ОБЗОРЫ

УДК 621.039.58

НАДЕЖНОСТЬ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ СИСТЕМ БЕЗОПАСНОСТИ АЭС

Токмачев Г.В., Токмачев И.Г.

Актуальность проблемы

Программируемые технические средства (ПТС) все больше и больше используют в защитных и управляющих системах АЭС. Ожидается, что все будущие проекты новых АЭС или проекты серьезной модернизации существующих станций будут компьютеризированными [1]. Некоторые действующие АЭС уже оснащены системами безопасности, управляемыми ПТС, например система аварийной защиты реактора на АЭС Sizewell-B в Великобритании [2], АЭС Temelin в Чехии [3] и унифицированной корейской АЭС [4].

Причины, определяющие необходимость использования программируемых средств на АЭС, включают следующее [5]:

существующие аналоговые системы устаревают, недостает запасных частей, многие поставщики уходят из атомной отрасли;

заказы новых управляющих систем атомной отраслью слишком немногочисленны, чтобы поддержать поставщиков аналоговых средств;

технология программируемых управляющих систем является основной для других отраслей промышленности, поэтому поставщики аналоговых средств остаются только на узкоспециализированных секторах рынка;

программируемые системы будут более рентабельны, особенно при длительной эксплуатации;

программируемые системы позволяют легко расширять их функциональные возможности и внедрять эксплуатационные усовершенствования, необходимые для более эффективной эксплуатации АЭС.

Несмотря на очевидные экономические преимущества от внедрения программируемой технологии, у эксплуатирующих организаций существуют немалые сомнения из-за возможного риска. Этот риск связан с недостатком знаний и неопределенностью последствий от внедрения новой технологии.

Считается, что для компьютеризированных систем, имеющих адекватное резервирование, интенсивность отказов в основном будет определяться ошибками в программном обеспечении (ПО), а вклад отказов технических средств будет относительно мал [1]. Опасность отказа ПО объясняется тем, что это может быть общей причиной множественных отказов резервированного оборудования систем безопасности АЭС. Примерами таких отказов по общей причине на АЭС США являются отказ автоматики ступенчатого пуска дизель-генераторов на АЭС Turkey Point или системы отображения параметров на АЭС South Texas [6]. Отказы ПО могут возникать и в процессе его взаимодействия с техническими средствами. Авария на АЭС Crystal River является особенно показательной: отказ аппаратуры инициировал передачу ложной исходной информации для ПО, что вызвало аварию с потерей теплоносителя, хотя ПО и работало корректно [7]. В наихудшем случае ошибки в ПО могут вызвать отказ при управлении станцией, т.е. исходное событие, и также привести к неготовности защитной системы реагировать на этот отказ [1].

В соответствии с требованием основного документа РФ по безопасности атомных станций [8] проект любых систем управления должен содержать анализ надежности функционирования технических и программных средств и системы в целом. Поэтому оценка надежности работы ПТС, включая оценку надежности ПО, имеет большую актуальность для атомной энергетики.

Следует отметить, что аналогичные проблемы стоят и перед другими отраслями [9—11], где отказ ПО может иметь катастрофические последствия, а именно: военной сферой, медициной (например, использование ритмизаторов сердца), системами контроля авиационных полетов, железнодорожным транспортом, космическими исследованиями и др. Известно, что ряд космических полетов закончился неудачей из-за ошибок в ПО, в частности первый полет европейского космического корабля «Ариан-5» и миссия межпланетного космического корабля НАСА (США) на Марс [11]. Поэтому НАСА финансирует работы по разработке систематической методологии, позволяющей оценивать вклад ПО в риск в рамках выполнения вероятностного анализа безопасности (ВАБ) [12]. Опыт других отраслей может быть полезен для атомной энергетики.

Причины отказов ПО

При использовании ПТС на АЭС одной из основных проблем является огромное число возможных комбинаций исходных данных, поступающих от датчиков и преобразованных в цифровую форму (множество входов). Опыт показывает, что проведение тестов реальных систем не позволяет перебрать и оттестировать все возможные комбинации наборов данных из-за того, что число тестов (хотя их может быть десятки тысяч) на практике является только небольшим подмножеством множества входов [1].

Кроме того, часто отказы ПО трудно воспроизвести, так как они зависят от специфических, трудно наблюдаемых условий, например работы других программ на том же компьютере [2]. При выявлении отказа ПО проводится его модификация, при которой корректировка программы может быть полностью успешной, а может лишь частично решить проблему или внести новые ошибки в ПО, которые в будущем приведут к новым отказам [13].

Относительная простота увеличения функциональных возможностей программируемых систем приводит к тенденции создания настолько сложных систем, что даже экспертам трудно хорошо разбираться в системе как едином целом и ее влиянии на безопасность АЭС. Это повышает вероятность ошибок в ПО [1].

Таким образом, ПО вводится в эксплуатацию с определенным числом ошибок, и они выявляются случайно при работе ПО.

Так как существует неопределенность в том, какой набор данных будет выбран из их множеств при текущем требовании на срабатывание и, следовательно, будет ли ПО корректно работать с этим набором, то процесс отказов ПО является стохастическим. В работе [2] введено определение множества, которое включает текущий набор значений как исходных данных, так и внутренних переменных, в свою очередь зависящих от предыдущих наборов значений исходных данных. Этому множеству соответствует подмножество наборов данных, на которых ПО будет работать некорректно (рис. 1).

Надежность ПО зависит от условий его работы, т.е. условная вероятность отказа ПО является функцией наличия или отсутствия отказов взаимодействующих с ним технических средств [14]. Поэтому при анализе надежности ПТС следует учитывать следующие факторы [12]:ПО работает на компьютеризированных технических средствах, на которые влияют параметры окружающей среды (сила земного притяжения, влажность, температура, электромагнитное излучение, радиация и т.п.);

ПО получает исходные данные и передает выходные данные другим участникам процесса управления (другое ПО, персонал или технические устройства) через устройства ввода/вывода. Эти операции могут быть критическими для работы ПТС. (Например, в работе [15] проанализированы данные по 199 критическим для безопасности эксплуатационным нарушениям ПО во время семи космических полетов в 1989—2001 гг. Наиболее частой (35%) причиной нарушений были некорректная передача и получение данных, а не работа самого ПО.)

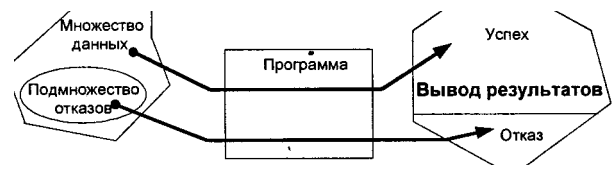

Рис. 1. Концептуальная модель процесса отказа ПО

По месту своего проявления отказы компьютеризованного оборудования могут быть разделены на четыре категории [12]:

отказы собственно ПО;

отказы со стороны ввода;

отказы при выводе данных;

отказы компьютерной платформы.

По своим последствиям большинство отказов ПО может быть классифицировано как один из следующих основных типов [5]:

некорректный вывод информации;

корректный вывод, но с недопустимой временной задержкой;

корректный вывод, но с нежелательными побочными эффектами;

отсутствие вывода информации на требова-
и(

Отказы исполняемых файлов ПО могут быть вызваны как ошибками при разработке ПО (ошибки в определении технических спецификаций для ПО, ошибки алгоритмизации и программирования, ошибки при компиляции из-за дефектов компилятора, ошибки конфигурации, такие, как ошибочное обращение к компоненту ПО), так и дефектами загруженного в память модуля ПО, вызываемыми физическими причинами (сбои в работе ячеек памяти, искажение информации в каналах связи), или ошибками пользователей [2].

Несовместимость различных программных компонентов может быть причиной катастрофических отказов ПО. На основании анализа опыта аварий в различных отраслях промышленности можно классифицировать виды несовместимости модулей ПО и их коренных причин следующим образом [7, 11]:

синтаксическая несовместимость, которая может быть вызвана несоответствиями в способах задания данных на уровне языка программирования, например, компонент-сервер может экспортировать числа с плавающей запятой, а программа клиента использовать рациональные числа;

семантическая несовместимость, которая может быть связана с логическими или физическими аспектами. Логическая несовместимость возникает, когда идентичные, синтаксически корректные символы используются для обозначения различных понятий, например, префикс Мега- может обозначать 1000000 в метрической системе или 1048576 при обозначении объема хранимой информации. Кроме того, даже синтаксически корректное использование точек и запятых при задании данных также может приводить к логической несовместимости. Несовместимость, связанная с использованием физических величин, возникает, когда одинаковые обозначения относятся к различным системам единиц измерения. (Такая ошибка вызвала потерю межпланетного космического корабля НАСА на Марсе, когда один модуль ПО экспортировал данные, отражающие физическую силу в фунтах-силы (британских единицах), а другой компонент воспринимал эти значения в ньютонах);

некорректное использование ПО, которое может произойти, если ранее разработанная программа применяется в качестве нового приложения, в частности после модификации аппаратной части системы. Две основные идеи содействуют продвижению этой практики: экономия затрат на разработке нового ПО и представление о высокой надежности старого ПО, уже проверенного в процессе его эксплуатации. Однако последний аргумент имеет силу, только если структура и режимы работы новой системы строго подобны старой. В противном случае может возникнуть нарушение входных связей или несоответствие диапазонов данных, используемых экспортирующим и импортирующим программными компонентами. (Из-за этого, например, закончился неудачей первый полет европейского космического корабля «Ариан-5», потому что программный модуль, обрабатывающий горизонтальную скорость и разработанный изначально для предыдущего проекта «Ариан-4», не был рассчитан на возросшие значения скорости.) Ограничения по входным связям могут быть обусловлены изменениями состояния системы, например при ее техническом обслуживании, а также при использовании нового ПО в старой системе. Опыт показывает, что обновление ПО с целью улучшить функциональные возможности системы часто приводит к тяжелым отказам, если при обновлении не учитывают физические пределы технических средств. Интерфейс между техническими средствами и ПО может рассматриваться как символ культурного интерфейса, который иногда становится барьером между специалистами по аппаратной части и ПО:

конфликт с операционной системой, который может возникнуть из-за ограничений параллельной работы и доступа к внешним ресурсам или нарушения требований к синхронизации, определяемых возможностями аппаратной части системы и пользовательского интерфейса. Так как управляемые процессы на АЭС имеют сложную многоуровневую природу, то ПО систем управления является многозадачным, когда необходимо одновременно реализовать сложные алгоритмы управления различными объектами, установленными на станции. При одновременном выполнении нескольких задач необходимо разделить ресурсы вычислительной системы в зависимости от их приоритета и различных событий, связанных с конкретными задачами. (Классический пример конфликта — так называемые «смертельные объятия», когда две задачи должны одновременно обменяться неразделяемыми ресурсами, но каждая может освободить свой только после выделения ей ресурса, который занят смежной программой.)

Способы защиты от отказов ПО

Существуют различные методы, используемые на этапе проектирования и внедрения программируемых систем управления для повышения их надежности и, следовательно, безопасности АЭС [5, 6, 16, 17, 18], —

апробированная практика разработки ПО. Один из апробированных подходов, который был использован во многих проектах управляющих систем как на АЭС, так и в неядерной промышленности, — разделение памяти системного и прикладного ПО, когда «чужая» память доступна только для чтения;

включение в ПО возможностей обнаружения и идентификации дефекта. Метод может быть реализован путем проверки достоверности (правильности) входной переменной при любом обращении к модулю и завершении работы с ним, вычисления контрольных сумм для исполняемых частей (чтобы гарантировать, что содержание ячеек памяти не менялось), а также сигнализации о несанкционированном вводе исходных данных. Резервирование на различных уровнях и в различных формах является ключевой мерой для выявления дефектов в системах, важных для безопасности. Существуют три основных типа резервирования (эта классификация применима как к ПО, так и к техническим средствам):

функциональное резервирование, когда функция выполняется двумя или несколькими различными каналами;

резервирование данных, когда информация хранится по крайней мере в двух местах или данные защищены кодом;

временное резервирование, когда функции выполняются в разные моменты времени;

разработка отказоустойчивого проекта. Считается, что практически невозможно разработать ПО без ошибок. Отказоустойчивость основывается на выявлении дефектов и соответствующей реакции системы, в результате чего или устраняется дефект, или система переводится в безопасное состояние, если это возможно. Целесообразно использовать анализ видов и последствий отказов с последующим формированием деревьев отказов для выявления критических ошибок в ПО и разработки средств их обнаружения и нейтрализации. Важным аспектом повышения отказоустойчивости проекта является внедрение программируемых управляющих систем в модульном исполнении с распределением функциональных и локальных логико-информационных задач;

защита технических средств и ПО от отказов по общей причине с использованием общих принципов повышения надежности систем (разделение, разнообразие и др.) и стратегии всесторонних проверок, в частности следующих подходов:

разработка системы из простых компонентов и с применением методов, упрощающих верификацию и валидацию;

использование стандартизированных компонентов системы так, чтобы все функции контроля и управления могли бы быть выполнены путем комбинации небольшого числа типов стандартизованных компонентов;

проектирование системы с высокой степенью независимости между функциями для ограничения последствий отказов;

резервирование ПО с использованием принципа разнообразия, чтобы обеспечить высокую функциональную надежность для критических функций.

Существуют различные способы применения принципа разнообразия в системах управления, построенных на ПТС [6], —

разнообразие технических средств, платформ и архитектур, в частности, могут использоваться разные процессоры от разных изготовителей. (Например, Интел 80486 и Моторола 6800 в системе аварийной защиты реактора АЭС Temelin и Sizewell-B);

разнообразие сигналов (алгоритмов, функций);

разнообразие проектирования, когда независимые фирмы (отделы, специалисты) проводят разработку, проверку, установку и верификацию разных каналов ПО;

разнообразие ПО, в частности различная архитектура или языки программирования (например, в системах, разработанных для АЭС Temelin использованы языки Ада и C++).

Применение принципа разнообразия при разработке резервированного ПО является таким же эффективным средством повышения надежности функционирования системы, как и при создании технических средств. Его реализация обеспечивает достаточно хорошую защиту против одновременно проявляющихся одинаковых программистских ошибок в резервированных компонентах ПО.

Кроме того, программные модули необходимо разрабатывать, предполагая, что другие части ПО или исходные данные могут содержать ошибки, которые нужно попытаться выявить. Поэтому работа диверсифицированного ПО обычно характеризуется большим числом внутренних сравнений и согласований входных, хранящихся и выходных данных и их гармонизацией.

Наиболее распространенные способы реализации разнообразия ПО — метод обновляющихся модулей или разработка нескольких версий ПО, работающих параллельно [5].

Основная цель метода обновляющихся модулей заключается в обнаружении отказов ПО в одном модуле в результате внутреннего тестирования, восстановление состояния устройства, существовавшего к моменту обращения к отказавшему компоненту, и повторное обращение к другому модулю с командой на выполнение той же функции [5]. Для того чтобы существовала возможность такого динамического изменения конфигурации, необходимо разработать несколько модулей.

Наиболее эффективным путем реализации принципа разнообразия признается независимая разработка двух или более версий одной программы, работающих параллельно с одними исходными данными таким образом, что модуль проверки рассогласования выходных значений может выдать один «лучший» результат [2, 5, 6, 16, 18]. Обычно независимость означает работу двух или более групп разработчиков без прямого контакта, хотя и с косвенным взаимодействием. Например, ошибка, найденная в общем техническом задании, по решению руководителя работ может внести коррективы в данные для всех групп. Разработчикам может быть предоставлена полная свобода в выборе методов и средств или определено «принудительное» разнообразие, например разные языки программирования и режимы тестирования программ. Считается, что «принудительное» разнообразие при разработке ПО является более эффективным методом защиты от отказов по общей причине, так как одинаковые технические требования могут быть источником множественных отказов диверсифицированного ПО из-за предрасположенности человека делать одни и те же определенные типы ошибок в сходных ситуациях.

Возможна также частичная реализация принципа разнообразия для защиты от повторяющихся программистских ошибок путем создания различий между требованиями на срабатывание резервируемых модулей (разнообразие данных). Это может быть сделано неявно с помощью неодновременного запуска программных копий и различного алгоритма опроса датчиков, от которых резервируемые копии ПО получают исходные параметры, или явно путем введения небольших различий при вводе данных [2, 19]. В частности, было показано, что надежность ПО можно повысить путем частых перезагрузок («омоложение» ПО), которые изменяют переменные состояния системы. При этом неверные или другие необычные значения, которые не тестировались для возникающих условий работы ПО, стираются.

Систематические дефекты ПО могут возникать не только из-за ошибок программистов, но и при трансляции (компиляции). Чтобы обеспе-

чить корректную трансляцию исходной программы в бинарный код, требуется проверенный на практике транслятор. Методом защиты может быть использование разных трансляторов (разнообразие технических средств), но это очень дорого. Опыт показал, что разнообразие ПО является дополнительной защитой против некорректной компиляции. В этом случае компилятор потенциально остается общей причиной отказов ПО, но одинаковые отказы разных версий диверсифицированного ПО будут проявляться в различные моменты времени и поэтому не будут приводить к одновременному отказу всей системы [16].

Кроме того, при отладочных и эксплуатационных проверках диверсифицированного ПО вероятность выявления отдельных ошибок увеличивается, что снижает потенциальную опасность множественных отказов. Возможности выявления отказов диверсифицированного ПО обусловлены независимостью групп разработчиков, точностью следования правилам программирования и защитой программ. Следует отметить, что эффективность своевременного выявления совпадающих скрытых отказов при использовании диверсифицированного ПО можно резко повысить, используя принцип разнообразия исходных данных (небольшое их варьирование) [19].

При разработке диверсифицированного ПО должны учитываться следующие потенциальные негативные моменты [2, 16]:

быстродействие и возможности такого ПО могут быть уменьшены из-за различий между версиями;

алгоритмы могут использовать разные критерии «просеивания» данных в разных версиях;

разработка резервируемых модулей ПО требует решения проблемы проверки рассогласования выходных значений;

применение принципа разнообразия значительно повышает стоимость разработки ПО.

Дополнительные затраты на реализацию принципа разнообразия складываются из нескольких составляющих:

разработка и верификация каждой дополнительной версии пропорционально увеличивает затраты по этой статье расходов;

стоимость других видов деятельности (тестирование, организационные затраты) также увеличивается, но уже не линейно.

Эффективность внедрения разнообразия резервируемого ПО была проверена в 80-е гг. в США, где был проведен ряд экспериментов, финансировавшихся НАСА [2]. В результате 1 миллиона тестов 27 версий ПО было выявлено, что ПО, состоящее из неодинаковых резервируемых компонентов, отказывает с большей вероятностью, чем должно было бы быть в случае полной независимости модулей. Тем не менее результаты экспериментов показывают, что резервирование с применением принципа разнообразия повышает надежность всей системы, например, мажоритарной схемы «2 из 3» — на порядок по сравнению с надежностью единичной версии ПО.

Более оптимистичные результаты были получены при практической проверке эффективности концепции диверсифицированного ПО, реализованной в системе защиты бомбардировщиков, с помощью имитатора систематического ввода отказов процессора [16]. Результаты моделирования отказов технической части показали, что, по консервативным оценкам, надежность двухканального диверсифицированного ПО повышается в 1000 раз, хотя, конечно, результаты моделирования не отражают в полной мере работу реального процессора.

Накопленный опыт показывает, что полной независимости диверсифицированных систем можно достичь, только если одна система реализована на базе ПТС, а вторая — с использованием жесткой логики. Если обе управляющие системы компьютеризированы, то некоторая зависимость между диверсифицированными программными модулями неизбежна, хотя ее степень является предметом анализа [1].

Методы анализа ПО

Важными методами оценки ПО, используемыми в процессе верификации и валидации [5, 20, 21], являются статический анализ (оценка программы без ее запуска) и динамическое тестирование (с запуском).

В рамках статического анализа, проведенного на АЭС Temelin при независимой оценке ПО, критического для безопасности, были выполнены:

проверка всех входных и выходных точек, а также всех циклов с их входными и выходными точками;

контроль использования данных, т.е. переменных и формальных параметров;

проверка всей информации, от которой зависит каждое выходное значение [5].

Важным аспектом статического анализа является поиск таких скрытых потоков управления, когда один программный канал может негативно воздействовать на данные или потоки управления другого канала. Использование указателей в ряде языков программирования является самым известным из скрытых механизмов, когда некорректно вычисленный указатель на переменную или функцию может направить исполняемую процедуру по неверному адресу [20].

Как показал опыт статического анализа ПО, используемого на АЭС Sizewell-B для управления главной системой аварийной защиты, такая проверка требует больших трудозатрат. Поэтому необходимы средства для автоматизации анализа, а также для обработки, классификации и прослеживания дефектов, найденных в исследуемом ПО [20].

Динамическое тестирование заключается в работе программы с выбранными тестовыми данными и сравнение результата с предположительно правильным ответом. Так как число исходных комбинаций часто является астрономическим, то исчерпывающее тестирование обычно невозможно. Следовательно, продолжительность тестирования является в первую очередь вопросом экономическим. Считается, что зависимость между реально проверенной частью программы и числом найденных ошибок почти линейная. Поэтому выбранная стратегия тестирования должна обеспечивать максимально возможный объем проверки в условиях ограниченного бюджета и сроков. Если существует возможность адекватно оценить этот объем, то результаты тестирования можно использовать для прогнозирования числа оставшихся в программе ошибок [21].

Существуют различные стратегии выполнения тестов и отбора тестовых исходных данных [5, 21]:

тестирование, в ходе которого проверяются все требования технического задания;

стохастическое тестирование, когда характеристики тестов описываются вероятностными распределениями;

«охватывающее» тестирование, при котором исполняются определенные подмножества элементов в структуре программы, например проверяется, что каждый переход и оператор программы выполнялся по крайней мере один раз. Для проведения такого вида тестов необходимо иметь в программе счетчики на каждом условном операторе. Один из видов «охватывающего» тестирования заключается в подсчете числа различных пройденных программных маршрутов. Так как на практике проверить все маршруты невозможно, то тестовые данные для такого тестирования выбирают случайным образом, используя вероятностное распределение;

тестирование путем внесения изменений в программу для проверки отсутствия определенных классов отказов. Это скорее не тест программы, а оценка эффективности применяемой стратегии тестирования. Кроме того, такие проверки полезны для исследования отказоустойчивости программы.

Модели надежности ПО

Количественная оценка надежности ПО является нерешенной проблемой. Скептики считают [1, 17], что провести количественную оценку надежности ПО ПТС трудно или вообще невозможно, так как на тестах, проверяющих только часть возможных комбинаций, нельзя основывать оценку показателей надежности системы. Тем не менее первые статистические модели надежности ПО были предложены еще 30 лет назад [13], а к настоящему времени разработано более 100 моделей оценки надежности ПО, в частности при проведении ВАБ [5, 22]. Оптимисты полагают, что модели надежности помогают получить количественную информацию о доверии, с которым можно предполагать корректную работу программы.

Для оценки надежности нерезервированного ПО на основе собранных данных по отказам и успешным тестам был разработан ряд моделей. Большинство моделей можно разбить на две категории [2]:

модели повышения надежности;

стационарные модели надежности.

Оба класса моделей являются экстраполяцией процесса тестирования на условия реальной эксплуатации ПО. Модели могут быть адекватны, только если условия тестирования и работы сравнимы, ибо надежность ПО может достаточно сильно меняться в зависимости от пользователя или инсталляции [2, 13]. Так как ошибки в ПО могут приводить к отказам при определенных режимах его использования, то оценки значений интенсивности отказов корректны только для соответствующего профиля эксплуатации ПО и эти оценки сильно зависят от допущений, принятых при моделировании.

Модели повышения надежности (наиболее популярные) базируются на результатах тестирования версий ПО в процессе его отладки (с исправлением выявленных ошибок) вплоть до окончательной проверки программного продукта [2, 5, 7, 22]. После выявления отказа ПО обычно корректируют, и поэтому его надежность меняется. Выявленный тренд изменения (обычно повышения) надежности экстраполируют для предсказания текущего уровня надежности и его изменения в будущем (наработки до следующего отказа и времени, требуемого для выявления всех ошибок в ПО).

Модели повышения надежности основаны на двух основных предположениях: отказы фиксируются в хронологическом порядке; все ошибки, обнаруженные в ПО, исправляются. При использовании этих моделей обычно не учитывают тяжесть выявленного отказа ПО, т.е. не различают реальные отказы и небольшие дефекты. Эти модели достаточно субъективны, а определяемые тренды надежности зависят от изменений в процессе отладки (например, возможно изменение состава группы специалистов, участвующих в отладке ПО, или интегрирование в ПО новых функциональных возможностей). Момент выявления последнего отказа ПО также оказывает достаточно большое влияние на характер тренда.

В статье [22] проведено сравнение 10 популярных моделей повышения надежности ПО. Для сравнения были использованы результаты испытаний ПО, в которых выявлено 136 отказов. Среди прочих факторов проверялась способность модели прогнозировать число реально зарегистрированных отказов (последние 20% случаев) на основании данных по первым 80% отказов. Было выявлено, что результаты, получаемые с помощью различных моделей, достаточно противоречивы и сильно зависят от используемых данных по отказам.

Нет общепринятой точки зрения на то, к какому пределу стремится кривая повышения надежности: будет ли это уровень абсолютной надежности или всегда будет оставаться некий остаточный класс отказов, который вызывается дополнительными ошибками, вносимыми при исправлении ранее выявленных отказов ПО (рис. 2).

Стационарные модели надежности основаны на результатах тестирования версий ПО, подготовленных для практического применения. Результаты тестов обрабатывают теми же методами, что и при оценке надежности технических средств. Эти модели могут быть использованы для подтверждения требуемого уровня надежности. Они меньше зависят от допущений, но их применение часто лимитируют бюджетные ограничения, так как необходимо проведение достаточно продолжительных испытаний на надежность. Например, для подтверждения требуемого уровня надежности (99% верхняя доверительная граница вероятности отказа на требование — $1 \cdot 10^{-4}$) главной системы аварийной защиты реактора на АЭС Sizewell-B, реализованной на ПТС, такие испытания включали 46 000 успешных тестов [2].

Существуют также модели, основанные на искусственном введении в ПО дополнительных отказов перед тестированием (модели «посева» отказов), которые используют для оценки числа остающихся в ПО ошибок [5]. Предполагают, что дополнительные отказы могут быть обнаружены с той же вероятностью, что и отказы-аборигены. Метод применяется к высоконадеж-

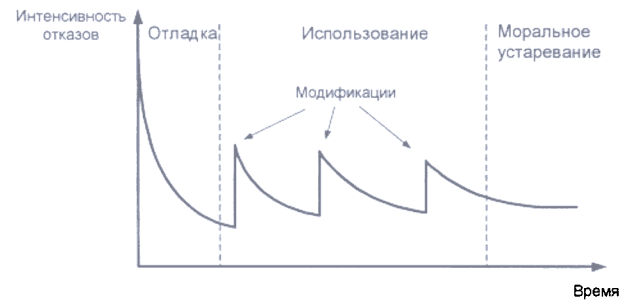

Рис. 2. Кривая изменения надежности ПО, имеющего остаточные отказы после проведения модификаций

ным программам, важным для безопасности, в которых не выявлено ни одного отказа при тестировании. Чтобы получить достаточно высокий уровень доверия, требуется внедрение в программу большого числа отказов. Эксперименты показали, что метод позволяет получить достаточно приемлемые оценки, когда более чем 50% внедренных в ПО ошибок обнаружены. Результатом использования моделей «посева» отказов является обоснование приемлемого тела ошибок в ПО, а не оценка показателя надежности. Поэтому они не могут быть непосредственно использованы при проведении ВАБ.

Для анализа диверсифицированного ПО разработаны вероятностные модели, учитывающие корреляцию между отказами разных версий ПО [2]. В основу моделей положено предположение, что разработчики ПО могут делать похожие ошибки в определенных обстоятельствах, причем чем сложнее задачи, решаемые независимыми группами разработчиков, тем больше вероятность зависимых отказов разных версий ПО. Модели базируются на рассмотрении двух стохастических процессов:

случайном выборе требований из пространства всех команд;

процессе разработки программы, который, как предполагается, также имеет вероятностную природу.

При моделировании второго процесса постулируется, что существует генеральная совокупность программ, которые могли бы быть написаны, и некое вероятностное распределение, характеризующее проблему, а акт написания программы — это ее случайный выбор из генеральной совокупности. Для случая «принудительного» разнообразия при разработке ПО вводится несколько вероятностных распределений, характеризующих программы, которые разработаны разными группами программистов.

Количественная оценка вклада отказов ПО в риск

Внедрение микропроцессорных и программируемых технологий приводит к концентрации риска на цифровых устройствах, потому что они могут быть предназначены для выполнения нескольких функций безопасности. При проведении ВАБ очень важно корректное моделирование многозадачности микропроцессорной техники. В статье [4] сформулированы проблемы, которые должны быть решены при проведении ВАБ для АЭС, оборудованной ПТС, —

определение видов отказов компьютеризированных систем;

оценка вероятности отказа ПО;

оценка эффективности разнообразия ПО и мероприятий по его верификации и валидации;

оценка отказоустойчивости;

моделирование отказов технических средств по общей причине;

моделирование взаимодействия ПО и аппаратуры;

моделирование воздействия окружающей среды на технические средства;

анализ влияния ошибок оператора.

В статье [23] описана методология моделирования и количественной оценки вероятности невыполнения функции ПО в процессе проведения ВАБ, разрабатываемая в США. Методология предусматривает шесть этапов выполнения анализа:

определение событий в модели ВАБ, вероятность реализации которых зависит от надежности ПО, т.е. выявление отказов элементов, управляемых с помощью ПО и моделируемых в ВАБ. Для всех таких событий/элементов проводится доработка логической модели ВАБ, чтобы учесть вклад в риск от отказов ПО;

определение функций ПО, которые должны быть выполнены в моделируемых аварийных сценариях (для каждого события, выявленного на предыдущем этапе);

разработка модели ПО, пример которой показан на рис. 3. Модель должна отображать функции ПО, зависящие от аварийного сценария, выполнение которых влияет на показатели риска, т.е.:

вывод данных, необходимых для достижения безопасного конечного состояния (следует отметить, что взаимодействие ПО с внешними устройствами также включается в модель);

своевременность этого вывода при возникновении быстропротекающих аварийных последовательностей;

детализация функций ПО для моделирования оказов коренных причин с помощью деревьев отказов;

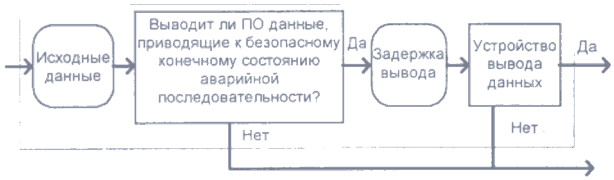

Рис. 3. Модель ПО, учитываемая в ВАБ

получение перечня минимальных сечений для построенных деревьев отказов;

количественная оценка необходимых параметров (функции распределения вероятности наступления нежелательного события и временных задержек) на основании результатов тестирования ПО с использованием генератора тестов.

Как показал анализ чувствительности результатов ВАБ к надежности ПО для унифицированной корейской АЭС [4], частота повреждения активной зоны реактора увеличивается в 2—3 раза при задании вероятности отказа ПО $1 \cdot 10^{-3}$ по сравнению со случаем абсолютной надежности ПО. Если варьировать и некоторые другие параметры, имеющие отношение к надежности ПО (вероятность ошибки оператора при устранении отказов ПО или отказа контрольного таймера), то частота повреждения активной зоны реактора может возрасти уже в 13 раз для самого консервативного случая. На этой АЭС система аварийной защиты реактора и система формирования аварийных сигналов на запуск систем безопасности построены на ПТС.

Нормативные требования к надежности ПО

Три документа, выпущенные в серии норм МАГАТЭ по безопасности [1, 17, 24], затрагивают проблемы надежности ПО.

Нормы по безопасности NS-G-1.1 [17] полностью посвящены проблемам, связанным с использованием ПО в системах, важных для безопасности АЭС. В стандарте указывается, что количественная оценка надежности программируемых систем является более сложной, чем непрограммируемых, и это может создать специфические трудности в обосновании ожидаемого уровня безопасности при использовании систем, построенных на ПТС. В документе сделан вывод, что в настоящее время (нормы выпущены МАГАТЭ в 2000 г.) невозможно доказать высокий уровень надежности ПО, т.к. существующие методы не обеспечивают получение результатов с уровнем доверия, который требуется для систем, наиболее важных для безопасности.

Рекомендуется, чтобы позиция регулирующего органа в отношении обоснования безопасности и требуемого уровня надежности ПО была бы сформулирована на ранней стадии при планировании проекта, а подход, которого следует придерживаться при решении проблем надежности ПО, должен быть определен, документирован, представлен надзорному органу и, если необходимо, согласован с ним.

Документ [1], выпущенный МАГАТЭ совместно с Агентством по атомной энергии Организации по сотрудничеству и развитию в Европе (OECD), разработан в качестве руководства надзорным органам по проведению экспертизы ВАБ АЭС, причем значительное внимание уделено анализу компьютеризированных систем. В нем также утверждается, что в настоящее время невозможно получить оценку интенсивности отказов программируемой системы на объективной и полностью защитимой основе. Поэтому показатели надежности, принятые для системы, базируются в конечном счете на экспертных оценках.

В документе [1] утверждается, что доказать отсутствие ошибок в ПО нереально. Поэтому предлагается делать акцент на обеспечении качества процесса разработки ПО и использовать процедуры, минимизирующие вероятность ошибок и максимально увеличивающие вероятность их выявления в процессе проверки программы (статический анализ) и тестирования всей системы (динамическое тестирование). Кроме того, отмечается важность разделения опасных ошибок (которые могут препятствовать выполнению функций безопасности) и ошибок, которые не оказывают воздействия на АЭС или изменяют ситуацию в безопасном направлении. При разработке рекомендуется обособлять части ПО, выполняющие наиболее критические функции безопасности и поэтому потенциально содержащие опасные ошибки. Это позволяет концентрировать проверку и тестирование ПО в наиболее важных областях.

При оценке вклада ПО в суммарную вероятность отказа системы рекомендовано принимать во внимание все факторы, которые могут повлиять на надежность ПО, —

размер и сложность ПО (число строк программы может быть использовано как индикатор);

новизну какой-либо характерной особенности ПО;

классификацию ПО по степени важности для безопасности;

степень соответствия процедурам и стандартам разработки, проверки и тестирования;

независимость групп специалистов, выполнявших статический анализ и динамическое тестирование;

число ошибок в ПО, выявленных во время этих двух процессов;

использование апробированных методик статического анализа ПО;

число проведенных динамических тестов ПО;

опыт разработчиков системы;

опыт работы подобных систем (который полезно учитывать, но маловероятно, что этот подход будет полезен для сложных систем).

Два нормативных документа МАГАТЭ устанавливают количественный критерий, относящийся к вероятности отказа на требование ПО. Хотя в обеих нормах фигурирует одно и то же значение вероятности — $1 \cdot 10^{-4}$, оно относится к несколько различным объектам. Документ [17] рекомендует относиться с осторожностью к проектам, в которых установлено, что вероятность отказа на требование ПО одноканальной компьютеризированной системы может быть меньше, чем этот критерий. В [1] констатируется, что и в больших интегрированных защитных системах вероятность отказа ПО составляет порядка $1 \cdot 10^{-4}$ на требование. Если заявляется, что вероятность намного меньше, чем это значение, то должна проводиться тщательная экспертиза всех аргументов в поддержку этой позиции.

Нормы МАГАТЭ по безопасности NS-G-2.3 [24] касаются проблемы модификации компьютеризованных систем. Так как отказы таких систем имеют скорее систематическую, чем вероятностную природу, то документ предписывает, чтобы в рамках процесса модификации проводился всесторонний анализ возможных отказов по общей причине резервируемых систем с идентичными версиями ПО. Как утверждается в [1], отказы по общей причине таких систем являются критической проблемой, и меры защиты против этого не так просто обеспечить.

На национальном уровне в ядерной энергетике существуют нормативные требования финского надзорного органа [25], которые ориентируются на публикацию Европейской комиссии по этой проблеме. Документ [25] предписывает необходимость снижения риска отказа ПО по общей причине, вызванного проектными ошибками, до достаточно низкого уровня путем использования принципа разнообразия и других возможных средств. Меры, предпринятые для предотвращения отказов по общей причине, нужно документировать, обосновывать и представлять в надзорный орган.

Документ [25] устанавливает требование к проекту компьютеризированных систем и оборудования 2-го класса безопасности, который должен быть максимально простым. При этом структура систем должна минимизировать последствия единичной ошибки в ПО и делать возможным верификацию требований, установленных для системы. Виды отказов ПО должны быть идентифицированы и проанализированы с достаточной степенью детализации.

На Украине разработан нормативный документ НП 306.5.02/3.035-2000, устанавливающий требования к управляющим системам АЭС, важным для безопасности, относящиеся как к характеристикам ПО, так и к процессу его разработки и согласованные с требованиями стандартов Международной электротехнической комиссии и МАГАТЭ [26].

Если суммировать международные и национальные нормативные документы по всем отраслям, связанным с повышенным риском, то количество действующих стандартов на разработку ПО, важного для безопасности, колеблется от 100 до 150 [7]. Среди них следует отметить стандарт IEC 61508:1998, выпущенный Международной электротехнической комиссией, который устанавливает требования к программируемым электронным системам, важным для безопасности, и рекомендует проводить лицензирование ПО на основе анализа риска. Лицензия может быть выдана, если интенсивность отказов ПО не превышает приемлемый уровень [27].

Список литературы

1. IAEA Review of Probabilistic Safety Assessments by Regulatory Bodies. — Safety Reports Series No. 25, Vienna, 2002.
2. Littlewood B., Popov P., Strigini L. Modeling Software Design Diversity. — A Review. In: ACM Computing Surveys, 2001, v. 33, No. 2, p. 177—208.
3. Mlady O. Plant Modifications at Temelin NPP: Status and Plans. — International Workshop on Safety of VVER-1000 Nuclear Power Plants. Piešťany, Slovakia, April 7—12, 2003.
4. Kang H.G., Jang S.-C., Ha J. The Risk Effect Analysis of the Digital Safety-Critical Systems in a Nuclear Power Plant. — In: Probabilistic Safety Assessment and Management PSAM7 — ESREL'04, June 14—18, 2004, Berlin, Germany, Springer, v. 2, p. 663—668.
5. IAEA Solutions for cost effective assessment of software based instrumentation and control systems in nuclear power plants. — IAEA-TECDOC-1328, December 2002.
6. Kharchenko V., Yastrebenetsky M., Sklyar V. Diversity Assessment of Nuclear Power Plants Instrumentation and Control Systems. — In: Probabilistic Safety Assessment and Management PSAM7 — ESREL'04, June 14—18, 2004, Berlin, Germany, Springer, v. 3, p. 1351—1356.
7. Carpignano A., Morisio M., Rambaudi E. Safety Assessment for Safety-Critical Systems: a Review and Commentary of the Available Techniques. — Ibid., v. 4, p. 2352—2357.
8. Госатомнадзор РФ Общие положения обеспечения безопасности атомных станций ОПБ-88/97 НП-001-97 (ПНАЭ Г-01-011-97), Москва, 1997.
9. Dehlinger J., Lutz R. Software Fault Tree Analysis for Product Lines. — 8th IEEE International Symposium on High Assurance Systems Engineering, Tampa, FL, 2004.
10. Schulze T., Gutgesell J. Report on the Comparison of Software Safety Concepts. — In: Probabilistic safety Assessment and Management PSAM7 — ESREL'04, June 14—18, 2004, Berlin, Germany, Springer, v. 4, p. 2107—2112.
11. Saglietti F., Jung M. Classification, Analysis and Detection of Interface Inconsistencies in Safety-Relevant Component-based Systems. — Ibid., v. 4, p. 1864.
12. Lee A., Smidts C., Li B., Li M. Validation of a Software-Related Failure Mode Taxonomy. — Ibid., v. 1, p. 152—157.
13. Elbel B., Maeckel O. Efficient reliability growth modelling for industrial software failure data. — Ibid., v. 2, p. 895—901.
14. Guarro S., Yau M., Oliva S Conditional Risk Model Concept for Critical Space Systems Software. — Ibid., v. 1, p. 158—163.
15. R. Lutz and I. Mikulski Empirical Analysis of Safety-Critical Anomalies During Operations. — IEEE Transactions on Software Engineering, v. 30, No. 3, March 2004.
16. Beckman H., Warholm L., Endresen J. Verification of Fault Detection Capability with Diversified Software. — In: Probabilistic Safety Assessment and Management PSAM7 — ESREL'04, June 14—18, 2004, Berlin, Germany, Springer, v. 2, p. 1108—1113.
17. IAEA Software for Computer Based Systems. Important to Safety in Nuclear Power Plants. — IAEA Safety Standards Series. Safety Guide. No. NS-G-1.1, 2000.
18. Littlewood B., Strigini L. Redundancy and diversity in security. — 9th European Symposium on Research in Computer Security ESORICS 2004, Sophia Antipolis, France, Springer-Verlag, Lecture Notes in Computer Science, September 2004.
19. Gashi I., Popov P., Stankovic V., Strigini L. On Designing Dependable Services with Diverse Off-The-Shelf SQL Servers. — In: Architecting Dependable Systems, Lecture Notes in Computer Science, 2004, v. 3069, p. 196—220, Springer-Verlag.
20. Bishop P.G., Bloomfield R.E., Clement T.P., Guerra A.S.L. and Jones C.C.M. Integrity Static Analysis of COTS/SOUP. — In: Proceedings of SAFECOMP 2003, 23—26 September, 2003, p. 63—76, Elsevier, Edinburg, Scotland.
21. Bishop P.G. MC/DC based estimation and detection of residual faults in PLC logic networks. — In: ISSRE2003 Fast Abstracts, 17—20 November, 2003, p. 297—298, Chillarege Press, Denver, USA.
22. Hu Y., Zhang W., Li B. Benchmarking Software Reliability Growth Models. — In: Probabilistic safety Assessment and Management PSAM7 — ESREL'04, June 14—18, 2004, Berlin, Germany, Springer, v. 2, p. 908—913.
23. Li B., Li M., Smidts C. Integrating Software into PRA: A Test-Based Approach. — Ibid., v. 1, p. 140—145.
24. МАГАТЭ. Модификации на атомных станциях. — Серия норм МАГАТЭ по безопасности. Руководства. No. NS-G-2.3, 2004.
25. Radiation and nuclear safety authority (STUK) Guide YVL 5.5 13 «Instrumentation systems and components at nuclear facilities», September 2002.
26. Kharchenko V., Yastrebenetsky M., Sklyar V. The Technique and the Experience of Expertise of Software for NPP Instrumentation and Control Systems. — In: «Probabilistic safety Assessment and Management PSAM7 — ESREL'04, June 14—18, 2004, Berlin, Germany», Springer, v. 4, p. 2096—2101.
27. Ehrenberger W. Licensing of Software for Safety-critical Applications on the Basis of Operating Experience. — Ibid., p. 2358—2363.

Ключевые слова

Атомные станции, управляющие системы безопасности, программируемые технические средства, программное обеспечение, отказы по общей причине, надежность.