УДК: 681.4

Утепбергенов Ирбулат Туремуратович – д.т.н профессор (г. Алматы, Казахская академия транспорта и коммуникаций им.М.Тынышпаева)

Иргебаев Дархан Бекетулы – магистрант (г. Алматы, Казахская академия транспорта и коммуникаций им.М.Тынышпаева)

ИССЛЕДОВАНИЕ И АНАЛИЗ ОСНОВ ТЕСТИРОВАНИЯ НАДЕЖНОСТИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

В рамках обзора показано, что оценка и анализ надежности программного обеспечения (ПО) должны осуществляться на каждом шаге создания отказоустойчивых программ. В любом случае очень важно использовать как можно больше априорной информации перед началом каждой фазы создания ПО, так как это обеспечит наиболее приемлемые и точные результаты. Так как нет единого подхода, методики и параметров для оценки надежности ПО, в настоящее время существует множество моделей и алгоритмов, проверенных на практике, но имеющих ряд недостатков. Проводится анализ методов, моделей и методологий в области создания надежного, отказоустойчивого ПО. Делается попытка систематизировать и унифицировать существующие методологии. Учитывается многоэтапность создания ПО и архитектурный аспект в рамках всего жизненного цикла отказоустойчивого программного обеспечения.

Интерес к оцениванию надежности программного обеспечения (ПО) возник одновременно с появлением программ. Он был вызван естественным стремлением получить традиционную вероятностную оценку надежности технического устройства (ЭВМ или персонального компьютера), работа которого в основном и предназначалась для функционирования ПО. Последнее было определено как одна из составляющих частей машины, поэтому подход к оцениванию надежности программной части первоначально мало отличался от оценивания надежности техники и заключался в переносе известных статистических методов классической теории надежности на новую почву, образовав ее отдельную ветвь – теорию надежности ПО [1–3]. В целом этот подход сохранился до сегодняшнего дня. Однако по мере развития вычисли- тельной техники пришло четкое понимание того, что ПО не просто составная часть вычислительной техники. В современных условиях развития цифровой техники специализированное ПО перестало быть принадлежностью одной вычислительной системы (как это было раньше), а стало использоваться на сотнях и тысячах аналогичных компьютеров (в основном персональных). Даже если не касаться вопросов информационной безопасности, проблема обеспечения устойчивого функционирования расчетных программ, выявления их ошибок сегодня крайне остро стоит перед разработчиками [4-6].

За прошедшие десятилетия было создано множество методов, методик и моделей исследования надежности ПО. Однако, к сожалению, единого подхода к решению этой проблемы предложено не было и, по- видимому, в ближайшее время не предвидится. Тем не менее при разработке сложных программных систем их создатели стараются в той или иной степени получить оценку надежности ПО. Более правильный подход заключается в последовательном оценивании надежности программ на каждом этапе разработки. Основная сложность при использовании статистических методов заключается в отсутствии достаточного количества исходных данных. Динамика выявления ошибок должна тщательно фиксироваться и обрабатываться. Важной проблемой является степень детализации элемента расчета надежности [7-9]. Выявить все связи обработки информации (как это порой предлагается) даже для достаточно несложной программы практически нереально. Исходя из этого, детализация элементов расчета надежности (условно называемых программными модулями) должна ограничиваться законченными

программными образованиями, которые, взаимодействуя между собой, составляют более сложное объединение (комплекс), надежность которого нас интересует. При этом допускается, что надежность самой вычислительной техники, операционной системы (ОС) и среды программирования полная, нас интересует лишь надежность функционирования специальных программных средств, решающих основную целевую задачу системы.

В данной работе проводится анализ методов, моделей и методологий в области создания надежного, отказоустойчивого ПО; делается попытка систематизировать и унифицировать существующие методологии; учитывается многоэтапность создания ПО и архитектурный аспект в рамках всего жизненного цикла гарантоспособного программного обеспечения.

Надежность программного обеспечения определяется как вероятность того, что программное обеспечение функционирует без сбоев, в определенной операционной среде, в течение определенного промежутка времени. Коэффициент надежности архитектуры ПО можно оценить по формуле (1)

$$R = \sum_{j=1}^{M} \sum_{i=1}^{N_j} PU_{ij} R_{ij} \quad (1)$$

где M – число уровней архитектуры ПО; N_j – число компонент на уровне j, j = 1, ..., M; U_{ij} – вероятность использования компонента; R_{ij} – надежность компонента.

Оценить данные параметры на фазе разработки архитектуры невозможно, численные показатели в модель берутся непосредственно на фазах кодирования и тестирования.

Фаза кодирования ПО. Количество ошибок в выполнимом коде напрямую зависит от ошибок, сделанных на стадии кодирования. Одним из самых часто используемых параметров оценки корректности ПО является плотность ошибок. Начальную плотность ошибок можно оценить как (2

$$D = C \cdot F_{ph} \cdot F_{pr} \cdot F_m \cdot F_s \quad (2)$$

где F_{ph} – коэффициент фазы тестирования; F_{pr} – коэффициент командного программирования; F_m – коэффициент опытности и «зрелости» процесса разработки ПО; F_s – коэффициент структурирования; C – константа, определяющая количество ошибок/KLOC (ошибок на тысячу строк исходного кода)

Коэффициенты F_{ph}, F_{pr}, F_m, F_s и C зависят только от мастерства и опыта команды разработчиков.

При оценке коэффициента командного программирования (F_{pr}) плотность ошибок зависит от конкретных людей, их опыта написания программ и отладки. Можно принять следующие значения параметра: «Высокий», «Средний», «Низкий». Числовые показатели определяются и задаются экспертом. Для коэффициента опытности и «зрелости» процесса разработки ПО (F_m) принимаются следующие значения параметра: «Уровень 1», «Уровень 2», «Уровень 3», «Уровень 4», «Уровень 5». Числовые показатели определяются и задаются экспертом.

Коэффициент структурирования (F_s) позволяет взять во внимание зависимость плотности ошибок от языка программирования (отношение количества кода на ассемблере и языка высокого уровня (3):

$$F_s = 1 + 0{,}4a, \quad (3)$$

где a – отношение количества кода на Ассемблере и языка высокого уровня. Предполагается, что код на ассемблере может содержать на 40 % ошибок больше

Фаза тестирования ПО. Фаза тестирования самая продолжительная и может занять до 60 % от всего проекта. Во время тестирования выявляется самое большое количество ошибок в ПО. Рассмотрим меры покрытия теста, используемые на данной фазе:

– покрытие выражений – доля всех условных выражений, выполненных во время теста;

– покрытие ветвлений – доля всех ветвлений, выполненных во время теста;

– покрытие предикатов – доля всех переменных, которые используются для проверки условий перед условным переходом.

В начальной плотности ошибок (2) коэффициент фазы тестирования (F_{ph}) принимает следующие значения [10-13]: «Тестирование модуля», «Подсистемы», «Системы», «Приемлемость». Числовые показатели определяются и задаются экспертом.

Параметры для оценки D по выражению (2) должны быть скорректированы с использованием данных, накопленных в результате деятельности разработчиков ПО. Коэффициент C обычно лежит в диапазоне от 6 до 20 ошибок/KLOC. Можно брать как средние значения, так и max и min значения для оценки диапазона плотности ошибок.

Для тестирования ПО берется набор выходов программы и наблюдается реакция системы. Если ответ отличен от ожидаемого, то ПО имеет хотя бы одну ошибку. Тестирование преследует две цели: увеличить надежность так быстро, как это возможно, и найти максимальное количество ошибок. С другой стороны, в течение испытаний цель состоит в том, чтобы оценить надежность, таким образом, уровень найденных ошибок должен соответствовать фактическому.

Методы тестирования программных средств можно разделить на два основных класса [14]:

1. «Черный ящик» (функциональное тестирование). Для тестирования рассматриваются только входы и выходы ПО. Внутренняя структура ПО во внимание не берется. Это наиболее общий способ тестирования.

2. «Белый ящик» (структурное тестирование). Для теста используются знания о структуре ПО.

На практике комбинация двух методов приводит к наилучшим результатам. Тестирование ПО как «черный ящик» требует функционального описания программы, хотя некоторая информация о структуре позволяет тестировщикам выбрать наилучшие пара- метры для входов системы. Выбор входов может происходить случайным образом или разбиваться на группы. Группы входов могут подвергаться более тщательному тестированию. Комбинирование двух способов выбора входных диапазонов тоже применяется для тестирования ПО.

Некоторые ошибки достаточно легко обнаружить. Это ошибки, имеющие высокий уровень обнаружимости. Ошибки, которые трудно обнаружить, называются ошибками с низким уровнем обнаружимости. Они появляются при очень редко встречающихся комбинациях значений входных диапазонов. В начале тестирования большое количество ошибок имеет высокий уровень обнаружимости. Они легко обнаруживаются и устраняются. На последующих стадиях остаются ошибки, имеющие низкий уровень обнаружимости.

Тщательность тестирования может быть измерена с помощью коэффициента полноты покрытия теста. Коэффициент покрытия ветвлений кода – более четкая мера, чем коэффициент покрытия выражений. Некоторые разработчики используют коэффициент покрытия ветвлений, равный 0,85, как минимальное значение коэффициента.

Чтобы оценить операционную (транзакционную) надежность ПО, тестирование должно быть выполнено в соответствии с операционными профилями. Операционный профиль – множество несвязанных и непересекающихся входных диапазонов и вероятностей их использования компонентами. Для разных ОС профили могут различаться. Получение операционного профиля требует разделения пространства

входных диапазонов на подмножества (листья) и затем оценки вероятностей использования каждого подмножества (листа). Подмножество с достаточно высокой вероятностью использования может быть разделено на зоны меньшего размера.

Моделирование роста надежности программного обеспечения. Средства для обеспечения требуемого уровня надежности ПО могут достигать 60 % ресурсов проекта. Следовательно, тестирование должно быть тщательным образом спланировано для реализации проекта к заданной дате. Даже после длительного периода тестирования дополнительное тестирование может выявить новые ошибки. ПО как результат проекта обладает должным уровнем надежности, но содержит ошибки. Для планирования и принятия решений используются SGRM (Software Growth Reliability Model) – модели роста надежности ПО [14]. В модели SGRM предполагается, что надежность растет пропорционально времени тестирования, которое может быть измерено време- нем использования процессора. Рост надежности определяют в терминах интенсивности сбоев λ(t) в зависимости от количества ожидаемых ошибок за время t μ(t), как [3]:

$$\lambda\,(\mathrm{t}) = \frac{\mathrm{d}}{\mathrm{dt}}\mu(\mathrm{t}) \tag{4}$$

Пусть количество ошибок за время *t – N(t)*. Предположим, что ошибки устраняются по мере обнаружения. За основу возьмем экспоненциальную модель. Предполагается, что количество найденных и исправленных ошибок пропорционально количеству существующих ошибок.

Можно показать, что β_1 определяется как

$$\beta_1 = \frac{\mathrm{k}}{\mathrm{SQ}\frac{1}{\mathrm{r}}} \tag{5}$$

где *S* – количество инструкций в коде; *Q* – количество объектных инструкций в каждой инструкции кода; *r* – уровень выполнения инструкции компьютером *k* – называется коэффициентом подверженности ошибкам и меняется от 1×10^{-7} до 10×10^{-7}; t измеряется в секундах выполнения процессорного времени:

$$N(t)=N(0)exp(-\beta_1 t) \tag{6}$$

где N(0) – начальное количество ошибок; общее количество ошибок за время t:

$$\mu(t) = N(0) - N(t) = N(0)(1 - exp(-\beta_1 t)), \tag{7}$$

в общем случае получаем

$$\mu(t) = \beta_0\,(1 - exp(-\beta_1 t)), \tag{8}$$

где β_0 – общее количество ошибок, которые могут быть обнаружены, равно *N(0)*. Это предполагает, что во время отладки не делаются новые ошибки.

Выражение для интенсивности сбоев использует равенство

$$\lambda(t) = \beta_0\beta_1\, exp(-\beta_1 t). \tag{9}$$

Экспоненциальная модель легка для понимания и применения. Преимущество этой модели в том, что параметры β_0 и β_1 четко определены еще до начала тестирования.

Подход SGRM может применяться при двух различных ситуациях.

1. До начала теста. Часто надо иметь предвари- тельный план тестирования. Для экспоненциальной или логарифмической модели можно оценить время для достижения требуемого значения интенсивности сбоев, MTTF или плотности ошибок.

Во многих случаях t должно измеряться в человеко-часах и должно быть умножено на соответствующий коэффициент, который определяется с использованием опыта предыдущих проектов.

2. Во время теста. Используя SGRM, можно оценить дополнительное время тестирования, необходимое для достижения желаемого уровня надежности. Основные шаги использования SGRM:

1. Сбор данных. Данные об интенсивности сбоев включают множество погрешностей и «шума». Данные часто нуждаются в сглаживании. Наиболее общая форма сглаживания – это использование группировки данных. Группировка включает разделение теста на интервалы, затем рассчитывается средняя величина интенсивности сбоев на каждом интервале.

2. Выбор модели и определение параметров. Лучший способ выбора модели – это положиться на прошлый опыт проектов, использующих такую методику. Экспоненциальная и логарифмическая модели для экстраполяции параметров используются чаще всего. Данные первых тестов содержат большое количество шума. На самых ранних стадиях тестирования параметры могут очень сильно отклоняться от реальных, и они не могут использоваться, пока не будут стабилизированы.

3. Выполнение анализа о количестве дополнительных тестов. Используя подходящую модель, мы можем оценить количество необходимых дополнительных тестов для достижения определенного уровня интенсивности сбоев. Из этого рассчитывается минимальное время, которое необходимо затратить на тестирование.

В модели SGRM предполагается, что во время тестирования используется единая методика поиска ошибок. Каждая новая стратегия изначально эффективна для поиска определенных классов ошибок, но дает «всплески» в интенсивности сбоев. Для этого надо использовать процедуру сглаживания. Большей проблемой является то, что ПО продолжает модифицироваться во время тестов. Если изменения были значительны, то приходится выбрасывать из выборки значения, полученные ранее.

Одним из важных параметров эффективности SGRM является коэффициент покрытия ошибок. Коэффициент покрытия ошибок C_d линейно зависит от $C_{\text{в}}$ – коэффициента покрытия ветвлений. Коэффициент покрытия ветвления показывает, насколько эффективно были «захвачены» все возможные пути ветвления программы:

$$C_d = -a + bC_{\text{в}}\,,\ C_{\text{в}} > 0 \qquad (10)$$

Значения параметров a и b зависят от размера ПО и начальной плотности ошибок. Преимущество использования коэффициентов покрытия в том, что они напрямую зависят от того, насколько тщательно программа исследуется. Для достижения высокой надежности ПО должна использоваться более строгая мера, такая как коэффициент покрытия предикатов. Предикаты – условные операторы, которые влияют на внутреннее поведение программы.

Выводы. Оценка и анализ надежности должны осуществляться на каждом шаге создания отказоустойчивого ПО. В связи с тем, что нет единого подхода, методики и параметров для оценки надежности ПО, в настоящее время существует множество моделей и алгоритмов, проверенных практически, но имеющих ряд недостатков. Основной недостаток всех моделей – «узкая специализация». Модели всегда привязаны к какой-либо одной фазе разработки ПО, а иногда имеют чисто теоретическую ценность из-за того, что в них должны быть использованы данные, полученные на других стадиях разработки ПО, а авторы таких моделей, как правило, отводят этому факту второстепенное значение.

Наиболее перспективным является подход, направленный на создание некоторой гибридной модели, включающей в себя все достоинства хорошо проверенных моделей и алгоритмов. Существующие модели и алгоритмы необходимо модифицировать таким образом, чтобы выходные данные одной модели стали входными для других; предусмотреть единство понимания параметров, пересчет единиц измерения, нормализацию данных и удовлетворение современным требованиям и тенденциям, например таким, как объектно-ориентированный анализ и программирование. Из данного обзора мы видим, что очень важно использовать как можно больше априорной информации перед началом каждой фазы создания ПО, так как это обеспечит наиболее приемлемые и точные результаты.

ЛИТЕРАТУРА

1. Липаев В. В. Обеспечение качества программных средств. Методы и стандарты. М. : СИНТЕГ, 2001. – 380 с.

2. Липаев В. В. Надежность программных средств. М. : СИНТЕГ, 1998.

3. Lyu M. R. Software Fault Tolerance. Published by John Wiley & Sons Ltd, 1996.

4. Ковалев И. В., Золотарёв В. В., Жуков В. Г., Жукова М. Н. Методика построения модели безопас- ности автоматизированных систем // Программные продукты и системы. 2012. № 2. С. 16.

5. Авиженис А. Н., Лапри Ж.-К. Гарантоспособные вычисления: от идей до реализации в проектах // ТИИЭР. 1986. Т. 74, № 5. С. 8–21.

6. Avizienis A. The N-Version approach to faulttolerant software // IEEE Trans. on Software Engineering. 1985. Vol. SE11, № 12. P. 1491–1501.

7. Tai A., Meyer J., Avizienis A. Performability Enhancement of Fault-Tolerant Software // IEEE Trans. on Reliability. 1993. Vol. 42, No. 2. P. 227–237.

8. Липаев В. В. Тестирование компонентов и комплексов программ. М. : СИНТЕГ, 2010. 400 с.

9. Характеристики качества программного обеспечения / Б. Боэм [и др.] М. : Мир, 1981. 208 с.

10. Боэм Б. У. Инженерное проектирование программного обеспечения: пер. с англ. М. : Радио и связь, 1985. 512 с. Математика, механика, информатика 89

11. Boehm B. W., Haile A. C. Information Processing // Data Automation Implications of Air Force Command and Control Requirements in the 1980's (CCPI – 1985). Vol. I. Highlights, Report SAMSO/XRS-71-1. U. S. Air Force Systems Command (NTIS: AD 900031L). Los Angeles, CA, 1982.

12. Ashrafi N., Berman O. Optimization Models for Selection of Programs, Considering Cost & Reliability // IEEE Transaction on reliability. 1992. Vol. 41, No 2. P. 281–287.

13. Zahedi F., Ashrafi N. Software reliability allocation based on structure, utility, price, and cost // IEEE Trans. on Software Engineering. 1991. Vol. 17, No. 4. P. 345–356.

14. Lyu M., Chen J. H., Avizienis A. Software diversity metrics and measurements // In Proc. IEEE COMPSAC 1992. Chicago, 1992. C. 69–78.