5. Kureychik V. M. Algoritmy odnomernoj upakovkij elementov [Algoritms of 1-D elements packing], *Izvestiya JuFU. Seriya. Tehnicheskie nauki* [Izvestiya SFedU. Series. Engineering sciences], 2013, no.7 (144), pp. 8 – 11.

6. Kurosh A.G. Kurs vysshey algebry [The course of higher algebra]. Moscow, Science Publ., 1968. 431 p.

7. Meyer B., Baudoin C. *Metody programmirovaniya* [Programming methods]. In 2 vol. Moscow, Mir Publ., 1982. Vol. 2. 368 p.

8. KaluznyBohdan L., Shaw R. H. A. David. Optimal aircraft load balancing. *International Transactions in Operational Research*, November 2009, Vol. 16, no. 6, pp. 767–787 (21). DOI: https://doi.org/10.1111/ j.1475-3995.2009.00723.x

9. Mongeau M., Be`s C., Optimization of aircraft container loading. *IEEE Transactionson Aerospace and Electronic Systems*, 2003, vol. 39, no. 1, pp. 140–150.

10. Vancroonenburg W., Verstichel J., Tavernier K., VandenBerghe G. Automatic air cargo selection and weight balancing: A mixed integer programming approach. *Transportation Research. Part E: Logistics and Transportation Review*, May 2014, Vol. 65, pp. 70–83. DOI: https://doi.org/10.1016/j.tre.2013.12.013.

УДК 004.94

ВЛИЯНИЕ ФОРМЫ ПРЕДСТАВЛЕНИЯ ЭЛЕМЕНТОВ ТЕХНИЧЕСКОЙ СИСТЕМЫ В ПРОГРАММНОМ ОБЕСПЕЧЕНИИ НА СЛОЖНОСТЬ ЕГО АЛГОРИТМИЧЕСКОГО ОБЕСПЕЧЕНИЯ

Статья поступила в редакцию 12.01.2019, в окончательном варианте – 10.02.2019.

Алексеев Сергей Юрьевич, Тамбовский государственный технический университет, 392000, Российская Федерация, г. Тамбов, ул. Советская, 106,

кандидат технических наук, научный сотрудник, ORCID https://orcid.org/0000-0001-8748-5116, e-mail: sergej.alexejew@gmail.com

В работе исследовано влияние формы представления программных абстракций на сложность алгоритмов работы программного обеспечения. Выполнено сравнение двух форм представления абстракций с точки зрения их влияния на сложность алгоритмов работы программного обеспечения. Использованы формальные методы оценки двух вариантов структуры программного обеспечения, в основу которых положено использование метрик. Одна форма выполнена в соответствии с классическими представлениями, регламентируемыми объектно-ориентированными методами, вторая – на основе принципа, согласно которому абстракции должны быть выполнены максимально приближенными к терминам предметной области. Реализация этого принципа достигается за счет сочетания объектно- и проблемно-ориентированных методов конструирования программного обеспечения. В результате получается вариант структуры, который на первый взгляд не очевиден с точки зрения объектно-ориентированных техник, но очень близок к специфике предметной области и обладает преимуществом с точки зрения статических и динамических характеристик работы программного обеспечения. Программное обеспечение, составленное из абстракций, представляет собой программную систему, сценарий работы которой в общем случае складывается из сценариев работы каждой абстракции и сценария их взаимодействия. Чаще всего сценарий оперирует экземплярами сложных структур данных. Эти структуры данных и их экземпляры, реализуя различные формы представления данных, описывают элементы предметной области. От формы представления данных, формы и уровня абстрагирования при представлении в программной системе элементов прикладной области зависят статические и динамические характеристики алгоритмов ее работы, которые оперируют этими представлениями. Исследование выполнено на примере программного обеспечения для проектирования деревянных конструкций. Этот пример является частным, но он демонстрирует общие закономерности использования абстракций в сложных сценариях.

Ключевые слова: информационные технологии, повышение эффективности вычислений, программная абстракция технической системы, проектирование технической системы, метрики программного обеспечения, уровень сложности, надежность разработки

INFLUENCE OF THE FORM OF REPRESENTATION OF THE ELEMENTS OF THE TECHNICAL SYSTEM IN THE SOFTWARE ON THE DIFFICULTY OF ITS ALGORITHMIC ENVIRONMENT

The article was received by editorial board on 12.01.2019, in the final version – 10.02.2019.

Alekseev Sergey Yu., Tambov State Technical University, 106 Sovetskaya St., Tambov, 392000, Russian Federation,

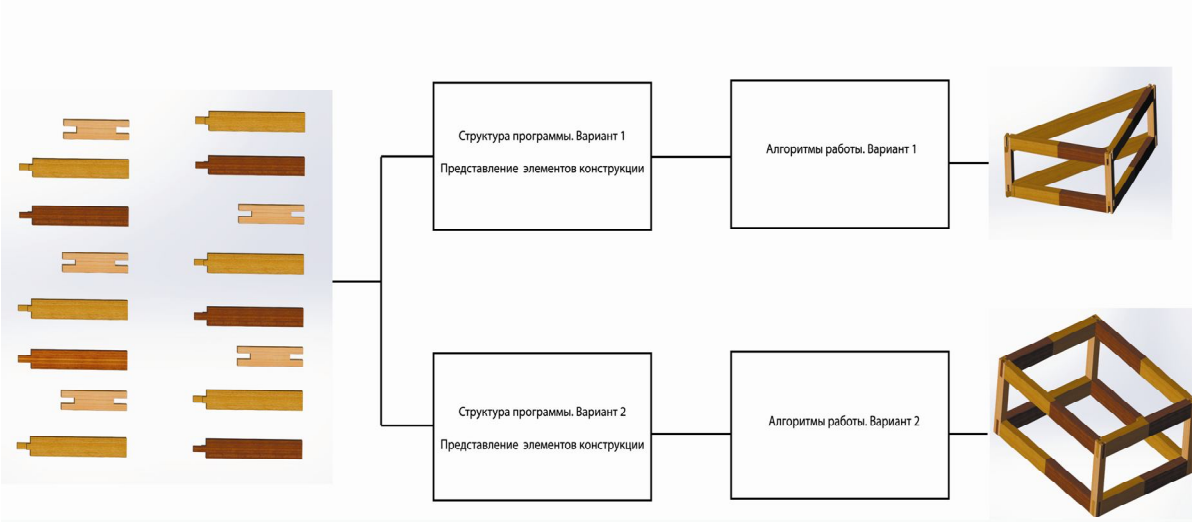
Cand. Sci. (Engineering), Researcher, ORCID https://orcid.org/0000-0001-8748-5116, e-mail: sergej.alexejew@gmail.com

The paper studies the influence of the form of representation of software abstractions on the complexity of software algorithms. A comparison of two forms of representation of abstractions from the point of view of their influence on the complexity of software algorithms has been performed. One form is made in accordance with the classical ideas regu-

lated by object-oriented methods, the second one is based on the principle that abstractions should be made as close as possible to the terms of the subject area. We used formal methods for evaluating two variants of the software structure, based on the use of metrics. The first version operates with abstractions designed in the traditional way. The second option is based on a combination of object- and problem-oriented software design techniques. The result is a variant of the structure, which at first glance is not obvious from the point of view of object-oriented techniques, but is very close to the specifics of the subject area and has the advantage from the point of view of the static and dynamic characteristics of the software. Software composed of abstractions is a software system, the script of which in general consists of the scenarios of each abstraction and the script of their interaction. Most often, the script operates with instances of complex data structures. These data structures and their instances, implementing various forms of data representation, describe the elements of the domain. The static and dynamic characteristics of the algorithms of its operation, which operate on these representations, depend on the form of data presentation, form and level of abstraction when presenting elements of the applied area in the software system. The study was performed on the example of software for the design of wooden structures. This example is private, but it demonstrates the general patterns in the use of abstractions in complex scenarios.

Key words: information technologies, increase of computing efficiency, software abstraction of a technical system, design of a technical system, software metrics, level of complexity, reliability of development

Graphical annotation (Графическая аннотация)



Введение. В настоящее время наблюдается тенденция усложнения современных технических систем. Одновременно растет и сложность программного обеспечения, которое используется при их проектировании и эксплуатации [1, 3, 4, 8,].

Исследования в области компьютерных наук развиваются в направлениях новых языков программирования, вычислительных технологий реального времени, методов представления информации, встроенных системных архитектур, системного программного обеспечения, обеспечения его надежности, повышения эффективности процессов разработки компьютерных систем. Одной из областей проводимых исследований является интеграция полученного нового знания в инженерные дисциплины, изучающие законы проектирования и функционирования технических систем. Использование вычислительных устройств для технических систем расширяет спектр решаемых ими задач и повышает качество их функционирования.

Сложность программного обеспечения снижается при его разделении на автономные компоненты [2, 7]. Эффективность их использования в программном обеспечении, предназначенном для технических систем, может быть повышена, если они будут реализованы как абстракции элементов для таких систем. Объектно-ориентированные методы определяют способ организации вычислений как взаимодействие объектов. За свою более чем сорокалетнюю историю теория этих методов хорошо изучена, подробно описана в литературе и развита. Они позволяют определять структурные программные аналоги технических систем на основе определения множества объектов, составляющих программную систему, множества возможных состояний каждого из них, взаимных отношений объектов и их классов. При этом сама программная система становится структурной абстракцией технической системы. Дальше появляется возможность определить алгоритмы работы программного обеспечения как абстракции действий, выполняемых над элементами технической системы.

Разные способы задания абстракций элементов технической системы позволяют повысить или снизить уровень сложности программного обеспечения. Они могут быть сформированы так, что технические детали компьютерной реализации не скрываются, а наоборот, проявляются и добавляются новые. При этом степень абстрагирования алгоритмов работы программного обеспечения снижается, они

перестают быть представлением действия, осуществляемого над технической системой. В нем появляется дополнительный код и логика, направленные на обеспечение работы с абстракциями. Объем этого побочного кода может варьироваться в зависимости от способа представления абстракции.

Все рассматриваемые в литературе методы, модели и типовые решения построения программных систем на основе объектов характеризуются своей универсальностью. Они или рассматривают отдельные фрагменты задачи, предлагая типовые решения для реализации фрагмента, или рассматривают широкий спектр задач на конкретных примерах. Рассмотрение задачи построения программных систем с этих точек эффективно с точки зрения удешевления процесса разработки. С точки зрения эксплуатационных характеристик программных систем, универсальность происходит в ущерб учету специфик особенностей предметной области и, как следствие – увеличению количества программных элементов в системе, усложнению их взаимодействия, повышению ресурсоемкости программной системы.

Формулировка задачи исследования. В общем случае реализация программных абстракций на основе вычислительных моделей сервисов, агентов и акторов позволяет представить программную систему как структурный и динамический аналог технической системы. Пример такого представления показан на рисунке 1.

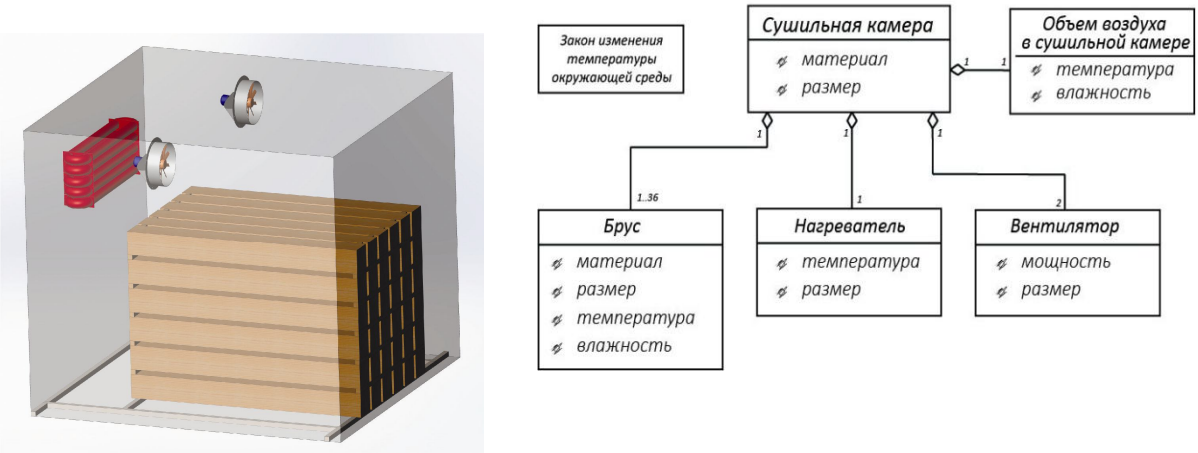


Рисунок 1 – Построение программной системы на основе абстракций технической системы (на примере сушильной камеры для пиломатериалов)

Можно определить три основных класса задач, связанных с техническими системами, эффективность решения которых повышается при использовании программного обеспечения. Это задачи проектирования технических систем, задачи моделирования и задачи дополнения. В задачах моделирования техническая система заменяется программным обеспечением, все функции технической системы, удовлетворяющие принятой системе допущений, реализуются в программном обеспечении. В задачах дополнения техническая система дополняется программным обеспечением, набор функций технической системы при этом расширяется или при неизменном их наборе повышается качество работы технической системы.

В задачах моделирования и дополнения поведение программной системы изменяется соответственно состоянию технической системы. Особенностью ее работы при этом является отсутствие заранее известного сценария поведения. Он является следствием текущего состояния программной системы, которое складывается из состояния абстракций, их поведения и взаимодействия. Полностью сценарий работы становится точно определенным только после того, как программная система закончила свою работу. Это свойство может быть использовано в исследовательских целях для изучения динамики поведения технических систем на их программных аналогах. Таким образом, можно рассматривать процесс работы программной системы, выполненной в виде абстракции технической системы, как моделирование ее работы.

Также программное обеспечение для проектирования может быть создано на основе абстракций, как программная система. В задачах проектирования выбирается структура технической системы, количественные и качественные характеристики ее подсистем, структура взаимосвязей этих подсистем. При условии, что есть возможность построить программную абстракцию технической системы, генерация ее вариантов и оценка их эффективности может осуществляться на программной абстракции. В этом случае можно сказать, что над абстракциями осуществляются действия в соответствии с определенным, заранее известным сценарием.

Сценарии задаются в виде последовательности действий и могут быть выражены на любом языке программирования. При увеличении доли абстракций при описании сценария, различия между языками программирования стираются. Сценарий, представленный на мощном универсальном языке

программирования по простоте восприятия, приближается к сценарию, представленному на специализированном языке. В нем скрыты детали, связанные с реализацией, а описание действий осуществляется в терминах прикладной области. Такой сценарий можно рассматривать как абстракцию процесса, который осуществляется над элементами технической системы (рис. 2).

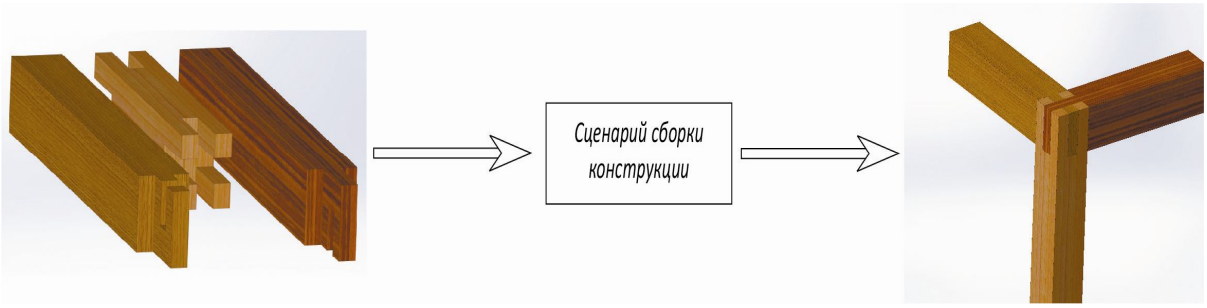


Рисунок 2 – Пример использования сценария (здесь сценарий представлен как абстракция процесса сборки конструкции)

Когда программная система моделирует функционирование и динамику технической системы, то способ представления абстракций ее элементов влияет на количество взаимосвязей между ними и количество передаваемых данных. Если моделируются действия над технической системой по заранее известному сценарию, то способ представления абстракции влияет на размер кода сценария, его логическую сложность.

Результаты исследования способов представления абстракций позволят повысить точность моделирования структуры и динамики технической системы, выражать сценарии с минимумом технических подробностей реализации в терминах прикладной области.

Решение проблемы. Исследование выполнено на примере программного обеспечения для проектирования деревянных конструкций. Этот пример является частным, но он демонстрирует общие закономерности использования абстракций в сложных сценариях. В качестве технической системы рассматривается проектируемая деревянная конструкция. Решение задачи проектирования связано с выбором размеров элементов конструкции, типов их соединения и расчетов параметров этих соединений. Конечной целью проектирования является техническая документация на деревянную конструкцию, включающая в себя перечень всех составляющих ее деталей с указанием размеров и последовательность сборки конструкции.

Существует большое количество промышленных программных продуктов для проектирования деревянных конструкций. Все они характеризуются своей универсальностью и возможностью затруднений при учете особенностей конкретных производств.

ArchiCad – программный пакет для архитекторов, основанный на технологии информационного моделирования (*Building Information Modeling – BIM*). Предназначен для проектирования и расчета архитектурно-строительных конструкций, а также создания элементов ландшафта, мебели и т.п. [9].

WoodCon – программа для проектирования деревянных зданий и крыш. Она предназначена для архитекторов, проектировщиков, инженеров и строительных предприятий. Программа позволяет создавать сложные деревянные конструкции стен и крыш, визуализировать их в виде трехмерных моделей, выводить список необходимого пиломатериала и т.д. Программа *WoodCon* состоит из нескольких модулей: «Стены», «Крыши», «Вывод данных и расчет» [10].

ArCon Eleco Small Business – это универсальный инструмент для создания архитектурного и дизайнерского проекта строительного объекта любой сложности. Программа создана как для профессиональных архитекторов и дизайнеров, так и для частных застройщиков, не являющихся профессионалами в области дизайна. Дополнительно к программе может быть подключен модуль расчета израсходованного пиломатериала «*ArCon*-Разбревновка» [11].

Программы для проектирования зданий и сооружений *RSTAB* и *RFEM* предназначены для расчета и проектирования таких деревянных конструкций, как балки, неразрезные балки, рамы, решетчатые фермы, клеёные балки, деревянные сэндвич-панели, деревянные панельные стены, деревянные каркасные дома, мосты, башни и др. [12].

LogHouse компания разрабатывает программы для «разбревновки» при производстве срубов. Программа позволяет спроектировать сруб «с нуля» и использовать полученные результаты для работы на автоматизированных линиях *Makron* и станках с ЧПУ [13].

CadWork является ведущей программной системой cad/cam в области деревянного строительства и столярного дела. Она имеет модульную структуру, что обеспечивает быструю настройку функциональ-

ности под требования конкретных производств. В своих специализированных модулях она предлагает комплексные программные решения на всех этапах – от планировки до производства. Она обеспечивает возможности планирования, произвольного проектирования, формирование деревянных конструкций различного назначения и различного уровня сложности, например – стропильные фермы, развертки крыш и другие деревянные конструкции. Отличительной особенностью этой программной системы является ее совместимость с производственным оборудованием. Проектирование конструкции выполняется с учетом особенностей того деревообрабатывающего станка, на котором планируется ее изготовление [14].

Исследуемое в данной работе программное обеспечение рассматривает ограниченный спектр типов деревянных конструкций и состоит из трех основных модулей – составления плана конструкции, расчета ее элементов и построения деталировки. На всех трех этапах программная система рассматривает и моделирует действия, выполняемые над деревянной конструкцией. Ее элементы в результате действий претерпевают качественные и количественные изменения. К качественным изменениям можно отнести выбор типа бруса и типа соединения брусьев, к количественным – их размеры.

Модуль составления плана выполнен на основе абстракций элементов проектируемой деревянной конструкции и в терминах тех действий, которые выполняет проектировщик на этой стадии. Например, это могут быть элементы оконных рам, дверей, лестниц, мебели, перемещение элементов, добавление новых элементов на план, определение взаимного положения элементов в пространстве и т.д. Вследствие того, что внутренняя логика модуля выполнена в терминах абстракций проектируемой конструкции и процесса построения ее плана, пользователю автоматически обеспечивается возможность при работе с программой оперировать элементами проектируемой конструкции. Обычно при использовании описанных в литературе методов конструирования абстракций требуется больше вычислительных ресурсов для перевода внутреннего представления программного объекта в представление для пользователя.

Модуль расчета элементов конструкции выполнен на основе абстракций деталей, составляющих проектируемую конструкцию. Это брус различных типов сечения и размеров, содержащий фигурные вырезы для взаимного соединения с другими брусьями или реализации полезных сервисных функций. Это могут быть, например, вырезы на крайних верхних и нижних брусьях для размещения дверных и оконных проемов. В модуле сосредоточена основная смысловая нагрузка и логика автоматизированного проектирования деревянных конструкций. Он выполняет анализ составленного плана – взаимного расположения элементов, их пересечений и генерирует пространственную деревянную конструкцию на основе брусьев. При этом подбираются типы сечений, размеры бруса и определяются вырезы на нем так, чтобы при соединении брусьев получалась запланированная конструкция. При выборе способов соединений брусьев предпочтение отдается клеевым соединениям.

Модуль построения деталировки выполнен на основе того же типа абстракций, что и модуль расчета элементов конструкции. Он выполняет работу по поиску уникальных элементов из бруса и составления на их основе списка деталей. В нем указываются размеры детали и количество, которое необходимо изготовить для данной конструкции. Этот модуль ответственен за проверку возможности изготовления деталей на деревообрабатывающем станке. Например, если деталь получилась длиннее, чем может обработать станок, она на этом этапе разбивается на составляющие, размеры которых удовлетворяют требованиям станка.

Алгоритм работы процедуры проектирования, охватывающей все три описанные выше стадии, показан на рисунке 3. Рисунок выполнен так, чтобы на нем легко можно было определить ребра и узлы графа потоков управления для оценки значения метрик потоков данных и управления в рассматриваемой программной системе.

Процедура использует абстракции элементов проектируемой деревянной конструкции и ее деталей. Исследуемые варианты отличались способами описания вырезов на брусьях. Они реализовывались двумя способами.

1. Каждый вырез рассматривался как материальный объект и реализовывался как отдельная самостоятельная абстракция, обладающая своими характеристиками и поведением. Для представления каждого типа выреза реализовывалось описание соответствующего абстрактного типа данных. Брус рассматривался как контейнер, в котором размещались абстракции, описывающие вырезы. Привязка вырезов к брусу осуществлялась по относительным координатам, которые отсчитывались от начала бруса. Координаты являлись общим свойством, которое присутствовало у всех типов вырезов. Иными словами, для каждого типа данных, описывающего вырез, определялись поля, где хранились координаты.

2. Вырез не рассматривался как отдельный объект и реализовывался как абстракция технологической операции по его производству. Единственной абстракцией материального объекта в этом случае выступал сам брус, правда, более сложной конфигурации, чем в первом случае. Он описывался набором вершин и порядком их обхода, которые в совокупности образуют брус с нужными вырезами. Здесь по относительным координатам привязывается каждая вершина.

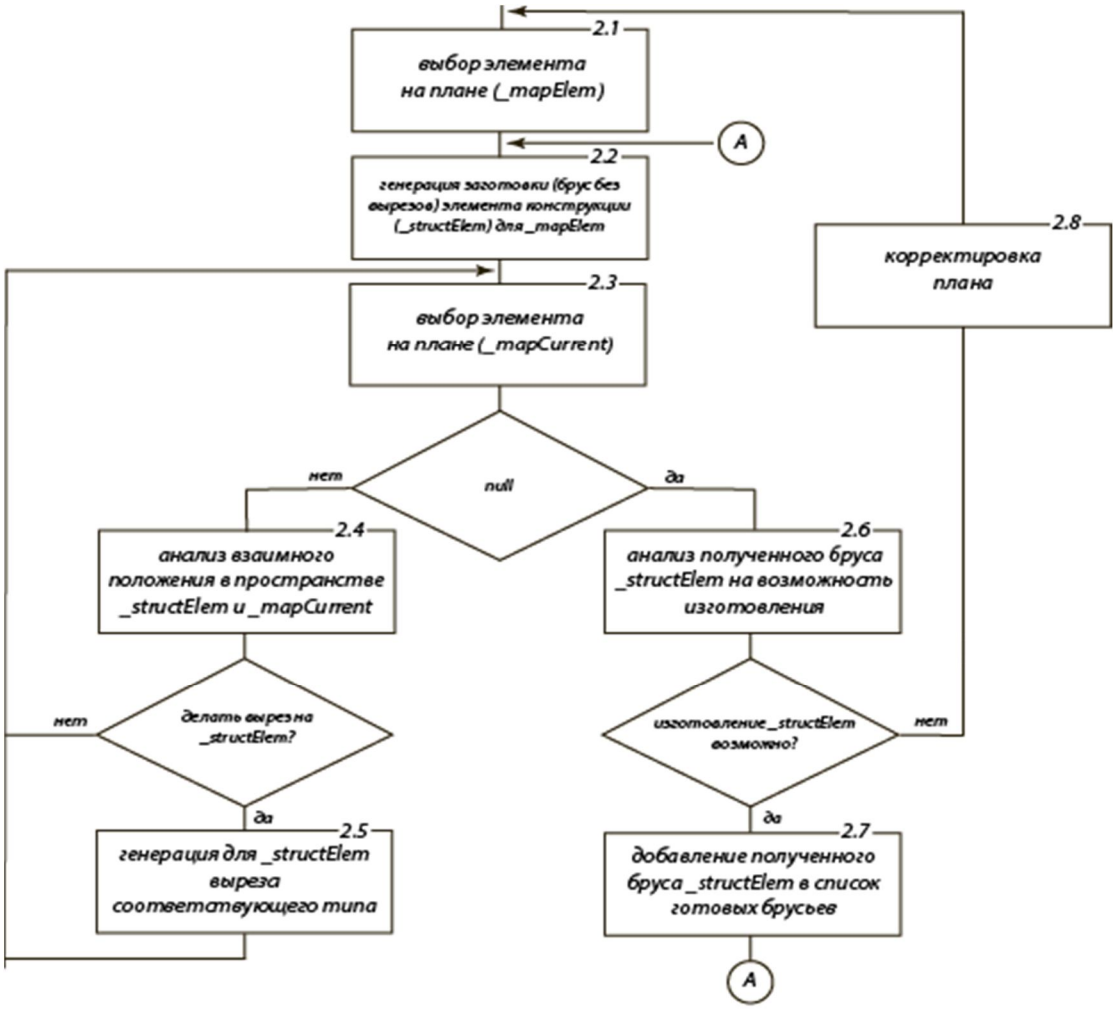


Рисунок 3 – Алгоритм работы процедуры расчета элементов конструкции

Рисунок 4 – Примеры типов соединений

Первый способ тривиальный и работоспособный. Он является примером классического использования декомпозиции программного кода на основе техник объектно-ориентированного проектирования. Здесь абстракциями представляются материальные сущности технической системы. Их статические и динамические отношения регламентируются принципами объектно-ориентированного проектирования.

Представление бруса во втором случае в большей степени соответствует физической картине процесса. В первом случае вырезы рассматриваются как самостоятельные объекты и соответствующие им абстракции могут существовать отдельно в контексте программы без абстракций бруса. Это противоречит принципам построения и функционирования рассматриваемой технической системы. Во вто-

ром случае структура и принципы функционирования технической системы смоделированы в программном обеспечении точнее. Здесь принципиально исключается возможность существования вырезов на брусе отдельно от самого бруса. Оценка корректности рассчитанных вырезов на брусе при этом может быть выполнена на этапе обхода контуров вырезов при их генерации для бруса. В первом случае для этого пришлось бы использовать дополнительную процедуру и рассматривать отношение двух абстракций. Это потребовало бы большего объема кода.

На основе результатов анализа двух вариантов организации абстракций можно сделать вывод, что изменение формы представления абстракций вырезов вызывает изменение структуры абстракций бруса, их взаимных отношений – статических и динамических.

Определение значений метрик сложности потоков данных и управления. Процедура проектирования, показанная на рисунке 1, была реализована на языке *C++*. Выбор языка обусловлен тем, что он минимально скрывает детали реализуемых операций. Следствием этого является возможность по сравнению с родственными языками, такими как *Java* и *C#,* однозначно разделять агрегацию и композицию.

В ходе реализации первого (начального) варианта процедуры, его отладки и тестирования было установлено, что в ходе процедуры необходима дополнительная реализация сервисных операций по инициализации абстракций вырезов, их сравнения, присваивания и копирования. Ввиду того, что в этом случае абстракции вырезов реализованы как отдельные объекты для выполнения перечисленных выше сервисных операций, необходимо всегда осуществлять предварительное определение типа выреза. В зависимости от него определяются способы выполнения перечисленных операций. Такая проверка добавляет к алгоритму, показанному на рисунке 3, количество условных операторов, равное количеству типов вырезов, которое рассматривается программным обеспечением в качестве потенциально возможных вариантов. При выборе типа узла в основной процедуре-анализаторе плана и пересечений стены присутствует оператор *switch ()*, который определяет действия при выборе того или иного узла как минимум для их конструирования с помощью оператора *new* (рис. 5). Организация балок в этом случае требует использования таких переключателей повторно, что значительно усложнит восприятие кода и сделает его в большей степени декларативным, а не описывающем алгоритм проектирования.

анализ пересечения брусьев (шаг 2.5 алгоритма на рис. 3)	*выбор действий в зависимости от типа выреза (шаги 2.6–2.8 алгоритма на рис. 3)*

```
type = determineIntersectionType (
        _structElem,
        _mapCurrent);

switch (type) {
        case`type_1`:
                _structElem.add (
                    new cutForType_1 (…)
                    );

        case`type_2"`:
                _structElem.add (
                    new cutForType_2 (…)
                    );
                      .
                      .
                      .
        case`type_N`:
                _structElem.add (
                    new cutForType_N (…)
                    );
        default:
        }
```

```
cutForType_1 * cut =
        dynamic_cast <cutForType_1 *> (
                currentCut);
if ( cut ) {
        doForType_1 (...);
        cut -> nonVirtualMethod_1 (...);
        }

cutForType_2 * cut =
        dynamic_cast <cutForType_2 *> (
                currentCut);
if ( cut ) {
        doForType_2 (...);
        cut -> nonVirtualMethod_2 (...);
        }
                .
                .
                .
cutForType_N * cut =
        dynamic_cast <cutForType_N *> (
                currentCut);
if ( cut ) {
        doForType_N (...);
        cut -> notVirtualMethod_N (...);
        }
```

Рисунок 5 – Наличие (использование) дополнительных операторов при использовании традиционных методов конструирования абстракций

Второй вариант свободен от этих недостатков по двум основным причинам: все вырезы представляются единым унифицированным способом; сокращено количество типов используемых абстракций. Абстракции технологических операций, отсутствующие в первом варианте, единовременно изменяют количественные и качественные характеристики абстракций бруса. Они не являются абстракциями материальных объектов, и для них отсутствует необходимость выполнять операции присваивания, инициализации и динамического определения их типа.

В рамках исследования выполнено сравнение вариантов реализации процедуры расчета элементов конструкций и построения деталировки. Она работает с абстракциями элементов проектируемой деревянной конструкции. Сравнение вариантов проведено с точки зрения метрик, характеризующих общую сложность кода процедуры, сложность потоков управления и данных.

Основной метрикой, характеризующей сложность потока управления, является метрика цикломатической сложности. В варианте процедур, в котором вырезы реализованы как отдельные объекты, значение метрики выше.

Для алгоритма на рисунке 1 была вычислена метрика цикломатической сложности [7]:

$$\mathrm{C}(G) = N - E + 2, \quad (1)$$

где N – количество дуг графа потоков управления процедуры; E – количество его узлов.

Значение метрики составило

$$\mathrm{C}_0(G) = 14 - 11 + 2 = 5. \quad (2)$$

При добавлении в процедуры к шагам 2.5–2.7 дополнительных условий, связанных с определением типов вырезов и выбора действия в зависимости от определенного типа, значение метрики возрастает пропорционально их количеству:

$$\mathrm{C}(G) = (5 + 2 \times k \times n) - (4 + k \times n) + 2 = \mathrm{C}_0(G) + k \times n, \quad (3)$$

где $n = 3$ – количество шагов алгоритма, куда добавляются операторы условия, связанные с обработкой вырезов; k – количество типов вырезов, которые обрабатывает ПО.

Способ описания вырезов на брусе, когда они представлены как отдельные объекты, характерен сложностями сопровождения программного обеспечения. Для расширения списка пересечений брусьев, которые могут быть обработаны, необходимо изменять большое количество строк кода в разных местах программы.

Добавление дополнительных условных операторов может быть скомпенсировано в специализированных процедурах. Сложность потоков управления в процедуре при этом будет снижена. Однако в любом случае в программе появится дополнительный сервисный код, который не несет в себе полезной вычислительной нагрузки, направленной на решение задачи.

При рассмотрении вырезов как отдельных абстракций характеристики каждого из них реализуются отдельно, в рамках объекта, состояние которого они характеризуют. Это сокращает размеры пространства действия переменных, и, соответственно, значения метрик этого класса в рамках отдельно взятой абстракции. Рассмотрение абстракций при их взаимодействии показывает, что сложность потоков данных не уменьшается. Причина – потоки данных внутри абстракций дополняются потоками данных для обеспечения обмена информацией между абстракциями.

Наглядно это может быть проиллюстрировано на примере метрики Чепина. Эта метрика наиболее распространена [5, 6] среди класса метрик, используемых для оценки информационной прочности фрагментов программного кода на основе результатов анализа характера использования переменных. Оценки, полученные на ее основе, однозначны. Для отдельно взятой абстракции она может быть записана так:

$$Q = a_1 \times P + a_2 \times M + a_3 \times C + a_4 \times T, \quad (4)$$

где P – это множество переменных, содержащих исходную информацию для расчетов и их результат; M – множество модифицируемых или создаваемых внутри программы переменных; C – множество переменных, участвующих в управлении работой программного модуля; T – множество переменных, не используемых в данной программе (паразитные переменные); a_1, a_2, a_3, a_4 – весовые коэффициенты, используемые для отражения различного влияния на сложность программы каждой функциональной группы, обычно принимаются по умолчанию $a_1 = 1$, $a_2 = 2$, $a_3 = 3$, $a_4 = 0{,}5$.

Для набора абстракций метрика может быть записана в виде:

$$\overline{Q} = \sum_{i=1}^{n}(a_1 \times P_i + a_2 \times M_i + a_3 \times C_i + a_4 \times T_i), \quad (5)$$

где n – количество взаимодействующих абстракций; P_i, M_i, C_i, T_i – множества переменных, характеризующих состояние i-й абстракции.

При том, что часть сервисной логики реализуется на верхнем уровне, необходимо рассматривать как переменные, содержащиеся внутри абстракций, так и переменные, представляющие экземпляры самих абстракций. Обращение к переменным, описывающим свойства абстракций и расположенным внутри них, может осуществляться двумя способами – через методы доступа или через передачу сообщений с параметрами. Количество параметров сообщения в большинстве случаев соответствует количеству всех переменных, описывающих свойства абстракции. На рисунке 6 представлены способы обращения к переменным абстракции.

обращение через методы доступа	*обращение через передачу сообщений с параметрами*

`object.setVar_1 (sourceVar_1)`
`object.setVar_2 (sourceVar_2)`
`object.setVar_3 (sourceVar_3)`
`.`
`.`
`.`
`object.setVar_<n-1> (sourceVar_<n-1>)`
`object.setVar_n (sourceVar_n)`
 	
`object.do ()`
	`object.init (sourceVar_1,`
	`sourceVar_2, sourceVar_3,`
	`…,`
	`sourceVar_<n-1>, sourceVar_<n>);`
	
	`object.do (sourceVar_1,`
	`sourceVar_2, sourceVar_3,`
	`…,`
	`sourceVar_<n-1>, sourceVar_<n>);`

Рисунок 6 – Способы обращения к переменным абстракции *(в одном случае составляющая P метрики Чепина равна количеству переменных, в другом – это значение принимает составляющая M)*

Представленные на рисунке 6 способы полностью находятся в рамках принципов разделения программного обеспечения на компоненты и сокрытия деталей реализации этих компонентов.

Для взаимодействующих абстракций метрика Чепина модифицируется к виду:

$$Q = \overline{Q} + a_1 \times P + a_2 \times M + a_3 \times C + a_4 \times T, \quad (6)$$

где $\overline{Q}$ – значение метрики для набора разрозненных абстракций; P, M, C, T – множества взаимодействующих в рамках сценария и объединяемых им абстракций.

Для пяти типов вырезов в брусьях, показанных на рисунке 4, общая метрика Чепина будет составлять:

$$Q_0 = (a_1 \times P + a_2 \times 6 + a_3 \times 2 + a_4 \times T) + (a_1 \times P + a_2 \times 5 + a_3 \times 3 + a_4 \times T) + \\ +(a_1 \times P + a_2 \times 5 + a_3 \times 3 + a_4 \times T) + (a_1 \times P + a_2 \times 1 + a_3 \times 3 + a_4 \times T) + \\ +(a_1 \times P + a_2 \times 2 + a_3 \times 5 + a_4 \times T) = 5P + 2.5T + 71 \quad (7)$$

Каждый вырез описывается уникальным набором переменных, содержащим различное их количество, в зависимости от сложности выреза определяются переменные, регламентирующие потоки управления в функциях отдельных абстракций. Для показанных выше на рисунке вырезов определены следующие значения *M* и *C*: *{{6, 2}, {5, 3}, {5, 3}, {1, 3}, {2 ,5}}*.

При выделении вырезов в отдельные абстракции метрика сокращается в 5 раз. Это говорит о том, что когда абстракции разрабатываются отдельно, сложность разработки программного кода заметно снижается. В рассматриваемом случае, когда часть сервисных операций выполняется в коде сценария, при взаимодействии абстракций, метрика составит:

$$Q = Q_0 + \sum_{i=1}^{n}(P + 2 \times M + 3 \times C + 0{,}5 \times T) = Q_0 + 2 \times n, \quad (8)$$

где $n = 3$ – количество шагов алгоритма (представленного на рисунке 1), которые выполняют манипуляции над абстракциями вырезов и бруса, на котором они вырезаются.

Алгоритм построен так, чтобы на каждом его шаге обрабатывалось только одно соединение брусьев. Поэтому для его шагов можно рассматривать две взаимодействующие абстракции – брус и вырезаемый на нем один замок. При этом для каждого шага принимается, что $P = 1$, $M = 2$, $C = 1$, $T = 0$. Другие варианты построения алгоритма, когда единовременно обрабатываются и взаимодействуют больше чем две абстракции, характеризуются большими значениями метрики Чепина.

Таким образом, в рассматриваемом случае использование техник построения автономных компонентов программного обеспечения на основе традиционных методов объектно-ориентированного проектирования не уменьшает сложность потоков данных. Отдельно уменьшаются значения метрик для каждой абстракции. Отдельно они неработоспособны. Согласно определению системы, ее поведение и результат достигается только при взаимодействии составляющих ее элементов. Общее значение метрики для абстракций и процедуры, описывающей их взаимодействие, не уменьшается.

Дополнительно может быть использована метрика спена [5, 6], показывающая число выражений программы, содержащих выделенный идентификатор между его первым и последним появлением в тексте. Она не информативна, так как по количеству вызовов переменной между ее первым и последним вызовом можно сделать вывод не только о сложности потоков данных, но и о степени связности абстракции. Поэтому для оценки рассматриваемых абстракций эта метрика модифицируется к виду:

$$S_m = \frac{S}{n}, \quad (9)$$

где S – количество выражений кода, содержащих идентификатор; n – количество строк кода, для которых определено значение S.

Значение S_m может быть использовано для оценки связности абстракции в большей степени чем, для оценки сложности потоков данных. Чем больше значение S_m для выделенной переменной, тем выше степень связности абстракции по ней.

Основному изменению подвергаются значения метрик, описывающих общую сложность программы, – количество строк кода *(SLOC – Source Lines Of Code*), среднее число строк для функций и абстракций, метрика Джилба, *ABC*-метрика [5, 6].

Общее количество строк кода в программной системе и средние их значения для перечисленных функций возрастают за счет появления дополнительных операторов присваивания, инициализации переменных состояния абстракции (рис. 2), логических операторов, направленных на определение типов вырезов.

Метрика Джилба показывает сложность программного обеспечения на основе насыщенности программы условными операторами и операторами цикла:

$$J = \frac{CL}{m}, \tag{10}$$

где CL – абсолютная сложность (количество управляющих операторов); m – общее число операторов программы. Для алгоритма, показанного на рисунке 3, значение метрики составит 4/7.

При добавлении к шагам 2.5–2.7 алгоритма, показанного на рисунке 3, дополнительных логических операторов, обеспечивающих определение анализ типа используемой в программе переменной, представляющую абстракцию выреза абстракции замка, (как показано на рисунке 3) типа выреза, эта метрика возрастает:

$$J = \frac{CL+k\times n}{m+k\times n} = \frac{3+k\times n}{11+k\times n}, \tag{11}$$

где $n = 3$ – количество шагов алгоритма, куда добавляются операторы условия, связанные с обработкой вырезов; k – количество типов вырезов, которые обрабатывает ПО.

При увеличении числителя и знаменателя выражения метрики на одинаковую величину значение метрики всегда возрастает:

$$\forall a, \frac{CL}{m} < \frac{CL+a}{m+a}, \tag{12}$$

Это происходит потому, что совокупное количество операторов цикла и условных операторов в программе всегда меньше общего количества операторов, т.е. значение числителя выражения метрики всегда меньше, чем значение знаменателя. Увеличение их на одинаковую величину приводит к росту значения общего выражения, так как числитель относительно знаменателя увеличивается больше в процентном соотношении.

ABC-метрика [5, 6] основана на подсчете присваиваний значений переменным, явных передач управления за пределы области видимости, т.е. вызовов функций, и логических проверок. Эта метрика позволяет учитывать большее количество случаев добавления операторов в текст сценария. Кроме того, для учета появляющихся в нем операторов условия можно учесть выполнение присваиваний и отправки дополнительных сообщений, показанных на рисунке 2. Метрика записывается тройкой значений – *<a, b, c>*, но для оценки сложности программы вычисляется одно число – как квадратный корень из суммы квадратов *a, b, c*:

$$F_0 = \sqrt{a^2 + b^2 + c^2}, \tag{13}$$

где a – количество присваиваний в блоке кода; b – количество передач управления за пределы области видимости; c – количество логических проверок.

При появлении в программе, реализующей алгоритм по рисунку 1, дополнительных логических операторов для определения типа выреза, операторов присваивания для инициализации абстракций вырезов или вызова дополнительных методов значение метрики возрастает:

$$F = \sqrt{(a + a_1)^2 + (b + b_1)^2 + (c + c_1)^2} = \sqrt{a^2 + b^2 + c^2 + A}, \tag{14}$$

при

$$A = (2 \cdot (a \cdot a_1 + b \cdot b_1 + c \cdot c_1) + a_1^2 + b_1^2 + c_1^2) > 0, \tag{15}$$

где $a_1 = p\times n$. $b_1 = n$, $c_1 = k\times n$ – количество дополнительно появившихся присваиваний, передач управления и логических проверок соответственно; p – количество переменных состояния абстракции; n – количество шагов алгоритма, куда добавляется код; k – количество обрабатываемых типов вырезов. В расчетах значения метрики принималось допущение о том, что для каждого шага сервисная инициализирующая функция с большим количеством параметров вызывается один раз. Все остальные функции абстракции вызываются во всех случаях.

Выше для всех трех классов метрик наглядно показано, что использование классических объектно-ориентированных техник может быть сопряжено с усложнением кода. Основным (и, может быть, самым главным недостатком) в этом случае является то, что в нем смешиваются сервисные функции и функции, которые направлены на непосредственное решение прикладной задачи. Это уводит человека,

разрабатывающего, модернизирующего или эксплуатирующего ПО, в сторону от образа мышления в терминах предметной области, разбавляя его техническими особенностями реализации. В свою очередь это делает невозможным написание сценария экспертом в предметной области, не обладающим специфическими знаниями процессов разработки ПО.

Выводы.

1. В работе показано возможное увеличение сложности программного обеспечения для технических систем при его декомпозиции на основе классических методов объектно-ориентированного программирования.

2. Построение программной системы как структурной и динамической абстракций технической системы позволяет снизить его общую сложность. Степень приближения структуры программной системы к структуре технической системы может быть увеличена при расширении набора типов используемых абстракций и методов их конструирования. Представление программы в терминах и абстракциях предметной области упрощает ее написание и сопровождение.

3. При построении программной системы как абстракции технической системы сокращается количество сервисных операторов, направленных на поддержание работы программной системы, а не на решение прикладных задач. Это одновременно повышает эксплуатационные характеристики программного обеспечения за счет снижения вычислительной нагрузки, приходящейся на выполнение сервисных операций.

4. При повышении уровня абстрагирования повышается связанность программного кода в отдельных абстракциях и в процедурах, которые связывают их между собой, реализуя сценарии манипуляций с ними, снижается зацепление между фрагментами программного кода.

5. Упрощение логики программной системы и сокращение сервисных операторов, повышение связности кода, снижение зацепления его фрагментов является комплексным показателем, повышающим надежность работы программной системы, определяемой уменьшением количества ошибок, допускаемых на этапах разработки программной системы и ее выполнения.

Библиографический список

1. Balaji B. Models, abstractions, and architectures: The missing links in cyber-physical systems / B. Balaji et al. // Proceedings of the 52nd Annual Design Automation Conference. – ACM, 2015. – P. 82.

2. Booch G. Object-oriented analysis and design with applications (Third Edition) / G. Booch. – Pearson Education Inc., 2007.

3. Bures T. Software abstractions for component interaction in the internet of things / T. Bures et al. // Computer. – 2016. – Vol. 49, № 12. – P. 50–59.

4. Hnetynka P. Software abstractions and architectures for smart cyber-physical systems: Keynote address / P. Hnetynka // Software Engineering Research, Management and Applications (SERA) : IEEE 15th International Conference on. – IEEE, 2017. – P. 3.

5. Kuznetsov M. A. Analysis of complexity metrics of a software code for obfuscating transformations of an executable code / M. A. Kuznetsov, V. O. Surkov // IOP Conference Series: Materials Science and Engineering. – IOP Publishing, 2016. – Vol. 155, № 1. – P. 012008.

6. Shudrak M. O. Improving fuzzing using software complexity metrics / M. O. Shudrak, V. V. Zolotarev // International Conference on Information Security and Cryptology. – Springer, Cham, 2015. – P. 246–261.

7. Sommerville I. Software Engineering. International computer science series / I. Sommerville. – Addison Wesley, 2004.

8. Zambonelli F. Key abstractions for iot-oriented software engineering / F. Zambonelli // IEEE Software. – 2017. – Vol. 34, № 1. – P. 38–45.

Программные средства

9. *ArchiCad* – программный пакет для архитекторов, основанный на технологии информационного моделирования. – Режим доступа: https://www.graphisoft.ru/archicad/, свободный. – Заглавие с экрана. – Яз. рус.

10. *WoodCon* – программа для проектирования деревянных зданий и крыш. – Режим доступа: http://www.arcon-eleco.ru/prog/woodcon/, свободный. – Заглавие с экрана. – Яз. рус.

11. *ArCon Eleco Small Business* – программа для создания архитектурного и дизайнерского проекта. – Режим доступа: http://www.arcon.ru/products/architectural-engineering-and-design/small-business/, свободный. – Заглавие с экрана. – Яз. рус.

12. *Dlubal* – программы для расчета и проектирования конструкций. – Режим доступа: https://www.dlubal.com/ru/produkty/dopolnitelnyje-moduli-rfem-i-rstab/, свободный. – Заглавие с экрана. – Яз. рус.

13. *LogHouse* – программное обеспечение для проектирования срубов. – Режим доступа: http://loghouse.pro/loghouse/, свободный. – Заглавие с экрана. – Яз. рус.

14. *CADWORK wood* – программа для деревянного строительства и столярного дела. – Режим доступа: https://cadwork.formes-service.de/, свободный. – Заглавие с экрана. – Яз. рус.

References

1. Balaji B. et al. Models, abstractions, and architectures: The missing links in cyber-physical systems. *Proceedings of the 52nd Annual Design Automation Conference*. ACM, 2015, p. 82.

2. Booch G. *Object-oriented analysis and design with applications (Third Edition).* Pearson Education Inc., 2007.

3. Bures T. et al. Software abstractions for component interaction in the internet of things. *Computer*, 2016, vol. 49, no. 12, pp. 50–59.

4. Hnetynka P. Software abstractions and architectures for smart cyber-physical systems: Keynote address. *Software Engineering Research, Management and Applications (SERA), 2017 : IEEE 15th International Conference on*. IEEE, 2017, p. 3.

5. Kuznetsov M. A., Surkov V. O. Analysis of complexity metrics of a software code for obfuscating transformations of an executable code. *IOP Conference Series: Materials Science and Engineering*. IOP Publishing, 2016, vol. 155, no. 1, p. 012008.

6. Shudrak M. O., Zolotarev V. V. Improving fuzzing using software complexity metrics. *International Conference on Information Security and Cryptology*. Springer, Cham, 2015, p. 246–261.

7. Sommerville I. *Software Engineering. International computer science series*. Addison Wesley, 2004.

8. Zambonelli F. Key abstractions for iot-oriented software engineering. *IEEE Software*, 2017, vol. 34, no. 1, pp. 38–45.

Software

9. *ArchiCad – software package for architects, based on information modeling technology*. Available at: https://www.graphisoft.ru/archicad/.

10. *WoodCon – a program for the design of wooden buildings and roofs*. Available at: http://www.arcon-eleco.ru/prog/woodcon/.

11. *ArCon Eleco Small Business – a program for creating an architectural and design project*. Available at: http://www.arcon.ru/products/architectural-engineering-and-design/small-business/.

12. *Dlubal – software for the calculation and design of structures*. Available at: https://www.dlubal.com/ru/produkty/dopolnitelnyje-moduli-rfem-i-rstab/.

13. *LogHouse – software for the design of log cabins*. Available at: http://loghouse.pro/loghouse/.

14. *CADWORK wood – program for wood construction and joinery*. Available at: https://cadwork.formes-service.de/.