УДК 004.4

А.М. Бершадский, А.С. Бождай, Ю.И. Евсеева, А.А. Гудков

ИССЛЕДОВАНИЕ И РАЗРАБОТКА МЕТОДОВ ДИНАМИЧЕСКОГО АНАЛИЗА КОДА ДЛЯ СОЗДАНИЯ САМОАДАПТИВНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Пензенский государственный университет, Пенза, Россия

В статье рассмотрены вопросы разработки методов динамического анализа кода для создания самоадаптивных программных систем. На сегодняшний день предпринято не так много попыток создания универсального теоретического аппарата синтеза самоадаптивных приложений, в то время как само направление исследований актуально: свойство самоадаптации позволит повысить качество разрабатываемого программного обеспечения и сократить временные и трудовые затраты на его разработку. Предлагаемый в работе подход развивает концепцию рефлексивной самоадаптации, предложенной в более ранних работах авторов. Центральной идеей нового подхода является разработка нового универсального метода самоадаптации программных систем, основанного на совместном использовании технологии динамического анализа кода и элементов теории трансляторов. На протяжении жизненного цикла программы осуществляется протоколирование вызовов основных функций, а затем на основе записанных вызовов строится множество динамических графов вызовов. Это множество становится основой более сложной структуры данных, используемой для анализа поведения системы. В такой структуре каждая вершина графа вызовов, представляющая собой функцию, имеет привязку к абстрактному синтаксическому дереву, которое является описанием действий, производимых функцией. Путем дальнейшего исследования полученной структуры данных находятся переменные, влияющие на результат выполнения программы. Дальнейший процесс самоадаптации заключается в варьировании значений данных переменных. Реализация полученных теоретических результатов может найти широкое применение в разработке самоадаптивных систем широкого круга, но в особенности, адаптивных тренажеров и обучающих приложений.

Ключевые слова: динамический анализ кода, граф вызовов, абстрактное синтаксическое дерево, интеллектуальный анализ данных, программная инженерия, самоадаптивные программные системы

1. Введение

Актуальность исследования обусловлена высокой сложностью современных программных систем и большими ресурсозатратами на их разработку и сопровождение.

Проблема сложности программного обеспечения порождает проблему снижения качества его функционирования: заранее сложно учесть, как поведет себя система при различных внешних условиях, и на подобные оценки не всегда есть время. Программа, показывающая хорошие (например, в плане производительности) результаты в одних ситуациях, может недостаточно хорошо работать в других. Более того, разработка сложной программной системы, предусматривающей функционирование в

различном программно-аппаратном окружении и при различных условиях (в том числе и пользовательских), может стать длительной и ресурсозатратной задачей.

Решением перечисленных проблем может стать создание новых методов динамического анализа кода, позволяющих реализовать самоадаптивное по отношению к конкретным внешним условиям (например, к различным программно-аппаратным конфигурациям, личностным характеристикам пользователей) поведение программного обеспечения. Данные методы позволят программной системе самоизменяться с минимальным человеческим участием. Также их применение позволит значительно снизить требования к квалификации лиц, занимающихся сопровождением программных продуктов, и уменьшить количество ресурсов, затрачиваемых на разработку и сопровождение.

В последние годы было проведено немало исследований по теме динамического анализа кода. Особенное внимание уделяется методам, в основе которых лежит технология виртуализации [1], применению динамического анализа в распознавании угроз [2], а также новым методам работы с динамическими графами вызовов [3]. Особенно следует выделить работы ИСП РАН в этом направлении, посвященные новому подходу к анализу защищенных программ [4], динамическому анализу пользовательских интерфейсов [5].

Однако, несмотря на имеющийся интерес к теме и наличие у методов динамического анализа соответствующего потенциала, до сих пор не была выдвинута идея об их применении в разработке адаптивного программного обеспечения. В то же время вопрос максимальной автоматизации труда программистов актуален уже не первое десятилетие. Можно выделить несколько групп исследований, посвященных проблеме: исследования, в которых возможным решением является "программирование без программиста", т. е. создание таких программных систем, которые способны самостоятельно писать работающий программный код; исследования, основное внимание в которых уделено созданию технических решений (программных каркасов, платформ, библиотек и т.д.) для разработки программного обеспечения, обладающего адаптивными свойствами; исследования в области программной кибернетики, посвященные переносу основных принципов классической кибернетики в сферу разработки и эксплуатации программного обеспечения, а также созданию нового теоретического аппарата самоадаптации, специфичного именно для сферы программных систем.

К первой группе относятся ряд исследований в области искусственного интеллекта, в частности, нейронных сетей. Например, в работе [6] говорится о создании нейронной сети, способной генерировать

работающий программный код, а работа [7] посвящена созданию алгоритма генерации кода веба-интерфейса на основе изображения макета. Однако применимость предложенных техник в реальных практических задачах на сегодняшний день сомнительна: нейронные сети способны пока что программировать только простейшие алгоритмы и генерировать наиболее примитивные интерфейсы. Исследования второй группы являются сугубо прикладными, разрабатываемые в их рамках технологии и архитектурные решения применимы к узкому классу задач. К таким работам относится, прежде всего, ряд публикаций о создании адаптивных компьютерных игр и виртуальных тренажеров [8-12].

В относительно молодой области программной кибернетики в последние годы издано достаточно большое количество работ, затрагивающих различные аспекты разработки, сопровождения и адаптации программных систем. Из имеющихся публикаций можно отметить: работу [13], посвященную процессам адаптации и самоорганизации в сетевых системах; работу [14], связанную с развитием концепции автономных вычислений; работу [15], в которой изложен основанный на конечных автоматах подход к самоадаптации программного обеспечения; работу [16], посвященную возможностям применения аппарата нечеткой логики для реализации адаптивного поведения программной системы. Обобщая, можно сделать вывод, что перечисленные работы представляют определенный интерес, в них содержатся подходы к решению проблемы минимизации труда специалистов, однако всем им присущ один недостаток: изложенные техники ограничены в применимости, не универсальны.

Основная идея предлагаемого подхода заключается в протоколировании вызовов функциональных объектов программы и применении к составленным протоколам методов интеллектуального анализа данных для выявления закономерностей. Выявление закономерностей позволит находить те компоненты программы, которые в текущих условиях работают не оптимально, и автоматически реконфигурировать их. Выделенные компоненты будут способны к дальнейшей реструктуризации на основе информации, циркулирующей в программном приложении (как внутренней системной информации, например, технических характеристик устройства, на котором запускается программа, так и получаемой извне, например, в форме пользовательских мнений и предпочтений). В качестве основной модели для анализа протоколируемых вызовов функциональных объектов предлагается использование динамического графа вызовов с расширенными свойствами.

2. Материалы и методы

2.1. Математическая модель рефлексии уровня программного кода.

Приведенная в работе [17] концепция рефлексии подразумевает, что все возможные адаптивные компоненты уже определены, неизвестны только зависимости между их конкретными состояниями и циркулирующими в программной системе данными. Однако вполне вероятна ситуация, когда не все адаптивные компоненты и их возможные состояния выделены и описаны. Чтобы решить эту проблему, необходимо ввести дополнительный уровень программной рефлексии, содержащий механизмы поиска и формирования рефлексивных компонентов. Назовем его уровнем рефлексии программного кода, а уровень, включающий в себя модели изменчивости и алгоритмы интеллектуального анализа данных, будем называть уровнем поведенческой рефлексии.

Для того, чтобы вычислить потенциально адаптивное поведение системы, необходимо иметь подробные сведения о ее функционировании на различных временных промежутках и с различными исходными данными. В случае с программной системой такое представление может дать динамический граф вызовов, который, по сути, представляет собой запись выполнения программы. Анализируя динамический граф, можно реализовать один из способов самоадаптации программного обеспечения: автоматическое изменение значений констант, влияющих на результат работы программы, в зависимости от определенных внешних условий (пользовательских, аппаратных и т.д.). Заранее, на этапе разработки, может быть неизвестно, какое именно значение той или иной константы оптимально для того или иного внешнего фактора. Для обнаружения и анализа влияющих констант удобно использовать абстрактное синтаксическое дерево - структуру данных, позволяющую выявить влияние одних сущностей программы на другие.

Основными компонентами математической модели уровня рефлексии программного кода будут:

1. Динамический граф вызовов, позволяющий отследить выполнение программы во времени. В нашем случае целесообразным является использование графа вызовов с расширенными свойствами: каждая вершина графа, представляющая собой вызов метода, будет иметь привязку к абстрактному синтаксическому дереву, содержащему переменные, используемые методом, и операции над ними.

2. Абстрактное синтаксическое дерево, с помощью которого можно определить взаимное влияние элементов системы друг на друга. Дополнительным преимуществом его использования является возможность отслеживания не только крупных сущностей программы, описываемых классами, но и атомарных коэффициентов, чья область

видимости может быть ограничена методом или циклом внутри метода. Влияние таких переменных часто недооценивается.

Под графом будем понимать пару $G = (V, E)$, где V - множество вершин, $E \subseteq V \times V$ - множество ребер. В случае с задачей динамического анализа интерес будут представлять только ориентированные графы, т.е. графы, ребра(дуги) которых имеют направление. Графом вызовов называется ориентированный мультиграф G с множеством вершин $V = V(G)$, каждый элемент которого представляет собой функцию, и множеством дуг $E(G) = V(G) \times V(G)$, где каждый элемент является вызовом одной функции из другой.

Абстрактное синтаксическое дерево можно представить парой $A = (L, B)$, где L - множество вершин, а B - множество ребер. При этом внутренние узлы такого дерева являются операторами языка программирования, а листья - операндами. Ребра показывают, какие операнды используются тем или иным оператором.

В случае с рассматриваемой задачей каждое абстрактное синтаксическое дерево является весом для некоторой вершины динамического графа вызовов.

Математическая модель рефлексии уровня программного кода будет иметь вид

$$M = (R, K, A), \quad (1)$$

где $R = \{R_i\}, , i = \mathbf{1}, ..., N$ - множество, каждый элемент которого содержит информацию об определенном этапе выполнения программы, N - число запусков; $A = \{A_j\}, j = \mathbf{1}, ..., |V(G_i)|$ - множество абстрактных синтаксических деревьев, элементы которого соответствуют функциям программы (вершинам динамических графов вызовов); $K = \{K_t\}, t = \mathbf{1}, ..., x$ - множество имен ключевых переменных и диапазонов оптимальных значений (переменных, по результатам которых определяется, оптимально ли работала программа, например, показатель производительности или результат прохождения пользователем обучающего приложения), при этом $K_t = (KN_t, D_t)$, где KN_t - имя переменной, D_t - диапазон значений, x - число переменных.

Применительно к элементам множества R справедливо:

$$R_i = (G_i, E_i, KV_i), \quad (2)$$

т.е. каждый элемент множества R представляет собой кортеж, элементы которого характеризуют состояние выполнения программы на i-м запуске (N - общее число запусков): G_i - динамический граф вызовов, полученный на i-м запуске; E_i - множество значений параметров окружения (характеристик пользователя, программно-аппаратных характеристик и т.д.), которые характеризуют программу на i-ом запуске;

$KV_i = \{KVij\}, j = \mathbf{1},..., x$ - множество значений ключевых переменных на i-ом запуске.

2.2. Метод рефлексивной самоадаптации программного кода

Метод рефлексивной самоадаптации уровня программного кода будет включать следующие этапы:

1. Определение списка параметров окружения (характеристик пользователя, программно-аппаратных характеристик и т.д.), которые характеризуют программу на каждом запуске.
2. Определение списка переменных, по значениям которых можно определить, достаточно ли оптимальным образом работает программа (ключевые переменные). Определение диапазонов значений оптимальности для переменных.
3. Протоколирование вызовов функций программы на протяжении ряда запусков.
4. Обработка множества протоколов: отбор только тех записей, которые относятся к запускам, завершившимся неудачно. Неудачным завершением программы является ситуация, при которой одна или несколько ключевых переменных выходят за границы диапазонов оптимальности.
5. Построение множества динамических графов вызовов на основе составленных протоколов, затем множеств R_i в соответствии с формулой (2).
6. Кластеризация множеств R_i на подмножества в соответствии с тем, какие именно ключевые переменные и для вызовов с какими именно значениями параметров окружения были неоптимальными. При кластеризации также учитывается то, насколько сильно ключевая переменная отклоняется от оптимальной зоны.
7. Нахождение центра каждого кластера.
8. Анализ графов вызовов в найденных центрах кластеров. Для переменной, значение которой было неоптимальным для графа, с помощью привязанных к каждой функции абстрактных синтаксических деревьев ищутся константы, повлиявшие на ее итоговое значение.
9. Варьирование значений найденных констант таким образом, чтобы значение ключевых переменных попало в диапазон оптимальности. Возможны следующие варианты реализации:
    a. При разборе синтаксического дерева находятся не только влияющие константы, но и зависимость ключевых переменных от них. На основе найденной зависимости можно установить значения влияющих констант таким образом, чтобы ключевые переменные принимали значение в диапазоне оптимальности.

b. Составляется таблица, в качестве столбцов содержащая параметры окружения, варьируемые значения влияющих констант и зависимое от них значение ключевых переменных. В специализированной "песочнице" осуществляется "воспроизведение" действий, записанных в текущий граф вызовов, для новых значений влияющих констант, получается новое значение ключевой переменной и заносится в таблицу. Как только количество записей в таблице достигает определенного (заранее заданного разработчиком или пользователем) объема, осуществляется их анализ (например, методами Data Mining) и подбираются нужные значения констант. Итогом работы метода становятся подобранные значения влияющих констант для определенных внешних условий (характеристик пользователя, аппаратуры и т.д.).

Схема процесса самоадаптации программной системы, реализуемого с помощью предложенного метода, представлена на Рисунке 1.

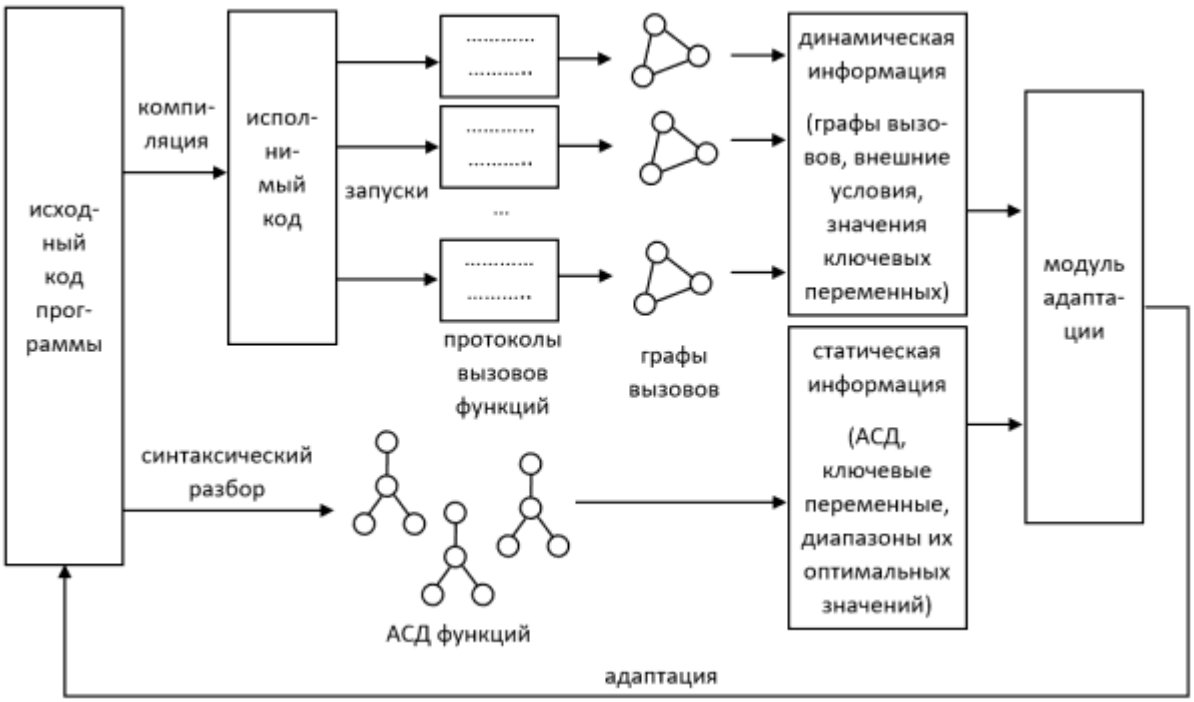

Рисунок 1 - Схема процесса самоадаптации программной системы

Рассмотрим более подробно реализацию этапа 8.

2.3. Алгоритм поиска влияющих констант

Алгоритм поиска констант, влияющих на значение ключевых переменных, включает следующие шаги:

1. Цикл по элементам множества значений ключевых переменных на *i*-м запуске программы KV_i. Для очередного значения KV_{ij}, $j = \mathbf{1},...x$, не попадающего в диапазон оптимальности D_j запоминается имя этой переменной KN_j. Если переменных больше нет, завершение работы алгоритма.
2. Обход тех элементов множества абстрактных синтаксических деревьев A, которые относятся к функциям, вызов которых был осуществлен на рассматриваемом запуске программы. Поиск среди вершин каждого дерева вершины, соответствующей текущей ключевой переменной KN_j. В случае, если вершина найдена, проверка родительской по отношению к ней вершины. Если родительская вершина является оператором присваивания, а сама вершина находится в левом дочернем узле, то необходимо перейти к пункту 3, если вызовом функции - к пункту 5, если вершиной иного типа - продолжить поиск. Если переменная не найдена, переход к пункту 1.
3. Обход правого поддерева родительской вершины. Поиск в нем влияющих констант. Проверка, является ли очередная вершина константой или переменной. В первом случае запоминаем эту вершину (поскольку оно и есть влияющая константа), во втором случае рекурсивно вызываем процедуру поиска влияющих констант для найденной переменной. Переход к пункту 2.
4. В случае если очередная вершина является вызовом функции и ключевая переменная передана в эту функцию по ссылке, то для параметра функции, соответствующего ключевой переменной, осуществляется поиск влияющих констант в абстрактном синтаксическом дереве данной функции. Переход к пункту 2.

4. Заключение

В отличие от метода адаптации, рассмотренного в работе [20], метод, основанный на динамическом анализе кода, не требует вынесения предположения о возможных зависимостях между переменными или компонентами программы. Модель изменчивости [20] модифицируется самой системой без участия в процессе самоадаптации человека.

Реализация полученных теоретических результатов может найти широкое применение в разработке самоадаптивных систем широкого круга, но в особенности, адаптивных тренажеров и обучающих приложений.

Исследование выполнено при финансовой поддержке РФФИ в рамках научного проекта № 18-07-00408.

ЛИТЕРАТУРА

1. Yu.Chernov, A.S.Konoplev. The use of virtualization technology in the dynamic analysis of software code // Automatic Control and Computer Sciences. - Springer, NY: 2015. - P. 834-837.
2. P.V.Shijo, A.Salimb. Integrated Static and Dynamic Analysis for Malware Detection // Procedia Computer Science. - Elsevier, Amsterdam: 2015. - P. 804-811.
3. M. Burch. The Dynamic Call Graph Matrix // VINCI '16 Proceedings of the 9th International Symposium on Visual Information Communication and Interaction.- ACM, NY: 2016. - P. 1-8.
4. Аветисян А.И., Тихонов А.Ю. Комбинированный (статический и динамический) анализ бинарного кода // Труды Института системного программирования РАН (электронный журнал). — 2012. — № 4 (22). — С. 131-52.
5. С.П. Вартанов, А.Ю. Герасимов, М.К. Ермаков, Д.О. Куц, А.А. Новиков. Динамический анализ приложений с графическим пользовательским интерфейсом на основе символьного исполнения // Труды Института системного программирования РАН (электронный журнал). — 2017. — № 1 (29). — С. 149-166.
6. Balog M., Gaunt A.L., Brockschmidt M., Nowozin S., Tarlow D. DeepCoder: Learning to Write Programs // https://www.sciencedirect.com/science/journal/18770509 ICLR 2017: 5th International Conference on Learning Representations.- ACM, NY: 2017. - P. 1-20.
7. Beltramelli T. pix2code: Generating Code from a Graphical User Interface Screenshot // https://www.sciencedirect.com/science/journal/18770509 ICLR 2018: 6th International Conference on Learning Representations.- ACM, NY: 2018. - P. 47-54.
8. Wang, A.L. Software architectures and the creative processes in game development / A.L. Wang, N. Nordmark // Entertainment computing / edited by K. Chorianopoulos [et al.]. — Cham: Springer International Publishing, 2015. — P. 272-285.
9. Cornforth, D.J. Cluster evaluation, description and interpretation for serious games / D. J.Cornforth, M. T.Adam // Serious games analytics / edited by C. S. Loh, Y. Sheng, D. Ifenthaler. — Cham: Springer International Publishing, 2015. — P. 135-155.
10. Carvalho, M.B. The journey: a service-based adaptive serious game on probability / M.B. Carvalho [et al.] // Serious games analytics / edited by C. S. Loh, Y. Sheng, D. Ifenthaler. — Cham: Springer International Publishing, 2015. — P. 97-106.

11. Fasli, M. Designing and developing electronic market games / M. J. Fasli, M. Michalakopoulos // Games and learning alliance / edited by A. De Gloria. — Heidelberg: Springer Berlin Heidelberg, 2007. — P. 117-151.
12. Bernon, C. Engineering self-organizing systems / C. Bernon [et al.] // Self-organizing software: from natural to artificial adaptation / edited by G. Di Marzo Serugendo, M.P. Gleizes, A. Karageorgos. — Heidelberg: Springer Berlin Heidelberg, 2011. — P. 283-312.
13. Ravindran, K., Rabby, M. Software cybernetics to infuse adaptation intelligence in networked systems // IEEE International Conference on the Network of the Future (NOF).. - Washington: IEEE Computer Society, 2013. - P. 1-6.
14. Ahuja, K., Dangey, H. Autonomic Computing: An emerging perspective and issues // IEEE International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT 2014). - Washington: IEEE Computer Society, 2014. - P. 471-475.
15. Wang P., Cai, K. Y. Representing extended finite state machines for SDL by a novel control model of discrete event systems // Sixth IEEE International Conference on Quality Software (QSIC 2006). - Washington: Ieee Computer Society, 2006. - P. 159-166.
16. Yang, Q., Lü, J., Xing, J., Tao, X., Hu, H., Zou, Y. Fuzzy control-based software self-adaptation: A case study in mission critical systems // IEEE 35th Annual Computer Software and Applications Conference Workshops (COMPSACW). - Washington: IEEE Computer Society, 2011. - P. 13-18.
17. Бершадский А.М., Бождай А.С., Евсеева Ю.И., Гудков А.А. Математическая модель рефлексии самоадаптивных программных систем // Известия волгоградского государственного технического университета. — 2018. — № 8 (218). — С. 7-14.

A.M. Bershadsky, A.S. Bozhday, Y.I. Evseeva, A.A. Gudkov

RESEARCH AND DEVELOPMENT OF METHODS OF DYNAMIC ANALYSIS OF CODE FOR CREATING SELF-ADAPTIVE SOFTWARE

Penza State University, Penza, Russia

The article deals with the development of methods for dynamic code analysis for creating self-adaptive software systems. To date, not so many attempts have been made to create a universal theoretical apparatus for the synthesis of self-adaptable applications, while the research direction itself is relevant: the self-adapting feature will improve the quality of the software being developed and reduce the time and labor costs of its development. The proposed approach develops the concept of reflexive self-adaptation proposed in the earlier works of the authors. The central idea of the new approach is the development of a new universal method of self-adaptation of software systems based on the joint use of technology for dynamic analysis of code and elements of the theory of translators. Throughout the life cycle of the program, the calls of the main functions are recorded, and then a set of dynamic call graphs is constructed on the basis of the recorded

calls. This set becomes the basis of a more complex data structure used to analyze the behavior of the system. In such a structure, each vertex of the call graph, which is a function, is bound to an abstract syntax tree, which is a description of the actions performed by the function. By further researching the obtained data structure, variables are found that influence the result of the program execution. The further self-adaptation process consists in the variation of these variables value. The implementation of the obtained theoretical results can be widely used in the development of self-adaptive systems of a wide range, but in particular, adaptive simulators and training applications.

Keywords: dynamic code analysis, call graph, abstract syntax tree, data mining, software engineering, self-adaptive software systems

The study was carried out with the financial support of the Russian Foundation for Basic Research (research project No. 18-07-00408).

REFERENCES

1. Yu. Chernov, A.S. Konoplev. The use of virtualization technology in the dynamic analysis of software code // Automatic Control and Computer Sciences. - Springer, NY: 2015. - P. 834-837.
2. P.V.Shijo, A.Salimb. Integrated Static and Dynamic Analysis for Malware Detection // Procedia Computer Science. - Elsevier, Amsterdam: 2015. - P. 804-811.
3. M. Burch. The Dynamic Call Graph Matrix // VINCI '16 Proceedings of the 9th International Symposium on Visual Information Communication and Interaction.- ACM, NY: 2016. - P. 1-8.
4. Avetisyan A.I., Tikhonov A.Yu. Combined (static and dynamic) binary code analysis // Proceedings of the Institute for System Programming of the Russian Academy of Sciences (electronic journal). - 2012. - № 4 (22). - pp. 131-52.
5. S.P. Vartanov, A.Yu. Gerasimov, M.K. Yermakov, D.O. Kuts, A.A. Novikov. Dynamic analysis of applications with a graphical user interface based on symbolic execution // Proceedings of the Institute for System Programming of the Russian Academy of Sciences (electronic journal). - 2017. - № 1 (29). - p. 149-166.
6. Balog M., Gaunt A.L., Brockschmidt M., Nowozin S., Tarlow D. DeepCoder: Learning to Write Programs // https://www.sciencedirect.com/science/journal/18770509ICLR 2017: 5th International Conference on Learning Representations.- ACM, NY: 2017. - P. 1-20.
7. Beltramelli T. pix2code: Generating Code from a Graphical User Interface Screenshot // https://www.sciencedirect.com/science/journal/18770509ICLR 2018: 6th

International Conference on Learning Representations.- ACM, NY: 2018. - P. 47-54.
8. Wang, A.L. Software architectures and the creative processes in game development / A.L. Wang, N. Nordmark // Entertainment computing / edited by K. Chorianopoulos [et al.]. — Cham: Springer International Publishing, 2015. — P. 272-285.
9. Cornforth, D.J. Cluster evaluation, description and interpretation for serious games / D.J.Cornforth, M. T.Adam // Serious games analytics / edited by C. S. Loh, Y. Sheng, D. Ifenthaler. — Cham: Springer International Publishing, 2015. — P. 135-155.
10. Carvalho, M.B. The journey: a service-based adaptive serious game on probability / M.B. Carvalho [et al.] // Serious games analytics / edited by C. S. Loh, Y. Sheng, D. Ifenthaler. — Cham: Springer International Publishing, 2015. — P. 97-106.
11. Fasli, M. Designing and developing electronic market games / M. J. Fasli, M. Michalakopoulos // Games and learning alliance / edited by A. De Gloria. — Heidelberg: Springer Berlin Heidelberg, 2007. — P. 117-151.
12. Bernon, C. Engineering self-organizing systems / C. Bernon [et al.] // Self-organizing software: from natural to artificial adaptation / edited by G. Di Marzo Serugendo, M.P. Gleizes, A. Karageorgos. — Heidelberg: Springer Berlin Heidelberg, 2011. — P. 283-312.
13. Ravindran, K., Rabby, M. Software cybernetics to infuse adaptation intelligence in networked systems // IEEE International Conference on the Network of the Future (NOF). - Washington: IEEE Computer Society, 2013. - P. 1-6.
14. Ahuja, K., Dangey, H. Autonomic Computing: An emerging perspective and issues // IEEE International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT 2014). - Washington: IEEE Computer Society, 2014. - P. 471-475.
15. Wang P., Cai, K. Y. Representing extended finite state machines for SDL by a novel control model of discrete event systems // Sixth IEEE International Conference on Quality Software (QSIC 2006). - Washington: Ieee Computer Society, 2006. - P. 159-166.
16. Yang, Q., Lü, J., Xing, J., Tao, X., Hu, H., Zou, Y. Fuzzy control-based software self-adaptation: A case study in mission critical systems // IEEE 35th Annual Computer Software and Applications Conference Workshops (COMPSACW). - Washington: IEEE Computer Society, 2011. - P. 13-18.
17. Bershadsky A.M., Bozhdai A.S., Evseeva Y.I., Gudkov A.A. Mathematical model of reflection of self-adaptive software systems // News of Volgograd State Technical University. - 2018. - № 8 (218). - p. 7-14.