки структурных сложных систем).

Идея: Путем исключения из схемы системы одного или двух элементов желательно привести эту систему к одному из известных классических способов соединения элементов.

Например, рассмотрим мостиковую структуру:



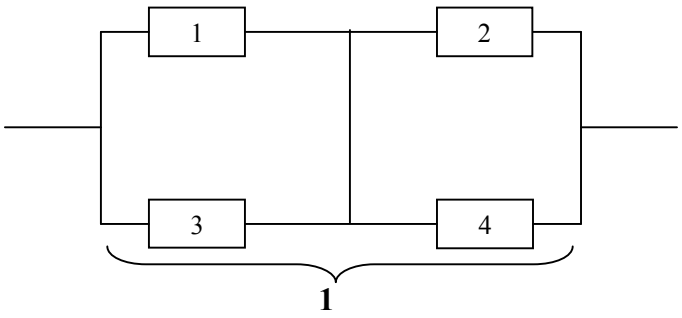
Классически оценить не можем. 5-особый элемент. Его будем убирать.

Чтобы убрать элемент из рассмотрения, рассмотрим многообразие его состояний:

М.б. 2 состояние:

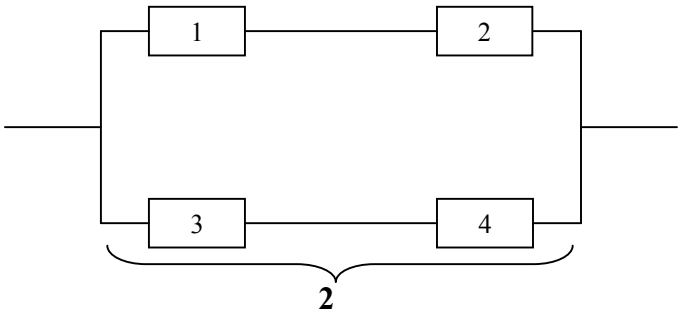
- Работоспособен - вероятность Pi
- Неработоспособен – вероятность Qi

Если он работоспособен, то этот элемент рассматривается, как короткое замыкание в системе, а если нет, то соединение разрывается. Это значит, что при некой вероятности Pi структурно надежности схема принимает вид:



Это классическая структура. Параллельно-последовательная структура.

Если неработоспоспособная с вероятностью Qi:



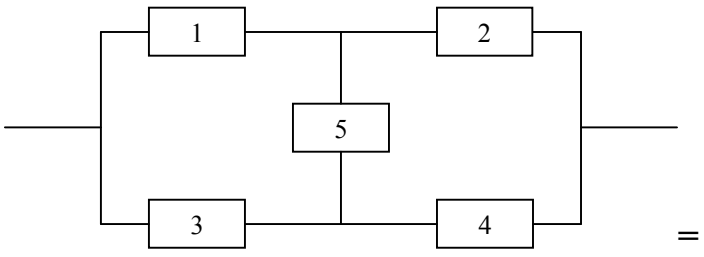
Это последовательно-параллельная структура.

Можно записать формулы:

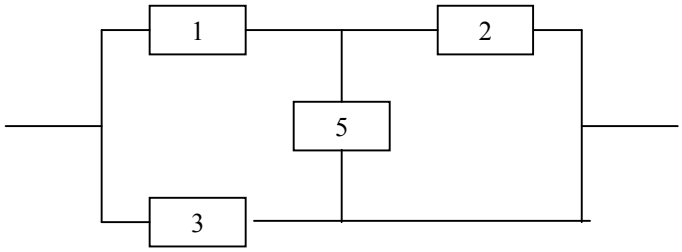
Pc = P5 (1-Q1Q3)(1-Q2Q4)+Q5(1-P1P2)(1-P3P4)

Здесь 5ый элемент особый. P5 – особый элемент работоспособен, Q5 – особый элемент неработоспособен.

Выберем особым элементом 4ый:

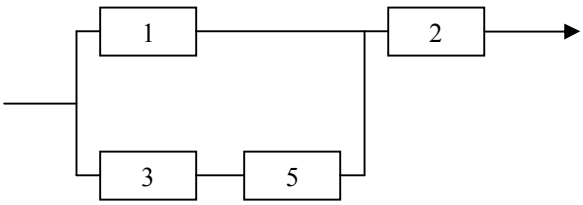


=

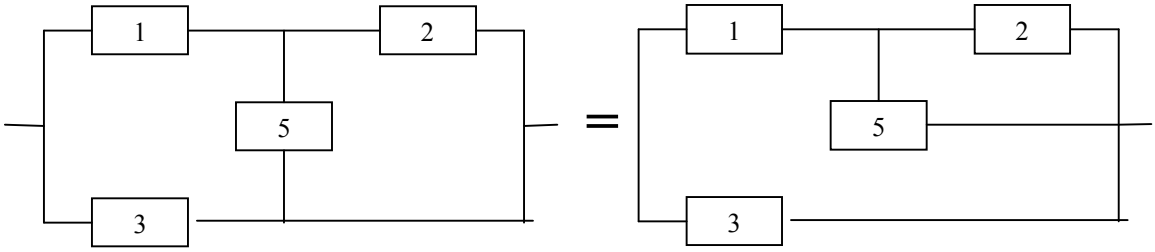


P4 *

+Q4 *

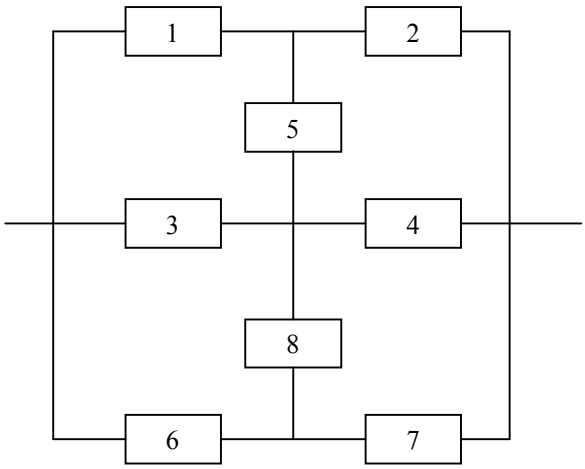


Здесь:



Данный метод расчета может применятся не один раз, т.е. на разных этапах расчета выбирать разные особые элементы или можно брать группу особых элементов.

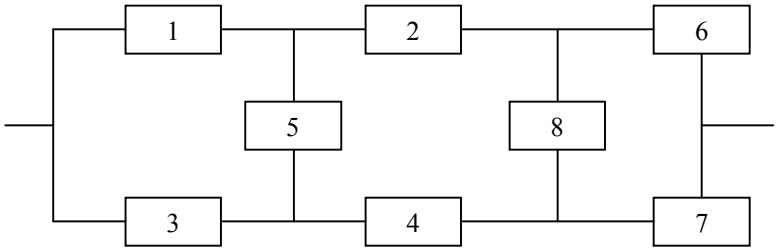
Рассмотрим систему:



Здесь одним особым элементом не обойтись. Как минимум двумя: 5 и 8.

4 состояния: $\underset{I}{P_5P_8}$ $\underset{II}{P_5Q_8}$ $\underset{II}{Q_5P_8}$ $\underset{IV}{Q_5Q_8}$

Тогда формула запишется: **P5P8** (Схема I.Оба элемента работоспособны) + **P5Q8** (Схема II)+**Q5P8**(схема III.8ой работоспособен. 5ый - нет)+**Q5Q8**(схема IV. Оба неработоспособны).

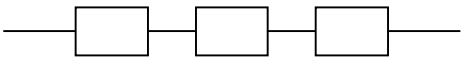


Здесь тоже два особых элемента.

Этот метод универсальный.

2.2 Метод полного перебора состояний системы.

Применим всегда. Не всегда реализуется из-за вычислительной сложности. Здесь должно быть известно все множество возможных состояний системы, которое будет. И для каждого состояния необходимо посчитать вероятность пребывания системы в нем.



3 элемента. У каждого может быть два состояние – работоспособно и нет.

Итого 8 состояний системы.

Запишем все 8 состояний:

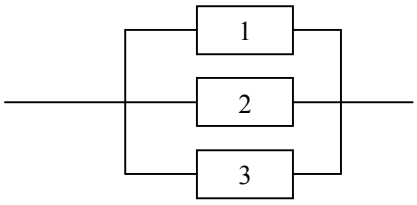
1. P1P2P3 – работоспособное изделие
2. P1P2Q3 – нет
3. P1Q2P3 – нет
4. Q1P2P3 – нет
5. P1Q2Q3 – нет
6. Q1P2Q3 – нет
7. Q1Q2P3 – нет
8. Q1Q2Q3 – нет

Pc= P1P2P3 (сумма всех работоспособных состояний)

Q= P1P2Q3 +…+ Q1Q2Q3 = 1 - P1P2P3

Этот метод на все случаи жизни.

Пусть есть еще одна система:



Многообразие состояний то же самое, но только одно состояние не работоспособно: Q1Q2Q3, все остальные работоспособны.

Итак: Pc = P1P2P3 + P1P2Q3 + P1Q2P3 + … + Q1Q2P3 = 1- Q1Q2Q3

Qc = Q1Q2Q3

Если каждый элемент имеет 2 состояния, то количество состояний 2^n, где *n*-количество элементов. Метод универсален, но потребует очень много ресурсов.

Рассмотрим, как строится приближенное решение:

Метод приближенной оценки

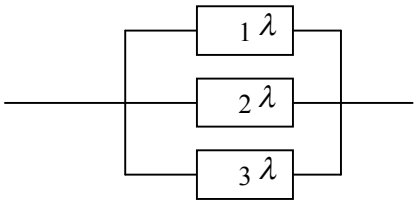
Qc + Pc = 1

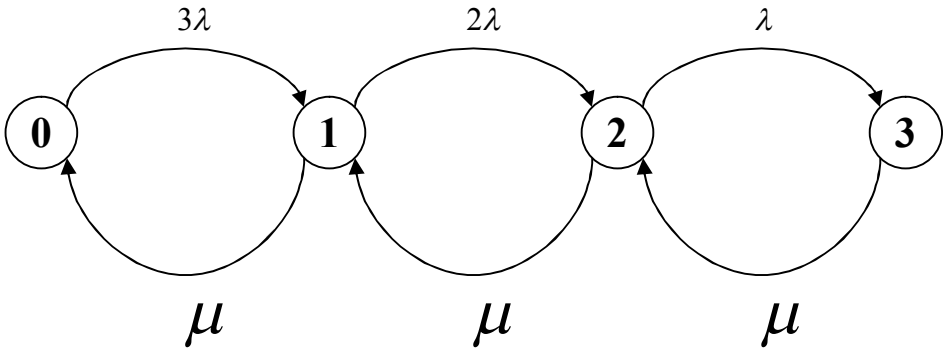
$1 - P_c - Q_c = \Delta$ - показывает ошибку, которую имеет в процессе работы. Эта величина должна уменьшаться по мере приближения Pc и Qc к реальным величинам.

2.3 Метод дифференциальных уравнений.

Взят из теории массового обслуживания. Работает, если времена наработки на отказ и время на отказ распределены экспоненциально. Применим для восстанавливаемых и невосстанавливаемых систем.

В качестве модели используется граф состояний такой, что вероятность появления двух событий была мала. Тогда задача сводится к решению дифференциальных уравнений Колмогорова.





В кружке количество отказавших элементов.

μ -параметр восстановления

$$\lambda = \frac{1}{T_0}$$

$$\mu = \frac{1}{T_B}$$

Система уравнений Колмогорова

$$\begin{cases} \dfrac{dP_0}{dt} = -3\lambda P_0 + \mu P_1 \\ \dfrac{dP_1}{dt} = 3\lambda P_0 - (2\lambda + \mu)P_1 + \mu P_2 \\ \dfrac{dP_2}{dt} = 2\lambda P_1 - (\mu + \lambda)P_2 + \mu P_3 \\ \dfrac{dP_3}{dt} = - \end{cases}$$

Четвертое уравнение выкидываем и заменяем на нормировочное уравнение.

P0+P1+P2+P3=1

Решив, найдем:

P0(t),P1(t),P2(t) и P3(t)

Начальные условия:

$P_0(0)$ =1

$P_1(0)$ =0 - не работают

$P_2(0) = 0$ (0 неисправных элементов)

$P_3(0) = 0$

$P_C(t) = P_0(t) + P_1(t) + P_2(t)$ - вероятность работоспособности системы

$Q_C(t) = P_3(t)$ - вероятность неработоспособности системы

Этапы:

1. Граф
2. Система Колмогорова
3. Решение
4. Выводы работоспособных состояний
5. Запись решения

Если система восстанавливаема, то часто не интересует динамика, а интересует $K_Г$.

Система превращается в алгебраическую

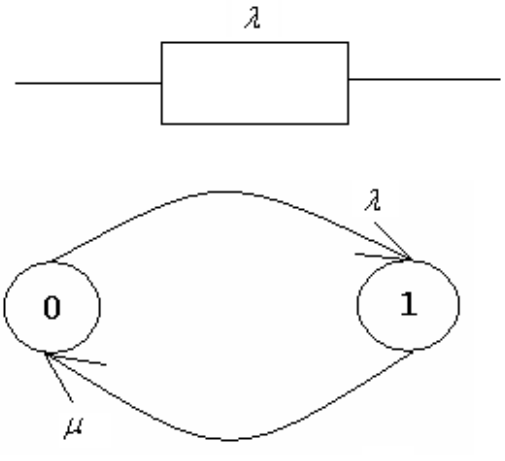
$$\frac{d\,P_i}{dt} = 0$$

Наше решение будет найдено $K_Г$

$K_{Гс} = K_{Г0} + K_{Г1} + K_{Г2}$

(системы)

Рассмотрим систему из одного элемента:



$$\frac{d P_0}{dt} = -\lambda \cdot P_0 + \mu \cdot P_1 \qquad P_0 + P_1 = 1$$
$$\frac{d P_1}{dt} = \lambda \cdot P_0 - \mu \cdot P_1$$

Запишем алгебраические уравнения:

$$0 = -\lambda \cdot P_2 + \mu \cdot P_1$$
$$P_0 = 1 - P_1$$
$$P_1 = 1 - P_0$$
$$0 = -\lambda \cdot P_0 + \mu \cdot (1 - P_0)$$
$$P_0 \cdot (\lambda + \mu) = \mu$$
$$P_0 = \frac{\mu}{\lambda + \mu} = K_Г$$
$$P_1 = \frac{\lambda}{\lambda + \mu} = K_П$$

Позволяет находить средние времена пребывания в состоянии.

Берём уравнения Колмогорова, оставив только уравнение для работоспособных состояний.

Вместо $\frac{d P_i}{dt}$, где i – начальное состояние системы, записываем -1.

Вместо $\frac{d P_j}{dt}$ $(j \neq i)$ записываем 0.

Все P_i заменяем на T_i.

Разрешаем систему относительно T_i. Это будут средние времена пребывания в состоянии.

Если

0-ое состояние работоспособно

0-ое начальное состояние

$$-1 = -\lambda \cdot P_0$$

Заменяем P_0 на T_0

$$-1 = -\lambda \cdot T_0$$

$$T_0 = \frac{1}{\lambda}$$

Если

1-ое состояние работоспособно

1-ое начальное состояние

$$-1 = -\mu \cdot p_1$$

$$T_1 = \frac{1}{\mu}$$

Метод работает при малой размерности системы или регулярной структуре.

2.4 **Логико-вероятностные методы. Математическая постановка задачи.**

В этих методах элемент

может находиться только в 2-х состояниях:

- работоспособном
- неработоспособном

Каждому элементу структуры системы ставится в соответствие логическая переменная x_i

$$x_i = \begin{cases} 1 & - \textit{работоспособен} \\ 0 & - \textit{неработоспособен} \end{cases}$$

$$P\left(x_i = 1\right) = P_i \quad - \textit{вероятность} \quad \textit{работоспособности}$$

$$P\left(x_i = 0\right) = P\left(\overline{x_i} = 1\right) = 1 - P_i = Q_i \quad - \textit{вероятность} \quad \textit{отказа}$$

Исходя из условий функционирования системы для неё могут записываться сложные логические функции, описывающие условия нормального (безотказного) функционирования –

Функции работоспособности (ФР) и

Функции неработоспособности (ФНР)

$\Phi P = \overline{\Phi HP}$ (с инверсией)

$$\left.\begin{matrix} \Phi P \\ \Phi HP \end{matrix}\right\} \quad \textit{ФАЛ} \; - \; \textit{Функции} \;\; \textit{алгебры} \;\; \textit{логики}$$

$$\Phi AЛ = f\left(x_1, x_2, \ldots, x_n\right) = \begin{cases} 1 \\ 0 \end{cases}$$

(n – размерность системы)

$P(f_{ФР}=1) = P_C$ – вероятность безотказного функционирования системы

$P(f_{ФР}=0) = 1-P_C = Q_C$ – вероятность отказа системы

$P(f_{ФНР}=1) = Q_C$

$P(f_{ФНР}=0) = 1-Q_C = P_C$

$f_{ФР} = \overline{f_{ФНР}}$

ФАЛ –> $P(P_i)$, где P_i – вероятностный полином. Связывает P_C с P_i.

(задаёт)

Этапы решения:

1. По схеме формируются ФАЛ.
2. ФАЛ преобразуется к одному из видов, где допускается замещение логических переменных вероятностными показателями.
3. Замещение.
4. Получаем вероятностный полином.

Модель имеет существенные ограничения, но область применимости гораздо шире остальных методов.

2.5 Методы записи функций алгебры логики .

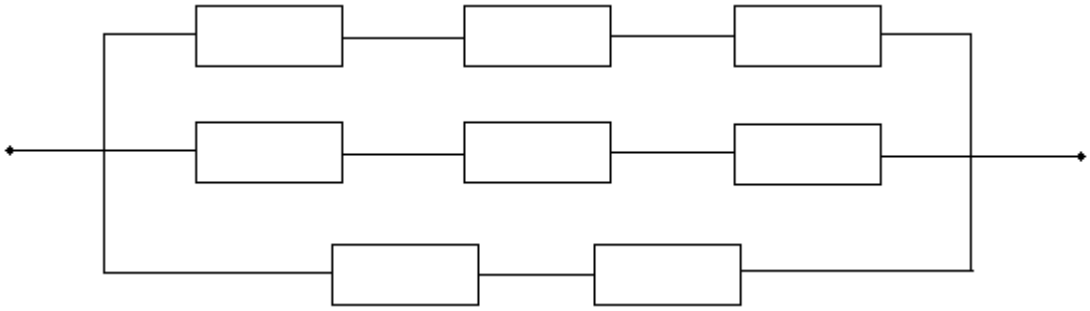
Существует 3 подхода:

a) Метод минимальных путей.
b) Метод минимальных сечений.
c) Метод записи ФАЛ в виде системы логических уравнений.

2.5.1 Метод минимальных путей.

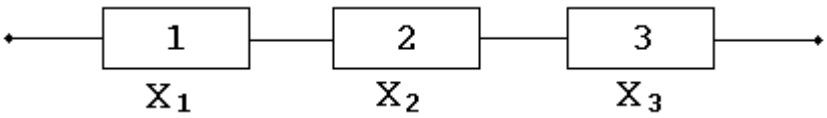
Минимальный путь – некая совокупность элементов системы, отказ любого элемента из которой приведёт к отказу системы, если все прочие элементы системы (вне совокупности) находятся в отказавшем состоянии.
Это последовательное соединение элементов.

Перечисление минимальных путей задаёт структурно-надёжностную схему:



Примеры:

1)

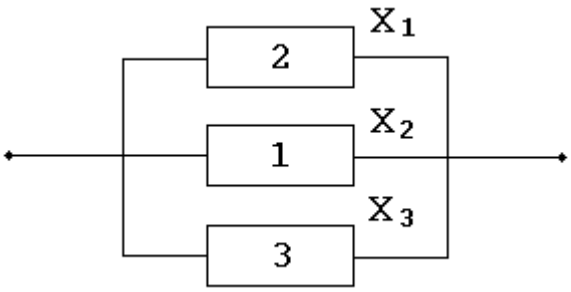


В этой системе только один минимальный путь.

$f = x_1 \& x_2 \& x_3$

Любой минимальный путь – конъюнкция логических переменных, соответствующих элементам, лежащим на пути.

2)



1-ый путь - элемент 1 - x_1

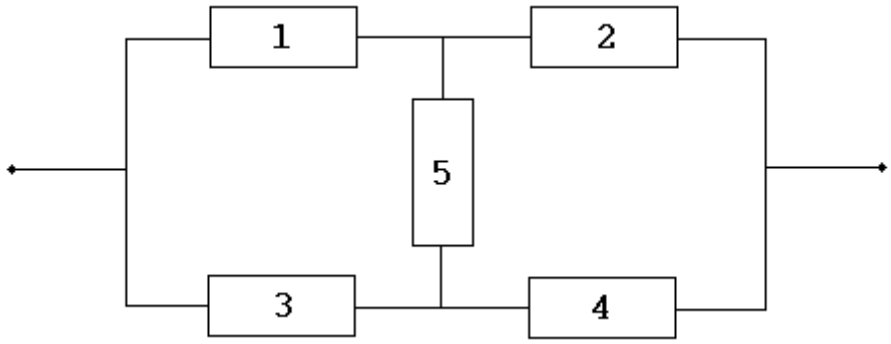
2-ый путь - элемент 2 - x_2

3-ий путь - элемент 3 - x_3

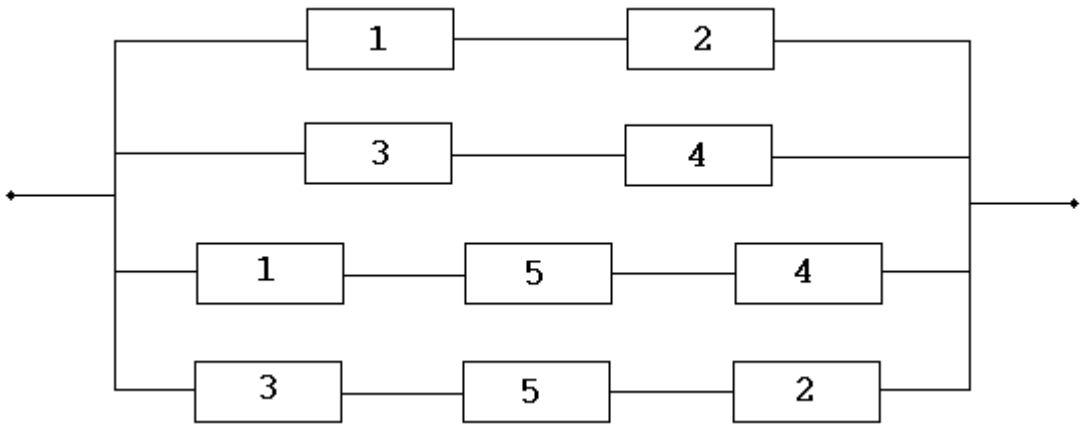
$f = x_1 \vee x_2 \vee x_3$

3) Мостик

- структурная схема



- структурно-надёжностная схема (пути)



Запишем:

$x_1 \cdot x_2 \vee x_3 \cdot x_4 \vee x_1 \cdot x_5 \cdot x_4 \vee x_3 \cdot x_5 \cdot x_2$

Система работоспособна если работает хотя бы один путь.

$$f = x_1 \cdot x_2 \vee x_2 \cdot x_4 \vee x_1 \cdot x_5 \cdot x_4 \vee x_3 \cdot x_5 \cdot x_2$$

ФАЛ в общем виде:

$$f = \bigvee_{l \in I_0} \; \& \; x_i$$

где: $\bigvee_{l \in I_0}$ – по всем минимальным путям

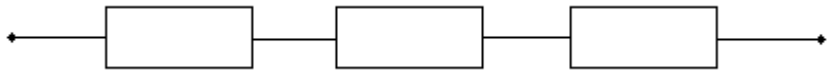
$\&$ – по всем i принадлежащему l-тому минимальному пути.

I_0 – множество минимальных путей системы.

2.5.2 **Метод минимальных сечений.**

Минимальное сечение – совокупность элементов системы, переход любого элемента которой из неработоспособного состояния в работоспособное приведёт к переходу системы в работоспособное состояние, если остальные элементы (вне совокупности) работоспособны.

Подход аналогичен с точностью до наоборот.

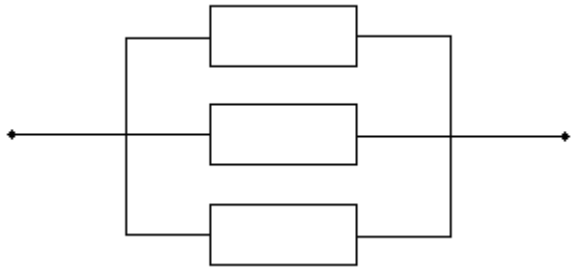


- три сечения

Получим: $f = \& \vee x_i$

где: & – по сечению

x_i – по элементам сечения



- одно сечение

Для получения точных ФАЛ нужно перебрать всё многообразие возможных путей и сечений.

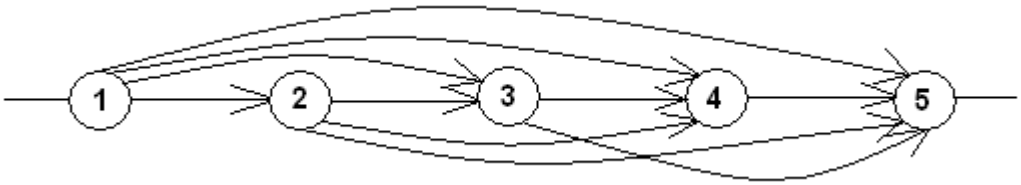
Если при формировании ФАЛ по путям перебрали не все пути, ФАЛ будет приближенно вероятностным показателем. (P_C будет заниженным)

Если при формировании ФАЛ по сечениям перебрать не всё многообразие сечений, то вероятность P_C будет завышенной.

2.5.3 Методы построения ФАЛ с помощью системы логических уравнений.

Запись ФАЛ в виде системы логических уравнений есть разновидность метода построения по путям, то процесс формализуется.

Полносвязный граф:



$$f = x_1\left(x_{12}\cdot f_2 \vee x_{13}\cdot f_3 \vee x_{14}\cdot f_4 \vee x_{15}\cdot f_5\right)$$

$$f_2 = x_2\left(x_{23}\cdot f_3 \vee x_{24}\cdot f_4 \vee x_{25}\cdot f_5\right)$$

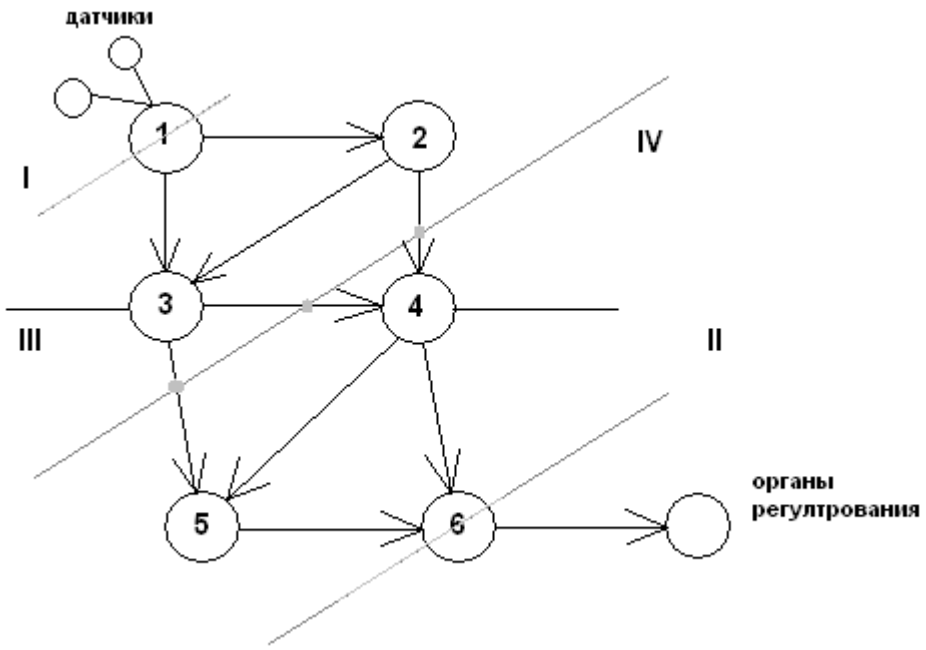
$$f_3 = x_3\left(x_{34}\cdot f_4 \vee x_{35}\cdot f_5\right)$$

$$f_4 = x_4\left(x_{45}\cdot f_5\right)$$

$$f_5 = x_5$$

Узлы и дуги – элементы графа.

- структурно-надёжностная схема



x_i , $i = 1..6$

$P(x_i=1) = P_i$ - вероятность функционирования узла

x_{ij}

$P(x_{ij}=1)=P_{ij}$ - вероятность функционирования связи

I. подход Составим функцию работоспособности системы:

$$f = x_1(x_{12} \cdot f_2 \vee x_{13} \cdot f_3)$$

f_2 *и* f_3 - это логические функции, которые дальше будут отображать условия безотказного функционирования, если за входной полюс взяты 2-й или 3-ий узел.

$$f_2 = x_2(x_{23} \cdot f_3 \vee x_{24} \cdot f_4)$$

$$f_3 = x_3(x_{34} \cdot f_4 \vee x_{35} \cdot f_5)$$

$$f_4 = x_4(x_{45} \cdot f_5 \vee x_{46} \cdot f_6)$$

$$f_5 = x_5 \cdot x_{56} \cdot f_6$$

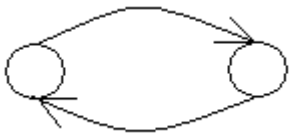
$$f_6 = x_6$$

Функция работоспособности для логико-вероятностного метода должна быть получена в явном виде, как система нам не подходит. В том случае, если:

- связи ориентированные - логико-вероятностный метод работает всегда

- связи частично-ориентированные - логико-вероятностный метод работает практически всегда

- связи неориентированные – никто не гарантирует точного решения

м.б. раздвоение: 

Наиболее универсальным способом решения систем логических уравнений является метод подстановок, т.е. находим f_6 из последнего уравнения, потом f_5 и т.д. последовательно.

$$f_4 = x_4(x_{45} \cdot x_5 \cdot x_{56} \cdot x_6 \vee x_{46} \cdot x_6)$$

$$f_3 = x_3((x_{34} \cdot x_4(x_{45} \cdot x_5 \cdot x_{56} \cdot x_6 \vee x_{46} \cdot x_6)) \vee x_{35} \cdot x_5 \cdot x_{56} \cdot x_6)$$

т.о.

$$f = x_1(x_{12} \cdot x_2 \cdot (x_{23} \cdot x_3 \cdot (x_{34} \cdot x_4 \cdot (x_{45} \cdot x_{56} \cdot x_6 \vee x_{46} \cdot x_6)) \vee x_{35} \cdot x_5 \cdot x_{56} \cdot x_6)) \vee$$
$$\vee x_{24} \cdot x_4 \cdot (x_{45} \cdot x_5 \cdot x_{56} \cdot x_6 \vee x_{46} \cdot x_6) \vee x_{13} \cdot (x_3 \cdot (x_{34} \cdot x_4 \cdot (x_{45} \cdot x_5 \cdot x_{56} \cdot x_6 \vee x_{46} \cdot x_6)) \vee x_{35} \cdot x_5 \cdot x_{56} \cdot x_6)$$

1. Работаем по слоям.
2. Процесс формализуется.
3. ФАЛ получается по путям, но в скобочной форме записи –> имеем минимальную форму записи.

ФАЛ – функции алгебры логики.

II. подход

Запишем систему логических уравнений, когда за исходный полюс (начальный) берётся выходной полюс системы.

Итак:

$f = x_6(x_{46} \cdot f_4 \vee x_{56} \cdot f_5)$ *появились* f_4, f_5

Логические функции f_4, f_5 – это некое логическое выражение, описывающее безотказную работу системы если выходной полином является любо 4-й либо 5-й.

$$f_4 = x_4(x_{24} \cdot f_2 \vee x_{34} \cdot f_3)$$

$$f_5 = x_5(x_{45} \cdot f_4 \vee x_{35} \cdot f_3)$$

$$f_3 = x_3(x_{13} \cdot f_1 \vee x_{23} \cdot f_2)$$

$$f_2 = x_2(x_{12} \cdot f_1)$$

$$f_1 = x_1$$

Аналогичеая система решается также методом подстановок.

При I-м варианте: за min сечение брался входной полюс.

Во II-м – выходной.

Но min сечений м. б. мала –> ФАЛ можно построить как хочешь. Возьмём за min сечение сечение 3-4.

III. подход т.е. если x_3, x_4 не работает –> система не работает

$$f = (x_{23} \cdot f_{21} \vee x_{13} \cdot f_{11}) \cdot x_3 \cdot (x_{34} \cdot f_{46} \vee x_{35} \cdot f_{56}) \&$$
$$\& (x_{24} \cdot f_{21} \vee x_{34} \cdot f_{31}) \cdot x_4 \cdot (x_{45} \cdot f_{56} \vee x_{46} \cdot f_{66})$$

То, что входит, обозначается со стороны входного полюса.

f_{21} – относительно входного полюса.

А то, что вых. –> свяжем с 6-м полюсом: f_{46}

У нас возникают проблемы как в ориентированном графе –> наше сечение неудобно, т.к. между узлами есть связь.

Возьмём сечение IV, там ничто между собой не зависит.

$$f_{21} = x_2 \cdot x_{12} \cdot x_1$$

$$f_{66} = x_6$$

$$f_{56} = x_5 (x_{56} \cdot x_6)$$

Здесь при IV мы разваливаем структуру системы на кучу простых подграфов –> решение будет простым.

Для классификации вероятностного метода ФАЛ должны быть в явном виде.

Следующий этап:

2.6 Формы, допускающие полное замещение логических переменных вероятностными показателями.

Формами, допускающими замещение логических переменных вероятностными показателями являются:

1. СДНФ – совершенно дизъюнктивная нормальная форма.

2. Без повторная форма в базисе конъюнкция-отрицание.
3. Дизъюнкция бесповторных ортогональных форм.

В силу того, что f(ФАЛ) (записанная по путям), всегда есть ДНФ необходимо осуществлять переходы к любому виду, допускающему замещение (иначе не построить полином).
Используются любые законы бинарной алгебры.

Из класса известных методов, применяющихся для перехода, есть 2:

- алгоритмы разрезания
- метод ортогонализации

2.6.1 Алгоритм разрезания

Базируется на th. Разложение Булевской алгебры

Th Разложение.

$$f(x_1, x_2, ..., x_n) = \bigvee_{\{\delta_1 \delta_2 ... \delta_3\}} x_1^{\delta_1} x_2^{\delta_2} ... x_n^{\delta_n} f(\delta_1 ... \delta_n)$$

$$\delta_i = \begin{cases} 0 \\ 1 \end{cases}$$

$$x_i^{\delta_i} = \begin{cases} x_i & \delta_i = 1 \\ \overline{x_i} & \delta_i = 0 \end{cases}$$

$\{\delta_1 ... \delta_n\}$ – *перебор по всему многообразию векторов этих переменных от* $\begin{matrix} \{0,0,000\} \\ \{1111\} \end{matrix} \Big\} 2^n$

1-ый частный случай Th:

$$f(x_1, x_2, ..., x_n) = \bigvee_{\{f(\delta_1 ... \delta_n)=1\}} x_1^{\delta_1} x_2^{\delta_2} ... x_n^{\delta_n}$$ – *этоСДНФ*

$f(\delta_1 ... \delta_n)=1$ - по всем наборам const, где ФАЛ=1

тогда это форма представления произвольной функции *f* в СДНФ.

СДНФ эквивалентно методу полного перебора состояний системы.

Есть сиситема, у неё есть ФАЛ: *f(x)* – *n*-мерная функция

Осуществить перебор всех наборов $\{\delta_0\}$

2^n $\begin{cases} от & 00000 \\ до & 11111 \end{cases}$ подставляем в *f(x)* и смотрим, система работоспособна или нет.

Считаем вероятность попадания в такое состояние.

2-ый частный случай Th:

$f(x_1, x_2, ..., x_n) = x_i f(x_1, ..., x_i = 1, ..., x_n) \vee \overline{x_i} f(x_1, ..., x_i = 0, ..., x_n)$ **-** это метод исключения особого элемента.

x_i – особый элемент. = 1 – работоспособность идеальна

= 0 – неработоспособно.

2.6.2 Метод ортогонализации

Позволяет из ДНФ построить дизъюнкцию ортогональных бесповторных форм.

В общем виде ДНФ:

$f = f_1 \vee f_2 \vee f_3 \vee f_4 \vee ... \vee f_n$

f_i – это некая дизъюнкция логических переменных.

Чтобы построить дизъюнкцию ортогональных форм надо построить f в следующем виде:

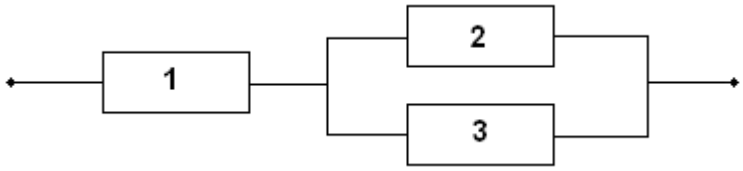
$f = f_1 \vee \overline{f_1} \cdot f_2 \vee \overline{f_1} \cdot \overline{f_2} \cdot f_3 \vee \overline{f_1} \cdot \overline{f_2} \cdot \overline{f_3} \cdot f_4 \vee ...$

Здесь все элементы по построению ортогональны.

Надо сделать, чтобы каждый элемент был бесповторным. Бесповоторная форма – для простых систем, а СДНФ и ортогональность – для любых.

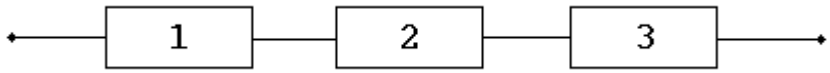
Примеры:

1.



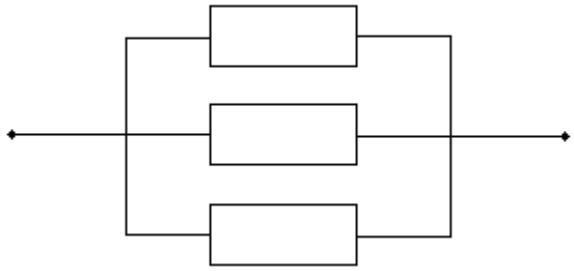
$f = x_1 \cdot x_2 \vee x_1 \cdot x_3 = x_1 \cdot \overline{(\overline{x_2 \vee x_3})} = x_1 \cdot \overline{\overline{x_2} \cdot \overline{x_3}}$ - это бесповторная форма с конъюнкцией и отрицанием.

2.



$f = x_1 \cdot x_2 \cdot x_3$ - СДНФ

3.



Форма, допускающая замещение логических переменных.

$f_1 = x_1 \vee x_2 \vee x_3$ - ДНФ

Переходим:

$$\begin{cases} f_1 = \overline{\overline{x_1 \vee x_2 \vee x_3}} = \overline{\overline{x_1} \cdot \overline{x_2} \cdot \overline{x_3}} \\ f_1 = x_1 \vee \overline{x_1} \cdot x_2 \vee \overline{x_1} \cdot \overline{x_2} \cdot x_3 \end{cases}$$

$$f_1 = x_1 \vee x_2 \vee x_3 = x_1 \cdot 1 \cdot 1 \vee x_2 \cdot 1 \cdot 1 \vee x_3 \cdot 1 \cdot 1 = x_1 \cdot (x_2 \vee \overline{x_2}) \cdot (x_3 \vee \overline{x_3}) \vee x_2 \cdot (x_1 \vee \overline{x_1}) \cdot (x_3 \vee \overline{x_3}) \vee$$
$$x_3 \cdot (x_1 \vee \overline{x_1}) \cdot (x_2 \vee \overline{x_2}) = x_1 \cdot x_2 \cdot x_3 \vee x_1 \cdot x_2 \cdot \overline{x_3} \vee x_1 \cdot \overline{x_2} \cdot x_3 \vee \overline{x_1} \cdot x_2 \cdot x_3 \vee x_1 \cdot \overline{x_2} \cdot \overline{x_3} \vee$$
$$\overline{x_1} \cdot x_2 \cdot \overline{x_3} \vee \overline{x_1} \cdot \overline{x_2} \cdot x_3$$

Получим СДНФ.

Последний этап –

2.7 Замещение логических переменных и построение вероятностного полинома.

Берётся любая форма, допускающая замещение (ФДЗ)

Здесь заменяются:

&	–	*
v	–	+
x_i	–	$P_i = P(x_i = 1)$
$\overline{x_i}$	–	$1\text{-}P_i = q_i = P(x_i = 0)$

Рассмотри пример 2:

$R_C = p_1 * p_2 * p_3 = \prod_{i=1}^{3} p_i$ - вероятность работоспособности системы

Для примера 3: (для СДНФ)

$$R_C = P\left(f_1 = 1\right) = p_1 p_2 p_3 + p_1 p_2 q_3 + p_1 q_2 p_3 + \ldots$$

$$f_1 = \overline{\overline{x_1} \cdot \overline{x_2} \cdot \overline{x_3}}$$

$$R_C = P\left(f_1 = 1\right) = 1 - \left(1 - p_1\right) \cdot \left(1 - p_2\right) \cdot \left(1 - p_3\right)$$

Это методы вычисления сложных вероятностей.

Класс логико-вероятностных методов (ЛВМ) позволяет оценивать показатели надёжности систем, состоящих из 20-25-ти элементов при ручном подсчёте.

2.8 **МОДИФИЦИРОВАННЫЙ ЛОГИКО-ВЕРОЯТНОСТНЫЙ МЕТОД**

Сейчас часто используется модифицированный логико-вероятностный метод (МЛВМ).

В ЛВМ наиболее трудоёмкий этап перехода от ФАЛ к ФДЗ.

Для упрощения этой процедуры был введён МЛВМ

1. Запись ФАЛ
2. Преобразование ФАЛ к ФДЧЗ (форма, допускающая частичное замещение логических переменных). И вместо ФАЛ уже есть СФФВ (смещённые формы функции вероятности) – есть и логические операции и операторы, и арифметические.
3. СФФВ –> ФДЧЗ –> и дальше. Здесь используется алгоритм разрезания.
4. И этот этап (3) повторяется столько раз, пока не будут замещены все логические переменные.

Для МЛВМ систем ФАЛ может быть представлен в любом виде; в логическом и в виде систем уравнений, причём для каждого уравнения всё работает.

I. *f(x)* – ФАЛ, *x* – вектор логических переменных. $i = 2\ldots n,\ x_i \in X_1$

Например:

1) $f(x) = x_1 f(x_1)$ – это бесповторная форма, допускающая замещение.

$$P\left(f=1\right)=1-q_1^{f_1(x_1)}=1-P\left(x_1=0\right)^{f_1(x_1)}=1-\left(1-P_1\right)^{f_1(x_1)}$$

2) $f\left(x\right)=x_1\cdot x_2\cdot x_3\cdot f_2\left(x_2\right) \qquad x_i\in X_2\ ,i=\overline{4,n}$

$x_1 \cdot x_2 \cdot x_3 = z$

$$P\left(f\ =1\right)=1-\left(P_1P_2P_3\right)^{f_2(x_2)}$$

- последовательное соединение некоторой группы:

$$f\ (x)=f_1(x_1)\cdot f_2(x_2) \qquad x_1\vee x_2=x$$

$$P\left(f\ =1\right)=1-P\ \left(f_1(x_1)=0\right)^{f_2(x_2)}$$

$$f\ (x\ )=x_1\cdot f\ (x_1)\vee x_2\cdot f_2(x_1)\vee x_3\cdot f_3(x_1)\vee\ldots x_k\cdot f_k(x_1)\ ,\quad x_i\in X_1,\ \ i=\overline{k+1,n}$$

тогда x_i , $i=1..k$ - бесповторные.

$$P\left(f\ =1\right)=1-q_1^{f_1}\cdot q_2^{f_2}\cdot\ldots\cdot q_k^{f_k}$$

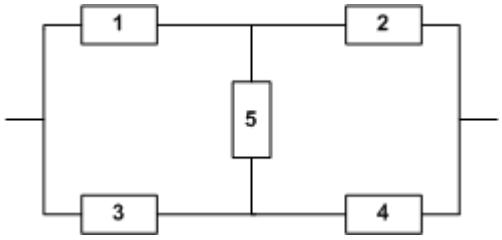
Как правило, в таких формах логические переменные, которые можно заместить, являются бесповторные логические переменные.

3) Исключение логических переменных из СФФВ осуществляется путём реализации алгоритма разрезания с выносом этой логической переменной за СФФВ и замещением её вероятностными показателями.
4) Можно исключать бесповторные логические функции из СФФВ.
   Безповторная логическая функция (БЛФ) – функция, которая не входит как функция ни в одну другую смешанную.
5) Обработка каждого слагаемого в СФФВ может осуществляться отдельно.

Глава 3. **ОЦЕНКА ПОКАЗАТЕЛЕЙ НАДЕЖНОСТИ СИСТЕМ СО СЛОЖНОЙ СТРУКТУРОЙ.**

:

3.1 Оценка показателей надежности структуры типа мостик.



$f(x) = x_1(x_2 \vee x_5x_4) \vee x_3(x_4 \vee x_5x_2)$

$f_1 = (x_2 \vee x_5x_4)$

$f_2 = (x_4 \vee x_5x_2)$

Заменяем логическую переменную x_5

$P(f=1) = 1 - q_1^{f_1}q_3^{f_2} = 1 - q_1^{x_1 \vee x_5x_4}q_3^{x_4 \vee x_5x_2} = P_5(1 - q_1^{x_2 \vee x_4}q_3^{x_4 \vee x_2}) + (1-P_5)(1 - q_1^{x_2}q_3^{x_4})$

Используем алгоритм разрезания:

$x_2 \vee x_4 = z$

Из первого слагаемого исключаем эту логическую функцию, отбрасываем первое слагаемое(так как первое и второе слагаемые не пересекаются).

$P(z=1)P_5(1 - q_1q_3) + P(z=0)P_5 \cdot 0 = (1 - q_2q_4)P_5(1 - q_1q_3)$ - это вероятностное значение первого слагаемого.

Алгоритм разрезания:

$f(x) = x_i f(x|_{x_i=1}) \vee \overline{x_i} f(x|_{x_i=0})$

Для второго слагаемого также будем использовать алгоритм разрезания.

Будем использовать алгоритм разложения по двум переменным: x_2 и x_4

$R = q_5(1 - q_1^{x_2}q_2^{x_4}) = P_2P_4q_5(1 - q_1q_3) + P_2q_4q_5(1 - q_1) + q_2P_4q_5(1 - q_3) + 0$

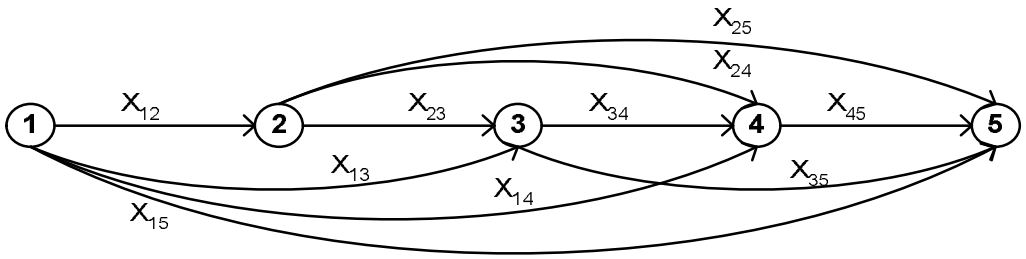
Таким образом мы получим все вероятностные составляющие.

Итак, показатель надежности для мостика:

$$R_c = P_5(1-q_2q_4)(1-q_1q_3) + P_2P_4q_5(1-q_1q_3) + P_2q_4q_5P_1 + q_2P_4q_5P_3$$

Итак, на первом шаге исключаем 2 переменные: x_2, x_4. Потом исключаем еще одну: x_5.

3.2 Оценка вероятности связности системы представленной пяти узловым полно связным графом.



5+4+3+2+1=15 элементов (дуги + узлы)

Каждый узел имеет свою вероятность $P_i, i = \overline{1,5}$

Если система не восстановления: $P_i = e^{-\lambda_i t}$

Система с восстановлением: $P_i = k_{\Gamma i} = \dfrac{\mu_i}{\mu_i + \lambda_i}$, где под P_i понимаем

коэффициент оперативной готовности: $P_i^{(t)} = k_{\hat{I}\tilde{A}} = k_{\tilde{A}i}P(t)/$

Каждому узлу ставим в соответствие переменную x_i, а каждой дуге x_{ij}

ФАЛ будем записывать в виде системы логических уравнений:

сечение по первому узлу.

$$\begin{cases} f = x_1(x_{12}f_2 \vee x_{13}f_3 \vee x_{14}f_4 \vee x_{15}f_5) \\ f_2 = x_2(x_{23}f_3 \vee x_{24}f_4 \vee x_{25}f_5) \\ f_3 = x_3(x_{34}f_4 \vee x_{35}f_5) \\ f_4 = x_4(x_{45}f_5) \\ f_5 = x_5 \end{cases}$$

Это вероятность связности входного узла 1 и выходного 5

В данной системе все логические переменные оказались не повторными и поэтому могут быть замещены.

$P(f=1)=(1-q_{12}^{f_2}q_{13}^{f_3}q_{14}^{f_4}q_{15}^{f_5})P_1$

$P(f_2=1)=P_2(1-q_{23}^{f_3}q_{24}^{f_4}q_{25}^{f_5})$, обозначим (*)

$P(f_3=1)=P_3(1-q_{34}^{f_4}q_{35}^{f_5})$

$P(f_4=1)=P_4(1-q_{45}^{f_5})$

$P(f_5=1)=P_5$

Тут уже надо замещать логические функции, а их можно замещать только бесповторные.

У нас только f_2 бесповторная.

Заменим в первом уравнении бесповторную f_2:

$P(f=1)=P(f_2=1)P_1(1-q_{12}^{1}q_{13}^{f_3}q_{14}^{f_4}q_{15}^{f_5})+P(f_2=0)P_1(1-q_{13}^{f_3}q_{14}^{f_4}q_{15}^{f_5})=$

мы реализовали алгоритм разрезания по логической функции

$P(f_2=0)=1-P(f_2=1)$, а это мы знаем (*), поэтому:

$=P_2(1-q_{23}^{f_3}q_{24}^{f_4}q_{25}^{f_5})P_1(1-q_{12}q_{13}^{f_3}q_{14}^{f_4}q_{15}^{f_5})+(1-P_2(1-q_{23}^{f_3}q_{24}^{f_4}q_{25}^{f_5}))P_1(1-q_{13}^{f_3}q_{14}^{f_4}q_{15}^{f_5})=$

Бесповторной оказалась одна функция f_3, теперь замещаем ее:

$$=P_3(1-q_{34}^{f_4}q_{35}^{f_5})\left\{P_2(1-q_{23}q_{24}^{f_4}q_{25}^{f_5})P_1(1-q_{12}q_{13}q_{14}^{f_4}q_{15}^{f_5})+(1-P_2(1-q_{23}q_{24}^{f_4}q_{25}^{f_5}))P_1(1-q_{13}q_{14}^{f_4}q_{15}^{f_5})\right\}+$$
$$+(1-P_3(1-q_{34}^{f_4}q_{35}^{f_5}))\left\{P_2(1-q_{23}^{f_3}q_{24}^{f_4}q_{25}^{f_5})P_1(1-q_{12}q_{14}^{f_4}q_{15}^{f_5})+(1-P_2(1-q_{24}^{f_4}q_{25}^{f_5}))P_1(1-q_{14}^{f_4}q_{15}^{f_5})\right\}$$

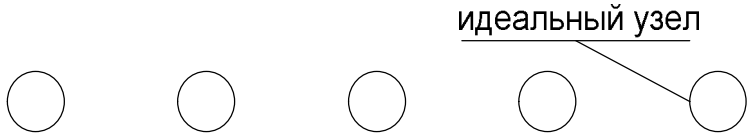
$q_{23}^{f_3}\to 1$.

Бесповторной осталась логическая функция f_4.

Далее исключаем логическую переменную f_4, а затем и f_5, находим вероятностный полином в общем виде. В этой формуле сумма коэффициентов равна 1, таким образом можно проверить правильность формулы.

Частный случай:

Если связей нет, то $P_{ij} = 0$, если нет внутреннего узла, то $P_i = 0$, если полюса нет, то $P_0 = 1$



Таким образом можно распространить формулу для n-узлового графа, но точного аналитического решения иногда не возможно получить, например, при больших размерностях.

3.3 Метод нахождения приближенного решения для случая классического логико-вероятностного метода.

Приближенное решение можно найти в следующих случаях:

1. Строим ФР(функцию работоспособности) приближенно
2. При точном задании ФР переводим ее в приближенную форму, для упрощения процедуры замещения логических переменных.
3. Изменяем процедуру метода и строго приближенного решения.

Таким образом, мы имеем следующие подходы:

1. Если запись идет по путям, то перечисляя все пути, мы найдем приближенное значение ФАЛ, которая даст заниженную оценку работоспособности.
2. Если по сечениям – то не перечисляя все сечения находим приближенное значение ФАЛ – завышенную оценку работоспособности.

Допустим, мы имеем ФАЛ, полученную путем перечислением путей, тогда тогда она будет в ДНФ. Чтобы облегчить процедуру построения формы, допускающей замещение логических переменных, дописываем x_i там, где они необходимы.

1. $x_1x_2x_3 \vee x_1\overline{x_2}$, допишем x_3, чтобы получить СДНФ: $x_1x_2x_3 \vee x_1\overline{x_2}x_3$, теперь можем получить приближенную, заниженную оценку.
2. Можно вычеркнуть некоторую x_j, тогда получим завышенную оценку.
   $x_1x_2x_3 \vee x_1\overline{x_2} \to x_1x_2 \vee x_1\overline{x_2}$
3. Вычеркивая в ДНФ целый путь получаем приближенное решение – заниженную оценку.

Можно использовать свойство монотонности логической функции:
Монотонность ФАЛ:

1. $P(f(x/x_i = 1) = 1) = 1 \quad i = \overline{1,n}$ - если в системе все комплексы работают, следовательно, система работоспособна, в целом.
2. $P(f(x/x_i = 0) = 0) = 0 \quad i = \overline{1,n}$ - если в системе все отказало, она не работает.
3. $P(f(x/x_i = \sigma_i x_j = 1) = 1) \geq P(f(x/x_i = \sigma_i x_j = 0) = 1) \quad i = \overline{1,n}$ - если в системе отказало 5 элементов, то отказ 6-го не улучшит параметров надежности.

Приведенный подход позволяет узнать приближенное решения, которые дают и завышенную, и заниженную оценку.

$f(x)$ - производная ФАЛ $x \quad x_i \in X \quad i = 1..n$.

Выбирая случайно *k* логических переменных, возьмем, для примера, k_1, k_2, k_3.

Реализуем алгоритм разрезания по переменным.

$$f(x) = x_1x_2x_3f_1 \vee x_1x_2\overline{x_3}f_2 \vee x_1\overline{x_2}x_3f_3 \vee \overline{x_1}x_2x_3f_4 \vee x_1\overline{x_2}\,\overline{x_3}f_5 \vee \overline{x_1}x_2\overline{x_3}f_6 \vee \overline{x_1}\,\overline{x_2}x_3f_7 \vee \overline{x_1}\,\overline{x_2}\,\overline{x_3}f_8$$

Запишем вероятностный полином, так как все слагаемые ортогональны.

$$P(f = 1) = p_1p_2p_3P(f_1 = 1) + p_1p_2q_3P(f_2 = 1) + p_1q_2p_3P(f_3 = 1) + q_1p_2p_3P(f_4 = 1) +$$

$$+ p_1q_2q_3P(f_5=1)+q_1p_2q_3P(f_6=1)+q_1q_2p_3P(f_7=1)+q_1q_2q_3P(f_8=1) \quad (*)$$

разрезаем по трем первым переменным:

$f_1 = f(1,1,1,x_1)$

$f_2 = f(1,1,0,x_1)$

$f_3 = f(1,0,1,x_1) \qquad x_1 : x \in X, i = \overline{4,n}$

$f_4 = f(0,1,1,x_1)$

$f_5 = f(1,0,0,x_1)$

В силу монотонности:

$P(f_1=1) \geq P(f_2=1)$

$P(f_1=1) \geq P(f_3=1)$

$P(f_1=1) \geq P(f_4=1)$

$\max\{P(f_2=1); P(f_3=1); P(f_4=1)\} = P_m$

$P_m \geq P(f_5=1)$

$P_m \geq P(f_6=1)$

$P_m \geq P(f_4=1)$

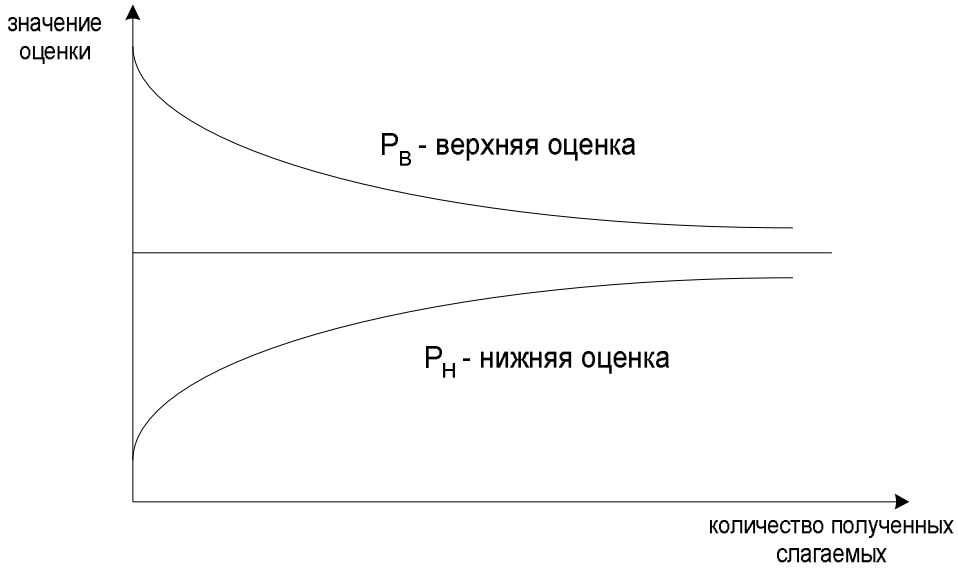
$\max\{P(f_5=1); P(f_6=1); P(f_7=1)\} = P_{m_2}$

$P_{m_2} \geq P(f_8=1)$

Рассмотрим уравнение (*)

$P(f=1) = P_1P_2P_3P(f_1=1)...P(f_8=1)$. Если заменить $P(f_1=1)$ и вставить его на место всех $P(f_i=1)$, то получим завышенную оценку, равную 1.

$P(f=1) = P_1P_2P_3P(f_1=1)$ - получим нижнюю оценку, так как мы отбросили возможно работоспособные элементы. Если вычислить больший $P(f_i=1)$, то заменив, максимальной из них, на все не вычисленные, получим вторую верхнюю оценку $\tilde{A}_{\hat{A}2}$, и отбросив не посчитанные, получим нижнюю оценку.



Нижнюю оценку можно строить на основе следующих формул.

$P(f_8 = 1) \le P(f_7 = 1)$

$P(f_8 = 1) \le P(f_6 = 1)$

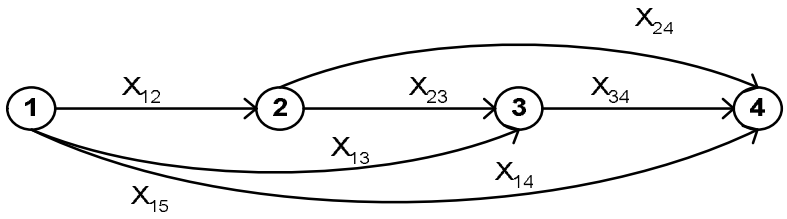
$P(f_8 = 1) \le P(f_5 = 1)$

Аналогично ищем максимум.

Отбросив слагаемые $P(f = 1) = P_1P_2P_3P(f_1 = 1) + ...$ Значение вероятности $P(f_1 = 1)$ можно считать приближенно. *Плюс в том, что имеем и нижнюю и верхнюю оценки.*

3.4 Методы построения приближенных решений в случае использования модифицированного логико-вероятностного метода.

Рассмотрим построение приближенных решений на примере оценки показателей надежности системы представимой полносвязанным 4-узловым графом.



$$f = x_1(x_{12}f_2 \vee x_{13}f_3 \vee x_{14}f_4)$$

$$f_2 = x_2(x_{23}f_3 \vee x_{24}f_4)$$

$$f_3 = x_3(x_{34}f_4)$$

$$f_4 = x_4$$

$$P(f=1) = P_1(1 - q_{12}^{f_2} q_{13}^{f_3} q_{14}^{f_4}) \qquad (1)$$

$$P(f_2=1) = P_2(1 - q_{23}^{f_3} q_{24}^{f_4}) \qquad (2)$$

$$P(f_3=1) = P_3(1 - q_{34}^{f_4}) \qquad (3)$$

$$P(f_4=1) = P_4 \qquad (4)$$

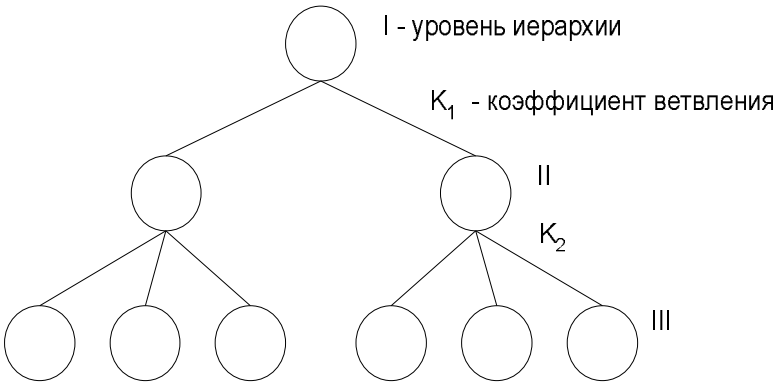
Ищем приближенное решение:

1. Положим $f_i = 0$, тогда получим заниженную оценку. При этом, $f_3 = 0$, например можно полагать в любом выражении. У нас возникает несколько вариантов:
   $P(f_2=1)$, отсюда $f_3 = 0$, получаем одно
   $P(f=1)$, отсюда получаем более приближенное.
2. Положим $f_i \le 1$ и получим завышенную оценку. Замену также можно делать в любом выражении, от места применения зависит то, насколько упростится решение, и насколько оно будет лучше.
3. можно считать $P(f_i=1) = P_i$, тогда мы тоже получим завышенную оценку.

3.5 Оценка надежности иерархических структур.

3.5.1 Показатели надежности иерархических структур.

Рассмотрим иерархическую структуру типа дерева.



Иерархические структуры, в общем случае, не сводятся к двухполюснику. Только в случае, когда она не избыточная, тогда

$R_c = \prod_{i \in I_0} p_i$ - все элементы(узлы, цепи)

1. Здесь трудно реализовать понятие отказа, переходят к минимально допустимому уровню качественного функционирования.
2. Берем среднее число ветвей, в смысле работоспособности. Здесь $N=k_1$,
   $k_2 = 2*3=6$ (ветвей). Среднее число работоспособных ветвей:
   $N_{ср}=N*P_{ветвей}$. $P_{ветвей}$ – вероятность безотказного функционирования ветвей.

$N_{ñð} = \sum_{i=0}^{N} iP_i$ i – число работоспособных ветвей, P_i – вероятность этого события.

Показатель надежности:

$R_c = \sum_{i=0}^{N} \alpha_i P_i$, α_i - весовой коэффициент, говорит, какой вклад в показатель R_c , если *i* ветвей работоспособны. Их достаточно сложно считать.

Но для того, чтобы характеризовать хорошо, необходимо иметь вектор показателей, в зависимости от допустимого уровня качества функционала.(*z* – число работоспособных ветвей).

Всего ветвей:

$0 = \frac{0}{N}$ - число отказавших ветвей, к общему числу ветвей.

$\frac{N_{îòêê}}{N}$ - коррелированный показатель *m*, находящийся в интервале [0,1].

Таким образом, нам все равно необходимо искать P_i . Такую процедуру нахождения P_i , дает МЛВМ.

3.5.2 Процедура нахождения производящего полинома для изотропной системы.

Процедура нахождения производящего полинома, для нахождения P_i , состоит из следующих этапов:

1. Построение ФАЛ для ветви.
2. Исключение логических переменных нижнего уровня иерархии и переход к смешанной форме записи $P(f = 1)$.
3. Формальное построение: $1 + (z - 1)P(f = 1)$
4. Возведение производную полинома в степень, равную коэффициенту ветвления на нижнем уровне.
5. Исключение логических переменных вышестоящих уровней.

6. Возведение в степень, равную коэффициенту ветвления вышестоящего уровня. Продолжать 5-6 до тех пор, пока не добрались до верхнего уровня.

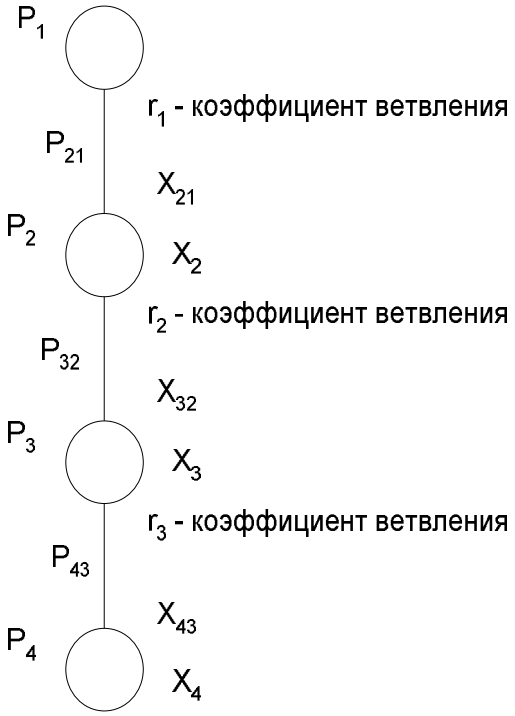
В результате мы найдем полином:

$\hat{O}(z) = \sum_{i=0}^{k} z^i P_i$, i – число работоспособных ветвей, тогда z^k - не избыточная,

если $z^k \prod_{i=1} P_i$

а при i=0 – отказ всех элементов в системе.

Итак, имеем изотропное дерево:



ФАЛ: $f = x_4 x_{34} x_3 x_{32} x_2 x_{21} x_1$

$P(f=1) = 1 - (1 - P_4 P_{43})^{x_3 x_{32} x_2 x_{21} x_1}$

x_4 и x_{43} - элементы нижнего уровня иерархии, тогда

$$\hat{O}(z) = \left[1 - (1-z)\left[1 - (1 - P_4 P_{43})^{x_3 x_{32} x_{21} x_1}\right]^{r_3}\right. = \left[p_3 p_{32}\left[1 - (1-z)\left[1 - (1 - P_4 P_{43})^{x_2 x_{21} x_1}\right]^{r_3} + (1 - p_3 p_{32})[1]\right]^{r_2} =$$

Теперь исключаем все логические переменные вышестоящего уровня иерархии.

Исключим теперь второй уровень иерархии: x_2 , x_{21}

$$\left\{p_2 p_{21}\left[p_3 p_{32}\left[1-(1-z)\left[1-(1-P_4 P_{43})^{x_1}\right]^{r_3}+(1-p_3 p_{32})\right]^{r_2}+(1-p_2 p_{21})\right\}^{r_1}=\right.$$

теперь исключаем x_1, тогда:

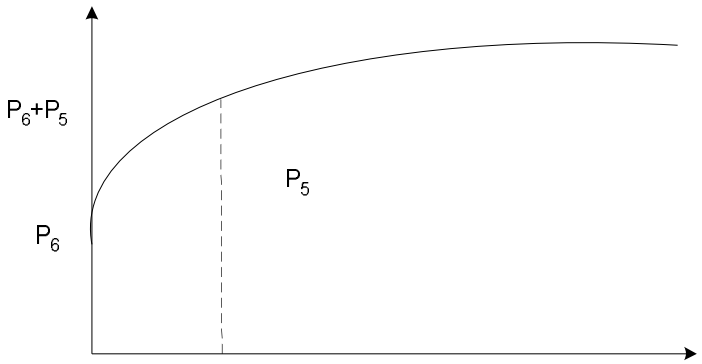
$$p_1\left\{p_2 p_{21}\left[p_3 p_{32}\left[1-(1-z)\left[1-(1-P_4 P_{43})^{x_1}\right]^{r_3}+(1-p_3 p_{32})\right]^{r_2}+(1-p_2 p_{21})\right\}^{r_1}+q_1\right.$$.

Запишем вероятностный полином для случая k=6 – общий вид Φ(z):

$z^6 p_6 + z^5 p_5 + .. + z p_1 + p_0$

p_6 - вероятность того, что в системе работоспособно 6 ветвей. Мы говорим, что система откажет, при отказе более 2х ветвей

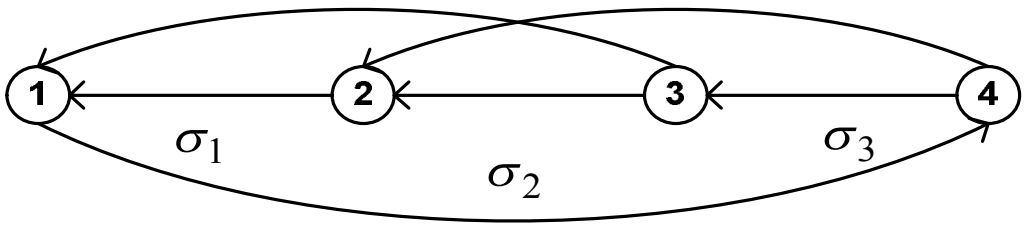
$k = p_6 + p_5 + p_4$ - коэффициент готовности.



- имея это, мы можем, как только нам скажут, чтотакое вероятность отказа, так мы сразу возрастем на этот показатель.

3.5.3 Построение производящего полинома для системы представимой 4х уровневой иерархической структурой.

Структура ветви рассматриваемой системы приведена на рисунке.



Составляем систему логических уравнений описывающих условие безотказного функционирования ветви:

$$f = x_4(x_{43}f_3 \vee x_{42}f_2 \vee x_{41}f_1)$$
$$f_3 = x_3(x_{32}f_2 \vee x_{31}f_1)$$
$$f_2 = x_2(x_{21}f_1)$$
$$f_1 = x_1$$

При такой форме записи в каждом уравнении оказывается бесповторными логические переменные соответствующего уровня.
Записываем смешанные формы функции вероятности

$$P(f = 1) = P_4(1 - q_{43}^{f_3} q_{42}^{f_2} q_{41}^{f_1})$$
$$P(f_3 = 1) = P_3(1 - q_{32}^{f_2} q_{31}^{f_1})$$
$$P(f_2 = 1) = P_2(1 - q_{21}^{f_1})$$
$$P(f_1 = 1) = P_1$$

Это позволяет сразу построить производящий полином для всех случаев

$$\Phi(z) = [1 - (z - 1)P_4(1 - q_{43}^{f_3} q_{42}^{f_2} q_{41}^{f_1})]^{r_3}$$

Надо исключить все элементы вышестоящего уровня иерархии. В данном случае исключаем просто f_3:

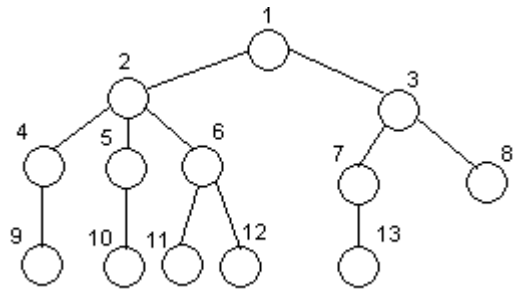
$$= [P_3(1 - q_{32}^{f_2} q_{31}^{f_1})[1 - (z-1)P_4(1 - q_{43}q_{42}^{f_2}q_{41}^{f_1})]^{r_1} + (1 - P_3(1 - q_{32}^{f_2} q_{31}^{f_1}))[1 - (z-1)P_4(1 - q_{42}^{f_2}q_{41}^{f_1})]^{r_1}]^{r_2} =$$

Нужно исключить все элементы следующие уравнения иерархии

$$= P_2(1-q_{21}^{f_1})[P_3(1-q_{32}q_{31}^{f_1})[1-(z-1)P_4(1-q_{43}q_{42}q_{41}^{f_1})]^{r_1} +$$
$$+(1-P_3(1-q_{32}q_{31}^{f_1})[1-(z-1)P_4(1-q_{42}q_{41}^{f_1})]^{r_1}]^{r_2} +$$
$$+(1-P_2(1-q_{21}^{f_1}))[P_3(1-q_{31}^{f_1})[1-(z-1)P_4(1-q_{43}q_{41}^{f_1})]^{r_1} +$$
$$+(1-P_3(1-q_{31}^{f_1}))[1-(z-1)P_4(1-q_{41}^{f_1})]^{r_1}]^{r_2} = \Phi_4(z)$$
$$\Phi(z) = q_1 \cdot 1 + P_1[\Phi_4(z)]^{r_1}\Big|_{f_1=1}$$

3.5.4 Построение производящего полинома для не изотропной структуры

Не изотропия возникает из-за различия показателей надежности элементов одного уровня иерархии, из-за различного числа уровней иерархии, структуры ветвей, коэффициентов ветвления на одном уровне иерархии



Пример

$$x_i = \overline{1,13}$$
$$P(x_i = 1) = P_i$$

Для каждой ветви записываем свою функцию алгебраической логики ФАЛ

$$f_1 = x_9 x_4 x_2 x_1$$
$$f_2 = x_{10} x_5 x_2 x_1$$
$$f_3 = x_{11} x_6 x_2 x_1$$
$$f_4 = x_{12} x_6 x_2 x_1$$
$$f_5 = x_{13} x_7 x_3 x_1$$
$$f_6 = x_8 x_3 x_1$$

Исключим в каждой ветви самый нижний уровень иерархии

$\Phi_1(z) = 1-(z-1)(1-q_9{}^{x_4x_2x_1})$ *(необходимо исключить x_4)*

$\Phi_2(z) = 1-(z-1)(1-q_{10}{}^{x_5x_2x_1})$ *(необходимо исключить x_5)*

$\Phi_3(z) = 1-(z-1)(1-q_{11}{}^{x_6x_2x_1})$ *(нужно объединять, перемножая,*

$\Phi_4(z) = 1-(z-1)(1-q_{12}{}^{x_6x_2x_1})$ *а потом исключить x_7)*

$\Phi_5(z) = 1-(z-1)(1-q_{13}{}^{x_7x_3x_1})$ *исключаем x_7*

$\Phi_6(z) = 1-(z-1)(1-q_8{}^{x_3x_1})$

$\Phi_1(z) = q_4 \cdot 1 + P_4(1-(z-1)(1-q_9{}^{x_2x_1}))$ *исключаем x_2*

$\Phi_2(z) = q_5 + P_5(1-(z-1)(1-q_{10}{}^{x_2x_1}))$

$\Phi_{3,4}(z) = q_6 + P_6(1-(z-1)(1-q_{11}{}^{x_2x_1}))\cdot(1-(z-1)(1-q_{12}{}^{x_2x_1}))$ *не трогаем, т.к. другой уровень иерархии*

$\Phi_5(z) = q_7 + P_7(1-(z-1)(1-q_{13}{}^{x_3x_1}))$ *перемножаем,*

$\Phi_6(z) = 1-(z-1)(1-q_8{}^{x_3x_1})$ *исключаем x_3*

$\Phi_{1,2,3,4}(z) = q_2 + P_2[q_4 + P_4(1-(z-1)(1-q_9{}^{x_1}))][q_5 + P_5(1-(z-1)(1-q_{10}{}^{x_1}))]$

$[q_6 + P_6(1-(1-z)(1-q_{11}{}^{x_1}))(1-(z-1)(1-q_{12}{}^{x_1}))]$

$\Phi_{5,6}(z) = q_3 + P_3[(q_7 + P_z(1-(z-1)(1-q_{13}{}^{x_1})))\cdot(1-(z-1)(1-q_8{}^{x_1}))]$

$\Phi(z) = q_1 + P_1 \cdot \Phi_{1234}(z)\Big|_{x_1=1} \cdot \Phi_{56}(z)\Big|_{x_1=1}$

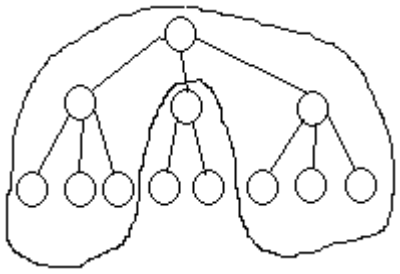
Теперь формализуем эту процедуру

1) для каждой ветви пишется своя ФАЛ
2) для каждой ветви строим свой производящий полином $\Phi(z)$, исключаем элементы самого нижнего уровня иерархии

3) ищется группа производящих полиномов, в которых одинаковы логические переменные вышестоящего уровня иерархии
4) производящие полиномы в этих группах перемножаются
5) из всех вновь полученных полиномов производится исключение логических переменных вышестоящего уровня иерархии
6) возвращаемся к пункту 3

Процедура продолжается до тех пор, пока не будет исключен самый верхний уровень иерархии.

В части структуры может быть изотропия.



В этом случае пользуемся методом расчета как для изотропной структуры.

Распределение коэффициентов готовности

Если $\Phi(z) = \sum_{i=0}^{n} z^n P_n = P_n z^n + P_{n-1} z^{n-1} + \ldots + P_0$

P_0 означает вероятность полной потери информации в системе

Если накладывается ограничение на другие системы. Например: система работоспособна, если могут отказать до 3-х ветвей!

$K_r(m) = P_n + P_{n-1} + P_{n-2} + P_{n-3}$

P_n – вероятность безотказного функционирования безызбыточной системы

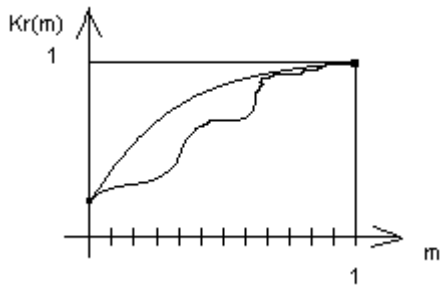
$P_n = \prod_{i \in I_0} p_i$ произведение всех элементов этой системы

Должны находить распределения

$K_r(0) = P_n$

$K_r(\frac{1}{k}) = P_n + P_{n-1}$

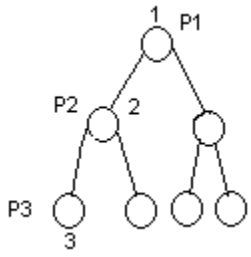
$K_r(\frac{2}{k}) = P_n + P_{n-1} + P_{n-2}$ имеет целый вектор таких показателей



Возможны различные варианты зависимостей показателя надежности системы от допустимого порога снижения качества функционирования.

3.5.5 Резервирование в иерархических структурах.

1) поэлементное резервирование



$P_1^* = 1 - q_1^2$ P_1 – после резервирования

Ф(z) не меняется, изменяется только вероятности показателей

$\Phi(z) = q_1 + P_1(...)$

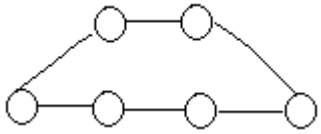
Если резервировать 1-й элемент, то $q_1 \downarrow$. Это означает, что поэлементное резервирование верхнего уровня иерархии улучшает показатели надежности при очень больших m.

Наличие большого числа нерабочих ветвей в системе.

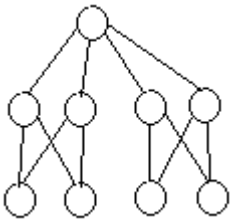
Для *к*-уровневой иерархической структуры в области *m* (от 0 до 1) можно выделить *к* диапазонов, в каждом диапазоне будет эффективно резервирование только своего уровня иерархии

Для высоконадежных систем эти области смещены в области очень малых значений *m*.

2) Групповое резервирование

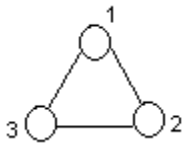


Структура ветви меняется. Требует всех новых расчетов



Если можно говорить о групповом резервировании, привязанном к уровням иерархии, то можно понятие ……

3.6 Оценка надежности сетевых структур.



В качестве основного показателя берется матрица вероятности связности узлов

$$\begin{pmatrix} P_{11} & P_{12} & P_{13} \\ P_{21} & P_{22} & P_{23} \\ P_{31} & P_{32} & P_{33} \end{pmatrix}$$

Нужно посчитать n^2 показателей, равных вероятности связности двухполюсника.

Не существует отработанных методик.

f_{1i} – логическое выражение связи 1-го элемента с *i*-м

$$f_{1i} = x_1(x_{12}f_{2i} \vee x_{13}f_{3i})$$
$$f_{2i} = x_2(x_{21}f_{1i} \vee x_{23}f_{3i})$$
$$f_{3i} = x_3(x_{31}f_{1i} \vee x_{32}f_{2i})$$

$$f_{1i} =^{v0} x_1x_{12}x_2 \vee x_1x_{13}x_3$$
$$f_{2i} = x_2x_{21}x_1^{v0} \vee x_2x_{23}x_3$$
$$f_{3i} = x_3x_{31}x_1 \vee x_3x_{32}x_2^{v0}$$

получилась система логических уравнений

Заменим $f_{ki} = x_k$

Имеем матрицу A_1, которая даст вероятность связности этого узла через одну линию связи

Далее реализуем метод подстановки

$$f_{1i} = x_1(x_{12}x_2(x_{21}f_1 \vee x_{23}f_{3i}) \vee x_{31}x_3(x_3f_{2i} \vee x_{32}f_{3i})) =$$

теперь необходимо убрать петли

$$= x_1(x_{12}x_2x_{23}f_{3i} \vee x_{13}x_3x_{32}f_{2i})$$

Теперь матрица A_2: связь каждого узла с каждым, но это проход по 2-м дугам связи.

Если еще раз подставляем => A_3, и т.д.

Если хочу найти матрицу А для любого числа проходов

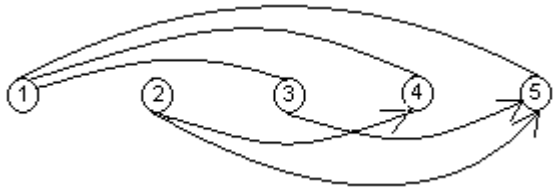
$$A = A_1 \vee A_2 \vee A_3 \vee A_4 \vee A_5 \vee A_7 \ldots$$
$$A = A_1 \vee A_2 \vee A_3$$

Число матриц считается по самому пути в сети, тогда получаем точное решение. Эта процедура позволяет оценить матрицу связи для любой структуры.

Возможен еще один способ решения

$$R_{1,n} = P(x_1 = 1) \cdot P(x_n = 1) \sum_{\{\sigma_2, \sigma_3, \ldots \sigma_{n-1}\}} [1 - \prod_{j=2}^{n} (1 - P(x_{1j} = 1))^{\sigma_j}] \prod_{k=2}^{n-2} A_k$$

$$A_k = \begin{cases} P(x_k = 1)[1 - \prod_{j=k+1}^{n} (1 - P(x_{kj} = 1))^{\sigma_j} & \sigma_k = 1 \\ 1 - P(x_k = 1)[1 - \prod_{j=k+1}^{n} (1 - P(x_{kj} = 1))^{\sigma_j}] & \sigma_k = 1 \end{cases}$$



n^2 раз применим формулу. Получим матрицу связности

$\begin{pmatrix} P_{11} & P_{12} \\ P_{21} & P_{22} \end{pmatrix}$ $\min\limits_{j}$ $\min\limits_{i}$ – этот показатель будем улучшать, используем резервирование $R = \sqrt{\sum_i \sum_j P_{ij}^{\ 2}}$

Сетевая структура достаточно трудоемкий алгоритм

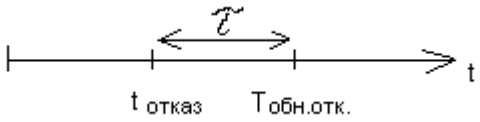
3.7 Учет характеристик средств контроля при оценке надежности сложных систем.

Логико-вероятностные методы предполагают идеальность контроля в системе. Идеальный контроль обнаруживает отказ мгновенно, в любом месте системы и указывает до сменного(?) блока, где произошел отказ.

1. Аппаратный контроль
2. Тестовый контроль

Характеристики

1. Оперативность



аппар.$\tau \equiv 0$ (близок по этому показателю к идеальному). Тестовая τ очень большая.

2. Полнота контроля

Какую часть аппаратно контролируем, в которой будут обнаружены все отказы

3. Глубина контроля

Особенность средства находить место, где произошел отказ.

Попытка использовать аппаратный контроль как идеал приводит к ухудшению части показателя надежности.

Комбинируем аппаратный и тестовый

λ, μ

$\lambda = 1/T_0, \quad \mu = 1/T_B$

Если мы скажем, что идеал, то завышаем оценку

$T_B = T_{обн} + T_{лок} + T_{зам} + T_{инф.восст}$

$\Delta = \frac{T_n}{2}$, T_n – период. $T_B^* = T_B + \Delta$

и теперь мы работаем с μ^*

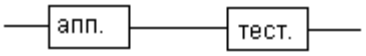
$\mu^* = 1 / T_B^*$

Получаем нижнюю оценку (близкую к марковским результатам).

Комбинированный контроль (напр., 20% - апп., 80% - тест.)

Можно разбить на 2 автономные части (μ^* - посчитать)

0,22 μ 0,82 μ^*



x_1 x_2

Усложнение структуры, увеличиваем трудоемкость решения 3r, подход позволяет учитывать характеристики средств контроля.

Такие подходы дают приближенные решения (чисто интен. подход)

Все это рассмотрение 3r анализа. 3r синтеза – создать систему, удовлетворяющую данным требованиям надежности.

/* тех. средства */

ГЛАВА 4. Надежность программного обеспечения

4.1. Основные понятия и определения

Основные понятия надежности комплексов программ базируются на понятиях теории надежности, первоначально развившейся для аппаратурных комплексов. При этом фундаментальным является понятие надежности как свойства объекта "выполнять заданные функции, сохраняя во времени значения установленных эксплуатационных показателей в заданных пределах, соответствующих заданным режимам и условиям использования, технического обслуживания, ремонтов, хранения и транспортирования". В соответствии с этим определением надежность является внутренним свойством системы, заложенным при ее изготовлении и проявляющимся при эксплуатации. Это свойство проявляется только во времени, и без более или менее длительного учета значений времени нельзя сделать заключения о надежности системы.

Надежность программного обеспечения может быть определена как свойство программы, которое выражается в выполнении заданных функций в заданных условиях работы и на заданной вычислительной машине. Однако механизмы возникновения отказа аппаратуры и отказа программного обеспечения существенно отличаются друг от друга. Отказ программного обеспечения обусловлен несоответствием программного обеспечения поставленным задачам. Несоответствие может возникать по двум причинам: либо разработчиком программы было допущено нарушение спецификации - технических требований к программе, либо спецификация неполная или неточная.

Несоответствие по первой причине встречается в первую очередь в сложных программных системах, где отдельные ошибки программиста трудно обозримы и могут оставаться нераскрытыми.

Несоответствие по второй причине возникает в первую очередь потому, что при составлении спецификации многие факторы, влияющие на работу

программы, неизвестны. Они выясняются только постепенно, в ходе эксплуатации программы. Особенно это относится к управляющим программам.

Следовательно, когда программа работает не так, как предполагает пользователь имеет место программная ошибка. Характерной особенностью ошибок, обуславливающих отказы программ, является скрытость ошибок - это означает, что скрытая ошибка проявляется только при отдельных редких комбинациях из огромного количества возможных комбинаций исходных данных и поэтому обнаруживаются не сразу, а только в ходе длительной эксплуатации. Ошибки, проявляющиеся при любых исходных данных, не опасны, поскольку обнаруживаются сразу, при первых прогонах программы.

Под системой в теории надежности принято понимать совокупность подсистем или элементов, функционально объединенных в соответствии с некоторым алгоритмом взаимодействия при выполнении заданной задачи в процессе применения по назначению. Под это определение системы полностью подходит программное обеспечение. Потеря надежности системой связывается с появлением отказов в работе. Следовательно, надежность программного обеспечения - это вероятность того, что программа какой-то период времени будет работать без сбоев с учетом степени их влияния на выходные результаты.

Надежность программного средства можно также характеризовать средним временем между возникновениями отказов в функционировании программы. При этом предполагается, что аппаратура ЭВМ находится полностью в работоспособном состоянии.

С точки зрения надежности принципиальное (отличие ПО от аппаратуры состоит в том, что программы не изнашиваются и, следовательно, их выход из строя из-за поломки невозможен. Поэтому характеристики функционирования ПО зависят только от его качества, предопределяемого процессом разработки.

Безотказность программного обеспечения определяется его

корректностью и, следовательно, целиком зависит от наличия в нем ошибок, внесенных на этапах его создания.

Надежность (или безотказность) аппаратуры и ПО существенно по-разному зависит от входных данных и времени функционирования системы. Процесс выхода из строя отдельных элементов аппаратуры не зависит от поступающих входных данных. В то же время проявление ошибок программного обеспечения связано с тем, что в некоторые моменты времени на обработку поступают ранее не встречавшиеся совокупности данных, которые программа не в состоянии корректно обработать. Таким образом, входные данные в значительной мере влияют на функционирование программного обеспечения.

Теория надежности аппаратуры применима к проблеме надежности программного обеспечения, учитывая следующие различия:

- элементы программного обеспечения не стареют из-за износа или усталости;
- для контроля программного обеспечения имеется намного больше путей и способов, чем для контроля аппаратуры;
- в программном обеспечении имеется намного больше объектов для контроля, чем в аппаратуре;
- количество документации по программному обеспечению огромное по сравнению с количеством документации по аппаратуре;
- внести изменения в программы просто, но трудно сделать это корректно.

4.2. Показатели надежности ПО и простейшие модели их оценки

Понятие надежности применимо к любому программному продукту. Однако понятие надежности не может быть в полной мере применимо к мелким программным единицам, изготовленным одним разработчиком в основном для личного использования. В дальнейшем будем говорить о

надежности программного обеспечения, если оно удовлетворяет следующим требованиям:

1 изготовлено большим коллективом разработчиков;
2 программный продукт имеет большой объем (10^4 – 10^6 команд) и модульную структуру;
3 время эксплуатации программного продукта значительно превышает время его разработки;
4 программный продукт тиражируется и эксплуатируется большим числом пользователей.

В этом случае под надежностью программного обеспечения будем понимать свойство выполнять заданные функции, сохраняя во времени значения установленных эксплуатационных показателей в заданных пределах, соответствующих заданным режимам и условиям использования, технического обслуживания, ремонта, хранения и транспортирования.

В соответствии с этим определением надежности оно является внутренним свойством системы, заложенным при ее изготовлении и проявляющееся при эксплуатации. Следовательно, для того чтобы составить представление о надежности ПО необходимо наблюдение за поведением системы в период эксплуатации в течение определенного интервала времени, причем эти наблюдения должны проводиться в строгом соблюдении заданных условий эксплуатации.

С понятием надежности тесно связаны понятия работоспособного и неработоспособного состояния системы и программного отказа. Факт перехода из работоспособного состояния в неработоспособное состояние связан с проявлением программного отказа. В настоящее время нет четкого определения отказа программного изделия. Так, одно известное определение программного отказа утверждает, что программная ошибка проявляется тогда, когда ПО работает не в соответствии со своими спецификациями, другое требует дополнительных условий на требования к эксплуатации. Широко используется определение программного отказа, как несоответствия

работы программы первоначальным требованиям пользователя – заказчика или ситуации, когда программа не соответствует сопутствующей ей документации.

Все эти определения программного отказа имеют ряд недостатков, так как они связаны с субъективизмом в принятии решения о факте появления отказа. Возьмем в качестве понятия программного отказа следующее определение: *программная ошибка проявляется тогда, когда программа работает не так, как предполагает пользователь, но только тогда, когда ошибка есть функция от программы.* В таком определении сделана попытка исключить различные человеческие факторы, такие как неправильное толкование текста документации, восприятия положений и т.д. как программные отказы.

Чтобы характеризовать комплексное свойство "надежность" с разных сторон, используются показатели надежности. Под показателями надежности принято понимать совокупность величин, характеризующих качественно или количественно степень приспособленности системы к выполнению поставленных задач при применении по назначению.

В практике проведения исследований рассматриваются *три вида показателей:* качественные, порядковые и количественные.

Качественные показатели надежности – это такие показатели, которые не могут выражены в виде числа и не содержат информации, позволяющей обосновывать предпочтение одного из нескольких конкурирующих вариантов системы при их сравнении. Качественные показатели дают возможность отличать системы друг от друга, но не позволяют сравнивать их по степени выполнения поставленной задачи.

Порядковые показатели надежности содержат информацию, позволяющую обосновывать предпочтение одного варианта перед другим при их сравнении без количественной оценки степени предпочтения. Порядковые показатели дают возможность расположить в ряд по степени возрастания надежности исследуемые варианты системы, но не позволяют

оценить, на какую величину отличается достигнутый уровень надежности рассматриваемых вариантов.

Количественные показатели надежности содержат информацию, обеспечивающую оценку степени предпочтения одного варианта системы перед другим при применении по назначению. Количественные показатели выражают надежность в виде числа. Определяются такие показатели на основе обработки информации о результатах применения или испытания систем, путем аналитических расчетов или моделирования процесса функционирования.

Для ПО, как и для технических систем, могут быть выделены некоторые *этапы жизненного цикла*. Для ранних этапов создания ПО характерно то, что на них количественные показатели надежности оказываются не вычислимы из-за недостаточности и недостоверности исходных данных. На этих этапах используются, как правило, качественные и порядковые показатели надежности.

На этапе выбора структуры программных средств могут оцениваться количественные показатели структурной надежности, которые в достаточно условной форме будут отражать уровень надежности программного обеспечения, так как такие расчеты базируются на низко достоверных исходных данных, но такие оценки позволяют проводить сравнительный анализ различных вариантов построения программного обеспечения и выбор структур с более лучшими показателями надежности. После этапов символического тестирования и автономной отладки модулей, когда устраняются практически все ошибки синтаксического характера, уточняются показатели надежности для структурных единиц ПО. Количественные показатели надежности ПО могут оцениваться уже с достаточной для инженерной практики точностью, а следовательно оцениваться и достигнутый уровень надежности.

Важнейшим показателем надежности ПО, не имеющего аналогов в технических средствах, является число ошибок, оставшихся в системе в

рассматриваемый момент времени t – $N(t)$. Наиболее широко в технической литературе используется показатель – N_0 – число ошибок, оставшихся в программном продукте на момент компоновки программного комплекса и начала детерминированного тестирования (отладки). В процессе отладки обнаруживаются и устраняются n_y ошибок. Идеальный процесс отладки предполагает невнесение новых ошибок при устранении обнаруженных. Обработка статистических данных по большому числу программных проектов показывает, что практически 30% исправлений приводят к появлению в ПО новых ошибок. В силу этого измеряемый в процессе отладки показатель n_y как N_0 является случайной величиной. С помощью введенных показателей может оцениваться число ошибок, оставшихся в ПО - n_0:

$$n_0 = N_0 - n_y$$

В качестве первого приближения для оценок количества ошибок в ПО достаточно знать длину программы в машинописных командах (или операторах языка высокого уровня). Сбор и обработка статистической информации по надежности программного обеспечения показали, что для неотлаженных программных комплексов 1 ошибка приходится в среднем на 1 тысячу машинных команд или 100 операторов языка высокого уровня. В процессе отладки этот показатель улучшается и в отлаженных ПО одна ошибка приходится на $10^4 – 10^5$ машинных команд.

Одной из первых математических моделей, которой пользовались для оценки N_0, была интуитивно найденная зависимость, которой пользовались в фирме IBM для оценки степени отлаженности операционной системы ОС 1360. Исходные данные для этой модели собирались на этапе автономной отладки модулей программного обеспечения. Зависимость числа ошибок, оставшихся в разработанном ПО на момент начала комплексной отладки имела вид:

$N_0 = 23k_1 + 2k_2$, где

k_1 - число программных модулей, в процессе отладки которых было обнаружено и устранено 10 и более ошибок;

k_2 - число модулей, в которых было проведено менее 10 исправлений.

Оценка N_0 может в принципе проводиться и на ранних этапах создания ПО исходя из общих характеристик решаемых алгоритмов. Если анализ этих характеристик позволяет путем прогноза найти или оценить:

η_1 - число простых операторов, используемых в программе;

η_2 - число простых операторов.

Тогда можно воспользоваться метриками Холстеда и оценить длину программы и ее объем. Длина программы через оцененные параметры η_1 и η_2 вычисляется по формуле:

$$\rho = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2$$

Объем же программы связан с ее длиной соотношением:

$$V = p^* \log_2(\eta_1 + \eta_2)$$

Среднее же число ошибок, которое допускает программист в силу своих психологических особенностей:

$$N_0 = \frac{\rho \log_2 (\eta_1 + \eta_2)}{3000} \quad (*)$$

Выражение (*) можно рассматривать как еще одну простую модель для оценки N_0

Еще одной простой моделью, позволяющей оценить N_0 является модель, предусматривающая сбор и обработку информации тестирования одной и той же программы двумя бригадами, использующими независимые наборы тестов. Тестирование выполняется параллельно в течение некоторого заданного периода времени. Пусть n_1 – число программных ошибок, обнаруженных и устраненных первой бригадой, соответственно n_2 – второй.

Число программных ошибок, являющееся пересечением множеств n_1 и n_2 обозначим через n_{12}.

Тогда оценка N_0 может быть получена на основе следующей зависимости:

$$N_0 = \frac{n_{12}}{E_1 * E_2}$$

Где $E_i = \frac{N_i}{N_0}$ – степень оттестированности ПО *i*-ой бригадой.

Подставляя E_i в выражение для N_0 и учитывая что $N_0 \neq 0$, находим:

$$N_0 = \frac{n_1 * n_2}{n_{12}}$$

В литературе описана модель, сеющая предварительные ошибки, которая также позволяет оценить показатель N_0

Модель используется при наличии генератора ошибок, который обеспечивает рассеивание в тестируемую программу некоторого количества известных ошибок. Эти ошибки случайным образом вставляются в программу, а затем предполагается, что ошибки, имевшиеся в программе, и рассеянные ошибки должны быть обнаружены с равной вероятностью в результате тестирования. В процессе тестирования программы в течение некоторого периода времени окажутся обнаруженными и рассеянные, и исходные ошибки. Пусть в результате тестирования обнаружено n_1 исходных ошибок и n_s из S рассеянных. Тогда оценка для N_0 может быть найдена из пропорции:

$$\frac{n_1}{N_0} = \frac{n_s}{S} \quad \rightarrow \quad N_0 = \frac{n_1 * S}{n_s}$$

При таком подходе оценка N_0 может постоянно уточняться в процессе тестирования. Данная модель дает хорошие результаты, если закон распределения рассеиваемых ошибок по программе совпадает с законом распределения исходных ошибок, который неизвестен.

Кроме оценок математических ожиданий величин N_0 , n_y, n_0 в качестве показателей надежности могут рассматриваться распределения вероятностей

типа $P(N_0 = k)$.

Для нахождения такого распределения используется модель, сеющая предварительные ошибки. Но, в данном случае процесс тестирования программы заканчивается в тот момент, когда будут обнаружены и устранены все рассеянные ошибки. Пусть было рассеяно M ошибок и все они были обнаружены, кроме того было обнаружено m исходных ошибок. В качестве математической модели для вычисления распределения вероятностей берется выражение:

$$P(N_0 = k) = \begin{cases} 1, \text{ если } m > k_{обнар} \\ \dfrac{M}{M + k + 1}, \text{ если } m \le k \end{cases}$$

Например, если утверждается, что программа не содержит ошибок ($N_0 = k = 0$), а в ней было рассеяно 9 ошибок и все они были обнаружены и не обнаружено ни одной исходной, то

$P\ (N_0 = k = 0) = 9/10 = 0{,}9$

Для того, чтобы поднять эту вероятность до 0,95 необходимо рассеять и обнаружить 19 ошибок, до 0, 99 уже 99 ошибок.

Вероятность безотказной работы ПО $P(t)$ - это вероятность того, что при заданных условиях эксплуатации в течение заданного интервала времени $[0, t]$ не возникнет программного отказа

$P(t) = P(T \ge t)$

Вероятность отказа ПО – $Q(t)$ вероятность того, что при заданных условиях эксплуатации в течение заданного интервала времени $[0, t]$ произойдет хотя бы один отказ

$Q(t) = P(T < t)$

$Q(t)$ в вероятностном смысле величина обратная $P(t)$, поэтому они связаны соотношением:

$Q(t) = 1 - P(t)$

Частота отказов – $a(t)$ есть плотность распределения времени работы системы до отказа:

$a(t) = Q`(t) – P`(t)$

Интенсивность отказа $\lambda(t)$ выражает интенсивность возникновения отказов, которая характеризует плотность распределения наработки на отказ, рассчитанную при условии, что до рассматриваемого момента времени система работала безотказно:

$$\lambda(t) = \frac{a(t)}{P(t)} = -\frac{d \ln P(t)}{dt} = -\frac{P'(t)}{P(t)}$$

Интенсивность отказов ПО как функция времени ведет себя так же, как в случае этапа приработки технических средств, если при обнаружении программного отказа происходит устранение ошибки. В этом случае:

$\lambda(t) = C\frac{dn_0}{dt}$, где C – коэффициент пропорциональности.

Если же при обнаружении программного отказа устранения ошибки не происходит, что характерно для этапа эксплуатации ПО, то изменения числа оставшихся в ПО ошибок нет, $\lambda(t)$ – постоянная величина, как и в случае этапа нормального функционирования для технических средств.

Наработка на отказ T_0 – среднее время пребывания системы в работоспособном состоянии. Этот показатель не является функцией времени только в случае не устранения ошибок при появлении программных отказов. На этапе отладки можно говорить о среднем времени между отказом после устранения k программных ошибок Tk или о времени между обнаружением i и $i+1$ программного отказа. T_0 и интенсивность отказов связаны соотношением:

$$\lambda(t) = \frac{1}{T_0(t)}$$

Для этапа отладки можно рассматривать в качестве показателя надежности среднее время устранения программной ошибки T_B - обозначая его как аналогичный показатель для технических средств.

В технической и научной литературе очень часто путают $P(t)$ как показатель надежности с еще одним показателем $R(t)$, который является аналогом стационарного коэффициента готовности, если пользоваться его точечной интерпретацией.

Вероятность безотказного выполнения задачи – $R_{ПР}$ – вероятность того, что при заданных условиях эксплуатации ПО на произвольно выбранном наборе исходных данных не произойдет программного отказа. В вероятностном смысле обратная величина $R_{ОТ} = 1 - R_{ПР}$ - есть аналог коэффициенту простоя, используемого для технических средств – есть вероятность того, что на произвольно выбранной комбинации входного вектора исходных данных появится программная ошибка.

Аналитически эти показатели вычисляются следующим образом. Предположим, что каждый $i^{-ый}$ набор входных значений поступает на ПО с вероятностью P_i , некая бинарная переменная y_i принимает значение 1, если программного отказа на данном наборе не будет и значение 0, если отказ произойдет.

Тогда:

$$R_{ПР} = \sum_{i \in I_0} P_i y_i$$

$$R_{ОТ} = \sum_{i \in I_0} P_i (1 - y_i)$$

Где I_0 – множество возможных наборов исходных данных для рассматриваемого ПО.

Именно эти показатели надежности ПО чаще всего оцениваются экспериментальным путем (при испытаниях). Если в результате испытаний обычно пропущено m тестов и в l случаях были зафиксированы программные отказы, то статистические оценки

$$R_{ПР} = \frac{m * l}{m}$$

$$R_{ОТ} = \frac{l}{m}$$

4.3. Проблемы надежности ПО и факторы, влияющие на нее

При оценке и анализе любого технического изделия важно знать, какие факторы вызывают переход его из работоспособного состояния в отказовое состояние. Следовательно, необходимо осуществлять рассмотрение факторов, приводящих к отказам, ранжировать и классифицировать их, искать пути и методы уменьшения влияния этих факторов на процесс функционирования изделия. Разработчики аппаратно-программных комплексов и систем не обращали внимания на надежность программных средств до тех пор, пока в звене "технические средства – ПО" (которое с точки зрения надежности следует рассматривать в простейшем случае как последовательное соединение этих двух элементов) программное обеспечение имело показатели надежности значительно лучшие, чем техническая структура. В силу того, что в последнее время уровень надежности технических средств из-за постоянного совершенствования технологии и схемотехнических решений существенно поднялся, а сложность и объем используемого программного обеспечения в свою очередь значительно увеличились, соотношение уровней надежности технических средств и ПО коренным образом изменилось. Уровень надежности программных средств оказался соизмерим, а в некоторых случаях и ниже, чем надежность технических средств. В этом случае надежность цепочки "технические средства – ПО" определяется в основном надежностью программной части. Это все и обусловило повышенный интерес к методам оценки надежности программных средств, совершенствованию технологии изготовления ПО, решению задач по рациональной организации комплексов программных средств.

В отличие от технических систем, оценку показателей надежности программного обеспечения и рассмотрение факторов, влияющих на эти характеристики, невозможно проводить без учета самой технической структуры, в комплексе с которой функционирует ПО.

Анализ факторов, влияющих на безотказность функционирования программного обеспечения, показал, что важнейшими являются:

1 ошибки проектирования, невыявленные как на этапе автономной отладки модулей, так и в процессе комплексной отладки программных средств;

2 искажения числовой и управляющей информации, возникающие в ПО из-за отказов или сбоев технических средств;

3 ошибки в исходных данных или директивных командах, поступающих в функционирующую систему извне;

4 отсутствие способов и мероприятий в структуре комплекса ПО и технических средствах, предусматривающих сокращение последствий искажения информации из-за отказов и сбоев технических средств и неправильного ввода данных;

5 отсутствие или низкая эффективность средств оперативного обнаружения, локализации и исправления искаженной информации.

Как бы тщательно не отлаживалось программное обеспечение, как много и каких бы тестов не прогонялось в процессе отладки, остается отличная от нуля вероятность того, что в созданном ПО имеются ошибки. Эти ошибки в основном определяются субъективными причинами. Ни один разработчик не может гарантировать того, что все ситуации, возникающие при работе ПО, им предусмотрены и все пути вычислительного процесса проверены и работают так, как это нужно. Множество различных сочетаний входных данных, вероятностный характер использования различных ветвей вычислительного процесса не позволяют говорить об абсолютной работоспособности программного продукта. Эти рассуждения верны и выполняются как для функциональных модулей программных комплексов, так и всего ПО, созданного на базе этих компонент.

А раз так, то в процессе эксплуатации программно – технических комплексов всегда могут появляться ситуации, при которых можно говорить о том, что ПО неработоспособно. Поведение и реакция программно –

технического комплекса в этих ситуациях аналогично тому, что наблюдается при отказе одного или нескольких элементов технических средств. Естественно желание уменьшить число необнаруженных ошибок в программном обеспечении, передаваемом заказчику. Наиболее действенными путями сокращения числа ошибок, оставшихся после отладки ПО, являются:

1 рациональное разбиение всего комплекса программных средств на функциональные модули;

2 более тщательная отладка модулей и всего ПО в целом.

Первое направление позволяет за счет рационального выбора размера модуля функций, реализованных в нем, числа связей между модулями и структуры самого ПО, во-первых, упростить и облегчить процедуры автономной отладки каждого модуля (тем самым уменьшить вероятность наличия в нем ошибок) и, если есть такая возможность, в структуру ПО вводить блоки контроля работоспособности; во-вторых, изменяя структуру самого комплекса ПО наиболее рационально использовать естественную и вводимую функциональную избыточность. С нашей точки зрения этот вопрос (влияние способов разбиения комплексов ПО на показатели надежности ПО), является недостаточно исследованным и первопричина такого состояния дел – отсутствие математических моделей.

Второе направление сокращения числа необнаруженных ошибок в ПО связано с большими затратами временных и материальных ресурсов. Причем, эффективность этого направления существенно определяется опять же рациональностью организации структуры ПО и способами разбиения на функциональные модули. Поэтому вопросам исследования влияния структур ПО на их надежность, формированию требований к характеристикам модулей должно уделяться особо пристальное внимание. На это должны быть направлены усилия специалистов, занимающихся созданием высоконадежных комплексов программных средств.

Вторым фактором, влияющим на показатели надежности ПО, является

искажение обрабатываемой в вычислительных системах информации из-за возникающих в технических средствах отказов и сбоев.

Эффективными средствами борьбы с такими отказами следует считать:

1 создание развитых и полных средств обнаружения и локализации сбойных ситуаций и отказов технических средств;

2 организацию высокоскоростной системы информационного восстановления;

3 использование средств повышения надежности функционирования самих технических средств вычислительных систем.

В настоящее время используемые вычислительные средства обладают низкоэффективными средствами контроля работоспособности. Этому вопросу с теоретической и практической точек зрения уделяется пристальное внимание в общей теории надежности лишь в последнее время. Однако ни в научной, ни технической литературе нет обоснованных и подтвержденных практикой показателей полноты и глубины существующих средств контроля. Некоторые оценочные показатели противоречивы и, как правило, не подтверждаются практикой, поэтому не могут быть использованы при формировании каких-либо рекомендаций и выводов.

Ситуация с обнаружением и идентификацией сбоев даже несколько хуже, чем борьба с последствиями отказов. Это объясняется тем, что сбои в вычислительных системах появляются на один – два порядка чаще, чем отказы, а эффективных методов, средств, систем идентификации сбойных ситуаций нет. Средства борьбы с последствиями отказов и сбоев в вычислительных системах связаны с введением в них информационной, функциональной и временной избыточностей, которые требуют также и аппаратурной избыточности. Задачи оценки надежности систем с такими видами избыточностей из-за "проклятия размерности" не решены в общем виде даже для одних технических средств.

Таким образом, к настоящему моменту не сформулированы четкие направления и рекомендации по уменьшению влияния на надежность ПО

искажений информации, возникающих из-за отказов и сбоев технических средств вычислительных систем.

Борьба с отказами ПО, связанными с ошибками в исходной информации и директивах, вводимых в систему, связана с организацией их жесткого контроля и относится в большей степени к организационным мероприятиям. С технической же стороны, создание средств контроля входной информации на диапазон изменений, на допустимое приращение и т.п. позволяет повысить вероятность получения достоверной информации. Введение такого контроля необходимо (и практически всегда организуется) в системах, к которым предъявляются повышенные требования по надежности.

Последние два фактора, влияющих на надежность, существенным образом определяют среднее время восстановления программных средств. Низкая эффективность средств обнаружения искажений информации, хранящейся в памяти вычислительной системы, может привести к лавинообразному развитию искажений. Потому-то и нужны средства и методы быстрого обнаружения отказового состояния, локализации места искажения и быстрое информационное восстановление. Как и в предыдущих случаях, действенных методов минимизации среднего времени восстановления программного обеспечения в настоящее время не существует. Вопросу восстанавливаемости уделяется мало внимания даже в теории надежности технических систем. Поэтому позаимствовать подходы в решении задач, связанных с восстанавливаемостью технических средств, и реализовать их для повышения надежности ПО было бы рационально.

Между надежностью технических средств и программного обеспечения есть аналогии. Однако надежность ПО в значительной степени отличается от надежности оборудования. ПО прежде всего не подвержено износу, старению и следовательно ему не свойственны постепенные отказы. Внезапные отказы технических средств зависят от времени работы изделия, ошибки же ПО в действительности являются функцией от текущей входной информации и текущего состояния системы. Появление программной

ошибки в некоторый момент времени связано с обработкой такого вектора входных данных, который вызвал путь программы, который был не оттестирован и содержал неустранимую ошибку. Ошибки ПО носят, как правило, системный характер, а отказы и сбои оборудования – случайный характер.

В ПО практически отсутствуют ошибки производства, так как они встречаются весьма редко и достаточно быстро устраняются.

В основе надежности ПО лежит предопределение ошибок разработки, не устраненных в процессе его создания и отладки.

4.4. Методы тестирования программного обеспечения

Высокое качество и надежность программ может достигаться безошибочным проектированием. Методы безошибочного проектирования, а точнее предотвращения ошибок, основываются на применении организационных и методологических правил проектирования комплексов программ, которые определяют организацию технологического процесса проектирования, структурное построение комплексов программ и его компонент, методологию проведения системного анализа, подготовку спецификаций, требований и т.д. Эти методы называют "пассивными". Они способствуют значительному повышению качества программ, но не могут гарантировать удовлетворения всех заданных требований к программному комплексу, а главное, не полностью предотвращают ошибки.

"Активные" методы основываются на выявлении и устранении ошибок. Они дополняют "пассивные" в процессе достижения заданного качества программного продукта и позволяют оценивать ряд показателей качества. Основным "активным" методом является тестирование, которое состоит в проверке программ на соответствие заданным правилам построения и конкретным результатам их исполнения.

Применяемые методы тестирования различаются по ряду признаков, среди которых:

- степень автоматизации процесса тестирования;
- форма представления и исполнения программы при тестировании;
- основные компоненты программы, подлежащие тестированию.

Кроме того, методы тестирования можно объединить в группы:

1) детерминированные, т.е. контролируется каждая комбинация эталонных данных и соответствующая ей комбинация результатов функционирования программы. Это позволяет выявлять отклонение результатов от эталона с конкретным фиксированием всех значений исходных и результирующих данных, при которых это отклонение обнаружено;
2) стохастические, при которых исходные тестовые данные задаются множеством случайных величин с соответствующими распределениями, и для сравнения полученных результатов используются также распределения случайных величин. В результате при стохастическом тестировании возможно более широкое варьирование исходных данных, хотя отдельные ошибки могут быть не обнаружены, если они мало искажают средние статистические значения или распределения;
3) тестирование в реальном масштабе времени, где проверяется исполнение программ и обработка исходных данных с учетом времени их поступления, длительности и приоритетности обработки, динамики использования памяти и взаимодействия с другими программами и т.п.

Выбор методов тестирования зависит от этапа разработки, объема и особенностей тестируемых программ и ряда других факторов. В процессе разработки модулей символьное тестирование используется преимущественно при детерминированных тестах, а стохастическое тестирование применяется при исполнении программ в машинных кодах.

Конечной целью тестирования является получение корректной и надежной программы, функционирующей в машинном коде на ЭВМ.

Нельзя утверждать, что абсолютно корректная программа в одной из форм представления будет столь же корректной в другой форме. Однако, повышение корректности программы в символьном виде способствует повышению корректности ее в объектном представлении, и наоборот. Поэтому созданы две группы методов тестирования:

1) статические, использующие только исходные тексты программ, преимущественно на языках высокого уровня без исполнения той же программы в машинном коде на ЭВМ;
2) динамические, при которых тестируемая программа исполняется на ЭВМ в соответствующем машинном коде, а тексты на языке программирования используются как вспомогательные.

Статическое тестирование учитывает формальный контроль корректности структуры программы, записи и чтения переменных, расчет длительностей исполнения программ и другие виды анализа корректности исходного текста программы. Кроме того, статическое тестирование позволяет планировать по различным критериям эффективное динамическое тестирование с учетом ограниченных ресурсов на его проведение.

При динамическом тестировании основные результаты получаются в процессе исполнения программы в машинном коде той ЭВМ, для которой предназначена программа. Однако, для описания тестов и отладочных заданий, а также для отражения результатов тестирования широко применяется символьное представление на языке, близком к языку программирования. Использование символьного представления при динамическом тестировании обеспечивается соответствующими трансляторами, и значительно упрощает анализ результатов тестирования.

Обратимся к методам тестирования обработки данных. Функционирование любой программы можно рассматривать как обработку потока данных, передаваемых от входа в программу к ее выходу. Входные данные последовательно используются для определения ряда промежуточных результатов вплоть до получения набора выходных данных.

Задача анализа потока данных состоит в установлении корректности их обработки и выявлении ошибок в тестируемой программе.

Последствия ошибок в программе проявляются как малые изменения некоторых переменных в процессе вычислений и как полное искажение или отсутствие на выходе требуемой величины. Тестирование программного модуля целесообразно проводить на упорядоченных наборах данных с учетом степени их влияния на выходные результаты. С этой позиции выделяют следующие частные методы тестирования:

- тестирование при значениях данных, определяющих ветвления в программе и маршруты исполнения программы;
- тестирование корректности записи и считывания переменных при вычислениях и полноты состава входных данных;
- тестирование точности результатов вычислений и корректности обработки каждой переменной;
- тестирование на полное соответствие состава и значений выходных данных программной спецификации.

В приведенной последовательности частные методы позволяют выявить первичные ошибки, которые способны искажать результаты в наибольшей степени. Последовательно акцентируется внимание на выявлении следующих ошибок обработки данных, влияющих на логику и маршруты исполнения программы; записи и считывания переменных; полноты состава результатов; точности расчета выходных данных. На основе таких проверок может оцениваться степень охвата тестированием всех условий, определенных спецификацией.

Глава 5. Тестирование- метод повышения надежности ПО.

5.1. Обеспечение надежности в процессе тестирования

Контроль разработанного ПО включает последовательную цепочку различного рода проверок с непересекающимися целями. Эта последовательность включает:

1 синтаксический и семантический контроль кодирования модулей при их трансляции в машинные программы;

2 единичный тестовый контроль каждого модуля, преследующий цели установления различий между логической схемой модуля, а также его интерфейса с внешними спецификациями системы;

3 функциональное тестирование, преследующее цели обнаружения противоречий между программой и функциями, определенными для реализации в системе внешними спецификациями;

4 системное тестирование, имеющее целью выявление противоречий между системой и целями, определенными для се создания пользователем;

5 приемо-сдаточные испытания:

6 установочные испытания.

Рассмотрим компоненты представленного выше перечня с учетом их значения для контроля создаваемой системы ПО.

5.2. Функциональное тестирование

Одним из качеств, характеризующих программиста, является его способность находить свои собственные ошибки. При этом начинающие программисты обычно слабо подготовлены в вопросах обнаружения и исправления ошибок, в то время как опытные программисты делают эту работу без особых усилий.

Если программа неверно работает, то она должна отлаживаться. Следовательно, отладка всегда должна начинаться с некоторых доказательств

неудачности программы. Если кажется, что программа работает верно, то она должна быть подвергнута контролю тестами. Часто после тестирования программа модуля возвращается на стадию отладки для устранения ошибок.

Таким образом, тестирование программ одиночного модуля позволяет узнать о наличии ошибок, а отладка призвана определить причину этих ошибок. Поэтому отладка и тестирование представляют собой две различные частично перекрываемые стадии контроля. В этой связи затраты времени на тестирование должны быть разделены на отладку и собственно тестирование одиночного модуля с тем, чтобы подчеркнуть, что обе стадии нуждаются в выполнении.

Отладка. Нередкой ошибкой в программировании, особенно среди начинающих программистов, является желание выполнять только отладку и не проводить тестирования. Так, если программа хорошо работает с группой осторожно подобранных данных, то они верят, что программа и дальше будет так же работать с любыми данными. Наступает момент искреннего удивления, когда после использования результатов нескольких прогонов программы, в верности которых у них нет сомнения, получают явно неверный результат.

Обычно существуют два подхода к отладке: либо большая часть времени тратится на то, чтобы выявить ошибки вручную, либо выполнить эту работу с использованием ЭВМ. Выбор альтернативы в значительной мере зависит от наличия машинного времени.

Существует и еще один подход, при котором отладка частично пересекается с написанием программ. Некоторые программисты предпочитают написать несколько строк кодов и тут же проверить их работу. Этот подход позволяет отыскать ошибки кодирования непосредственно в процессе кодирования. Достоинством такого подхода является и то, что программа свежа в памяти, что позволяет легче выявить ошибки.

Отладка начинается с момента компиляции программы, так как обнаружение компилятором синтаксических и частично семантических

ошибок является одной из стадий отладки. Большинство ошибок обнаруживается и исправляется именно на этой стадии контроля. Начинающий программист убежден, что при компиляции обнаруживаются все ошибки. Более опытный программист знает, что наиболее коварные синтаксические ошибки не будут обнаружены компилятором.

Если синтаксическая ошибка определяется как "все, что противоречит спецификациям языка", то многие ошибки при компиляции не будут замечены (например, неопределенные переменные, вход внутрь цикла, выход за границы описания). Большинство синтаксических ошибок определяется языком программирования и используемой машиной, и по этой причине мало что можно сказать об исправлении каждой конкретной синтаксической ошибки.

Отладка начинается с момента, когда все системные сообщения компилятора об ошибках синтаксиса рассмотрены и действительные синтаксические ошибки исправлены.

Первый шаг в отладке - точная локализация ошибки. Одним из наиболее общих случаев трудности отладки является отсутствие выходной информации об ошибке. Чтобы оценить достаточность тестовых данных, их необходимо проверить. Для обеспечения достаточности данных, возможно, потребуется дополнительное включение тестовых вариантов.

Следующим шагом является развитие одной или нескольких теорий об ошибке и составление плана подтверждения выдвинутых утверждений. Суть плана состоит в попытке доказать выдвинутую теорию. При плане, включающем несколько шагов реализации, необходимо организовать проверку каждого шага. В случае, когда точное местоположение ошибки обнаружено, необходимо сделать дополнительные проверки перед ее исправлением. Целью дополнительных проверок является полнота определения ошибки, ее взаимосвязь с другими участками программы и возможность порождения после исправления других ошибок.

Одной из причин пропуска ошибок при отладке является ситуация,

когда контролирующий видит то, что хочет видеть, а ни то, что есть в действительности. Важно вести проверку с закрытыми комментариями, т.е. проверять код программы, а не его интерпретацию - комментарии.

Когда точное местонахождение ошибки определено, следующим шагом является ее исправление. Наибольшей неприятностью, встречающейся при исправлении ошибки, является не полное устранение всей ошибочной ситуации, а только ее внешних признаков. Попытка быстрого устранения ошибки путем экспериментирования без достаточного всестороннего анализа впоследствии может привести к более сложной и запутанной ошибке.

В силу своих особенностей исправление ошибки всегда имеет некоторый отрицательный эффект на структуру программы, ее читаемость и четкость. Из практики известно, что исправление почти в 50% случаев порождает дополнительные ошибки в программе. Поэтому отладку целесообразно поручать наиболее подготовленным программистам проекта с обеспечением документирования процесса.

Основной проблемой средств, автоматизирующих отладку, является то, что они заставляют программиста, работающего на языке высокого уровня, быть экспертом в деталях машинного языка. При этом проводится в жизнь идеи полной независимости программиста от машины, т. е. защита программиста от всех машинных деталей. Одним из путей решения этой проблемы является проектирование всех вспомогательных средств таким образом, чтобы они связывали программиста только с терминами языка программирования высокого уровня и не касались понятий машинного языка.

Тестирование модуля. Целью тестирования модуля является обнаружение различий между логической схемой модуля и его интерфейсом и их внешними спецификациями. Тестирование модуля начинается после его отладки.

Тестирование модуля может проводиться произвольным образом, без определенного плана. Другой, научный подход, когда тестирующий

программу с осторожностью выбирает необходимую тестовую информацию, внимательно составляет набор вопросов для тестирования и тщательно проводит сам тест.

При использовании произвольного подхода к тестированию модуля невозможно избежать тестирования одного и того же участка по два и более раза, нельзя оценить, насколько всеобъемлющим является тестирование, и определить момент его завершения.

Тестирование модуля включает проведение определенного объема работ: проектирование набора тестовых комбинаций на основе анализа внешних спецификаций и программы модуля, написание программы тестирования и ее проверку, выполнение программы тестирования. Рассмотрим содержание этих работ.

Проектирование программы тестирования – процесс творческий, требующий аналитического мышления. Однако существует определенный набор простых правил, которые позволяют получить разумное множество промерочных тестов. Эти правила базируются на рассмотрении программы модуля как "черного ящика" и использовании внешних спецификаций модуля для построения тестов, далее на базе изучения программы модуля строятся дополнительные тесты. Эти действия выполняются за четыре шага.

1. Используя внешние спецификации модуля, необходимо получить проверочные тесты для каждого условия и варианта данных, границ всех заданных интервалов на входе и выходе и для неверных условии.
2. Проверяется код модуля с тем, чтобы убедиться, что все условные переходы будут выполняться по каждому из возможных направлений разветвления. Если необходимо, то добавить проверочные гесты там, где это требуется.
3. Проверяется код модуля с тем, чтобы убедиться, что возможно столько путей, сколько тесты смогут охватить. Например, оператор цикла должен контролироваться с помощью тестовых данных для

значений переменных циклов 0,1 и максимального значения.

4. Проверяется код модуля на чувствительность к определенным комбинациям входных значений, и добавляются, если это необходимо, дополнительные тестовые варианты.

Первый шаг программы тестирования предполагает рассмотрение модуля в качестве "черного ящика" и получает проверочные тесты путем обработки входных данных модуля Если модуль содержит целое число различных входных значений, то необходимо разработать тестовые комбинации для каждого из них. При некотором множестве входных параметров, каждый из которых имеет в свою очередь некоторое число действительных значений, необходимо разработать тестовые комбинации для каждой совокупности.

Более усложненный вариант выбора тестовых комбинаций будет иметь место тогда, когда входные данные модуля принимают значения из широкого диапазона. Одним из решений этой проблемы будет получение тестовых комбинаций на границах вводимых интервалов. Рассмотрим несколько частных рекомендаций. Так, если входное значение берется на интервале от –1,0 до +1,0 то, кроме всего прочего, необходимо обязательно провести проверку граничных значений. Желателен также контроль на комбинацию 0,0 для обнаружения таких ошибок, как индексация или деление на ноль. Если используемая ЭВМ имеет два представления нуля, т.е. –0,0 и +0,0 то необходимо провести проверку каждого возможного случая.

Ряд программ имеет также функциональные границы, порождающие большое количество тестовых комбинаций. Предположим, что необходимо проверить модуль, который выполняет контроль вводимых записей. Функциональные границы проявляются в том случае, когда введенный список пуст, содержит только одну запись, введенные записи уже рассортированы и когда все введенные записи имеют равные значения. По каждому из возможных вариантов следует выполнить проверку и принять решение.

Обязательным условием тестирования является назначение тестовых комбинаций для недействительных условий. Недействительные вводы, используемые в программах тестирования, обычно лежат за тряпицами действительных входных данных.

Три завершающих шага назначения тестовых комбинаций базируются на проверке логики модуля. Для выбираемого множества тестовых комбинаций в качестве критерия полноты может выступать утверждение о том, что каждая команда контролируемого модуля выполняется хотя бы один раз. Этот критерий необходим, но он не всегда является выполнимым. Лучшим критерием является обеспечение достаточного количества тестовых условий, позволяющих для каждой проектируемой конструкции модуля, которая порождает многовариантный переход, прохождение каждой ветви. Для автоматизации выполнения контроля лучшим решением будет построение граф – схемы программы модуля и на ее основе получение необходимых тестовых комбинаций, обеспечивающих работу по шагам два и три общего подхода.

Последним шагом проектирования тестовой программы является проверка логической схемы модуля на чувствительность к различным входным характеристикам. Здесь же необходимо предусмотреть контроль границ обрабатываемых массивов.

Получение полного набора тестовых комбинаций для модуля может показаться трудоёмким, но никто и никогда не утверждал, что проверка легко осуществима. Следуя перечисленным выше шагам выбора тестовых комбинаций, придем к принципиально спроектированной программе тестирования с высоким уровнем обнаружения ошибок. Этот процесс займет больше времени, чем случайный процесс подбора тестовых комбинаций, но если он ведет к возможности обнаружения одной – двух дополнительных ошибок, то стоит пренебречь потерей времени скрупулезного подбора программы тестирования.

Очередным этапом тестирования является написание и контроль

проверочных программ. Физическая форма проверочной программы в определенной мере диктуется методом проведения интеграции, рассмотренным ранее. Сам процесс написания проверочных программ прост и сводится к написанию небольших программ вызова проверяемых модулей, подключению их на пропуск тестовых комбинаций и документированию результатов пропуска. С целью уменьшения объемов выдаваемых результатов признается лучшим вариантом программирование контроля выходных результатов в программе формирования обращения к модулю там, где это возможно.

В идеальном случае проверочная программа должна быть проконтролирована перед тем, как использовать ее для тестирования. Это выполнимо только разработчиком, создавшим такую программу, однако ясно, что такие программы в принципе могут содержать ошибки.

Критической частью работы по тестированию модуля являются пропуск проверочной программы и проверка получаемых результатов. Общей ошибкой проектирования тестовых программ являются большие затраты времени для обнаружения ошибок, а также то, что до получения действительного вывода важно получить ожидаемый вывод каким-нибудь возможным способом.

Существует много типов ошибок, которые не могут быть обнаружены в результате рассмотрения входных характеристик модуля. Ряд ошибок может быть классифицирован как ошибочные побочные эффекты, определение которых затруднено. Единичную проверку модуля лучше выполнять не непосредственному разработчику, а лицу, которое разрабатывало модуль, вызывающий этот проверяемый модуль.

После завершения тестирования модулей (может быть, некоторой совокупности модулей) производится их интеграция с параллельной проверкой межмодульного интерфейса.

Целью внешнего функционального тестирования является обнаружение противоречий между программой и ее внешними спецификациями.

Предварительным условием, определяющим качественный исход функционального тестирования, являются полнота, точность и непротиворечивость внешних спецификаций. Если это условие не будет выполнено, то функциональный тест может быть реализован, но программа останется ошибочной.

При проектировании функционального теста внешнюю спецификацию рекомендуется разделить на частные внешние функции, а затем тщательно проверить каждую из них. Для этих целей могут использоваться подходы, рассмотренные для случая тестирования единичного модуля. Они могут быть распространены на все входные условия и режимы, а также на все границы входных и выходных потоков данных. Эти способы обеспечивают также проверку программы на предмет ее поведения на функциональных границах и при появлении недействительных (ошибочных) или неожиданных данных на входе.

Как и в случае тестирования единичного модуля, каждый тест должен быть документирован с тем, чтобы при необходимости иметь возможность повторить его. В этом есть прямой смысл, так как возможные изменения на высшем уровне контроля могут потребовать повторной проверки всех низших уровней проверки.

Функциональное тестирование представляет собой проверочный процесс, выполняющийся обычно не в рабочих условиях. Эта проверка может выполняться до завершения разработки всей системы. Например, не обязательно иметь налицо предусмотренную проектом систему ввода-вывода, а достаточно написать программу, моделирующую ее работу.

Критерием минимума для функционального тестирования является выполнение всех переходов, по крайней мере, однажды во всех направлениях.

Процесс объединения крупных систем ПО начинается с создания плана построения, подробно описывающего, как, кем, где и когда система будет построена из ее составных частей. Основная цель плана построения состоит в

том, чтобы рационально определить процесс создания системы. К каждой версии или уровню системы обращаются как к некоторой составной части. Нулевая составная часть определяет "пустую" версию системы, на основе которой строятся (наращиваются) все последующие составные части. Для вновь создаваемой прикладной системы в качестве нулевой составной части выступает операционная система. В качестве последней составной части выступает полная система проектируемого ПО.

План построения определяет номера составных частей, их содержание, графическое построение и функциональные характеристики. В идеальном случае каждая очередная составная часть отличается от предыдущей на один модуль. Такой подход в полном объеме реализует понятие непрерывной интеграции системы, а это позволяет всегда выполнять тестирование легче, позволяет вернуться к любым новым функциям, после подсоединения которых система ведет себя неправильно. Описываемый подход был использован при создании ОС/360.

Интеграция системы невозможна без решения задачи управления. Группа, проводящая работы по интеграции, должна знать заранее, что содержится на каждом уровне системы. Единственный допустимый способ внесения изменений в систему – это внесение через интеграцию или путем формального процесса исправления ошибок. Все исправления ошибок должны иметь форму замены всего модуля новым. Другие типы изменений, такие, как подпрограммы исправлений в объектном или исходном коде, должны быть запрещены. При обнаружении ошибки группа интеграции должна удалить весь модуль, а может быть даже их некоторую связанную совокупность, и предупредить об этом разработчика.

В обязанности группы интеграции входит также поддержание текущего уровня системы, утверждение всех изменений, построение каждого уровня, документирование использования, доставка каждого готового уровня разработчиком. После создания последнего уровня система готова для испытаний.

Группа разработки работает одновременно с группой интеграции, т.е. завершает разработку очередных модулей и их тестирование. Завершенные компоненты передаются для интеграции. Когда готов последний уровень системы, группа разработчиков должна провести функциональное тестирование системы. После появления очередной версии системы группа разработки должна изъять все предыдущие версии из употребления. Это обеспечивает использование всеми заинтересованными организациями последней версии системы.

5.3. Системное тестирование

Целью системного тестирования является выявление противоречий между разработанной системой и первоначальными целями ее создания. Компонентами системного тестирования являются разработанная система ПО, конечные цели и вся документация, прилагаемая к системе. Внешние спецификации, составляющие основу функционального тестирования, при системном тестировании не играют никакой роли.

При изложении целей функционирования в свое время подчеркивалась необходимость их измерения. Измеримые цели способствуют руководству процессом проектирования: ясно, что если цели поставлены в таких выражениях, как "система должна быть быстродействующей, надежной и безопасной", то не существует способов проверки соответствия системы этим целям.

Системное тестирование может быть проведено в условиях, приближенных к реальной ситуации пользователя. Однако из этого общего правила могут быть исключения, диктуемые причинами объективного характера. Кроме того, определенные типы систем ПО экономически нецелесообразно испытывать в реальных условиях. В этих случаях системное тестирование проводится в условиях, имитирующих реальные.

Проектирование системных тестов. Системное тестирование больше, чем все остальные типы тестирования, рассмотренные до сих пор, требует творческого подхода. Проектирование эффективных системных тестов требует больше изобретательности, чем даже проектирование самой системы. Не существует простых руководств для проведения тестирования, однако рассматриваемые далее рекомендации дают представление о тех основных видах тестов, которые могут быть необходимы.

Испытание на предельные нагрузки. Общим недостатком больших систем ПО является то, что при малых или умеренных нагрузках они работают нормально, а большие нагрузки, возникающие в реальных условиях эксплуатации, могут вызывать ошибки в работе. Под такими нагрузками

понимается концентрация заявок к элементам операционной системы (переполнение очереди или задача о "нетерпеливом клиенте"), системе управления данными и к другим ресурсам ЭВМ. Характерной особенностью подобных ситуаций является наличие "узких мест" в системе обработки в короткие промежутки времени (стрессовая ситуация).

Другим примером испытания на предельные нагрузки является случай, когда система обрабатывает многотомный файл, размещенный на нескольких магнитных носителях. Здесь важно ввести такое количество данных, чтобы проверить переход с одного тома на другой. Другими примерами испытаний на ограничения являются компиляция большой исходной программы, подготовка и вывод документа по полному предполагаемому объему в то время, когда в обычной ситуации документы не достигают предельных размеров, и т.д.

Целью проверки системы на "узкие места" (стрессы) является испытание ее при чрезмерной нагрузке. Этим типом испытаний чаще пренебрегают, так как персоналу, ответственному за разработку ограничений, кажется, что такие высокие нагрузки не встретятся в реальных условиях, но это далеко не всегда так.

Испытание оборудования. Ряд систем типа операционных включает разнообразные комплексы оборудования, выполненного как аппаратно, так и программно. Желательно, чтобы это оборудование было проверено в совместной работе с создаваемой системой ПО, т.е. система должна быть испытана с каждым аппаратным устройством и с каждой программой, с которой она взаимодействует. Если пользователь выбирает для установки только определенное оборудование и режимы его работы, то испытанию совместное разрабатываемой системой подвергаются именно эти объекты.

Испытание памяти. Разрабатываемые системы ПО могут иметь определенные ограничения по используемой основной и вспомогательной памяти. Тесты должны быть составлены так, чтобы доказать, что система не удовлетворяет требованиям к ней в этом отношении.

Испытание совместимости. Большинство разработанных систем является не абсолютно новыми, а некоторыми версиями ранее созданных систем. В этом смысле интерфейс с пользователем в предыдущей системе должен быть по возможности сохранен и для новой системы. Основной целью испытаний на совместимость является выявление несовместимости.

Испытание на бессбойность. Среди целей функционирования обычно фигурируют цели для характеристик бессбойности и ремонтопригодности систем. Обслуживающие средства системы типа программ вывода, программ фиксации данных о ходе выполнения программы, программы диагностики и т.д. должны быть проверены. Внутренняя логика системы должна быть изучена, чтобы обслуживающий персонал мог быстро и точно указать причину ошибки при выдаче только некоторых ее признаков. Все процедуры ремонтопригодности также должны быть испытаны.

Испытание на восстанавливаемость. Составной частью целей функционирования таких систем, как операционные, системы управления базой данных, системы передачи данных, являются функции восстановления, например восстановление потерянных данных в базе данных. Наличие таких функций улучшает эксплуатационные характеристики систем.

Испытание на безопасность. Многие системы ПО по разным причинам должны обладать функцией защиты информации. Например, при совместной работе в основной памяти нескольких программ необходимо обеспечить исключение их взаимного влияния. Цель испытаний – разрушить безопасность системы.

Испытание производительности. Цели производительности и эффективности, такие, как время реакции, производительность при различных нагрузках и работе различного оборудования, являются важной частью требований к большинству систем ПО. Цель испытаний – доказать, что заданная производительность и эффективность не достигнуты.

Испытание на надежность и готовность. Одно из основных испытаний системы, имеющего цель доказать, что не достигнут заданный

уровень надежности, т.е. не достигнуты заданные показатели наработки на отказ, – защитные функции обнаружения и исправления или допуск ошибок программного или аппаратного обеспечения. Испытание на надежность достаточно трудно проводить, но тем не менее необходимо проверить все заданные исходные цели по этому показателю. Если целью системы является некоторое заданное количество часов наработки на отказ, то необходимо прежде всего организовать наработку заданного времени и проработать большее количество часов с тем, чтобы убедиться в удовлетворительных свойствах системы.

Одним из приемов испытания на надежность и готовность может служить ввод программных и аппаратных ошибок в систему с последующей проверкой защитных свойств по обнаружению или допуску ошибок. Показать, что система содержит больше ошибок, чем допускается данным показателем надежности, практически трудно, так как если бы было можно подсчитать количество оставшихся ошибок, то можно было бы их и исправить.

Испытание документации. Проверка полноты и правильности документации пользователя является важной частью системного тестирования. Все тестовые комбинации должны разрабатываться только с использованием документации пользователя.

Испытания переносимости. Большие системы ПО достаточно сложно адаптируются при переносе на другие установки. Испытание переносимости и размещения необходимо, так как здесь часто имеет место одна из обескураживающих ситуаций, с которой сталкиваются пользователи – система попросту оказывается неработоспособной.

Испытание средств взаимодействия с пользователем. В ходе системного тестирования данный тип проверки должен быть проведен как контроль выполнения поставленных целей разработки, но он не может играть решающей роли, так как уже поздно исправлять недостатки в этом направлении. Однако второстепенные задачи могут быть выявлены и

решены. Например, можно еще потребовать устранить такие недостатки, как неудовлетворительное отражение в документации сообщений, выдаваемых системой.

Приведенные рекомендации требуют, чтобы ход мысли испытателя и пользователя совпадал, а для этого необходимо тонкое понимание того, как будет использоваться система. Из этого вытекает вопрос о том, как будет проводиться тестирование и, в частности, кто будет разрабатывать тестовые комбинации. Системное тестирование должно проводиться не программистами и не организацией, отвечающей за разработку системы. Группа, проводящая тестирование, должна быть самостоятельной организацией, включающей в себя экспертов по тестированию, возможных пользователей системы, аналитиков и проектировщиков системы, может быть даже психологов. Очень важно, чтобы в группу входили проектировщики данной системы. Причем это не нарушает тезиса о том, что сам проектировщик не должен испытывать свою систему, так как с того момента, как он окончил свою часть работы, система прошла через многие руки. В действительности он испытывает не свою собственную систему, а только пытается найти противоречия между окончательной версией системы и своим первоначальным замыслом.

Системное тестирование – специфическая и ответственная операция, так что в будущем, возможно, будут созданы подразделения, специализирующиеся на системном тестировании ПО, разработанного другими организациями.

Как упоминалось выше, компонентами системного тестирования являются первоначальные цели, документация и публикации пользователя и сама система. Все тесты для системы должны быть составлены с использованием публикаций пользователя в качестве исходного материала, а не внешних требований к системе. Внешние требования должны быть использованы только для того, чтобы разрешить выявленные противоречия между системой и ее публикациями.

Тесты и последовательность испытания системы часто составляют как сценарий, представляющий собой совокупность последовательных действий. В связи со сложностью тесты состоят из нескольких частей: самого сценария, введенных данных и предполагаемого результата. Сценарий точно сообщает испытателю, какие операции надо выполнять в ходе тестирования.

Проведение тестирования. Одним из методов системного тестирования является испытание системы в эксплуатационных условиях на вычислительных центрах одной или нескольких организаций пользователя. Это выгодно для всех сторон: организация-разработчик уведомляется об ошибках проектирования и программирования, которые она могла не обнаружить, а организация-пользователь получает возможность изучать систему и экспериментировать с ней до того, как она будет официально передана.

Другой полезный метод – использовать систему в рабочем режиме до того, как она будет официально использована. Это может быть сделано не при всех, но при многих обстоятельствах. Например, изготовитель вычислительных машин может установить новую операционную систему в рабочем режиме в своем вычислительном центре перед тем, как официально использовать ее, а группа по программному обеспечению может использовать свои собственные компиляторы, программы обработки данных и т.д.

При системном тестировании наблюдается тенденция начинать испытания с простых тестов, кончая более сложными. Такая последовательность не рекомендуется, так как системное тестирование приходится на самый конец цикла создания системы и в ходе его возникает проблема времени для отладки и исправления ошибок. Так как при реализации сложных тестов часто обнаруживаются ошибки, которые трудно исправить, следует поменять последовательность – начать со сложных тестов и закончить простыми.

Зависимости между исправлением ошибок и временем наработки носит

экспоненциальный характер. Первая зависимость, показывающая, что стоимость исправления ошибки резко возрастает на последних этапах цикла разработки, должна быть очевидна для большинства читателей. Вторая и менее известная зависимость тоже определяется этой кривой. Вероятность неправильной фиксации замеченной ошибки тоже возрастает на последних стадиях. Этот случай представляет собой известный интерес, так как неправильная фиксация причиняет больше вреда, чем пропуск ошибки. Отсюда вытекает вопрос: может быть целесообразно намеренно не исправлять ошибку?

Перед тем как ответить на этот вопрос, рассмотрим следующую ситуацию. Предположим, что любая ошибка может быть ранжирована в интервале от 1 до 10 при ее попадании к пользователю. Заметим, что интервал ошибки и ее размер в виде числа операторов, которые должны быть исправлены, прямо не взаимосвязаны. Допустим, что исправление любой ошибки во время проведения теста имеет вероятность 0,2 быть не зафиксированным и вероятность 0,3 сделать новую ошибку, и эта новая ошибка имеет случайный интервал от 1 до 10. Отсюда следует, что лучше сознательно отказаться от исправления ошибок с левой границы интервала, так как их исправление ухудшило бы работу системы.

Такое заключение трудно понять. С одной стороны, проектировщик хотел бы сказать, что спроектированная им система не содержит известных ошибок, но, с другой стороны, в ней можно усмотреть описанный случай. Чем лучше конструкция системы, тем, разумеется, менее ощутим этот возможный момент, но нельзя сказать, что он исчезает совсем даже в системах с совершенной конструкцией.

Отсюда вытекает, что управляющее решение должно быть принято для каждой незначительной ошибки, обнаруженной во время системного тестирования системы и других более поздних форм тестирования: исправлять ли ошибку или отложить ее исправление до будущих версий системы. Ключ к решению этой проблемы – ранжирование ошибки. Мнение

программиста по поводу интервала ошибки часто расходится с мнением пользователя. Любое решение – оставить ошибку или исправить ее – должно быть основано на тонком понимании нужд пользователя.

Проектирование и управление. Весь процесс проектирования, кодирования, единичного и функционального тестирования настраивает на то, что во время системного тестирования не может быть обнаружено ошибок. В связи с этим расчет ресурсов для проведения испытаний производится в предположении, что тесты пройдут безошибочно с первого раза. Это приводит к тому, что имеют место грубая недооценка потребных ресурсов, включающих людей, машинное время, графики выполнения работ, и снижение качества тестирования из-за недостатка ресурсов.

Как и в любом сложном деле, план проведения работ является важным компонентом процесса тестирования. Этот документ должен быть разработан задолго до начала тестирования и иметь по минимальной оценке следующие основные характеристики.

1. Цели тестирования должны быть определены по каждой из фаз проверки.
2. График работ с расписанием ответственности. Необходимо определить, кто разрабатывает тестовые комбинации, схемы тестирования, выполнение на каждой фазе проверки, график выполнения работ.
3. Инструменты проведения работ. Описать необходимые инструменты для проведения работ, а также то, как, кто и когда их использует.
4. Машинное время. Определить количество потребного машинного времени, а также то, когда, кем и кому это время выделяется.
5. Конфигурация вычислительной системы. Необходимо дать описание потребной конфигурации вычислительной системы, а также специального оборудования для проведения испытаний. Отражается информация по графику представления оборудования и

распределению ответственности.

6. Комплектование тестов. Необходимо определить требования к тестовым комбинациям, стандарты на их составление, порядок их разработки и хранения, а также распределение ответственности.
7. Процесс отладки. Определить единую стратегию к сообщениям об ошибках и методах их исправления.
8. Критерии проведения работ. Определить критерии проведения работ по каждой фазе тестирования с целью оценки уровня достаточного тестирования по каждой из фаз.
9. Последовательность интеграции. В силу того, что интеграция системы производится в процессе тестирования, необходимо определить последовательность и методику проведения работ, график и ответственность за его выполнение.

Наиболее трудной для исполнения позицией плана работ является определение критерия завершения тестирования. Обычно планируют, что 100% функциональных испытаний пройдут успешно. Это достаточно слабый критерий, так как он не отражает глубину и нагрузку тестирования. Полагаться же только на добросовестность разработчиков тестов и лиц, проводящих тестирование, по меньшей мере нелогично.

Для обнаружения максимального числа ошибок применяется упоминаемый ранее негативный критерий, цель которого – найти ошибку Лучше, если критерий будет сформулирован в виде числа ошибок, которые должны быть обнаружены, а не в виде ничего не значащего процента тестов, завершающихся успешно (безошибочно). Если при проведении тщательного испытания не было обнаружено ошибок, т.е. не удовлетворен критерий, то требуется провести детальное изучение всего подхода к тестированию

Этот же принцип может быть использован для определения момента, когда испытание системы можно считать завершенным. В качестве варианта формулировки критерия можно предположить следующий текст "тестирование завершается тогда, когда истекло время его проведения по

графику и обнаружено, по крайней мере, некоторое количество ошибок".

Таким образом, проектирование испытаний заключает в себе некоторый парадокс: с одной стороны, нежелательно находить много ошибок, с тем чтобы удовлетворить график; с другой стороны, цель тестирования – найти как можно больше ошибок Все это еще раз подчеркивает рассмотренные ранее идеи: критерий завершения тестирования должен быть выражен негативно, а тестирование системы должно проводиться специализированными организациями, не испытывающими никакого влияния со стороны разработчиков.

5.4. Приемо-сдаточные испытания

Целью приемо-сдаточного тестирования является проверка удовлетворения первоначальных требований, предъявляемых к системе. В отличие от предыдущих видов тестирования (функционального и системного) этот вид контроля осуществляется пользователями. Пользователь разрабатывает некоторый набор тестов с целью доказать, что система не удовлетворяет требованиям на разработку. Если же эти тесты не завершились успехом с точки зрения пользователя, то система формально принимается для использования

Приемо-сдаточные испытания включают лабораторные испытания, при которых специально готовятся тесты, экспериментальные эксплуатационные испытания, при которых система утверждается на экспериментальной основе, и эксплуатационное испытание, при котором система параллельно эксплуатируется вместе с существующей системой.

После завершения приемосдаточных испытаний система передается пользователям и может проходить так называемые установочные испытания. Большие программные системы имеют сложную процедуру установки на вычислительных центрах пользователей, где должен быть выбран режим эксплуатации, размещены файлы и библиотеки системы. Цель установочных испытаний – выявить ошибки, сделанные в процессе установки.

План проведения установочных испытаний вырабатывается разработчиком, а реализуется пользователем. Особенностью тестов этого вида является то, что они должны проверять наличие всех необходимых файлов и аппаратуры, содержание первой записи па каждом файле, наличие всех составных частей системы ПО. Кроме того, все программы системы должны быть снабжены небольшим количеством тестов для обнаружения ошибок установки.

Завершая краткое изложение вопросов тестирования с целью достижения заданного уровня надежности, нельзя не отметить еще раз необходимость серьезного отношения к проведению контроля, так как далее

предстоит процесс сопровождения созданного ПО, где невыявленные элементы ненадежности могут проявить себя самым неожиданным образом.

Глава 6. Математические модели надежности программного обеспечения.

6.1. Общий подход к построению математических моделей надежности ПО

Выбор и обоснование любой математической модели надежности ПО должен базироваться на статистике программных отказов, которая дает достаточно полное представление о характере и особенностях протекания процесса создания и отладки программы. Для фиксированных условий создания ПО (степени сложности программы, уровня квалификации программистов, длительности отладки и т.д.) само развитие и протекание процесса устранения программных ошибок во времени подчиняется некоторой, вполне определенной закономерности, которую и нужно выявить. Выявить можно, если провести выравнивание полученного ряда наблюдений. Выравнивание обычно состоит из:

1 подбора типа кривой, форма которой соответствует характеру изменения данного ряда наблюдений;
2 определения численных значений параметров выравнивания кривой.

Полученная таким образом кривая есть математическая модель, которая позволяет не только описывать исследуемый процесс по результатам наблюдений в прошлом, но дает возможность осуществить прогнозирование развития исследуемого процесса в последующие моменты времени.

На практике для выравнивания экспериментальных данных применяют достаточно простые функции в виде полиномов различной степени, экспонент или их суперпозиций. Выбор вида функции прежде всего должен быть основан на анализе характера изменений исследуемого ряда наблюдений и самого процесса по существу, хотя можно воспользоваться и рекомендациями, основанными на анализе поведения формальных показателей поведения ряда наблюдений.

Так, например, для выбора в качестве математической модели модифицированной экспоненты достаточно убедиться в линейном характере изменений показателя $Ln\ U_t$, представляющий собой логарифм среднего прироста (или уменьшения) с постоянным темпом первых разностей рассматриваемого ряда экспериментальных данных.

Как показывают результаты проведенных исследований, при подборе кривой для описания процесса детерминированной и статистической отладки программ наблюдается именно такой характер поведения собираемых данных.

Точности полученных с помощью математических моделей оценок показателей надежности ПО во многом определяются объемом и точностью исходных данных, на базе которых были получены параметры этих моделей. Исходные данные собираются в процессе автономной отладки модулей, комплексов детерминированного и статистического тестирования или при эксплуатации программного продукта.

Чаще всего собирается статистика $\{ T_1, T_2, \ldots T_k \}$ интервалов времени работы программного продукта между моментами проявления смежных программных отказов. Эта статистика позволяет строить экспериментальные зависимости вида:

$$T_k = f_1(k), \qquad T_k = f_2(\tau),$$

$$\lambda_k = 1/ T_k = f_3(k), \qquad \lambda_k = 1/ T_k = f_4(\tau),$$

где

k – число обнаруженных и устраненных программных ошибок к рассматриваемому моменту времени,

τ – длительность тестирования или эксплуатации.

В некоторых случаях вместо статистики времени между отказами собирают данные о числе прогонов тестов или реализаций ПО между проявлениями программных отказов. В этом случае имеем статистику $\{ n_1,$

$n_2, \ldots n_k$ }, где n_k – число успешно реализованных тестов после устранения в ПО $i\text{-}1^{\text{-ой}}$ программной ошибки до проявления $i^{\text{-го}}$ программного отказа.

Для любого ПО существует и известно среднее время решения задачи $t_{ср}$, поэтому статистики { Ti } и { n_i } могут быть пересчитаны с помощью соотношения:

$$n_i = \frac{T_i}{t_{ср}}$$

Иногда в процессе тестирования выделяют интервалы времени, на которых фиксируют число успешных прогонов тестов и прогонов, на которых проявились программные ошибки.

Используя статистику { n_i } могут быть построены экспериментальные зависимости вида $P_k(t) = f_5(k)$; $P_k(t) = f_6(\tau)$, где

k – число устраненных программных ошибок за время тестирования τ

Исходные статистики ограничены. Использование этих статистик для оценки параметров математических моделей целесообразно, если они обеспечивают устойчивость получаемых оценок. Перед использованием полученной статистики для решения задачи оценки параметров математической модели целесообразна проверка её на достаточность. Параметры математической модели будут статистически устойчивы, если исходные данные удовлетворяют следующему неравенству:

$$\frac{\sum_{i=1}^{k} T_i (i-1)}{\sum_{i=1}^{k} (i-1)} < \frac{\sum_{i-1}^{k} T_i}{k}$$

Ограниченная исходная статистика позволяет находить точечные оценки параметров математических моделей надежности ПО.

Точечными оценками параметров называют такие оценки, которые характеризуются одним числом.

Для каждого из оцениваемых параметров можно найти множество

оценок. В общем случае задача оценки параметров $\theta_1,\theta_2,....,\theta_k$ сводится к нахождению функций $\overline{\theta_1}(x_1,x_2,.....,x_k),\overline{\theta_2}(x_1,x_2,.....,x_k),\overline{\theta_k}(x_1,x_2,.....,x_k)$, зависящие только от известных величин. Эти функции использовать в качестве оценок значений соответствующих параметров $\theta_1,\theta_2,....,\theta_k$. Оценка параметра зависит от результатов испытаний и является случайной величиной.

Чтобы знать какую из оценок предпочесть, нужно сформулировать требования к ним. С точки зрения точности и надежности оценок желательно, чтобы они были тесно сконцентрированы около значений оцениваемых параметров. Критериями качества точечных оценок служат несмещенность, состоятельность и эффективность.

Оценка $\overline{\theta}$ некоторого параметра θ называется несмещенной, если математическое ожидание оценки совпадает с оцениваемым параметром при любом конечном объеме выборки, т.е. чтобы

$$M\left[\overline{\theta}(x_1,x_2,...,x_k)\right]=\theta$$

Состоятельной называется оценка $\overline{\theta}$ параметра θ, если при увеличении числа наблюдений до бесконечности оценка сходится по вероятности к оцениваемому параметру θ, т.е.

$$\lim_{k\to\infty} P\left[\mid \overline{\theta}(x_1,x_2,...,x_k)-\theta \mid<\varepsilon\right]=1$$

где ε - сколь угодно малое положительное число.

Оценка будет эффективной, если существует такая несмещенная оценка $\overline{\theta}$, дисперсия которой равна $D\text{э}(\theta)$. $D\text{э}(\theta)$ - нижняя граница дисперсии оценки параметра θ. Сравнительная эффективность оценки измеряется коэффициентом эффективности $\varepsilon\le 1$:

$$\varepsilon=\frac{D\text{э}(\theta)}{D(\overline{\theta})}$$

Точечную оценку параметров можно осуществить различными методами. Рассмотрим два основных метода, которые часто используются:

метод максимального правдоподобия и *метод наименьших квадратов.*

Метод максимально правдоподобия был предложен английским статистом Р. Фишером в 1912г. и является наиболее распространенным и сильным методом для нахождения оценок параметров моделей по опытным данным.

Предположим, что имеется выборка случайной величины X объема k которая принимает значения $x_1, x_2, ..., x_k$. Если случайная величина X - непрерывная, то функция правдоподобия имеет вид:

$$L(x_1, x_2, ..., x_k; \theta) = \prod_{i=1}^{k} f_i(x_i, \theta)$$

где функция $f(x_i, \theta)$ - плотность распределения случайной величины X, зависящая от неизвестного параметра θ. Если случайная величина X - дискретная, то функция правдоподобия имеет иной вид, а именно:

$$L(x_1, x_2, ..., x_k; \theta) = \prod_{j=1}^{k} P_{ij}(\theta)$$

где индексы i у вероятностей показывают, что наблюдались значения $x_{i1}, x_{i2}, ..., x_{ik}$.

Считая значения $x_1, x_2, ..., x_k$ данными, L рассматривают как функцию зависящую от неизвестного параметра θ, оценку которого необходимо найти.

Сущность метода максимального правдоподобия заключается в том, что в качестве оценки параметра θ, берется то значение аргумента θ, которое обращает функцию L в максимум.

Для нахождения оценки этого параметра необходимо решить уравнение:

$$\frac{dL}{d\theta} = 0$$

и отобрать то решение $\theta = \overline{\theta}(x_1, x_2, ..., x_k)$, которое обращает функцию L в

максимум. Каждое решение этого уравнения носит название *оценки максимального правдоподобия*.

Для упрощения процедуры нахождения оценок вместо L часто рассматривают функцию ln(L). Это возможно в силу того, что $L(x_1, x_2, ..., x_k; \theta)$ - положительная функция, достигающая максимума в той же точке, что и функция ln(L), при одном и том же значении θ. В этом случае нахождение оценок параметра осуществляют решая уравнение:

$$\frac{d \ln L}{d\theta} = 0$$

Умножение функции L на постоянный множитель не меняет максимума этой функции L.

Если требуется найти несколько параметров $\theta_1, \theta_2, ..., \theta_k$, то оценки максимального правдоподобия для этих параметров находятся из совместного решения системы:

$$\begin{cases} \dfrac{d \ln L(\theta_1, \theta_2, ..., \theta_k)}{d\theta_1} = 0 \\ \dfrac{d \ln L(\theta_1, \theta_2, ..., \theta_k)}{d\theta_2} = 0 \\ \ldots\ldots\ldots\ldots\ldots\ldots \\ \dfrac{d \ln L(\theta_1, \theta_2, ..., \theta_k)}{d\theta_k} = 0 \end{cases}$$

При обработке и анализе экспериментальных данных широко применяют также метод наименьших квадратов.

Предположим, что имеется *k* независимых наблюдений случайной величины X, которая принимает значения $x_1, x_2, ..., x_k$. Измерения Y при этих значениях дают результаты $y_1, y_2, ..., y_k$. Значения X и Y являются приближенными. Между ними полагается зависимость, вид которой известен. Требуется построить зависимость Y*. Функция Y* зависит от параметра θ, оценку, которого необходимо найти.

Принцип метода наименьших квадратов заключается в том, что

параметр θ функции Y* считаются наилучшими, если минимальна сумма квадратов отклонений:

$$F(x_1, x_2, ..., x_k; \theta) = \sum_{i=1}^{k} \left(Y_i(x_i) - Y_i^*(x_i) \right)^2$$

Для нахождения оценки параметра θ необходимо решить уравнение:

$$\frac{dF}{d\theta} = 0$$

и отобрать то решение $\theta = \overline{\theta}(x_1, x_2, ..., x_k)$, которое обращает функцию F в минимум. При правильном решении задачи сумма квадратов отклонений близка к нулю.

Если требуется найти несколько параметров $\theta_1, \theta_2, ..., \theta_k$, то оценки параметров находятся из совместного решения системы *k* уравнений:

$$\begin{cases} \dfrac{dF(\theta_1, \theta_2, ..., \theta_k)}{d\theta_1} = 0 \\ \dfrac{dF(\theta_1, \theta_2, ..., \theta_k)}{d\theta_2} = 0 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ \dfrac{dF(\theta_1, \theta_2, ..., \theta_k)}{d\theta_k} = 0 \end{cases}$$

Метод наименьших квадратов позволяет в классе линейных оценок находить такие, которые эффективны среди всех несмещенных оценок. Это свойство не зависит от вида распределения элементов выборки $y_1, y_2, ..., y_k$.

При использовании любого метода для нахождения оценок параметров математических моделей встает задача решения системы трансцендентных уравнений. Аналитическое решение для этих систем найти не удается практически никогда, поэтому необходимо искать либо приближенные решения или же решать их методами вычислительной математики.

Одной из основных задач теории оценок параметров следует считать построение общих методов нахождения хороших оценок.

Сравним два рассмотренных выше метода.

Метод максимального правдоподобия выполняет почти все требования к точечным оценкам. Он обеспечивает состоятельные, иногда смещенные, эффективные и асимптотически нормально распределенные оценки. Для их нахождения используется вся информация о неизвестном параметре. Однако на практике этот метод часто приводит к необходимости решать весьма сложные системы уравнений.

Метод наименьших квадратов полагает, что "наилучшие" оценки обращают в минимум сумму квадратов отклонений наблюденных значений Y от вычисленных при тех же значениях X и находятся вблизи среднего значения X. На практике он приводит к решению более легких систем уравнений, чем в предыдущем методе.

Исходя из вышесказанного, будем считать, что для получения оценок неизвестных параметров достаточным является использование метода наименьших квадратов. Получаемые при этом системы линейных и нелинейных уравнений достаточно легко решаются аналитически, т.е. без применения численных методов. Это немаловажно, т.к. реализация последних в программе может привести к снижению точности решения, а также к зависимости этого решения от значений начального приближения, что при неустойчивом алгоритме может дать большой разброс результатов.

Из целого ряда численных методов решения системы трансцендентных уравнений наиболее часто используются метод Ньютона, метод итерации и метод спуска (градиентный метод).

Метод Ньютона представляет собой метод последовательных приближений. Сложность метода заключается в том, что на каждом шаге вычисления следующего приближения к искомому значению вектора корней системы необходимо определять матрицу Якоби системы функций $f_1, f_2, \ldots, f_n$ относительно переменных $x_1, x_2, \ldots x_n$ и её обратную матрицу. Кроме того, на скорость сходимости метода Ньютона существенное влияние оказывает шаг, с которым выполняется последовательное приближение и, что наиболее важно, степень приближения начальных условий к вектору искомых

значений.

Метод итерации для решения систем нелинейных уравнений представляет собой, в сущности, модифицированный процесс Ньютона. Это также процесс последовательного определения значений вектора переменных, с той лишь разницей, что при условии достаточной близости начального приближения к искомому решению, обратную матрицу для матрицы Якоби не вычисляют на каждом шаге итерации, а используют единожды вычисленные значения коэффициентов матрицы. На характеристики процессов сходимости метода итерации влияют те же параметры, что и для метода Ньютона.

Метод спуска (градиентный метод) заключается в получении значений, приближающихся к вектору корней исходной системы путем движения по нормали к поверхности уровня $V(x) = \sum_{i=1}^{k} [f_i(x)]^2 = (f(x), f(x))$ до тех пор, пока эта нормаль не коснется в некоторой точке новой поверхности уровня *V(x)*, т.е. задача сводится к поиску минимума функции *V(x)* в *n*-мерном пространстве. Основным преимуществом градиентного метода перед другими известными методами решения систем уравнений является его безусловная сходимость. Однако в зависимости от шага и начального приближения искомого решения скорость сходимости может быть мала, кроме того градиентный метод может приводить к локальным решениям. Учитывая, что диапазоны поиска параметров математических моделей весьма широки, а высокая точность искомых параметров не нужна ибо точность исходных данных низка, наиболее целесообразно при решении задачи оценки параметров математических моделей надежности ПО использовать градиентный метод. К тому же этот метод наиболее прост и наиболее удобен для реализации его на ЭВМ.

6.2. Модели надежности программного обеспечения

Существующие математические модели позволяют оценивать

характеристики ошибок в программах и прогнозировать их надежность при проектировании и эксплуатации. Модели имеют вероятностный характер, и достоверность прогнозов зависит от точности исходных данных и глубины прогнозирования по времени. Эти математические модели предназначены для оценки:

- показателей надежности комплексов программ в процессе отладки;
- количества ошибок, оставшихся не выявленными;
- времени, необходимого для обнаружения следующей ошибки в функционирующей программе.

6.2.1. Модель Шумана

Для оценки достигнутого в процессе отладки уровня надежности в этой модели используются данные о числе ошибок, устраненных в процессе компоновки программы и ее тестирования. По этим данным вычисляются параметры математической модели надежности, которая может использоваться для прогнозирования поведения ПО в будущем.

Предполагается, что при последовательных прогонах программы наборы векторов входных данных являются случайными и выбираются в соответствии с законом распределения, соответствующим реальным условиям функционирования ПО.

Модель основывается на следующих допущениях:

1 в начальный момент тестирования программного комплекса в нем имеется N_0 ошибок, при проявлении программной ошибки она устраняется, при этом корректировка не вызовет появления новых ошибок;

2 общее число машинных команд I в программном комплексе остается постоянным в течение всего времени тестирования и не зависит от числа устраненных ошибок;

3 интенсивность появления программных отказов λ пропорционально

числу ошибок, оставшихся в ПО после отладки в течение времени τ:

$\lambda = C\left[\frac{N_0}{I} - \varepsilon(\tau)\right]$, где $\varepsilon(\tau) = \frac{k_{устр}}{I}$ отношение числа ошибок, устраненных в течение времени отладки τ к общему числу машинных команд ПО.

В данной модели различаются два значения времени:

1 τ – время отладки;

2 t – суммарная наработка ПО.

Время отладки включает затраты времени на выявление ошибок с помощью тестов, контрольные проверки и т.п. Время безошибочного функционирования ПО при этом чаще всего не учитывается.

В модели интенсивность проявления программных ошибок λ считается постоянной в течение интервала времени между проявлениями i и $i+1$ программного отказа. Значение λ изменяется лишь при обнаружении и устранении очередной ошибки, при этом время t вновь начинает отсчитываться от нуля.

Модель ошибок предполагает также экспоненциальность закона распределения случайных интервалов времени между смежными отказами.

В силу принятых допущений для некоторого фиксированного времени отладки τ вероятность безотказного функционирования ПО в течение заданного времени интервала времени $(0, t)$ будет иметь вид:

$$P(t,\tau) = exp(-C\left[\frac{N_0}{I} - \varepsilon(\tau)\right] * t)$$

Средняя наработка на программный отказ определяется выражением:

$$m_t = \frac{1}{\lambda} = \left\{C\left[\frac{N_0}{I} - \varepsilon(t)\right]\right\}^{-1}$$

Рассматриваемая модель является двухпараметрической, для практического использования при оценке показателей надежности необходимо найти значения параметров N_0 и C. Оценивание параметров

модели осуществляют по экспериментальным данным, собираемым на этапе тестирования. Для нахождения оценок параметров можно воспользоваться, например, методом моментов или методом максимального правдоподобия или же методом наименьших квадратов.

Применять метод моментов для оценки N_0 и C можно, если имеются данные для двух периодов отладки программы (0, τ_1) и (0, τ_2), причем обязательно $\tau_2 > \tau_1$.

Пусть продолжительность работы программы, соответствующая τ_1 равна T_1, а число ошибок, устраненных за этот интервал времени – n_1. продолжительность работы программы соответствующая τ_2 равна T_2, а число ошибок, устраненных за этот интервал времени – n_2 .

Выражение для нахождения интересующих нас параметров при этом принимает вид:

$$\hat{N}_0 = \frac{I[\gamma\varepsilon(\tau_1) - \varepsilon(t_2)]}{\gamma - 1}, \text{ где } \gamma = \frac{T_1 n_2}{T_2 n_1}$$

$$\hat{C} = \frac{n_1}{T_1\left[\frac{N_0}{I} - \varepsilon(\tau_1)\right]}$$

Для использования метода максимального правдоподобия необходимо в процессе отладки собирать статистику $\{ T_i \}$ – интервалов времени между проявлениями программных отказов. Для этой статистики, исходя из предпосылок рассматриваемой модели, должно выполняться условие $T_i > T_{i-1}$, $i = 1 \ldots k$ (k – число устраненных программных ошибок).

Функция максимального правдоподобия в этом случае принимает вид:

$$L = \prod_{i=1}^{k}\left(C\left[\frac{N_0}{I} - \frac{i-1}{I}\right]\right)exp\left(-CT_i\left[\frac{N_0}{I} - \frac{i-1}{I}\right]\right)$$

удобнее брать ln L:

$$L = \sum_{i=1}^{k} ln\left(C\left[\frac{N_0}{I} - \frac{i-1}{I} \right] \right) - \sum_{i=1}^{k} \left(CT_i \left[\frac{N_0}{I} - \frac{i-1}{I} \right] \right)$$

Система трансцендентных уравнений, решение которой позволяет найти оценки интересующих нас параметров, принимает вид:

$$\frac{\partial ln\, L}{\partial C} = \frac{k}{C} - \sum_{i=1}^{k} \left[\frac{N_0}{I} - \frac{(i-1)}{I} \right] * T_i = 0$$

$$\frac{\partial\, ln\, L}{\partial N_0} = \sum_{i=1}^{k} \frac{C}{\left[\frac{N_0}{I} - \frac{(i-1)}{I} \right]} - \sum_{i=1}^{k} CT_i = 0$$

Параметры математической модели могут быть оценены и с помощью метода наименьших квадратов. И в этом случае необходимо в процессе отладки собирать статистику $\{ T_i \}$ – интервалов времени между проявлениями программных отказов. В этом случае для метода наименьших квадратов могут быть записаны следующие функционалы:

$$\Phi_1 = \sum_{i=1}^{k} \left[\frac{1}{T_i} - C\left[\frac{N_0}{I} - \frac{i-1}{I} \right] \right]^2 ; \qquad \Phi_2 = \sum_{i=1} \left[T_1 - \frac{1}{C\left[\frac{N_0}{I} - \frac{i-1}{I} \right]} \right]^2$$

Система трансцендентных уравнений для нахождения оценок параметров модели ошибок, например, для случая минимизации функционала Φ_1, принимает вид:

$$\frac{\partial \Phi_1}{\partial C} = \sum_{i=1}^{k} \left[\frac{1}{T_i} - C\left[\frac{N_0}{I} - \frac{i-1}{I} \right] \right] * \frac{N_0 - (i-1)}{I} = 0 ;$$

$$\frac{\partial \Phi_1}{\partial N_0} = \sum_{i=1}^{k} \left[\frac{1}{T_i} - C\left[\frac{N_0}{I} - \frac{i-1}{I} \right] \right] * \frac{C}{I} = 0$$

6.2.2. Модель Джелинского-Моранды

Рассматриваемая модель является частным случаем модели ошибок. Она позволяет оценивать некоторые параметры надежности ПО и прогнозировать надежность.

При создании модели предполагается:

1 интервалы времени отладки между смежными моментами обнаружения программных ошибок являются случайными величинами с экспоненциальным законом распределения;

2 частота программных ошибок или интенсивность программных отказов пропорциональна числу устраненных в программном обеспечении ошибок $\frac{dn}{dt} = kn_{ост} = \lambda$;

3 каждый программный отказ приводит к обнаружению и устранению одной ошибки, при этом число оставшихся в ПО ошибок уменьшается на единицу;

4 интенсивность программных ошибок на интервалах, где не происходит изменения их числа, постоянна и определяется выражением: $\lambda_i = R\ (N - (i - 1))$;

Функция плотности распределения времени обнаружения $i^{-ой}$ ошибки, отсчитываемого от момента выявления $(i - 1)$ ошибки имеет вид:

$$f(t_i) = \lambda_i * e^{-\lambda_i t_i}$$

Вероятность безотказной работы ПО на интервале (0, t) после устранения $(i - 1)$ программного отказа:

$$P(t) = e^{-R(N_0 - (i-1))t}$$

сопоставление этих уравнений с уравнениями модели Шумана (модели ошибок) показывает их совпадение при условии $R = \frac{C}{I}$.

Модель, как и предыдущая, является двухпараметрической,

неизвестными параметрами являются число ошибок в ПО на момент начала тестирования N_0 и коэффициент пропорциональности R. Оценки этих параметров могут быть получены, например, методом максимального правдоподобия, если имеется статистика { T_i } моментов проявления программных ошибок.

Функция максимального правдоподобия для рассматриваемой модели принимает вид:

$$L = \prod_{i=1}^{k} R\left[N_0 - (i-1)\right] * e^{-R\left[N_0 - (i-1)\right]T_i}$$

Система алгебраических уравнений для нахождения оценок параметров модели:

$$\frac{\partial \ln L}{\partial R} = \sum_{i=1}^{k} \frac{1}{R} - \sum_{i=1}^{k} \left[N_0 - (i-1)\right] * T_i = 0;$$

$$\frac{\partial \ln L}{\partial N_0} = \sum_{i=1}^{k} \frac{R}{R\left[N_0 - (i-1)\right]} - R\sum_{i=1}^{k} T_i = 0$$

Для удобства решения задачи уравнения следует преобразовать к виду:

$$\sum_{i=1}^{k} \frac{1}{N_0 - (i-1)} = \frac{k\sum_{i=1}^{k} T_i}{\sum_{i=1}^{k} N_0 T_i - \sum_{i=1}^{k} iT_i + \sum_{i=1}^{k} T_i}$$

$$R = \frac{k}{\sum_{i=1}^{k} N_0 T_i - \sum_{i=1}^{k} iT_i + \sum_{i=1}^{k} T_i}$$

В этом случае первое уравнение нужно минимизировать по невязке и найти оценку для N_0 , тогда оценка для коэффициента пропорциональности находится путем вычисления его значения по второму уравнению.

В случае использования метода наименьших квадратов для нахождения

оценок параметров модели построим функционал вида:

$$\Phi = \sum_{i=1}^{k} \left[\frac{1}{T_i} - R\left[N_0 - (i-1)\right] \right]^2$$

В этом случае система алгебраических уравнений для нахождения оценок параметров модели принимает вид:

$$\sum_{i=1}^{k} \frac{N_0 - (i-1)}{T_i} - \sum_{i=1}^{k} R(N_0 - (i-1))^2 = 0$$

;

$$\sum_{i=1}^{k} \frac{1}{T_i} - R \sum_{i=1}^{k} (N_0 - (i-1)) = 0$$

6.2.3. Модель Гоуэла-Опумоты

Данная модель явно указывает на экспоненциальный характер интенсивности отказов и описывается следующим уравнением: $\lambda = a \cdot b \cdot e^{-bt}$

где

$$\lambda_i = a \cdot b \cdot e^{-b \cdot \sum_{j}^{i} (j-1) T_i}$$

тогда

$$F = \sum_{i=1}^{k} \left[\frac{1}{T_i} - a \cdot b \cdot e^{-b \cdot \sum_{j}^{i} (j-1) \cdot T_i} \right]^2$$

необходимо решать

$$\begin{cases} \dfrac{\partial F}{\partial a} = 0 \\ \dfrac{\partial F}{\partial b} = 0 \end{cases}$$

6.2.4. Модель роста 1

Модель роста 1 базируется на следующих ограничениях:

1 в процессе тестирования при обнаружении и исправлении ошибки новые ошибки в ПО не вносятся, число оставшихся в ПО ошибок только уменьшается;

2 процесс тестирования ПО осуществляется в ограниченном интервале времени. После завершения тестирования в ПО остаются необнаруженные ошибки и финальная вероятность $R_{П} = R_{П\infty}$ будет меньше единицы.

Математическое выражение для модели Роста 1 имеет вид:

$$R_{П}(k) = R_{П\infty} - \frac{a}{k},$$ где

1 $R_{П}(k)$ – вероятность безошибочного выполнения задачи на произвольном наборе исходных данных после устранения k программных ошибок;

2 a – неизвестная константа;

3 $R_{П\infty}$ – финальная вероятность безошибочного выполнения задачи, достигнутая после завершения этапа отладки ПО.

Модель двухпараметрическая, неизвестными параметрами являются $R_{П\infty}$ и a. Оценка этих параметров возможна при наличии статистики $\{ n_i \}$ (n_i – число успешно завершенных реализаций программы на интервале между появлением $i^{-ой}$ и $i+1$ ошибки).

Для оценки параметров математической модели методом максимального правдоподобия составим функцию правдоподобия:

$$L = \prod_{i=1}^{k} C_{n_i+1}^{n_i} (R_{П\infty} - \frac{a}{i})^{n_i} (1 - R_{П\infty} + \frac{a}{i})$$

Для оценки неизвестных параметров математической модели необходимо решать следующую систему алгебраических уравнений:

$$\frac{\partial \ln L}{\partial R_{П\infty}} = \sum_{i=1}^{k} \frac{n_i}{R_{П\infty} - \frac{a}{i}} - \sum_{i=1}^{k} \frac{1}{1 - R_{П\infty} + \frac{a}{i}} = 0$$

;

$$\frac{\partial \ln L}{\partial a} = -\sum_{i=1}^{k} \frac{n_i}{i(R_{П\infty} - \frac{a}{i})} + \sum_{i=1}^{k} \frac{1}{i(1 - R_{П\infty} + \frac{a}{i})} = 0$$

Оценка параметров математической модели методом наименьших квадратов требует построения следующего функционала:

$$F = \sum_{i=1}^{k} \left[\frac{n_i}{n_i + 1} - R_{П\infty} + \frac{a}{i} \right]^2$$

В этом случае система алгебраических уравнений для нахождения оценок параметров примет вид:

$$\frac{\partial F}{\partial R_{П\infty}} = \sum_{i=1}^{k} (R_{П\infty} - \frac{a}{i}) - \sum_{i=1}^{k} \frac{n_i}{n_i + 1} = 0$$

;

$$\frac{\partial F}{\partial a} = \sum_{i=1}^{k} \frac{n_i}{i(n_i + 1)} - \sum_{i=1}^{k} \frac{1}{i} (R_{П\infty} - \frac{a}{i}) = 0$$

6.2.5. Модель роста 2

При построении модели роста 2 предполагается, что процесс тестирования осуществляется столько времени, что финальная вероятность безошибочного решения задачи равна "1" и изменяется аппроксимирующая функция.

Математическое выражение модели роста 2 имеет вид:

$$R_П(k) = 1 - ae^{-C(k-1)}$$

Модель, как и в предыдущем случае, двухпараметрическая. Неизвестными параметрами, которые следует оценивать, являются константы *a* и *C*.

Для нахождения оценок параметров методом максимального правдоподобия строим функцию:

$$L = \prod_{i=1}^{k} C_{n_i+1}^{n_i} (1 - ae^{-C(i-1)})^{n_i} (ae^{-C(i-1)})$$

Система алгебраических уравнений, решение которой позволит найти искомые оценки, принимает в данном случае вид:

$$\frac{\partial \ln L}{\partial C} = \sum_{i=1}^{k} \frac{n_i (i-1) a e^{-C(i-1)}}{1 - ae^{-C(i-1)}} - \sum_{i=1}^{k} (i-1) = 0 ;$$

$$\frac{\partial \ln L}{\partial a} = \frac{k}{a} - \sum_{i=1}^{k} \frac{n_i e^{-C(i-1)}}{1 - ae^{-C(i-1)}} = 0$$

6.2.6. Модель роста 3

Аналитическое выражение, рассматриваемое как модель роста 3, имеет вид:

$$R_П(k) = R_{П\infty} - (R_{П\infty} - R_{П0})(1 - \frac{a}{R_{П\infty}})^k$$, где

2 $R_{П\infty}$ – имеет тот же смысл, что и в модели роста 1;

3 $R_{П0}$ – вероятность безошибочной однократной реализации программного продукта на момент начала этапа отладки.

4 a – неизвестная константа.

Данная математическая модель, в отличие от рассмотренных ранее, является трехпараметрической. Необходимо по экспериментальным данным оценивать $R_{П\infty}$, $R_{П0}$ и a.

Функция правдоподобия при использовании для оценки параметров метода максимального правдоподобия в данном случае принимает вид:

$$L=\prod_{i=1}^{k} C_{n_i+1}^{n_i}\left\{R_{П\infty}-(R_{П\infty}-R_{П0})(1-\frac{a}{R_{П\infty}})^i\right\}^{n_i}\left\{1-(R_{П\infty}-(R_{П\infty}-R_{П0}))(1-\frac{a}{R_{П\infty}})^i\right\}$$

Система алгебраических уравнений, решение которой позволяет найти оценки неизвестных параметров принимает вид:

$$\frac{\partial \ln L}{\partial R_{П\infty}}=\sum_{i=1}^{k}\frac{n_i(1-(1-\frac{a}{R_{П\infty}})-(R_{П\infty}-R_{П0})i(1-\frac{a}{R_{П\infty}})^{i-1}\frac{a}{R_{П\infty}^2}}{R_{П\infty}-(R_{П\infty}-R_{П0})(1-\frac{a}{R_{П\infty}})^i}-$$

$$-\sum_{i=1}^{k}\frac{1-(1-\frac{a}{R_{П\infty}})-i(1-\frac{a}{R_{П\infty}})^{i-1}\frac{a}{R_{П\infty}^2}(R_{П\infty}-R_{П0})}{1-R_{П\infty}+(R_{П\infty}-R_{П0})(1-\frac{a}{R_{П\infty}})^i}=0$$

$$\frac{\partial \ln L}{\partial R_{П0}}=\sum_{i=1}^{k}\frac{n_i(1-\frac{a}{R_{П\infty}})^i}{R_{П\infty}-(R_{П\infty}-R_{П0})(1-\frac{a}{R_{П\infty}})^i}-\sum_{i=1}^{k}\frac{(1-\frac{a}{R_{П\infty}})^i}{1-R_{П\infty}+(R_{П\infty}-R_{П0})(1-\frac{a}{R_{П\infty}})^i}=0;$$

$$\frac{\partial \ln L}{\partial a}=\sum_{i=1}^{k}\frac{n_i i(1-\frac{a}{R_{П\infty}})^{i-1}\frac{R_{П\infty}-R_{П0}}{R_{П\infty}}}{R_{П\infty}-(R_{П\infty}-R_{П0})(1-\frac{a}{R_{П\infty}})^i}-$$

$$-\sum_{i=1}^{k}\frac{i(1-\frac{a}{R_{П\infty}})^{i-1}(R_{П\infty}-R_{П0})}{R_{П\infty}(1-R_{П\infty}+(R_{П\infty}-R_{П0})(1-\frac{a}{R_{П\infty}})^i)}=0$$

6.2.7. Модель роста 4

В процессе отладки вероятность однократного безошибочного выполнения программы возрастает от значения P_0 до некоторого $P\infty$, которое можно рассматривать как величину близкую к единице. В начальный этап отладки очень грубо можно оценивать $P\infty$, а вот P_0, которое явно проявляется в этот момент, нужно оценивать. Устранение ошибок будет

приводить к увеличению вероятностного показателя. Пусть $P_k = P_0 + a(1 - \frac{1}{k}) = P_0 + a\frac{k-1}{k}$, тогда функционал для метода наименьших квадратов примет вид:

$$F = \sum_{i=1}^{k} \left[\frac{n_i - 1}{n_i} - P_0 - a\frac{i-1}{i} \right]^2$$

Система алгебраических уравнений, решение которой позволяет найти оценки неизвестных параметров принимает вид:

$$\frac{\partial F}{\partial P_0} = -\sum_{i=1}^{k} \frac{n_i - 1}{n_i} + P_0 k + a\sum_{i=1}^{k} \frac{i-1}{i} = 0;$$

$$\frac{\partial F}{\partial a} = -\sum_{i=1}^{k} \frac{(n_i - 1)(i-1)}{n_i * i} + P_0 \sum_{i=1}^{k} \frac{i-1}{i} + a\sum_{i=1}^{k} \frac{(i-1)^2}{i^2} = 0$$

6.2.8. Модель путей

Структурно – надежностная схема для любого программного продукта представляет собой последовательно – параллельную структуру, где каждая последовательная цепочка есть путь обработки некоторого подмножества набора исходных данных для получения результата. Каждый l путь выбирается в процессе функционирования ПО с некоторой вероятностью P_l. Вероятность безошибочного выполнения пути есть произведение вероятностей безошибочного выполнения команд, входящих в эти пути, если ошибки в элементах программы независимы. Вероятность того, что единичная реализация ПО будет безошибочной при рассмотренных предпосылках будет определяться выражением:

$$RП = \sum_{l \in L} P_l \prod_{i=1}^{m_l} P_{il},$$ где

1 L – множество всех путей программного продукта;

2 m_l – число команд, входящих в l путь;

3 P_{il} – вероятность безошибочного выполнения $i^{-ой}$ команды, входящей в l путь.

На момент начала детерминированного тестирования, когда после этапа автономной отладки в ПО осталось N_0 ошибок, вероятность безошибочной единичной реализации ПО обозначим $R_{П0}$.

На этапе тестирования при обнаружении ошибки происходит корректировка программы. Если в результате произведенной корректировки число ошибок в программном продукте уменьшилось, будет наблюдаться улучшение показателей надежности системы. Если же проведенная корректировка привела к появлению в ПО новых дополнительных ошибок, показатель надежности системы будет ухудшаться. Для того, чтобы учесть все эти факторы, выражение для вероятности безошибочной единичной реализации ПО после корректировки запишем в виде:

$R_{П1} = 1 - (1 - R_{П0}) * q$, где

q – коэффициент, который в случае внесения при коррекции ПО новых, дополнительных ошибок будет $q > 1$, при действительном исправлении программы $q < 1$. В случае, когда никакой коррекции при обнаружении факта проявления программной ошибки не производится $q = 1$.

После проведения второй коррекции ПО выражение для $R_{П2}$ принимает вид:

$$R_{П2} = 1 - (1 - R_{П1}) * q = 1 - (1 - (1 - (1 - R_{П0}) * q) * q) = 1 - (1 - R_{П0}) * q^2$$

После проведения $n^{-ой}$ коррекции: $R_{Пn} = 1 - (1 - R_{П0}) * q^n$

Полученное математическое выражение будем рассматривать как модель путей. Данная модель может рассматриваться как модель надежности ПО для оценки и прогноза показателей надежности для этапа детерминированного тестирования, если при его реализации выполняются следующие требования:

1 вектора исходных данных для тестов выбираются случайным

образом из множества возможных;

2 усилия по обнаружению и устранению программных ошибок в течение всего времени проведения тестирования постоянны;

В отличие от предыдущих моделей рассматриваемая не накладывает никаких ограничений на законы распределения случайных величин и допускает внесение в процессе исправления программных ошибок новых.

Эта модель может использоваться на этапе эксплуатации (когда не происходит исправления ПО при обнаружении программной ошибки) для подтверждения достигнутого уровня надежности. Воспользовавшись этой моделью можно оценить уровень квалификации специалистов, занимающихся отладкой ПО ($q < 1$ – хороший уровень подготовки, $q > 1$ – низкий уровень, при $q = 1$ отладчики вносят столько же новых ошибок, сколько и исправляют).

Модель путей является двухпараметрической, неизвестными параметрами, требующими оценки, являются $R_{П0}$ и q.

Для оценки этих параметров на этапе отладки должна собираться статистика числа успешных прогонов тестов между моментами проявления программных ошибок $\{ n_i \}$.

Для оценки параметров математической модели методом максимального правдоподобия построим функцию подобия:

$$L = \prod_{i=1}^{k} C_{n_{i+1}}^{n_i} \left(1 - (1 - R_{П0}) q^{i-1}\right)^{n_i} (1 - R_{П0}) q^{i-1}$$

Система алгебраических уравнений, решение которой обеспечит нахождение оценок неизвестных параметров математической модели путей в этом случае принимает вид:

$$\frac{\partial \ln L}{\partial R_{П0}} = \sum_{i=1}^{k} \frac{n_i q^{i-1}}{1 - (1 - R_{П0}) q^{i-1}} - \frac{n}{1 - R_{П0}} = 0;$$

$$\frac{\partial \ln L}{\partial q} = -\sum_{i=1}^{k} \frac{n_i (1 - R_{П0})(i-1) q^{i-2}}{1 - (1 - R_{П0}) q^{i-1}} + \frac{1}{q} \sum_{i=1}^{k} (i-1) = 0$$

Для оценки неизвестных параметров методом наименьших квадратов построим функционал:

$$F = \sum_{i=1}^{k} \left[\frac{n_i}{n_i + 1} - (1 - (1 - R_{П0})q^{i-1}) \right]^2$$

Система алгебраических уравнений, решение которой обеспечит нахождение оценок неизвестных параметров $R_{П0}$ и q в этом случае принимает вид:

$$\frac{\partial \ln F}{\partial R_{П0}} = -\sum_{i=1}^{k} \frac{n_i q^{i-1}}{n_i + 1} + \sum_{i=1}^{k} (1 - (1 - R_{П0})q^{i-1})q^{i-1} = 0$$

;

$$\frac{\partial \ln F}{\partial q} = \sum_{i=1}^{k} \left[\frac{n_i}{n_i + 1} - \left(1 - \left(1 - R_{П0}\right)q^{i-1}\right) \right] * (1 - R_{П0})(i-1)q^{i-2} = 0$$

6.2.9. Модель тестов

При проведении детерминированного тестирования, если в процессе корректировок не вносятся дополнительные ошибки, интервалы времени между моментами обнаружения программных ошибок увеличиваются по мере устранения ошибок. Если предположить, что время реализации теста слабо зависит от комбинации входных данных, то можно говорить о том, что число тестов, которое необходимо прогнать для обнаружения очередной программной ошибки зависит от числа устраненных программных ошибок.

Пусть модельный интервал до обнаружения следующей ошибки описывается функцией:

$$N_{мод} = a + b\, i^{\,2}$$

где i − номер интервала;

N − продолжительность интервала по времени.

Оценить параметры a и b можно с помощью метода наименьших квадратов или метода максимального правдоподобия.

По параметрам a и b можно оценить продолжительность интервала до

следующей ошибки и функцию надежности *R(t)*.

6.2.10. Линейная модель

Линейная модель является наиболее широко распространенной и применяется в различных областях, где встает вопрос о регрессионном анализе данных, полученных экспериментально. Она является наиболее простой и ее использование оправдано в большинстве случаев, когда чрезмерная сложность модели резко снижает ее адекватность поставленному эксперименту.

Уравнение линейной модели:

$$P_k = a + b \cdot k \,,$$

где P_k − вероятность обнаружения k-той ошибки, определяемая как

$$P_k = \frac{n_k - 1}{n_k} \,,$$

где n_k − число повторных запусков ПО для обнаружения *k*-той ошибки.

Указанная модель практически не отражает характер процесса качественно, констатируя лишь факт возрастания вероятности.

Модель является двухпараметрической. Нахождение коэффициентов уравнения регрессии проводится по методу наименьших квадратов.

6.2.11. Квадратичная модель

Квадратичная модель является частным случаем полиномиальной, когда степень полинома равна 2. В отличие от большинства других моделей, она содержит три параметра. Уравнение имеет вид:

$$P_k = a + b \cdot k + c \cdot k^2 ,$$

Функция квадратного полинома дает более полноценную картину о характере процесса, однако также не может быть рекомендована для аппроксимации. Уравнение квадратичной регрессии уместно использовать на локальных участках для получения более или менее качественной аппроксимации. Параметры уравнения вычисляются методом наименьших квадратов.

6.2.12. Степенная модель

Степенная модель является двухпараметрической. Ее уравнение имеет вид:

$$P_k = a \cdot k^b$$

Вид степенной функции приближенно отражает характер процесса, однако сама функция неограниченно возрастает с ростом аргумента, поэтому ее использование не рекомендуется.

6.2.13. Логарифмическая модель

Модель является двухпараметрической. Имеет некоторое сходство со степенной моделью. Уравнение для логарифмической модели имеет вид:

$$P_k = a + b \cdot ln(k)$$

Как и степенная модель, эта функция неограниченно возрастает с ростом аргумента, поэтому также не рекомендуется для использования.

6.3. Анализ математических моделей

На стадии разработки и тестирования программного обеспечения для определения первоначальных требований, предъявляемых к программному комплексу, могут быть использованы различные количественные модели. Выделив из множества факторов, определяющих надежность программного обеспечения, фактор оставшихся в программе ошибок, рассмотрим модели, дающие оценку надежности, определяя эту характеристику у тестируемого комплекса.

В моделях Шумана и Джелинского - Моранды, предназначенных для прогнозирования надежности программного обеспечения, рассмотренных выше, в качестве независимой переменной выступает время, и надежность вычисляется по формуле $R(t) = e^{-\lambda t}$. Значение интенсивности отказов λ предполагается постоянным в течение всего периода функционирования системы и изменяется только при обнаружении и устранении ошибок, после чего t снова отсчитывается от нуля. Эти модели верны при условиях:

- t - суммарное время работы программы относительно некоторого начального момента;
- t больше средней длительности выполнения одного прогона программы Δt;
- набор входных данных для последовательных прогонов программы должен выбираться случайным образом в соответствии с законом распределения, который приближенно отражает реальные условия функционирования.

Авторы той и другой модели попытались расширить их в предположении, что интенсивность отказов пропорциональна числу

оставшихся в программе ошибок, а затем применить эти модели к тестированию программ.

Эти модели основаны главным образом на таких общих закономерностях теории вероятности, как экспоненциальная зависимость надежности от числа испытаний. Выбор той или иной модели обуславливается особенностями конкретной поставленной задачи, в рамках которой для обеспечения надежности программного комплекса могут быть применены те или иные допущения, определены границы применимости различных приближений.

Применение модели роста вероятности правильного функционирования программы строится на двух предположениях:

- установившееся значение вероятности правильного функционирования программы при длительном процессе выявления ошибок будет близко к единице;
- по истечении некоторого времени достигается точка практического "прекращения роста" кривой вероятности.

Подобранная кривая роста описывает процесс выявления и устранения ошибок в программе. С помощью этой кривой можно не только оценить степень отлаженности программы в настоящий момент, но и можно осуществлять и прогнозирование развития процесса в будущем.

Существующие математические модели позволяют оценивать характеристики ошибок в программах и прогнозировать их надежность при проектировании и эксплуатации. Модели имеют вероятностный характер, и достоверность прогнозов зависит от точности исходных данных и глубины прогнозирования по времени.

Использование моделей позволяет эффективно и целеустремленно проводить отладку и испытания комплексов программ, помогает принять рациональное решение о времени прекращения отладочных работ. Малый объем выборок реального количества обнаруженных ошибок и большой разброс времени обнаружения последовательных ошибок при завершении

отладки не позволяют построить высокоточные математические модели. Поэтому целесообразно применять простейшие модели, точность которых близка к точности, обусловленной исходными данными.

ЗАКЛЮЧЕНИЕ

Известно, что одним из наиболее трудоемких и длительных этапов создания ПО является этап отладки. Статистика утверждает, что на этот этап выпадает около или даже более 40% всего времени создания ПО. На этом этапе происходит устранение ошибок, которые были привнесены разработчиками на всех предшествующих этапах создания ПО. Уменьшение числа ошибок, оставшихся в ПО, повышает показатели надежности, которые напрямую связаны с показателями эффективности создаваемого ПО. Поэтому в процессе отладки желательно постоянно оценивать эти показатели надежности, следить за динамикой их изменения с тем, чтобы при достижении заданного уровня надежности своевременно прекратить тестирование, сократив время, отводимое для этого этапа, а следовательно уменьшить финансовые затраты, связанные с этим этапом.

На этапе тестирования оценки показателей надежности ПО не могут быть получены с помощью статистических методов, так как ПО имеется в единственном экземпляре, и после устранения любой ошибки или привнесения новой в результате корректировки изменяются показатели надежности данного ПО. В этом смысле имеем не стационарный процесс, а случайную выборку, которая всегда состоит из единственного наблюдения или эксперимента.

ЛИТЕРАТУРА

1. Дружинин Г.В., Надежность автоматизированных производственных систем - 4-е изд. перераб. и доп. – М: "Энергомашиздат", 1986г.
2. Кофман А., Модели и методы исследования операций – М: "Мир", 1966г.
3. Кофман А. , Фор Р., Займемся исследованием операций – М: "Мир", 1966г.
4. Липаев В.В., Надежность программного обеспечения АСУ – М: "Энергоиздат", 1981г.
5. Липаев В.В., Тестирование программ – М: "Радио и связь", 1986г.
6. Полак Э., Численные методы оптимизации, – М: "Мир", 1974г.
7. Смирнов Н.В., Дунин-Барковский И.В., Курс теории вероятностей и математической статистики для технических приложений, – М: "Наука", 1969г.
8. Тейер Т., Липов М., Нельсон Э., Надежность программного обеспечения: - пер.с английского – М: "Мир", 1981г.
9. Черкесов Г.Н., Основы теории надежности АСУ, – Л: Ленинградский политехнический институт им. М.И. Калинина, 1975г.
10. Шаракшанэ А.С., Железнов И.Г., Ивницкий И.В., Сложные системы, – М: "Высшая школа", 1977г.
11. Шторм Р., Теория вероятностей, математическая статистика, статистический контроль качества, – М: "Мир", 1970г.

12.Шураков В.В., Надежность программного обеспечения систем обработки данных, – М.:"Финансы и статистика", 1987г.

13. Г. Н. Черкесов Надежность аппаратно- программных комплексов. Учебное пособие. СПб. Питер. 2005 г.

14. А.М. Половко, С.В. Гуров Основы теории надежности . Практикум. СПб. 2006 г.

15. Б.А.Козлов, И.А.Ушаков. Справочник по расчету надежности.-М.Сов.радио,1975.

16. И.А.Рябинин, Г.Н.Черкесов. Логико-вероятностные методы исследования надежности структурно-сложных систем .-М.Радио и связь,1981.

17. Лонгботтом Р. Надежность вычислительных систем. М.Энергоатомиздат,1985.

18. Д. Я. Герман Основы теории надежности. Конспект лекций. МГТУ им . Баумана. 2005.

19. Е. В. Крылов, В.А. Острейковский, Н.Г. Типикин Техника разработки программ. Книга 2. Технология, надежность и качество программного обеспечения. М. Высшая школа. 2008 г.