СБОРНИК **НАУЧНЫХ ТРУДОВ НГТУ. – 2014. – № 3(77).** – 125–136

СООБЩЕНИЯ

УДК 62-50:519.216

ОБЗОР РАБОТ ЖУРНАЛА JOURNAL OF SYSTEMS AND SOFTWARE ЗА 2012–2014 ГОДЫ, ПОСВЯЩЕННЫХ АНАЛИЗУ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ*

А.В. МАРКОВ[1], Д.О. РОМАННИКОВ[2]

[1.] *630073, РФ, г. Новосибирск, пр. Карла Маркса, 20, Новосибирский государственный технический университет, аспирант кафедры автоматики. E-mail: muviton3@gmail.com*
[2.] *630073, РФ, г. Новосибирск, пр. Карла Маркса, 20, Новосибирский государственный технический университет, кандидат технических наук, старший преподаватель кафедры автоматики. E-mail: rom2006@gmail.com*

В данной работе приводится обзор наиболее интересных, с точки зрения авторов работ, опубликованных в журнале Journal of Systems and Software за 2012–2014 годы. Для анализа были выбраны работы по тестированию и разработке программного обеспечения, которые условно можно разделить на несколько групп: 1) работы по локализации ошибок; 2) работы по генерации тестов; 3) работы по построению моделей программ и дальнейшего их численного анализа. В результате выполненного обзора работ можно сделать вывод: методы локализации ошибок основаны в основном на данных, полученных от результатов выполнения тестов и математической модели, построенной методом «среза пучка» или аналогами, позволяют достаточно точно определить содержащий ошибку оператор. В основе методов генерации тестов лежат различные модели, охватывающие программу полностью или частично, на основе которых составляется список тестов и тестовых данных. Также в данной статье рассмотрен ряд работ, посвященных статистическому анализу исходного кода программного обеспечения. Причем в нескольких работах в качестве эталона принимается рейтинг ответов на таких ресурсах, как StackOverflow.com, в других – анализ журналов с описанием ошибок. Также проанализирован ряд работ, в которых строится модель программы с целью получения каких-либо численных характеристик.

Ключевые слова: программное обеспечение, тестирование, формальная верификация, динамическая верификация, верификация, проверка моделей, модели программного обеспечения, графы, тотальная корректность программ, генерация тестов, локализации ошибок

---

* Статья получена 10 июня 2014 г.

Работа выполнена при финансовой поддержке Минобрнауки России по государственному заданию № 2014/138. Тема проекта «Новые структуры, модели и алгоритмы для прорывных методов управления техническими системами на основе наукоемких результатов интеллектуальной деятельности».

ВВЕДЕНИЕ

На сегодняшний день в индустрии разработки программного обеспечения (ПО) сложилась ситуация, при которой ПО достигло громадных размеров. Среднее программное обеспечение достигает порядка одного миллиона строк. Такое количество информации сложно прочесть, а «держать в уме» и осознавать, каким образом изменения в коде могут повлиять на всю систему в целом, практически невозможно. Для решения вышеприведенной проблемы в индустрии было придумано множество техник, приемов и инструментов, которые помогают в контроле качества разрабатываемого ПО. Но данные инструменты и методы не позволяют решать задачу полноценного анализа ПО с точки зрения доказательства его верной работы и/или поиска ошибок во всем диапазоне значений и вариантов использования.

В разделе СВЯЗАННЫЕ РАБОТЫ приводится описание существующих наработок в сфере анализа программного обеспечения. Обзор наиболее интересных подходов и идей по упрощению анализа программного обеспечения приведен в разделе ОБЗОР. Статья заканчивается разделом ВЫВОДЫ, в котором приведены основные результаты данной работы.

1. СВЯЗАННЫЕ РАБОТЫ

Создание инструментов по анализу ПО ведется по нескольким направлениям: проверка моделей[1], основные достижения в которой связаны с именами ученых Пеледа и Кларка; статический анализ [6]; динамическая верификация, а также смешанные техники и подходы.

Наиболее интересным и перспективным направлением, с точки зрения авторов, представляется проверка моделей. Статический анализ, который по сути является проверкой кода на некоторые известные шаблоны, сильно ограничен и не гарантирует точности выдаваемых результатов (точнее, выдает заранее неточные результаты, а «хороший» статический анализатор отличается от «плохого» процентом ложных срабатываний результатов). Динамический анализ в значительной степени ограничен трудоемкостью задачи – проверка одного из сценариев требует полного выполнения программы. При этом проблематично использование результатов предыдущих вычислений.

2. ОБЗОР РАБОТ ПО ЛОКАЛИЗАЦИИ ОШИБОК

В работе [3] предлагается способ локализации ошибок, которые найдены во время разработки и тестирования. Предлагаемый способ является статистическим подходом и основан на итеративном анализе результатов тестов и

[1] Кларк Э., Грамберг О., Пелед Д. Верификация моделей программ. Model Checking. – М.: МЦНМО, 2002. – С. 416.

оценке компонент программы (строчек кода) в тестах, которые прошли успешно и обнаружили ошибку в продукте. Данный подход проверен на нескольких промышленных программных продуктах, однако авторы не сообщают о влиянии ошибочных тестов (т. е. тестов, которые сообщили о наличии проблемы, но проблема находится не в продукте, а в самом тесте) на точность локализации ошибок.

В работе [13] рассматривается анализ программы по ее исходному коду при помощи метода *среза пучка* (*slice cluster*). В основе данного метода используется понятие *графа системы зависимости* (*system dependence graph*) [14, 15][2]. В данных работах способ формирования «срезов» отличается от существующих. Несмотря на то что в статье рассматривается связь между исследуемой зависимостью в коде и ошибками, данная работа более интересна используемой математикой, чем полученными результатами. На взгляд авторов, используемый математический аппарат (или его модификация: в частности, использование только дуг зависимости данных в графе системы зависимости) может быть применен не только для определения зависимости в коде, но и для поиска ошибок.

В работе [18] предполагается подход определения места ошибок в программе с помощью нового метода, основанного, в отличие от [16], не на покрытии исходного кода программы, а на исполнении среза программы. Причем в программе в качестве среза используется не полный срез, а предлагаемый гибридный спектральный срез (Hybrid spectrum slice).

Улучшенный алгоритм для тестирования программ предложен в [19]. По сравнению к общепризнанным алгоритмом покрытия измененных решающих операторов (Modified Condition Decision Coverage) предлагаемый подход minimal-MUMCUT позволяет выполнять меньшее количество тестов для полного покрытия операторов управления. Еще один метод локализации ошибок предложен в [25].

В [27] предложен метод локализации ошибок, в котором вычисляется коммит (внесение изменений в систему хранения версий программы) с ошибкой. Вычисление коммита выполняется при помощи последовательного перебора изменений согласно истории и выполнения набора тестов до тех пор, пока не будет найден нужный коммит. Очевидно, что данный метод может выполняться только для небольших проектов, потому что временные и вычислительные расходы на сборку проекта (с учетом того, в больших проектах количество коммитов может достигать сотен в день) слишком высоки.

---

[2] Под данным графом понимают граф, состоящий из вершин с операторами и двух типов дуг – зависимости данных и зависимости управления. Причем дуги зависимости данных соединяют переменную от места ее создания к каждому месту, где она используется. А дуги зависимости управления соединяют вершины графа с управляющими конструкциями [15].

3. ОБЗОР РАБОТ ПО ГЕНЕРАЦИИ ТЕСТОВ

Тестирование взаимодействия нескольких программных продуктов рассматривается в [7]. В данной работе новизной является предложенный подход, с помощью которого выполняется генерация тестовых случаев для проверки межпрограммных коммуникаций. Построение тестовых случаев основывается на ранее построенной модели, в основе которой лежат временные входные и выходные автоматы. Стоит заметить, что построенная модель абстрагирована от данных и оперирует не с программным кодом, а с более высокими абстракциями, выраженными в UML-диаграммах. Вопрос о том, насколько идея абстрагирования от данных и рассмотрения только переходов при анализе программных продуктов может быть использована, остается открытым. Возможно, что она может быть применена частично, при этом анализ может быть разбит на несколько частей: в первой выполняется упрощение дерева переходов на основе модели без данных, а во второй – более детальный анализ, основанный на полноценной модели.

В работе [10] приводится алгоритм по генерации тестов. При создании тестов имеет значение два аспекта: сценарии тестов и данные, на которых выполняются тесты. Функция, для которой генерируются тесты, выбирается разработчиком. Тестовые данные подбираются при помощи генетического алгоритма, в котором выполняется последовательное приближение тестовых данных на основе вычисления оценочной функции.

ATEMES, представленный и разработанный в [11], позволяет генерировать набор тестов для CppUnit фреймворка и тестовых данных. Тесты генерируются для функций, которые принимают не только простые типы данных, но и составные структуры. Также в данной работе приводится пример подсчета покрытия тестовых данных и производительности. Данный подход достаточно перспективен для использования на практике, так как он позволяет автоматизировать рутинную работу разработчиков и повысить контроль над разрабатываемым кодом, но, к сожалению, существенным недостатком данный работы является отсутствие четких критериев достаточности тестов, т. е. нет определения того, какой набор тестов необходим для генерации, чтобы гарантировать качество программного кода.

4. ОБЗОР РАБОТ О СТАТИСТИЧЕСКОМ АНАЛИЗЕ

В [12] приводится исследование по влиянию различных аспектов при написании ПО: язык программирования, парадигма написания ПО, сложность программы, использование ручной и автоматизированной проверки кода. В качестве используемой методики предлагается опрос разработчиков, которые участвовали в проектах по созданию систем планирования предприятий.

Одно из самых оригинальных исследований – работа [21], в которой приведено исследование зависимости ошибок в программе от частоты обновления комментариев. В данной работе исследуются несколько больших проектов (от одного миллиона строк кода) с длинной историей разработки и поддержки (до 30 лет). Основным выводом статьи является то, что если при изменении кода находящиеся рядом комментарии остаются неизмененными (и наоборот), то такое действие влечет повышенное количество программных ошибок.

Совершенно новый подход к обработке ошибок представлен в [26], где предлагается способ анализа ПО с помощью общей базы данных (с доступом через интернет), в которой собирается и анализируется информация об исходных кодах различных авторов. В работе рассматривается два способа анализа: 1) проверка исходного кода на безопасность на основе внутренних проверок; 2) проверка на безопасность и корректность с учетом мнения других разработчиков ресурсов вопросов и ответов (например, StackOverflow.com). В данной работе интересен сам подход: анализ ПО с учетом общей базы исходных кодов. Потенциально такой подход может дать феноменальные результаты. Используя общую базу исходных кодов, где системы (например, базы данных, операционная система, различные утилиты и библиотеки) представлены в виде спецификаций или результатов моделирования, можно анализировать поведение программы в различном комплексе систем с учетом всех (!) особенностей и различных вариантов их поведения.

5. ОБЗОР РАБОТ ОБ АНАЛИЗЕ МОДЕЛЕЙ ПРОГРАММ

В работе [8] выполняется анализ встраиваемого ПО. Предложенный в данной статье метод заключается в построении межпроцедурного графа потока управления и анализа списка переменных [9] согласно построенному графу. Однако в данной работе рассматриваются только переменные, вовлеченные в «межслоевое» и межпроцессорное взаимодействие.

В работе [24] приведен анализ с использованием среза с помощью графа зависимостей и дополнительно текстового анализа исходного кода. В итоге были получены модель и численное описание программы. В целом значительной выгоды из этой модели в статье получено не было.

ОБЗОР ДРУГИХ РАБОТ

Работы [1, 2] посвящены анализу аспектно-ориентированного программирования. В них предложены несколько предложений по разработке языка DSL, который должен быть языком высокой абстракции, с поддержкой множества контекстов, платформо-независимым, с сохранением истории. Важной особенностью предлагаемого языка является разделение контекстов на несколько групп: физические, компьютерные, персональные и другие. Таким

образом, различные аспекты программы можно описывать в различных контекстах, что приводит к значительному упрощению задачи для тех контекстов, для которых существует описание.

В работе [5] предлагается способ нахождения ошибок в интеграции программных приложений. Основная идея данного подхода состоит в том, чтобы встроиться в процесс передачи сообщений между приложениями и искать ошибочные ситуации, т. е. такие последовательности сообщений, которые считаются ошибочными для взаимодействия приложений. Основная идея данного подхода не нова. Например, она используется при статическом анализе программного кода.

Символьный фреймворк для статического анализа программного кода рассматривается в [6]. Если рассматривать процесс статического анализа программного кода как последовательность из вычисления символических выражений для всех валидных комбинаций переменных и нахождения ошибок по известным паттернам, основываясь на полученных символьных выражениях, то в данной статье приводится решение автоматизации первого шага.

Несомненной новизной обладает материал в [16], в котором представлен метод *автоматического* исправления ошибок в программе. Предлагаемый метод основан на двух идеях: 1) использовании техники определения оператора, который содержит ошибку[3]; 2) изменении найденного оператора и выполнении тестов. В [16] приводится статистика исправления ошибок (в среднем одно исправление на десять найденных ошибок). Необходимо отметить, что для использования данного подхода нужен большой набор тестов с полным покрытием исходного кода. Также есть большая вероятность того, что исправление ошибки в программе может не ограничиться исправлением только одного оператора, и в таком случае данный метод не может быть применен.

ВЫВОДЫ

Анализ работ журнала Journal of System and Software за 2012–2014 годы показал, что в индустрии разработки и анализа программного обеспечения сложилось несколько основных направлений развития.

1. Локализация ошибок в программе [3, 13, 18, 19, 25, 27, а также 16], но в последней присутствует особенность – мутация программы с целью автоматического исправления ошибок. Данные подходы основаны на статистических данных результатов выполнения тестов, методе среза в различные его вариантах.

2. Покрытие кода программы набором тестов на различных входных данных, которые генерируются автоматически [7, 10, 11].

[3] Конкретно в данной работе используется техника «Тарантул» (Tarantula), поиск оператора в которой осуществляется на основе выполнения набора тестов и вероятностного определения ошибочного оператора.

3. Статистический анализ ПО [12, 21, 26] – исследование количества/частоты программных ошибок в исходных кодах от различных аспектов. Причем среди аспектов встречаются как «стандартные» языки программирования и парадигмы написания ПО, так и специфические: обновление комментариев при изменении программного кода в окрестностях, количество набранных баллов на различных программных ресурсах (StackOverflow.com, например).

4. Построение моделей с целью получения количественных характеристик ПО. Полученные в [8, 24] модели были построены на основе графов зависимостей и управления. Модель в [4] получена на основе диаграммы последовательности и позволяет выполнять анализ протокола на наличие уязвимостей аналогично методу, применяемому для статического анализа. В [20] представлена модель на основе сетей Петри для оценки производительности ПО. Несмотря на то что при создании модели использовалось множество примеров, полноценной модели программы построено не было. В работе [17] предложен способ отладки программ при помощи запоминания отдельных событий в ходе исполнения программы (логгирования) и дальнейшего их ручного анализа.

Основными инструментами анализа программы являются методы среза программы: статический, динамический, обратный и прямой срезы, основанные на графе зависимости системы [9]. Необходимо отметить, что среди рассмотренных работ не было ни одной, где вышеназванные методы применялись к проверке моделей, и, возможно, стоит рассмотреть применение метода среза в этой сфере.

СПИСОК ЛИТЕРАТУРЫ

1. *Hoyos J.R., García-Molina J., Botía J.A.* A domain-specific language for context modeling in context-aware systems // Journal of systems and software. – 2013. – Vol. 86, iss. 11. – P. 2890–2905. – doi: 10.1016/j.jss.2013.07.008.

2. A design rule language for aspect-oriented programming / A.C. Neto, R. Bonifácio, M. Ribeiro, C.E. Pontual, P. Borba, F. Castor // Journal of systems and software. – 2013. – Vol. 86, iss. 9. – P. 2333–2356. – doi: 10.1016/j.jss.2013.03.104.

3. *Perez A., Abreu R., Riboira A.* A dynamic code coverage approach to maximize fault localization efficiency // Journal of systems and software. – 2014. – Vol. 90. – P. 18–28. – doi: 10.1016/j.jss.2013.12.036.

4. *Diaz J., Arroyo D., Rodriguez F.B.* A formal methodology for integral security design and verification of network protocols // Journal of systems and software. – 2014. – Vol. 89. – P. 87–98. – doi: 10.1016/j.jss.2013.09.020.

5. *Frantz R.Z., Corchuelo R., Molina-Jiménez C.* A proposal to detect errors in Enterprise Application Integration solutions // Journal of systems and software. – 2012. – Vol. 85, iss. 3. – P. 480–497. – doi: 10.1016/j.jss.2011.10.048.

6. *Burgstaller B., Scholz B., Blieberger J.* A symbolic analysis framework for static analysis of imperative programming languages // Journal of systems and software. – 2012. – Vol. 85, iss. 6. – P. 1418–1439. – doi: 10.1016/j.jss.2011.11.1039.

7. InRob: An approach for testing interoperability and robustness of real-time embedded software / F. Mattiello-Francisco, E. Martins, A.R. Cavalli, E.T. Yano // Journal of systems and software. – 2012. – Vol. 85, iss. 1. – P. 3–15. – doi: 10.1016/j.jss.2011.02.034.

8. An approach to testing commercial embedded systems / T. Yu, A. Sung, W. Srisaan, G. Rothermel // Journal of systems and software. – 2014. – Vol. 88. – P. 207–230. – doi: 10.1016/j.jss.2013.10.041.

9. *Sinha S., Harrold M.J., Rothermel G.* Interprocedural control dependence // ACM Transactions on software engineering and methodology. – 2001. – Vol. 10, iss. 2. – P. 209–254. – doi: 10.1145/367008.367022.

10. *Pachauri A., Srivastava G.* Automated test data generation for branch testing using genetic algorithm: an improved approach using branch ordering, memory and elitism // Journal of systems and software. – 2013. – Vol. 86, iss. 5. – P. 1191–1208. – doi: 10.1016/j.jss.2012.11.045.

11. Automatic testing environment for multi-core embedded software – ATEMES / C. Koong, C. Shih, P. Hsiung, H. Lai, C. Chang, W.C. Chu, N. Hsueh, C. Yang // Journal of systems and software. – 2012. – Vol. 85, iss. 1. – P. 43–60. – doi: 10.1016/j.jss.2011.08.030.

12. Coding-error based defects in enterprise resource planning software: Prevention, discovery, elimination and mitigation / I. Woungang, F.O. Akinladejo, D.W. White, M.S. Obaidat // Journal of systems and software. – 2012. – Vol. 85, iss. 7. – P. 1682–1698. – doi: 10.1016/j.jss.2012.02.034.

13. Coherent clusters in source code / S. Islam, J. Krinke, D. Binkley, M. Harman // Journal of systems and software. – 2014. – Vol. 88. – P. 1–24. – doi: 10.1016/j.jss.2013.07.040.

14. *Horwitz S., Reps T., Binkley D.* Interprocedural slicing using dependence graphs // ACM Transactions on Programming Languages and Systems (TOPLAS). – 1990. – Vol. 12, iss. 1. – P. 26–60. – doi: 10.1145/77606.77608.

15. *Ferrante J., Ottenstein K.J., Warren J.D.* The program dependence graph and its use in optimization // ACM Transactions on Programming Languages and Systems (TOPLAS). – 1987. – Vol. 9, iss. 3. – P. 319–349. – doi: 10.1145/24039.24041.

16. *Debroy V., Wong W.* Combining mutation and fault localization for automated program debugging // Journal of systems and software. – 2014. – Vol. 90. – P. 45–60. – doi: 10.1016/j.jss.2013.10.042.

17. *Kolomvatsos K., Valkanas G., Hadjiefthymiades S.* Debugging applications created by a domain specific language: the IPAC case // Journal of systems and software. – 2012. – Vol. 85, iss. 4. – P. 932–943. – doi: 10.1016/j.jss.2011.11.1009.

18. HSFal: Effective fault localization using hybrid spectrum of full slices and execution slices / X. Ju, S. Jiang, X. Chen, X. Wang, Y. Zhang, H. Cao // Journal of systems and software. – 2014. – Vol. 90. – P. 3–17. – doi: 10.1016/j.jss.2013.11.1109.

19. *Kaminski G., Ammann P., Offutt J.* Improving logic-based testing // Journal of systems and software. – 2013. – Vol. 86, iss. 8. – P. 2002–2012. – doi: 10.1016/j.jss.2012.08.024.

20. *Balsamo S., Harrison P.G., Marin A.* Methodological construction of product-form stochastic Petri nets for performance evaluation // Journal of systems and software. – 2012. – Vol. 85, iss. 7. – P. 1520–1539. – doi: 10.1016/j.jss.2011.11.1042.

21. On the relationship between comment update practices and software bugs / W.M. Ibrahim, N. Bettenburg, B. Adams, A.E. Hassan // Journal of systems and software. – 2012. – Vol. 85, iss. 10. – P. 2293–2304. – doi: 10.1016/j.jss.2011.09.019.

22. Petri net based techniques for constructing reliable service composition / G. Fan, H. Yu, L. Chen, D. Liu // Journal of systems and software. – 2013. – Vol. 86, iss. 4. – P. 1089–1106. – doi: 10.1016/j.jss.2012.11.037.

23. *Alama O., Adams B., Hassan A.E.* Controversy Corner: Preserving knowledge in software projects // Journal of systems and software. – 2012. – Vol. 85, iss. 10. – P. 2318–2330. – doi: 10.1016/j.jss.2012.03.028

24. Recovering test-to-code traceability using slicing and textual analysis / A. Qusef, G. Bavota, R. Oliveto, A. De Lucia, D. Binkley // Journal of systems and software. – 2014. – Vol. 88. – P. 147–168. – doi: 10.1016/j.jss.2013.10.019.

25. Slice-based statistical fault localization / X. Mao, Y. Lei, Z. Dai, Y. Qi, C. Wang // Journal of systems and software. – 2014. – Vol. 89. – P. 51–62. – doi: 10.1016/j.jss.2013.08.031.

26. *Keivanloo I., Rilling J.* Software trustworthiness 2.0 – A semantic web enabled global source code analysis approach // Journal of systems and software. – 2014. – Vol. 89. – P. 33–50. – doi: 10.1016/j.jss.2013.08.030.

27. Towards automated debugging in software evolution: evaluating delta debugging on real regression bugs from the developers' perspectives / K. Yu, M. Lin, J. Chen, X. Zhang // Journal of systems and software. – 2012. – Vol. 85, iss. 10. – P. 2305–2317. – doi: 10.1016/j.jss.2011.10.016.

Марков Александр Владимирович – магистр техники и технологии по специальности «Автоматизация и управление» (аспирант кафедры автоматики Новосибирского государственного технического университета). Основное направление исследования – анализ UML-диаграмм и сетей Петри. Имеет более 30 публикаций. E-mail: muviton3@gmail.com

Романников Дмитрий Олегович – кандидат технических наук, старший преподаватель кафедры автоматики НГТУ. Основное направление научных исследований – формальная верификация, проверка моделей. Имеет 30 публикаций. E-mail: rom2006@gmail.com

Review of the articles of Journal of Systems and Software for 2012–2014 years that devoted of software analysis*

A.V. Markov[1], D.O. Romannikov[2]

[1].*Novosibirsk State Technical University, 20 Karl Marks Avenue, Novosibirsk, 630073, Russian Federation, postgraduate student of the department of automation. E-mail: muviton3@gmail.com*

[2].*Novosibirsk State Technical University, 20 Karl Marks Avenue, Novosibirsk, 630073, Russian Federation, candidate of Technical Sciences, senior lecturer at the department of automation. E-mail: rom2006@gmail.com*

Review of the most interesting work based on authors point of view that were published in Journal of Systems and Software for 2012–2014 years. Works on testing and software development were selected for analysis. These articles can be divided into several groups: 1) papers on error localization; 2) papers on tests generations; 3) papers on software model constructions and further numerical analysis. We can conclude based on analysis considered work that error localization methods base on received from tests data and mathematical model that built by «slice cluster» method or analogs allow quite accurately detect operator that contains error. Different models that cover whole program or some part of it underlie of tests generation methods and these models allow list of tests and test data. Also some works of statistics analysis of source code are considered in the paper. And there are several works where rating of answers from program resources like stackoverflow.com is considered as grade standard.

Keywords: software, testing, formal verification, dynamic verification, verification, model checking, software models, graphs, total correctness of programs, test generation, errors localization

REFERENCES

1. Hoyos J.R., García-Molina J., Botía J.A. A domain-specific language for context modeling in context-aware systems. *Journal of systems and software*, 2013, vol. 86, iss. 11, pp. 2890–2905. doi: 10.1016/j.jss.2013.07.008

2. Neto A.C., Bonifácio R., Ribeiro M., Pontual C.E., Borba P., Castor F. A design rule language for aspect-oriented programming. *Journal of systems and software*, 2013, vol. 86, iss. 9, pp. 2333–2356. doi: 10.1016/j.jss.2013.03.104

3. Perez A., Abreu R., Riboira A. A dynamic code coverage approach to maximize fault localization efficiency. *Journal of systems and software*, 2014, vol. 90, pp. 18–28. doi: 10.1016/j.jss.2013.12.036

4. Diaz J., Arroyo D., Rodriguez F.B. A formal methodology for integral security design and verification of network protocols. *Journal of systems and software*, 2014, vol. 89, pp. 87–98. doi: 10.1016/j.jss.2013.09.020

5. Frantz R.Z., Corchuelo R., Molina-Jiménez C. A proposal to detect errors in Enterprise Application Integration solutions. *Journal of systems and software*, 2012, vol. 85, iss. 3, pp. 480–497. doi: 10.1016/j.jss.2011.10.048

* *Received 10 June 2014.*

6. Burgstaller B., Scholz B., Blieberger J. A symbolic analysis framework for static analysis of imperative programming languages. *Journal of systems and software*, 2012, vol. 85, iss. 6, pp. 1418–1439. doi: 10.1016/j.jss.2011.11.1039

7. Mattiello-Francisco F., Martins E., Cavalli A.R., Yano E.T. InRob: An approach for testing interoperability and robustness of real-time embedded software. *Journal of systems and software*, 2012, vol. 85, iss. 1, pp. 3–15. doi: 10.1016/j.jss.2011.02.034

8. Yu T., Sung A., Srisa-an W., Rothermel G. An approach to testing commercial embedded systems. *Journal of systems and software*, 2014, vol. 88, pp. 207–230. doi: 10.1016/j.jss.2013.10.041

9. Sinha S., Harrold M.J., Rothermel G. Interprocedural control dependence. *ACM Transactions on Software Engineering and Methodology*, 2001, vol. 10, iss. 2, 209–254. doi: 10.1145/367008.367022

10. Pachauri A., Srivastava G. Automated test data generation for branch testing using genetic algorithm: An improved approach using branch ordering, memory and elitism. *Journal of systems and software*, 2013, vol. 86, iss. 5, pp. 1191–1208. doi: 10.1016/j.jss.2012.11.045

11. Koong C., Shih C., Hsiung P., Lai H., Chang C., Chu W.C., Hsueh N., Yang C. Automatic testing environment for multi-core embedded software – ATEMES. *Journal of systems and software*, 2012, vol. 85, iss. 1, pp. 43–60. doi: 10.1016/j.jss.2011.08.030

12. Woungang I., Akinladejo F.O., White D.W., Obaidat M.S. Coding-error based defects in enterprise resource planning software: Prevention, discovery, elimination and mitigation. *Journal of systems and software*, 2012, vol. 85, iss. 7, pp. 1682–1698. doi: 10.1016/j.jss.2012.02.034

13. Islam S., Krinke J., Binkley D., Harman M. Coherent clusters in source code. *Journal of systems and software*, 2014, vol. 88, pp. 1–24. doi: 10.1016/j.jss.2013.07.040

14. Horwitz S., Reps T., Binkley D. Interprocedural slicing using dependence graphs. *ACM Transactions on Programming Languages and Systems*, 1990, vol. 12, iss. 1, pp. 26–60. doi: 10.1145/77606.77608

15. Ferrante J., Ottenstein K.J., Warren J.D. The program dependence graph and its use in optimization. *ACM Transactions on Programming Languages and Systems*, 1987, vol. 9, iss. 3, pp. 319–349. doi: 10.1145/24039.24041

16. Debroy V., Wong W. Combining mutation and fault localization for automated program debugging. *Journal of systems and software*, 2014, vol. 90, pp. 45–60. doi: 10.1016/j.jss.2013.10.042

17. Kolomvatsos K., Valkanas G., Hadjiefthymiades S. Debugging applications created by a domain specific language: the IPAC case. *Journal of systems and software*, 2012, vol. 85, iss. 4, pp. 932–943. doi: 10.1016/j.jss.2011.11.1009

18. Ju X., Jiang S., Chen X., Wang X., Zhang Y., Cao H. HSFal: Effective fault localization using hybrid spectrum of full slices and execution slices. *Journal of systems and software*, 2014, vol. 90, pp. 3–17. doi: 10.1016/j.jss.2013.11.1109

19. Kaminski G., Ammann P., Offutt J. Improving logic-based testing. *Journal of systems and software*, 2013, vol. 86, iss. 8, pp. 2002–2012. doi: 10.1016/j.jss.2012.08.024

20. Balsamo S., Harrison P.G., Marin A. Methodological construction of product-form stochastic Petri nets for performance evaluation. *Journal of systems and software*, 2012, vol. 85, iss. 7, pp. 1520–1539. doi: 10.1016/j.jss.2011.11.1042

21. Ibrahim W.M., Bettenburg N., Adams B., Hassan A.E. On the relationship between comment update practices and Software Bugs. *Journal of systems and software*, 2012, vol. 85, iss. 10, pp. 2293–2304. doi: 10.1016/j.jss.2011.09.019

22. Fan G., Yu H., Chen L., Liu D. Petri net based techniques for constructing reliable service composition. *Journal of systems and software*, 2013, vol. 86, iss. 4, pp. 1089–1106. doi: 10.1016/j.jss.2012.11.037

23. Alama O., Adams B., Hassan A.E. Controversy Corner: Preserving knowledge in software projects. *Journal of systems and software*, 2012, vol. 85, iss. 10, pp. 2318–2330. doi: 10.1016/j.jss.2012.03.028

24. Qusef A., Bavota G., Oliveto R., De Lucia A., Binkley D. Recovering test-to-code traceability using slicing and textual analysis. *Journal of systems and software*, 2014, vol. 88, pp. 147–168. doi: 10.1016/j.jss.2013.10.019

25. Mao X., Lei Y., Dai Z., Qi Y., Wang C. Slice-based statistical fault localization. *Journal of systems and software*, 2014, vol. 89, pp. 51–62. doi: 10.1016/j.jss.2013.08.031

26. Keivanloo I., Rilling J. Software trustworthiness 2.0 – A semantic web enabled global source code analysis approach. *Journal of systems and software*, 2014, vol. 89, pp. 33–50. doi: 10.1016/j.jss.2013.08.030

27. Yu K., Lin M., Chen J., Zhang X. Towards automated debugging in software evolution: Evaluating delta debugging on real regression bugs from the developers' perspectives. *Journal of systems and software*, 2012, vol. 85, iss. 10, pp. 2305–2317. doi: 10.1016/j.jss.2011.10.016.