протектором в отношении синхронных морфофункциональных изменений лимфатических узлов разной локализации в ходе регидратации организма.

Технические науки

ИЗОЛЯЦИЯ МОДУЛЕЙ МУЛЬТИВЕРСИОННОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ НА ЭТАПАХ РАЗРАБОТКИ

Морозов В.А.
Сибирский Государственный Аэрокосмический Университет
Красноярск, Россия

Среди методов обеспечения повышенной надёжности программного обеспечения весьма удачно зарекомендовала себя концепция мультиверсий [1-4]. Мультиверсионный подход предполагает независимую разработку двух и более функционально эквивалентных программ (версий). Затем версии исполняются параллельно под управлением среды исполнения [1]. При исполнении версий средой исполнения принимается решение о том, какие версии «сработали» корректно, а какие нет. Во многих случаях в качестве «правильных» результатов выбираются результаты, которые возвратило большинство версий [3,4]. При этом данные версий, которые выполнены «не корректно» заменяются данными версий признанных «корректными». Очевидно, что ситуация когда большинство версий вернули одинаковые ошибочные результаты может привести к отказу всей программной системы. Ситуации, в которых несколько версий возвращают одинаковые ошибочные результаты, называются «идентичными ошибками» (identical-and-wrong answer) [3]. Для того, чтобы избежать идентичные ошибки, версии стараются сделать максимально изолированными. Изоляция версий необходима для того, чтобы версии были «несхожими» и тем самым вероятность идентичных ошибок была минимальна. Поэтому в контексте данной работы под термином «изоляция» понимается «изоляция с целью обеспечения несхожести». Предложено ряд мер по достижению изолированности версий, в том числе использованию различных языков программирования и алгоритмов работы [1-4]. Целью данной статьи является обобщение этих методов. Предложена методика применения методов на основных стадиях разработки мультиверсионного программного обеспечения.

Основными стадиями разработки мультиверсионного ПО являются следующие: определение требований к программной системе, спецификация NVX, кодирование.

Стадия 1. Определения требований к программной системе

На этой стадии должны быть определены следующие требования по обеспечению изолированности.

Во-первых, следует решить будет ли использоваться *случайная изоляция* или *принудительная*. При случайной изоляции каждый разработчик версии сам выбирает средства разработки программы-версии, не зная какие средства выбрали остальные разработчики. Принудительная изолированность предполагает явное указание того, какие разработчики какие средства должны использовать (при этом для всех разработчиков должны быть определены различные средства).

Во-вторых, нужно определить качественные меры отличий в разработке версий: языки программирования, алгоритмы, программные средства разработки и другие.

В-третьих, следует учесть экономические затраты на обеспечение изолированности версий (добавление каждой меры изолированности требует дополнительных экономических затрат).

После определения моментов, описанных выше, можно определить финальные требования по изолированности версий ко всему мультиверсионному проекту. Типичные требования включают стоимостные ограничения и перечень мер достижения изолированности и отличий.

Стадия 2. Стадия спецификации среды исполнения (NVX).

Среда исполнения играет ключевую роль в работе всей мультиверсионной системы. С точки зрения обеспечения изолированности версий к NVX должны предъявляться следующие требования

1. Ограничение совпадающих свойств версий. Для централизованного управления версиями необходимо чтобы версии имели некоторые точки соприкосновения: контрольные точки (cross-check points), точки восстановления (recovery points) и др. Такая необходимость приводит к тому, что программы-версии имеют «схожие» участки. Данный эффект должен быть минимизирован на этапе определения механизмов взаимодействия с версиями.

2. Ограничение факторов, которые сокращают изолированность. Поскольку версии должны взаимодействовать со средой исполнения, к разрабатываемым версиям предъявляются некоторые требования (фиксированная структура предоставляемых данных, реакция на управляющие команды и др.). При спецификации этих требований необходимо уделить внимание вопросу «что должна уметь версия» и предъявлять как можно меньше ограничений касательно вопроса «как». Это позволит использовать разнообразные приёмы программирования, что снизит вероятность *идентичных ошибок*.

Стадия 3. Стадия кодирования

Данная фаза предполагает непосредственную реализацию программного кода. На этой фазе так же необходимо обеспечить, чтобы версии были максимально изолированными. Добиться такого эффекта можно с помощью реализации следующих мер:

1. Определение обязательных правил изоляции. Целью данных правил является обеспечение независимости разработки программ-версий. В общем случае такие правила устанавливают: строгий запрет обсуждения технических вопросов между командами программистов (с небольшими исключениями), широко разнесенные рабочие места (офисы, компьютерные терминалы и др.), обязательное использование различных машин для разработки версий (не допускается разработка различных версий на одном компьютере), ограничение доступа к технической документации.

2. Определение строгого C&D-протокола (communication and documentation protocol). C&D-протокол должен исключать возможность неконтролируемого взаимодействия между командами занятыми разработкой различных версий. Программисты, реализующие код одной версии должны предоставлять другим командам только минимум необходимой информации.

3. Формирование координационной команды. Основными функциями координационной команды являются: разработка спецификаций и данных для тестирования, реализация C&D-протокола, информирование команд программистов по вопросам разработки системы, синхронизация разработки версий и сбор технической документации.

СПИСОК ЛИТЕРАТУРЫ:

1. Algirdas A. Avizienis. The methodology of N-version programming. Software fault tolerance. 1995;

2. Alexander Romanovsky. Abstract object state and version recovery in N-version programming. Computing Science. No CS-TR-669, 1999;

3. Mladen A. Vouk. An empirical evaluation of consensus voting and consensus recovery block reliability in the presence of failure correlation. Материалы NASA Grant No NAG-1-983.

4. Mladen A. Vouk, David F. McAllister. Software reliability through fault-avoidance and fault-tolerance. Computer science, technical report #4. 1991.

КОМБИНИРОВАННАЯ АРХИТЕКТУРА ОТКАЗОУСТОЙЧИВОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Царев Р.Ю., Царев М.Ю., Волков В.А.
Красноярский государственный технический университет
Красноярск, Россия

Отказоустойчивая архитектура программного обеспечения гарантирует устойчивость к сбоям и логическим ошибкам, сгенерированным на этапе программирования. Выделяют два основных подхода к обеспечению отказоустойчивости и повышению надежности программного обеспечения: мультиверсионное программирование и блоки восстановления.

Оба эти подхода заимствованы из моделей аппаратных средств, где вследствие избыточности неисправные компоненты можно временно исключить из системной конфигурации. Однако программная избыточность принципиально отличается от аппаратной. Проектирование и реализация избыточных программных модулей предполагает их отдельную и независимую друг от друга разработку, так как простое дублирование одних и тех же модулей при проектировании является эффективным лишь против случайных ошибок, имеющих физическую природу, но не эффективно при отказах программного обеспечения. Простое копирование модулей – это также и копирование необнаруженных на этапе тестирования ошибок [1].

Таким образом, введение программной избыточности предполагает реализацию различных версий модулей. Эти версии предназначены для решения одной и той же задачи, но используют разные алгоритмы и, как правило, реализуются различными группами разработчиков.

Существенным отличием между мультиверсионным программным обеспечением и программным обеспечением с блоком восстановления является то, что в первом все версии выполняются одновременно, параллельно, а полученные результаты оцениваются, и корректный результат принимается посредством алгоритма голосования. В программном обеспечении с блоком восстановления выполняется только одна версия, и оценка ее результата происходит согласно приемочному тесту. Если результат принимается неверным, тогда происходит откат и выполняется следующая версия [2].

Алгоритмы голосования в мультиверсионном программном обеспечении не всегда способны определить корректный результат. Например, голосование абсолютным большинством требует, чтобы строго больше половины версий вернули идентичные результаты. Голосование согласованным большинством не может выдать однозначный результат, если число версий, вернувших один результат, равно числу версий, выдавших другой результат.