Визуализация программных метрик при описании архитектуры программного обеспечения

Романов В.Ю.

***Аннотация* — В статье приводится обзор методов визуализации результатов метрического анализа программных систем, используемых для извлечения описания архитектуры при решении задачи обратного проектирования. Визуализация программных систем с помощью графической нотации языка унифицированного языка моделирования UML повсеместно используется CASE-инструментами. Вместе с тем существует также множество апробированных и хорошо зарекомендовавших себя методов визуализации, которые могут дополнять графическую нотацию языка UML, либо быть интегрированы в нее. В статье приводится обзор таких методов визуализации, которые могут быть использованы при анализе и обратном проектировании программных систем.**

***Ключевые слова* — object oriented metrics, software visualization, metrics visualization, architecture recovery, reverse engineering..**

I. Введение

Разработка программных систем с помощью библиотек с исходными кодами получает все более распространение. Вместе с тем такой подход к разработке имеет и ряд недостатков. Большой объем исходных текстов этих библиотек, наличие большого числа библиотек со схожей функциональностью, наличие большого количества версий библиотеки, зачастую сопровождаемых различными группами разработчиков, существенно увеличивает время на изучение таких библиотек. Это делает актуальной задачу обратного проектирования (reverse engineering) для построения и визуализации модели такого программного обеспечения. Для этих целей повсеместное использование получил унифицированный язык моделирования UML[1]. Зачастую построение и визуализация модели всей библиотеки вручную трудоемко и нуждается автоматизации. Вместе с тем и автоматизированное построение визуального представления библиотеки в целом не решает проблемы, поскольку объем необходимой для просмотра информации по-прежнему чрезмерно велик. Необходимо построение и визуализация модели наиболее существенной для понимания части системы — ее архитектуры.

В результате совместной работы международных организаций по стандартизации ISO и IEEE был принят стандарт, определяющий, что может считаться описанием архитектуры программного обеспечения [2]. Были также изданы книги [3,4] являющиеся хорошими комментариями к данному стандарту. В этих книгах приведены также примеры графической нотации, которая может быть использована для визуализации.

При визуализации архитектуры программной системы может быть необходимым показать не только наличие взаимосвязей между элементами программы, но также и показать, почему возникли такие взаимосвязи и насколько такие взаимосвязи сильны. Поэтому в описании архитектуры могут быть полезны также диаграммы, показывающие на диаграмме значения метрик.

В предыдущей статье автора [5] рассматривались один из возможных способов визуализации — полиметрические виды. При этом способе визуализации программного обеспечения традиционная визуализация с помощью графической нотации языка моделирования UML (узлы и ребра графа) дополнялась новыми измерениями — цветом, шириной и высотой узлов графа на диаграммах, цветом и шириной ребер графа. Эти дополнительные к традиционной нотации измерения используются для представления на диаграмме значений объектно-ориентированных метрик и других свойств программных систем. В данной статье представлены методы визуализации программного обеспечения, которые не связаны с нотацией языка UML, а зачастую и вовсе не используют визуализацию с помощью графа.

II. Карта распределения

Результаты анализа программного обеспечения часто является разделением его элементов (классов, пакетов, интерфейсов) на новые группы или связывает эти группы посредством взаимоисключающих свойств. Бывает сложно понять как новые группы или свойства соотносятся с исходным разделением кода на классификаторы (классы, интерфейсы и перечисления) и пакеты. Визуализация с помощью карт распределения предназначена для решения таких проблем, позволяя показать распределение свойств среди кода элементов. Имена элементов программы могут быть представлены на диаграмме с помощью всплывающих подсказок.

На рисунке 1. показана карта распределения для графического редактора Jedit [6]. Красным цветом, в частности, на рисунке показаны классы ядра, представляющие классы и интерфейсы расположенные в

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

верхней части иерархии наследования. Голубым цветом показаны классы, представляющие собой интерфейс пользователя. Зеленым цветом показываются внутренние классификаторы редактора с различной видимостью классификатора.

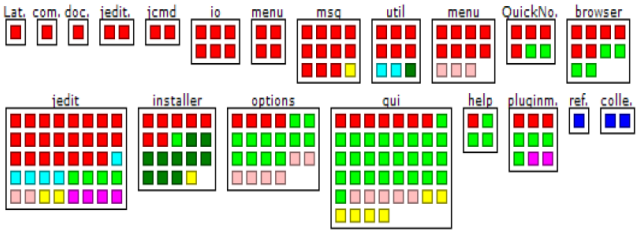

Рис.1. Карта распределения для редактора JEdit.

В дополнение к цвету, на картах распределения могут использоваться и размерности показываемых элементов. Так, например, на рисунке 2 показываются метрики классов и интерфейсов пакете, показывающие размер кода в байтах (высота элемента) и количество методов в классе или интерфейсе (ширина элемента).

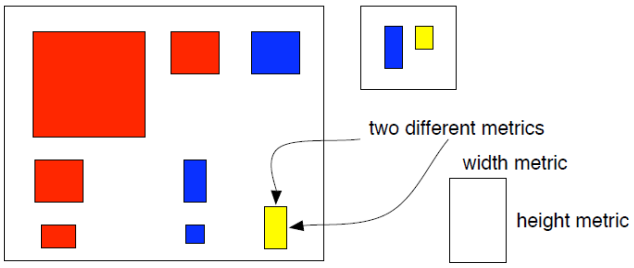

Рис.2. Использование ширины и высоты для представления на карте распределения метрик.

Карты распределения являются достаточно гибким способом визуализации и применимы к папкам/файлам, пакетам/классам, классам/методам и другим группам. Отображаемые свойства элементов должны быть взаимно исключающими. Для навигации по картам распределения можно использовать вложенность таких карт, перемещаясь, например, от карты показывающей вложенность исполняемых файлов в папки к карте показывающей вложенность в файл пакетов языка Java.

III. Деревья – карты

Визуализация структуры программного обеспечения с помощью деревьев известна давно и получила широкое распространение [7,8,9]. Способ визуализации с помощью более компактных карт деревьев позволяет более рационально использовать пространство экрана. Все дерево в этом случае представляется прямоугольником. Каждое поддерево дерево представляется прямоугольником в родительском прямоугольнике. Первый уровень в иерархии представляется с помощью вертикального расщепления родительского прямоугольника. Второй уровень расщепляет родительский прямоугольник горизонтально. Такое чередование вертикального и горизонтального расщепления продолжается при перемещении на следующие уровни иерархии. Рисунок 3 иллюстрирует такой компактный способ визуализации дерева.

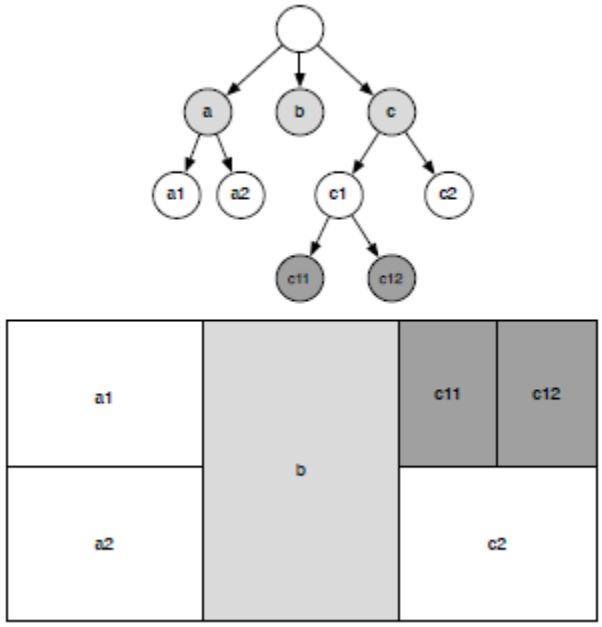

Рис.3. Принцип конструирования дерева-карты.

Пример использование карты дерева для представления дерева, состоящего из элементов Java - программы: пакетов, классов и методов приведен на рисунке 4.
Весь прямоугольник в целом представляет пакет языка Java. Вложенные прямоугольники первого уровня с белыми границами — классы языка Java. Вложенные прямоугольники представляют методы класса. С помощью цвета методов показываются значения метрик, вычисленных для этих методов.

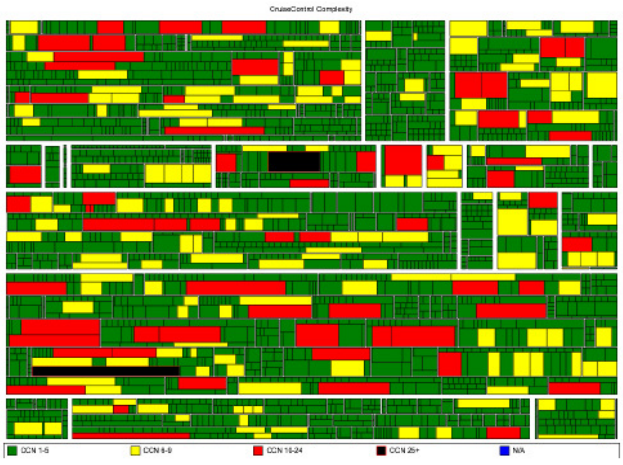

Рис.4. Визуализация структуры Java программы с помощью карты дерева.

Для представления значений метрик возможно использование не только цвета, но также ширины и высоты прямоугольников. В этом случае необходимо использование более сложных алгоритмов для компактного заполнения пространства родительского прямоугольника.

IV. Деревья – кольца

Этот способ визуализации деревьев используется для более компактного их представления на экране и упрощения навигации по ним. Деревья — кольца показывают топологию и размер узлов дерева [10]. Пример использования такого дерева приведен на рисунке 5.

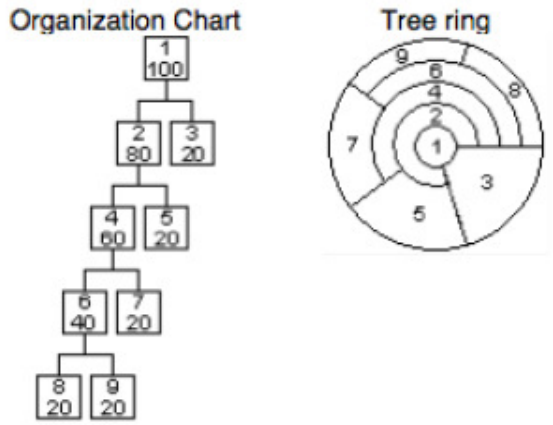

Рис.5. Принцип конструирования деревьев-колец.

Размер узла в этом случае определяется углом сектора занимаемого на диаграмме. На дереве-кольце можно показать с помощью цвета и размера узла две метрики.
К недостатку данного метода визуализации следует отнести необходимость интерактивного масштабирования диаграммы, что бы оценить узлы расположенные ближе к центру. Достоинством такого способа визуализации, по сравнению с деревьями-картами, является более ясное представление вложенности элементов программы.

V. Ступенчатый чертеж

Ступенчатый чертеж (Icicle plot), как и два предыдущих способа визуального представления, используется для визуализации деревьев [7]. Каждая «строка» чертежа представляет собой уровень дерева, как показано на рисунке 6.

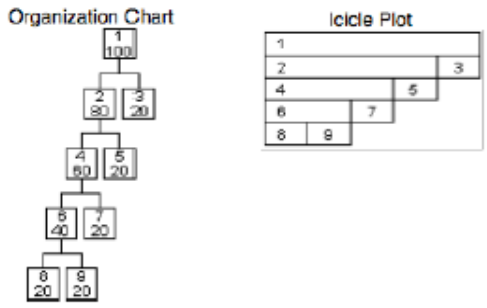

Рис.6. Принцип конструирования ступенчатого чертежа.

Каждая строка разбивается в соответствии с числом узлов-потомков у поддерева. В отличие от деревьев-колец, ступенчатый чертеж может содержать пустое пространство. Размер и цвет узла в каждой линии могут представлять значения метрик для этого. Этот способ представления лучше показывает структуру дерева, чем дерево-карта, но имеет меньшее количество представляемых метрик.

VI. Полиметрические виды

Полиметрические виды [5, 11] позволяют расширить дополнительными размерностями визуализацию программы с помощью графа, позволяющими представить также и значения метрик для элемента графа. Для представления метрик узлом графа могут использоваться координаты *x* и *y* узла, ширина и высота узла, а также цвет узла. Для представления метрик с помощью ребра графа могут использоваться цвет и толщина этого ребра.

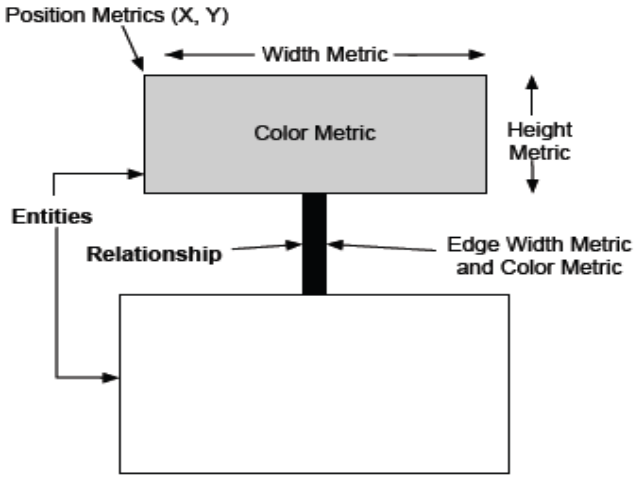

Рис.7. Принцип конструирования полиметрического вида

Пример использования полиметрического вида для одновременной визуализации трех метрик приведен на рисунке 8. Прямоугольниками на диаграмме представлены классы. Эти прямоугольники связаны отношениями наследования. Ширина прямоугольника представляет значение метрики NOA (Number of Attributes) — количество атрибутов класса. Высота прямоугольника представляет значение метрики NOM (Number of Methods) — количество методов класса. Цвет прямоугольника представляет значение метрики LOC (Line of Code) — количество строк кода в исходных текстах класса.

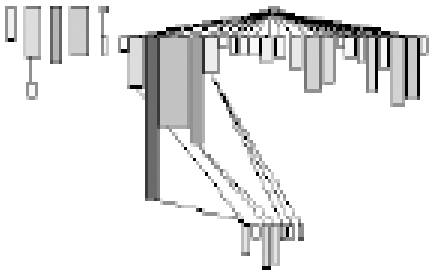

Рис.8. Визуализация значения метрик NOA, NOM и LOC

Более подробное описание использования полиметрических видов для обнаружения дефектов

программного обеспечения можно найти в статье [5].

VII. Визуализация файла кода

Визуальное представление «элементы файла» (File Dot) позволяет оценить содержимое файла, основываясь на элементах строк [12]. Файл представляется как сетка, состоящая из квадратов. Каждый квадрат представляет строку файла с исходными текстами программы. Такое представление может быть использовано для визуализации значений метрик при оценке сложности кода или визуализации управляющей структуры кода. Пример визуализации управляющих конструкций программы приведен на рисунке 9.

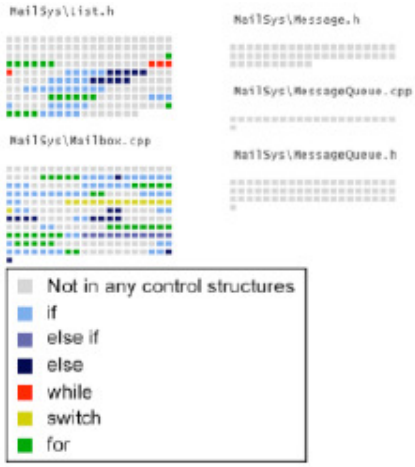

Рис.9. Визуализация управляющих конструкций в файлах.

Описанный способ визуализации файла кода может применяться также для представления авторства кода, кода модифицирующего переменные или только использующего их, кода имеющего доступ к глобальным переменным.

VIII. Полярная диаграмма

Полярная диаграмма (Kivat chart, Radar chart) предназначена для одновременной визуализации нескольких свойств элемента программной системы [13, 14]. Из центра диаграммы (полюса) рисуется несколько осей координат, как показано на рисунке 10.

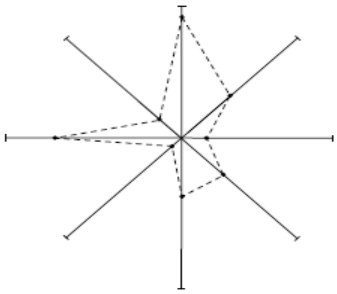

Рис.10. Принцип формирования полярной диаграммы.

По каждой оси откладывается значение метрики. Минимальные значения помещаются в центре диаграммы. Максимальные значения на конце оси координат. Значения по каждой оси масштабируются линейно. Затем значения на каждой из осей соединяются ломаной линией. На рисунке 11 показана визуализация значений метрик используемых для оценки исходного кода программы. В частности, на этой диаграмме показаны количество функций в классе, количество строк кода, количество комментариев и операторов.

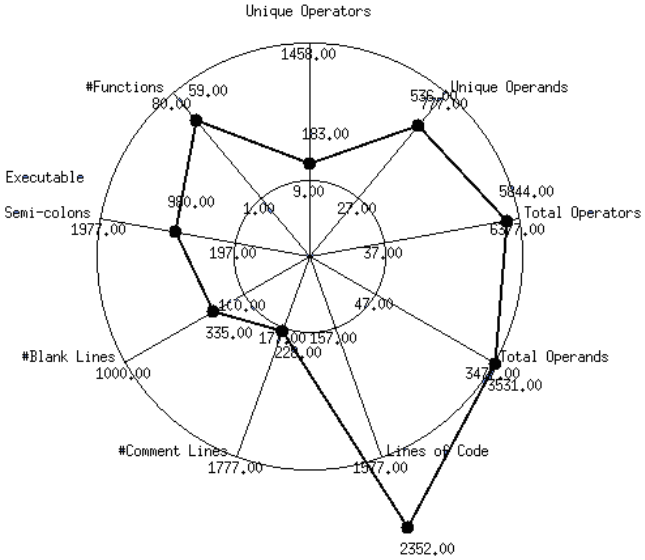

Рис.11. Визуализация метрик с помощью полярной диаграммы.

Полярные диаграммы удобны также для визуализации эволюции программного обеспечения. На рисунке 12 показаны изменения значения метрик для модуля DOM браузера Mozilla [14].

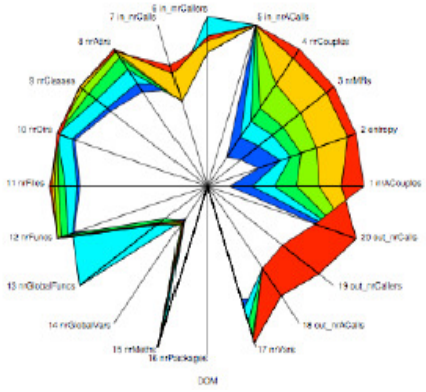

Рис.12. Визуализация эволюции модуля DOM браузера Mozilla.

Полярную диаграмму можно использовать и для сравнения на диаграмме значений метрик. Некоторые из метрик программной системы являются взаимосвязанными. Например, в каком либо из отношений класс/пакет может быть клиентом/поставщиком отношения. Класс может быть предком/потомком отношения наследования. Некоторые метрики вычисляются на основе количества таких отношений. Поэтому используемые для отображения значений метрик оси полярной диаграммы имеет смысл расположить симметрично (слева и справа). Полученное изображение значений метрик напоминает изображение бабочки, давшее название такого рода диаграммам [13].

На рисунке 13 показана диаграмма с такими симметрично отражаемыми метриками, показывающими взаимосвязь пакетов друг с другом.

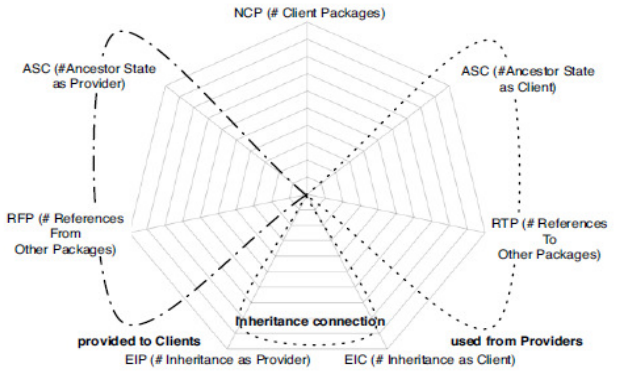

Рис.13. Визуализация с помощью диаграммы-бабочки отношений между пакетами программы

На диаграмме представлены значения следующих метрик:

ASC (Number of Ancestor State as Client). Количество доступов к переменным экземпляра родительского класса определенного в другом пакете.

RFP (Number of Class References From Other Packages). Количество ссылок на классы из классов, принадлежащих другим пакетам, на классы, принадлежащие анализируемому пакету.

EIC (Number of External Inheritance as Client). Количество отношений наследования, в которых анализируемый класс есть потомок, а родительский класс расположен в другом пакете.

EIP (Number of External Inheritance as Provider). Количество отношений наследования, где родительский класс в анализируемом пакете, а класс-потомок в другом пакете.

NCP (Number of Classes in a Package). Количество классов в пакете.

Левое крыло «бабочки» на диаграмме показывает, что пакет предоставляет другим пакетам. Правое крыло показывает, что пакет использует из других пакетов. Нижняя часть бабочки показывает, входят ли в пакет классы, имеющие потомков в других пакетах, а также являются ли классы пакета потомками классов из других пакетов.

IX. Точечная диаграмма

Часто оказывается необходима простая визуализация, позволяющая понять, в каких отношениях друг с другом находятся элементы программы. Недостатком такого способа визуализации может быть размер диаграммы при большом количестве взаимосвязанных элементов. Однако для визуализации взаимосвязей между элементами программы верхнего уровня (например, между пакетами программы на языке Java [15]), данный способ визуализации может быть приемлем.

На рисунке 14 показана диаграмма взаимосвязей между пакетами, возникающая в результате вызова методов классов расположенных в различных пакетах.

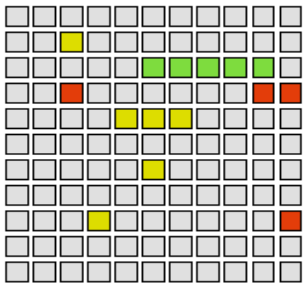

Рис.14. Визуализация с помощью точечной диаграммы отношений между пакетами программы

Для навигации с помощью этой диаграммы могут использоваться имена как строки и колонки матрицы, так и элементы матрицы.

Точечные диаграммы удобны для визуализации циклических зависимостей между пакетами, реализации алгоритмов выделения уровней пакетов в программной системе и алгоритмов удаления циклических зависимостей между пакетами [16]. На рисунке 15 представлена точечная диаграмма, показывающая пакеты с циклической зависимостью и без такой зависимости.

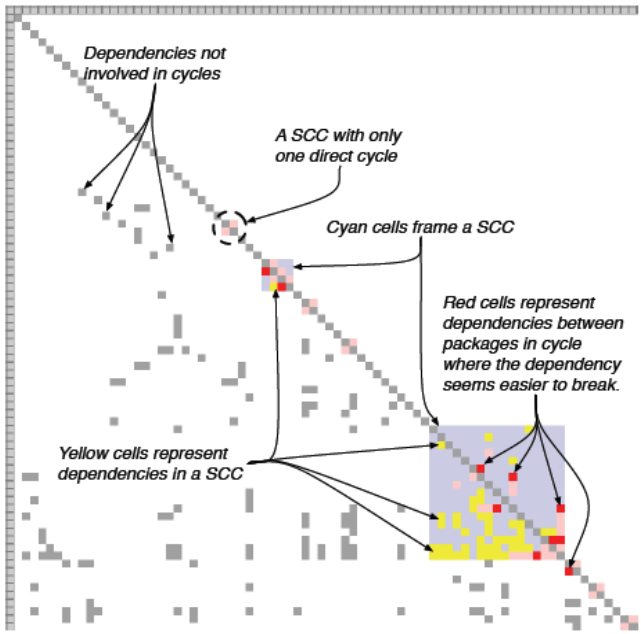

Рис.15. Выявление циклических зависимостей между пакетами.

X. Матрицы эволюции

Появление большого количество свободно-распространяемых в исходных кодах программных систем, расположенных в публичных репозиториях, которые поддерживают системы управления версиями с программным интерфейсом, дало возможность отслеживать эволюцию архитектуры программных систем. Для визуализации эволюции программной системы в визуализацию включается дополнительная размерность — версия системы. Это позволяет оценить изменения в системе при проходе системы через различные стадии разработки и рефакторинга. Назначение матрицы эволюции показать изменения для

каждого типа элемента — пакета, класса, интерфейса. На рисунке 16 показана структура матрицы эволюции [17]:

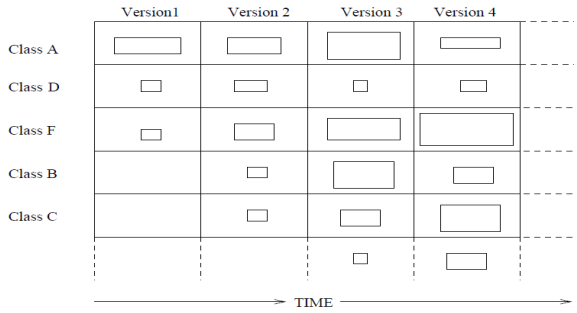

Рис.16. Структура матрицы эволюции.

По вертикальной оси откладываются элементы программной системы, по горизонтальной оси — ее версии. Каждая ячейка — элемент программной системы. Для различного типа элементов (пакеты, классы) строятся различные матрицы. Элементы в каждом столбце упорядочены в порядке создания. Размеры и цвет элемента используются для отображения значений метрик. Пустой элемент матрицы означает, что этот элемент в этой версии системы еще не создан или уже удален. Пример отображения стадий разработки программной системы с помощью матрицы эволюции приведен на рисунке 17:

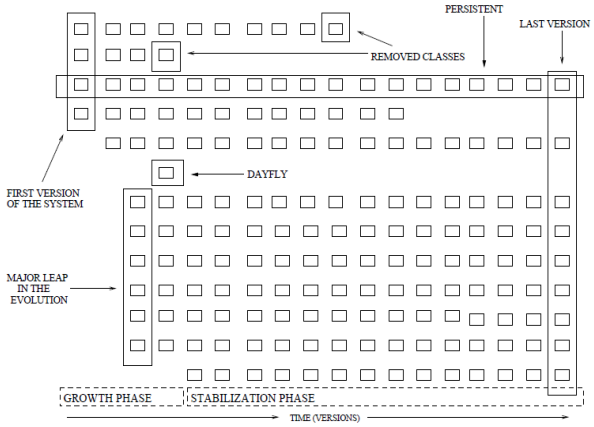

Рис.17. Стадии разработки системы на матрице эволюции.

В левом верхнем углу прямоугольником показана первая версия системы. После этого разработка системы переходит в фазу роста. Появляется основной срез (leap) этой фазы эволюции. После этого наступает фаза стабилизации, в ходе которой выполняется рефакторинг системы и удаление некоторых из ее элементов. Удаленные элементы располагаются в верхней части матрицы. Последняя версия системы показывается самой колонкой матрицы.

XI. Заключение

Статья является продолжением цикла публикаций по программной инженерии и применению UML, начатой в журнале INJOIT работами [5,18,19], а также отраженной в более ранних публикациях [20, 21]. Эта работа относится к числу одного из направлений исследований в Лаборатории ОИТ факультета ВМК МГУ [22].

Библиография

[1] Object Management Group, UML 2.4 Superstructure Specification, OMG document. http://www.omg.org/spec/UML/2.4.1/
[2] ISO/IEC/IEEE 42010:2011, Systems and software engineering — Architecture description, http://www.iso-architecture.org/ieee-1471/
[3] 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
[4] 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
[5] Романов В. Ю. Визуализация для измерения и рефакторинга программного обеспечения //International Journal of Open Information Technologies. – 2013. – Т. 1. – №. 9. – С. 1-10.
[6] Stephane Ducasse, Tudor Girba, and Adrian Kuhn. Distribution Map. In Proceedings of 22nd IEEE International Conference on Software Maintenance (ICSM '06), p.203-212, IEEE Computer Society, Los Alamitos CA, 2006.
[7] Todd Barlow and Padraic Neville. A comparison of 2-d visulization of hierarchies. In Proceedings of the IEEE Symposium on Information Visualization 2001 (INFOVIS'01), 2001.
[8] Guillaume Langelier, Houari Sahraoui, and Pierre Poulin. Visualizationbased analysis of quality for large-scale software systems. In ASE '05: Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering, pages 214–223, New York, NY, USA, 2005. ACM.
[9] Richard Wettel and Michele Lanza. Program comprehension through software habitability. In Proceedings of ICPC 2007 (15th International Conference on Program Comprehension), pages 231–240. IEEE CS Press, 2007.
[10] John T. Stasko, Richard Catrambone, Mark Guzdial, and Kevin Mcdonald. An evaluation of space-filling information visualizations for depicting hierarchical structures. International Journal Humain-Computer Studies, 53(5):663–694, 2000.
[11] Michele Lanza and Stephane Ducasse. Polymetric views—a lightweight visual approach to reverse engineering. Transactions on Software Engineering (TSE), 29(9):782–795, September 2003.
[12] Abram Hindle, Michael W. Godfrey, and Richard C. Holt. Reading beside the lines: Indentation as a proxy for complexity metrics. In ICPC '08: Proceedings of the 2008 The 16th IEEE International Conference on Program Comprehension, pages 133–142, Washington, DC, USA, 2008. IEEE Computer Society.
[13] Stephane Ducasse, Michele Lanza, and Laura Ponisio. Butterflies: A visual approach to characterize packages. In Proceedings of the11th EEE International Software Metrics Symposium (METRICS'05), pages 70–77. IEEE Computer Society, 2005
[14] Martin Pinzger, Harald Gall, Michael Fischer, and Michele Lanza.. Visualizing multiple evolution metrics. In Proceedings of SoftVis 2005 (2nd ACM Symposium on Software Visualization), pages 67–75, St. Louis, Missouri, USA, May 2005.
[15] Neeraj Sangal, Ev Jordan, Vineet Sinha, and Daniel Jackson. Using dependency models to manage complex software architecture. In Proceedings of OOPSLA'05, pages 167–176, 2005.
[16] Jannik Laval and Stephane Ducasse, Resolving cyclic dependencies between packages with Enriched Dependency Structural Matrix, Software: Practice and Experience, 2012.
[17] Michele Lanza and Stephane Ducasse, Understanding Software Evolution Using a Combination of Software Visualization and Software Metrics, Proceedings of Langages et Modeles a Objets (LMO'02), 135–149, Lavoisier, 2002
[18] 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.
[19] 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.
[20] 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
[21] Романов В.Ю. Реализация метамодели языка UML на основе хранилища данных фирмы Google. Сб. трудов VII Международной научно-практической конференции "Современные информационные технологии и ИТ-образование". М.:, 2012. с.605-610.
[22] Намиот Д., Сухомлин В. О проектах лаборатории ОИТ //International Journal of Open Information Technologies. – 2013. – Т. 1. – №. 5. – С. 18-21.

Software Metrics Visualization for Software architecture description

Romanov V.Y.

***Abstract* — the article makes review of visualization methods for software object oriented metrics. The metrics visualization methods may be used during the reverse engineering and architecture recovering for programming systems.**

***Keywords* — object oriented metrics, software visualization, metrics visualization, architecture recovery, reverse engineering.**