# Анализ и визуализация зависимостей между пакетами программных систем

Романов В.Ю.

***Аннотация*** **- В статье рассматриваются методы визуализации архитектуры программной системы в составе инструмента обратного проектирования и восстановления архитектуры программной системы. Рассматриваются методы визуализации и анализа зависимости пакетов программной системы, написанной на языке Java, с помощью матриц структуры зависимостей и элементов этих матриц. Описанные в статье возможности инструмента предоставляют инфраструктуру для последующего обнаружения и исправления ошибок проектирования программных систем.**

***Ключевые слова*** **— software visualization, reengineering, dependency structural matrix, package, dependency, architecture recovery, reverse engineering.**

## I. Введение

Использование свободно распространяемых в сети Internet библиотек, разработанных на языке Java, получает все более широкое распространение. Многие такие библиотеки распространяются в исходных текстах, что дает возможность их развития независимыми группами разработчиков. Для распространения таких библиотек все чаще используются не только сайты разработчиков библиотек, но и централизованные хранилища (репозитории) со стандартизованным интерфейсом [1, 2]. Это позволяет ссылаться на библиотеки непосредственно по их уникальному адресу в репозитории. Разработка программного обеспечения в таком случае представляет собой сборку программного обеспечения с указанием уникального идентификатора группы (фирмы или команды разработчиков), уникального идентификатора артефакта (библиотеки) и номера версии артефакта. Разработана унифицированная структура репозиториев для хранения таких библиотек [3] и программный интерфейс [4] для работы с репозиториями артефактов, которые имеют стандартизованную структуру.

Многие библиотеки, предоставляющие проекту необходимую функциональность, вместе с тем требуют переработки (рефакторинга) исходного текста библиотеки. Зачастую это обусловлено ошибками проектирования совершенными при не согласованной работе многих разработчиков библиотеки меняющихся на протяжении нескольких лет. Такие ошибки проектирования приводят к необходимости подключения той функциональности библиотеки, которая реально не используется в проекте. Не используемый в проекте код в свою очередь может требовать подключения других библиотек. В итоге размеры подключаемого и реально используемого программного кода могут отличаться в несколько раз.

Другой распространенной причиной, которая требует рефакторинга используемых библиотек, является отсутствие исходных текстов у части используемых ими свободно распространяемых библиотек. Для некоторых проектов критичным является отсутствие в используемом коде так называемых «закладок», выполняющих непредсказуемые действия. Например, передачу информации об окружении исполняемой программы по сети интернет. Поэтому отсечение или замена кода, использующего библиотеки без исходных текстов, для таких проектов требует понимания архитектуры библиотеки показывающей связи исключаемого или заменяемого кода с остальным кодом библиотеки. Для выявления ошибок проектирования библиотеки CASE-инструментом выполняется построение, визуализация и верификация модели архитектуры библиотеки.

Архитектуру программной системы [5, 6, 7] во многом определяют зависимости между пакетами программной системы, а также зависимости пакетов программной системы от пакетов используемых библиотек. Поэтому восстановления архитектуры (architecture recovering) инструмент позволяющий визуализировать, понять, оценить и оптимизировать конкретную организацию пакетов системы и их взаимосвязей (логическую структуру системы), а также распределение пакетов по исполняемым файлам (физическую структуру системы) используемых библиотек и самой системы. Понимание организации пакетов программной системы критическая задача при восстановлении архитектуры системы. Для решения этой задачи было предложено множество подходов для представления информации о пакетах и их отношениях, метрик для измерения взаимосвязей пакетов, их внутренней структуры и эволюции.

Для понимания внутренней структуры пакетов была предложена визуализация пакетов с помощью карт распределения [8], показывающих распределение по пакетам классификаторов, обладающих заданными пользователем свойствами.

Визуализация связей пакета с другими пакетами системы как фасадов (surface) пакетов предложена в [9]. При этом виде визуализации пакетов показывается как

Владимир Юрьевич Романов, факультет ВМК МГУ им. М.В. Ломоносова, vromanov@cs.msu.su, vladimir.romanov@gmail.com.

внутренняя структура пакетов, так и связь пакета с пакетами поставщиками, показывающими предоставляющие выбранному для визуализации пакету классификаторы - интерфейсы, классы, перечисления.

В работе [10] для понимания структуры пакетов был предложена визуализации структуры пакетов как графа, узлы которого соединены с помощью портов и соединителей.
Визуализация и анализ зависимостей пакетов выполняется также с помощью матриц структурной зависимости [11, 12]

В данной статье рассматривается использование матриц структурной зависимости в инструменте обратного проектирования [13, 14, 15]. Описанные в статье возможности инструмента предоставляют инфраструктуру для последующего обнаружения, визуализации и исправления ошибок проектирования программных систем. Такие возможности будут рассмотрены в следующей статье.

## II. Инструмент восстановления архитектуры

Задача восстановления архитектуры является одной из задач при обратном проектировании программной системы [13, 14, 15]. При решении задачи обратного проектирования для уже существующего программного обеспечения строится его UML-модель [16, 17, 18], которая затем используется для построения UML-модели архитектуры программного обеспечения. На рисунке 1 показан вид такого инструмента обратного проектирования и восстановления архитектуры, интегрированного в среду Eclipse.

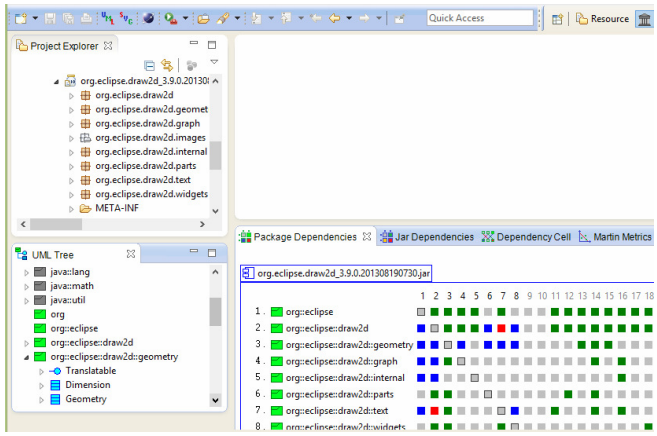

Рис.1. Инструмент восстановления архитектуры системы

UML-модель строится инструментом обратного проектирования для анализируемого проекта Eclipse, выбранного пользователем инструмента в виде Project Explorer среды Eclipse. Иерархия элементов построенной модели визуализируется в виде UML Tree инструмента, также показанном на рисунке 1. Затем элементы этих двух видов (проекты, библиотечные jar-файлы, пакеты, классы, интерфейсы), могут быть выбраны для визуализации в других специализированных видах инструмента. В частности, таким видом может быть показанный на рисунке 1 вид Package Dependencies, содержащий матрицу структурной зависимости пакетов выбранного проекта, библиотечного файла или пакета верхнего уровня. Существует, в частности, вид Jar Dependencies для визуализации с помощью матрицы зависимости представленных в UML-модели библиотек. Для визуализации элементов матриц зависимости используется вид Dependency Cell. Каждый элемент UML-модели, показываемый на любом из специализированных видов также может использоваться для выбора визуализируемых видов.

Изображенный в виде Package Dependencies пакет может использоваться для визуализации в одном из специализированных видов для метрик выбранного пакета [19, 20]. Например, в показанном на рисунке 2 виде Martin Metrics, который позволяет оценить правильность распределения классификаторов по пакетам [21].

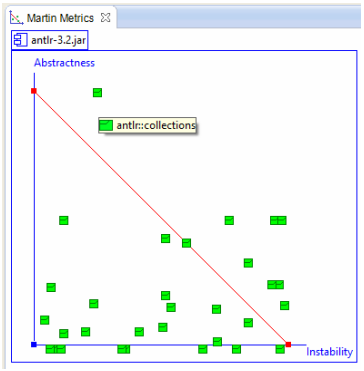

Рис.2. Визуализация метрик Роберта Мартина для пакетов в библиотеке antlr.jar

Имена пакетов в этом виде метрик показываются с помощью всплывающих подсказок.

Внутренняя структура пакетов и свойства входящих в них классификаторов могут быть показаны с помощью Code Properties (карты распределения), показанной на рисунке 3.

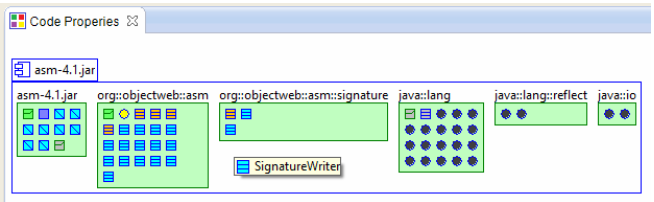

Рис.3. Визуализация внутренней структуры и свойств классификаторов для пакетов в библиотеке asm-4.1.jar

Важно отметить, что анализ зависимости пакетов представляет собой анализ логической структуры программной системы. Вместе с тем, излагаемые в статье подходы во многом применимы и при анализе физической структуры программной системы - распределении пакетов и классификаторов по компонентам. В случае программ написанных на языке Java - по файлам исполняемого кода с расширением файла jar.

Далее в статье рассматриваются матрицы зависимости пакетов, методы визуализации самой матрицы и ее ячеек. Заметим, что в данной статье рассматриваются

лишь простейшие циклы, когда два пакета зависят друг от друга непосредственно, и показываются в матрице красным цветом. Методы обнаружения, визуализации и удаления более сложных циклов, а также инструментальная поддержка таких методов заслуживают рассмотрения в отдельной статье.

## III. МАТРИЦА СТРУКТУРЫ ЗАВИСИМОСТЕЙ ПАКЕТОВ

На рисунке 4 представлена простейшая матрица структуры зависимостей. На этой двоичной матрице показано лишь наличие или отсутствие связей между элементами программной системы.

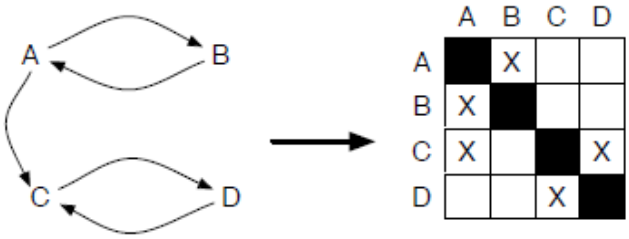


Рис.4. Пример двоичной матрицы структуры зависимостей.

При наличии зависимости (какого либо рода ссылки) элемента в колонке от элемента в строке, на пересечение колонки и строки проставляется маркер. Например, для рисунка 4 класс из пакета A в программе на языке Java импортирует по какой либо причине класс из пакета C. В этом случае зависимость между классами "наследуется" содержащими классы пакетами.

В сложных системах количество анализируемых с помощью матрицы зависимостей элементов программы (библиотечных исполняемых файлов, пакетов языка Java) может достигать нескольких десятков. В таком случае, отслеживание зависимостей при помощи приведенной на рисунке 4 простейшей матрицы бывает затруднено. Упростить понимание зависимостей может раскраска элементов матрицы. Две ячейки матрицы для взаимозависимых элементов окрашиваются в красный цвет, зависимый элемент окрашивается в зеленый цвет, а элемент, от которого зависят, в синий. Матрица, представленная на рисунке 4, в раскрашенном виде показана на рисунке 5.

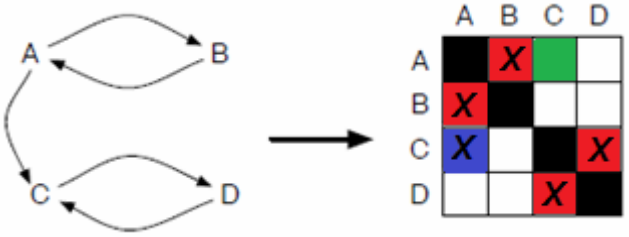


Рис.5. Пример раскрашенной матрицы структуры зависимостей.

На рисунке 6 показан фрагмент матрицы зависимости пакетов содержащихся в библиотеке Java Development Tools (JDT), насчитывающей, с учетом вложенности, 83 пакета классов.

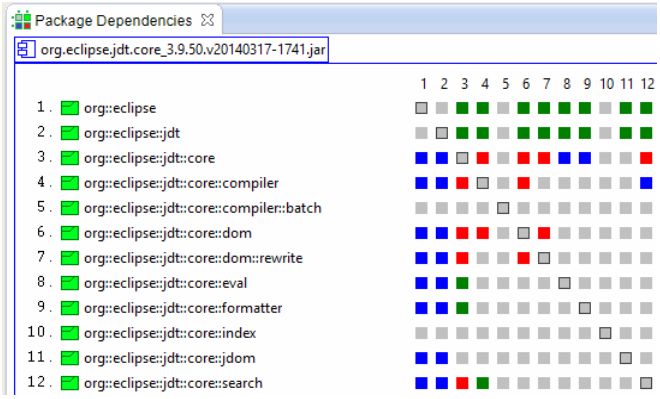


Рис.6. Пример раскрашенной матрицы структуры зависимостей для библиотеки Java Development Tools.

Для упрощения визуального анализа объемных матриц возможно их структурирование. Простейший способ - из рассмотрения могут быть убраны вложенные пакеты, и далее рассматриваться лишь зависимости между пакетами верхнего уровня. Для некоторых таких пакетов верхнего уровня циклическая зависимость с другими пакетами верхнего уровня будет отсутствовать, а удаление циклов внутри пакетов верхнего уровня может представлять отдельную, и возможно, простую задачу.

Существенно упростить просмотр объемных матриц могут контекстно-зависимые всплывающие подсказки для ячеек матрицы, как показано на рисунке 7.

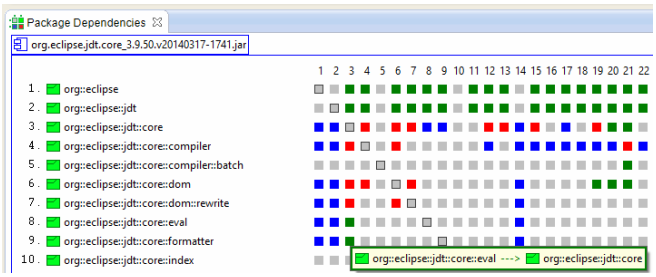


Рис.7. Пример контекстно-зависимых всплывающих для ячеек матрицы.

На рисунке 7 всплывающая подсказка в зеленой рамке (поскольку подсказка для зеленой ячейки) показывает, что пакет *org::eclipse::jdt::core::eval* зависит (направление стрелки в сторону пакета от которого зависит) от пакета *org::eclipse::jdt::core::core*. Наличие таких подсказок позволяет просматривать матрицу и в случае, если имена строк и колонок при "листании" матрицы будут не видны.

Для оптимизации восприятия зависимостей элементы, которые зависят от других элементов, смещаются вверх/влево, а элементы, от которых зависят, смещаются вниз/вправо. Таким образом, пакеты ядра программы оказываются справа и внизу матрицы, а пакеты верхнего уровня оказываются справа вверху.

Для оптимизации матрицы могут быть использованы алгоритмы разделения (partition) [22] и алгоритмы кластеризации матриц [23, 24]. В случае разделения матрицы строки и колонки матрицы перемещаются (множество отображаемых пакетов сортируется) таким

образом, что ячейки зависимых пакетов перемещаются в правый верхний угол. Получается правая треугольная матрица, как показано в примере на рисунке 8.

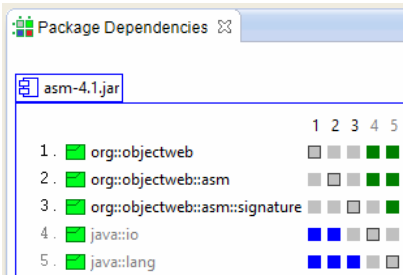


Рис.8. Пример треугольной матрицы зависимости после применения операции разделения.

Простейшие циклы, когда два пакета зависят друг от друга непосредственно, показываются в матрице красным цветом. В больших системах возможны и более сложные циклы с зависимостями - посредниками. Так, например, на рисунке 9 показана зависимость пакетов A -> B->C->A.

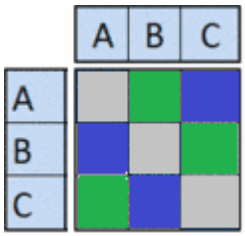


Рис.9. Пример сложного цикла зависимостей.

В рассматриваемом случае зависимость B->C является зависимостью - посредником.

Для поиска сложных циклов используется алгоритм, описанный в работе [25].

Важно отметить, что приведенные в качестве примеров матрицы структурной зависимости показывали лишь наличие зависимостей между парами пакетов. Какова причина существующей между пакетами зависимости на матрице не показывалось. Также не показывалось, насколько сильна такая зависимость, сколько классификаторов (классов, интерфейсов и перечислений) со стороны каждого пакета в этой зависимости участвуют. Вместе с тем, такая информация имеет большое значение при реструктурировании пакетов с целью уменьшения зависимости пакетов, удаления таких ошибок проектирования, как циклы зависимостей, выявления уровневой организации сложной программной системы. Для детальной визуализации причин и степени зависимости пакетов необходим отдельный вид, показывающий содержимое ячейки-зависимости. Вместе с тем с самой матрице структурной зависимости необходимо показать, какие ячейки заслуживают детального рассмотрения.

Частично перечисленные задачи визуализации могут быть решены простановкой числовых значений в ячейки матрицы. На рисунке 10 в ячейках матрицы показано количество участвующих в отношении зависимости классификаторов.

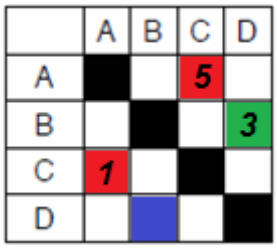


Рис.10. Оценка степени зависимости пакетов с помощью числовых значений.

Между пакетами A и C на рисунке 10 существует взаимозависимость, что показано в матрице ячейками красного цвета. В пакете A пять классов зависят от классов в пакете C. В пакете C один класс зависит от классов в пакете A. Возможно перемещение этого класса из пакета C в пакет A приведет к исчезновению взаимозависимости пакетов, что позволит структурировать программную систему на уровни.

Визуализация количества классификаторов возможна как один из возможных задаваемых пользователем режимов визуализации матрицы зависимости. Вместе с тем визуализация количества классификаторов участвующих в зависимости теряет свою привлекательность при большом количестве элементов матрицы зависимости. Существенно затрудняется восприятие такой матрицы, если количество классификаторов в ячейках представляется двухзначными числами. Необходимо также цветом выделять зависимости, которые прямо или косвенно участвуют в циклических зависимостях.
Для визуализации степени прямой зависимости между двумя пакетами возможно использование оттенков красного цвета. При некотором заданном соотношении (например, один к трем) количества классификаторов в симметричных ячейках красным цветом раскрашивается ячейка с меньшим количеством классификаторов, как показано на рисунке 11.

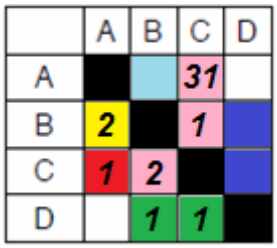


Рис.11. Расцветка степени прямой и косвенной зависимости пакетов.

В этом случае выделенная красным цветом зависимость пакета *C* от пакета *A*, может быть рассмотрена более детально в отдельном виде для анализа причин зависимости и ее возможного удаления.

Помимо прямых взаимозависимостей, между пакетами могут быть и косвенные зависимости через пакеты - посредники. Ячейки, зависимости в которых приводят к таким косвенным циклам, расцвечиваются желтым цветом (менее ярким, чем красный), а симметричные им ячейки - голубым цветом (менее

ярким, чем синий). Так на рисунке 11 существует цикл C->B->A->C. Поэтому зависимость B->A показана в матрице желтым цветом.

## IV. Оценка причины зависимости двух пакетов

Для визуализации содержимого ячейки матрицы структурной зависимости (зависимости между двумя пакетами в матрице) используется отдельное изображение, состоящее из четырех расположенных вертикально панелей. Принцип формирования такого изображения поясняется с помощью рисунка 12.

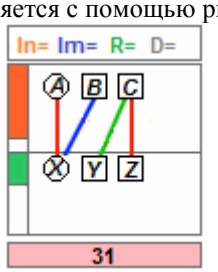


Рис.12. Принцип формирования вида для ячейки матрицы структуры зависимости.

Первая панель - для визуализации количества отношений разного рода между двумя пакетами. Вторая и третья панель показывают классификаторы двух пакетов и отношения между этими классификаторами. Последняя панель совпадает с визуальным представлением ячейки в матрице зависимости.

Зависимость между пакетами возникает как следствие зависимостей между классификаторами внутри пакетов. Причина существования зависимостей - различные отношения между классификаторам: отношения наследования между классами и между интерфейсами, реализация классами интерфейсов, ссылка в одном классификаторе на экземпляр другого классификатора, вызов метода одного классификатора в другом классификаторе, а также ряд других отношений зависимости. Верхняя панель вида предназначена для визуализации количества отношений каждого из видов. Например, количество отношений наследования (*Inheritance*) показывается с помощью строки **In=**, а количество отношений реализации (Implementation) - с помощью строки **Im=**. Отношение ассоциации языка UML (ссылка экземпляра одного классификатора на экземпляр другого классификатора) представлено в верхней панели строкой **R=**. В настройках визуализации возможно задание и более детального разделения отношений. Например, ссылки на экземпляр другого классификатора могут быть полем класса, параметром метода класса или локальной переменной метода класса. Таким же образом могут быть отдельно представлены разновидности отношения зависимости языка UML, показанные на рисунке 10 с помощью единой для всех разновидностей строки D=.

Далее, на второй и третьей панели изображения ячейки матрицы расположены классификаторы-клиенты (вверху) и классификаторы-поставщики (внизу), как показано на рисунке 12 и 13.

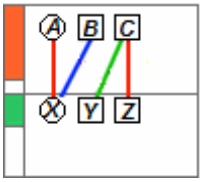


Рис.13. Панели классификаторов-клиентов и классификаторов-поставщиков.

Другими словами, вверху те, которые зависят, и внизу те, от которых зависят. Классификаторы показываются в этих панелях пиктограммами, в соответствии с графической нотацией языка UML: интерфейсы - кругами, классы - прямоугольниками, перечисления - прямоугольниками с двумя секциями. В компактном изображении панелей, классификаторы показываются только как пиктограммы, а их имена показываются как всплывающие подсказки. При переключении режима визуализации могут быть показаны как пиктограммы, так и имена классификаторов.

Для представления отношений между классификаторами в нотации языка UML существует стандартизованная графическая нотация. Однако, в нашем случае, для сохранения компактности изображения таких отношений используются простые линии (без стрелок), а разновидность отношения показывается цветом линии. Например, отношение наследования на рисунке 11 показывается красной линией между интерфейсами ***X*** и ***A***, а также классами ***Z*** и ***C***. Реализация классом ***B*** интерфейса ***X*** показывается синей линией. А ссылка из экземпляра класса *C* на экземпляр класса *Y* показана зеленой линией.

Для оценки степени вовлеченности классификаторов двух пакетов в отношения зависимости, на левой стороне второй и третьей панели показывается соотношение (оранжевой и зеленой полосой) количества клиентов и поставщиков к общему количеству классификаторов пакета. Цвет полосы зависит от количества вовлеченных в отношения классификаторов. Зеленый цвет - для небольшой вовлеченности, оранжевый для большой вовлеченности. Красный цвет - для 100% вовлеченности классификаторов, как сигнал о явной ошибке проектирования системы.

Для связи с матрицей структурной зависимости используется последняя панель - того же цвета, что ячейка в матрице структурной зависимости и с указанием количества зависимых классификаторов в ячейке. Эта панель может также использоваться как гиперссылка для открытия изображения матрицы содержащей эту ячейку.

## V. Заключение

В работе рассмотрены вопросы анализа и визуализации зависимостей между пакетами программных систем написанных на языке Java. Данная задача весьма актуальна для восстановлении архитектуры программной системы при решении задачи обратного проектирования. В статье рассмотрены методы визуализации таких зависимостей с помощью матриц структурной зависимости. Показано

использование данного и некоторых других методов визуализации зависимостей между пакетами в инструменте обратного проектирования и восстановления архитектуры. Данный инструмент основан на языке моделирования UML и реализован как расширение среды Eclipse. Настоящая статья является вводной для последующей работы о верификации проектов программных систем, обнаружении, визуализации и исправлении ошибок проектирования программных систем.

Статья является продолжением цикла публикаций по программной инженерии и применению UML, начатой в журнале INJOIT работами [13, 14, 15, 19, 21, 26, 27, 28]. Эта работа относится к числу одного из направлений исследований в Лаборатории ОИТ факультета ВМК МГУ [29, 30].

## Библиография


[1] Apache Maven, http://maven.apache.org/

[2] Ivy - The agile dependency manager, http://ant.apache.org/ivy/

[3] Maven architecture, http://maven.apache.org/ref/3.0.5/

[4] Aether- the library for working with artifact repositories, http://www.eclipse.org/aether/

[5] ISO/IEC/IEEE 42010:2011, Systems and software engineering - Architecture description, http://www.iso-architecture.org/ieee-1471/

[6] Nick Rozanski, Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives, Second Edition, Addison-Wesley Professional, October 25, 2011, ISBN 978-0-321-71833-4

[7] Paul Clements; Felix Bachmann; Len Bass; David Garlan; James Ivers; Reed Little; Paulo Merson; Robert Nord; Judith Stafford, Documenting Software Architectures: Views and Beyond, Second Edition Publisher: Addison-Wesley Professional, October 05, 2010, ISBN 978-0-321-55268-6

[8] Ducasse S, Gˆırba T, Kuhn A. Distribution map. Proceedings of 22nd IEEE International Conference on Software Maintenance (ICSM '06), IEEE Computer Society: Los Alamitos CA, 2006; 203–212.

[9] Ducasse S, Pollet D, Suen M, Abdeen H, Alloui I. Package surface blueprints: Visually supporting the understanding of package relationships. ICSM '07: Proceedings of the IEEE International Conference on Software Maintenance, 2007; 94–103.

[10] Dong X, Godfrey M. System-level usage dependency analysis of object-oriented systems. ICSM 2007, IEEE Comp. Society, 2007,

[11] Jannik Laval, Simon Denier, Stéphane Ducasse and Alexandre Bergel. Identifying cycle causes with Enriched Dependency Structural Matrix. In WCRE '09: Proceedings of the 2009 16th Working Conference on Reverse Engineering, Lille, France, 2009.

[12] N. Sangal, E. Jordan, V. Sinha, and D. Jackson. Using dependency models to manage complex software architecture. In OOPSLA'05, pages 167–176, 2005.

[13] Романов В.Ю. Инструмент обратного проектирования и рефакторинга программного обеспечения написанного на языке Java //International Journal of Open Information Technologies. – 2013. – Т. 1. – №. 8. – С. 1-6.

[14] Романов В.Ю. Моделирование свободно-распространяемого программного обеспечения с помощью языка UML //International Journal of Open Information Technologies. – 2013. – Т. 1. – №. 7. – С. 11-15.

[15] Романов В.Ю. Моделирование и верификация архитектуры программного обеспечения разработанного на языке Java. Сб. трудов VIII Международной конференции «Современные информационные технологии и ИТ-образование», Москва, 2013, с. 343-348

[16] Object Management Group, UML 2.4 Superstructure Specification, OMG document. http://www.omg.org/spec/UML/2.4.1/

[17] Object Management Group, UML Diagram Interchange, OMG document, http://www.omg.org/spec/UMLDI/1.0/PDF

[18] Object Management Group, XML Metadata Interchange, OMG document, http://www.omg.org/spec/XMI/2.4.1/

[19] Романов В.Ю. Визуализация программных метрик при описании архитектуры программного обеспечения //International Journal of Open Information Technologies. – 2014. – Т. 2. – №. 2. – С. 21-28.

[20] M.Lanza, R.Marinesku. Object-Oriented Metrics in Practice. Springer-Verlag. 2006.

[21] Романов В.Ю. Анализ объектно-ориентированных метрик для проектирования архитектуры программного обеспечения//International Journal of Open Information Technologies. – 2014. – Т. 2. – №. 3. – С. 11-17.

[22] J.N. Warfield. Binary Matrices in System Modeling. IEEE Transactions on Systems, Man, and Cybernetics, vol. 3, no. 5, pages 441–449, 1973.

[23] T.R. Browning. Applying the design structure matrix to system decomposition and integration problems: a review and new directions. IEEE Transactions on Engineering Management, vol. 48, no. 3, pages 292–306, 2001.

[24] Design Structure Matrix, http://www.dsmweb.org

[25] Robert Endre Tarjan. Depth-First Search and Linear Graph Algorithms. SIAM J.Comput., vol. 1, no. 2, pages 146–160, 1972.

[26] Романов В. Ю. Визуализация для измерения и рефакторинга программного обеспечения //International Journal of Open Information Technologies. – 2013. – Т. 1. – №. 9. – С. 1-10.

[27] Романов В. Ю. Использование шаблонов пакетов для анализа архитектуры программной системы //International Journal of Open Information Technologies. – 2014. – Т. 2. – №. 4. – С. 18-24.

[28] Романов В. Ю. Визуализация и анализ больших программных систем с помощью их трехмерного представления //International Journal of Open Information Technologies. – 2014. – Т. 2. – №. 5. – С. 1-9.

[29] Намиот Д., Сухомлин В. О проектах лаборатории ОИТ //International Journal of Open Information Technologies. – 2013. – Т. 1. – №. 5. – С. 18-21.

[30] Гурьев Д. Е., Намиот Д. Е., Шнепс М. А. О телекоммуникационных сервисах //International Journal of Open Information Technologies. – 2014. – Т. 2. – №. 4. – С. 13-17.

# Analysis and visualization of dependencies between packages of software systems

Romanov V.Y.

*Abstract* — **This article discusses software system architecture visualization as part of the tool for software reverse engineering and architecture recovery. The methods of visualization and analysis software packages with dependency structure matrices system and their cells are discussed. This research is a base for the next article about software design bugs detection, visualization and correction.**

*Keywords* — **software visualization, reengineering, dependency structural matrix, package, dependency, architecture recovery, reverse engineering.**