Задача 4. Используя результаты предыдущих задач описать систему выявления требований качества на основе текста спецификаций, как целостной системы «разбор-представление-выявление». Тут необходимо провести операцию интеграции подсистем посредством универсального представления семантической информации. В качестве результата данной задачи должна быть спецификация системы в целом, а также точные спецификации всех подсистем.

Задача 5. Реализовать подсистемы, используя разработанное математическое и алгоритмическое обеспечение. Результатом данного этапа должна стать готовая к внедрению и эксплуатации система выявления требований качества из текстов спецификаций написанных естественным языком.

Выводы. Таким образом, выявление требований качества является неотъемлемой частью разработки программного обеспечения. Однако реализация систем для автоматизации данной задачи весьма усложнена в области выявления требований качества на основе спецификаций написанных естественным языком.

Для создания средств автоматизированного сбора требований качества к программному обеспечению необходимо провести дополнительные исследования по направлениям:

- лингвистический разбор (включает в себя морфологический, синтаксический разборы, а также ряд других алгоритмов, набор которых зависит от конкретной методологии);
- представление семантической информации средствами программного обеспечения;
- автоматизированное исследование синтаксической информации.

В результате проведения данных исследований необходимо получить математическое и алгоритмическое обеспечение системы выявления требований на основе спецификаций с последующей интеграцией и реализацией в виде программного обеспечения.

Список литературы: 1. *M.Glinz.* Rethinking the Notion of Non-Functional Requirements // Proc. Third World Congress for Software Quality (3WCSQ 2005), p. 55-64 **2.** *Kazman, M.Klein, P.Clemens.* ATAM: Method of Architecture Evaluation. Technical Report // CMU/SEI-2000-TR-004, CMU, 2000. **3.** *A.Kostanyan, V.Shekhovtsov.* Towards entropy-based requirements elicitation // Proc. ISTA'2007, LNI P-107, 2007. – P. 105-116. **4.** *I. Mel'čuk.* Vers une linguistique Sens-Texte. Leçon inaugurale Paris: Collège de France 1997. **5.** *N. Chomsky.* Barriers. Linguistic Inquiry Monograph 13. Cambridge: MIT Press, 1986. **6.** *С.Пинкер.* Язык как инстинкт. М.: Едиториал УРСС, 2004.

Поступила в редколлегию 22.02.08

УДК 512.086

Ю.ТОМИЛКО, НТУ «ХПИ»

АВТОМАТИЗИРОВАННОЕ РЕЦЕНЗИРОВАНИЕ ИСХОДНОГО КОДА НА ОСНОВЕ ОНТОЛОГИИ ПРАВИЛ

В статті коротко розглянута проблема рецензування вихідного коду. Проведен огляд прикладної програми FxCop, яка дозволяє автоматизувати процес реценцування. Показані складності існуючого процесу та запропоноване рішення на основі використання онтології правил. Представлені його переваги та запропоновані майбутні напрямки роботи.

In the paper the problem of code review is briefly reviewed. The survey on FxCop tool that allows automating the process of code review is conducted. The difficulties of the existent process are shown; solution based on the use of rules' ontology is presented. Its advantages are emphasized and future directions of work are outlined.

Введение. Под рецензированием исходного кода (code review) будем понимать следующее [1]: это особый вид инспектирования (inspection), в процессе которого просматривается и проверяется исходный код программ с целью обнаружения и исправления в нём ошибок допущенных при разработке программного обеспечения. В стандарте [2] определены 5 типов рецензирования программного обеспечения (software review - согласно стандарту, процесс, в ходе которого программный продукт подвергается проверке персоналом проекта, менеджерами, пользователями, заказчиками, представителями пользователей и другими заинтересованными лицами для критического разбора или одобрения): административное рецензирование, техническое рецензирование, инспектирование, сквозной контроль и аудит. Рецензирование исходного кода следует рассматривать как вид инспектирования, а не технического рецензирования (technical review). В литературе также используется термин инспектирование программ, см. например [3], т.е. просмотр и проверка исходного кода программного обеспечения с целью обнаружения в нём ошибок. Под рецензированием исходного кода понимается, как правило, менее формальный процесс.

Доказано, что инспектирование является эффективным методом обнаружения ошибок. Оно дешевле тестирования программного обеспечения, в работах [4, 5] показано, что инспектирование программного кода является более эффективным и менее дорогостоящим, чем тестирование. Согласно [6] более 60% ошибок в программном обеспечении можно обнаружить с помощью неформального исследования (инспектирования). Процесс инспектирования также может оценить качественные характери-

стики систем, соответствие стандартам, переносимость и удобство сопровождения.

Одним из наиболее эффективных и распространённых методов инспектирования является применение контрольных примеров, см. например [7, 8]. Инспекции исходного кода позволяют найти и устранить такие известные уязвимости как: переполнения буфера, утечки памяти, состояния гонки и многие другие.

1. Программное обеспечение поддержки рецензирования. Упростить процедуру рецензирования исходного кода позволяют различные прикладные программы, с помощью которых можно производить автоматизированный поиск известных ошибок, нарушения общепринятых правил и рекомендаций, а также программных конструкций, которые не рекомендуется использовать при программировании. Особое место занимает программный продукт FxCop от Microsoft [9]. Это программное обеспечение позволяет производить проверку .NET кода на предмет нарушения широкого круга правил программирования и соглашений. Правила, которые включены в FxCop построены на основе руководящих указаний Microsoft по проектированию и описаны в [10]. Правила, которые могут быть проверены FxCop, включают следующие группы:

- модельные (Design);
- глобализация;
- интероперабельность (Interoperability);
- соглашения по именованию;
- производительность;
- переносимость;
- безопасность;
- удобство использования.

В отличие от традиционных средств анализа кода (например Flawfinder, Lint) FxCop не анализирует исходный код целевого языка программирования, например C# или VB.NET. Вместо этого производится анализ бинарного CIL языка сгенерированного .NET компилятором и сохранённого в сборках. Проведение анализа возможно благодаря достаточному количеству метаданных, являющихся частью CIL. Это позволяет анализировать код, написанный на различных .NET языках.

Возможности FxCop охватывают широкий спектр функций по анализу исходного кода. Можно осуществить проверку соглашений по именованию типов, значение параметров, и даже правописание и наличие необходимых XML-комментариев. Доступными для анализа являются как мета-

данные уровня сборки, так и информация об отдельных управляющих конструкциях, операциях и операндах.

Важной особенностью FxCop является то, что пользователь не ограничен набором правил, предоставляемых в составе программного продукта. Разработчики могут определять свои правила, реализуя классы, использующие типы, определённые в FxCop SDK, представляющее собой набор сборок, которые и предоставляют необходимый API.

На наш взгляд FxCop является эффективным решением для проведения автоматизированного рецензирования исходного кода для платформы .NET и является целесообразным использование его как основы для разработки собственной системы правил, исходя из следующих предпосылок:

1. Независимость от целевого языка программирования .NET, поддержка .NET Framework 2.0, 3.0, и 3.5. В случае реализации собственного приложения для анализа исходного кода необходимо разработать парсер для каждого языка, инспекция которого необходима, объектной модели для представления исходного кода в виде некоторой структуры и работы с ним. Эти задачи являются достаточно трудоёмкими и сложными в реализации.
2. Наличие прикладного приложения реализующего оконный интерфейс и позволяющего производить анализ необходимых сборок, создание проектов, позволяющих группировать нужные сборки, генерацию отчётов о найденных ошибках и др.
3. Доступен API, который позволяет разрабатывать собственные правила для инспекции кода.
4. Наличие широких возможностей по анализу программного кода.
5. Наличие более 200-х встроенных правил для проверки исходного кода, возможность отключения проверки правил для заданных сборок.
6. Отсутствие необходимости оплаты за использование FxCop.

2. Постановка задачи исследования. Разработка классов для реализации проверки правил для каждого проекта может быть достаточно трудоёмкой задачей. Необходимо каждый раз писать код взаимодействия с API, реализовывать классы, осуществляющие проверку правил, определять набор необходимых параметров для конфигурации правил. Кроме того, повторное использование разработанных типов, реализующих проверку правил, может оказаться сложной задачей, поскольку для нового проекта в большинстве случаев будут нужны другие правила либо же существующие необходимо будет изменить. В тоже время для каждого проекта может требоваться схожий набор правил, например:

- имена типов удовлетворяют установленными правилам именования;
- используется заданный набор компонент и классов, а не их альтернативы;
- проверка управляющих структур, например ветвлений и циклов, для вычисления метрик кода;
- определение всех методов, которые использует данный класс, а также классов или методов, использующих заданный метод или класс;
- проверка связей между компонентами, например для запрета вызова методов классов уровня доступа к данным классами, отвечающими за создание пользовательского интерфейса, в обход классов, реализующих бизнес-логику.

Целесообразным является создание такого представления информации о правилах, которое позволило бы её дальнейшее повторное использование, расширение, модификацию и эффективный доступ к хранимой информации.

3. Базовые сведения об онтологиях. В настоящее время актуальными являются исследования в области онтологий и онтологических систем в различных областях информатики. Достаточно сложно дать точное определение понятию онтология. Существует большое количество определений, предложенных различными авторами и коллективами. Одним из первых определений является определение, данное в работе [11]: онтология определяет базовые понятия и связи, составляющие словарь предметной области, а также правила для комбинации понятий и связей для определения расширений словаря. Это описательное определение предоставляет базовое понимание того, что необходимо сделать для построения онтологии. Наиболее цитируемым в литературе посвящённой исследованиям в области онтологий является определение, данное Т.Грубером [12]:онтологией называется явная спецификация концептуализации. В работе [13] даётся следующее определение: онтология – это формальная, явная спецификация концептуализации. Под концептуализацией понимается некоторая абстрактная модель некоторого феномена, определяющая важные концепции данного феномена. Явная означает, что тип, используемых концепций, а также ограничения на их использование явно определены. Формальная – отражает тот факт, что онтология должна быть представлена в такой форме, чтобы была возможность эффективной компьютерной обработки содержащейся в ней информации. Также подчеркивается тот факт, что онтология содержит согласованные знания.

Следует отметить, что в настоящее время понимание термина онтология различно, в зависимости от контекста и целей его использования.

Различные определения представляют различные, дополняющие друг друга представления об одной и той же сущности. Некоторые определения независимы от процессов, сопутствующих разработке онтологии и её использованию, в то время как другие тесно связаны с процессом разработки. Делая вывод, можно сказать, что целью онтологии является представление согласованных знаний в общей форме таким образом, чтобы она могла быть повторно и совместно использована различными группами людей и программными приложениями [14].

Под формальной моделью онтологии O будем понимать упорядоченный набор трёх объектов следующего вида [15]:

$$O = <X, R, F>,$$

где X – конечное непустое множество концептов (понятий, терминов) предметной области, которую представляет онтология O;

R – конечное множество отношений между концептами (понятиями, терминами) заданной предметной области;

F – конечное множество функций интерпретации (аксиоматизации), заданных на множествах X и R.

Варьируя свойства указанных множеств можно получать различные виды онтологий. Существует большое число различных видов онтологий. Онтологии могут быть классифицированы по степени формальности, по наполнению, содержимому, по цели создания [16].

4. Использование онтологий для поддержки рецензирования кода. На наш взгляд целесообразным является создание онтологии правил для исходного кода. Под правилами здесь будем понимать конструкции языка программирования, которые запрещено использовать при разработке некоторого приложения. Это может быть использование какого-либо класса, определённая последовательность вызовов методов, вызов определённого метода класса из другого, особенности реализации некоторых классов, например наличие реализации определённых интерфейсов, свойств и т.д. Фактически это контрольные примеры ошибок в некотором обобщённом виде, то, чего делать не нужно при написании исходного кода. Целями создания данной онтологии являются:

- создание согласованного представления информации о недопустимых конструкциях языка программирования;
- создание источника информации для последующего использования прикладными приложениями для автоматизированной проверки исходного кода на наличие определённых ошибок.

Онтология должна содержать информацию о том, как можно используя термины целевого языка программирования (класс, интерфейс, метод, свойство и т.п.), а также дополнительные концепты, конструировать контрольные примеры ошибок. Создание такой онтологии позволило бы решить ряд задач перечисленных выше. Такая онтология могла бы быть совместно использована разработчиками программного обеспечения для создания новых правил, изучения уже существующих и не допущения известных ошибок (нарушения установленных правил) на основе полученной информации. Информация, содержащаяся в данной онтологии, может быть повторно использована и расширена для каждого последующего проекта.

5. Возможные подходы к реализации. На начальном этапе простейшая онтология подобного вида может быть построена на основе XML-схем (XSD). Они позволяют описывать допустимую структуру XML-документов, составляющих их элементов. На значения атрибутов могут накладываться различные ограничения, возможно задание типов данных. В данном случае набор XML-документов соответствующих разработанной схеме будет представлять собой некоторый набор правил. В последующем могут потребоваться более сложные ограничения и использование языков, обладающих большей выразительной мощностью, поэтому целесообразнее использовать языки, специально предназначенные для создания онтологий, например язык OWL.

Существенным является то, как разработанная онтология будет использоваться прикладными программами. Принимая во внимание тот факт, что анализ кода и поиск ошибок осуществляется с помощью FxCop API, одним из возможных решений может быть использование онтологии правил для автоматической генерации исходного кода классов, реализующих проверку правил. Для этого могут быть использованы XSLT-преобразования.

Перечислим преимущества использования XSLT-преобразований для генерации исходного кода классов, реализующих проверку правил:

1. Наличие эффективных средств обработки XSLT-шаблонов. В настоящее время существует большое число реализаций XSLT-процессоров, позволяющих производить преобразования XML-документов. Таким образом, нет необходимости разрабатывать отдельную технологию преобразования информации о правилах, достаточно воспользоваться существующими готовыми решениями.
2. Простота создания, использования, лёгкость модификации и высокая скорость разработки.

3. Переносимость, отсутствие привязки к определённым аппаратным или программным технологиям.

Преимущества предложенного решения в целом состоят в следующем:

1. Программный код для взаимодействия с API, реализации определённых типичных действий создаётся один раз для некоторого набора XSLT-шаблонов. В последующем один и тот же код повторно используется для создания классов, реализующих проверку правил.
2. Упрощается процесс создания новых правил. В случае необходимости создания правил, которые не могут быть описаны с помощью существующих терминов, онтология может быть расширена, необходимо создание новых XSLT-преобразований. В тоже время существующие части кода и отвечающие им XSLT-шаблоны продолжают использоваться. Необходимо только создание кода отвечающего за реализацию проверки новых правил.
3. Упрощается процесс создания правил на основе существующих. Фактически необходима лишь конфигурация существующих правил, которая осуществляется с помощью задания соответствующих параметров экземпляров правил. Нет необходимости каждый раз писать код взаимодействия с API - исходный код классов, реализующих проверку правил, будет генерироваться с заданными параметрами автоматически.

6. Возможные направления дальнейших исследований. Можно выделить два направления развития представленного решения. Первое связано с разработкой прикладного приложения позволяющего пользователю на основе информации представленной в онтологии правил создавать новые правила, используя для этого графический интерфейс. Второе направление связано с разработкой прикладного приложения для вычисления метрик исходного кода с помощью FxCop API. Информация о количестве ошибок, найденных в данном исходном коде, их тип, важность, количество также являются важными индикаторами, на основании которых можно сделать некоторые выводы о качестве исходного кода. На основании информации регулярно получаемой в результате анализа исходного кода можно строить тренд изменения показателей во времени. Данная информация может быть полезна при принятии решений о планировании и прогнозировании необходимого объёма времени для выполнения проекта или его части, количества требуемых ресурсов, учёта и снижения рисков при принятии управленческих решений, формировании оценок различных компонент программного обеспечения.

Выводы. Таким образом, в статье был рассмотрен программный продукт FxCop, позволяющий автоматизировать процесс рецензирования исходного кода путём поиска известных ошибок. Была показана необходимость разработки собственных правил, предъявляемых к исходному коду, указаны сложности существующего решения. Предложено решение на основе использования онтологии правил, позволяющее упростить разработку собственных правил, приведены его преимущества. Намечены дальнейшие направления развития работы.

Список литературы: **1.** *A.Stellman, J.Greene.* Applied Software Project Management, O'Reilly, 2005. **2.** IEEE Std 1028-1997, IEEE Standard for Software Reviews. **3.** *И.Соммервил.* Инженерия программного обеспечения, 6-е издание. М. Вильямс, 2002. **4.** *V.R.Basili, R.W.Selby.* Comparing the effectiveness of software testing strategies, 1987. SE-13(12). **5.** *T.Gilb, D.Graham.* Software Inspection. Addison-Wesley, 1993. **6.** *M.E.Fagan* Advances in software inspections. – IEEE Trans. on Software Engineering, 1986, SE-12(7). **7.** *J.Poley* Best Practices: Code Reviews, Microsoft Corporation, October 2007, http://msdn2.microsoft.com/en-us/library/bb871031.aspx. **8.** *J.D. Meier et al.*, How To: Perform a Security Code Review for Managed Code (Baseline Activity), Microsoft Corporation, October 2005, http://msdn2.microsoft.com/en-us/library/ms998364.aspx. **9.** FxCop Home Page, http://msdn2.microsoft.com/en-us/library/bb429476(VS.80).aspx. **10.** *K.Cwalina, B.Abrams.* Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries, Addison-Wesley Professional, 2005. **11.** *R.Neches, R.E.Fikes et al.* Enabling technology for knowledge sharing. AI Magazine 12(3), 1991, p.36–56. **12.** *T.R.Gruber*. A translation approach to portable ontology specification. Knowledge Acquisition 5(2), 1993, p.199–220. **13.** *R.Studer, V.R.Benjamins, D.Fensel.* Knowledge Engineering: Principles and Methods. IEEE Transactions on Data and Knowledge Engineering, Vol.25(1-2), 1998, p.161–197. **14.** *A.Gómez-Pérez, M.Fernández-López, O.Corcho.* Ontological Engineering, Springer-Verlag, London 2004. **15.** *Т.А. Гаврилова, В.Ф. Хорошевский.* Базы знаний интеллектуальных систем, СПб.:Питер, 2000. **16.** *C. Calero , F. Ruiz, M. Piattini.* Ontologies for Software Engineering and Software Technology, Springer-Verlag, Berlin Heidelberg 2006.

Поступила в редколлегию 20.02.08

УДК 004.728.8

К.І. ЯЦЕНКО, аспірант факультету кібернетики Київського Національного Університету імені Тараса Шевченка, kyatsenko@gmail.com

ПРОТОКОЛИ ДЛЯ ІНТЕГРАЦІЇ МОБІЛЬНИХ ПРИСТРОЇВ ІЗ СТАНЦІЯМИ ПО НАДАННЮ ПОСЛУГ ЧЕРЕЗ БЕЗДРОТОВІ ЕЛЕКТРОННІ КАНАЛИ

Запропоновано новий метод в області інформаційних технологій по наданню локальних послуг через безшовну інтеграцію мобільних пристроїв. Підхід базується на класі протоколів XDEP (XML data exchange protocol). Основним завданням цього підходу є заміна існуючих методів надання інформації, які функціонують завдяки спеціалізованим пристроям. Пропонується перенести цю функціональність на пристрої масового використання - персональні електронні записники, мобільні телефони, MP3 плеєри, та інші.

A new approach in the area of information technologies of providing local services via seamless integration of mobile devices has been proposed. The approach resides on the basis of XDEP (XML data exchange protocol). The solution is intended to substitute existing methods of information services providing, which reside on a specialized hardware. The functionality is to migrate to the commonly used devices like PDAs, Mobile Phones, MP3 players, etc.

1. Вступ. В сучасних динамічних умовах розвитку інформаційних технологій, рівень інтеграції мобільних пристроїв, персональних комп'ютерів та серверів через електронні канал є високим. Кожний мобільний пристрій містить персональну інформацію, яку користувачі потребують для вирішення атомарних питань. Персональні комп'ютери акумулюють інформацію, яку користувачі збирають на протязі деякого часу. Серверні системи представляють засіб довготривалого зберігання інформації від різних користувачів. Таким чином, інформацію, яку користувачі потребують в різні періоди часу можна класифікувати на три категорії:

Використання для дії (зберігається на мобільних пристроях)

Використання для діяльності (набір дій, зберігається на персональному комп'ютері)

Використання для історії та планування (набір різних діяльностей, які зберігаються на серверній платформі)

За цією класифікацію, дані зберігаються та організовуються на різних фізичних платформах. Такий підхід до зберігання інформації потребує постійної синхронізації обладнання. Логічний процес взаємодії різних програмно-апаратних комплексів розглядається в цій статті, яка пропонує уніфікований метод його організації.

Клас протоколів XDEP (XML Data Exchange Protocol) вперше представлений під час конференції ICTA 2007 [1]. Головною ідеєю цього класу