УДК 519.68

С.Ю. Шабанов-Кушнаренко, А.А. Мамедов

Харьковский национальный университет радиоэлектроники, Харьков

ПОСТРОЕНИЕ ПРЕДИКАТНОЙ МОДЕЛИ ПРОЦЕССА ФОРМИРОВАНИЯ ПРОГРАММНОЙ АРХИТЕКТУРЫ

Рассмотрена проблема усовершенствования метода контроля процесса создания программных средств, что позволяет повысить эффективность процесса разработки ПО в условиях изменяющихся требований. Предложен подход к использованию гибких методологий разработки, позволяющий своевременно и эффективно вносить изменения в проект в соответствии с новыми требованиями, а также выявлять и устранять отклонения от желаемого результата на ранних этапах разработки продукта за счет постоянной верификации продукта заказчиком.

Ключевые слова: *гибкие методологии разработки ПО, жизненный цикл ПО, управление проектами.*

Введение

Программное обеспечение – важнейший вид обеспечения информационной системы, обеспечивающий практическую реализацию процессов обработки информации. На разработку программного обеспечения затрачивается большая часть средств, выделенных на проектирование информационной системы в целом. В связи с этим важно обеспечить применение эффективной методологии разработки программного продукта.

На протяжении всего жизненного цикла разработки программного продукта необходимо осуществлять контроль своевременности и качества разработки. От результатов выполнения каждого этапа зависит общий успех программного проекта. Эффективное построение процесса разработки программного продукта позволит снизить риски к минимуму, а также максимально учесть требования заказчика. Существует множество современных методологий разработки программного обеспечения. Однако существующие методологии не являются универсальными и имеют применение лишь в проектах определенного типа [1].

Поэтому актуальной является проблема усовершенствования метода контроля процесса создания программных средств, что позволяет повысить эффективность процесса разработки ПО в условиях изменяющихся требований.

Традиционный подход к управлению проектами является линейным, где все делается в одном цикле. Все подробно планируется, и по результатам выполнения всех этапов жизненного цикла проект сдается целиком, и это – основное отличие традиционного подхода от гибкого, или Agile-подхода [2].

В настоящее время методология Agile достаточно популярна и предлагает эффективные пути решения многих проблем, существующих в проектах разработки программных продуктов.

При использовании гибкой методологии разработки планируется только лишь некоторый ограниченный функционал. После завершения этапа разработки заказчик может видеть работающую версию программного продукта, а также оценить, в правильном ли направлении он движется. Это позволяет заказчику определить новые либо переопределить существующие требования. Такой подход к разработке предусматривает возможные изменения в требованиях заказчика и позволяет применить эти изменения во время разработки проекта.

Использование гибких методологий разработки позволяет своевременно и эффективно вносить изменения в проект в соответствии с новыми требованиями. Также это дает возможность выявить и устранить отклонения от желаемого результата на ранних этапах разработки продукта за счет постоянной верификации продукта заказчиком.

Большинство гибких методологий направлены на сведение риска к минимуму путем уменьшения итераций – коротких отрезков времени, которые обычно длятся одну-две недели. Каждая итерация выглядит как сокращённый жизненный цикл разработки и включает в себя все задачи: планирование, анализ требований, проектирование, кодирование, тестирование и документирование. Несмотря на то, что результатов работы отдельной итерации недостаточно для выпуска новой версии продукта, каждая итерация на выходе имеет работающий продукт с реализованным функционалом, запланированным заранее. Заказчики могут видеть результат разработки на протяжении всего жизненного цикла проекта.

Цель и задачи исследования

Целью исследования является усовершенствование метода построения процесса контроля создания программных средств на основе гибких методологий разработки, что позволит повысить эффективность процесса разработки ПО в условиях изме-

няющихся требований. Гибкие методологии основываются на адаптации итерационного процесса разработки к особенностям решаемых задач и производительности команды [3]. В то же время формализованные методы адаптации отсутствуют, что требует построения моделей реально выполняющихся процессов разработки и их усовершенствование на следующей итерации.

Исходными данными для методов Process mining являются журнальные файлы (логи) информационных систем. Проблема входных данных состоит в том, что для анализа может быть использован не каждый журнальный файл. Журнал событий должен иметь формат данных, пригодных для методов ИАП. Приложение ProM использует стандартный формат для журнальных файлов – MXML. MXML описывает процесс как последовательность событий. Поэтому в рамках построения модели необходимо модифицировать названий задач с учетом их статуса и исполнителя для формирования последовательности уникальных событий, имена которых состоят из комбинации названия задачи, ее состояния и исполнителя.

Анализ структуры журнального файла

В данный момент существует множество процессно-ориентированных систем, которые используют свои правила организации ЖЦ БП, и часто анализ функционирования таких систем затруднителен в силу особенностей архитектуры и организации реализации тех или иных бизнес функций. Общим свойством всех этих систем является то, что они все ведут журнальные файлы регистрации событий. Источником информации для технологий анализа процессов является журнал событий информационных систем. Однако не все протоколы событий подходят для анализа процессов – нужно, чтобы в нем была информация, достаточная для применения методов Process mining. К необходимым требованиям можно отнести следующее:

– все события, записанные в протоколе, должны быть идентифицированы с экземплярами процессов;

– все события должны быть упорядочены по времени их выполнения;

– разнотипные события должны различаться.

На основании проведенного анализа была выделена структура журнальных файлов систем управления проектами. Таким образом, во всех проанализированных журнальных файлах имеются следующие общие атрибуты:

– временная метка;

– задача (идентификатор задачи);

– состояние события;

– пользователь (идентификатор пользователя);

– дополнительные атрибуты действия (сложность, затраченное время, приоритет и др.).

Согласно структуре журналов регистрации событий в системе [4], каждая запись характеризуется следующим набором параметров:

$$S_A =< T, A, U, O >,$$

где S_A – журнальный файл; T – временной параметр; A – действие (событие); U – исполнитель; O – объект процесса, над которым совершаются действия.

В общем виде для журнальных файлов характерны следующие особенности:

– зарегистрированные события отражают элементарные действия в информационной системе в рамках некоторого процесса;

– зарегистрированные события имеют метку времени выполнения;

– зарегистрированные события связаны исполнителями и/или некоторыми ресурсами.

Использование методов анализа журнальных событий систем контроля разработки программных продуктов дает следующие преимущества:

– существует возможность получить результаты о выполнении процессов (конкретных задач), основанные на реальных фактах, отраженных в журнале регистрации событий;

– существует возможность построить «черновую» модель бизнес-процессов или уточнить существующую;

– существует возможность выявления потенциала для оптимизации и «узких места» бизнес-процессов;

– на основании полученных результатов можно организовать эффективный контроль выполнения бизнес-процессов;

– полученные результаты можно использовать в дальнейшем для прогнозирования/планирования.

Для закрепления требований и унификации протоколов, обрабатываемых алгоритмами Process Mining, был предложен стандарт записи протоколов MXML.

MXML – это расширяемый формат, основанный на языке разметки XML

Он используется для представления и хранения информации в виде логов событий. Формат фокусируется на ключевой информации, необходимой для применения методов Process Mining, однако существует возможность расширения формата для записи дополнительной информации [5].

Корневым узлом каждого MXML-документа является WorkflowLog, представляющий лог-файл. Каждый WorkflowLog может содержать произвольное количество узлов Process. Каждый элемент типа Process является группой событий, которые произошли в течение выполнения какого-либо процесса. Од-

нократные выполнения этого процесса представлены элементами типа ProcessInstance. Таким образом, каждый ProcessInstance представляет собой однократное протекание процесса. Каждый ProcessInstance содержит группу из произвольного количества элементов типа AuditTrailEntry (контрольные записи), каждая из которых соответствует уникальному событию в логе. Каждая контрольная запись должна содержать как минимум два элемента: WorkflowModelElement (название задачи, которая была выполнена) и EventType (тип события, который описывает стадию выполнения задачи). Формат также поддерживает два необязательных, но, тем не менее, часто встречающихся поля. Timestamp содержит точную дату и время, когда событие произошло, а Originator идентифицирует ресурс, т.е. человека или информационную систему, которые являлись инициатором события. Расширяемое поле Data может содержать произвольное количество атрибутов, которые являются парами строк <название-значение>.

Структура MXML-формата изображена на рис. 1 в виде диаграммы классов.

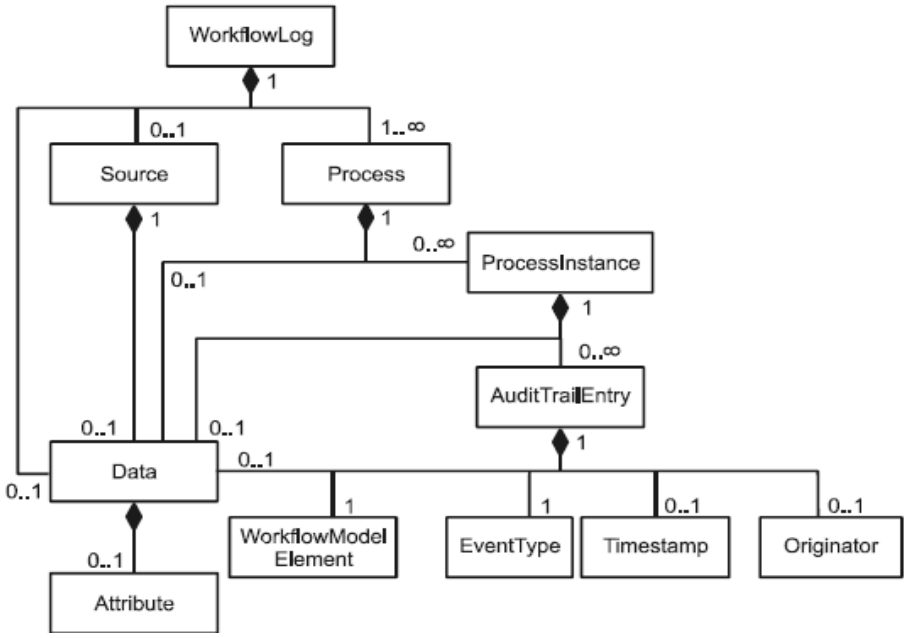

Рис. 1. UML-диаграмма MXML-формата

Процесс представляет собой граф P, вершины которого отражают состояние S моделируемой системы в различные моменты времени. Ребра графа R отражают действия системы. Функционирование процесса происходит с начального состояния и моделируется переходами между вершинами по ребрам S_0: $r_j \in R, j = \overline{1,J}$.

При этом с каждым таким переходом связаны конкретные действия a_r из множества A допустимых действий процесса: $R \subseteq S \times A \times S$.

Таким образом, процесс описывается графом, вершины которого отражают состояния моделируемой динамической системы, дуги – переходы между этими состояниями. Переходы связаны с выполнением действий над объектами процесса: $P = (S, s_0, R, A, O)$.

Функционирование процесса представляет собой последовательность переходов между его состояниями, причем каждый из таких переходов связан с выполнением соответствующих действий:

$$s_0 \xrightarrow{a_0} \ldots s_i \xrightarrow{a_i} s_{i+1} \xrightarrow{a_{i+1}} \ldots s_I,$$
$$s_i \in S, a_i \in A, i = \overline{0,I}.$$

Существует стандартный набор состояний [6]:

– To do – задача состоит в очереди выполнения;

– Blocked – задача заблокирована по каким-либо причинам;

– In development – задача выполняется;

– Ready for testing – задача выполнения и готова к тестированию;

– In testing – задача тестируется;

– Done – задача протестирована и полностью завершена.

Первоначально процесс находится в состоянии S_0, в дальнейшем на каждом i-шаге – в состоянии S_i. В дальнейшем переход между состояниями происходит следующим образом. В i-состоянии процесс выбирает дугу с действием a_i, которое может быть выполнено в данный момент времени. Затем действие a_i выполняется, и процесс переходит в следующее состояние S_{i+1}. Процесс продолжается до достижения одного из конечных состояний.

Выполнение одного из возможных в текущем состоянии действий процесса зависит от входного состояния процесса и набора выполненных действий. Конечным состоянием является такое, из которого отсутствуют дуги в другие состояния процесса.

Реализация каждого процесса (след, трасса процесса) представляет собой последовательность действий, переводящих процесс из начального состояния в конечное. В общем случае каждый процесс обладает множеством трасс. След процесса отражается в рассмотренном выше файле лога.

Для каждого процесса P могут быть заданы ограничения с учетом атрибутов операций. Так, ограничение по объектам определяется следующим образом. Пусть P – процесс, а O* – подмножество объектов, над которым выполняются операции A данного процесса.

Тогда ограничением процесса P по множеству объектов O* является граф P*, который получен из графа P удалением тех операций, которые связаны с обработкой объектов из подмножества O*.

Предикатная модель процесса

Состояния будем описывать переменными $x_1, x_2, ..., x_n$. При анализе полного лога выполняется его разбиение на более мелкие фрагменты, которые в дальнейшем могут быть объединены последовательно, параллельно, циклически или через выбор. Последовательность операций процесса представим в виде конъюнкции бинарных предикатов:

$$R_1(x_1, x_2) \wedge R_2(x_2, x_3) \wedge ... \wedge R_n(x_n, x_{n+1}).$$

Для выделения шаблонов процесса выделим базовые элементы ветвления, цикла, параллельности между последовательностями событий и представим их в виде предикатных операций:

Ветвление может быть реализовано операцией исключающего или неисключающего выбора:

$$XOR(R_i, R_k) = R_i \oplus R_k, OR(R_i, R_k) = R_i \vee R_k;$$

параллельность – операцией логического И:

$$AND(R_i, R_k) = R_i \wedge R_k;$$

цикл – предикатом, задающим число повторений в цикле или условие повторения t:

$$\Omega(R_i, R_k, t).$$

Модель гибкого процесса в максимально обобщенном виде представляется как система бинарных предикатов:

$$M = \{R_j \mid j = \overline{1, n}\}.$$

Выводы

В работе был произведен анализ гибких методологий разработки программных проектов, выявлены их преимущества и недостатки, среди которых отсутствие модели процесса разработки, что затрудняет выявление узких мест процесса разработки.

В рамках работы был усовершенствован процесс ретроспективы гибкой методологии Scrum путем построения модели процесса разработки прошедшего этапа с использованием методов Process mining, что позволяет более эффективно выявлять причины неэффективной разработки.

Список литературы

1. Хамбл Д. Непрерывное развертывание ПО. Автоматизация процессов сборки, тестирования и внедрения новых версий программ [Текст] / Д. Хамбл. – М.: Вильямс, 2011. – 361 с.

2. Мартин Р. Быстрая разработка программ. Принципы, примеры, практика [Текст] / Р. Мартин. – Tennessy, 2011. – 264 с.

3. Cohn M. Agile checklist [Текст] / M. Cohn. – Tennessy, 2011. – 123 с.

4. Aalst van der W. Business Process Management Demystified: A Tutorial on Models, Systems and Standards for Workflow Management / W. Aalst van der, J. Desel, W. Reisig, G. Rozenberg // Springer-Verlag. – 2004. – P. 1-65.

5. Барсегян А. Анализ данных и процессов 3-е издание [Текст] / А. Барсегян, М. Куприянов, И. Холод. – М.: Мир, 2009. – 513 с.

6. Crispin L. Agile Testing [Текст] / L. Crispin. – NY, 2013. – 200 с.

Поступила в редколлегию 15.04.2015

Рецензент: д-р техн. наук, проф. С.Ф. Чалый, Харьковский национальный университет радиоэлектроники, Харьков.

ПОБУДОВА ПРЕДИКАТНОЇ МОДЕЛІ ПРОЦЕСУ ФОРМУВАННЯ ПРОГРАМНОЇ АРХІТЕКТУРИ

С.Ю. Шабанов-Кушнаренко, А.О. Мамедов

Розглянуто проблему вдосконалення методу контролю процесу створення програмних засобів, що дозволяє підвищити ефективність процесу розробки ПЗ в умовах вимог, що змінюються. Запропоновано підхід до використання гнучких методологій розробки, що дозволяє своєчасно і ефективно вносити зміни в проект відповідно до нових вимог, а також виявляти та усувати відхилення від бажаного результату на ранніх етапах розробки продукту за рахунок постійної верифікації продукту замовником.

Ключові слова: *гнучкі методології розробки ПЗ, життєвий цикл ПЗ, управління проектами.*

CONSTRUCTION OF THE SOFTWARE ARCHITECTURE FORMATION PREDICATE MODEL

S.Yu. Shabanov-Kushnarenko, A.O. Mamedov

The problem of improving the method of control of the process of creating software that improves the efficiency of the software development process in a changing requirements. An approach to the use of agile development, enabling timely and efficient manner to make changes to the project in accordance with the new requirements, as well as to identify and correct deviations from the desired result in the early stages of product development through continuous verification of the product by the customer.

Keywords: *Agile software development, life cycle, project management.*