УДК 004.4
doi:10.21685/2227-8486-2021-2-10

МЕТОДЫ САМОАДАПТАЦИИ ПРОГРАММНЫХ СИСТЕМ, ОСНОВАННЫЕ НА МАШИННОМ ОБУЧЕНИИ И ИНТЕЛЛЕКТУАЛЬНОМ АНАЛИЗЕ ДАННЫХ

А. С. Бождай[1], Ю. И. Евсеева[2], А. А. Гудков[3]

[1, 2, 3] Пензенский государственный университет, Пенза, Россия
[1] bozhday@yandex.ru, [2] shymoda@mail.ru, [3] alexei-ag@yandex.ru

Аннотация. *Актуальность и цели.* Цель исследования – изучение вопросов по созданию универсальной технологии самостоятельной адаптации программных систем, математического аппарата, на котором основана данная технология. *Материалы и методы.* Чтобы создать универсальную технологию самостоятельной адаптации, используются методы интеллектуального анализа данных и машинного обучения. *Результаты.* Методы, которые предлагается применять для самоадаптации программных систем, позволяют выявить те знания про предметную область программы, о которых ничего не было известно, когда она только разрабатывалась. *Выводы.* Использование предложенных методов открывает возможности для создания программного обеспечения, которое имеет расширенный жизненный цикл, более высокие показатели надежности и производительности. Затраты ресурсов на его разработку и сопровождение будут ниже.

Ключевые слова: самоадаптивное программное обеспечение, интеллектуальный анализ данных, машинное обучение

Для цитирования: Бождай А. С., Евсеева Ю. И., Гудков А. А. Методы самоадаптации программных систем, основанные на машинном обучении и интеллектуальном анализе данных // Модели, системы, сети в экономике, технике, природе и обществе. 2021. № 2. С. 144–152. doi:10.21685/2227-8486-2021-2-10

METHODS OF SELF-ADAPTATION OF SOFTWARE SYSTEMS BASED ON MACHINE LEARNING AND INTELLECTUAL DATA ANALYSIS

A.S. Bozday[1], Yu.I. Evseeva[2], A.A. Gudkov[3]

[1, 2, 3] Penza State University, Penza, Russia
[1] bozhday@yandex.ru, [2] shymoda@mail.ru, [3] alexei-ag@yandex.ru

Abstract. *Background.* The issues of creating a universal technology for self-adaptation of applied software systems, as well as the mathematical apparatus underlying such a technology, are being studied. *Materials and methods.* Methods of data mining and machine learning are considered as methods for creating a universal technology of self-adaptation. *Results and conclusions.* Methods of self-adaptation of software systems are proposed, a distinctive feature of which is the identification of this knowledge about the subject area of the program that was unknown at the development stage. The proposed methods will make it possible to create software with an extended life cycle, which is distinguished by greater reliability and performance, as well as lower resource cost for development and maintenance.

Keywords: self-adaptive software, data mining, machine learning

For citation: Bozday A.S., Evseeva Yu.I., Gudkov A.A. Methods of self-adaptation of software systems based on machine learning and intellectual data analysis. *Modeli, sistemy, seti v ekonomike, tekhnike, prirode i obshchestve = Models, systems, networks in economics, technology, nature and society*. 2021;2:144–152. (In Russ.). doi:10.21685/2227-8486-2021-2-10

Введение

Создание методов, которые обеспечат эффективную самоадаптацию программного обеспечения, – актуальная проблема, что обусловлено высокой сложностью существующих сегодня программных систем, большими затратами на их разработку и сопровождение. Если система будет способна самостоятельно адаптироваться, появится возможность для выявления скрытых знаний и факторов об ее предметной области, которые могут не знать даже эксперты, когда проектируется приложение. В будущем данные знания и факторы можно использовать, чтобы создавать новые модели алгоритмов, определять самостоятельно ситуации, где требуется увеличение/снижение количества ресурсов, необходимых, чтобы оптимизировать производительность.

Чтобы выявлять скрытые знания про предметную область, необходимо применять технологию, которая предполагает интеллектуальный анализ данных. В ходе анализа данных, которые показывают, как работала программа ранее в течение определенного промежутка времени, можно увидеть, какие зависимости и закономерности существуют. На основании полученных результатов в программе формируются новые поведенческие обратные связи.

Самоадаптивное программное обеспечение с применением интеллектуального анализа информации

Авторы статьи в работе [1] предлагают для самоадаптации метод, в основе которого лежит интеллектуальный анализ информации. Метод базируется на концепции рефлексивной самоадаптации, предложенной авторами в работе [2]. Данная концепция характеризуется наличием базовых принципов, при ее внедрении в сфере программной инженерии должны соблюдаться определенные требования:

1. Механизм самостоятельной адаптации должен быть неизменчивым в отношении предметной области, типа и назначения программной системы.

2. Должна быть возможность выдерживать рабочую нагрузку, когда ресурсы увеличиваются (масштабируемость). Когда анализируются способности программной системы, важно знать, что ее уровень может быть разным. Это и самая простая утилита, и огромный программный комплекс, объединяющий в себе две системы и больше. Нужно также сделать акцент на том, что адаптироваться может вся система в целом, а также компоненты, которые входят в ее состав. Для этого применяются одни методы.

3. Информация, накопленная в информационных массивах минимум за два года, подвергается интеллектуальному анализу. Для возможности формирования новых обратных связей сначала нужно понять, какие между данными закономерности, как они зависят друг от друга, чтобы сделать вывод о том, как за определенный период данная программная система функционировала ранее.

Когда применяется принцип масштабируемости, архитектурная организация программной системы должна отвечать ряду определенных требований. Прежде всего, это возможность создавать модульные архитектуры, ко-

торые могут адаптироваться. Другими словами, должна быть возможность не только внесения изменений в программу по результатам, которые удалось получить при проведении рефлексивного анализа, но и внешней модификации программного обеспечения. Также нужно вести учет изменений системы уже при использовании программы. Когда выполняется подключение нового модуля, одновременно нужно включить общую рефлексию. Это позволит выявлять его связь с другими модулями, определять, как они воздействуют друг на друга. Также требуется выявление закономерностей во время его работы.

Обобщив имеющийся опыт в разработке программного обеспечения, которое имеет архитектуру модульного типа, авторы определили, что нужно обязательно учитывать, чтобы создавать программное обеспечение, которое сможет самостоятельно приспосабливаться:

1. Блоки изменчивости. Данные компоненты программной среды, которые подвержены адаптации, должны быть обязательно. Для каждого блока характерны свои функции, чтобы реализовать класс, функцию, модель объекта, предметную область и др.

2. Логически связанные между собой блоки изменчивости в совокупности из одной предметной области используются, чтобы решать одну задачу, – рефлексивный компонент.

3. В составе программного обеспечения рефлексивного типа может быть не только один, но и несколько рефлексивных элементов. Сколько именно будет таких элементов, определяет сложность программы и задачи адаптации.

4. Чтобы создать систему таких компонентов, используется общий интерфейс. В начале формирования у компонентов может не быть информации друг о друге. Но, когда начинается рефлексивный анализ, выявляются степень и характер влияния между составляющими систему элементами. Чтобы обмен информацией был качественным, требуется создание каналов.

Главная роль при построении данного метода отводится блоку изменчивости. Имеется в виду возможность исполнения функции, класса, модели, объекта предметной области и других атомарных элементов программы. Что касается рефлексивного компонента, то под ним подразумеваются все блоки изменчивости, между которыми существует логическая связь. Они должны относиться к одной предметной области, обеспечивать решение одной задачи.

Рассматриваемый блок может задавать разные стратегии адаптивного поведения отдельного компонента сервиса, в целом всего сервиса, в целом всей системы. Следовательно, нельзя назвать атомарным компонентом блок изменчивости, потому что его структура имеет иерархию, в состав могут входить другие блоки.

Блок изменчивости можно определить так:

$$V_B = (SubF, BlockParams, BlockStates, Rule),$$

где $SubF$ – модель поведения блока изменчивости как ориентированный гиперграф; $BlockParams = \{BlockParam_1, BlockParam_2, ..., BlockParam_k\}$ – параметры, определяющие состояние блока изменчивости; $BlockStates = \{BlockState_1, BlockState_2, ..., BlockState_z\}$ – состояния, в которых пребывает блок, при этом $BlockState_i \subseteq SubF \; \forall i = 1,...,z$; $Rule : BlockParams' \rightarrow BlockStates$ – функция, ставящая в соответствие каждому элементу

$BlockParams'$ определенный компонент из $BlockStates$. Под совокупностью элементов $BlockParams'$ следует понимать подмножества из $BlockParams$.

Возможно также представить все элементы в $BlockParams$ в следующем виде:

$$BlockParams = ExtParams \cup InnerParams \cup TargetParams,$$

где $ExtParams = \{ExtParam_1, ExtParam_2, ..., ExtParam_{k1}\}$ – все параметры, которые в отношении анализируемого блока будут внешними; $InnerParams = \{InnerParam_1, InnerParam_2, ..., InnerParam_{k2}\}$ – внутренние параметры блока, которые могут варьироваться; $TargetParams = \{TargetParam_1, TargetParam_2, ..., TargetParam_{k3}\}$ – целевые параметры. Элементы $TargetParams$ и некоторые целевые функции связаны между собой. Чтобы их определить, применяется экспертный метод.

Основные этапы метода самоадаптации:

1. Разработчик (эксперт) вручную определяет множества $ExtParams$, $InnerParams$ и $TargetParams$.

2. Вручную задаются условия для $TargetParams$.

3. Вручную определяется количество (в процентном выражении) значений, которые будут оставлены на пятой стадии.

4. Автоматически системой осуществляется сбор информации про свои функции за конкретный период. Из входных данных этапа создается таблица, где первые столбцы – это $ExtParams$, остальные – $InnerParams$ и $TargetParams$.

5. Выполняется обработка значений таблицы: остаются строки, где значения $TargetParams$ удовлетворяют функции, которые должны выполняться (целевые). Для данного этапа возможно применение кластеризации. Выходные данные – таблица записей (столбцы с $TargetParams$ не учитываются).

6. Определяются зависимости $ExtParams$ и $InnerParams$, для чего проводится алгоритм интеллектуального анализа информации.

7. Результаты анализа позволяют сформулировать функцию $Rule: BlockParams' \rightarrow BlockStates$. Это будет функция-обертка над функцией, которую возвращает алгоритм Data Mining. *Rule* обеспечивает формирование подграфа из множества $BlockStates$ элементов в составе $ExtParams$.

Чтобы понять суть данного метода и особенностей его применения, возьмем мобильное приложение, которое обладает функцией полноценной автономной работы (нет необходимости в подключении к интернету). Это приложение позволяет выполнить тестирование, как пользователь может мыслить пространственно. Для достижения данной цели используются определенные алгоритмы, которые создают картинки. В их числе следующие: на основе глубоких нейросетей, генетический, случайный перебор картинок и комбинация их фрагментов. Чтобы сделать правильный выбор, важен такой критерий, как аппаратная конфигурация используемого устройства. При этом считается, что основной код программного обеспечения не содержит данных про конкретные конфигурации мобильных устройств, также не устанавлива-

ются условные операторы выбора. Приложение само должно установить зависимости и выбрать требуемый вид алгоритма сразу же, когда будет завершена инсталляция и выполнен запуск приложения.

Внешние параметры приложения (*ExtParams*) – это основные характеристики мобильного девайса. Важно знать, что производительность устройства, когда выбирается определенный алгоритм, зависит не от всех характеристик, которые были перечислены. К задачам рефлексии также относится следующая: выявить конкретные характеристики, от которых зависит производительность. Внутренние варьируемые параметры (*InnerParams*) – это непосредственно сам алгоритм, который может быть простым, генетическим, нейросетевым. Также к ним относятся такие параметры алгоритма, как картинки, количество клеток или фрагментов, а также сложность. В число целевых параметров (*TargetParams*) входит период времени, в течение которого визуализируется картинка, важно, чтобы данный период был максимально коротким, насколько это возможно.

Метод, позволяющий программному обеспечению самостоятельно адаптироваться, характеризуется универсальностью и масштабируемостью, дает возможность увидеть, как взаимодействуют разные системы, а также, какие зависимости существуют в пределах одной системы. Таким образом, улучшение будет происходить в течение всей жизни программной системы. Чтобы решить данную задачу, не нужна помощь профессиональных специалистов по программированию.

Самоадаптивное программное обеспечение через машинное обучение

Авторы работы [3] предлагают метод, в основе которого машинное обучение с подкреплением, чтобы программная система имела возможность эффективно регулировать потребляемые ресурсы. Суть данного метода в том, что в процессе взаимодействия со средой система (агент) обучается. В большинстве случаев данное взаимодействие происходит в течение определенного времени (временные шаги) [4]. Когда агент совершает шаг, ведется наблюдение за состоянием программной среды. По полученным результатам осуществляется выбор действий. Когда агентом выполнено действие, начисляется определенное числовое значение как вознаграждение. После этого происходит изменение его состояния. У нас агент – это программа, которая обладает способностью самостоятельно приспосабливаться, она, используя свои датчики, ведет наблюдение за тем, как меняется среда. Чтобы рассчитать вознаграждение, измеряются разные факторы, которые имеют значение для работы системы, формируются функции полезности [5].

Но ситуация, описанная в работе [6], которая является одной из центральных концепций в самоадаптивном программном обеспечении, в классическом варианте обучения с подкреплением не учитывается. Состояние агента может быть неизменным, но в разных ситуациях его действия должны быть разными. В зависимости от ситуации выбор разных вариантов действий может быть ограничен. Например, суперкомпьютером может осуществляться управление процессом, в ходе которого распределяются собственные ресурсы. Когда рабочая нагрузка растет, чтобы повысить производительность, требуется увеличение количества ресурсов. Если же нужно снижение потребления энергии, когда снижается рабочая нагрузка, требуется уменьшить ресурсы, которые потребляются.

Чтобы применить данный метод при создании программного обеспечения, которое может самостоятельно приспосабливаться, требуется расширить определение классического варианта марковского процесса, когда принимаются решения. Нужно предусмотреть возможность определять ситуацию [7].

Пусть $I = \{i_1, i_2, ..., i_n\}$ – все имеющиеся ситуации. Ситуация $i_i \in I$ является показателем среды на данный момент или отражает событие внутри программного обеспечения, при котором начинается адаптация агента.

При этом $A \mid s_i, i_i$ говорит о том, что действия агента могут быть разными в s_i состоянии и i_i ситуации. Предположим, что $S*$ – это комбинация всех состояний S и ситуаций I. Таким образом, марковский процесс принимает вид $\{S*, A, R, T\}$. Здесь A – конечное количество действий, R – функция вознаграждения, получаемого при переходе из одного состояния в другое, T – функция, которая возвращает вероятность, что система может измениться, если совершить определенное действие. Такой вид использования рассматриваемого понятия дает возможность применять его, когда создается программное обеспечение, способное к самостоятельному приспосабливанию. Определяя ситуацию, системные инженеры могут создавать проекты хранилища системных политик, внося в него новые определения, соблюдая стандарты, которым должны отвечать программные системы, обладающие способностью приспосабливаться самостоятельно. Кроме этого, появляется возможность привести программные системы к единой форме в глобальных масштабах.

Что касается самого метода самостоятельной адаптации, то он заключается в том, что создается модель операционной среды. Данный процесс состоит из последовательных шагов:

1) определяется реальное состояние рассматриваемой системы, действий, а также ситуации;

2) контекст детально анализируется, после чего формируется модель неопределенностей;

3) все неопределенности отражаются в действиях;

4) реализуются переходы между состояниями системы.

В процессе формирования программного обеспечения, которое может самостоятельно приспосабливаться к среде, важной задачей является определение реального состояния и действий. Именно на этом этапе устанавливаются цели и задачи, которые нужно решить, чтобы достигнуть поставленных целей. Суть состояния заключается в совокупности условий, имеющих в работе системы особую важность. Функцию или процесс, которые изменяют состояние системы, называют действиями. Когда происходит определенное событие, при котором начинается процесс приспосабливания агента, это будет ситуация. Результаты шага являются основанием, чтобы создать начальную модель изменения состояния.

Вторая стадия – это анализ реализации системы. Он проводится с целью создания модели неопределенностей, влияние которых может изменить работу программной системы. Чтобы исключить негативные моменты в работе системы в будущем, важно учесть каждый фактор на данном этапе моделирования.

В рамках заключительного этапа сопоставляются найденные неопределенности и системы для получения вероятностного распределения по модели. Чтобы сформировать модели распределения вероятностей или чтобы у специалистов был большой опыт в создании подобных моделей, требуется про-

ведение глубокого анализа. Изучить некоторые факторы, по которым нет 100 % уверенности, можно уже после запуска системы, когда она будет эксплуатироваться.

Использовать этот метод можно, когда сайт размещается в сети с помощью облачных услуг. Режимов работы может быть два: графический и текстовый. В первом случае пользователи получают больше данных, но на него требуется больше ресурсов. Все сайты создаются с одной целью – обеспечить пользователя возможностью получать актуальные данные, при этом задержки времени должны быть минимальные. Владельцы сайтов стремятся снизить затраты на услуги облачных сервисов, сводя к минимуму количество ресурсов, которое в некоторые периоды является чрезмерным.

Таким образом, сайт представляет собой программную систему, которая имеет способность самостоятельного приспосабливания. Когда планируется поведение ресурса, самым важным фактором является именно скорость отклика. Результаты анализа поставленной задачи показали, что возможны два варианта сценария:

1) использовать текстовый режим, что позволит тратить меньше времени на вычисления;

2) добавить ресурсы для повышения производительности.

При ситуации, когда трафик уменьшился в размерах, возможны сценарии:

1) изменить режим на графический, чтобы пользователь смог получить больше информации;

2) сократить неиспользуемые на данный момент ресурсы.

Для выполнения поставленной задачи, а также с учетом специфических особенностей предметной области были выведены такие состояния системы:

1) Режим = {Графический, Текстовый};

2) Время Отклика = {Нормальное, Низкое};

3) Ресурсы = {Мало, Средне, Много}.

Изменение состояния возможно такими способами:

1) добавление ресурса или освобождение;

2) переход в один из режимов: графический или текстовый.

Вознаграждение, как правило, определяется временем, в течение которого сайт даст отклик. Но и другие факторы влияют на него: рабочий режим (лучше графический), количество используемых ресурсов (чем меньше, тем лучше).

Далее создается модель неопределенности. Ориентиром является предмет. Для этого сначала проводится экспертный анализ. Чтобы максимально упростить процесс в данной ситуации, используется одна модель неопределенности, суть которой в том, что облачный ресурс является доступным. Количество ресурсов ограничено, поэтому можно предположить, что провайдер не сможет постоянно поставлять их в достаточном объеме. Здесь вероятность будет 2 %. Возможны такие последствия: состояние системы изменяется с задержкой, происходят незапланированные изменения. Авторы сопоставили эту неопределенность и действия системы. Результаты показали, что имеется связь данной неопределенности и действий, когда ресурсы добавляются. Отсюда следует связь вероятностного распределения и переходов из одного состояния в другое, когда реализуется данное действие. Внутренняя неопределенность во внимание не принимается, а вероятность возможных изменений состояний принята за 1 %, чтобы было проще.

Использование предложенного метода дает возможность системе осуществлять выбор нужного поведения в зависимости от времени отклика в разных условиях.

Заключение

Программное обеспечение характеризуется высокой сложностью, в этой связи трудно спрогнозировать, как поведет себя система в разных внешних условиях, не всегда достаточно свободного времени для этого. В одних ситуациях производительность системы может быть отличной, при других – недостаточной. А чтобы разработать сложную программную систему, которая будет показывать отличную производительность в разных средах (в том числе в тех, которые задает пользователь) и при разных условиях, требуются большие затраты времени и ресурсов.

В решении данной проблемы помогут специализированные методы, которые обеспечивают самостоятельную адаптацию программного обеспечения. Сегодня к его разработке применяются разные подходы, но ни один из них не учитывает в процессе самоадаптации всю внешнюю и внутреннюю информацию, которую использует в своей работе система. Чтобы решить данную задачу, авторы предлагают две концепции самостоятельной адаптации программных систем: рефлексивную и на основе машинного обучения. Данные концепции являются основой для разработки методов самоадаптации. При комплексном применении данных методов в значительной степени расширяются возможности для самостоятельного приспособления, работа программного обеспечения становится более эффективной, значительно сокращается количество ресурсов на сопровождение и развитие. В основе методики интеллектуального анализа информации лежит предположение о том, что имеются взаимозависимости в предметной области системы, которые не выявлены. Данная методика предназначается для того, чтобы выявлять данные взаимосвязи. Изначально метод машинного обучения разрабатывали, чтобы выявлять количество аппаратных ресурсов, затраты которых для системы будут оптимальными в конкретной ситуации. Данный метод делает систему более производительной и надежной, но его использование сильно ограничено, когда нужно находить скрытые закономерности в предметной области.

Список литературы

1. Бождай А. С., Евсеева Ю. И. Метод рефлексивной самоадаптации программных систем // Известия высших учебных заведений. Поволжский регион. Технические науки. 2018. № 2. С. 74–86.
2. Бершадский А. М., Бождай А. С., Гудков А. А., Евсеева Ю. И. Математическая модель рефлексии самоадаптивных программных систем // Известия Волгоградского государственного технического университета. 2018. № 3. С. 7–14.
3. Бождай А. С., Артамонов Д. В., Евсеева Ю. И. Использование машинного обучения с подкреплением в создании самоадаптивного программного обеспечения // Известия высших учебных заведений. Поволжский регион. Технические науки. 2019. № 3. С. 58–68.
4. Ghezzi C., Salvaneschi G., Pradella M. ContextErlang: A language for distributed context-aware self-adaptive applications // Science of Computer Programming. Amsterdam : Elsevier, 2015. P. 20–43.
5. Lei Y., Ben K., He Z. A model driven agent-oriented self-adaptive software development method // 12th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD). New Jersey : IEEE, 2015. P. 255–265.
6. Li Y., Li L., Wang W., Wu T. ADAPT: An agent-based development toolkit and operation platform for self-adaptive systems. // IEEE Conference on Open Systems (ICOS). New Jersey : IEEE, 2017. P. 145–167.

7. Bencomo N., Belaggoun A. Supporting decision-making for self-adaptive systems: From goal models to dynamic decision network // International Working Conference on Requirements Engineering: Foundation for Software Quality. San Francisco : IEEE, 2013. P. 221–237.

References

1. Bozhday A.S., Evseeva Yu.I. Method of reflexive self-adaptation of software systems. *Izvestiya vysshikh uchebnykh zavedeniy. Povolzhskiy region. Tekhnicheskie nauki = Proceedings of universities. Volga region. Technical science.* 2018;(2):74–86. (In Russ.)
2. Bershadskiy A.M., Bozhday A.S., Gudkov A.A., Evseeva Yu.I. Mathematical model of reflection of self-adaptive software systems. *Izvestiya Volgogradskogo gosudarstvennogo tekhnicheskogo universiteta = Bulletin of the Volgograd State Technical University*. 2018;(3):7–14. (In Russ.)
3. Bozhday A.S., Artamonov D.V., Evseeva Yu.I. Use of machine learning with reinforcement in the creation of self-adaptive software. *Izvestiya vysshikh uchebnykh zavedeniy. Povolzhskiy region. Tekhnicheskie nauki = Proceedings of universities. Volga region. Technical science*. 2019;(3):58–68. (In Russ.)
4. Ghezzi C., Salvaneschi G., Pradella M. ContextErlang: A language for distributed context-aware self-adaptive applications. *Science of Computer Programming*. Amsterdam: Elsevier, 2015:20–43.
5. Lei Y., Ben K., He Z. A model driven agent-oriented self-adaptive software development method. *12th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*. New Jersey: IEEE, 2015:255–265.
6. Li Y., Li L., Wang W., Wu T. ADAPT: An agent-based development toolkit and operation platform for self-adaptive systems. *IEEE Conference on Open Systems (ICOS).* New Jersey: IEEE, 2017:145–167.
7. Bencomo N., Belaggoun A. Supporting decision-making for self-adaptive systems: From goal models to dynamic decision network. *International Working Conference on Requirements Engineering: Foundation for Software Quality*. San Francisco: IEEE, 2013:221–237.

Информация об авторах / Information about the authors

Александр Сергеевич Бождай
доктор технических наук, доцент,
профессор кафедры систем
автоматизированного проектирования,
Пензенский государственный университет
(Россия, г. Пенза, ул. Красная, 40)
E-mail: bozhday@yandex.ru

Aleksandr S. Bozhday
Doctor of technical sciences,
associate professor,
professor of the sub-department
of computer aided design,
Penza State University
(40 Krasnaya street, Penza, Russia)

Юлия Игоревна Евсеева
кандидат технических наук,
доцент кафедры систем
автоматизированного проектирования,
Пензенский государственный университет
(Россия, г. Пенза, ул. Красная, 40)
E-mail: shymoda@mail.ru

Yuliya I. Evseeva
Candidate of technical sciences,
associate professor of the sub-department
of computer aided design,
Penza State University
(40 Krasnaya street, Penza, Russia)

Алексей Анатольевич Гудков
кандидат технических наук, доцент,
доцент кафедры систем
автоматизированного проектирования,
Пензенский государственный университет
(Россия, г. Пенза, ул. Красная, 40)
E-mail: alexei-ag@yandex.ru

Aleksey A. Gudkov
Candidate of technical sciences,
associate professor,
associate professor of the sub-department
of computer aided design,
Penza State University
(40 Krasnaya street, Penza, Russia)