Визуализация и анализ больших программных систем с помощью их трехмерного представления

Романов В.Ю.

Аннотация. **В статье рассматриваются способы визуализации кода программных систем с использованием изображения кода в трехмерном пространстве. Рассматриваются изображения, основанные на метафоре представления программной системы как города со зданиями и районами, а также дополнение третьим измерением традиционного представления программной системы как графа - UML диаграммы.**

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

I. Введение

Задача обратного проектирования (reverse engineering) программной системы является весьма важной при разработке программной системы с использованием библиотек с исходными кодами. Построение и визуализация UML-модели для вновь разрабатываемой программной системы, так и для используемых ею библиотек существенно упрощает понимание их структуры и функциональности, выбор необходимой версии и разработчика библиотеки. Способы построения и визуализация модели программного системы были рассмотрены в предыдущих работах автора[1][2][3]. Объем информации получаемой при решении этих задач может быть слишком велик для их восприятия человеком, а получение и визуализация всех связей может требовать чрезмерно большого времени. Поэтому визуализация программной системы необходима лишь для наиболее существенной ее части - архитектуры. Для построенной UML-модели необходимо вычисление и визуализация значений объектно-ориентированных метрик, позволяющих оценить качество проектирования системы. В предыдущих работах автора особо рассматривались способы визуализации архитектуры системы [4] и визуализации результатов измерения качества программной системы с помощью объектно-ориентированных метрик [5]. В работе [6] сделан обзор и анализ объектно-ориентированных метрик. Рассмотрены простейшие объектно-ориентированные метрики для анализа проектирования отдельных классов. Затем рассмотрены метрики связанности класса, позволяющие оценить качество проектирования структуры класса. В работе автора [7] рассмотрен анализ архитектуры программной системы на основе шаблонов пакетов языка Java. В работах [4,5] рассматривались визуализация архитектуры и объектно-ориентированных метрик с помощью уже хорошо известных двумерных изображений. В данной работе рассматриваются возможности использования для визуализации программных систем трехмерных изображений.

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

II. Визуализации Программы как графа в трехмерном пространстве

Попытка использовать третье измерение для визуализации программного обеспечения предпринимались и ранее[8][9][10]. Однако зачастую такая визуализация на самом деле представляла программу как привычный граф с узлами и ребрами, расположенными в трехмерном пространстве. Для обозрения таких графов обозреватель мог свободно перемещаться в трехмерном пространстве. Так, в работе [8] структура вложенных пакетов для программы, написанной на языке Java, представлялась как в виде вложенных сфер расположенных в трехмерном пространстве, как показано на рисунке 1.

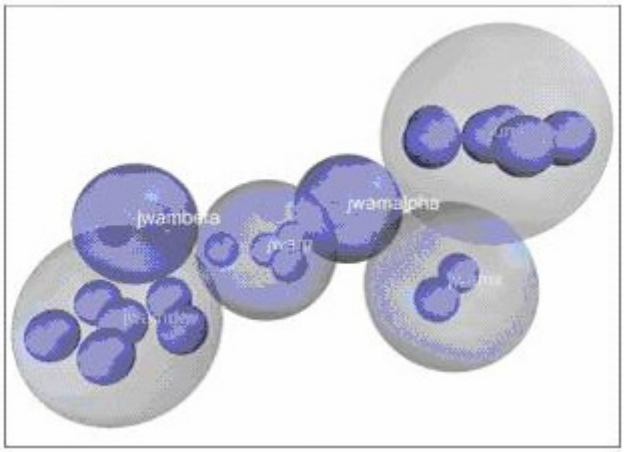


Рис.1. Ландшафт. Представление иерархии пакетов как вложенных сфер.

Такое расположение предлагалось интерпретировать как искусственно сформированные ландшафты. Для исследования такой структуры пакетов обозреватель свободно перемещался в трехмерном пространстве. Содержимое таких пакетов (классы и интерфейсы) представлялось как расположенная в сфере плоскость с расположенными на ней зданиями, как показано на рисунке 2.

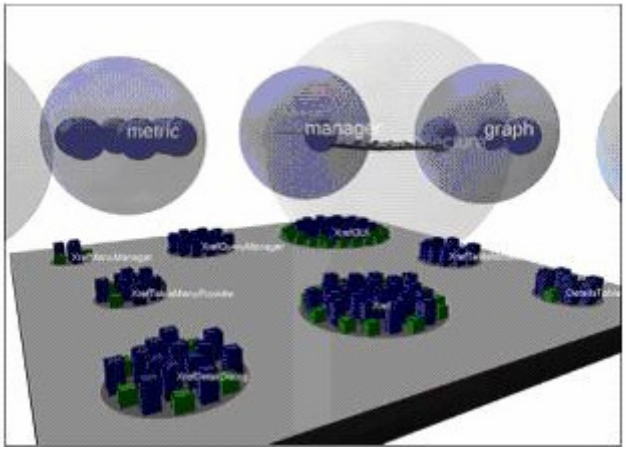


Рис.2. Ландшафт. Представление классов и интерфейсов пакета.

Представление отношений между элементами программной системы, наследование, реализация классами интерфейсов, использование классов и интерфейсов как типов полей классов и параметров методов представлялось как обобщенные связи. Пример таких связей показан на рисунке 3.

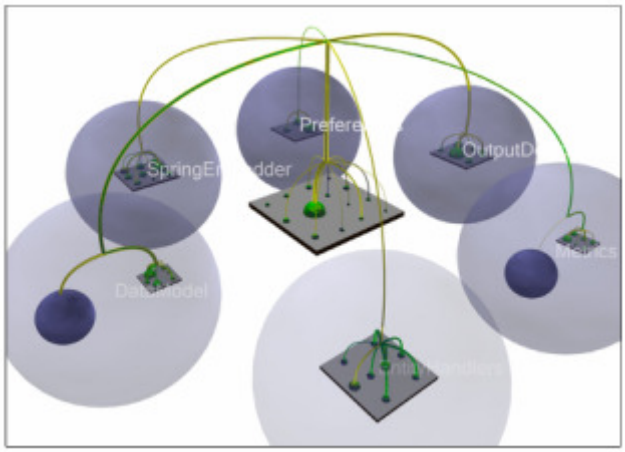


Рис.3. Ландшафт. Представление отношений между элементами программной системы.

Для более детальной информации об отношениях предлагалось использовать фильтры и раскраску определенным цветом отношения конкретного вида, например отношения наследования.

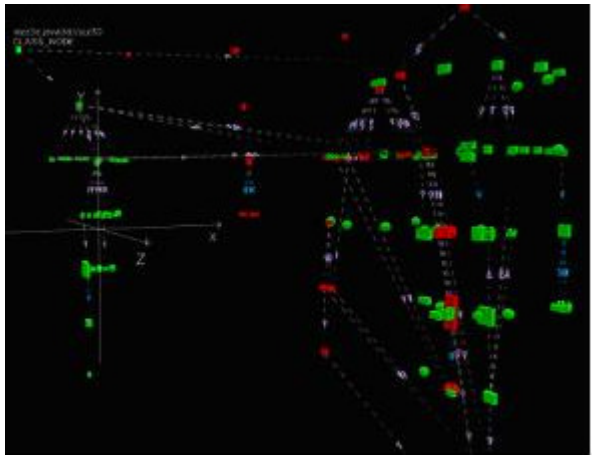


Рис.4. Представление отношений наследования в трехмерном пространстве системой Vizz3D.

В системе Vizz3d [9] для визуализации отношений между элементами программы с помощью графа размещенного в трехмерном пространстве, как показано на рисунке 4. Для детального рассмотрения отношений в системе использовалось вращение системы координат. Узлы-пакеты этого графа (вместе с их содержимым) могли быть сжаты и раскрыты. Таким образом, достигался необходимый уровень детализации.

III. Визуализация Программа как города

Использование трехмерного пространства для представления искусственно созданных ландшафтов и трехмерных графов не позволяет обозревателю программной системы ориентироваться в трехмерном пространстве привычным для него образом, что затрудняет понимание структуры и взаимосвязей в системе. В следующем способе визуализации предполагается использовать уже имеющиеся навыки ориентации в трехмерном пространстве.

A. Метафора представления кода программы как города

Данная метафора используется для визуализации объектно-ориентированных программ для упрощения понимания таких конструкций, как пакеты, классы, методы и атрибуты классов, их явные и неявные отношения. Классы, согласно этой метафоре, представляются как здания, размещенные в районах города - пакетах.
Использование такой метафоры существенно упрощает понимание программы. Ориентированные снизу вверх знания с районами и пригородами задают обозревателю знакомый контекст, упрощающий ориентацию в изображении программы.

Полученное представление программы с ясной ориентацией в ней позволяет перемещаться по программе и взаимодействовать с ее изображением. В свою очередь, после этого возникают следующие вопросы:

1. Как использовать данное изображение программы для представления объектно-ориентированных метрик этой программы?
2. Каким образом необходимо задать планировку и топологию в изображении программы-города? В реальном городе эти характеристика фиксированы, а в программном городе эти характеристики можно менять.
3. Каким образом можно взаимодействовать с таким изображением программы?
4. Каким образом можно перемещаться по программе-городу?

В системе CodeCity [10] изображение программы-города использовалось для визуализации объектно-ориентированных метрик, как показано на рисунке 5. В этом примере для классов и интерфейсов показаны значения метрик NOM (Number Of Methods) и NOA (Number Of Attributes).

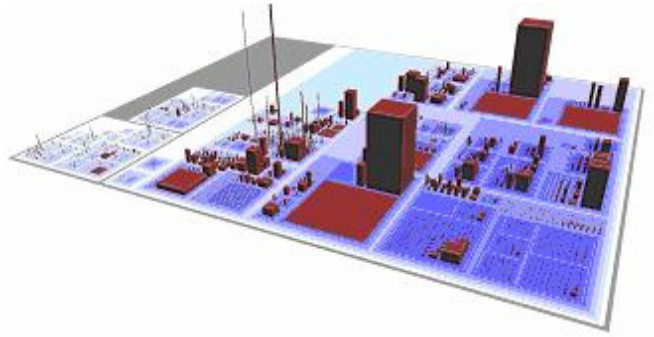


Рис.5. Пример программы-города с визуализацией метрик NOM и NOA.

Классы и интерфейсы (здания) программы показаны красным цветом. Пакеты (районы) синим цветом. Вложенность пакетов представлена яркостью синего цвета. Значение метрики NOM представляется на изображении высотой здания (класса или интерфейса), а значение метрики NOA представляется шириной и длиной здания. Как можно видеть на рисунке 1, в программе присутствуют интерфейсы (здания-шпили) и классы с большим числом атрибутов и небольшим числом методов (напоминают парковки в городе). Массивные здания, показанные на рисунке, представляют собой "божественные" классы, имеющие в этой программе избыточную ответственность. Представленная на рисунке 1 программа-город имеет два пригорода (стандартных библиотеки) для пространств имен *java* и *javax*.

Примеры способов представления топологии программы-города представлены на рисунках 6 и 7.
На рисунке 6, как рельеф местности, показана вложенность пакетов программы-города.

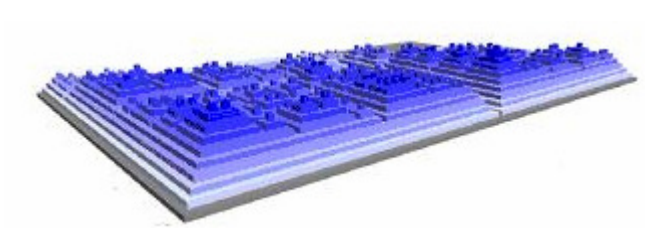


Рис.6. Пример представления топологии программы-города (вложенности пакетов) с помощью рельефа местности.

На рисунке 7 показан пример расположения зданий программы-города (классов и интерфейсов) на так такой рельефной местности.

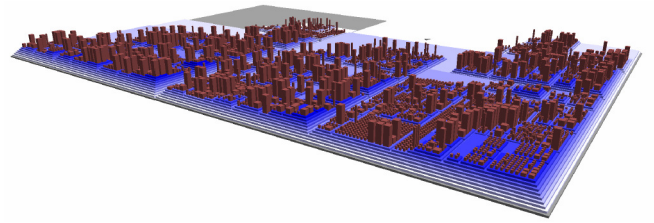


Рис.7. Пример представления топологии программы-города с помощью рельефа местности.

Планировка реального города (расположение зданий в районе), районов в городе, во многом определяется историческими причинами и не может быть легко изменена. В программе-городе такая планировка может быть легко изменена и оптимизирована. Для этого решается задача размещения максимального числа квадратов в квадрат минимального размера. Задача решается рекурсивно, начиная с пакетов наибольшего уровня вложенности. Первым размещаются прямоугольники наибольшего размера.

Способность перемещаться по изображению программы-городу и взаимодействовать с ним является критичной возможностью. Для взаимодействия с изображением можно использовать следующие действия:

Выбор. Любой элемент или группа элементов программы может быть выбран для последующего взаимодействия. Выбор может быть осуществлен вручную в результате нажатия мышью на элемент. Или же автоматически как результат выполнения запроса на поиск элементов обладающих некоторыми свойствами. Для выбранных элементов можно выполнить операции по добавлению или удалению элементов из выбранного множества, инвертирования выбора, очистки выбора. Так на рисунке 8 зеленым цветом показан выбранный район программы-города (пакет).

Порождение нового вида. Выбранные элементы могут быть использованы для порождения нового вида, позволяющего выполнить более детальное рассмотрение выбранных элементов. На рисунке 9 выбранный ранее на рисунке 8 район уже представлен как отдельный новый вид.

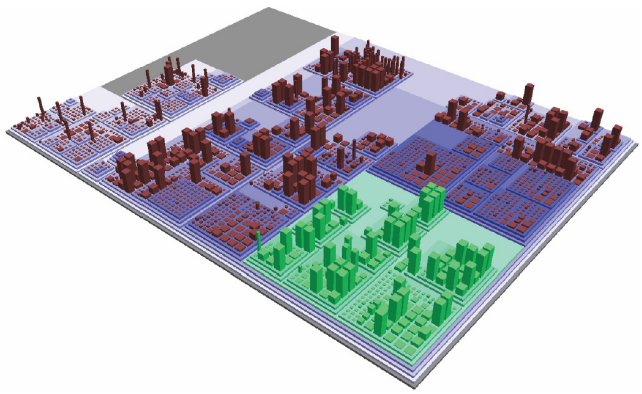


Рис.8. Выделенные элементы программы показаны зеленым цветом.

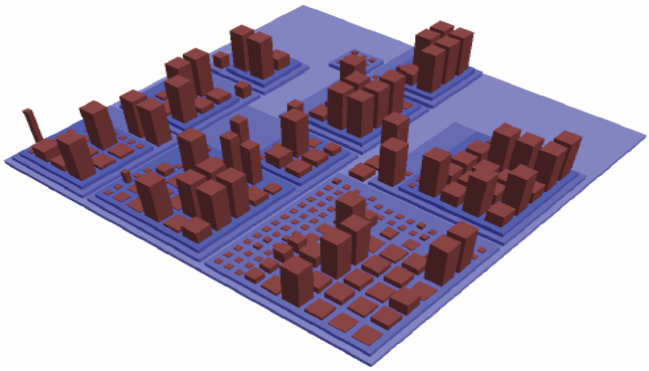


Рис.9. Порождение нового вида из сделанного выбора

Пометка. Можно пометить необходимым цветом множество зданий (классов, интерфейсов) в программе-

городе. Например, таким образом, могут быть помечены одним цветом классы, находящиеся в одной иерархии наследования. Имеется также возможность пометки зданий программы-города "прозрачностью". В этом случае контуры этих зданий заполняются не сплошной заливкой, а сеткой определенного цвета. Для прозрачности можно задавать коэффициент прозрачности, изменяя частоту сетки. Прозрачными можно делать здания (классы, интерфейсы), не обладающие определенными свойствами, и упрощая, таким образом, рассмотрение расцвеченных зданий обладающих заданными свойствами. На рисунке 6 показана программа-город со зданиями (классами, интерфейсами) помеченными цветом и прозрачностью.

Ручной выбор элементов зачастую может быть громоздким. Для упрощения выбора предоставляется средство для автоматического поиска, который позволяет выполнить поиск элементов программы с соответствующим именем (Name*), или типами (например, классы из определенного пакета-района), или категории (все корневые классы в иерархиях наследованиях), или элементы, связанные с выбранным элементом (все предки или потомки выбранного элемента).

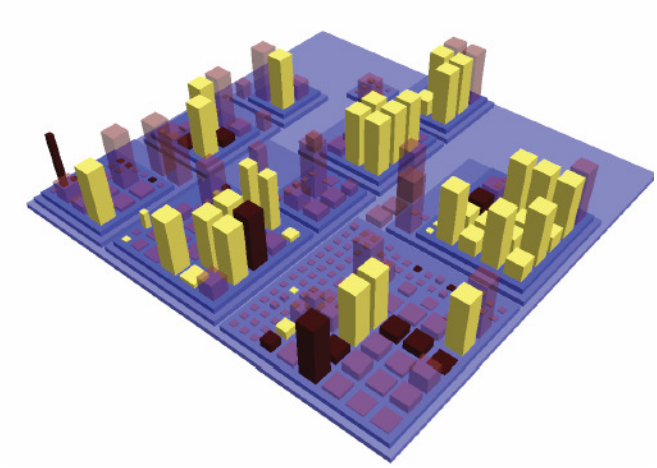


Рис.10. Программа-город со зданиями (классам, интерфейсами) помеченными цветом и прозрачностью.

Навигация. В некоторых 3-мерных системах визуализации обозреватель может поворачивать и перемещать элементы программы. При перемещении по программе-городу у обозревателя ограниченные возможности. Делается различие между вертикальной навигацией, когда обозреватель летает вокруг города, перемещаясь вперед и назад, изменяя высоту и горизонтальной навигацией. Горизонтальная навигация означает перемещение вокруг города посреди зданий, но с ограниченными возможностями перемещения заданными физическими ограничениями мира: невозможно пройти насквозь здания или перемещаться под землей. Обозреватель может быть только в режиме вертикального перемещения, либо в режиме горизонтального перемещения.

IV. Дефекты проектирования

Весьма интересным было исследовать возможность использования изображения программы-города для обнаружения и визуализации дефектов проектированиях[11][12][13]. Далее делается краткий обзор распространенных дефектов проектирования. Для обнаружения таких дефектов применяются логические выражения, использующие значения вычисленных объектно-ориентированных метрик. Отсутствие дефектов в проекте далее будет называться гармонией.

Рассмотрим следующие виды гармонии в проектировании программы:

1. *Гармония в идентификации.* Ответ на вопрос: как элемент программы идентифицирует сам себя? Можно определить следующие

Божественный класс (God Class) - это класс, который выполняет большой объем работы самостоятельно, не используя методы других классов. Вместе с тем этот класс интенсивно использует поля других классов.

Класс-умник (Brain Class) - это класс, содержащий несколько методов-умников.

Метод-умник (Brain Method) - это метод, который централизовал в себе функциональность класса.

Класс данных (Data Class) - это примитивный класс для хранения данных. При этом он не содержит какой-либо сложной функциональности. На этот класс ссылаются другие классы.

Завистливый метод (Feature Envy) - метод класса ссылается на методы в других классах. Данные в других классах для данного класса более интересны, чем собственные данные.

2. *Гармония взаимодействия.* Как выполняется взаимодействие с другими классами для решения классом какой-либо задачи? Класс все делает сам или использует другие классы и интерфейсы? Как он их использует? Не чрезмерное ли это использование. Возможны следующие дефекты взаимодействия:

Интенсивность сцепления (Intensive Coupling) относится к методу, который связан со многими методами, размещенными только в нескольких классах системы.

Дисперсное сцепление (Dispersed Coupling) является дополнением к интенсивности сцепления. Метод связан со многими методами, размещенными в большом числе классов системы.

Хирург с дробовиком (Shotgun Surgery) относится к факту, что изменение в этом методе приводит к изменениям в большом числе других методов и классов.

3. *Гармония классификации.* Как определены связи класса со своими предками и потомками. Использует ли подкласс все унаследованные методы, или игнорирует некоторые из них.

Описания дефектов проектирования этой категории можно найти в [12][13].

Стратегию обнаружения дефектов можно представить формально, как показано на рисунке 11.

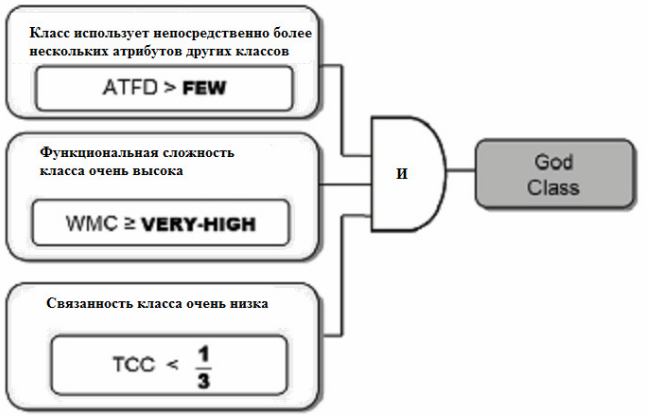


Рис.11. Пример стратегии обнаружения дефекта Божественный Класс (God Class).

Условия фильтрации можно представить с использованием значений следующих метрик.

Доступ к внешним данным (Access To Foreign Data - ATFD) представляет количество внешних классов, подмножество атрибутов которых доступно в данном классе.

Количество взвешенных методов (Weighted Method Count - WMC) как сумма статистической сложности класса вычисленной с использованием метрики цикломатической сложности класса.

Тесная связанность классов (Tight Class Cohesion - TCC) - относительное количество методов связанных через доступ к атрибутам.

V. Карты дефектов

Весьма полезным для визуализации обнаруженных дефектов использование изображения программы-города. Создаваемые таким образом карты дефектов имеют специфику для дефектов уровня классов и дефектов уровня методов. Рассмотрим начала карты дефектов уровня классов.

A. *Карта дефектов класса*

На рисунке 8 показано базовое изображение, которое затем будет использоваться для выделения на карте дефектов проектирования. На этом изображении показаны классы стандартной библиотеки языка Java - JDK 1.5. Для характерных классов на этом изображении показаны также значения метрик NOA и NOM.

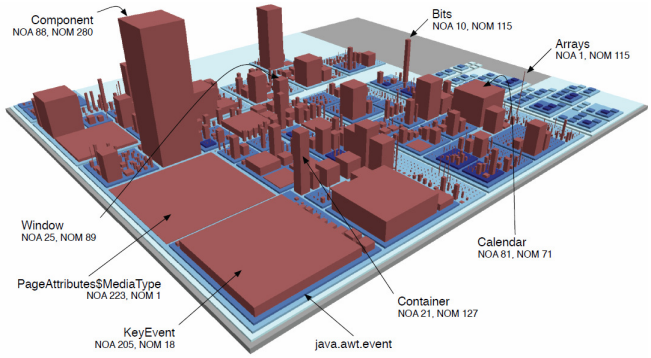


Рис.12. Базовое изображение программы-города для библиотеки JDK 1.5

Имеющееся базовое изображение можно использовать как черновик для поиска дефектов. Классы с высоким значением метрики NOM могут, возможно, обладать дефектом "божественный класс", а файлы с высоким значением метрики NOA и небольшой функциональностью, могут обладать дефектом "класс данных". После задания дополнительных условий для фильтра поиска и выделения классов цветом могут быть выбраны реальные дефекты "божественный класс". Это показано синим цветом на рисунке 13. Для классов, не обладающих указанным дефектом, используется серый цвет.

Возможна и более сложная пометка классов в программе-городе, если классы обладают одновременно несколькими дефектами. В этом случае для сочетаний дефектов проектирования выделяются отдельные цвета, а классы, обладающие высоким значением метрики NOM, снабжаются свойством прозрачности. Так, на рисунке 14 зеленым цветом помечен дефект "класс данных", желтым цветом помечен дефект "класс умник", синим цветом чистый дефект "божественный класс", красным цветом сочетание дефектов "класс умник" и "божественный класс".

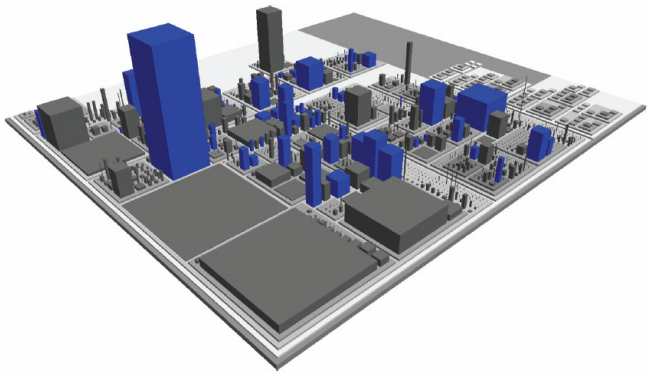


Рис.13. Классы с дефектом "Божественный класс"

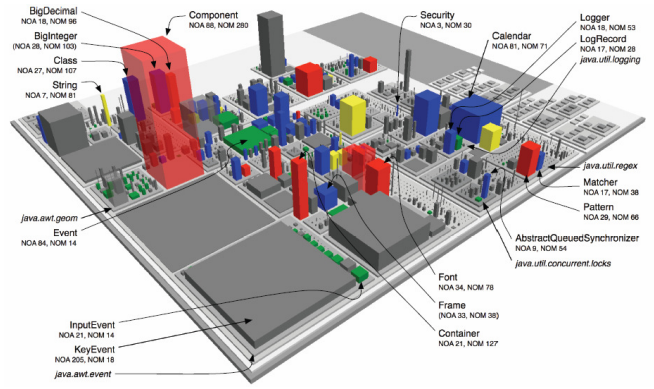


Рис.14. Раскраска нескольких дефектов уровня класса для JDK 1.5.

B. Карта дефектов методов

Для визуализации дефектов уровня методов планировка программы-города несколько видоизменяется. Классы отображаются как ступеньки одинаковой высоты, с надстроенными над ступеньками-пакетами. На этих ступеньках располагаются здания классов, состоящие из методов.

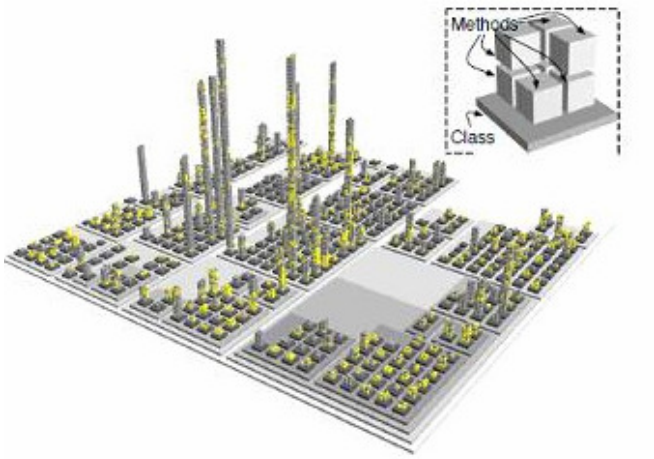


Рис.15. Изображение дефектов уровня методов в программе-городе.

Показанное на рисунке изображение программы-города для некоторых систем может быть неприемлемым, если город содержит здания небоскребы, в сравнении с которыми остальные здания имеют одинаковую высоту. В этом случае изображение города может быть настроено так, что бы размер класса (количество методов) представлялся не высотой здания, а его объемом. Толщина стены такого здания - один метод. Такое представление дефектов методов показано на рисунке 16. Использование для сравнения объема здания, а не его высоты позволяет упростить восприятие значений метрик и расположение дефектов в зданиях-классах.

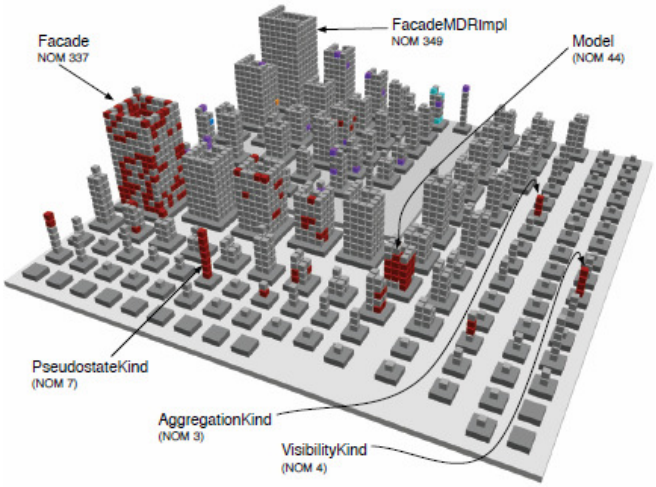


Рис.16. Изображение дефектов методов на "объемных" зданиях-классах.

VI. Визуализация эволюции программной системы

Широкое распространения библиотек программного кода, хранящихся в системах управления версиями с программным интерфейсом, позволяющим осуществить доступ к версиям системы через интернет, позволяет отследить и визуализировать эволюцию многих популярных библиотек. Такая визуализация эволюции естественно представляется с помощью программы-города [14].

Для визуализации эволюции используются в основном два подхода. При первом подходе для визуализации возраста зданий-классов и районов-пакетов используется шкала цветов, как показано на рисунке 17.

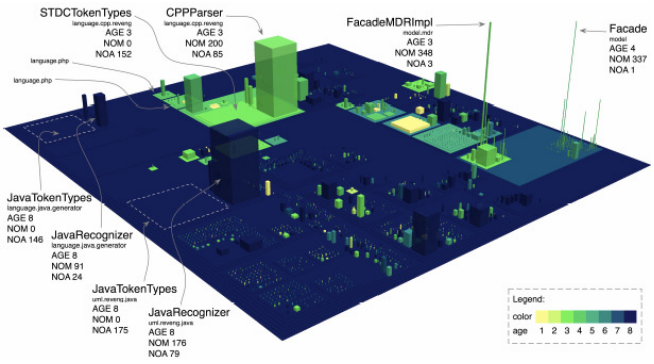
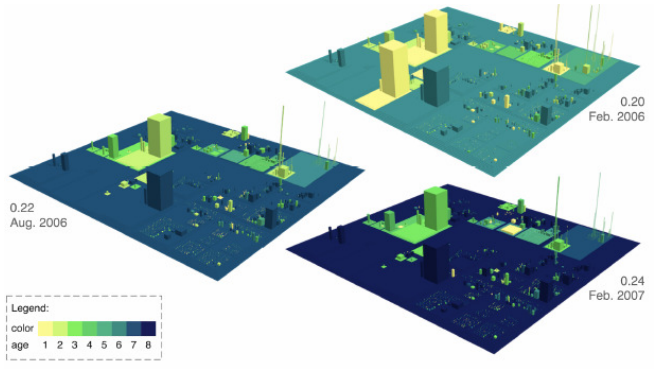


Рис.17. Визуализация эволюции программной системы с помощью карты возраста города

Темно-синим цветом на карте программы-города показываются наиболее старые классы-районы. Желтым цветом показываются новые классы и районы.

Возможен также подход, когда возраст здания-класса показывается как его высота. В этом случае серия изображений представляет снимки роста программы-города. Происходит своего рода путешествие во времени.

Как серия снимков, могут быть представлены и последовательности карт возраста зданий. Этот подход проиллюстрирован на рисунке 18.



.
Рис.18. Сочетание карты возраста и "путешествия во времени"

VII. Сочетание карты программы-города и языка UML

Рассматриваемые до сих по карты программы-города не содержали изображений отношений между элементами программ. В работе [15] была сделана попытка объединить трехмерную визуализацию программы и унифицированный язык моделирования UML[16]. Первый подход, показанный на рисунке 19, используется для визуализации на UML диаграммах значений объектно-ориентированных метрик. При этом подходе в трехмерном пространстве располагается плоский чертеж - UML диаграмма. Третье измерение используется для визуализации значений метрик с помощью расположенных на измеряемых элементах программы цилиндров.

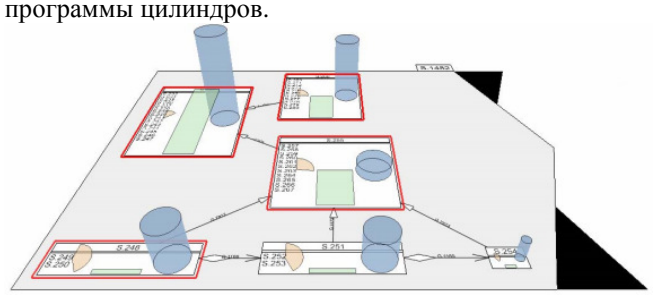


Рис.19. Сочетание отображения плоских UML-диаграмм и метрических данных.

При втором подходе, как показано на рисунке 20, расположенные в трехмерном пространстве элементы UML диаграмм (классы, интерфейсы) соединены изображениями отношений между элементами в традиционной графической нотации языка UML.

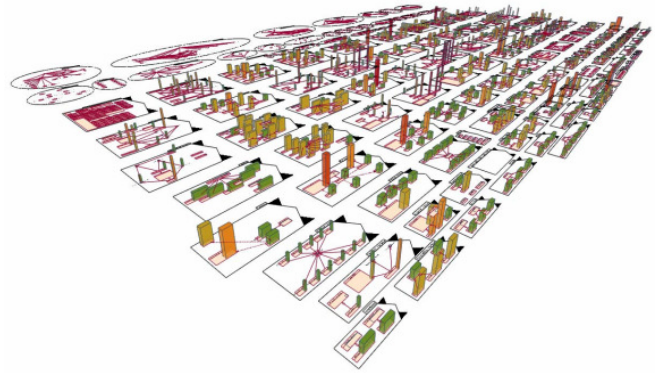


Рис. 20. UML-город: сочетание языка UML и программы-города.

VIII. Заключение

Рассмотренные в статье подходы к трехмерной визуализации сложных программных систем получают все большее распространение. Использование метафоры программы-города, а также использование графической нотации языка UML-для визуализации отношений между элементами города должно использоваться в рассматриваемом в предыдущих статьях инструменте моделирования и визуализации архитектуры программной системы.

Статья является продолжением цикла публикаций по программной инженерии и применению UML, начатой в журнале INJOIT работами [1][2][3][4][5][6][7], а также отраженной в более ранних публикациях [17]. Эта работа относится к числу одного из направлений исследований в Лаборатории ОИТ факультета ВМК МГУ [18].

Библиография

[1] 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.

[2] 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.

[3] 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

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

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

[6] 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.

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

[8] M. Balzer, A. Noack, O. Deussen, and C. Lewerentz. Software landscapes: Visualizing the structure of large software systems. In VisSym 2004, Symposium on Visualization, Konstanz, Germany, May 19-21, 2004, pages 261–266. Eurographics Association, 2004.

[9] T. Panas, R. Lincke, and W. L¨owe. Online-configuration of software visualization with Vizz3D. In Proceedings of ACM Symposium on Software Visualization (SOFTVIS 2005), pages 173–182, 2005.

[10] R.Wettel, M.Lanza, R.Robbes. Software Systems as Cities: A Controlled Experiment. In Proceedings of ICSE 2011 (33rd International Conference on Software Engineering), pp. 551 - 560, ACM Press, 2011.

[11] R.Wettel, M.Lanza. Visually Localizing Design Problems with Disharmony Maps. In Proceedings of Softvis 2008 (4th International ACM Symposium on Software Visualization), pp. 155 - 164, ACM Press, 2008.

[12] Романов В.Ю. Визуализация для измерения и рефакторинга программного обеспечения, в журнале International Journal of Open Information Technologies ISSN: 2307-8162, 2013г, том 1, № 9, с. 1-10

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

[14] R.Wettel, M.Lanza. Visual Exploration of Large-Scale System Evolution. In Proceedings of WCRE 2008 (15th Working Conference on Reverse Engineering), pp. 219 - 228, IEEE Computer Society, 2008.

[15] Christian F.J. Lange, Martijn A.M. Wijns, Michel R.V. Chaudron. A Visualization Framework for Task-Oriented Modeling using UML. Proceedings of the 40th Hawaii International Conference on System Sciences - 2007

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

[17] 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.
[18] Намиот Д., Сухомлин В. О проектах лаборатории ОИТ //International Journal of Open Information Technologies. – 2013. – Т. 1. – №. 5. – С. 18-21.

Visualization and analysis of large software systems by its 3D representation

Romanov V.Y.

***Abstract* — this article discusses software code visualization systems using 3D space. We consider the images based graph in 3D space, the metaphor a software system as a code city with buildings and districts, and of UML diagrams and code city.**

Keywords — software visualization, architecture recovery, reverse engineering.