004.415.2.045 (076.5)

МОНИТОРИНГ ДЕФЕКТОВ ПРОЕКТИРОВАНИЯ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

А.С. Нечай

Национальный авиационный университет,
Киев, проспект Космонавта Комарова, 1, корп. 6.
E-mail: alexander.nechay@livenau.net

Дефекты проектирования вносятся в программное обеспечение в процессе его сопровождения в результате невыполнения правил проектирования. Несмотря на то, что дефекты проектирования негативно влияют на сопровождаемость программного обеспечения, и должны устраняться, в некоторых случаях их внесение является лучшим проектным решением проблемы. В работе описываются графические представления, предназначенные для облегчения мониторинга дефектов проектирования. Определен каталог категорий, объединяющих дефекты проектирования по общности их истории, позволяющий упростить интерпретацию графических представлений. Показано что мониторинг с помощью визуализации позволяет быстро выделить прогрессирующие дефекты проектирования.

Design flaws are introduced into software during its maintenance as a result of failure to comply with design rules. Despite the fact that defects in the design adversely affect the maintainability of software, and should be eliminated, in some cases, their introduction is the best design solution. The paper describes the graphical representations designed to facilitate the monitoring of design flaws. Catalog of categories is defined which combine defects in the design by their shared history and allows simplifying the interpretation of graphic representations. It is shown that by monitoring using visualization one can quickly identify progressing design flaws.

Вступление

Во время сопровождения в программное обеспечение вносятся изменения, часто в жестких условиях ограниченных сроков и ресурсов. В результате нарушаются правила проектирования, его конструкция деградирует и, как следствие, становится сложной в понимании и модификации. Данное явление известно как распад программного обеспечения [1, 2] и является чрезвычайно вредным, поскольку имеет тенденцию сначала быть незамеченным, но потом нарастать со временем [3].

Несоответствие структурных характеристик элемента или фрагмента конструкции программы правилу проектирования известно как дефект проектирования [4]. Значительная часть усилий во время сопровождения направлена на определение пораженных дефектами проектирования конструкций программного обеспечения и их восстановление. Существующие методы и средства диагностики объектно-ориентированного программного обеспечения дают большое количество ошибочных [5] или не совпадающих с оценками разработчиков результатов [6]. Поэтому обнаружение дефектов проектирования остается ресурсоемкой, выполняемой преимущественно вручную, задачей. Это связано с тем, что хотя дефекты проектирования и не желательны, существуют случаи, когда их внесение наиболее обоснованное решение задачи. Такими случаями могут быть:

- автоматизация сложных предметных областей;
- использование кода сторонних разработчиков, который не сопровождается силами организации;
- использование сгенерированного кода, сопровождение которого выполняется путем повторной генерации, и не требует ручного внесения изменений;
- использование элементов конструкции программного обеспечения, использующихся в качестве шлюза между объектно-ориентированной и не объектно-ориентированной системами ;
- существуют строгие требования по производительности, которые противоречат требованию по сопровождаемости и понимаемости.

Дефекты, внесенные обоснованно, обычно не усложняют сопровождение и имеют постоянную степень развития на протяжении своей истории. Таким образом актуальна задача мониторинга того как дефекты проектирования зарождаются, развиваются и устраняются для повышения точности обнаружения наиболее опасных дефектов проектирования и, следовательно, понижения стоимости сопровождения программного обеспечения.

Анализ последних исследований и публикаций

Много работ изучают различные аспекты задач обнаружения и исправления дефектов проектирования. В работе [7] описаны 22 дефекта проектирования, структуры в коде программы, требующие реструктуризации. Дефекты проектирования, такие как дублированный код, длинный метод, большой класс, длинный список параметров определяются несколькими симптомами и предполагают определенные реструктуризации. В работе [8] описаны 40 анти-шаблонов, включая хорошо известные «Blob» и «Spaghetti code». В работе [9] определена 61 эвристика, характеризующая хорошее объектно-ориентированное проектирование для оценки и улучшения проекта программного обеспечения вручную.

В литературе предложены ряд подходов к решению задачи обнаружения дефектов проектирования [10-14]. В работе [10] используются запросы к модели программного обеспечения на языке Prolog для выявления дефектов. В работе [11] предлагается применение шаблонов кода для выявления дефектов проектирования. Применение правил, основанных на метриках, для выявления дефектов проектирования предложено в работе [12]. Кроме этого применялись методы визуализации для понимания конструкции программного обеспечения. В работе [13] предложен предметно-ориентированный язык для описания дефектов проектирования. На основе программ, написанных на данном языке, автоматически генерируются алгоритмы обнаружения дефектов проектирования на языке программирования Java. Данные подходы применяют анализ лишь одной версии программной системы, поэтому не могут оперировать полезной информацией, связанной с историей системы.

Метод, предложенный в работе [14] решает задачу анализа эволюции элементов конструкции, пораженных дефектами проектирования, и позволяет использовать историческую информацию для повышения точности обнаружения дефектов проектирования. В последнее время несколько работ были посвящены изучению влияния дефектов проектирования на сопровождаемость программного обеспечения. В работе [15] показано, что устранение дефектов проектирования может упростить понимание программ. Исследование влияния дефектов проектирования на предрасположенность элементов конструкции программного обеспечения к изменениям на примере проектов с открытым кодом Azures и Eclipse [16] показало, что элементы конструкции, пораженные дефектами проектирования, больше склонны к изменениям, чем элементы, лишенные дефектов.

Данные работы повысили уровень знаний о дефектах проектирования, их влиянии на эволюцию программного обеспечения и методах обнаружения. Однако каждый дефект проектирования возникает и развивается по-своему в результате внесения изменений в программное обеспечение. Именно способ их развития определяет их опасность для эволюции программного обеспечения. Если дефект не меняется со временем, то он менее опасен, чем дефект, степень развития которого увеличивается. Ни одна из рассмотренных работ не направлена на выявление дефектов проектирования на этапе их зарождения и наблюдение за их развитием с целью своевременного планирования работ по реструктуризации.

Дефекты проектирования

Понятие дефекта проектирование охватывает проблемы разного уровня детализации – от низкоуровневых проблем, таких как плохой запах (bad smell), до архитектурных проблем, таких как анти-шаблоны проектирования (anti-patterns). Дефект Blob [8] (God Class [9]) – типичный пример дефекта проектирования. Класс, пораженный данным дефектом – большой класс-контроллер, который монополизирует значительную часть вычислений программы. В нем объявлено много полей-данных и методов, связанность между которыми слабая. Класс-контроллер зависит от данных, которые хранятся в простых классах-данных. Последние содержат только данные и методы доступа к ним.

Анализ литературы [12, 17, 18] показывает, что существуют следующие причины возникновения дефектов проектирования:

- ограничения сроков и ресурсов выполнения проектных работ – программное обеспечение вступает в фазу сопровождения, имея хорошо спроектированную структуру, но инженеры по сопровождению сталкиваются с жесткими сроками внесения изменений и вынуждены выбирать самые простые проектные решения, вместо тех, которые необходимы для сохранения целостности структуры и предотвращения распада программного обеспечения;
- недостаток знаний архитектуры – инженеры по сопровождению не имеют достаточно знаний начальной архитектуры проекта, а внесение изменений в программное обеспечение требует модификации архитектуры;
- сопровождение унаследованного программного обеспечения – унаследованное программное обеспечение написано во времена, когда парадигма объектно-ориентированного программирования только появилась и большинство разработчиков не имели достаточного понимания принципов его проектирования;
- несоответствующая архитектура – архитектура, которая не поддерживает необходимые изменения;
- неточные требования – могут помешать инженерам придерживаться правил проектирования учитывая необходимость внесения большого количества изменений;
- неадекватные средства разработки, например, отсутствие системы управления версиями или невозможность обрабатывать параллельные изменения;
- непостоянство персонала – частая замена инженеров, сопровождающих программную систему, новым инженерам нужно много времени, что бы изучить и понять проектные решения предшественников.

Классификация дефектов проектирования

Дефекты проектирования классифицируют по природе, типу элемента и возможности измерения.

По своей природе, дефекты разделяют на такие виды [19]:

- идентификационные – дефекты, связанные с не правильным определением элементов конструкции программной системы (методов, классов, подсистем), пораженные элементы слишком сложные, слишком большие или маленькие, неправильно представляют сущность моделируемой предметной области (для классов) и т. д.;

- кооперативные – дефекты, связанные с неправильным определением взаимоотношений между элементами конструкции программной системы, пораженные элементы слишком связанные с другими элементами, зависят от слишком многих элементов и т.д.;
- классификационные – дефекты, связанные с некорректным использованием иерархий классов, пораженные классы участвуют в иерархиях аномальной глубины или ширины, пораженные унаследованные классы не используют унаследованные методы и т. д.

По типу пораженного элемента конструкции ПО, дефекты разделяют на следующие виды [12]:

- дефект метода – методы реализуют абстракцию управления, поэтому к ним применяются правила структурного программирования, но поскольку в работе рассматриваются дефекты объектно-ориентированного проектирования, к данному типу относятся дефекты, связаны либо с неправильным распределением функциональности по методу класса, или неправильным размещением метода;
- дефект класса – большинство известных дефектов проектирования относятся к классу или кластеру классов;
- дефект подсистемы – класс имеет слишком подробный уровень детализации, что бы быть полезным элементом организации больших программных систем, поэтому системы строятся из подсистем, которые так же могут быть поражены дефектами проектирования.

По возможности измерения, дефекты разделяют на следующие виды [4]:

- измеряемые дефекты, которые в процессе сопровождения ПО, помимо появления и исчезновения могут изменять степень своего развития;
- неизмеряемые дефекты, которые в процессе сопровождения ПО могут только появляется и исчезать в результате реструктуризации.

Модель дефекта проектирования

Поскольку неизмеряемые дефекты проектирования не могут изменять своей степени развития, их мониторинг не имеет смысла. Поэтому далее в работе сконцентрируемся на мониторинге измеряемых дефектов проектирования. Для этого для каждого дефекта проектирования необходимо построить его модель, с помощью которой можно определять его степень развития. Степень развития дефекта проектирования – это количественная характеристика отклонения структурных характеристик элемента или фрагмента конструкции программы от правила проектирования [20]. Модель дефекта проектирования – это описание элемента программного обеспечения, пораженного дефектом. Построение модели выполняется путем построения функций с помощью операторов агрегирования на основе онтологии знаний объектно-ориентированного проектирования. Функции модели позволяют определить степень развития дефекта проектирования [20], а также среднюю интенсивность признаков дефекта, поскольку оба эти свойства дефекта проектирования необходимы для проведения его мониторинга.

Модель дефекта проектирования состоит из таких функций:

- $\varphi_{\mathrm{d}} : \mathrm{E} \to [0,1000]$ – функция для определения степени развития дефекта $d \in D$;
- $\mu_{\mathrm{d}} : \mathrm{E} \to [0,1000]$ – функция для определения средней интенсивности простых признаков дефекта $d \in D$,

где $E = \{e_1, e_2, e_3, \ldots e_k\}$ – множество элементов конструкции ПО, а $\mathrm{D} = \{\mathrm{d}_1, \mathrm{d}_2, \mathrm{d}_3, \ldots \mathrm{d}_\mathrm{p}\}$ – множество дефектов проектирования, которые могут возникать у элемента конструкции типа $e \in E$.

Для построения модели необходимо выполняются следующие шаги:

- сформулировать правило проектирования элемента конструкции программного обеспечения;
- выполнить анализ правила для определения признаков его нарушения;
- определить метрику для вычисления интенсивности каждого простого признака;
- установить пороговое значение для каждой из метрик;
- построить функций модели дефекта проектирования путем комбинирования функций для определения интенсивностей его признаков.

Графические представления для мониторинга дефектов проектирования

Известно, что человек лучше воспринимает информацию, представленную в графическом виде, чем текстовом. Поэтому в нашей предыдущей работе [21] предлагается набор графических представлений для упрощения мониторинга дефектов проектирования. Ключевую роль при построении представлений для мониторинга дефектов играет их степень развития. Представления отображают дефекты проектирования элементов конструкций таких уровней абстракции: уровень метода, класса, подсистемы, и в таких аспектах: история распада программного обеспечения, история развития дефекта проектирования и признаков дефекта проектирования. Представления строятся на основе метамодели истории дефектов проектирования (DDHM – Design Flaws History Model) объектно-ориентированного программного обеспечения [21].

Для отображения дефектов проектирования в аспекте истории распада программного обеспечения используется представление «Рентгенограмма». Данное представление предназначено для решения следующих задач:

- мониторинга распределения дефектов по элементам конструкции;
- формирование общей картины распада программного обеспечения.

В качестве основы представления «Рентгенограмма», взято представление, предложенное в [22], которое предназначено для мониторинга результатов тестирования.

Рассмотрим правила построения представления «Рентгенограмма» (рис. 1). Представление строится на основе матрицы. Строчка матрицы отображает составной элемент конструкции программного обеспечения. Группа строк отображает контейнерный элемент конструкции, содержащий элементы, представляемые строками. Столбец отображает версию программного обеспечения, более ранние версии располагаются слева. Цвет прямоугольника отображает степень развития наиболее развитого дефекта проектирования соответствующей версии элемента конструкции. Используются оттенки серого цвета – чем темнее цвет, тем больше степень развития дефекта. В качестве контейнерного элемента конструкции пользователем может быть выбран класс или подсистема, а в качестве составного – класс или метод.

Алгоритм построения матрицы для представления «Рентгенограмма» использует:

- DDHM;
- сделанный пользователем выбор типа контейнерного элемента конструкции;
- сделанный пользователем выбор типа составного элемента конструкции.

Выходом алгоритма является матрица, элементами которой являются узлы графа DDHM, представляющие версии элементов конструкции программного обеспечения. Алгоритм визуализации использует правила представления «Рентгенограмма» и полученную матрицу.

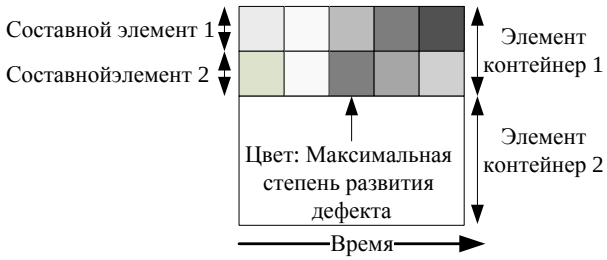

Рис. 1. Правила представления «Рентгенограмма»

Для отображения дефектов проектирования в аспекте истории развития дефекта проектирования используется представление «История дефекта». Данное представление предназначено для мониторинга дефектов проектирования определенного типа. Тип элементов конструкции (подсистема, класс, метод) и соответствующий тип дефектов проектирования выбирается пользователем.

В качестве основы для данного представления выбрано представление «Evolution Matrix» [23], целью которого является поддержка понимания эволюции классов объектно-ориентированного программного обеспечения. Рассмотрим правила построения представления «История дефекта» (рис. 2). Представление так же строится на основе матрицы. Каждая строчка матрицы отображает элемент конструкции программного обеспечения. Столбцы отображают время слева на право. Каждый столбец отображает версию программного обеспечения. Прямоугольники на пересечениях отображают дефекты проектирования, а пораженные ими версии элементов конструкции программного обеспечения соответствуют пересечениям строк и столбцов матрицы. Три возможных цвета прямоугольников отображают степень развития дефекта проектирования в соответствии с подобластью, в которую попадает ее значение. На области значений степень развития дефекта проектирования выделим следующие подобласти:

- зеленая подобласть – интервал (0,75] – значение степени развития дефекта проектирования далеко от превышения порога;
- желтая подобласть – интервал (75,100] – значение степени развития дефекта проектирования близко к превышению порога;
- красная подобласть – интервал (101,1000] – значение степени развития дефекта проектирования превысило порог.

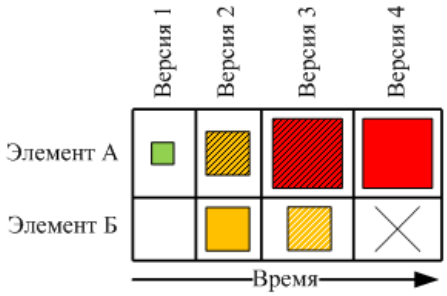

Рис. 2. Правила представления «История дефекта»

Длина диагонали прямоугольника соответствует степени развития дефекта. Если прямоугольник отсутствует на пересечении, то дефект в данной версии элемента конструкции отсутствует. Узор черного цвета внутри прямоугольника означает, что степень развития данного дефекта увеличилась по сравнению с предыдущей его версией, или осталась неизменной, но увеличилась средняя интенсивность признаков дефекта. Узор белого цвета внутри прямоугольника означает, что степень развития данного дефекта уменьшилась по сравнению с предыдущей его версией, или осталась неизменной, но уменьшилась средняя интенсивность признаков дефекта. Таким образом, небольшие изменения степени развития дефекта проектирования, которые визуально не заметны, все равно будут зафиксированы. Алгоритм построения матрицы для представления «История дефекта» использует:

- DDHM;
- сделанный пользователем выбор типа элементов конструкции подлежащего визуализации;
- предикат Φ, обеспечивающий фильтр при формировании списка элементов конструкции подлежащих визуализации.

Выход алгоритма – матрица, элементами которой являются узлы графа DDHM, представляющие версии элементов конструкции программного обеспечения. Алгоритм визуализации использует правила представления «История дефекта» и полученную матрицу.

Для отображения дефектов проектирования в аспекте истории развития признаков дефекта используется «История признаков дефекта». Данное представление предназначено для мониторинга признаков дефекта проектирования. Элемент конструкции (подсистема, класс, метод) определяется выбранным пользователем дефектом проектирования.

Рассмотрим правила построения представления «История признаков дефекта» (рис. 3). В качестве основы выбран набор правил информационной визуализации, успешно использующийся как для визуализации программного обеспечения [24], так и в других областях [25]. Каждый признак имеет назначенный ему цвет и отображается слоем, толщина которого соответствует интенсивности проявления признака. По горизонтальной оси откладывается время. Толщина всех слоев соответствует сумме интенсивностей всех признаков и пропорциональна средней интенсивности признаков дефекта.

Алгоритм подготовки входных данных для представления «История признаков дефекта» использует:

- DDHM;
- сделанный пользователем выбор истории дефекта.

Выходом алгоритма – вектор, элементами которого являются узлы графа DDHM, представляющие версии дефектов проектирования элементов конструкции программного обеспечения.

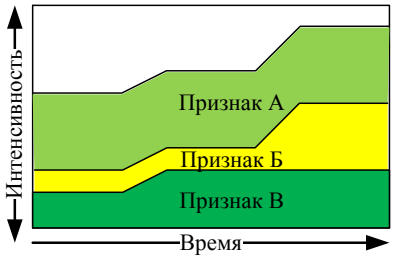

Рис. 3. Правила представления «История признаков дефекта»

Категоризация дефектов проектирования на основе их истории

При мониторинге дефектов проектирования основную роль играет представление «История дефекта», поскольку позволяет проводить наблюдение за конкретным дефектом проектирования. Поэтому для упрощения интерпретации этого представления предлагается каталог категорий дефектов проектирования. Идея использования категорий для интерпретации визуальных представлений была взята из работы [23], где категории классов используются для упрощения понимания эволюции объектно-ориентированного программного обеспечения.

Категорию дефектов проектирования определим как группу дефектов проектирования, объединенных общностью своей истории. Каждой категории соответствует следующее:

- имя, по которому категория идентифицируется в каталоге;
- образец, по которому может быть установлена принадлежность дефекта категории при анализе представления «История дефекта», в данном случае образец визуальный, хотя может быть задан в иной форме, например, для автоматического установления принадлежности дефекта категории;
- образец интерпретации истории дефекта отображенной на представлении «История дефекта».

Таким образом, для интерпретации представления «История дефекта» с помощью категорий необходимо выполнить следующее:

- поиск фрагментов представления, соответствующих визуальным образцам категорий;
- отнесение дефектов, истории которых представлены найденными фрагментами представления к соответствующим категориям;
- интерпретация историй отнесенных к категориям дефектов на основе образцов интерпретации соответствующих категорий.

Кроме упрощения интерпретации представлений, каталог категорий формирует словарь, который может быть использован для коммуникации инженеров, проводящих мониторинг, что имеет большое значение в контексте сложности программного обеспечения и необходимости взаимодействия членов команды инженеров, проводящих диагностику и сопровождение программного обеспечения. Отметим, что категории не являются взаимоисключающими, т. е. один дефект на разных участках своей истории может принадлежать разным категориям дефектов.

Рассмотрев понятие категории дефектов проектирования и применение категорий при мониторинге дефектов проектирования перейдем к описанию каталога категорий. Интерпретации историй построены на основе опыта полученного при экспериментальном мониторинге программного обеспечения и не претендуют на то, что бы быть единственно правильными.

Увеличение. Данной категории принадлежат дефекты проектирования степень развития которых увеличивается на протяжении истории.

Визуальный образец представлен на рис. 4.

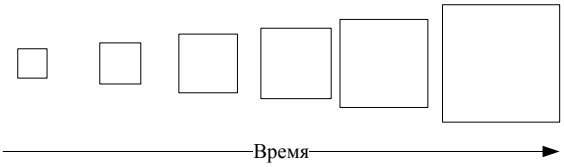

Рис. 4. Визуальный образец категории «Увеличение»

Интерпретация. Рост дефекта говорит о том, что при подготовке новых версий, работы по сопровождению пораженного дефектом элемента конструкции программного обеспечения проводятся игнорируя правила проектирования, или элемент конструкции разработан таким образом, что инженеры вынуждены постоянно его изменять в худшую сторону, выполняя задачи по сопровождению, и тем самым усложняя их выполнение в дальнейшем. Причиной увеличения степени развития может так же быть побочный эффект проведенной реструктуризации. Особое внимание необходимо уделять дефектам, растущим постоянно, поскольку резко возросший или появившийся дефект может говорить о добавлении сторонних компонентов в систему или добавление сгенерированного кода.

Уменьшение. Данной категории принадлежат дефекты проектирования, степень развития которых уменьшается на протяжении истории.

Визуальный образец представлен на рис. 5.

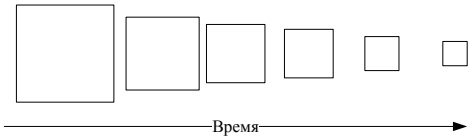

Рис. 5. Визуальный образец категории «Уменьшение»

Интерпретация. Уменьшение дефекта говорит, прежде всего, о проведенной реструктуризации, направленной на устранение данного дефекта.

Пульсирование. Данной категории принадлежат дефекты проектирования степень развития которых попеременно то увеличивается, то уменьшается на протяжении истории.

Визуальный образец представлен на рис. 6.

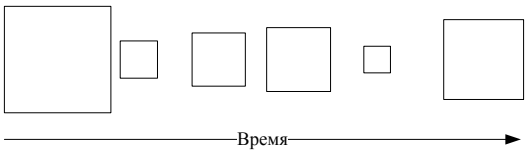

Рис. 6. Визуальный образец категории «Пульсирование»

Интерпретация. Пульсирование степени развития дефекта проектирования говорит о том, что пораженный элемент конструкции подвергается изменениям, нарушающим принципы проектирования программного обеспечения. Далее дефект обнаружен, установлено его негативное влияние на сопровождаемость, и устранен. Однако со временем инженеры по сопровождению снова вынуждены нарушать правила проектирования. Такие дефекты наиболее опасны для сопровождаемости, поскольку показывают серьезные проблемы в сопровождении пораженного элемента конструкции программного обеспечения и неспособность персонала устранить проблему. Здесь необходимо обратить внимание на время пульсирования степень развития дефекта. Если степень развития дефекта пульсировала в начале своей истории, но затем стабилизировалась, то дефект не представляет собой большой опасности и требует дальнейшего наблюдения.

Стабильность. Данной категории принадлежат дефекты проектирования, степень развития которых остается стабильной на протяжении истории.

Визуальный образец представлен на рис. 7.

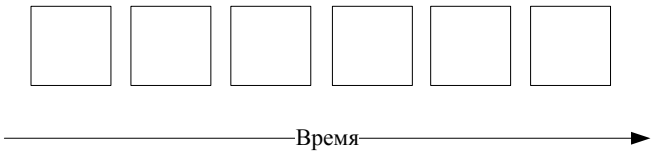

Рис. 7. Визуальный образец категории «Стабильность»

Интерпретация. Стабильность дефекта проектирования говорит о том, что его наличие не вызывает проблем сопровождения пораженного элемента конструкции, поэтому такой дефект безопасен. Причинами появления стабильного дефекта могу быть как сгенерированный код и внешние компоненты, так и сложная моделируемая предметная область, в которой, даже выполняя правила проектирования сложно добиться гибкой конструкции программного обеспечения. Причиной стабильного дефекта может стать реализация нефункционального требования к программному обеспечению, которое конфликтует из правилом проектирования. Однако в случае стабильного дефекта, особенно степень развития которого находится в красной подобласти, необходимо обратить внимание на узор прямоугольников, представляющих такой дефект. Если прямоугольники заштрихованы черной штриховкой то средняя интенсивность простых признаков дефекта возрастает, поэтому необходимо обратится к представлению «История признаков дефекта» для данного дефекта и установить причины такого его изменения.

Учитывая что в одной категории могут оказаться дефекты проектирования, степень развития которых находится как в зеленой подобласти, так и в красной подобласти, неправильно их трактовать одинаково опасными. Для решения этой проблемы предлагается каждую из категорий, кроме «Стабильность», разделить на подкатегории. Первой подкатегории принадлежат дефекты, степень развития которых на протяжении всей истории выше 100 % (верхняя), второй – степень развития которых на протяжении истории переходит значение 100 % в любом направлении (переходная), третьей – степень развития которых на протяжении всей истории ниже 100 % (нижняя). Таким образом, каталог содержит 10 категорий, название которых формируется следующим образом: «[название подкатегории] [название категории]». Например, «верхнее пульсирование» – категория дефектов, степень развития которых пульсирует, однако на протяжении всей истории не опускается ниже 100%, а «нижнее увеличение» – категория дефектов, степень развития которых увеличивается, однако на протяжении всей истории не превышает 100 %.

Применение категорий дефектов играет большую роль при решении задач ранней диагностики (отнесение дефектов к категориям «нижнее увеличение» и «нижнее пульсирование»), дефектов проектирования и обнаружения реструктуризации (отнесение дефектов к категории «переходное уменьшение»). Однако их основная польза – в решении задачи выделения наиболее опасных дефектов проектирования. На рис. 8 показана поочередность, в которой инженерам по сопровождению следует рассматривать категории дефектов проектирования.

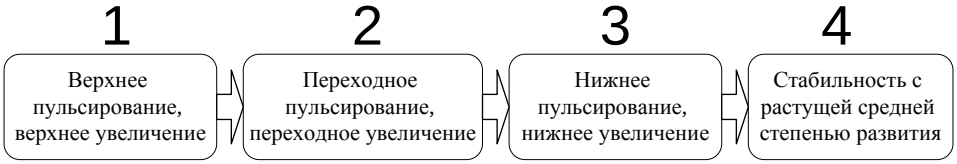

Рис. 8. Поочередность рассмотрения категорий дефектов проектирования

Рассмотрение конкретного примера мониторинга

В данном разделе выполняется мониторинг дефектов проектирования на протяжении истории существующей программной системы. Затем выполняется категоризация обнаруженных дефектов в соответствии с предложенным каталогом. Выделяются и описываются наиболее опасные дефекты, для которых рекомендуется устранение, а также приводится пример неопасного дефекта, внесение которого было оправдано. Для инструментальной поддержки мониторинга дефектов проектирования с помощью графических представлений были разработаны соответствующие средства, которые используются в данном рассмотрении.

В качестве программной системы примера используется инструмент для UML-моделирования ArgoUML с открытым исходным кодом. ArgoUML выбран благодаря тому, что имеет значительный размер и сложность, известную широкому кругу читателей предметную область (UML-моделирование), а так же доступный исходный код. Для рассмотрения конкретного примера анализируются 12 версий ArgoUML созданные в период с 09.10.02 по 16.08.09, т. е. семилетняя история программной системы. Система ArgoUML в последней версии состоит из 2,126 классов, однако на протяжении ее истории было создано 4,056 классов.

На протяжении истории выбранной программной системы выполняется мониторинг дефекта проектирования «God Class». Данный дефект выбран потому, что часто встречается в объектно-ориентированном программном обеспечении и негативно влияет на сопровождаемость. По сравнению с другими классами, класс, пораженный дефектом «God Class», выполняет значительно больше функций в системе, т. е. централизует системную логику в своих методах. Дефект «God Class» делегирует лишь небольшую часть работы набору простых классов и использует данные простых классов. В примере используется модель данного дефекта, построенная в [20].

Для начала рассмотрим количество дефектов «God Class» в каждой версии системы. На рис. 9 рост количества классов и количества классов, пораженных дефектом. Класс считается пораженным дефектом, если степень развития дефекта превысила 100 %. Рис. 9 показывает, что рост системы относительно линейный. Количество пораженных классов на протяжении всей истории остается относительно стабильным, и только в последней версии наблюдается значительное ухудшение. Кроме того, из рис. 9 видно, что количество классов возрастает значительно быстрее, чем количество пораженных классов, т.е. большинство новых классов вносятся без дефектов проектирования, а большинство пораженных классов созданы в первой версии.

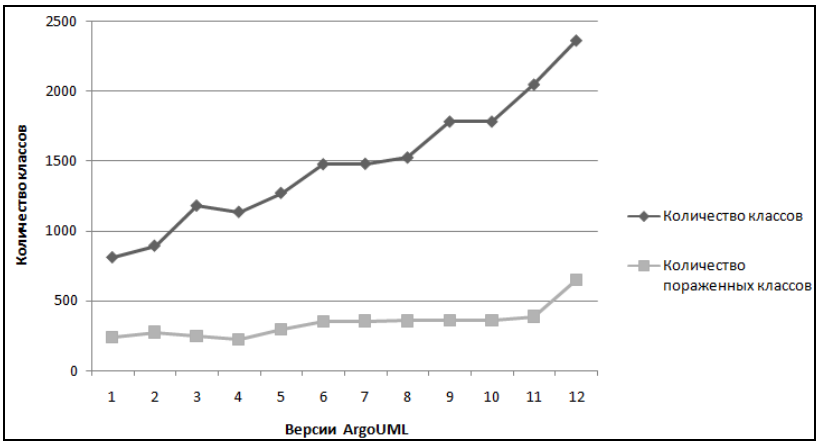

Рис. 9. Количество дефектов «GodClass» в ArgoUML

Распределение дефектов проектирования по предложенным категориям представлено на рис. 10, а. Видно, что 77 % дефектов на протяжении всей истории не изменялись и составляют самую численную категорию. Данные результаты объясняют неспособность существующих средств точно распознавать дефекты проектирования – около четверти дефектов не изменяются, а значит, на сопровождение пораженных ими элементов конструкции не расходуются усилия. Распределение дефектов по подкатегориям категорий «Пульсирование» и «Увеличение» показаны соответственно на рис. 10, б, в. Из рис. 10 видно, что самая численная подкатегория – нижняя, в которой находятся зарождающиеся дефекты, то есть те, степень развития которых не превысила 100 %. Самая малочисленная категория – верхняя, содержащая в себе наиболее опасные дефекты: «Верхнее увеличение» – 4 % (8), «Верхнее пульсирование» – 4 % (16). Таким образом, в первую очередь рекомендуется провести реструктуризацию 24 класса, пораженных данными дефектами.

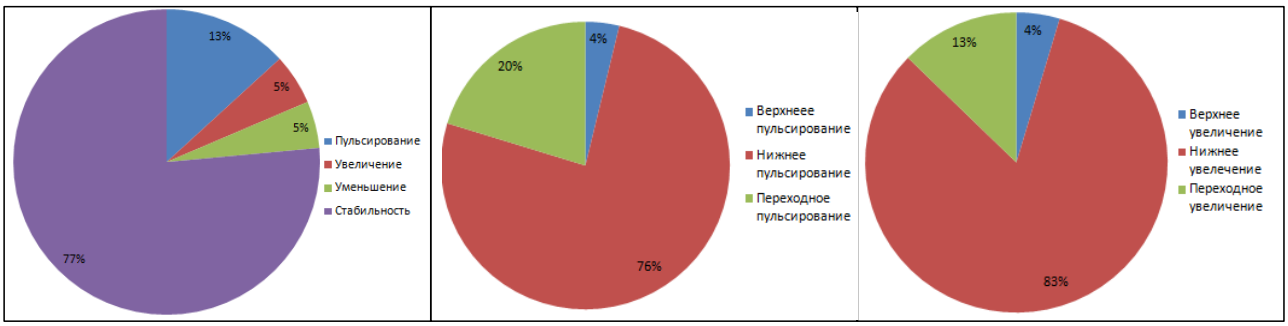

Рис.10. Распределение дефектов проектирования по категориям:
а – распределение дефектов проектирования по основным категориям;
б – распределение дефектов проектирования категории «Пульсирование» по подкатегориям»;
в – распределение дефектов проектирования категории «Увеличение» по подкатегориям»

На рис. 11 показан снимок экрана средств, где представлена «История дефектов» ArgoUML. Первый выделенный дефект проектирования (выделение помечено «Пульсирование») находится в классе FigNodeModelElement. Видно, что степень развития дефекта на протяжении всей истории превышает 100 % и пульсирует. Второй выделенный дефект (выделение помечено «Пульсирование») находится в классе FigMessage и является представителем категории «Уменьшение». Из представления видно, что во второй версии степень развития данного дефекта превысила 100 %. Однако в последующих версиях разработчики, очевидно, провели реструктуризацию, и в восьмой версии дефект был окончательно устранен. Наконец, третий выделенный дефект (выделение помечено «Увеличение») находится в классе PropPanel и является представителем категории «Увеличение». Рост данного дефекта начался в восьмой версии, после чего степень развития дефекта стабилизировалась и до последней версии остается постоянной. Несмотря на рост, рассматриваемый дефект в последнее время стабильный, а значит не вызывает проблем при сопровождении, поэтому нет необходимости его устранять, однако при внесении изменений в содержащий класс, разработчик должен тщательно их спроектировать, чтобы не допустить прогресса дефекта и избежать необходимости проводить реструктуризацию в будущем.

Рис. 11. «История дефектов» ArgoUML

Заключение

Результатом работы является развитие метода диагностики программного обеспечения путем визуального мониторинга дефектов проектирования. Описаны графические представления, предназначенные для облегчения мониторинга. Определен каталог категорий, объединяющих дефекты проектирования по общности их истории, позволяющий упростить интерпретацию графических представлений.

Важность полученных результатов состоит в их практической значимости. Мониторинг дефектов проектирования позволяет распознавать опасные прогрессирующие дефекты проектирования и своевременно планировать работы по их устранению, имея под рукой снимок истории дефектов, разработчик может предотвратить развитие дефекта, тщательно проектируя изменения. Кроме того, в рамках работы разработаны средства диагностики программного обеспечения, используемые в Национальном авиационном университете в учебном процессе при преподавании дисциплины «Архитектура и проектирование программного обеспечения» для студентов направления «Программная инженерия».

1. Lehman, M.M. On Understanding Laws, Evolution, and Conservation in the Large-Program Life Cycle / M.M. Lehman // The Journal of Systems and Software. – 1980. – vol. 1. – P. 213–221.
2. Izurieta C. How Software Designs Decay: A Pilot Study of Pattern Evolution / Clemente Izurieta, James M. Bieman // Proceedings of the First International Symposium on Empirical Software Engineering and Measurement (ESEM'07), September 20–21 2007. – Washington, 2007. – P. 449–451.
3. Godfrey M. The past, present, and future of software evolution / Godfrey, M.W. German, D.M. // Frontiers of Software Maintenance, 2008. – Beijing, 2008. – P. 129–138.
4. Нечай О.С. Методи та засоби виявлення дефектів проектування об'єктно–орієнтованого програмного забезпечення / О.С. Нечай, М.О. Сидоров // Вісник НАУ. – 2009. – № 3. – С. 200–205.
5. Kim S. Which warnings should I fix first? / Sunghun Kim, Michael D. Ernst // Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering (ESEC/FSE'07). – New York: ACM, 2007. – P. 45–54.
6. Mantyla M. Bad Smells " Humans as Code Critics / Mika V. Mantyla, Jari Vanhanen, Casper Lassenius // in Proceedings of the 20th IEEE International Conference on Software Maintenance (ICSM'04). – Washington: IEEE Computer Society, 2004. – P. 399–408.
7. Fowler M. Refactoring: Improving the Design of Existing Code / Martin Fowler. – Addison–Wesley, 1999. – 464 p.
8. Brown J. AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis / William J. Brown, Raphael C. Malveau, Hays W. "Skip" McCormick, Thomas J. Mowbray. – Wiley, 1998 . – 336 p.
9. Riel A. Object Oriented Design Heuristics/ Arthur J. Riel. – Addison–Wesley Professional, 1996 . – 400 p.
10. Ciupke O. Automatic detection of design problems in object–oriented reengineering / Oliver Ciupke // Proceedings of the Technology of Object–Oriented Languages and Systems (TOOLS'99). – Washington: IEEE Computer Society, 1999. – P. 18–32.
11. Hovemeyer D. Finding bugs is easy / David Hovemeyer, William Pugh // ACM SIGPLAN Notices. – 2004. – Vol.39, No.12. – P.92–106.
12. Marinescu R. Measurement and Quality in Object–Oriented Design: Ph.D thesis / R. Marinescu. – "Politehnica" University of Timisoara, 2002. – 155 p.
13. Moha N. A Domain Analysis to Specify Design Defects and Generate Detection Algorithms / N. Moha, Y. Guéhéneuc, F. Le Meur, L. Duchien // Proceedings of the 11th Intern. Conf. on Fundamental Approaches to Software Engineering. – Springer-Verlag, March-April 2008. – P. 276−291.
14. Ratiu D. Using history information to improve design flaws detection / D. Ratiu, S. Ducasse, T. Girba, R. Marinescu // Proceedings of European Conference on Software Maintenance and Reengineering (CSMR'04), March 24 – 26 2004. – Washington, 2004. – P. 223–232.
15. Bois B. Does god class decomposition affect comprehensibility? / B. D. Bois, S. Demeyer, J. Verelst, T. Mens, M. Temmerman // Proceedings of the 10th International Conference on Software Engineering (IASTED'06). – Calgary : Acta Press, 2006. – P. 346–355.
16. Khomh F. An Exploratory Study of the Impact of Code Smells on Software Change–proneness / Foutse Khomh, Massimiliano Di Penta, Yann–Gaël Guéhéneuc // Proceedings of the 24th International Conference on Software Maintance (ICSM'08) . – Washington : IEEE Computer Society, 2009. – P. 75–85.
17. McConnell S. Code Complete: A Practical Handbook of Software Construction / Steve McConnell. – Microsoft Press, 2004. – 960 p.
18. Eick S. Does Code Decay? Assessing the Evidence from Change Management Data / Stephen G. Eick, Todd L. Graves, Alan F. Karr, J. S. Marron, Audris Mockus // IEEE Transactions on Software Engineering. – 2001. – Vol. 27, N 1. – P. 1–12.
19. Lanza M. Object–Oriented Metrics in Practice / Michele Lanza, Radu Marinescu. – Berlin : Springer–Verlag, 2006 . – 206 p.
20. Нечай О.С. Метод побудови моделей дефектів проектування обєктно–орієнтованого програмного забезпечення / О.С. Нечай, М.О. Сидоров // Наукоємні технології. – 2009. – № 2. – С.58–64.
21. Нечай О.С. Метод діагностики об'єктно–орієнтованого програмного забезпечення / О.С. Нечай // Вісник НАУ. – 2009. – № 5. – С. 100–111.
22. D'Ambros M. "A Bug's Life" Visualizing a Bug Database / M. D'Ambros, M. Lanza, M. Pinzger // Proceedings of 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT 2007). – Washington : IEEE Computer Society, 2007. – P. 113–120.
23. Lanza M. The Evolution Matrix: Recovering Software Evolution using Software Visualization Techniques / Michele Lanza // Proceedings of International Workshop on Principles of Software Evolution (IWPSE'01). – New York : ACM Press, 2001. – P. 37–42.
24. Lungu M. Reverse Engineering Super–Repositories / Mircea Lungu, Michele Lanza, Tudor Girba, Reinout Heeck // Proceedings of 14th Working Conference on Reverse Engineering (WCRE 2007). – Washington : IEEE Computer Society, 2007. – P. 120–129.
25. Wattenberg M. Baby Names, Visualization, and Social Data Analysis / Martin Wattenberg // Proceedings of IEEE Symposium on Information Visualization (InfoVis 2005). – Washington : IEEE Computer Society, 2005. – P. 1–6.