Глава 8

ПОВЫШЕНИЕ И ОБЕСПЕЧЕНИЕ НАДЁЖНОСТИ МНОГОМОДУЛЬНЫХ ПРОГРАММНЫХ СИСТЕМ

8.1. ВВЕДЕНИЕ

В связи с непрерывным совершенствованием СВТ и массовым её использованием в самых разнообразных областях техники и науки остаётся устойчивой тенденция роста сложности ПО. Чрезмерная сложность вновь создаваемых ПК, высокая стоимость П и сравнительно низкий уровень качества их производства не позволяют исключать возможность возникновения ошибок, приводящих к нарушению работоспособности СВТ и снижению их производительности.

Используя термин «надёжность ПО», по аналогии с понятием «надёжность аппаратуры», необходимо помнить, что ошибки ПО имеют совершенно другую физическую природу, чем отказы техники, о чём говорилось в первой главе книги. Однако это не является причиной невозможности использования некоторых терминов и показателей надёжности техники при исследовании качества ПО [3, 24]. В частности, это оправдывается и необходимостью решения задачи распределения ресурсов или затрат между самими ЭВС и их ПО при достижении заданного значения показателя надёжности. Одним из важнейших общих показателей надёжности, которые представляют интерес для практики, является вероятность его безошибочного функционирования.

Проверка правильности разработанного ПО (корректности, устойчивости) и удовлетворение его требованиям спецификаций осуществляются при отладке и тестировании. Как правило, основным фактором отладки является затраченное на неё время. Поэтому в некоторых моделях оценивания надёжности П наряду

с необходимым временем их функционирования при решении конкретных задач рассматривается и второй временной фактор – время отладки этих П до использования по назначению.

Для удобства анализа показателей надёжности сложных ПК целесообразно представлять их в виде совокупностей менее сложных составляющих, обычно называемых программными модулями (ПМ). ПМ, в свою очередь, может быть разделён на более мелкие составные части и т.д. Таким образом, ПМ является аналогом элемента расчёта в теории надёжности. Обычно он представляет самостоятельную программу и вводится, исходя из соображений исследователя.

8.2. АНАЛИЗ И СИНТЕЗ ПОКАЗАТЕЛЯ НАДЁЖНОСТИ ПРОГРАММНОГО КОМПЛЕКСА ПРИ НЕПРЕРЫВНЫХ ВРЕМЕННЫХ ПАРАМЕТРАХ

При исследовании надёжности функционирования ПК обычно решаются две задачи. Первая – по заданной структуре ПК, состоящего из некоторой совокупности взаимосвязанных ПМ, имеющих показатели надёжности, необходимо найти показатель надёжности ПК. Эту задачу традиционно назовём прямой.

Наряду с первой, прямой задачей может решаться вторая задача достижения максимального (минимального) значения показателя надёжности при определённых ограничениях на ресурсы, в качестве которых могут выступать время, стоимость и другие факторы. Также может решаться задача минимизации величины некоторого ограничения при достижении требуемого (заданного) значения показателя надёжности. Любую их этих указанных задач будем называть обратной.

Постановка задачи. ПК состоит из M модулей, связанных между собой по управлению. По его структуре строится стохастический граф, содержащий $M + 2$ вершин. Вершина 0 означает исток, а вершина $M + 1$ – сток графа. Каждый ПМ вызывается на решение задачи с определённой вероятностью, исходя из целей функционирования или значения исходных данных. Прямая задача заключается в нахождении вероятности безошибочного решения задачи ПК, если известны вероятности безошибочных решений задач всех ПМ. Обратная задача заключается в том, что необ-

ходимо найти максимум вероятности безошибочного решения задачи ПК при ограничении на общее время отладки всех ПМ, а также в определении минимального времени отладки всего ПК при заданной вероятности его безошибочного функционирования.

Прямая задача. Для определения вероятности безошибочного функционирования ПК воспользуемся методом расчёта вероятностно-временных характеристик пребывания заявок в сети массового обслуживания (СМО) [57]. Однако непосредственно в том виде, как описано в указанном источнике, метод неприменим, так как в СМО времена пребывания заявок в модулях суммируются. В нашей же задаче должен выполняться «принцип слабого звена», свойственный основному соединению элементов в теории надёжности. Поэтому неправильно использовать изображения Лапласа плотностей распределений времени до ошибок в модулях. Необходимо вместо них поставить вероятности правильной работы соответствующих модулей, производящих вычисления на заданных временных интервалах. При этом будем предполагать, что ошибки модулей как случайные события статистически независимы.

Рассмотрим матрицу $G = G(t)$, $t = (t_0, t_1, \ldots, t_{M+1})$, элементами которой являются произведения $p_{ij}P_i(t_i)$, $i, j = (0, 1, \ldots, M+1)$, где p_{ij} – вероятность перехода от i-го ПМ к j-му, а вероятность безошибочного функционирования i-го ПМ в течение времени t_i. Так как нулевая и $(M+1)$-я вершины фиктивные, будем полагать, что время нахождения в них равно нулю, а вероятности безошибочной работы – единице.

Введём понятие шага вычислений, понимая под ним единичный переход от одного ПМ к другому. Чтобы найти вероятности безошибочной работы за два шага, нужно просуммировать с соответствующими вероятностями произведения вероятностей по всем путям, содержащим две вершины (одна из них нулевая). Это достигается возведением матрицы G в квадрат. При возведении матрицы G в куб получаем вероятности безошибочного функционирования за три шага и т.д.

Построим матрицу

$$T = I + G(t) + G^2(t) + \ldots = I(I - G(t))^{-1}, \qquad (8.1)$$

где I – единичная матрица.

Элемент матрицы T с номером $(0, M+1)$ представляет собой выражение для вероятности безошибочной работы всего ПК с учётом всех возможных последовательностей вызовов отдельных ПМ.

В соответствии с правилами вычисления значений элементов обратной матрицы [14], выражение для вероятности безошибочной работы ПК будет равно:

$$Y(t) = Q(t) / R(t), \qquad (8.2)$$

где $Q(t)$ – алгебраическое дополнение элемента с номером $(M+1, 0)$ матрицы $(I - G(t))$, а $R(t)$ – главный определитель матрицы $(I - G(t))$.

Выполнив указанные преобразования, получим искомое выражение для вероятности безошибочного функционирования ПК с учётом задействования всевозможных маршрутов вычислений.

Обратная задача. Определим минимальное время отладки ПК при заданной вероятности его безошибочного функционирования $P_{\text{зад}}$. Процесс отладки i-го ПМ обычно определяется временем его отладки τ_i. В известных аналитических и эмпирических моделях оценивания надёжности ПМ параметр τ_i, как и параметр t_i – заданное время работы i-го модуля, входят в соответствующее выражение для показателя надёжности ПМ. При этом предполагается, что оценки искомых показателей являются детерминированными известными выражениями, определяемыми по результатам испытаний. Подобных моделей несколько, рассматривать их все не имеет смысла. В качестве примера, не нарушая общности подхода, приведём лишь одну из самых ранних моделей – модель J. Musa.

Вероятность безошибочной работы i-го ПМ, согласно данной модели, можно выразить следующей формулой:

$$P_i(t_i, \tau_i) = e^{-\lambda_i t_i e^{-\nu_i \tau_i}}, \qquad (8.3)$$

где λ_i – интенсивность проявления ошибки; ν_i – интенсивность отладки; t_i, τ_i – время вычислений и отладки ПМ.

Обычно $\lambda_i = 1/T_i$, T_i – начальное среднее время безошибочной работы ПМ; $\nu_i = K_i/(N_{\text{ош}\,i} T_i)$, K_i – коэффициент сжатия времени отладки (тестирования) по сравнению с временем вычислений, $N_{\text{ош}\,i}$ – предполагаемое начальное число ошибок в ПМ.

Найдём минимальное время отладки ПК $\tau = \sum_{i=0}^{M+1} \tau_i$, при котором $P(t, \tau) \geq P_{\text{зад}}$, предполагая, что для ПМ справедливо (8.3). Решение получим на основе метода неопределённых множителей Лагранжа. Запишем функцию Лагранжа

$$F(t,\tau,\gamma)=\sum_{i=1}^{M+1}\tau_j+\gamma(P(t,\tau)-P_{\text{зад}}), \qquad (8.4)$$

где γ – неопределённый множитель Лагранжа.

Дифференцируя (8.4) по аргументам τ_i, γ и, приравнивая полученные выражения к нулю, получим систему уравнений:

$$\begin{cases}\dfrac{\partial F(t,\tau,\gamma)}{\partial\tau_i}=0,\\ P(t,\tau)-P_{\text{зад}}=0.\end{cases} \qquad (8.5)$$

Решая (8.5) относительно $\tau_i=\tau_i^0$ получим величину времени отладки ПМ

$$\tau^0=\sum_{i=1}^{M+1}\tau_i^0.$$

Теперь определим максимальное значение вероятности безошибочного функционирования ПК при заданном времени его отладки. В данном случае будем искать максимум функции $P(t,\tau)$ при заданном времени отладки $\tau_{\text{зад}}$. Функция Лагранжа в этом случае имеет вид:

$$U(t,\tau,\rho)=P(t,\tau)+\rho\left(\sum_{i=1}^{M+1}\tau_i-\tau_{\text{зад}}\right), \qquad (8.6)$$

где ρ – неопределённый множитель Лагранжа.

Дифференцируя (8.6) по τ_i и ρ, и, приравнивая результаты к нулю, получим систему уравнений:

$$\begin{cases}\dfrac{\partial U(t,\tau,\rho)}{\partial\tau_i}=0,\\ \sum\limits_{i=1}^{M+1}\tau_i-\tau_{\text{зад}}=0.\end{cases} \qquad (8.7)$$

Решая (8.7), находим искомые значения τ_i^0, и подставляя их в выражение для $P(t,\tau)$, найдём максимальное значение вероятности безошибочного функционирования ПК.

Пример 8.1. 1. Найти значение вероятности безошибочного функционирования ПК, стохастический граф которого показан на рис. 8.1.

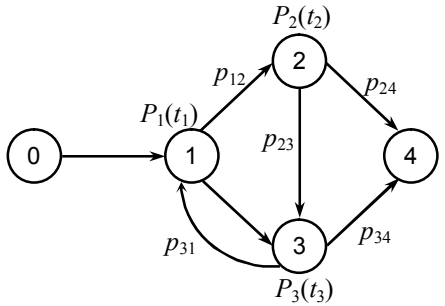

Рис. 8.1

Известны: $K_i=1$, $t_1=1$ с, $t_2=7$ с, $t_3=10$ с, $N_{\text{ош}1}=10$, $N_{\text{ош}2}=5$, $N_{\text{ош}3}=3$, $\lambda_1=0{,}01$ 1/с, $\lambda_2=0{,}02$ 1/с, $\lambda_3=0{,}03$ 1/с, $p_{12}=0{,}7$, $p_{13}=0{,}3$, $p_{23}=0{,}6$, $p_{24}=0{,}4$, $p_{31}=0{,}8$, $p_{34}=0{,}2$. Вероятность $P_i(t_i,\tau_i)$ описывается формулой (8.3). В данном случае $M=3$. Составим матрицы G и $(I-G)$:

$$G=\begin{vmatrix}0&1&0&0&0\\0&0&p_{12}P_1&p_{13}P_1&0\\0&0&0&p_{23}P_2&p_{24}P_2\\0&p_{31}P_3&0&0&p_{34}P_3\\0&0&0&0&0\end{vmatrix},$$

$$(I-G)=\begin{vmatrix}1&-1&0&0&0\\0&1&-p_{12}P_1&-p_{13}P_1&0\\0&0&1&-p_{23}P_2&-p_{24}P_2\\0&-p_{31}P_3&0&1&-p_{34}P_3\\0&0&0&0&1\end{vmatrix}.$$

Элемент матрицы $(I-G)^{-1}$ с номером (0, 4) согласно (8.2) $Y=Q/R$, где Q – алгебраическое дополнение элемента (4, 0) матрицы $(I-G)$. Раскрывая данные определители, получим:

$$Y=P(t,\tau)=\frac{p_{12}P_1p_{23}P_2p_{34}P_3+p_{12}P_1p_{24}P_2+p_{13}P_1p_{34}P_3}{1-p_{13}P_1p_{31}P_3-p_{12}P_1p_{23}P_2p_{31}P_3}. \qquad (8.8)$$

В (8.8) аргументы t_i и τ_i для краткости опущены.

Подставляя исходные данные в (8.8) и полагая $\tau_i = 0$, будем иметь $P(t,0) = 0{,}563$.

2. Пусть задано значение вероятности для ПК $P_{\text{зад}} \geq 0{,}999$. Требуется найти времена отладки каждого ПМ и суммарное время отладки ПК. Поступая, как описано в п. «Обратная задача», находим $\tau_1^0 = 3460$ с; $\tau_2^0 = 1871$ с; $\tau_3^0 = 848$ с; $\tau^0 = 6179$ с.

3. Пусть задано время отладки ПК $\tau_{\text{зад}} = 6000$ с. Требуется найти время отладки каждого ПМ τ_i^0 и максимальное значение вероятности его безошибочного функционирования при тех же значениях исходных данных. Получаем $\tau_1^0 = 3327$ с; $\tau_2^0 = 1838$ с; $\tau_3^0 = 835$ с; $\max P(t,\tau) = 0{,}999$. Приведённый числовой пример иллюстрирует возможность использования рассмотренного метода в решении прикладных задач, связанных с анализом надёжности ПК и обеспечением требуемых значений показателей их надёжности при отладке.

Разработчики П и лица, выполняющие отладку вновь создаваемых ПК, могут получать временные оценки для выбора оптимальных величин времени отладки ПМ и ПК в целом. При решении задач обеспечения заданной надёжности ПК нетрудно учесть с помощью коэффициентов K_i квалификацию разработчиков программ конкретных ПМ.

В данном методе могут использоваться любые модели «роста надёжности», полученные теоретически или на основе экспериментальных данных.

8.3. АНАЛИЗ И СИНТЕЗ ПОКАЗАТЕЛЯ НАДЁЖНОСТИ КОЛЛЕКТИВА ОПЕРАТОРОВ И ПРОГРАММНОГО КОМПЛЕКСА ПРИ ДИСКРЕТНЫХ ВРЕМЕННЫХ ПАРАМЕТРАХ

Задача, рассматриваемая в данном разделе, связана не только с ПО, но и с деятельностью человека-оператора, выполняющего некоторую отдельную элементарную функцию, например включение автомата, принятие решения в некоторой операции и т.д. При выполнении сложных функций оператором их необходимо представлять в виде совокупности последовательных действий, элементарных операторов. Главная цель, преследуемая в этом разделе, разработка модели элементарного дискретного действия оператора, способного обучаться. Можно использовать известные модели обучения [8], но это, по нашему мнению, представляется достаточно сложным для рассматриваемых здесь прикладных задач. Поэтому снова обратимся к модели J. Musa, и распространим её идею при представлении дискретного во времени процесса. Решив задачу для человека-оператора, дискретную модель можно также применить к оцениванию надёжности функционирования ПО с привлечением понятия его прогонов на некоторых исходных данных, как это рекомендуется в известной модели Э. Нельсона.

Построение дискретной математической модели оператора. В модели J. Musa вероятность безошибочной работы модуля ПО представлена формулой

$$P(t,\tau) = e^{-\lambda t e^{-\nu\tau}}, \qquad (8.9)$$

в которой t, τ – время вычислений и отладки ПМ; λ – интенсивность проявления ошибки, обычно $\lambda = 1/T$ где T – начальное среднее время безошибочной работы ПМ; ν – интенсивность отладки ПМ, $\nu = K/(N_{\text{ош}}T)$, K – коэффициент сжатия времени отладки (тестирования) по сравнению с временем вычислений; $N_{\text{ош}}$ – первоначальное предполагаемое число ошибок в ПМ. В работе [52] формула обобщена для независимых процессов функционирования и отладки ПМ и представлена в виде:

$$P(t,\tau) = e^{-\int_0^t \lambda(z,\varepsilon)dz \cdot e^{-\int_0^\tau \nu(u,\xi)du}}, \qquad (8.10)$$

где ε, ξ – режимы функционирования оператора в процессах работы и обучения. По аналогии с работой [44] показатель первой экспоненты назван ресурсом работоспособности (надёжности) оператора. Он равен:

$$P(t,\varepsilon;\tau,\xi) = \int_0^t \lambda(z,\varepsilon)dz \cdot e^{-\int_0^\tau \nu(u,\xi)du}, \qquad (8.11)$$

здесь $r(t,\varepsilon)=\int_0^t \lambda(z,\varepsilon)dz$ – ресурс надёжности, выработанный оператором за время t в условиях ε. Величина $v(\tau,\xi)=\int_0^\tau \nu(u,\xi)du$ была названа восполняемым ресурсом надёжности оператора, получаемым им в процессе обучения в режиме ξ для обучения будущей основной, профессиональной деятельности в режиме ε. Тогда с учётом принятых обозначений можно записать:

$$R(t,v)=r\cdot e^{-v}. \qquad (8.12)$$

В выражениях (8.9)–(8.11) сомножитель, представленный второй экспонентой, можно интерпретировать как вероятность уменьшения интенсивности отказа или ресурса надёжности оператора в основном режиме $\lambda(t,\varepsilon)$, $\int_0^t \lambda(z,\varepsilon)dz$. Причём, чем меньше эта вероятность, получаемая в процессе обучения оператора или тестирования ПО, тем больше вероятность безошибочного функционирования в основном режиме работы. Таким образом, процесс предварительной тренировки сводится к уменьшению величины вероятности, представленной второй экспонентой выражений (8.9)–(8.11), а сама она выражает степень прореживания интенсивности отказа или ресурса в основном режиме работы оператора или программного обеспечения.

Рассмотрим кратко содержание процесса выполнения человеком-оператором некоторого единичного элементарного действия, называемого в дальнейшем операцией. Многочисленные операции в технике выполняются за случайное время с ограниченными носителями его распределения – областями возможных значений. Они играют важную роль во многих прикладных задачах вероятностно-статистического исследования вариаций наблюдаемых в эксперименте признаков. К распределениям этого вида можно отнести равномерное, вырожденное, четырёхпараметрическое бета-распределение и другие. Даже если операция по времени характеризуется вырожденным распределением, плотность вероятности которой можно представить как

$$f(t)=\delta(t-T), \qquad (8.13)$$

где δ – дельта-функция Дирака; T – параметр распределения, и хотя интенсивность наступления события представляется дельта-функцией, израсходованный ресурс будет конечен. Поэтому формула (8.11) будет справедлива не только для распределений с неограниченными носителями, но и для распределений с ограниченными носителями. В дальнейшем в соответствии с формулой (8.9) будем использовать выражение для вероятности безошибочного выполнения элементарной операции оператором

$$P=p^d, \qquad (8.14)$$

где p – вероятность успешного выполнения элементарной операции оператором, не прошедшим предварительной тренировки или обучения; d – вероятность исключения ошибки в действиях оператора после завершения процесса тренировки или обучения, предшествовавших выполнению основной (рабочей) операции. Очевидно, что чем меньше величина d, тем успешнее были завершены предварительная тренировка или обучение оператора, тем больше будет величина P. Автору представляется, что без приведённого предшествующего исследования и изложения непосредственный вывод формулы (8.14) не очевиден. Ещё раз следует напомнить о том, что оба случайных процесса – выполнения операции и обучения её выполнению – независимы.

Формулу (8.14), в которой не содержатся значения непрерывных переменных t, τ, будем называть дискретным аналогом математической модели J. Musa.

Формализация процесса обучения.

Биномиальная модель обучения. Предположим, что возможно проведение n тренировок, из которых m должны привести к успешному результату. Вероятность того, что тренировка окажется успешной, равна α, а вероятность того, сто не успешной – $1-\alpha$. Тогда вероятность того, что не менее m из n тренировок будут успешными, равна:

$$d(m)=\sum_{i=m}^{n} C_n^i \alpha^i (1-\alpha)^{n-i}. \qquad (8.15)$$

Вероятность правильного завершения элементарной операции тренированным оператором будет равна:

$$P(m) = p^{\sum_{i=m}^{n} C_n^i \alpha^i (1-\alpha)^{n-i}}. \qquad (8.16)$$

Пусть $p = 0{,}7$; $\alpha = 0{,}8$; $m = 3$. Тогда $P(0) = 0{,}7$; $P(1) = 0{,}702$; $P(3) = 0{,}726$; $P(4) = 0{,}833$. При определённых значениях параметров в формуле (8.15) можно использовать и некоторые асимптотические оценки.

Модель обучения – модель роста надёжности Л.И. Волкова. Очевидно, что в выражении (8.14) могут использоваться все известные зависимости, представляющие собой модели роста надёжности ПО и модели роста надёжности техники, на которой проводятся доработки. В данном случае воспользуемся моделью доработок [12], которая успешно используется также при тестировании ПО с целью выявления ошибок в нём и повышения его надёжности. Приращение вероятности безошибочного решения выполнения j-го этапа обучения представляется в виде

$$\Delta R(j) = \alpha_j (1 - R(j-1)) - \beta_j R(j-1), \qquad (8.17)$$

где R_j – вероятность успешного выполнения этапа после устранения j-й ошибки при обучении, α_j – вероятность устранения j-й ошибки; β_j – вероятность необнаружения j-й ошибки на соответствующем этапе обучения. Переходя от соотношения (8.17) к рекуррентному выражению, полагая для простоты, что $\alpha_j = \alpha$, $\beta_j = \beta$, после некоторых преобразований получим следующее выражение для вероятности безошибочного выполнения j-го этапа обучения:

$$R(j) = R_\infty - (R_\infty - R_0)\left(1 - \frac{\alpha}{R_\infty}\right)^j \qquad (8.18)$$

где R_∞, R_0 – вероятности при числе этапов $j \to \infty$, $j = 0$ соответственно. Тогда в принятых обозначениях величина d будет равна:

$$d(m) = 1 - R(m) = 1 - R_\infty + (R_\infty - R_0)\left(1 - \frac{\alpha}{R_\infty}\right)^m. \qquad (8.19)$$

Параметры данной модели R_0, α, β должны определяться экспериментально, с использованием метода максимального правдоподобия. Величина $R_\infty = \alpha/(\alpha+\beta)$. Подставляя (8.19) в формулу (8.14) и вычисляя её значения при конкретных данных, получим величину вероятности безошибочного выполнения элементарной операции при условии, что в процессе предварительного обучения им успешно было завершено m этапов. При этом всегда необходимо, чтобы выполнялось условие нормировки, а это означает, что всегда $R_0 = 0$. Пусть $R_\infty = 0{,}95$; $\alpha = 0{,}7$; $\beta = 0{,}4$; $R_0 = 0$. Тогда величина $P(m)$ будет равна

$$P(m) = 0{,}7^{(0{,}05+0{,}95\cdot 0{,}36^m)}. \qquad (8.20)$$

Подсчёт по формулам (8.19) и (8.20) можно представить следующими рядами значений:

$$d(0) = 1,\ d(1) = 0{,}392,\ d(2) = 0{,}173,\ d(3) = 0{,}094,$$
$$d(4) = 0{,}066, \cdots, d(10) = 0{,}05;$$
$$P(0) = 0{,}7,\ P(1) = 0{,}87,\ P(2) = 0{,}94,\ P(3) = 0{,}967,$$
$$P(4) = 0{,}977, \cdots, P(10) = 0{,}982.$$

Формализация решения задачи для коллектива операторов. Предположим, что коллектив операторов включает в себя M операторов, связанных меду собой так, чтобы выполнить поставленную перед ним задачу. По структуре коллектива построим стохастический граф, содержащий $M+2$ вершин. Вершина 0 – исток, а вершина $M+1$ – сток графа. Каждый оператор вызывается на решение его задачи с определённой вероятностью, исходя из целей функционирования или значений, полученных предшествующим оператором исходных данных. Тогда на данном графе можно решить прямую и обратную задачи. Прямая – нахождение вероятности безошибочного решения задачи коллективом операторов при известных вероятностях безошибочных решений своих задач всех операторов. Обратная – нахождение максимума безошибочного решения задачи коллективом операторов при ограничении либо на общее время предварительного обучения операторов безошибочности их действий, либо на общую стоимость обучения. Может также предусматриваться нахождение общего минимального времени обучения или минимальной общей стои-

мости обучения коллектива операторов при заданной вероятности безошибочного решения им задачи. Формальное представление и решение указанных задач аналогично изложенному в 8.1.

Прямая задача. Введём матрицу $G = G(m)$, $m = (m_1, m_2, ..., m_{M+1})$, элементами которой являются произведения $p_{ij} P_i(m_i)$, $i, j = (0,1,...,M+1)$, где p_{ij} – вероятность перехода от i-го оператора к j-му; $P_i(m_i)$ – вероятность безошибочного решения задачи i-м оператором при условии, что он предварительно прошёл m_i циклов обучения. Далее строится матрица $T = I + G + G^2 + ... = I(I-G)^{-1}$. Элемент этой матрицы с номером $(0, M+1)$ представляет собой выражение для вероятности безошибочной работы всего коллектива с учётом всех возможных последовательностей вызовов отдельных операторов. Вероятность безошибочной работы коллектива операторов равна $Y = Q/R$, где Q – алгебраическое дополнение элемента с номером $(M+1, 0)$ матрицы $(I-G)$; R – главный определитель матрицы $(I-G)$.

Выполнив указанные преобразования, получим искомое выражение для вероятности безошибочного функционирования коллектива операторов с учётом задействования всех возможных маршрутов вычислений.

Обратная задача. Минимальное число тренировок всех операторов в процессе обучения при заданной вероятности безошибочного функционирования коллектива $P_{зад}$ будет равно $m = \sum_{i=0}^{M+1} m_i$, когда $P(m) \ge P_{зад}$. Предполагается, что справедливо выражение (8.13). Далее составляется функция Лагранжа с учётом ограничения типа равенство $F(m, \gamma) = \sum_{i=0}^{M+1} m_i + \gamma(P(m) - P_{зад})$, которая после дифференцирования по m_i, γ и приравнивания их нулю сводится к системе уравнений.

Решив эту систему уравнений, получим $m_i = m_i^0$, $m^0 = \sum_{i=0}^{M+1} m_i^0$.

Максимальное значение вероятности безошибочного функционирования коллектива при заданном общем числе тренировок операторов $m_{зад}$ находится после составления функции Лагранжа $U(m, \rho) = P(m) + \rho\left(\sum_{i=0}^{M+1} m_i - m_{зад}\right)$, последующего её дифференцирования с целью получения системы уравнений и решения последней. В результате решения получаем значения m_i^0, которые после подстановки в выражение для $P(m)$ приводят к искомому результату.

Обратная задача также может решаться, когда в качестве ограничений принимаются продолжительности тренировок операторов или стоимостные затраты на их проведение.

Пример прямой задачи. Группа из трёх операторов совместно выполняет задачу управления объектом. Алгоритм их взаимодействия представлен графом рис. 8.1. Исходные данные: $P_1 = 0{,}75$; $P_2 = 0{,}64$; $P_3 = 0{,}83$; $P_0 = P_4 = 1$; $p_{01} = 1$; $p_{12} = 0{,}7$; $p_{13} = 0{,}3$; $p_{23} = 0{,}4$; $p_{24} = 0{,}6$; $p_{31} = 0{,}2$; $p_{34} = 0{,}8$.

Процессы обучения операторов безошибочной работе определяются моделью Л.И. Волкова. Общее выражение для вероятности безошибочной работы оператора равно:

$$P_i(m_i) = P_i^{\left((1-R_i(\infty)+R_i(\infty)\left(1-\frac{\alpha_i}{R_i(\infty)}\right)^{m_i}\right)},$$

где $i = 1, 2, 3$ – номер оператора. Значения $R_1(\infty) = 0{,}92$; $R_2(\infty) = 0{,}98$; $R_3(\infty) = 0{,}89$; $\alpha_1 = 0{,}52$; $\alpha_2 = 0{,}74$; $\alpha_3 = 0{,}87$. Составляя матрицы G, $(I-G)$ и выполнив решение, как указано в 8.1, получим $P(m) = 0{,}89$, а полагая $m_i = 0$, будем иметь $P(0) = 0{,}468$.

Пример обратной задачи. Пусть требуется определить необходимое число тренировок для всех операторов коллектива, если задана его вероятность безошибочного функционирования

$P_{зад} = 0{,}90$. Поступая, как описано ранее решение обратной задачи, округляя численные значения до целых, получим: $m_1 = 3$; $m_2 = 2$; $m_3 = 1$; $\rho = -41$; $m = 6$. При $P_{зад} = 0{,}95$ получим: $m_1 = 7$; $m_2 = 4$; $m_3 = 2$; $\rho = -396$; $m = 13$.

Пусть требуется определить максимальную вероятность безошибочного функционирования коллектива операторов при $m = 12$. Поступая как ранее было изложено, находим: $P_{max} = 0{,}952$; $m_1 = 6{,}27$; $m_2 = 4{,}22$; $m_3 = 1{,}51$; $\rho = -1{,}24 \cdot 10^{-3}$.

Предположим, что стоимости тренировок операторов равны: $c_1 = 5$; $c_2 = 10$; $c_3 = 20$ условных единиц, а общая стоимость всех тренировок операторов равна $C = 150$ условных единиц. Требуется найти максимальное значение вероятности безошибочного функционирования коллектива. В результате решения задачи условной максимизации получим: $P_{max} = 0{,}955$ при $m_1 = 11{,}07$; $m_2 = 8{,}97$; $m_3 = 2{,}74$; $\rho = -1{,}62 \cdot 10^{-5}$.

Аналогичным образом решается задача с учётом ограничений на величину времени тренировок операторов.

Приводимые рекомендации могут применяться не только для оценивания выполнения элементарных операций, выполняемых человеком-оператором, но и для оценивания работы коллективов операторов, объединяемых единой целью управления. Для этого должна составляться модель функционирования коллектива, включающая в себя совокупность операторов и алгоритм их взаимодействия. При этом могут решаться задача анализа результата работы коллектива и задача синтеза ожидаемого результата его работы с предъявлением требований к качеству работы операторов при определённых ограничениях.

Данные рекомендации также могут успешно использоваться при решении подобных задач программного обеспечения, но не в терминах непрерывных временных параметров, а в терминах прогонов решений задач для отдельных его модулей с исходными данными из областей их спецификаций.

В отдельных случаях параметр обучения или тестирования и отладки может представляться не дискретном, а непрерывном виде.

8.4. СТРАТЕГИЯ ПОСЛЕДОВАТЕЛЬНЫХ ПРИБЛИЖЕНИЙ ДЛЯ ПОВЫШЕНИЯ И ОБЕСПЕЧЕНИЯ НАДЁЖНОСТИ СЛОЖНЫХ ПРОГРАММНЫХ КОМПЛЕКСОВ

На основе модели Седякина-Джелинского-Моранды для случайного потока событий и функции правдоподобия можно построить модель прогнозирования момента наступления очередного случайного события в потоке событий [52]. Данная модель позволяет по наблюдениям k интервалов между событиями – ошибками функционирования ПМ при условии справедливости существования экспоненциального распределения интервалов времени – определять оценку ожидаемого числа ошибок в модуле $\overset{0}{N}$ и оценку интенсивности проявления любой одной ошибки $\overset{0}{\eta}$. Интервалы между ошибками могут определяться в процессе тестирования и отладки ПМ, а также в процессе его функционирования при решении основной задачи. Средняя длина интервала времени до наступления очередного, ненаблюдаемого, прогнозируемого $(k+1)$ – события потока после k-го наблюдаемого события по прогностической оценке модели определяется по формуле

$$\bar{t}_{k+1} = \frac{1}{\overset{0}{\eta}\,\overset{0}{(N-k)}} \qquad (8.20а)$$

где $\overset{0}{\eta}$, $\overset{0}{N}$, k –оценка интенсивности ошибки, оценка числа ошибок в ПМ, число интервалов или зафиксированных ошибок в процессе тестирования и отладки ПМ.

Сложный программный комплекс (СПК) в общем случае состоит из m отдельных ПМ, каждый из которых в течение определённого времени должен выполнять присущую только ему вычислительную работу. Очерёдность работы ПМ СПК определяется управляющей программой и может зависеть от результатов работы ПМ-предшественников.

Рассмотрим одну из возможных стратегий прогнозирования моментов наступления очередных ошибок ПМ СПК, которая может обеспечить его функционирование при выполнении требуемых функций с заданной вероятностью.

Для всех m модулей СПК определяется число зафиксированных ошибок и интервалов времени между ошибками $k_{i,j}$, $t_{i,j}$, где i – номер ошибки и интервала, а j – номер модуля СПК, $i = 1,\ldots,k_j$; $j = 1,\ldots,m$. Для каждого модуля СПК находятся значения $\overset{0}{\eta}_j, \overset{0}{N}_j$ и по формуле (8.19.а) определяются средние продолжительности времени до проявления следующей ошибки у всех ПМ. Используя экспоненциальное распределение времени между ошибками, средние продолжительности времени до ожидаемых ошибок модулей и известные продолжительности непрерывной работы каждого модуля, находят значения вероятностей безошибочной работы всех ПМ. Используя эти вероятности и модель функционирования СПК, находят вероятность безошибочного функционирования всего СПК. Если эта вероятность не удовлетворяет заданному на неё требованию, то модули СПК подвергаются дальнейшей отладке и тестированию. Критичным становится тот модуль, у которого значение вероятности достигает наименьшего значения. Данный модуль первым подвергается дополнительному тестированию и отладке до выявления следующей ошибки и определения величины времени до неё. Для этого модуля производится определение новых значений оценок $\overset{0}{\eta}_j, \overset{0}{N}_j$ и вероятности его безошибочного функционирования за заданное время. Затем с учётом нового значения вероятности для этого критичного модуля вновь производят расчёт вероятности безошибочного функционирования всего СПК, как это было показано в 8.3. Если и это значение вероятности не удовлетворяет предъявляемому к ней требованию, то определяется следующий критичный модуль, производится его дополнительное тестирование и отладка, рассчитывается вероятность безошибочного функционирования данного модуля и всего СПК. Подобный процесс уточнения надёжности ПМ продолжается до тех пор, пока заданное требование по вероятности безошибочного функционирования СПК не будет достигнуто.

Другой разновидностью данной стратегии является достижение максимальной вероятности безошибочного функционирования СПК, когда задано общее время тестирования и отладки

СПК. В этом случае процесс отладки и тестирования начинается с критичного модуля и продолжается последовательно на модулях с возрастающей критичностью до тех пор, пока заданное общее время тестирования и отладки для СПК не будет израсходовано. Окончательный расчёт вероятности безошибочного функционирования СПК производится с помощью той же модели, которая объединяет все ПМ СПК в единое целое.

8.5. СИСТЕМНЫЙ ПОДХОД К ПОВЫШЕНИЮ НАДЁЖНОСТИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

8.5.1. Управление общей сложностью программного обеспечения

Работа над проектом ПО начинается с формулирования требований, которое ПО должно удовлетворять. При этом формулируются и функциональные требования, то есть формируется список функций, для реализации которых, собственно, и создаётся данное ПО. В процессе выбора списка формируется весь облик проекта с соответствующим уровнем его общей сложности. Необходимо осуществлять управление этим уровнем с целью обеспечения его приемлемого значения. Данная задача актуальна потому, что заказчик склонен предъявлять повышенные требования к проекту, но не в состоянии определить, чего это будет стоить (его надёжность, сроки создания, стоимость и др.).

Сложность ПО оценивается в зависимости от стадии его разработки, то есть по той информации, которая на этой стадии о ПО имеется. На рассматриваемой стадии сложность ПО предлагается измерять числом исходных команд K^k в нём, которое прогнозируется экспертом по информации о предыдущих разработках.

Трудность задачи, стоящей перед разработчиками ПО $(T^{\text{з}})$, определяется её интеллектуальной трудностью $T^{\text{и}}$ и ресурсными трудностями $T^{\text{р}}$, то есть $T^{\text{з}} = \varphi(T^k, T^{\text{р}})$, причём T^k, в свою очередь, определяется сложностью создаваемого ПО, поэтому $T^{\text{з}} = K^k \beta^b \beta^{\text{а}} \beta^{\text{р}}$, $\beta^b \geq 1$ – коэффициент, учитывающий степень научной новизны задачи для данного коллектива программистов, но-

визну используемого языка программирования и т.д.; $0 \le \beta^{a} \le 1$ – коэффициент, учитывающий адаптацию проекта к предыдущим разработкам, то есть степень использования в нём ранее разработанных П; $\beta^{p} \ge 1$ – коэффициент, учитывающий степень загруженности ресурсов СВТ.

Влияние уровня требований, предъявляемых к надёжности ПО на усиление трудности разработки ПО учитывается коэффициентом $\beta^{н} \ge 1$. Количественные значения приведённых коэффициентов определяются качественной оценкой экспертами факторов, соответствующих этим коэффициентам [5]. Например, требования к надёжности ПО или степень загруженности САТ могут оцениваться как низкие, средние, номинальные, высокие, очень высокие, сверхвысокие.

Таким образом, уровень требований к надёжности ПО, степень научной новизны проекта, степень его адаптации к предыдущим разработкам, степень ресурсных трудностей разработчиков ПО учитываются эквивалентным изменением числа команд проекта относительно исходного их числа. Вновь полученное число команд (полное число команд – K^{n}) и определяет на данном этапе трудность разработки ПО

$$T^{з} = K^{П}\beta^{b}\beta^{a}\beta^{p}\beta^{н}. \qquad (8.21)$$

Этот показатель является первичным для оценивания целого ряда показателей процесса проектирования ПО. Так, в работе [5] приведены зависимости между $K^{П}$ и затратами труда на разработку ПО, измеряемые в человеко-месяцах $A = \varphi(K^{П})$, между A и сроками разработки ПО в месяцах τ. По значениям A и τ может быть вычислено требуемое число исполнителей $n_l = A/\tau$.

Перейдём непосредственно к задаче выбора функций, реализуемых ПО проекта. Эта задача может быть сформулирована в различных постановках для различных ситуаций, встречающихся на практике. Рассмотрим одну из них. Предположим следующее:

– определено общее число программистов n_l, участвующих в создании ПО;

– определены как штатные вычислительные средства, на которых будет выполняться создаваемое ПО, так и их ресурсы: b^{r} – количество ресурса типа r, $r = \overline{1, n_r}$;

– определены сроки на разработку ПО $\tau^{д}$.

Пусть n_f – функций включено первоначально в список функций, желательных для реализации в создаваемом ПО. Далее, с помощью экспертов на основе опыта предыдущих разработок оценивается:

– исходное количество команд или операторов $K_i^{и}$, написание которых потребуется для реализации каждой i-й функции;

– количество ресурсов a_i^{r} типа r, требуемое для реализации i-й функции из r;

– эффект от реализации i-й функции ПО проекта C_i $(0 \le C_i \le 1)$.

В процессе выбора функций будем стремиться максимизировать суммарный эффект C от реализации функций, включённых в проект, при условии обеспечения требуемой надёжности ПО $H^{д}$ и выполнении ограничений на используемые ресурсы СВТ b^{r}, а также сроки разработки ПО $\tau^{д}$, задействованные специалистами n_l, то есть

$$C = \sum_{i=1}^{n_f} c_i x_i \to \max$$

при условиях

$$1)\ H_{ПО} \ge H^{д};\ 2)\ \tau \le \tau^{д};\ 3)\ n_l = const;\ 4)\ \sum_{i=1}^{n_f} a_i^{r} x_i \le b^{r}, \qquad (8.22)$$

где $x_i = \begin{cases} 1, \text{если функция входит в список реализуемых функций;} \\ 0, \text{в противном случае.} \end{cases}$

В (8.22) ограничениями 2, 3 определяется максимально возможный объём работы, который может быть выполнен за срок $\tau^{д}$ коллективом из n_l программистов, а значит – и $K^{П}$. Действительно, $A = n_l\tau$, $K^{П} = \varphi^{-1}(A)$, где $\varphi^{-1}(A)$ – обратная функция для

$\varphi(K^{П})$. С учётом известных коэффициентов β^{b}, β^{a}, зная требования, предъявляемые к надёжности проект $\beta^{н}$, и, ограничившись загрузкой СВТ, например, 75%, чтобы зафиксировать $\beta^{р}$ (упростить оптимизационную задачу) найдём из (8.21) допустимое число исходных команд $K^{и}_{доп}$ для данного проекта.

Таким образом, вместо ограничений на число программистов, СВТ и сроки разработки в задаче (8.22) введём ограничение на общую сложность создаваемого ПО, когда может справиться данный коллектив, удовлетворив требования по надёжности ПО и срокам его создания.

Тогда от (8.22) можно перейти к следующей постановке задачи:

$$C = \sum_{i=1}^{n_f} c_i x_i \to \max$$

при условиях

$$1)\ \sum_{i=1}^{n_f} K^{и}_{i} x_i \le K^{и}_{доп}\ ;\ 2)\ \sum_{i=1}^{n_f} a^{r}_{i} x_i \le b^{r}\ ;\ 3)\ x_i = \{0,1\}\ . \qquad (8.23)$$

Задача (8.23) является задачей дискретного программирования и для её решения могут быть использованы методы, применяемые для решения стандартной «задачи о ранце» [27].

Эта задача требует своей дальнейшей модификации, если проектирование осуществлять «сверху вниз». В соответствии с этим принципом после начального выбора функций осуществляется выделение в каждой их них подфункций, которые, в свою очередь, делятся на более мелкие компоненты и т.д. Этот процесс продолжается до тех пор, пока каждый компонент не станет представима в виде отдельного ПМ. При ПМ проектировщики руководствуются принципом модульности [10], который предполагает ограниченность размеров модулей, их функциональную самостоятельность и информационную независимость.

В связи с тем, что при выделении каждого компонента ПО идёт процесс уточнения числа исходных команд и ресурсов, необходимых для её реализации, на каждом уровне детализации необходимо решать следующую задачу:

$$\sum_{i=1}^{n_f} c_i x_i \to \max$$

при условиях

$$1)\ \sum_{i=1}^{n_f}\sum_{j=1}^{n_{il}} K^{и}_{i\xi j} x \le K^{и}_{доп}\ ;\ 2)\ \sum_{i=1}^{n_f}\sum_{j=1}^{n_{il}} a^{r}_{i\xi j} x_i \le b^{r}\ ;\ 3)\ x_i = \{0,1\}\ . \qquad (8.24)$$

Здесь $K^{и}_{i\xi j}$ – количество исходных команд j-го компонента i-й функции на ξ-м уровне детализации, а $a^{r}_{i\xi j}$ – количество ресурса типа r, требуемое для его реализации.

Процесс прекращается, когда на очередном уровне γ для любого j-го компонента любой i-й функции будет выполняться неравенство $K^{и}_{i\xi j} \le K^{и}_{доп}$, где $K^{и}_{доп}$ – допустимые размеры модуля.

Формализованные постановки задач (8.23) и (8.24) не учитывают тот факт, что на соответствующем этапе разработки ПО известны лишь приближённые значения $K^{и}_{i}$ ($K^{и}_{i\xi j}$) и a^{r}_{i} ($a^{r}_{i\xi j}$). Для учёта этого фактора необходимо применение теории нечётких множеств [22], в которой используется понятие нечётких чисел. Подход к решению оптимизационных задач с нечёткими исходными данными излагается в работе [38].

Следует отметить, что на практике задача управления сложностью создаваемого ПО может встретиться и в другой постановке. Например, наряду с выбором функций ПО определяются и штатные СВТ.

8.5.2. Управление трудностью проектирования программного обеспечения

Если до стадии проектирования ПО его разработкой занимается небольшая группа специалистов, то на этой стадии ПО делится на части (группы программ) таким образом, чтобы ответственность за реализацию каждой из них можно было бы возложить либо на одного разработчика, либо на группу исполнителей. При этом для каждой определённой таким образом части ПО выделяются необходимые ресурсы СВТ. Упомянутую выше группу исполнителей называют коллективом программистов (КП).

Считаем, что штатными СВТ для создаваемого ПО являются средства двухуровневой управляющей распределённой системы (УРВС), состоящей из центрального вычислительного модуля (ВМ) – (СВТ – верхний уровень) и ряда терминальных ВМ (ТВМ – нижний уровень). Предполагается, что распределение функций управления объектом программ между ВМ УРВС осуществляется на стадии проектирования. Задачу определения числа ВМ в УРВС, необходимых для реализации создаваемого ПО проекта и распределения множества программ по ним назовём распределительной (РЗ). При этом за коллективом программистов закрепляется группа программ, реализуемая на одном ВМ, то есть в процессе решения РЗ одновременно осуществляется распределение частей создаваемого ПО по (КП).

Трудность проектирования $T^{\text{п}}$ ПО будем измерять верхним уровнем трудности T_i задач, стоящих перед отдельными КП по созданию соответствующих групп программ, то есть $T^{\text{п}} = \sup\limits_{1 \le i \le n_k} T_i$, где n_k – число КП.

В процессе решения РЗ управление $T^{\text{п}}$ сводится к

$$T^{\text{п}} = \sup_{1 \le i \le n_k} T_i \to \min . \qquad (8.25)$$

При равноценных КП решение задачи (8.25) обеспечивается за счёт управления уровнем сложности соответствующих групп программ, а также уровнем ресурсных трудностей, которые определяются загруженностью ресурсов ВМ и характером распределения их между программами.

В работе [32] изложена методика, в которой сложность группы программ $\theta_i^{\text{г}}$ определяется взвешенной суммой числа связей по управлению S_1 и информации (числом глобальных S_2 и обменных переменных S_3) между всеми входящими в неё программами, то есть $\theta_i^{\text{г}} = \sum\limits_{j=1}^{3} \alpha_j S_{ij}$ где α_j – вес j-й связи.

Для оценивания ресурсных трудностей будем считать, что каждая программа полностью удовлетворяется всеми необходимыми ресурсами ВМ и они загружены не более, например, чем на

75%. Поэтому дополнительно учтём влияние на T_i только характера распределения ресурсов ВМ между программами. Для учёта этого фактора введём понятие «подкомплекс» программ, под которым понимается группа программ с выделенным под неё ВМ.

Введём понятие «сложность подкомплекса программ» θ, в котором наряду со сложностью соответствующей группы программ учтём и характер распределения ресурсов ВМ между программами. Тогда от задачи (8.25) перейдём к задаче

$$\sup_{1 \le i \le n_{\text{П}}} \theta_i \to \min . \qquad (8.26)$$

При оценивании θ_i, кроме явных связей между его программами (это связи по управлению и информации, которые названы необходимыми), надо учитывать неявные межпрограммные связи. Такими связями являются связи по ресурсам ВМ, совместно используемым программами одного подкомплекса. Подобные связи назовём вынужденными. Будем учитывать вынужденные связи по ресурсам памяти (оперативной и постоянной) и ресурсам процессора. Учёт при оценивании θ_i вынужденных связей между программами обусловлен необходимостью делить ресурсы ВМ между ними, а затем из-за их ограниченности идти на различные ухищрения, чтобы в них уложиться.

По аналогии с оцениванием сложности группы программ сложность подкомплекса программ будем определять взвешенной суммой числа их необходимых и вынужденных связей. Введём дополнительный коэффициент α_4, учитывающий одновременно вес вынужденных связей по всем ресурсам и названный поэтому обобщённым.

При определении значений весовых коэффициентов надо иметь в виду, что эффект усложнения подкомплекса за счёт связи по информации больше, чем за счёт связи по управлению, а у связи по глобальной переменной он больше, чем у связи по обменной переменной [30]. Конкретные значения α_j определяются группой экспертов, но при этом предполагается, что $0 \le \alpha_j \le 1$.

Подкомплексы программ для УРВС с определённой выше структурой могут быть соответственно двух типов – центральные или терминальные. Каждый i-й терминальный подкомплекс

включает в себя диспетчер ТВМ D_r и подмножество специальных программ N_i, а центральный подкоцплекс – центральный диспетчер $D_{\text{ц}}$ и подмножество специальных программ $N_{\text{ц}}$. Между D_r и всеми или частью специальных программ терминального подкомплекса $\nu \in N_i$ существуют связи (число их обозначим $S_{d\nu}$), которые можно отождествить с необходимыми связями по управлению между специальными программами. Кроме того, между D_r и любой ν-й программой так же, как и между всеми специальными программами, существуют вынужденные связи по ресурсам ТВМ.

Все необходимые связи терминального подкомплекса будем делить на внутренние и внешние. Внутренние связи определяют специальность программ внутри подкомплекса. Внешние – его связность с другими подкомплексами.

Определим сложность терминального подкомплекса. Считаем, что из двух подкомплексов сложнее будет тот, число усреднённых связей в котором больше. Каждый тип связи приводится к усреднённой с помощью соответствующего коэффициента α_j. Если учесть, что каждый подкомплекс имеет внутренние необходимые связи, которые определяют сцеплённость его элементов, и внешние связи, определяющие связность его элементов с элементами других подкомплексов, то формула для вычисления сложности i-го терминального подкомплекса в общем виде будет следующей:

$$\theta_i^{\text{T}} = \sum_{j=1}^{4} \theta_{ij} \ . \qquad (8.27)$$

Определим каждое из слагаемых. Для этого предварительно введём следующее обозначение: $n_i = n_i^c + 1$, где $n_i^c = \left|N_i\right|$. Тогда

$$\theta_{i1} = 1/2\alpha_4 n_i (n_i - 1) \qquad (8.28)$$

– компонентов θ_i^{T} обуславливается вынужденными связями;

$$\theta_{i2} = \alpha_3 \sum_{\nu \in N_i} S_{d\nu} \qquad (8.29)$$

– компонентов θ_i^{T} обуславливается связями по управлению между программой-диспетчером и специальными программами подкомплекса;

$$\theta_{i3} = 1/2 \sum_{\nu \in N_i} \sum_{\nu' \in N_i} w_{\nu\nu'} \qquad (8.30)$$

– компонентов θ_i^{T} учитывает внутренние для N_i необходимые связи;

$$\theta_{i4} = 1/2 \sum_{\nu \in N_i} \sum_{\nu' \in N \setminus N_i} w_{\nu\nu'} \qquad (8.31)$$

– компонентов θ_i^{T} учитывает внешние для N_i необходимые связи.

Центральный диспетчер выполняет внутреннюю функцию аналогично той, которую выполняет терминальный диспетчер, и внешнюю функцию. Для УРВС с определённой выше структурой внешняя функция центрального диспетчера заключается в том, что он непосредственно управляет обменом по общей магистрали между ТВМ. Поэтому организация этих связей, их тестирование и отладка возлагаются на коллектив программистов центрального подкомплекса. Подобные связи назовём несобственными связями центрального подкомплекса в отличие от его собственных необходимых связей, которые, в свою очередь, как и в ТВМ делятся на внутренние и внешние. Тогда формула для определения сложности центрального подкомплекса примет вид $\theta_{\text{ц}} = \sum_{j=1}^{4} \theta_{\text{ц}\,j}$ где первые два компонента вычисляются так же, как и соответствующие компоненты терминального подкомплекса:

$$\theta_{\text{ц}3} = 1/2 \sum_{\nu \in N_{\text{ц}}} \sum_{\nu' \in N_{\text{ц}}} w_{\nu\nu'} + \sum_{\nu \in N_{\text{ц}}} \sum_{\nu' \in N_{\text{ц}} \setminus N_{\text{ц}}} w_{\nu\nu'} \qquad (8.32)$$

– компонентов $\theta_{\text{ц}}$ учитывает собственные необходимые связи, как внутренние, так и внешние;

$$\theta_{\text{ц}4} = 1/2 \sum_{(\nu \in N_i,\, \nu \in N_j \setminus i \neq j)} \sum w_{\nu\nu'} \qquad (8.33)$$

– компонентов $\theta_{\text{ц}}$ учитывает несобственные связи.

Из анализа выражений (8.27)–(8.33) можно сделать первые выводы о возможности снижения уровня сложности программ-

ных подкомплексов УРВС. Так, представление каждому коллективу программистов под собственную группу программ собственного ВМ уже упрощает задачу, стоящую перед этими коллективами, так как устраняет вынужденные связи программ данной группы с программами других групп. Для центрального же подкомплекса дополнительное (в предположении, что множество его специальных программ определено) снижение сложности возможно за счёт сокращения числа несобственных связей (уменьшения связности программных подкомплексов). Для терминальных подкомплексов – за счёт уменьшения необходимых связей как внутренних, так и внешних.

Таким образом, снижение общего уровня сложности для терминальных и центрального подкомплексов может быть обеспечено в ходе решения распределительной задачи:

Найти

$$n_p = |M| \text{ и } f : N \to M , \qquad (8.34)$$

которые бы $\sup\limits_{1 \le i \le n_p+1} \theta_i \to \min$,

где M – множество вычислительных модулей; N – множество программ; i-й подкомплекс – терминальный, если $i \in \{1,2,...,n_p\}$; i-й подкомплекс – центральный, если $i = n_p + 1$.

Для решения (8.34) используется многошаговая процедура, обеспечивающая на k-м шаге $\theta_{\text{ц}}(k) \to \min$ при $\theta_i \le \theta^{\text{д}}(k)$, где $1 \le i \le n_p$, а $\theta^{\text{д}}(k)$ – допустимый уровень сложности терминальных подкомплексов на её k-м шаге. Решение (8.34) считается найденным (n_p^* и $f^* : N \to M$), если на очередном шаге k обеспечивается

$$(\theta_{\text{ц}}(k) - \sup_{1 \le i \le n_p} \theta_i(k)) \le \varepsilon ,$$

где ε – некоторая малая величина.

Таким образом, на k-м шаге работы процедуры решается распределительная задача, обеспечивающая

$$\theta_{\text{ц}}(k) \to \min \qquad (8.35)$$

при условиях

1) $\sup\limits_{1 \le i \le n_p} \theta_i(k) \le \theta_T^{\text{д}}(k)$; 2) $\sum\limits_{\nu \in N_i(k)} a_\nu^r(k) \le b^r$ для любых i и r,

где $N_i(k)$ – подмножество программ, образующих i-й программный подкомплекс; $a_\nu^r(k)$ – ресурс ВМ типа r, потребный для реализации программы ν; b^r – ограничение по ресурсу r, обусловленное возможностями ВМ.

При решении задачи (8.35) используются следующие исходные данные. Характеристики ν-й программы: t_ν – время решения, λ_ν – частота решения, $V_\nu^{\text{о}}$ – требуемая ёмкость оперативной памяти, V_ν^P – требуемая ёмкость постоянной памяти. Характеристики системы программ: $w_{\nu\nu'}$ – вес связей между программами ν и ν'. Характеристики ВМ: $V_{\text{ВМ}}^{\text{о}}$ – ёмкость оперативной при памяти ВМ, $V_{\text{ВМ}}^P$ – ёмкость постоянной памяти ВМ, $Z_{\text{ВМ}}^{\text{д}}$ – допустимая загрузка процессора ВМ.

Учитывая, что при фиксированном составе центрального подкомплекса $\theta_{\text{ц}}(k) \to \min$ обеспечивается за счёт $\theta_{\text{ц4}} = 1/2 \sum\limits_{(\nu \in N_i, \nu' \in N_j, i \ne j)} \sum w_{\nu\nu'} \to \min$, перейдём к следующей формализованной постановке задачи (8.34):

$$1/2 \sum_{\nu \in N \setminus N_{\text{ц}}} \sum_{\nu' \in N \setminus N_{\text{ц}}} w_{\nu\nu'} \bar{x}_{\nu\nu'} \to \min \qquad (8.36)$$

при условиях 1) $\theta_\nu \le \theta^{\text{д}} \forall \nu$; 2) $\sum\limits_{\nu=1}^{n_p} \lambda_\nu t_{\nu'} x_{\nu\nu'} \le Z_{\text{ВМ}}^{\text{д}} \forall \nu$;

3) $\sum\limits_{\nu=1}^{n_p} V_{\nu'}^{\text{д}} x_{\nu\nu'} \le V_{\text{ВМ}}^{\text{о}} \forall \nu$; 4) $\sum\limits_{\nu=1}^{n_p} V_{\nu'}^{p} x_{\nu\nu'} \le V_{\text{ВМ}}^{p}$; 5) $\sum\limits_{\nu=1}^{n_p} x_{\nu\nu'} = 1 \; \forall \nu'$;

$$x_{\nu\nu'} = \begin{cases} 1, \text{ если программы } \nu \text{ и } \nu' \text{ выполняются на одном ВМ} \\ \quad \text{с номером } \nu; \\ 0, \text{ в противном случае.} \end{cases}$$

Номер ν-й программы приписывается соответствующему программному подкомплексу, сложность которого равна θ_j.

Задача (8.36) является задачей дискретного программирования с булевыми переменными. Алгоритм её решения базируется на известном методе ветвей и границ [7].

8.5.3. Безошибочное кодирование программ

Условия для безошибочного кодирования программ создаются на стадиях, предшествующих стадии непосредственного кодирования. Этими условиями являются:

– обеспечение приемлемой общей сложности ПО проекта;

– обеспечение минимально возможного уровня трудности проектирования ПО при фиксированной его общей при ложности;

– использование при проектировании ПО принципов модульности и проектировании «сверху вниз».

Следование этим принципам позволяет, во-первых, выделить программные модули ограниченных размеров и, во-вторых, усилить их функциональную и информационную «прочность», что, в свою очередь, способствует написанию безошибочных программ. Так, в работе [7], отмечается существование некоторого оптимального размера программного модуля, при превышении которого количество ошибок в нём растёт уже нелинейно. Функционально «прочные» ПМ имеют по одному входу и выходу и реализуют только одну функцию. Информационная «прочность» ПМ обеспечивается усилением информационной сцеплённости его элементов, то есть объединением в ПМ элементов, сильно связанных между собой по информации и управлению, что, в свою очередь, ослабляет связность между ПМ.

Организация функционально прочных при условии соблюдения оптимальности их размеров, ослабление связности между ними, является фактором снижения сложности программ, состоящих из этих ПМ. Кроме того, снижение связности между ПМ ведёт к ослаблению волнового эффекта, то есть к сокращению путей распространения изменений или ошибок в одном ПМ на другие по связям между ними.

Важным организационным фактором обеспечения безошибочного кодирования является объединение программистов в специальным образом сформированные коллективы. В работе [10] приводятся различные подходы к формированию этих коллективов, например, в виде бригады главного программиста (БГП). Подобная бригада состоит из главного программиста, его помощника и библиотекаря программ. В бригаду могут по необходимости привлекаться дополнительно другие программисты. Главный программист разрабатывает и программирует основные части программ, а остальные распределяет между другими членами бригады. Помощник главного программиста не уступает по мастерству руководителю и помогает ему в разработке программ, формирует данные для тестирования, вместе с главным проверяет программы, составленные другими членами бригады. Библиотекарь программ отслеживает поступления и хранит все записи по соответствующей части проекта в библиотеке поддержки разработки. Дополнительные сотрудники вводятся, если часть проекта, закреплённая за данным КП, велика или необходимы специальные знания по применению аппаратных средств или ПО.

Таким образом, безошибочность кодирования при организации программистов в БГП обеспечивается за счёт выполнения следующих условий:

– осуществляется контроль опытными программистами (главным и его помощником) всего ПП, созданного бригадой;

– опытные программисты пишут большую часть программ, так как не отвлекаются, например, на документирование создаваемого ПО;

– опытный программист (бригадир) отвечает за всё ПО, созданное его бригадой;

– сокращается количество межличностных каналов передачи информации в ходе работы над проектом, так как на межбригадном уровне общаются только бригадиры, а не каждый программист с каждым (как при обычной организации). Это ведёт не только к сокращению непроизводительных временных затрат, но и повышает достоверность информации, циркулирующей на межбригадном уровне.

Рассмотрим факторы обеспечения безошибочного кодирования за счёт снижения трудности самого процесса программирования. К ним следует отнести:

– снижение уровней интеллектуальной и ресурсных трудностей на этом этапе;

– снижение сложности выделенных ранее ПМ (для них определены реализуемые ими функции и связи с другими ПМ);

– использование современных средств автоматизации процесса программирования.

Фактором снижения интеллектуальной трудности процесса программирования является использование языка высокого уровня. При этом повышается уровень абстрагирования при описании ПМ, на котором его сложность будет ниже, чем при представлении его же на языке более низкого уровня. Качество программирования повышается при задействовании программистов, знакомых с предметной областью, к которой относятся программируемая функция, с используемым языком программирования, с сервисными программными и аппаратными средствами и т.д. Для них интеллектуальные трудности решаемой задачи ниже, чем для иных специалистов.

Снижение уровня ресурсных трудностей может быть обеспечено ослаблением ограничений как на ресурсы штатных СВТ, так и на ресурсы СВТ разработчика ПО или их оптимальным распределением между пользователями, а также выбором приемлемых сроков разработки ПО.

Снижение сложности ПМ непосредственно в процессе написания П достигается применением структурного кодирования.

Структурное кодирование базируется на строго доказанной теореме о структурировании, которая утверждает, что любую правильную П можно написать только с использованием следующих логических структур:

– последовательности двух или более операторов (MOVE, ADD, …);

– выбора одного из двух операторов (IF, …,THEN,…,ELSE);

– повторения (или управления циклом) оператора, пока выполняется некоторое условие (DO,…,WHILE,…).

При этом каждая структура имеет один вход и один выход. Программу любого размера можно получить, применяя итерацию и вложение этих основных структур. В этом случае отпадает необходимость в безусловных переходах и метках. Таким образом, применение структурного кодирования позволяет уменьшать

сложность ПМ за счёт снижения разнообразия его элементов и связей между ними.

Структурное кодирование, принципы модульности и проектирование сверху вниз образуют, так называемую технологию структурного программирования.

8.5.4. Повышение надёжности программ на этапе отладки

В настоящее время принято полагать, что надёжность П определяется количеством дефектов, заложенных на стадиях проектирования и кодирования, которые при определённых условиях способны вызвать ошибку при выполнении П.

Отладка как один из основных этапов создания П призвана уменьшить число оставшихся дефектов в ней до количества, обеспечивающего приемлемую надёжность при эксплуатации П. Этап отладки включает в себя следующие основные операции:

– выбор тестовых входных наборов данных из всего множества входных наборов данных, допустимых для реализации на создаваемом ПМ, и их эталонных откликов;

– прогон ПМ на выбранных тестовых входных наборах данных и сравнение полученных результатов с эталонными откликами с последующим выявлением и локализацией дефектов;

– исправление выявленных дефектов ПМ, то есть собственно отладку.

Задача выбора тестовых входных наборов данных должна решаться с учётом следующих обстоятельств:

– характера алгоритма, описываемого создаваемым ПМ, и распределения переменных, являющихся исходными для данного алгоритма;

– характеристик вычислительной среды, в которой предстоит функционировать создаваемому ПМ; под вычислительной средой при этом следует понимать СВТ, на которых предполагается реализация создаваемого ПМ, а также программную среду – общесистемное и специальное ПО, с которым предстоит взаимодействовать П;

– характеристик возмущающих воздействий, влияющих на программную среду;

– структуры создаваемой П.

Процесс выбора тестовых входных наборов может быть осуществлён:

– вручную, когда каждый вариант исходных данных формируется разработчиком П;

– автоматизировано, с использованием специальных имитирующих П, рассчитывающих тесты непосредственно в СВТ, на котором функционируют тестируемые П;

– автоматизировано, с использованием универсального технологического СВТ для генерации тестовых наборов входных данных с последующим их введением в СВТ, на котором ведётся отладка ПМ;

– автоматизировано, с применением специализированной аналоговой и цифровой аппаратуры для генерации тестовых входных наборов данных.

Соответствие выбранных тестовых входных наборов данных характеру алгоритма, описанного создаваемым ПМ, и распределению входных переменных алгоритма может быть установлено на основе оценки состоятельности тестовых входных наборов данных, расчёт которой был изложен ранее.

Выбор тестовых входных наборов данных и их проверка на соответствие структуре создаваемой П могут быть осуществлены, если использовать подход, основанный на учёте чувствительности создаваемой П к воздействию тестовых входных наборов данных, выбираемых из различных подмножеств всего множества входных наборов данных.

При этом выбирается определённая совокупность входных наборов данных в виде точек в многомерном пространстве. Размерность пространства определяется количеством переменных, являющихся исходными для описанного в программе алгоритма. Этой совокупности ставится в соответствие совокупность выходных наборов данных, полученная в результате прогона на создаваемом ПМ совокупности входных наборов данных. Совокупность входных наборов данных и совокупность выходных наборов данных, представленные в виде точек в многомерных пространствах $x = \{x_i\}_{i=\overline{1,N}}$, $y = \{y_i\}_{j=\overline{1,M}}$, в общем случае $N \neq M$, аппроксимируются какой-либо достаточно гладкой кривой. В качестве такой кривой, обеспечивающей наибольшую гладкость при прочих равных условиях, может быть использован многомерный сплайн, получаемый в результате сплайн-аппроксимации, использующей в качестве узлов экспериментально полученные точки. Далее, процесс выбора эффективных входных наборов данных или проверка существующих входных наборов данных на соответствие данной П основывается на анализе гладкости сплайнов, аппроксимирующих совокупности входных и соответствующих им выходных наборов данных и выявлении интервалов немонотонного изменения сплайнов, аппроксимирующих совокупности выходных переменных при соответствующем монотонном изменении сплайнов, аппроксимирующих совокупности входных наборов данных. Выделение этих интервалов позволяет выбрать ограниченное количество тестовых входных наборов данных, соответствующих интервалам монотонного изменения сплайна, аппроксимирующего изменение выходных переменных и более полное количество тестовых входных наборов данных из подмножеств, соответствующих интервалам немонотонного изменения, аппроксимирующего изменение выходных переменных сплайна.

Тестирование, проведённое с учётом вида алгоритма и характера изменения его исходных переменных, поведения вычислительной среды, возмущающих воздействий и структуры создаваемой П и реализованное в объёмах, минимально необходимых для создаваемой П, позволяет существенно повысить надёжность П на стадии отладки.

8.6. ПОВЫШЕНИЕ НАДЁЖНОСТИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ПРИ РАЗРАБОТКЕ

Работа над проектом начинается с формирования требований к ПО. При этом формулируются и функциональные требования – список функций, которые надо реализовать в ПО АСУ. Список функций формирует весь облик объекта, соответствующий уровню его сложности. Заказчик склонен предъявлять повышенные требования к проекту, но не может предвидеть затрат на его создание (сроки, стоимость, надёжность и др.).

На фазе разработки ПО АСУ формируются различные требования по обеспечению качества и надёжности ПО. В стандартах и моделях ЖЦ ПО с различной глубиной определяется содержание этапов и частных работ при создании и модификации компо-

нентов ПО и ПО в целом. Эти модели служат базой работ и документов при реализации требований к показателям качества ожидаемых результатов. Необходимая надёжность ПО АСУ формируется и обеспечивается в процессе выполнения частных, работ каждого этапа (стадии) и удостоверяется испытаниями и документами. Показатели качества и надёжности регистрируются поэтапно, сопоставляются с заданными требованиями и корректируются при необходимости.

Для реализации мероприятий по планированию и управлению обеспечением качества и надёжности необходимы прежде всего организационные действия, направленные на обучение и воспитание специалистов двух категорий соблюдению технологии создания ПО.

Основные задачи специалистов ***первой категории*** заключаются в контроле качества процессов, результатов выполнения работ и принятии мер для достижения необходимого качества, обеспечивающего выполнение всех требований технического задания (ТЗ) на ПО АСУ.

Специалисты ***второй категории*** непосредственно создают компоненты и ПО в целом, с заданными показателями качества и надёжности. Их функции заключаются в тщательном соблюдении принятой в фирме технологии разработки и формировании всех предписанных руководствами исходных и отчётных документов.

Разделение специалистов на две указанные категории обеспечивает независимый, достоверный контроль качества результатов разработки ПО и эффективное достижение заданных характеристик его надёжности и безопасности.

Основным документом по управлению качеством ПО АСУ служит *план обеспечения качества и надёжности* (ПОКН). Сама формализация показателей качества отражается в *спецификациях* на ПО. Основными исходными данными для создания плана обеспечения качества и надёжности являются технические требования (ТТ) заказчика к ПО АСУ.

Для контроля характеристик ПО на промежуточных этапах (стадиях) разработки необходимы *эталонные данные*, к достижению которых должны стремиться разработчики компонентов ПО. Они получаются в результате декомпозиции исходных требований на ПО АСУ заказчика, деформируются в процессе разработки, а также формируются на основании опыта и аналогов создания предшествующих проектов.

Из-за отсутствия достоверных требований по качеству и надёжности ПО на начальных стадиях разработки, итерационного процесса их конкретизации в течение разработки показатели качества и надёжности ПО последовательно уточняются и корректируются при взаимодействии заказчика и разработчика.

Примером достаточно универсальной структуры плана может стать структура плана, рекомендованная стандартом «Руководство по планированию обеспечения качества программных средств» – ANSI/IEEE983-1986 [21]. В плане должны быть отражены цели управления качеством, методы управления и достижения уровней качества, базовые документы и стандарты для этапов (стадий) разработки, средства автоматизации разработки, структура и содержание отчётных документов.

На выполнение проекта разработки ПО АСУ предусматриваются ресурсы различных видов. При этом они подразделяются на ресурсы, необходимые для решения функциональных задач ПО и на ресурсы, требующиеся для обеспечения надёжности и безопасности функционирования ПО АСУ. Соотношение между ними зависит от ряда факторов. В различных видах ресурсы на обеспечение надёжности составляют от 5…20% до 100…300% от ресурсов, используемых на решение функциональных задач. Для критических военных систем они могут превышать последние в 2…5 раз.

Важным фактором, влияющим на обеспечение надёжности ПО сложных критических объектов, является доступность вычислительных ресурсов реализующих и технологических СВТ. При проектировании необходимо выделять определённые ресурсы на обеспечение надёжности. Ресурсы технологического СВТ по величине обычно значительно превосходят ресурсы реализующего ПО СВТ. Тип реализующего СВТ, его ресурсы, архитектура, система команд определяют возможность размещения на нём создаваемого ПО и средств обеспечения надёжности. При создании ряда типов ПО АСУ ресурсы реализующего СВТ ориентируются только на функционирование разрабатываемого ПО. Комплекс автоматизации разработки приходится размещать на технологическом СВТ. Важным фактором при выборе технологии разра-

ботки становится совместимость команд обоих СВТ. Выбор типа технологического СВТ зависит от его ресурсов и их соответствия ресурсам реализующего СВТ и параметрам разрабатываемого ПО АСУ. На технологическом СВТ должна размещаться, кроме того, база данных проектируемого ПО. Поэтому необходимые ресурсы технологического СВТ, как правило, значительно превышают ресурсы реализующего СВТ и объём создаваемого ПО.

8.7. ТЕСТИРОВАНИЕ КАК СРЕДСТВО ПОВЫШЕНИЯ НАДЁЖНОСТИ

Все фазы ЖЦ ПО должны поддерживаться методами и средствами систематического автоматизированного тестирования.

Тестирование – это процесс измерения качества, определения корректности и надёжности функционирования ПС на любых этапах (стадиях) ЖЦ в результате его испытаний. Для выполнения тестирования должны задаваться исходные данные и эталоны для принятия решений по результатам испытаний ПС в целом или его компонентов. Тестирование должно тщательно планироваться с учётом всех результатов, полученных на предыдущих стадиях ЖЦ. Основная задача при этом состоит в достижении максимальной достоверности испытаний, определения качества и надёжности ПС при ограниченных ресурсах на проведение тестирования.

Для современного ПС характерна последовательная реализация принципов модульного построения прикладных программных комплексов. Процесс тестирования тесным образом связан с такой реализацией ПС АСУ. ***Программный модуль*** – простая программа или группа простых программ, решающих достаточно простую функциональную задачу. Методы тестирования ПМ предназначены для обнаружения ошибок в их структуре и реализуемых маршрутах обработки информации (тестирование потоков управления) и выявления ошибок в вычислениях и преобразовании информации (тестирование потоков данных). В отличие от них методы комплексной отладки и тестирования сложного ПС существенно отличаются, так как в последних приоритет отдаётся проверке надёжности функционирования ПС при различных сочетаниях данных от внешней среды.

Различают детерминированное и стохастическое тестирование. При *детерминированном* тестировании задаются конкретные

совокупности исходных данных, конкретные значения эталонов результатов. Полученные в процессе тестирования результаты сравниваются с эталонными, что позволяет обнаружить или локализовать ошибку. *Стохастическое* тестирование использует в качестве исходных данных совокупности случайных величин, заданных распределениями. Таким распределениям исходных данных должны соответствовать эталонные распределения результатов, образующих в совокупности тесты. Оценка качества тестирования производится по степени соответствия полученных распределений результатов и их параметров эталонным распределениям.

По результатам тестирования осуществляется документирование испытаний ПС: планирование тестирования, спецификация тестов, отчёты о результатах тестирования.

Для повышения и обеспечения надёжности ПС используются следующие основные виды тестирования.

Значительную помощь в повышении надёжности сложного критического ПС может оказать избирательное систематизированное по видам тестирование, упорядочивание его проведение при испытаниях. Этот вид тестирования ориентирован на дифференциальное выявление определённых классов дефектов.

При заключительных испытаниях или сертификации ПС может проводиться интегральное тестирование при максимально широком варьировании тестов в условиях, близких к нормальной эксплуатации.

В реальном времени испытаний используются следующие виды тестирований:

– тестирование полноты решения функциональных задач при типовых исходных данных;

– тестирование функционирования ПС АСУ в критических ситуациях;

– тестирование для измерения достигнутых значений надёжности базовых версий ПС;

– тестирование корректности использования ресурсов памяти и производительности СВТ;

– тестирование параллельного выполнения программ;

– тестирование защиты от искажений исходных данных от сбоев аппаратуры, ошибок программ и данных;

– тестирование удобства эксплуатации и взаимодействия человека-оператора с ПО АСУ;

– тестирование базовых версий ПС при их переносе и изменении конфигурации оборудования.

Для выбора методов тестирования анализируют трудоёмкость («стоимость») и достигаемую результативность («эффективность»). В качестве показателей эффективности тестирования (методов и применяемых средств автоматизации) используют интенсивность (вероятность) обнаружения и устранения ошибок за единицу времени тестирования или значение достигаемой корректности (надёжности) ПС или его компонентов.

На начальных стадиях разработки ПС АСУ, когда в нём много простейших ошибок, наиболее полезны ручные методы тестирования. Они обеспечивают высокую интенсивность выявления грубых ошибок при сравнительно малых затратах.

По мере использования каждого метода сокращения числа и повышения сложности выявления ошибок его эффективность снижается, что уменьшает целесообразность применения метода. Применение же сложных и трудоёмких методов на ранних стадиях разработки не выгодно из-за значительного увеличения стоимости затрат.

Таким образом, для каждого метода, вида тестирования, типа объекта, этапа, времени существует область максимальной эффективности его использования. В других областях оказываются эффективными другие методы. В процессе отладки ПС уменьшатся число ошибок в ПС и, следовательно, снижается вероятность их обнаружения даже эффективными до этого методами.

В соответствии с областью активного использования для каждого метода имеется и область наиболее интенсивных затрат. Максимумы значений затрат соответствуют максимумам частости применения и выявления ошибок каждым методом. Усложнение методов приводит к возрастанию затрат на них. Рост максимумов затрат обусловлен ростом трудоёмкостей выявления ошибок и снижением вероятности их проявления при тестировании в дальнейшем.

По мере разработки ПО происходит постепенный переход к более сложным и дорогостоящим методам. На завершающей стадии динамической отладки и испытаний сложные методы становятся доминирующими. При стохастическом тестировании и тестировании в реальном времени из-за расширения областей варьирования переменных и увеличения числа тестовых значений возрастает эффективность обнаружения тестов на единицу затрат. Однако далее, по мере устранения определённых видов ошибок, интенсивность обнаружения оставшихся дефектов падает, что приводит к необходимости снижения затрат сначала на стохастическое, а затем на тестирование в реальном времени. На рис. 8.2 показаны качественные зависимости числа ошибок в ПС, средней наработки на ошибку ПС, эффективности методов отладки и тестирования во времени.

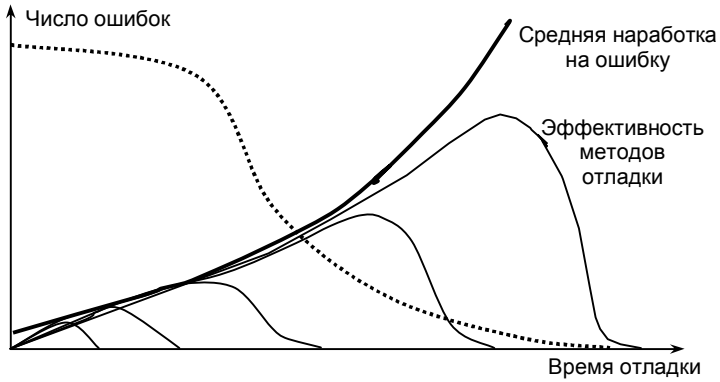

Рис. 8.2

Эффективность тестирования различными методами по мере их сложности возрастает. Для каждого метода она достигает точки максимума, покрывая левой частью кривой предшествующие кривые более простых методов. После точки максимума эффективность метода снижается, уступая место другому более сложному методу тестирования (тонкие кривые под кривой средней наработки на ошибку).

Приведённый качественный анализ показателя «эффективность / стоимость» методов тестирования во времени показывает пути возможной оптимизации надёжности ПО. Однако для выполнения оптимизации необходимы конкретные математические модели для каждого метода тестирования и выбранного вида ПО.

Сделаем несколько замечаний. Если интенсивное тестирование ПО в течение длительного времени не приводит к обнаружению ошибок или дефектов, то создаётся ощущение бесполезности дальнейшего тестирования данного ПО. Поэтому ПО передаётся в

эксплуатацию. Экспериментальные исследования сложного ПО позволили оценить темп обнаружения ошибок, при котором оно передаётся в эксплуатацию: 0,002…0,005 ошибок в день на человека [32]. Таким образом, при отладке и испытаниях выявляется только около одной ошибки каждые два месяца эксплуатации ПО. Темп обнаружения ошибок ниже 0,001 ошибок в день на человека (меньше одной ошибки в год на 3…4 человек, участвующих в отладке и эксплуатации) может служить, по-видимому, эталоном высокого качества отладки и надёжности ПО для систем обработки информации. Эти показатели соответствуют средней наработке на дефект или ошибку около $(5-10)\cdot 10^3$ час.

8.8. ПОВЫШЕНИЕ НАДЁЖНОСТИ С ИСПОЛЬЗОВАНИЕМ ИЗБЫТОЧНОСТИ

В процессе проектирования недостаточно создавать корректные П, дающие правильные результаты при конкретных исходных данных и отсутствии любых возмущений. Необходимо разрабатывать надёжные П, устойчивые к различным негативным возмущениям и способные обеспечивать требуемое качество результатов. Особенно это касается ПС, функционирующих в реальном времени.

Программное средство в процессе функционирования находится под воздействием шумов-возмущений различных видов из-за разных причин. Для снижения их влияния следует применять фильтры, обнаруживающие, селектирующие, устраняющие и уменьшающие их вредные последствия. Множество видов искажений приводит к необходимости построения системы фильтров, адаптированных к различным видам искажений. При этом основная задача сводится к максимально быстрому восстановлению качества функционирования ПС, ограничению последствий действия в нём дефектов. Для этой цели необходимы дополнительные вычислительные ресурсы, которые могут быть обеспечены за счёт введения в вычислительный процесс избыточности различного характера.

Временная избыточность – дополнительный ресурс времени СВТ, используемый для повторения или контроля вычислений (исполнения программ) и восстановления (рестарта) вычислительного процесса. Величина временного ресурса находится в пределах от 5…10% производительности однопроцессорного СВТ до 3…4-кратного резервирования в многопроцессорных СВТ.

Информационная (символьная) избыточность – дополнительный ресурс для дублирования исходных данных, промежуточных результатов вычислений. Используется для повышения достоверности обработки и хранения информации в СВТ. В ряде случаев она позволяет не только обнаруживать искажение данных, но и исправлять в них ошибки. Для данных, имеющих низкую важность, она может представляться помехозащитными кодами, только обнаруживающими ошибки (например, проверкой на чётность). Для данных высокой важности она может представляться кодами, исправляющими ошибки (например, код Хэмминга, мажоритарный код и др.).

Программная избыточность – дополнительный программный ресурс, использующий для контроля и обеспечения достоверности некоторых результатов обработки информации. При этом может осуществляться сопоставление результатов вычислений, обработки одинаковых исходных данных программами с различными алгоритмами и исключение искажений при несовпадении результатов.

Структурная избыточность – дополнительный аппаратный ресурс, используемый для повышения надёжности, достоверности вычислений, контроля процесса обработки данных и др.

Нагрузочная избыточность – дополнительная энергетическая нагрузка на элементы СВТ. По отношению к ПС в отличие от аппаратуры не нашла практического применения.

Следует отметить, что каждый из названных видов избыточности в чистом виде в СВТ и ПО реализован быть не может. Применение одного их них влечёт неизбежное использование других её видов.

Оперативный контроль (контроль работоспособности) ПС АСУ, исправление ошибок и восстановление вычислительного процесса сокращает ресурс производительности СВТ. Это сокращение ресурса отражается на уменьшении времени решения задач или на увеличении коэффициента простоя СВТ (коэффициента использования СВТ). Данный коэффициент характеризует относительные затраты времени на повышение надёжности функцио-

нирования ПС. Из-за усложнения средств помехозащиты П возрастают затраты времени СВТ на их реализацию. Это же приводит к снижению производительности и эффективности применения СВТ. Сами средства защиты являются сложными системами и служат новыми источниками ошибок, приводят к дополнительному снижению качества функционирования СВТ. Здесь мы сталкиваемся с вечным вопросом «кто же будет сторожить самих сторожей?».

С позиции изложенного аспекта ещё раз подчеркнём различие понятий корректности и надёжности ПС.АСУ. Для корректности введение оперативного контроля и восстановления ПС бесполезно и даже вредно. Это связано с возможностью появления дополнительных ошибок в результате взаимодействия основных функциональных и контрольно-восстановительных П. Однако именно средства контроля и восстановления хода вычислительного процесса позволяют повысить надёжность функционирования ПС.

Затраты на отладку, тестирование ПС оправданы повышением надёжности его функционирования. Их можно снижать затратами на оперативный контроль и восстановление функционирования ПС. Введение средств контроля и восстановления в ПС позволяет скомпенсировать его недостаточную отлаженность, снизить влияние возмущающих факторов на вычислительный процесс. Однако только этими средствами нельзя достигнуть высокой надёжности функционирования ПС. Необходимо рациональное распределение ресурсных затрат на тестирование и введение средств помехозащитны в ПО. В плане исследований можно поставить оптимизационную задачу распределения ресурсов на отладку и помехозащиту ПС с целью обеспечения заданной надёжности при минимальных затратах ресурсов или достижения максимальной надёжности при заданном ограничении на величину ресурсов. В работе [32] предлагается распределять ресурсные затраты на тестирование и введение средств контроля и восстановления ПС таким образом, чтобы максимизировать коэффициент готовности ПС. Коэффициент готовности представлен в виде:

$$K_{\text{г}} = k_{\text{г}} + (1 - k_{\text{г}}) P(t_{\text{в}} < t_{\text{д}}), \qquad (8.37)$$

где $k_{\text{г}}$ – вероятность в момент поступления данных застать ПС в работоспособном состоянии, $P(t_{\text{в}} < t_{\text{д}})$ – вероятность завершения

контроля и восстановления процесса обработки за время меньше допустимого $t_{\text{д}}$. Периодичность контроля ПС определяется выражением:

$$t_k \equiv \sqrt{2 t_1 T / (c_2 + 2c_3)}, \qquad (8.38)$$

где t_1 – продолжительность одного контроля, c_2 – удельные потери эффективности функционирования от необнаружения ошибки, c_3 – удельные потери эффективности функционирования из-за повторения программ, T – среднее время между ошибками в ПС. Формула (8.38) получена в предположении пуассоновского потока ошибок, знак пропорциональности вместо знака равенства применён из-за несоблюдения в ней размерностей использованных величин.

Для реализации любого проекта ПС необходимо проводить системный анализ и формировать требования к надёжности его функционирования. На основе этих требований нужно распределять затраты ресурсов на тестирование и отладку ПС и на создание средств оперативного повышения надёжности его функционирования в СВТ. Для создания таких средств должна разрабатываться группа программ оперативного контроля вычислительных процесса, основных программ и данных, а также средств автоматизации подготовки и реализации решений по восстановлению при обнаружении ошибок. Эта группа программ, обеспечивающая повышение надёжности ПС, должна объединяться с основными функциональными программами и подвергаться автономным и комплексным испытаниям в составе информационно-вычислительной системы.

8.9. СЕРТИФИКАЦИЯ ПРОГРАММНЫХ СРЕДСТВ

Сертификат ПС определим как документ, удостоверяющий его качество [21]. В международном стандарте ISO/IES-0002 – «Общие термины и определения в области стандартизации и смежных видов деятельности» ***сертификация соответствия*** определяется как действие третьей стороны, наряду с разработчиком и пользователем, доказывающей, что обеспечивается необходимая уве-

ренность в том, что идентифицируемая продукция (ПС) соответствует конкретному стандарту или нормативному документу.

Целью сертификации ПС является принятие решения о целесообразности выдачи сертификата с учётом трёх групп факторов:

– полноты, точности и достоверности исходных данных и измеряемых параметров, представленных в документации на ПС;

– достоверности, точности измерения и обобщения результатов испытаний и получения реальных значений показателей качества и надёжности;

– методологии и качества интерпретации данных о ПС с учётом достоверности оценок, квалификации и объективности испытателей, заказчиков и пользователей.

Срок действия сертификата ограничивается либо временем (например, 4 года), либо необходимостью модификации ПС. Он утверждает право на использование ПС, допускает его к эксплуатации (юридически) и использованию по прямому назначению. Сертификат является гарантией высокого качества, надёжности и безопасности применения ПС. В зависимости от области применения ПС, назначения, класса, его сертификация может быть *обязательной* или *добровольной*. Затраты на неё могут нести и разработчик и пользователь в зависимости от их взаимодействия.

Решение о выдаче сертификата на ПС основывается на оценке степени соответствия действующим или специально разработанным документам (ГОСТам, ОСТам, ТУ и др.).

Программа испытаний, методика их проведения, оценки результатов разрабатываются совместно разработчиком и заказчиком при участии специалистов по сертификации, согласовываются и утверждаются. Для проведения испытаний ПС создаются специализированные ***лаборатории сертификации*** (ЛС). Независимые от разработчиков и заказчиков ЛС могут иметь статус международных, государственных, ведомственных или фирменных. Создание проблемно-ориентированных ЛС может быть необходимым для повышения качества и надёжности специального ПС, используемого на объектах военной техники.

Глава 9

ФОРСИРОВАННЫЕ ИСПЫТАНИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ НА НАДЁЖНОСТЬ

В настоящее время в самых различных сферах народного хозяйства, военного дела и других отраслях человеческой деятельности получили широкое применение персональные компьютеры. Их программное обеспечение становится всё более сложным. В дальнейшем в этой тенденции будет наблюдаться прогрессирующий рост [58].

В работе [1] отмечается: «в мире постоянно происходят катастрофы, большие и малые аварии и всё чаще их причиной становится ненадёжное функционирование компьютерных систем, и в частности их программного обеспечения; оборона, авиация и космос, медицина, технологические процессы на современных ядерных, химических и других производствах – вот неполный перечень тех предметных областей, где низкое качество сервера и даже единичные дефекты в нём находят воплощение в терминах человеческих жизней и разрушающих материальных ценностей». Над подобными «ответственными» системами работает целая отрасль с огромными денежными затратами, располагающая значительным количеством высококвалифицированных программистов и проектировщиков, хорошо поставлен менеджмент, отлажены процессы разработки, испытаний и контроля. И тем не менее программное обеспечение даёт порой такой сбой, резонанс от которого бывает весьма громким [32].

Основными причинами возникновения ошибок являются недостаточно высокий уровень технологии производства программных средств и их чрезмерная сложность. Несмотря на то, что в области качества и надёжности программных средств за последнее

время достигнуты определённые положительные результаты и ошибки в процессе его функционирования сравнительно редки, проблема обеспечения высокой надёжности сложного программного обеспечения остаётся достаточно злободневной. Для решения этой проблемы нужен комплексный, системный подход. Охватить все стороны сложной проблемы повышения надёжности путём организации и проведения ускоренных и форсированных испытаний программного обеспечения в этой главе не представляется возможным. Здесь мы только наметим подход к получению количественных оценок надёжности программного обеспечения на основе известного, традиционного принципа теории надёжности технических систем – принципа форсированных испытаний.

9.1. ОБ ОДНОМ ФИЗИЧЕСКОМ ПРИНЦИПЕ ФОРСИРОВАННЫХ ИСПЫТАНИЙ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ НА НАДЁЖНОСТЬ

Причина обращения к ускоренным и форсированным испытаниям в теории надёжности и для техники, и для ПС одна и та же. Современная теория оперирует с численными оценками показателей. Хорошо известно парадоксальное противоречие: чем более надёжный продукт мы создаём, тем труднее оценить и подтвердить его надёжность. Однако на практике эту задачу как-то необходимо решать, чтобы при реализации системой цели можно было принять правильное решение. Сжатие информации об ошибках системы во времени в данном случае есть единственный выход из ситуации, когда в процессе её испытаний в реальном времени может не хватить ни времени испытаний, ни ресурсных затрат, или, в крайнем случае, сама система может морально устареть.

Применительно к ПО следует различать две разновидности ускоренных испытаний. К первой отнесём испытания программ с помощью компьютеров, работающих в реальном масштабе времени. Сжатие информации об ошибках здесь достигается либо более интенсивной загрузкой компьютеров задачами, либо преднамеренным сокращением ресурсных возможностей компьютеров определённой архитектуры. В этом случае по сравнению со штатным режимом эксплуатации компьютеры ставятся в более тяжёлые условия, приводящие к возникновению дополнительных «не-

расчётных» ошибок. Далее, на основе некоторых моделей, например моделей подобия процессов, эти ошибки переносятся или пересчитываются в условия штатного режима эксплуатации.

Вторая разновидность ускоренных испытаний – форсированные. Сущность их сводится к следующему. Программы, функционирование которых должно происходить в реальном времени и штатных технических условиях, подвергаются ускоренному прогону в компьютере. Это достигается увеличением тактовой частоты компьютера. Ячейки логических элементов, работая вблизи границ областей устойчивости, чаще ошибаются, чем в нормальных условиях. На «ненадёжное поле» ячеек компьютера накладывается процесс вычислений, траектория которого определяется алгоритмом программы, у которой в силу ряда причин могут быть свои потенциальные ошибки. Эти потенциальные ошибки в неустойчивой среде проявляются интенсивнее, чем в обычных условиях. Конечно, проявившиеся ошибки будут определяться и техническими и программными дефектами. Но это вполне соответствует реальным процессам вычислений в компьютере. Отделить причину возникновения ошибки одну от другой часто не представляется возможным – в ней «виноват компьютер». Пользователю же нужен правильный результат. Поэтому, на наш взгляд, подобное форсирование одновременно работы структуры компьютера и её программы целесообразно. Форсирование может обуславливаться и другими факторами, как, например, изменением напряжения питания элементов, температуры и др. Эквивалентность этих видов форсирований может устанавливаться экспериментально. Но они должны обязательно сочетаться с функционированием ПО.

Эта, вторая, разновидность ускоренных испытаний наиболее целесообразна для вычислительных алгоритмов типовых задач.

И, наконец, более сложными являются испытания при совмещении обеих разновидностей. В данном разделе делается упор на обсуждение форсированных быстродействием испытаний ПО.

Элементарная модель для коэффициента ускорения. Целью форсированных испытаний ПО является ускоренное обнаружение и устранение ошибок или повышение его надёжности благодаря тестированию компьютером обладающим большим быстродействием (производительностью) по сравнению со штатными

вычислительными средствами, для которых оно предназначено. Конечно, вычислительный процесс в обоих случаях должен быть различным. Но в определённых пределах подобия оба процесса вычислений могут быть адекватными. Это утверждение требует и дальнейшего изучения, и экспериментальной проверки [53].

Предположим, что длительность тестирования некоторого ПО на штатном компьютере с быстродействием B_0 равна τ_0 единиц времени. Вероятность ошибки на одну команду в компьютере с быстродействием B положим равной

$$\beta(B) = \beta_0 + \alpha(B - B_0)^m, , \qquad (9.1)$$

где β_0 – вероятность ошибки на одну команду в компьютере с быстродействием B_0, α – коэффициент пропорциональности вероятности ошибки быстродействию компьютера, степенной коэффициент, B – быстродействие технологического (испытывающего) компьютера. Спрашивается, каково должно быть время тестирования τ_T на технологическом компьютере с быстродействием $B > B_0$, чтобы вероятность отсутствия ошибки на штатном компьютере за время τ_0 была бы равна вероятности отсутствия ошибки на технологическом компьютере?

Считая вероятность ошибки на одну команду малой величиной, запишем выражение для вероятности отсутствия ошибки в программных средствах за время их испытаний на технологическом компьютере:

$$P(\tau_T) = (1-\beta)^{\tau_T B} = \exp(-[\beta_0 + \alpha(B - B_0)^m]B\tau_T) . \qquad (9.2)$$

Вероятность отсутствия ошибки на штатном компьютере за время испытаний τ_0 будет равна:

$$P(\tau_0) = (1-\beta_0)^{\tau_0 B_0} = \exp(-\beta_0 B_0 \tau_0) . \qquad (9.3)$$

Приравнивая (9.2) и (9.3), находим:

$$\tau_T = \frac{\tau_0}{\left[1 + \dfrac{\alpha B_0^m}{\beta_0}(K_B - 1)^m\right] K_B} , \qquad (9.4)$$

где $K_B = B / B_0$ – коэффициент форсирования быстродействия компьютера.

Пример 9.1. Пусть $\beta_0 = 10^{-4}$, $\alpha = 10^{-10}$ оп/с, $B_0 = 10^5$ оп/с, $B = 10^6$ оп/с, $m = 1$. Тогда из (9.4) получим $\tau_T = \tau_0 / 100$. Если $m = 1{,}1$, тогда $\tau_T = \tau_0 / 140$. Данный пример показывает реальную возможность реализации процесса форсированного тестирования ПО на технологическом компьютере для обнаружения и устранения ошибок. ▲

Приведённая модель и пример носят гипотетический характер. В реальных условиях модель пересчёта времени (9.4) должна строиться с учётом конкретных особенностей структуры ПО, штатного и технологического компьютеров.

Структура испытательного стенда. Стенд должен содержать следующие основные компоненты. Прежде всего его элементом является технологический компьютер, который должен иметь возможность изменять в достаточно широком диапазоне частоту задающего генератора тактовых импульсов, захватывая область неустойчивой работы. Частота может задаваться оператором непрерывно или дискретно. Практически легче реализовать дискретное изменение частоты.

Загрузка ядра – технологического компьютера – должна производиться типовыми программами последовательно после завершения предшествующих программ или фиксации их прерываний из-за обнаруженных ошибок. Этот процесс следует осуществлять с помощью управляющего компьютера, снабжённого соответствующим программным управлением.

Третьим элементом стенда является программный или аппаратно-программный контроль состояния функционирующей программы. При этом нужно осуществлять контроль окончания программы и правильности полученного результата, а также процесс срыва нормального её функционирования.

Четвёртый элемент стенда служит для накопления числа успешно выполненных прогонов и числа ошибочно выполненных на компьютере прогонов программ. Им же осуществляется и математическая обработка полученных статистических данных о надёжности ПО в результате форсированных испытаний.

Пересчёт результатов форсированных испытаний к нормальным условиям. *Стационарный процесс Пуассона*. Статистические данные об ошибках отражают влияние различных факторов

(исходных данных, аппаратурной неустойчивости, собственных ошибок программного обеспечения и др.). Поэтому модель в интегрированном виде должна отражать влияние комплекса воздействующих факторов. В силу редкости возникновения ошибок (даже при сильном ужесточении быстродействия компьютера) можно полагать справедливым применение закона Пуассона, согласно которому для стационарного случайного процесса вероятность появления k ошибок на интервале времени t будет равна:

$$p_k(t) = \frac{(\lambda t)^k}{k!} e^{-\lambda t}, \qquad (9.5)$$

где λ – интенсивность ошибок в функционирующей программе в некотором произвольном режиме и в предположении, что ошибки после обнаружения не устраняются. Но согласно нашему допущению о форсировании тактовой частоты величина интенсивности представится как $\lambda(B)$, поэтому

$$p_k(t, B) = \frac{(\lambda(B)t)^k}{k!} e^{-\lambda(B)t}. \qquad (9.6)$$

Положим

$$\lambda(B) = \lambda_0 + \gamma(B - B_0)^r, \qquad (9.7)$$

где γ – коэффициент пропорциональности, а r – коэффициент эффективности форсирования. Величины $\lambda(B)$, γ, B, r должны определяться из эксперимента. Коэффициент ускорения испытаний будет равен:

$$K_y = \frac{t}{x(t)} = \frac{\lambda(B)}{\lambda(B_0)}. \qquad (9.8)$$

Выражение (9.8) находится из условия $\lambda(B)x(t) = \lambda(B_0)t$, означающего равенство статистических ресурсов согласно физическому принципу Н.М. Седякина [44].

Пример 9.2. В результате эксперимента определены:

$$B_{1,2,3,0} = 400, 500, 600, 200 \text{ МГц}; \qquad (9.9)$$

$$\lambda(B_1) = 0{,}0017 \text{ с}^{-1}; \lambda(B_2) = 0{,}0055 \text{ с}^{-1}; \quad \lambda(B_3) = 0{,}0129 \text{ с}^{-1}. \qquad (9.10)$$

Составив систему трёх уравнений вида (9.7) с тремя неизвестными λ_0, γ, r, найдём значения этих величин:

$$\lambda_0 = \lambda(B_0) = 0{,}0001 \text{ с}^{-1}; \gamma = 2 \cdot 10^{-10} \text{ оп}^{-r} \cdot \text{с}^{r-1}; r = 3. \qquad (9.11)$$

Коэффициенты ускорения испытаний соответственно равны: $K_1 = 17$; $K_2 = 55$; $K_3 = 129$. Если требуемое время функционирования ПО в штатных условиях $t = 10$ с, то вероятность безошибочной работы равна $P(10) = e^{-10 \cdot 0{,}0001} = 0{,}999$. Время форсированных испытаний при быстродействии $B = 600 \text{ оп} \cdot \text{с}^{-1}$ составит $x(t) = \frac{\lambda(200)}{\lambda(600)} t = 0{,}08$ с. ▲

Нестационарный процесс Пуассона. Если в процессе испытаний программ выявлена зависимость интенсивности ошибок от времени, то для её пересчёта к интенсивности ошибок в нормальных условиях необходимо использовать выражение:

$$\int_0^t \lambda(z, B_0) dz = \int_0^{x(t)} \lambda(z, B) dz, , \qquad (9.12)$$

следующее из равенства вероятностей p_k для различных условий испытаний. Для нахождения $\lambda(t, B_0)$ из (9.12) дополним данное выражение инвариантом вида

$$B_0 t = B x(t), \qquad (9.13)$$

справедливым с точностью до постоянного множителя для левой и правой частей и означающим постоянство «энергетических затрат» на производство испытаний в двух различных режимах при условии, что израсходованные ПО статистические ресурсы (9.12) одинаковы. В работе [54] условие (9.13) названо «линейным энергетическим постулатом». Решая (9.12) и (9.13) совместно относительно $\lambda(t, B_0)$, получим:

$$\lambda(t, B_0) = \frac{B_0}{B} \lambda\left(\frac{B_0}{B} t, B\right). \qquad (9.14)$$

Данное выражение и может служить в качестве модели пересчёта результатов форсированных испытаний к нормальным условиям. Нетрудно заметить, что (9.14) сохраняет инвариантом любое распределение времени до ошибки, то есть не изменяю-

щим вид закона распределения в зависимости от режима форсирования. Наряду с (9.14), являющимся самым простым выражением пересчёта, могут быть получены и другие, более сложные выражения, отвечающие другим моделям испытаний.

Пример 9.3. В результате форсированных испытаний некоторого ПО установлено:

$$B_0 = 200 \text{ МГц}; B = 600 \text{ МГц}; \qquad (9.15)$$

распределение времени между ошибками – в соответствии с распределением Вейбулла. Построена кривая интенсивности ошибок ПО в зависимости от времени. С кривой сняты следующие значения интенсивности:

$$t_1 = 20 \text{ с}; t_2 = 100 \text{ с}; \lambda_1(t_1) = 5 \cdot 10^{-6} \text{ с}^{-1}; \lambda_2(t_2) = 10^{-6} \text{ с}^{-1}. \qquad (9.16)$$

Интенсивность возникновения ошибок для распределения Вейбулла определяется выражением:

$$\lambda(t, B) = \lambda_0 \alpha \left(\frac{B}{B_0}\right)\left(\frac{B}{B_0} t\right)^{\alpha - 1}, \qquad (9.17)$$

где λ_0, α – параметры закона, являющиеся неизвестными. Подставляя значения указанных величин в (9.17) и учитывая, что $B / B_0 = 3$, получим

$$\alpha = 1 + \frac{\ln \frac{\lambda_2}{\lambda_1}}{\ln \frac{t_2}{t_1}} \approx 0{,}57; \ \lambda_0 = \lambda_2 \frac{B_0}{B}\left(\frac{B}{B_0} t_2\right)^{-\frac{\ln \frac{\lambda_2}{\lambda_1}}{\ln \frac{t_2}{t_1}}} \approx 4 \cdot 10^{-6} \text{с}^{-0{,}57}. \qquad (9.18)$$

Тогда интенсивность ошибок при произвольном быстродействии

$$\lambda(t, B) = 2 \cdot 2 \cdot 10^{-6}\left(\frac{B}{B_0}\right)\left(\frac{B}{B_0} t\right)^{-0{,}43} \text{с}^{-1}. \qquad (9.19)$$

При нормальном режиме эксплуатации ПО $B = B_0$, поэтому из (9.19) получаем

$$\lambda(t, B_0) = 2 \cdot 2 \cdot 10^{-6} t^{-0{,}43} \text{ с}^{-1}. \qquad (9.20)$$

▲

Рассмотренный принцип форсированных испытаний ПО нуждается в экспериментальной проверке. Эффективность испытаний различных видов программ также может быть установлена опытным путём, при этом могут быть предложены другие, более сложные, модели пересчётов результатов испытаний к нормальным условиям эксплуатации.

9.2. ФОРСИРОВАННЫЕ ИСПЫТАНИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ПРИ РЕДЕЮЩЕМ ПОТОКЕ ОШИБОК

В предыдущем разделе 9.1 рассмотрен принцип получения данных о надёжности функционирования ПО на основе форсирования быстродействия персональных компьютеров. Полученные данные испытаний представляются в виде плотности или функции распределения времени до ошибок, возникающих в процессе выполнения программ. Затем эти распределения пересчитываются в соответствующие распределения при нормальных (штатных) условиях выполнения программ. Пересчёт от одних условий к другим может быть выполнен только в том случае, когда известна зависимость интенсивности возникновения ошибки (или другой связанной с ней характеристики) от параметра форсирования, в качестве которого была выбрана тактовая частота компьютера.

Физическое обоснование этого принципа не было приведено, хотя, на наш взгляд, оно достаточно очевидно. Это следует, например, из результатов экспериментальных исследований, приведённых в книге [33]. Среднее время до отказа полупроводниковых элементов приборов по различным причинам обратно пропорционально частоте переключения их состояний. С другой стороны, среднее время между сбоями и ошибками ПО, по крайней мере, на порядок меньше среднего времени между отказами полупроводниковых элементов, что также установлено экспериментально. Тем не менее, положения предложенного принципа форсированных испытаний у отдельных программистов и схемотехников вызывают определённые возражения.

В данном разделе рассмотрим программные средства с редеющим потоком ошибок, устраняемых в них в процессе отладки и тестирования. Длительности между ошибками зависят от параметра форсирования процесса функционирования программ. На осно-

вании работы [54] вводится понятие «энергетического запаса прочности» на каждую ошибку программы. В определённых условиях ужесточения режима форсирования вычислений предполагается, что этот запас не изменяется. Он принимается в качестве инварианта испытаний. На его основе вводится уравнение связи, которое и используется для определения показателя надёжности ПО в нормальном (штатном) режиме функционирования.

Форсированные испытания программного обеспечения. В главе 4 была представлена модель Седякина-Джелинского-Моранды для редеющего потока ошибок ПО. Воспользуемся данной моделью.

Пример 9.4. Предположим, что некоторое ПО было подвергнуто испытаниям в штатном режиме эксплуатации. Зафиксировано и устранено $k = 5$ ошибок с интервалами между ними:

$$\tau_1 = 1 \text{ ч}, \tau_2 = 17 \text{ ч}, \tau_3 = 197 \text{ ч}, \tau_4 = 4257 \text{ ч}, \tau_5 = 8362 \text{ ч}.$$

В результате решения уравнений правдоподобия найдены значения оценок ошибок в ПО и интенсивности ошибки: $\widetilde{N} = 29{,}936$, $\widetilde{\eta} = 2{,}113 \cdot 10^{-3}$ ч$^{-1}$ Среднее время устранения всех ошибок ПО равно $T = 1891$ ч. Вероятность безошибочной работы ПО в течение 10 часов составит $P(10) = 0{,}577$. Вероятность устранения всех ошибок ПО за 3000 часов будет равна $P_{\geq 30} = 0{,}948$. Отметим, что статистические данные для этой модели – времена между ошибками – связаны со строго определёнными штатными условиями эксплуатации. Формально это можно представить как $\tau_i = \tau_i(B_0)$, где B_0 – штатный режим эксплуатации ПО.

Под режимом эксплуатации следует понимать весь комплекс внешних и внутренних условий, воздействующих на вычислительный процесс (температура, влажность, вибрационные воздействия и другие факторы). Но мы в дальнейшем под режимом эксплуатации будем понимать такой режим, когда все внешние и внутренние факторы, кроме только вычислительного быстродействия, сохраняются постоянными, а быстродействие вычислений может подвергаться изменению. Именно такой гипотетический режим, сводящийся к быстродействию реализации программ, и будем в дальнейшем обозначать символом B.

Кроме рассмотренного допущения будем также полагать, что при любом значении быстродействия вычислительный процесс для данного ПО не может изменять характер и число случайных ошибок. Конечно, это будет допустимо только для определённого скоростного диапазона, величину которого заранее предсказать невозможно.

На основании сформулированных допущений мы вправе полагать, что случайные величины – длительности интервалов между ошибками – являются некоторыми функциями, зависящими от быстродействия компьютера:

$$\tau_i = \tau_i(B). \qquad (9.21)$$

В соответствии с этим все ранее приведённые формулы, содержащие величины τ_i, включают в качестве аргумента переменную величину B. Так величина η будет представлена как $\eta(B)$, а величина N не зависит от B. Плотность вероятности длительности времени между $i-1$ и i-й ошибками примет следующий вид:

$$f_i(\tau) = f_i(\tau(B)) = \eta(B)(N - i + 1)\exp[-\eta(N - i + 1)\tau(B)]. \qquad (9.22)$$

Используя выражение (9.22), можно составить уравнения максимального правдоподобия, найти оценки величин $\eta(B)$, N и использовать их в соответствующих формулах расчёта показателей. В частности, если бы нам было известно, что за время эксперимента t_c было зафиксировано r ошибок, то неизвестные величины $\eta(B)$ и N были бы связаны зависимостью:

$$\eta(B) = \frac{r}{\sum_{i=1}^{r}(N - i + 1)\tau_i(B) + (N - k + 1)\left(t_c - \sum_{i=1}^{r}\tau_i(B)\right)}. \qquad (9.23)$$

Формула (9.23) по своей структуре напоминает формулу для оценки интенсивности отказа при испытании партии элементов, когда не все элементы партии отказывают. Однако в нашем случае формула (9.23) оказывается бесполезной, так как обе величины $\eta(B)$ и N неизвестны и их оценки могут определяться только в результате решения системы уравнений:

$$\begin{cases} \dfrac{\partial \ln[L(N,\eta)]}{\partial N} = 0, \\ \dfrac{\partial \ln[L(N,\eta)]}{\partial \eta} = 0, \end{cases} \qquad (9.24)$$

где L – функция правдоподобия. Отметим ещё раз, что эти уравнения справедливы только для конкретной величины быстродействия компьютера. Нас интересует такое испытание ПО, которое проводится при $B >> B_0$. Именно такое испытание и является форсированным. В результате его осуществления мы получим необходимые статистические данные о количестве ошибок времена между ошибками в сжатом временном интервале. Далее, если мы будем располагать уравнением пересчёта результатов форсированных испытаний к нормальным (штатным) условиям функционирования, то сможем получить оценки надёжности ПО в этих условиях.

Инвариант надёжности программного обеспечения. В статье [54] введено понятие «энергии прочности» технической системы. Это такой запас её прочности, израсходовав который система утрачивает свою работоспособность. Скорость расхода запаса прочности («мощность разрушения») зависит от интенсивности воздействующей на систему нагрузки. В силу неразрывности физических процессов и существования закона сохранения энергии можно утверждать, что имеет место «инвариантность до эпсилон[1] энергии прочности» системы в определённом диапазоне изменения воздействующих на неё нагрузок. «Энергию прочности» и соответствующий ей диапазон нагрузок измерить не представляется возможным. Но утверждение об инвариантности позволяет формально записать следующее выражение:

$$A = \int_0^t W(z,B)dz = const, \qquad (9.25)$$

где A – величина «энергии прочности» (запас прочности), $W(z,B)$ – величина «мощности разрушения прочности» (скорость разрушения) при значении B воздействующего фактора нагрузки, t – время израсходования «энергии прочности». Формула (9.25) имеет место только в области инвариантных режимов нагружения $B \in D_B$. В самом простейшем случае одного определяющего параметра на выходе системы и постоянной скорости расходования запаса прочности имеем:

$$\Delta = \zeta t = const,, \qquad (9.26)$$

[1] Термин заимствован из теории автоматического регулирования.

где Δ, ζ, t – соответственно запас прочности, скорость и время его расходования. Этот инвариант можно использовать в качестве уравнения перового приближения для пересчёта характеристик надёжности от одного режима нагружения на систему к другому.

Что может служить в качестве инварианта надёжности ПО? Согласно экспериментальным данным [33] число отказов и сбоев в работе компьютеров пропорционально частоте переключения их элементов. С другой стороны, траектория активных элементов компьютера определяется ходом функционирующей программы, зависящим от конкретных исходных данных и от выполнения условных переходов – предикатных соотношений.

Таким образом, в ходе реализации вычислительного процесса одновременно осуществляются процесс переключения элементов компьютера и процесс наложения траектории вычислений, то есть выбора тех элементов, которые необходимы для реализации именно данной траектории вычислений. Поскольку количество возможных программных ошибок предопределено, то для сформулированных ранее допущений время их выявления будет обратно пропорционально величине быстродействия B компьютера. Поэтому в самом простейшем случае в качестве инварианта надёжности программ можно принять:

$$Bt = const, \qquad (9.27)$$

где t – время наблюдения определённого количества программных ошибок. Выражение может быть использовано в качестве уравнения пересчёта показателей надёжности ПО в различных режимах по быстродействию.

Пример 9.5. Предположим, что форсированные быстродействием испытания проведены на технологическом компьютере с быстродействием, в 10 раз превышающим быстродействие штатного компьютера. Зафиксированы 5 ошибок со следующими длительностями времени между ошибками:

$$\tau_1 = 0{,}1 \text{ ч}, \tau_2 = 1{,}7 \text{ ч}, \tau_3 = 19{,}7 \text{ ч}, \tau_4 = 425{,}7 \text{ ч}, \tau_5 = 836{,}2 \text{ ч}$$

Эти числовые данные являются уменьшенными в 10 раз числовыми данными предыдущего примера. В результате решения уравнений максимального правдоподобия (9.33) получены следующие оценки:

$$\widetilde{N} = 102{,}406, \widetilde{\eta} = 5{,}666 \cdot 10^{-3} \text{ ч}^{-1}.$$

Следует отметить, что значения этих оценок весьма чувствительны к значениям начальных исходных данных N и η, но, тем не менее, оценки показателей надёжности оказываются достаточно устойчивыми. Среднее время безошибочного функционирования ПО равно: $T = 919$ ч. Вероятность безошибочной работы его в течение времени 10 часов составит $P(10) = 0{,}388$. Вероятность устранения всех ошибок за 3000 часов будет равна: $P_{\geq 103} = 1$. Если перейти к штатному режиму, то время непрерывного функционирования следует уменьшить в 10 раз и тогда вероятность безошибочного функционирования ПО будет равна 0,574. Алгоритмическая ошибка метода составит около 5%. Приведённый пример, конечно, носит иллюстративный характер.

Представленное поисковое исследование, основанное на аналогии исследований в области технических систем, может быть подвергнуто критике с различных позиций. Но актуальность данной тематики должна стимулировать поиск прикладных методов исследования проверки и подтверждения надёжности ПО различных систем. Ограничиваться моделями ускоренных испытаний, основанных на повышении интенсивности искажений исходных данных, расширения варьирования их значений, увеличения количества потоковой информации и загрузки вычислительных средств, а также ограничений объёма памяти и др., на наш взгляд, не является достаточным. Нужно сочетать различные пути.

В рамках проведённого исследования можно поставить некоторые вопросы:

1. Справедлив ли инвариантный подход при форсированных испытаниях сложных программных комплексов и реальных информационных систем?

2. Что должно лежать в основе инвариантов: сохранение энергии, количества информации или что-то иное?

3. Как строить технологические вычислительные средства, сохраняющие все свойства вычислительных процессов штатных средств?

4. Сложный вычислительный процесс представляется иерархической структурой. Как результаты испытаний отдельных его ветвей объединять воедино? Иначе говоря, как при проведении форсированных испытаний ПО решить прямую (анализ) и обратную (синтез) задачи его надёжности?

ГЛАВА 10

ОСНОВЫ ТОЧНОСТНОЙ ТЕОРИИ НАДЁЖНОСТИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

10.1. ЭЛЕМЕНТАРНОЕ ВВЕДЕНИЕ В ТОЧНОСТНУЮ ТЕОРИЮ НАДЁЖНОСТИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Потенциальным источником опасности и риска в сложных системах являются их информационные ошибки. В данной главе наряду с известным классическим походом построения теории надёжности ПО предлагается подход, основанный на исследовании точности его операторов.

«Вычислительные машины служат, в общем, для записи и хранения чисел, для действий над ними и выдачи результата в числовой форме» [11]. Они были созданы на основе успехов математики и в интересах решения задач математики. Их преимущества – быстродействие и высокая точность вычислений – сыграли огромную роль в решении ряда важных практических задач самого различного характера.

В настоящее время практически нет такой области науки и техники, человеческой деятельности, где бы ни использовались эти совершеннейшие технические средства. Бурное развитие техники связи и теории информации во многом обусловлено прогрессом в области вычислительной техники. В настоящее время СВТ служат не только для производства сложнейших и точнейших расчётов и управления различными объектами, но и для представления информации в нечисловом количественном, смысловом виде. Однако в каком бы виде информация ни была представлена вычислительными средствами, она в всегда в своей основе является числовой, цифровой.