УДК 004.031.6

В.В. Литвинов, д-р техн. наук
И.В. Богдан, ассистент
Черниговский национальный технологический университет, г. Чернигов, Украина

АВТОМАТИЗИРОВАННАЯ СИСТЕМА ВЕРИФИКАЦИИ МОДЕЛЕЙ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

В.В. Литвинов, д-р техн. наук
І.В. Богдан, асистент
Чернігівський національний технологічний університет, м. Чернігів, Україна

АВТОМАТИЗОВАНА СИСТЕМА ВЕРИФІКАЦІЇ МОДЕЛЕЙ ОБ'ЄКТНО-ОРІЄНТОВАНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Vitaliy Litvinov, Doctor of Technical Sciences
Irina Bogdan, assistant
Chernigov National University of Technology, Chernigov, Ukraine

THE AUTOMATED SYSTEM OF VERIFICATION OF THE OBJECT-ORIENTED SOFTWARE MODELS

Обоснована необходимость создания инструментального средства верификации моделей объектно-ориентированного программного обеспечения, которое бы максимально точно выполняло оценку корректности UML-диаграмм. Предложена автоматизированная система верификации моделей объектно-ориентированного программного обеспечения, которая позволяет выполнять верификацию каждой из входящих в состав модели UML-диаграммы несколькими методами.

Ключевые слова: *верификация, диаграмма, UML, метод.*

Обґрунтовано необхідність створення інструментального засобу верифікації моделей об'єктно-орієнтованого програмного забезпечення, який би максимально точно виконував оцінювання коректності UML-діаграм. Запропоновано автоматизовану систему верифікації моделей об'єктно-орієнтованого програмного забезпечення, яка дозволяє виконувати верифікацію кожної з UML-діаграми, що входить до складу моделі, декількома методами.

Ключові слова: *верифікація, діаграма, UML, метод.*

The necessity of creating an instrumental means of verification of the object-oriented software models that would perform the valuation of the UML-diagrams correctness most accurately is justified. The automated system of verification of the object-oriented software models that allows to verify every model of the UML-diagram that is included into the consist using several ways is offered.

Key words: *verification, chart, UML, method.*

Постановка проблемы. Постоянно возрастающая сложность программного обеспечения приводит к увеличению количества ошибок в нем, а одновременный рост количества и критичности выполняемых им функций влечет рост ущерба от этих ошибок. Потери одной экономики США от некачественного программного обеспечения составляют около 60 миллиардов долларов в год [1]. Для обеспечения корректности и надежности работы программного обеспечения большое значение имеют различные методы верификации, позволяющие выявлять ошибки на ранних этапах разработки программного обеспечения, таких как этап проектирования.

Объектно-ориентированный подход является на данный момент одним из самых популярных при разработке программного обеспечения. Создание объектно-ориентированного программного обеспечения начинается с создания его модели, представляемой чаще всего в виде UML-диаграмм. Именно с верификации UML-диаграмм и начинается создание корректно работающего и надежного программного обеспечения. Однако далеко не всегда существующие на данный момент методы и инструменты [2] верификации моделей объектно-ориентированного программного обеспечения позволяют максимально точно оценить корректность UML-диаграмм.

Анализ последних исследований и публикаций. Исследования показали, что большинство существующих на данный момент инструментальных средств верификации моделей объектно-ориентированного программного обеспечения является либо проприетарными, либо же незаконченными продуктами, которые не предоставляют исчерпывающей информации относительно корректности UML-диаграмм.

Выделение не решенных ранее частей общей проблемы. Проанализировав существующие системы верификации объектно-ориентированного программного обеспечения и его моделей, можно сделать вывод о том, что разрабатываемый инструмент верификации программного обеспечения должен позволять:

– максимально точно оценивать корректность модели;

– сводить к минимуму количество ошибок на этапе разработки.

Цель статьи. Главной целью данной статьи является разработка инструментального средства для верификации моделей объектно-ориентированного программного обеспечения, которое бы выполняло оценку корректности UML-диаграмм различными методами и максимально точно.

Изложение основного материала. Прежде чем приступать к разработке системы, необходимо выделить основные функциональные возможности разрабатываемой системы. Общая диаграмма вариантов использования представлена на рис. 1.

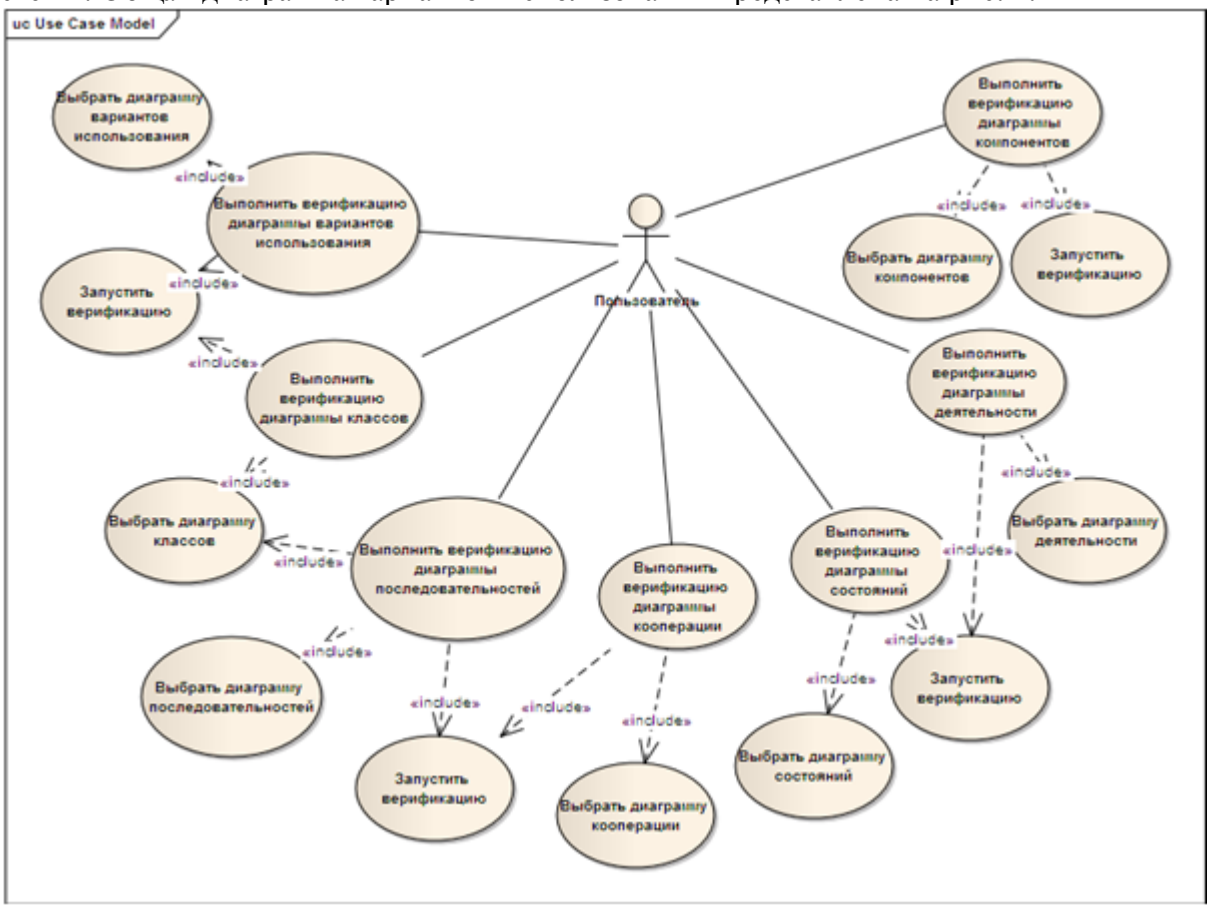

Рис. 1. Диаграмма вариантов использования системы

Любой пользователь системы может выполнить верификацию диаграммы вариантов использования, диаграммы классов, диаграммы последовательностей, диаграммы кооперации, диаграммы состояний, диаграммы деятельности и диаграммы компонентов. Для этого ему необходимо выбрать соответствующую диаграмму и запустить непосредственную верификацию. Исключение составляет диаграммы последовательностей, так как для ее верификации необходимо выбрать не только диаграмму последовательности, но и соответствующую ей диаграмму классов, которая обязательно должна пройти верификацию ранее.

Разработка архитектуры системы. На рис. 2 приведена архитектура инструментального средства верификации, которое позволяет оценивать корректность UML-диаграмм.

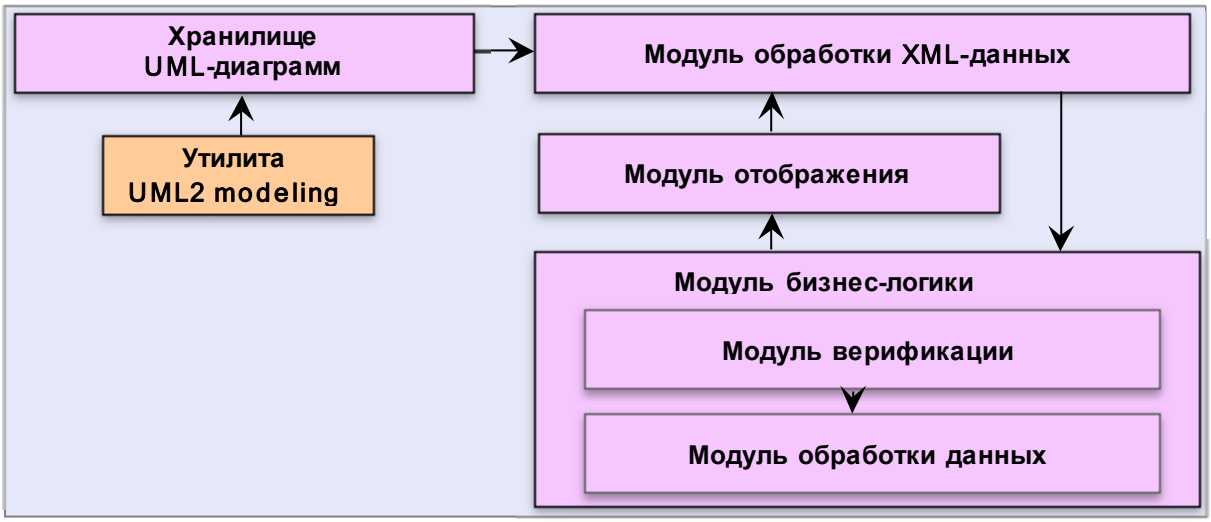

Рис. 2. Архитектура разрабатываемой системы

Так как для создания приложения необходима была утилита, которая бы являлась открытой и имела возможность создания максимального количества описанных в стандарте UML2-диаграмм, то была выбрана утилита uml2 modelingtools, которая является бесплатной, обеспечивает создание базовых диаграмм, а также интегрируется с открытой средой разработки Eclipse. Поскольку это Java-приложение, оно работает на любой платформе, которая поддерживает Java (начиная с Java 1.5).

Таким образом, разрабатываемая автоматизированная система верификации UML-диаграмм состоит из четырех основных модулей, имеющих в своем составе подмодули, которые взаимодействуя между собой и утилитой uml2 modelingtools, обеспечивают выполнение необходимых функций для эффективной оценки корректности UML-диаграмм. К данным модулям относятся:

– модуль обработки XML данных;

– модуль бизнес-логики, состоящий из модуля верификации и модуля обработки данных;

– хранилище UML-диаграмм;

– модуль отображения.

Ниже представлено описание модулей разрабатываемой системы.

Модуль обработки XML-данных. При создании диаграммы с помощью выбранной утилиты появляются два файла: непосредственно сама диаграмма, а также файл *.uml, в котором диаграмма представляется в виде XML-файла. Именно этот файл используется далее для парсинга диаграммы, так как формат XML является одним из простейших и существенно упрощает работу разработчика.

После получения нужных диаграмм в формате XML, данные XML обрабатываются и представляются в виде java-объектов. Получение этих объектов происходит при помощи парсеров с использованием библиотеки dom4j, которая позволяет парсить DOM-дерево для Java.

Парсер диаграммы вариантов использования позволяет получить информацию об актерах и названиях, связанных с ними вариантов использования. Парсер диаграммы классов позволяет получить названия классов, их идентификаторы, названия методов и атрибутов. Что касается парсера диаграммы последовательности, он позволяет получить линии жизни, сообщения, их идентификаторы, фрагменты, показывающие процесс передачи сообщений и их последовательность.

Модуль бизнес-логики. В модуле бизнес-логики, который состоит из модуля верификации и модуля обработки данных, происходит непосредственная оценка корректности поступающих в него диаграмм.

Модуль верификации. В разрабатываемой автоматизированной системе верификации предполагается комплексное использование всех существующих и предложенных разработчиками подходов и методов верификации UML-диаграмм.

Для верификации диаграммы вариантов использования важное значение имеет тот факт, что случаи использования, отображенные на ней, не являются представлениями программного обеспечения. Они представляют требования, которым должно соответствовать программное обеспечение. Случаи использования чаще всего формулируются на естественном языке, поэтому и их верификация практически сводится к проверке синтаксиса. Так, поскольку вариант использования описывает процесс или активность, то согласно стандарту UML 2 [3] каждый случай использования на диаграмме вариантов использования должен быть описан в виде глагола. Кроме того, каждый вариант использования должен относиться как минимум к одному актеру.

Диаграмма классов является одной из основных в модели объектно-ориентированного программного обеспечения, поэтому ее верификация будет выполняться одновременно несколькими методами, детально каждый из которых описан в статье [4].

Одним из наиболее простых подходов к верификации диаграммы классов является подход, основанный на построении драйвера. Данный подход позволяет формально оценить правильность создания диаграммы классов и выявить наиболее характерные ошибки.

Верификация классов из диаграммы классов чаще всего выполняется путем разработки драйвера, который создает экземпляры каждого класса и окружает эти классы соответствующей средой. Таким образом, становится возможным выполнение драйвера. Драйвер посылает одно или большее количество сообщений экземпляру класса в соответствии со спецификацией тестового случая, затем проверяет исход этих сообщений на основании значений ответа, изменения экземпляра и (или) один или большее число параметров сообщения. В обязанности драйвера чаще всего входит удаление любого созданного им экземпляра в том случае, если в языке программирования, таком как C++, имеет место управляемое программистом распределение памяти.

Если для конкретного класса характерны статические элементы данных и (или) операции, то для них также необходимо выполнять верификацию. Такие элементы данных и методы принадлежат самому классу, но не каждому экземпляру этого класса. Класс можно рассматривать как объект. Например, в Java – это экземпляр класса Class.

Если поведение экземпляров класса базируется на значениях атрибутов уровня класса, то все случаи, предназначенные для верификации этих атрибутов уровня класса, должны рассматриваться как расширение состояний этих экземпляров.

Если между двумя и более классами на диаграмме классов присутствует связь «наследование» (обобщение), то драйверу необходимо проверить отсутствие множественного наследования для языка Java, а также отсутствие двунаправленного наследования, при котором каждый из двух классов является и родительским, и дочерним одновременно для всех объектно-ориентированных языков программирования.

Если между классом и интерфейсом на диаграмме классов присутствует связь «реализация», то драйверу необходимо проверить, реализует ли класс все те методы, сигнатуры которых указаны в интерфейсе.

Если в классе присутствует хотя бы один абстрактный метод, то драйверу необходимо проверить, чтобы и класс был абстрактным.

Еще одним подходом к верификации диаграммы классов является метод шаблонов. Данный метод предложила в 2004 году Мира Балабан [5]. Понятие шаблона в данном подходе отличается от хорошо известного и знакомого понятия шаблона проектирования, который представляет собой формализованное описание часто встречающейся задачи проектирования, удачное решение данной задачи, а также рекомендации по применению этого решения в различных ситуациях. Шаблон согласно методу Мира Балабан, напротив, является примером ошибок при построении диаграммы классов.

Mira Balaban предложила несколько шаблонов.

Один из самых популярных шаблонов носит название Цикл множественной иерархии (Multiplicity Hierarchy Cycle).

Согласно данному шаблону, если между двумя классами имеется связь «наследование» (обобщение), которая означает наличие между ними отношения один-к-одному или же несколько дочерних классов к одному родительскому классу, то и любая другая связь между данными классами также должна указывать на наличие отношения один-к-одному или же несколько дочерних классов к одному родительскому классу.

Подразновидностью шаблона Цикл множественной иерархии является шаблон Взаимодействие циклов множественной иерархии (Interaction of multiplicity and inter-association hierarchy constraints).

Согласно данному шаблону, если между двумя классами имеется связь «наследование» (обобщение), которая означает наличие между ними отношения один-к-одному или же несколько дочерних классов к одному родительскому классу, и каждый из них ассоциативно связан с третьим классом, то данные связи должны указывать на наличие одинаковых отношений.

Не менее популярный и часто встречаемый шаблон носит название Простой множественный цикл (Pure Multiplicity Cycle).

Согласно данному шаблону, если между двумя классами имеется несколько ассоциативных связей и каждая связь имеет свое отношение, то все отношения со стороны одного и того же класса должны либо быть равными, либо же включать друг друга.

Еще один подход к верификации диаграммы классов основан на построении идентификационного графа. Разработкой данного метода в период с 1990 по 2001 года занимались американские ученые Hartmann, Lenzerini, Nobili и Thalheim [6].

Суть данного метода состоит в построении идентификационного графа, представляющего собой ориентированный граф. Узлами данного графа являются классы, а также ассоциативные связи между ними, а дуги связывают ассоциации с теми классами, между которыми на диаграмме классов и указаны соответствующие ассоциативные связи. В качестве весов дуг используются отношения, которыми ассоциативно связаны классы. Как и в обычном графе, в идентификационном графе вес пути вычисляется как произведение весов всех дуг, входящих в состав данного пути.

Основным предназначением идентификационного графа является определение причины нарушения конечной выполнимости диаграммы классов. Также как и метод шаблонов, данный метод позволяет выявить противоречивые связи – так называемые критические циклы.

Еще один подход к верификации диаграммы классов основан на представлении класса в качестве множества. Данный метод был предложен Calvanese и Lenzerini [7] и позволяет оценить корректность, прежде всего, иерархий классов.

Согласно данному методу каждый класс, входящий в состав иерархии, представляется в виде множества. Далее создается система неравенств, в которую включаются все возможные неравенства и равенства, если такие имеются, между классами-множествами. Затем необходимо решить систему неравенств и если результатом станет не пустое подмножество, то данная иерархия классов построена корректно, если же пустое множество – иерархия классов не корректна.

Не менее значимой в рамках модели объектно-ориентированного программного обеспечения является и диаграмма последовательности, верификация которой также будет выполняться несколькими методами, детально каждый из которых описан в статье [8].

Одним из наиболее простых является подход, связанный с построением драйвера. Данный подход позволяет формально оценить правильность создания диаграммы последовательности и выявить наиболее характерные ошибки.

Так как некоторые объекты на диаграмме последовательностей будут существовать постоянно, а некоторые временно (только в период выполнения ими требуемых дей-

ствий), то, прежде всего, следует убедиться в том, что уничтожение или же создание объектов, которые создаются на время выполнения своих действий, происходит корректно и для них предусмотрены явные сообщения.

Далее следует выполнять непосредственную верификацию взаимодействия между объектами. Основное внимание во время верификации взаимодействий в условиях контактного подхода уделяется проверке, выполнены ли объектом-отправителем предусловия методов получающего объекта. Верификация не проходит в том случае, если нарушаются эти предусловия. Смысл подобной проверки заключается в том, чтоб установить, выполняет ли объект-отправитель проверку предусловий объекта-получателя, прежде чем отправлять заранее неприемлемое сообщение. Одновременно с этим проверяется, корректно ли объект-отправитель прекращает свою деятельность, возможно, генерируя при этом исключение.

Верификацию диаграмм последовательностей также можно выполнять методом протоколов [9]. По мере того, как некоторый объект вступает во взаимодействие с другими объектами, увеличивается количество сообщений, которые он получает. Все эти сообщения упорядочиваются в определенную последовательность. Верификация по протоколу проверяет реализацию на соответствие этой последовательности. Различные протоколы, в которых принимает участие тот или иной объект, могут быть логически выведены из пред- и постусловий, регламентирующих выполнение отдельных операций, объявленных в классе этого объекта.

Идентифицирующая последовательность вызовов методов, в которую объединяются методы, чьи постусловия удовлетворяют предусловиям следующего метода, образуют протокол. Такие последовательности обнаруживаются на диаграмме классов путем анализа методов каждого конкретного класса. При данном подходе каждый отдельный протокол содержит методы отдельного класса.

Еще один подход к верификации диаграммы последовательностей основан на представлении диаграммы последовательностей в качестве абстрактного цифрового автомата.

Абстрактная теория автоматов, отвлекаясь от структуры автомата и не интересуясь способом его построения, изучает поведение автомата относительно внешней среды. Абстрактный цифровой автомат А определяется совокупностью из пяти объектов {X, S, Y, φ, λ}, где X = $\{x_i\}$, i$\in\overline{1,m}$ – множество входных сигналов автомата А (входной алфавит автомата А); S = $\{s_j\}$, j$\in\overline{1,n}$ – множество состояний автомата А (алфавит состояний автомата А); Y = $\{y_k\}$, k$\in\overline{1,l}$ – множество выходных сигналов автомата А (выходной алфавит автомата А); φ – функция переходов автомата А, задающая отображение (X × S)→ S, т. е. ставящая в соответствие любой паре элементов декартового произведения множеств (X × S) элемент множества S; λ – функция выходов автомата А, задающая либо отображение (X × S)→ Y, либо отображение S→ Y [10].

В данном случае один автомат описывает последовательность взаимодействия объектов на одной диаграмме последовательности, где под состоянием подразумевается совокупность состояний всех объектов, присутствующих на данной диаграмме. В зависимости от того, участвует ли тот или иной объект в определенный момент времени во взаимодействии определяется его активность или пассивность (1 или 0).

Таким образом, множество состояний автомата S представляет собой множество из совокупностей состояний каждого объекта, находящегося на диаграмме последовательностей, представленных в двоичной системе. Как множество входных, так и множество выходных сигналов автомата X и Y соответственно составляет множество сообщений на диаграмме последовательности, которые делают объект активным или же пассивным.

Верификацию диаграммы кооперации так же, как и верификацию диаграммы последовательностей, можно выполнять методом протоколов, метод тестового драйвера и метод, основанный на построении абстрактного цифрового автомата, для данной диаграммы не применимы, поскольку учитывают временной аспект.

Каждая диаграмма состояний представляет некоторый автомат, который описывает поведение отдельного объекта в форме последовательности состояний, которые охватывают все этапы его жизненного цикла, начиная от создания объекта и заканчивая его уничтожением. Таким образом, верификация диаграммы состояний происходит путем проверки условий, вытекающих с теории автоматов и специальной семантики языка UML, которые описаны в статье [11].

Верификацию диаграммы компонентов можно выполнять исключительно путем проверки связей между вышеперечисленными видами компонентов:

– компоненты со стереотипом «table» не могут быть непосредственно связаны с какими-либо компонентами, только через компонент со стереотипом «DB»;

– компоненты-рабочие продукты не могут быть непосредственно связаны с компонентами со стереотипом «DB», только через компоненты исполнения;

– компоненты развертывания не могут быть непосредственно связаны с компонентами со стереотипом «DB».

Поскольку конкретная реализация логического представления модели разрабатываемого программного обеспечения зависит от используемого программного инструментария, то более детальная верификация диаграммы компонентов невозможна.

Так как на диаграмме развертывания отображаются физически существующие элементы, распределения компонентов системы и физические связи между узлами, которые могут быть произвольными, то и верификация данной диаграммы невозможна.

Таким образом, модуль верификации получает от модуля обработки XML-данных информацию из диаграмм и обрабатывает ее при помощи методов верификации, а затем передает результаты работы методов в модуль обработки данных.

Модуль обработки данных. В модуле обработки данных происходит непосредственная идентификация ошибок и получение результата, который далее при помощи модуля отображения представляется в удобном и понятном для пользователя виде.

Выводы и предложения. В статье описана архитектура автоматизированной системы верификации моделей объектно-ориентированного программного обеспечения. Данная система не только является кроссплатформенной и свободно распространяемой, но также и позволяет выполнять верификацию основных видов UML-диаграмм одновременно несколькими различными методами и получать более точную информацию о их корректности.

Список использованных источников

1. *The Economic* Impacts of Inadequate Infrastructure for Software Testing. NIST Report, May 2002.

2. *Литвинов В. В.* Инструментальные средства верификации моделей программного обеспечения / В. В. Литвинов, И. В. Богдан, К. С. Сливко // Вісник Чернігівського державного технологічного університету. – 2013. – № 2. – С. 120–125.

3. *Стандарт ЮМЛ2* [Электронный ресурс]. – Режим доступа: http://www.omg.org/spec/UML/2.5/Beta2/.

4. *Литвинов В. В.* Формальная верификация диаграммы классов / В. В. Литвинов, И. В. Богдан // Математичні машини і системи. – 2013. – № 2. – С. 41–47.

5. *Balaban M.* Management of Correctness Problems in UML Class Diagrams – Towards a Pattern-based Approach / Balaban M., Maraee A., Stur A. – Beer Sheva : Department of Computer Science, Ben-Gurion University of the Negev, 2002. – 33 p.

6. *Thalheim, B.* Fundamentals of Entity-Relationship Modeling / Hartmann, Lenzerini, Nobili, Thalheim // Annals Mathematics and Artificial Intelligence. – 1993. – № 7. – P. 197–256.

7. *Calvanese, D.*, Lenzerini, M. On the Interaction between ISA and Cardinality Constraints. In Proceedings of the 10th IEEE International Conference on Data Engineering, Houston, Texas, USA. IEEE Computer Society. – Washington, DC, USA, 1994. – P. 204–213.

8. *Formal* verification of the sequence diagram / Vitaliy Lytvynov, Irina Bogdan // International journal "Information content & processing". – 2014. – № 1. – С. 79–86.

9. *Макгрегор Дж.* Тестирование объектно-ориентированного программного обеспечения : практическое пособие : пер. с англ. / Дж. Макгрегор, Д. Сайкс. – К., 2002. – 432 с.

10. *Прикладная* теория цифровых автоматов / К. Г. Самофалов, А. М. Романкевич, В. Н. Валуйский, Ю. С. Каневский, М. М. Пиневич. – К. : Вища школа, 1987. – 374 с.

11. *Литвинов В. В.* Тестирование моделей объектно-ориентированного программного обеспечения / В. В. Литвинов, И. В. Богдан // Математичні машини і системи. – 2012. – № 2. – С. 117–125.

УДК 004.896:629.3.081.4

М.В. Двоєглазова, канд. техн. наук
Чернігівський національний технологічний університет, м. Чернігів, Україна

ДОСЛІДЖЕННЯ ПРОЦЕСУ ФУНКЦІОНУВАННЯ ІНТЕГРОВАНОЇ ІНФОРМАЦІЙНОЇ СИСТЕМИ ПІДПРИЄМСТВА ТА ПРОЕКТУ

М.В. Двоеглазов, канд. техн. наук
Черниговский национальный технологический университет, г. Чернигов, Украина

ИССЛЕДОВАНИЕ ПРОЦЕССА ФУНКЦИОНИРОВАНИЯ ИНТЕГРИРОВАННОЙ ИНФОРМАЦИОННОЙ СИСТЕМЫ ПРЕДПРИЯТИЯ И ПРОЕКТА

Maryna Dvoiehlazova, PhD in Technical Science
Chernihiv National University of Technology, Chernihiv, Ukraine

RESEARCH OF PROCESS OF FUNCTIONING OF THE INTEGRATED INFORMATION SYSTEMS OF THE ENTERPRISE AND THE PROJECT

Проведено дослідження ефективності функціонування інтегрованої інформаційної системи машинобудівного підприємства та інвестиційного проекту його розвитку. Побудова інтегрованої інформаційної системи в роботі проведена за допомогою побудови шарів ефективності при використанні моделі аналізу середовища функціонування. Встановлено, що ефективність функціонування інтегрованої інформаційної системи збільшується протягом життєвого циклу проекту.

Ключові слова: *інтегрована інформаційна система, підприємство, проект, ефективність.*

Проведено исследование эффективности функционирования интегрированной информационной системы машиностроительного предприятия и инвестиционного проекта его развития. Установлено, что эффективность функционирования интегрированной информационной системы увеличивается в течение жизненного цикла проекта.

Ключевые слова: *интегрированная информационная система, предприятие, проект, эффективность.*

Research of efficiency of functioning of the integrated information system of machine-building enterprise and the investment project of its development is conducted. It is established that efficiency of functioning of the integrated information system increases during life cycle of the project.

Key words: *the integrated information system, the enterprise, the project, efficiency.*

Постановка проблеми. Проблема інтеграції інформаційних систем проектів та підприємства зумовлена багатьма факторами, пов'язаними зі зростанням обсягів даних, які застосовують автоматизовані інформаційні системи при розрахунку та реалізації інвестиційних проектів.

Аналіз останніх досліджень та публікацій. Питанням інформаційної взаємодії інформаційних систем присвячені роботи Д.В. Ланде, В.М. Левикіна, О.Є. Литвиненка, О.А. Павлова, Ю.М. Теслі та ін. [1–5]. Проте в межах проблеми інтеграції інформаційних систем недостатньо уваги приділено питанням дослідження процесу функціонування інтегрованих інформаційних систем підприємств та інвестиційних проектів.

Мета. Метою статті є дослідження процесу функціонування інтегрованої інформаційної системи підприємства та інвестиційного проекту, що реалізується на підприємстві машинобудівної галузі.

Виклад основного матеріалу. На основі механізму інтеграції інформаційних систем, розробленого в роботі [6], була побудована нейронна мережа для вибору найбільш раціональної конфігурації їх кінцевого стану. На основі цих досліджень було доведено,