розв’язуються по забезпеченню безпеки за одиницю часу проведення контролю і спостереження за ЯРМ.

Згідно запропонованому критерію, рентабельність буде максимальною при низькій собівартості виробництва інформаційної продукції. Мінімізуючи час виконання кожного вирішуваного завдання по забезпеченню безпеки ЯРМ отримуємо збільшення продуктивності по першому показнику. Досягнення цього дозволяє збільшити об'єм завдань, які вирішуються що згідно другому показнику, також приводить до збільшення продуктивності САКС.

1. *Лисиченко Г.В., Забулонов Ю.Л., Буртняк В.М.* Распределенная интегрированная система контроля и слежения за ядерно-радиационными материалами, радиационными отходами и источниками ионизирующего излучения на объектах ядерно-топливного цикла. *Наука та інновації. 2009. Т. 5. № 5. С. 57- 61.*
2. *Погребинский С.Б., Стрельников В.П.* Проектирование и надежность микропроцессорных ЭВМ. – М.: Радио и связь, 1088, - 166 с.
3. *Брюхович Е.И.* Стратегия разработки вычислительных средств и сетей с позиции экономических критериев их владельцев. – УсиМ, 1989, №3, с. 3 – 11

Поступила 27.01.2011р.

УДК 004.415.53

О.Ю. Малинина, НТУУ «КПИ», Киев

АЛГОРИТМ ФОРМИРОВАНИЯ И ОТБОРА ТЕСТОВ ДЛЯ РЕГРЕССИОННОГО ТЕСТИРОВАНИЯ БАЗ ДАННЫХ

Automation of regression testing is facing several problems such as the difficulty of tests determining and the lack of cheap and effective tools for automation. But it is necessary to create new approaches to regression testing and invent new tools that could accelerate it. Article is devoted to the development of algorithm that is aimed to simplify this process for systems which contain database (DB) stored procedures.

Тестирование представляет собой один из способов обеспечения качества программного обеспечения (ПО). Оно входит в жизненный цикл ПО, являясь одной из основных фаз разработки, и характеризуется достаточно большим вкладом в суммарную трудоемкость разработки продукта [1]. Регрессионное тестирование – это процесс проверки того, что существующая функциональность не пострадала от добавления новой функциональности или исправления дефектов [2]. Поскольку, как это видно из определения, тесты регрессионного тестирования должны выполняться

многократно на протяжении всего цикла разработки программного обеспечения, то автоматизация данного вида тестирования может существенно сократить трудоемкость и ресурсоемкость всего процесса тестирования. На практике автоматизация регрессионного тестирования сталкивается проблемами сложности определения тестового набора и отсутствия дешевых и эффективных инструментов для автоматизации. Поэтому разработка новых подходов к организации регрессионного тестирования является актуальной научно-технической проблемой.

Целью данной работы является разработка алгоритма отбора и упорядочения тестов для регрессионного тестирования баз данных (БД).

Различают две стратегии регрессионного тестирования: повторного прогона всех тестов и выборочного тестирования. В первом случае выполняется прогон всех регрессионных тестов, независимо от внесенных в ПО изменений. Основными преимуществами данного метода являются простота реализации и высокая вероятность обнаружения ошибки при большом тестовом покрытии кода [3]. В то же время этот метод требует много времени и ресурсов, поэтому на практике оказывается неэффективным. Альтернативная стратегия заключается в выборе из старого набора только тех тестов, которые являются необходимыми для тестирования измененной программы. Эти тесты, в случае необходимости, могут дополняться новыми, для того, чтобы удовлетворить выбранный критерий покрытия кода. Данная стратегия хороша тем, что позволяет сократить расходы и время тестирования за счет исключения лишних тестов. Однако, при ее внедрении, требуются дополнительные затраты на создание метода отбора нужных тестов, также возможны пропуски ошибок вследствие исключения нужных тестов.

При выборочном регрессионном тестировании различают следующие методы:

- случайные, при которых тестовый набор определяется случайным образом либо же зависит от предыдущего опыта инженера по качеству ПО;
- безопасные, т.е. такие, что при некоторых четко определенных условиях не исключают тестов (из доступного набора тестов), которые обнаружили бы ошибки в измененной программе, то есть обеспечивает выбор всех тестов, обнаруживающих изменения;
- минимизации набора тестов, целью которых является отбор минимального (в терминах количества тестов) подмножества тестов, необходимого для покрытия каждого элемента программы, зависящего от изменений;
- покрытия кода, которые гарантируют сохранение выбранным набором тестов требуемой степени покрытия элементов измененной программы относительно некоторого критерия структурного покрытия C, использовавшегося при создании первоначального набора тестов.

Можно отметить, что алгоритмы определения набора тестов, существующие в настоящее время, не совершенны. Они не всегда математически обоснованы и часто носят эвристический характер, так как поставлены в

зависимость от конкретного проекта [4].

Анализируя существующие подходы к организации регрессионного тестирования можно отметить, что в большинстве случаев оно проводится на модуле, завершенном с точки зрения пользователя. Инженер по качеству ПО или программа автоматизированного тестирования воспроизводят действия пользователя, что увеличивает вероятность нахождения ошибок, которые могут быть им найдены. Однако, если посмотреть на процесс разработки, который представлен на рисунке 1, то можно отметить, что в некоторых случаях такой подход сопровождается значительными потерями времени разработчиков.

При наилучшем развитии ситуации затраты времени на одну итерацию разработки и тестирования будут следующими: $t_1 + t_2 + t_3 + t_5(t_6) + t_7$, при наихудшем – $t_1 + t_2 + t_3 + t_4 + t_5 + t_6 + t_7$.

$(t_4 + t_6 + t_2)$ - потери времени, которые могут быть незначительны при разработке небольших проектов. При увеличении сложности проекта и количества итераций разработки они будут равны $(t_4 + t_6 + t_2) * n$, где n - количество итераций.

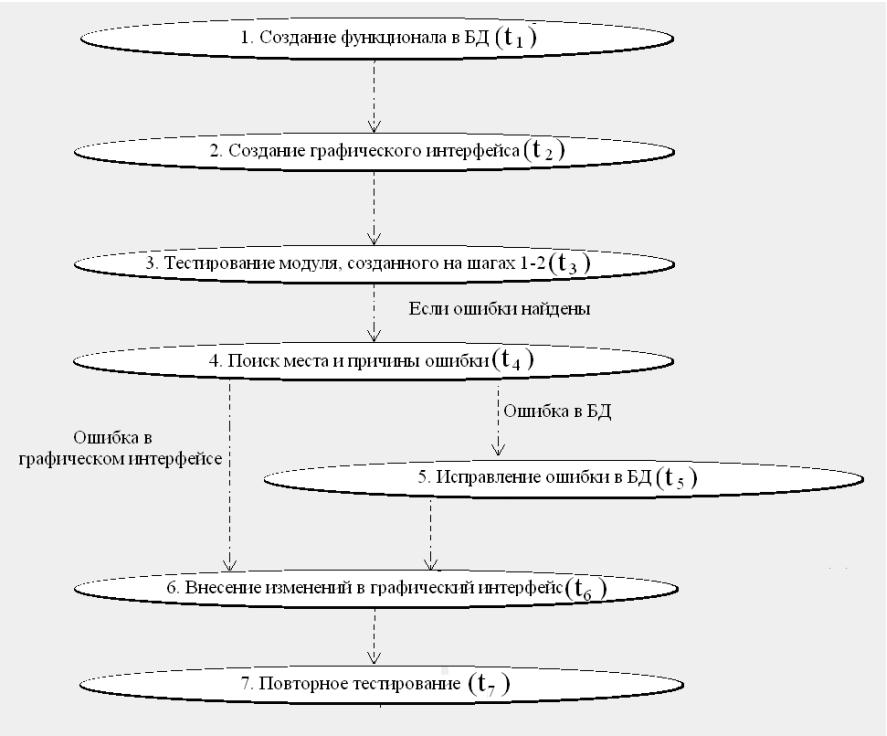

Рис. 1. Алгоритм разработки ПО, использующего БД

Регрессионное тестирование также связано с проблемой определения тестового набора для каждой итерации тестирования, т.е. такого множества $T'_{\text{реальное}}$, которое должно включать все тесты из T, активирующие измененный код, и не включать никаких других тестов. Но даже наличие информации о том, какой участок кода был изменен, не всегда позволяет определить все взаимосвязи. Тестирование встроенных процедур БД упрощает решение данной проблемы, так как при создании тестовых случаев в БД видна взаимосвязь между ними. Если определенный результат достигается последовательным вызовом нескольких процедур, то можно предположить, что изменения одной из них затронут именно эту последовательность.

Суть предлагаемого алгоритма состоит в том, что хранимой процедуре *I* ставится в соответствие множество тестов $T_i = \{t_{i,j}\}$, используемых при первичной разработке программы P, где i – номер тестируемой процедуры, j – номер теста, $j = 1...m$, m – количество тестов, разработанных для процедуры *I*. Тогда $T = T_1 \cup T_2 \cup ... \cup T_n$, где n – количество тестируемых процедур – множество всех тестов используемых при первичной разработке программы P, а $T' \subseteq T$ – подмножество регрессионных тестов для тестирования новой версии программы P'. При этом существуют или могут существовать такие $t_{k,l} \subset T_i \cap T_j$.

Рассмотрим процедуры *Q* и *Z*, для которых условием корректного выполнения процедуры *Z* является предварительное корректное выполнение процедуры *Q*. Для каждой из процедур существуют простые тесты $t_{q,l}$ и $t_{z,n}$, то есть такие, что содержат лишь вызов тестируемой процедуры. Однако, исходя из условия зависимости между процедурами, некоторые тесты для процедуры *Z,* могут быть представлены в виде последовательности тестов $t_{q,l}$ и $t_{z,n}$, которая не обладает свойством коммутативности. Такие составные тесты будут входить в подмножество тестов как процедуры *Q,* так и процедуры *Z*. Также следует отметить, что возможно существование некоторой процедуры *W*, все тесты которой будут составными.

Задача отбора тестов из набора T для заданной программы P и измененной версии этой программы P' состоит в выборе подмножества $T' \subseteq T$ для повторного запуска на измененной программе P', где $T' = \{t \in T \mid P'(t) \neq P(t)\}$. Однако, соотнесение множества тестов $T_i = \{t_{i,j}\}$ с процедурой *I* сводит поставленную задачу к отбору тестов, поставленных в соответствие измененной процедуре, т.е. $T' = \{t \in T \mid P'(t) \neq P(t)\} = \{t \in T \mid i'(t) \neq i(t)\} = T_i$.

После определения множества T', возникает задача его упорядочения. Данная задача не является существенной при небольшой размерности T', однако с увеличением данного множества, ее значимость возрастает.

Как было отмечено выше, могут существовать составные тесты $t_{k,l} \subset T_i \cap T_j$. При этом возможны две ситуации:

- тест, относящийся к тестируемой процедуре *I*, будет завершающим в последовательности;
- тест, относящийся к тестируемой процедуре *I*, не будет завершающим в последовательности.

Рассмотрим ситуацию, когда тест, относящийся к тестируемой процедуре *I*, будет завершающим в последовательности $t_{k,l} \subset T_i \cap T_j$. В данном случае $t_{k,l}$ является непосредственно тестом процедуры *I*, так как выполнение всех остальных тестов из $t_{k,l}$ – необходимое условие выполнения теста для процедуры *I*. Данные, а также все простые тесты процедуры *I,* будут относиться к подмножеству T_i^1.

В ситуации, когда тест, относящийся к тестируемой процедуре *I*, не будет завершающим в последовательности $t_{k,l} \subset T_i \cap T_j$, можно сказать, что тест $t_{k,l}$ является тестом взаимодействия процедуры *I* с остальными процедурами, так как от результата выполнения процедуры *I* зависит выполнение других процедур. Такие тесты относятся к подмножеству T_i^2.

Первыми выполняются те тесты, которые проверяют непосредственно процедуру, т.е. тесты из подмножества T_i^1. Если работоспособность процедуры подтверждена (все тесты из подмножества T_i^1 выполнены), то можно переходить к тестированию взаимодействия данной процедуры с другими, т.е. выполнять тесты из подмножества T_i^2. Если работоспособность процедуры не подтверждена, то ее следует отправить на доработку.

Информация о местонахождении ошибки и причинах ее возникновения, может значительно ускорить процесс исправления. Поэтому интересна следующая ситуация: все тесты из подмножества T_i^1 пройдены, но не пройден тест (или тесты) из подмножества T_i^2, который тестирует взаимодействие процедуры *I* с процедурой *J*. Такое возможно в двух ситуациях:

- не корректно функционирует процедура *J*;
- процедуры *I* и *J* корректно функционируют по отдельности, однако нарушены связи между ними.

Чтобы получить дополнительную информацию и более точно определить, какая из двух ситуаций имеет место, необходимо провести дополнительное тестирование – выполнить тесты из подмножества T_j^1, либо, если T_j^1 состоит из одного, уже не пройденного теста, выполнить тесты из подмножества T_j^2. Если все остальные тесты из подмножества T_j^1 или тесты из

подмножества T_j^2 были пройдены, то это означает, что в целом процедура J функционирует корректно, и проблема заключается во взаимодействии двух процедур. Если же тесты не были пройдены, это означает, что процедура J функционирует не корректно.

Алгоритм регрессионного тестирования процедуры I представлен на рисунке 2. Предполагается, что определение множества тестов $T_i = \{t_{i,j}\}$ и разбиение их на подмножества T_i^1 и T_i^2 проводится при первичном тестировании.

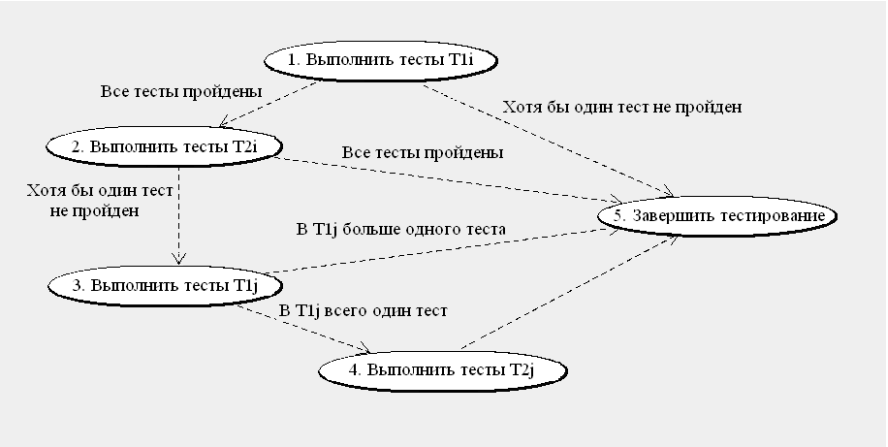

Рис. 2. Алгоритм регрессионного тестирования хранимой процедуры БД

По результатам работы можно сформулировать следующие выводы.

- Основной проблемой регрессионного тестирования является выбор тестов для проведения тестирования, так как общее количество регрессионных тестов возрастает экспоненциально с увеличением функциональной сложности разрабатываемого ПО.
- Автоматизация регрессионного тестирования является частичным решением проблемы, при этом затраты на автоматизацию не должны превышать затраты на проведение самого тестирования.
- Методы, которые обеспечивают максимальную вероятность нахождения всех регрессионных ошибок, являются чрезвычайно дорогостоящими, а более дешевые методы, не позволяют обеспечить должное качество программного обеспечения.

В работе предложен алгоритм, позволяющий определить:

- какие тесты должны быть выполнены в ходе регрессионного тестирования;
- порядок выполнения выбранных тестов;

- в некоторых случаях – дополнительную информацию о местонахождении ошибки, а также возможную причину ее возникновения.

Применение описанного алгоритма позволяет сократить время регрессионного тестирования без потери качества тестирования. Практическая ценность работы состоит в том, что использование предлагаемого алгоритма позволяет получить новый инструмент, который может быть применен для тестирования программных продуктов различного уровня сложности и в различных предметных областях.

1. *Дідковська М.В., Тимошенко Ю.О.* Тестування: Основні визначення, аксіоми та принципи, 2010, 58 с. [Электронный ресурс]. – Режим доступа: CD-ROM.
2. *Котляров В.П.* Основы тестирования программного обеспечения / *Котляров В.П., Коликова Т.В.* – М.: БИНОМ, 2006. – 286 с.
3. *Boehm B.* Software Engineering Economic – N.J. Prentice-Hall, Inc, 1981. – 767 pp.
4. *Майерс Г.* Искусство тестирования программ / Майерс. Г., пер с англ. под ред. Б.А. Позина – М.: Финансы и статистика, 1982. – 172 с.

Поступила 3.03.2011р.

УДК 004.056.55:004.272.23:004.274

С.Я. Гильгурт, канд.техн.наук, ИПМЭ им. Г.Е.Пухова НАНУ, г. Киев
А.К. Гиранова, ИПМЭ им. Г.Е.Пухова НАНУ, г. Киев

РЕКОНФИГУРИРУЕМЫЙ ПРОЦЕССОР, РЕАЛИЗУЮЩИЙ УСИЛЕННЫЕ АЛГОРИТМЫ ЗАКРЫТИЯ ИНФОРМАЦИИ

*Аннотация***.** В статье предложена обобщенная структура реконфигурируемого процессора, реализующего усиленные алгоритмы блочного симметричного шифрования. Разработаны структурные схемы для различных методов усиления.

Библиогр.**:** 6 наим.

Ключевые слова: закрытие информации, блочное симметричное шифрование, реконфигурируемый вычислитель.

Стремительный рост производительности вычислительной техники помимо положительных сторон приводит в качестве побочного эффекта к ослаблению защиты информации, в частности, к снижению стойкости криптографических средств. Алгоритмы шифрования, которые еще несколько лет назад считались безопасными, в настоящее время уже не являются таковыми [1]. С другой стороны, создание совершенно новых алгоритмов закрытия информации представляется весьма непростой задачей. Этим обусловлена актуальность направления, связанного с развитием