УДК 004.41

ГЕНЕРАЦИЯ ИСХОДНОГО КОДА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ НА ОСНОВЕ МНОГОУРОВНЕВОГО НАБОРА ПРАВИЛ

Э.Н. Самохвалов, **Г.И. Ревунков**, **Ю.Е. Гапанюк**

МГТУ им. Н.Э. Баумана, Москва, Российская Федерация
e-mail: gapyu@bmstu.ru; revunkov@bmstu.ru; eduard.samohvalov@yandex.ru

Предложен подход к разработке программного обеспечения на основе многоуровневого набора правил для генерации исходного кода текстов программ. Рассмотрены особенности и недостатки объектно-ориентированного и сервис-ориентированного подходов при проектировании программного обеспечения, проведено сравнение указанных подходов с предлагаемым. Определены основные требования к предлагаемому подходу, показано выполнение этих требований в рамках данного подхода. Предложено использование уровней синтаксиса, семантики и прагматики для организации правил системы. Рассмотрены метаграфы в качестве структуры представления семантики системы. Представлена формализованная модель системы генерации исходного кода программного обеспечения на основе многоуровневого набора правил. Предложена обобщенная методика проектирования с использованием системы генерации исходного кода программного обеспечения на основе многоуровневого набора правил. Рассмотрена проблема разработки автоматизированных тестов в рамках предложенного подхода.

***Ключевые слова*:** проектирование программного обеспечения, объектно-ориентированный подход, сервис-ориентированный подход, метаграф, генерация исходного кода, многоуровневый набор правил.

SOURCE CODE GENERATION OF SOFTWARE BASED ON MULTILEVEL SET OF RULES

E.N. Samohvalov, **G.I. Revunkov**, **Yu. E. Gapanyuk**

Bauman Moscow State Technical University, Moscow, Russian Federation
e-mail: gapyu@bmstu.ru; revunkov@bmstu.ru; eduard.samohvalov@yandex.ru

Approach for software engineering based on multilevel set of rules for source code generation of sources program is proposed. Characteristics and disadvantages of object-oriented and service-oriented approaches for software development are examined; comparison of these approaches with proposed approach is given. Basic requirements for proposed approach are given; fulfilment of these requirements is shown. Using syntactic, semantic and pragmatic levels for organization of systems rules is proposed. Metagraphs are considered as a structure of the definition for system semantics. Formalized model of system for source code generation based on multilevel set of rules is presented. A generalized design methodology using a system for source code generation based on multilevel set of rules is proposed. The problem of development of automated tests in the context of the proposed approach is examined.

***Keywords*:** software engineering, object-oriented approach, service-oriented approach, metagraph, sourcecode generation, multilevel set of rules.

Введение. В настоящее время программное обеспечение становится все более сложным продуктом интеллектуальной деятельности. В связи с этим актуальной задачей становится повышение качества проектирования разрабатываемого программного обеспечения.

Проектирование программных систем традиционно рассматривается как сложная человеко-машинная методология. В рамках этого подхода предложены различные модели жизненного цикла разработки программного обеспечения: каскадная; итерационная; спиральная [1]. Предложены различные методологии разработки программного обеспечения: RUP; экстремальные подходы к проектированию и программированию [2].

Однако какая бы методология не использовалась для разработки программного обеспечения, ключевым звеном любой методологии является этап проектирования. Как правило, остальные этапы предназначены для устранения ошибок на указанном этапе, для документирования системы, обеспечения взаимодействия между командой проектировщиков и т.д.

В настоящее время можно выделить следующие основные подходы к проектированию программных систем.

1. Объектно-ориентированный подход (ОО-подход), как правило, с использованием шаблонов проектирования [3].

2. Проектирование на основе сервис-ориентированного подхода [4] (СОА-подход). В этом случае проектировщик создает отдельные модули системы в виде сервисов, а затем интегрирует сервисы в рабочие процессы на основе технологии workflow. СОА-подход часто применяется для высокоуровневой интеграции больших программных модулей, однако нет никаких препятствий для использования этого подхода на более детальных уровнях проектирования.

3. Проектирование на основе экспертных знаний и моделей (экспертный подход). В отличие от первых двух подходов, которые представляют собой сложившиеся методологии, это направление представляет собой скорее набор отдельных подходов, объединяющих такие общие черты, как использование при проектировании методов интеллектуальных систем, активное применение кодогенерации.

В рамках экспертного подхода можно отметить подходы:

- экспертное программирование, предложенное Г.Б. Евгеневым [5];
- концептуальное программирование, предложенное Э.Х. Тыугу [6];
- методы преобразования и перепроектирования программных систем, рассматривающие код программы как структуру данных, например, проект rascal-mpl (http://www.rascal-mpl.org/).

С точки зрения авторов экспертный подход наиболее перспективен. В рамках этого подхода для генерации исходного кода программных систем предлагается использование многоуровневого набора правил.

Постановка задачи. В настоящей работе поставлена задача разработки способа генерации программных систем, который обеспечивает выполнение требований, приведенных ниже.

Требование I. Позволяет гибко модифицировать (перегенерировать) код системы в зависимости от изменяющихся требований.

Требование II. Дает возможность гибко заменять элементы используемых технологий, например, замена одной сетевой библиотеки другой, более быстрой или надежной.

Требование III. Обеспечивает независимость от целевого языка (языков) программирования.

Требование IV. Обладает возможностью пояснения сгенерированного кода, позволяя установить связь текста программы с целями проектирования.

Организация уровней системы. Идея генерации программных систем на основе правил не является новой. Ключевая проблема заключается в организации уровней системы и правил преобразования.

Начиная с 1970-х годов, в рамках изучения вопросов ситуационного управления был предложен семиотический подход к построению информационных систем [7, 8]. В работе [7] отмечено, что “принципиальное отличие семиотических операций от формальных состоит в том, что в них отражается не только синтаксис, но также семантика и прагматика решаемых задач”. Таким образом, система является семиотической, если в ней рассматриваются синтаксис, семантика и прагматика решаемой задачи, а также отношения между ними. Детальные определения понятий синтаксиса, семантики и прагматики зависят от решаемой задачи.

Приведем определения понятий синтаксиса, семантики и прагматики для рассматриваемого случая:

1) синтаксис системы — набор конструкций языков программирования, в рамках которых генерируется программное обеспечение;
2) прагматика системы — набор исходных требований к генерируемому программному обеспечению;
3) семантика системы — структура данных, которая позволяет отобразить многообразие возможных отношений между прагматикой и синтаксисом.

Прагматику, семантику и синтаксис будем считать различными уровнями системы, которые используются для организации многоуровневого набора правил системы.

Возникает вопрос выбора подходящей структуры данных для реализации семантики. В настоящей работе в качестве такой структуры предложено использовать метаграф.

Метаграф как структура данных для представления семантики системы. Основная работа по теории метаграфов — монография А. Базу и Р. Блэннинга [9]. Единая теория метаграфов до сих пор не сформирована, поэтому можно встретить различные определения метаграфа, которые отличаются в деталях.

Используем определение, которое достаточно близко к определению, сформулированному Базу и Блэннингом:

$$\boldsymbol{MG} = \langle \boldsymbol{V}, \boldsymbol{E} \rangle, \boldsymbol{v}_i \in \boldsymbol{V}, \boldsymbol{e}_j \boldsymbol{E},$$

где $\boldsymbol{MG}$ — метаграф; $\boldsymbol{V}$ — множество вершин метаграфа; $\boldsymbol{E}$ — множество ребер метаграфа; $\boldsymbol{v}_i$ — вершина метаграфа; $\boldsymbol{e}_j$ — ребро метаграфа.

Определяемая вершина метаграфа

$$\boldsymbol{v}_k = \langle \{\boldsymbol{v}_i\}, \{\boldsymbol{e}_j\} \rangle.$$

Вершина метаграфа может содержать множество других вершин и ребер и фактически является подграфом. Поэтому в теории метаграфов вершины принято называть метавершинами, или гипервершинами. В одних определениях метаграфов атомарные вершины и метавершины полагаются различными объектами, в других — частным случаем метавершин.

В работе [9] также вводится понятие атрибутивного метаграфа MG^A. В атрибутивном метаграфе каждой вершине (метавершине) и ребру может быть приписано произвольное число атрибутов (числовых, строковых и др.):

$$\boldsymbol{MG}^A \stackrel{\text{def}}{=} \boldsymbol{MG}, \boldsymbol{v}_i = \{\boldsymbol{atr}_k\}, \boldsymbol{e}_j = \{\boldsymbol{atr}_n\}, \qquad (1)$$

где $\boldsymbol{MG}^A$ — атрибутивный метаграф; $\boldsymbol{atr}_k, \boldsymbol{atr}_n$ — атрибуты.

Перечисленные особенности делают метаграф удобной структурой для описания семантики системы. Очевидно, что метавершины с атрибутами позволяют моделировать внутренние зависимости практически для любых языков программирования (объектно-ориентированных, логических, функциональных и др.). Детальные описания таких моделей выходят за рамки настоящей статьи.

Формализованная модель системы. Систему генерации исходного кода программного обеспечения на основе многоуровневого набора правил представим в виде кортежа

$$\boldsymbol{S} = \langle \boldsymbol{PR}, \boldsymbol{SE}, \boldsymbol{SY}, \boldsymbol{R}, \boldsymbol{TR} \rangle,$$

где $\boldsymbol{S}$ — система генерации исходного кода программного обеспечения на основе многоуровневого набора правил; $\boldsymbol{PR}$ — прагматика системы; $\boldsymbol{SE}$ — семантика системы; $\boldsymbol{SY}$ — синтаксис системы; $\boldsymbol{R}$ — многоуровневый набор правил преобразования системы; $\boldsymbol{TR}$ — трек системы.

Прагматика представляет собой дерево целей проектируемой системы:

$$\boldsymbol{PR} \stackrel{\text{def}}{=} \boldsymbol{GT};$$

$$\boldsymbol{GT} = \langle \boldsymbol{GN}, \boldsymbol{GL} \rangle, \boldsymbol{GN} = \{\boldsymbol{g}_i\}.$$

Здесь $\boldsymbol{GT}$ — дерево целей; $\boldsymbol{GN}$ — множество вершин дерева целей; $\boldsymbol{GL}$ — множество ребер дерева, отражающее иерархические связи между целями; $\boldsymbol{g}_i$ — i-я цель проектирования системы.

Семантику системы представим в виде атрибутивного метаграфа в соответствии с (1):

$$\boldsymbol{SE} \stackrel{\text{def}}{=} \boldsymbol{MG}^A,$$

Синтаксис системы — это множество символов в целевом алфавите системы:

$$\boldsymbol{SY} = \{\boldsymbol{t}_i\}, \quad \boldsymbol{t}_i \in \boldsymbol{T_L} \subset \boldsymbol{T},$$

где $\boldsymbol{t}_i$ — символ целевого алфавита системы; $\boldsymbol{T_L}$ — подмножество целевого алфавита системы для языка $\boldsymbol{L}$; $\boldsymbol{T}$ — целевой алфавит системы.

Целевой алфавит системы представляет собой объединенное множество $\boldsymbol{T}$ синтаксических конструкций языков программирования $\boldsymbol{T_L}$, в рамках которых генерируется текст программы, а $\boldsymbol{t}_i$ — допустимую синтаксическую конструкцию языка программирования $\boldsymbol{T_L}$.

Набор правил преобразования $\boldsymbol{R}$ включает в себя следующие виды правил преобразования:

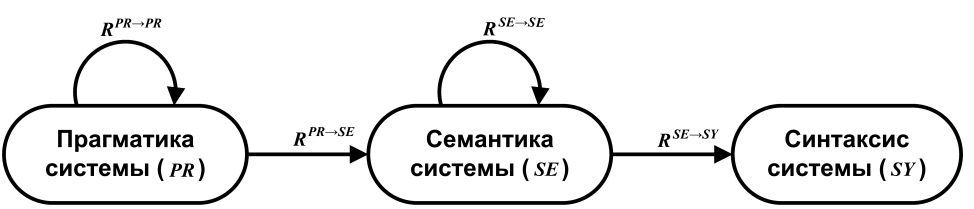
$$\boldsymbol{R} = \left\langle \boldsymbol{R}^{\boldsymbol{PR}\to\boldsymbol{PR}}, \ \boldsymbol{R}^{\boldsymbol{PR}\to\boldsymbol{SE}}, \ \boldsymbol{R}^{\boldsymbol{SE}\to\boldsymbol{SE}}, \ \boldsymbol{R}^{\boldsymbol{SE}\to\boldsymbol{SY}} \right\rangle,$$

где $\boldsymbol{R}^{\boldsymbol{PR}\to\boldsymbol{PR}}$ — правило преобразования в рамках прагматики системы; $\boldsymbol{R}^{\boldsymbol{PR}\to\boldsymbol{SE}}$ — правило преобразования из прагматики системы в ее семантику; $\boldsymbol{R}^{\boldsymbol{SE}\to\boldsymbol{SE}}$ — правило преобразования в рамках семантики системы; $\boldsymbol{R}^{\boldsymbol{SE}\to\boldsymbol{SY}}$ — правило преобразования из семантики системы в ее синтаксис.

Взаимосвязь между элементами системы и правилами преобразования приведена на рисунке.

Правило преобразования в рамках прагматики содержит четыре вида правил

$$\boldsymbol{R}^{\boldsymbol{PR}\to\boldsymbol{PR}} = \left\{ \boldsymbol{R}_{\boldsymbol{I}}^{\boldsymbol{PR}\to\boldsymbol{PR}}, \ \boldsymbol{R}_{\boldsymbol{II}}^{\boldsymbol{PR}\to\boldsymbol{PR}}, \ \boldsymbol{R}_{\boldsymbol{III}}^{\boldsymbol{PR}\to\boldsymbol{PR}}, \ \boldsymbol{R}_{\boldsymbol{IV}}^{\boldsymbol{PR}\to\boldsymbol{PR}} \right\}.$$



Взаимосвязь между элементами системы и правилами преобразования

Рассмотрим эти правила более подробно.

1. Правило добавления вершины $\boldsymbol{g}_j$, вложенной в вершину $\boldsymbol{g}_i$:

$$\boldsymbol{R}_{\boldsymbol{I}}^{\boldsymbol{PR\to PR}} : \boldsymbol{g}_i \to \boldsymbol{g}_i/\boldsymbol{g}_j,$$

где / — оператор вложенности вершин. Антецедент правила — вершина $\boldsymbol{g}_j$, консеквент правила — фрагмент дерева, в котором вершина $\boldsymbol{g}_j$ вложена в вершину $\boldsymbol{g}_i$.

2. Правило удаления вершины $\boldsymbol{g}_j$, вложенной в вершину $\boldsymbol{g}_i$:

$$\boldsymbol{R}_{\boldsymbol{II}}^{\boldsymbol{PR\to PR}} : \boldsymbol{g}_i/\boldsymbol{g}_j \to \boldsymbol{g}_i.$$

Используется для моделирования высказываний вида: “для того, чтобы выполнить цель $\boldsymbol{g}_i$, необходимо выполнить подцель $\boldsymbol{g}_j$”. Антецедент правила — фрагмент дерева, в котором вершина $\boldsymbol{g}_j$ вложена в вершину $\boldsymbol{g}_i$, консеквент правила — вершина $\boldsymbol{g}_i$.

3. Правило изменения вложенности вершин:

$$\boldsymbol{R}_{\boldsymbol{III}}^{\boldsymbol{PR\to PR}} : \boldsymbol{g}_k,\ \ \boldsymbol{g}_i/\boldsymbol{g}_j \to \boldsymbol{g}_i,\ \ \boldsymbol{g}_k/\boldsymbol{g}_j.$$

Полезно, когда цель (подцель) была задана ошибочно, и проектировщику необходимо ее удалить. Вершина $\boldsymbol{g}_j$, вложенная в антецеденте правила в вершину $\boldsymbol{g}_i$, становится вложенной в вершину $\boldsymbol{g}_k$ в консеквенте правила.

4. Правило слияния вершин:

$$\boldsymbol{R}_{\boldsymbol{IV}}^{\boldsymbol{PR\to PR}} : \boldsymbol{g}_k/\boldsymbol{g}_n,\ \ \boldsymbol{g}_i/\boldsymbol{g}_j \to \boldsymbol{g}_{ki}/\boldsymbol{g}_j,\ \ \boldsymbol{g}_n.$$

Применяется в случае, когда проектировщику необходимо перенести подцель в другую ветвь дерева целей. В результате вершины $\boldsymbol{g}_i$ и $\boldsymbol{g}_k$ объединяются, вершины $\boldsymbol{g}_j$ и $\boldsymbol{g}_n$ становятся вложенными в объединенную вершину $\boldsymbol{g}_{ki}$.

Правило преобразования из прагматики системы в ее семантику

$$\boldsymbol{R}^{\boldsymbol{PR\to SE}} : \boldsymbol{GT}_i \to \boldsymbol{MG}_j^A$$

используется для объединения целей. В этом случае осуществляется слияние подцелей объединяемых целей. Антецедент правила — фрагмент дерева целей проектируемой системы, консеквент правила — фрагмент атрибутивного метаграфа, который добавляется в семантику системы.

Правило преобразования в рамках семантики системы

$$\boldsymbol{R}^{\boldsymbol{SE\to SE}} : \boldsymbol{MG}_i^A \to \boldsymbol{MG}_j^A.$$

В этом правиле выполняется замена фрагмента атрибутивного метаграфа $\boldsymbol{MG}_i^A$ фрагментом атрибутивного метаграфа $\boldsymbol{MG}_j^A$.

Детальные разновидности правил преобразования из прагматики в семантику и в рамках семантики не рассматриваются в настоящей статье. Они могут зависеть от используемой парадигмы целевого языка

программирования (объектно-ориентированная, функциональная, логическая) и других факторов, что является предметом отдельного исследования.

Правило преобразования из семантики системы в ее синтаксис

$$\boldsymbol{R}^{\boldsymbol{SE\to SY}} : \boldsymbol{MG}_i^A \to \{\boldsymbol{t}_i\}$$

осуществляет преобразование фрагмента атрибутивного метаграфа $\boldsymbol{MG}_i^A$ во множество символов целевого алфавита системы $\boldsymbol{t}_i$. Это преобразование "смысловой структуры" в текст программы.

Для реализации таких преобразований разработано большое количество технологий. Это и языки шаблонов (например, T4 на платформе .NET), и технологии, подобные XSLT.

Трек системы определим следующим образом:

$$\boldsymbol{TR} = \{\boldsymbol{tr}_i\};$$

$$\boldsymbol{tr}_i = \langle \boldsymbol{R}_j,\ \boldsymbol{S}_A,\ \boldsymbol{S}_C \rangle;$$

$$\boldsymbol{R}_j \in \boldsymbol{R};$$

$$\boldsymbol{S}_A,\ \boldsymbol{S}_C \in \boldsymbol{PR} \cup \boldsymbol{SE} \cup \boldsymbol{SY},$$

где $\boldsymbol{tr}_i$ — элемент трека системы; $\boldsymbol{R}_j$ — выполненное правило, принадлежащее множеству правил системы; $\boldsymbol{S}_A$ — множество элементов, составляющих антецедент правила; $\boldsymbol{S}_C$ — множество элементов, образующих консеквент правила; элементы, составляющие антецедент и консеквент правила, в зависимости от вида правила могут принадлежать прагматике, семантике или синтаксису системы.

Обобщенная методика проектирования. Такая методика проектирования с использованием системы генерации исходного кода программного обеспечения на основе многоуровневого набора правил содержит следующие шаги.

1. Задание прагматики проектируемой системы в виде дерева целей.

2. Разработка правил системы.

3. Генерация текстов программ с помощью разработанных правил.

4. Компиляция текстов программ в исполняемые файлы (в случае использования компилируемых языков программирования).

5. Тестирование сгенерированной системы.

6. Возврат к пункту 2 в случае исправления ошибок в сгенерированном коде.

7. Возврат к пункту 1 в случае изменения требований к системе.

В систему постепенно добавляются и отлаживаются новые правила, что позволяет поэтапно наращивать функциональность системы.

Разработка автоматизированных тестов. К сожалению, в рамках предлагаемого подхода существует проблема с разработкой автомати-

зированных тестов. Конечно, автоматизированные тесты в рамках подходов TDD и BDD можно генерировать с помощью правила $\boldsymbol{R}^{SE \to SY}$. В этом случае тесты будут зависимы от правил генерации кода, что не позволит находить ошибки в правилах $\boldsymbol{R}^{PR \to SE}$ и $\boldsymbol{R}^{SE \to SE}$. Ошибка в правиле высокого уровня может привести к одновременной генерации ошибочного кода программы и ошибочного теста.

В соответствии с изложенным корректным вариантом является ручная разработка тестов на основе требований к системе. Предлагаемый подход позволяет облегчить работу проектировщика, но не облегчает работу тестировщика.

Реализация требований постановки задачи. Рассмотрим, как требования I–IV реализуются в рамках предложенного подхода.

Требование I. Проектировщик задает требования к системе в виде дерева целей проектирования. При перепроектировании системы дерево целей может изменяться с использованием правила $\boldsymbol{R}^{PR \to PR}$. На основании трека проектировщик может отследить цепочки правил, которые соответствуют изменившимся требованиям. Таким образом, проектировщику будет предоставлено множество правил, в которые он потенциально должен внести изменения. К сожалению, этот подход не позволяет системе "подсказать" проектировщику новые правила, которые он должен добавить в систему в связи с изменившимися требованиями. После изменения необходимых правил проектировщик проводит перегенерацию кода системы.

Требование II. Элементы технологий подключаются на этапе преобразования из семантики в синтаксис с использованием правила $\boldsymbol{R}^{SE \to SY}$. Если проектировщику необходимо провести замену библиотеки, то он изменяет соответствующие правила, что приводит к генерации текста программы с вызовом новой библиотеки.

Требование III. Независимость от целевого языка программирования обеспечивается правилом $\boldsymbol{R}^{SE \to SY}$. Изменение указанного правила позволяет изменить целевой язык программирования, например перейти от C++ к C#. При этом логика работы программы, задаваемая правилами более высоких уровней, остается неизменной. Однако, если проектировщику требуется изменение парадигмы целевого языка (например, с объектно-ориентированной на функциональную), то это может потребовать внесения изменений в правила более высоких уровней $\boldsymbol{R}^{PR \to SE}$ и $\boldsymbol{R}^{SE \to SE}$.

Требование IV. Для реализации поясняющего компонента в модели используется трек, с помощью которого можно отследить цепочку правил, приводящих к генерации определенного фрагмента кода $\boldsymbol{t}_i$ на основании цели проектирования $\boldsymbol{g}_i$ через промежуточные элементы семантического метаграфа.

Использование подхода на основе приведенных выше правил позволяет повторно применять правила в других проектах, внося в них необходимые изменения.

Обсуждение полученных результатов. Сравним предложенный подход с ОО- и СОА-подходами на основе требований I–IV.

Известно, что важную проблему представляет не проектирование, а перепроектирование системы. При использовании ОО-подхода эта проблема очень серьезна, так как подход на основе шаблонов проектирования является “хрупким”, перепроектирование часто приводит к полной переработке шаблонов системы. Эту проблему, в частности, пытаются решать с помощью элементарных шаблонов проектирования [10]. В случае СОА-подхода система при перепроектировании является менее “хрупкой”, поскольку она создана на основе небольших сервисов, элементы workflow поддаются изменениям достаточно хорошо. Однако workflow охватывает не все элементы системы, пользовательский интерфейс приходится перепроектировать традиционными методами. Предлагаемый подход обеспечивает максимальную гибкость при перепроектировании, изменение требований приводит к изменению правил, программный код генерируется автоматически.

ОО- и СОА-подходы не обеспечивают гибкой замены используемых технологий и языков программирования. За счет использования workflow, СОА-подход обладает несколько большей гибкостью по сравнению с ОО-подходом, однако вызовы библиотек, как правило, происходят на уровне сервисов в тексте программы. Предлагаемый подход обеспечивает гибкость, так как технологии и языки программирования задаются на уровне правил.

В ОО- и СОА-подходах объяснение текста программы выполняется с использованием комментариев, которые могут отсутствовать. В предлагаемом подходе с помощью трека можно проследить путь от целей проектирования до сгенерированного кода. Результаты сравнения приведены в таблице.

Результаты сравнения ОО-подхода, СОА-подхода и предлагаемого подхода

Требование	ОО-подход	СОА-подход	Предлагаемый подход
I. Гибкость модификации кода системы в зависимости от изменяющихся требований	Не обеспечивает	Частично обеспечивает	Обеспечивает
II. Гибкая замена элементов используемых технологий	Не обеспечивает	Частично обеспечивает	Обеспечивает
III. Независимость от целевого языка программирования	Не обеспечивает	Не обеспечивает	Обеспечивает
IV. Возможность объяснения сгенерированного кода	Не обеспечивает	Не обеспечивает	Обеспечивает

Заключение. В настоящее время актуальной задачей становится повышение качества проектирования разрабатываемого программного обеспечения. Однако существующие ОО- и СОА-подходы к проектированию обладают рядом недостатков. Предлагаемый подход позволяет обеспечить гибкость при перепроектировании программного обеспечения за счет применения многоуровневого набора правил для генерации исходных текстов программ.

В рамках развития подхода предложена детализация рассмотренной модели, разработка методов кодогенерации для различных целевых языков программирования.

ЛИТЕРАТУРА

1. *Вендров А.М.* Проектирование программного обеспечения экономических информационных систем. М.: Финансы и статистика, 2006. 544 с.
2. *Амблер С.* Гибкие технологии: экстремальное программирование и унифицированный процесс разработки. Библиотека программиста. СПб.: Питер, 2005. 412 с.
3. *Гамма Э.*, *Хелм Р.*, *Джонсон Р.*, *Влиссидес Дж.* Приемы объектно-ориентированного проектирования. Паттерны проектирования. СПб: Питер, 2001. 368 с.
4. *Биберштейн Н.*, *Боуз С.* Компас в мире сервис-ориентированной архитектуры (SOA); пер. с англ. М.: Кудиц-Пресс, 2007. 256 с.
5. *Евгенев Г.Б.* Интеллектуальные системы проектирования. М.: Изд-во МГТУ им. Н.Э. Баумана, 2009. 334 с.
6. *Тыугу Э.Х.* Концептуальное программирование. М.: Наука. Главная редакция физико-математической литературы, 1984. 256 с.
7. *Клыков Ю.И.*, *Горьков Л.Н.* Банки данных для принятия решений. М.: Сов. радио, 1980. 208 с.
8. *Клыков Ю.И.* Ситуационное управление большими системами. М.: Энергия, 1974. 136 с.
9. *Basu A.*, *Blanning R.* Metagraphs and Their Applications. Springer, 2007. 174 p.
10. *Смит Дж. Мак-Колм*. Элементарные шаблоны проектирования; пер. с англ. М.: ООО “И.Д. Вильямс”, 2013. 304 с.

REFERENCES

[1] Vendrov A.M. Proektirovanie programmnogo obespecheniya ekonomicheskikh informatsionnykh system [Software engineering of economic information systems]. Moscow, Finansy i Statistika Publ., 2006. 544 p.

[2] Ambler S. Agile modeling: effective practices for extreme programming and the unified process. N.Y., J. Wiley Publ., 2002. 400 p. (Russ. Ed.: Ambler S. Gibkie tekhnologii: ekstremal’noe programmirovanie i unifitsirovannyy protsess razrabotki. St. Petersburg, Piter Publ., 2005. 412 p.).

[3] Gamma E., Johnson R., Helm R., Vlissides J. Design patterns. Elements of reusable object-oriented software. Addison-Wesley Publ., 1994. 417 p. (Russ. Ed.: Gamma E., Khelm R., Dzhonson R., Vlissides Dzh. Priemy ob’ektno-orientirovannogo proektirovaniya. Patterny proektirovaniya. St. Petersburg, Piter Publ., 2001. 368 p.).

[4] Bieberstein N., Bose S., Fiammante M., eds. Service-oriented architecture (SOA) compass: business value, planning, and enterprise roadmap. USA, IBM Press, 2005. 272 p. (Russ. Ed.: Bibershteyn N., Bouz S. Kompas v mire servis-orientirovannoy arkhitektury (SOA). Moscow, KUDITs-Press Publ., 2007. 256 p.).

[5] Evgenev G.B. Intellektual'nye sistemy proektirovaniya [Intelligent design systems]. Moscow, MGTU im. N.E. Baumana Publ., 2009. 334 p.
[6] Tyugu E.Kh. Kontseptual'noe programmirovanie [Conceptual programming]. Moscow, Nauka Publ., 1984. 256 p.
[7] Klykov Yu.I., Gor'kov L.N. Banki dannykh dlya prinyatiya resheniy [Databanks for decision making]. Moscow, Sovetskoe Radio Publ., 1980. 208 p.
[8] Klykov Yu.I. Situatsionnoe upravlenie bol'shimi sistemami [Situation control of large systems]. Moscow, Energiya Publ., 1974. 136 p.
[9] Basu A., Blanning R. Metagraphs and their applications. USA, Springer, 2007. 174 p.
[10] Smith Jason McC. Elemental Design Patterns. 1st ed. Addison-Wesley Publ., 2012. 368 p. (Russ. Ed.: Smit Dzh. Mak-Kolm. Elementarnye shablony proektirovaniya. Moscow, Vil'yams Publ., 2013. 304 p.).

Статья поступила в редакцию 20.02.2014

Эдуард Николаевич Самохвалов — канд. техн. наук, профессор кафедры "Системы обработки информации и управления" МГТУ им. Н.Э. Баумана. Автор более 50 научных работ в области проектирования информационных систем в сфере обучения.
МГТУ им. Н.Э. Баумана, Российская Федерация, 105005, Москва, 2-я Бауманская ул., д. 5.

E. N. Samokhvalov — Cand. Sci. (Eng.), professor of "Information Processing System and Control" department of the Bauman Moscow State Technical University. Author of more than 50 publications in the field of information systems development in the education sphere.
Bauman Moscow State Technical University, Vtoraya Baumanskaya ul. 5, Moscow, 105005 Russian Federation.

Георгий Иванович Ревунков — канд. техн. наук, доцент кафедры "Системы обработки информации и управления" МГТУ им. Н.Э. Баумана. Автор более 40 научных работ в области баз данных, проектирования автоматизированных систем.
МГТУ им. Н.Э. Баумана, Российская Федерация, 105005, Москва, 2-я Бауманская ул., д. 5.
G. I. Revunkov — Cand. Sci. (Eng.), assoc. professor of "Information Processing System and Control" department of the Bauman Moscow State Technical University. Author of more than 40 publications in the field of database, automation systems development.
Bauman Moscow State Technical University, Vtoraya Baumanskaya ul. 5, Moscow, 105005 Russian Federation.

Юрий Евгеньевич Гапанюк — канд. техн. наук, доцент кафедры "Системы обработки информации и управления" МГТУ им. Н.Э. Баумана. Автор 15 научных работ в области проектирования автоматизированных систем.
МГТУ им. Н.Э. Баумана, Российская Федерация, 105005, Москва, 2-я Бауманская ул., д. 5.
Yu.E. Gapanyuk — Cand. Sci. (Eng.), assoc. professor of "Information Processing System and Control" department of the Bauman Moscow State Technical University. Author of 15 publications in the field of automation systems development.
Bauman Moscow State Technical University, Vtoraya Baumanskaya ul. 5, Moscow, 105005 Russian Federation.