Лошманов Антон Юрьевич
Loshmanov Anton Yurjevich
ФГБОУ ВПО «Комсомольский-на-Амуре государственный технический университет»
Komsomolsk-na-Amure State Technical University
Доцент / Associate Professor
Кандидат физико-математических наук
E-Mail: loshmanov@kmscom.ru

Григорьев Ян Юрьевич
Grigorjev Yan Yurjevich
ФГБОУ ВПО «Комсомольский-на-Амуре государственный технический университет»
Komsomolsk-na-Amure State Technical University
Доцент / Associate Professor
Кандидат физико-математических наук
E-Mail: jan198282@mail.ru

Петрова Анна Николаевна
Petrova Anna Nikolaevna
ФГБОУ ВПО «Комсомольский-на-Амуре государственный технический университет»
Komsomolsk-na-Amure State Technical University
Доцент / Associate Professor
Кандидат технических наук
E-Mail: petrovaan2006@yandex.ru

Технические науки

Организация работ по сопровождению информационной системы ВУЗа

Support of information system of higher education institution

Аннотация: Работа посвящена вопросам сопровождения программ и информационных систем применительно к вузу. По различным оценкам сопровождение подобных систем составляет от 40 до 90% стоимости всего жизненного цикла приложения. Конечно, в указанные расчеты затрат на сопровождение включены усовершенствования, которые было лучше считать дополнительными разработками. В любом случае объем работ по сопровождению систем обычно достаточно велик [1]. Учитывая особенности функционирования вузов в современных условиях, опыт других вузов по разработке своих собственных информационных систем и внедрению готовых решений, а также учитывая трудности при сопровождении и тех, и других систем, авторы данной статьи делают выбор в пользу создания своей корпоративной информационной системы [2].

The Abstract: Work is devoted to questions of maintenance of programs and information systems in relation to higher education institution. By various estimates support of similar systems makes from 40 to 90% of cost of all life cycle of the application. Certainly, improvements which were better to be considered as additional development are included in the specified calculations of expenses for maintenance. In any case the amount of works on support of systems is usually enough great [1]. Considering features of functioning of higher education institutions in modern conditions, experiment of other higher education institutions on development of own information systems and introduction of ready decisions, and also considering difficulties at maintenance and those, and other systems, authors of this article make a choice for creation of the corporate information system [2].

Ключевые слова: Информационные системы; методология управления проектом; сопровождение.

Keywords: Information systems; management methodology of the project; support.

***

Введение

В 80-е и 90-е годы в области разработки *программного обеспечения (ПО)* преобладали две тенденции. Одна – это быстрый рост приложений, в том числе создаваемых для Web. Другая – это расцвет инструментальных средств и парадигм (подходов к проектированию, таких как объектно-ориентированный [3]).

Однако, несмотря на появление новых тенденций, основные этапы разработки программного обеспечения остались неизменными: *определение* процесса разработки ПО, *управление* проектом разработки, *описание* целевого программного продукта, *проектирование* продукта, *разработка* продукта, *тестирование* частей продукта, *интеграция* частей и тестирование продукта в целом, *сопровождение* продукта [1]. Указанные этапы разработки ПО с незначительными изменениями являются и этапами проектирования информационных систем (ИС).

Лехман [7-8] предложил «закон», заключающийся в том, что любая система, адаптацией которой к изменяющимся условиям никто не занимается, с течением времени неизбежно теряет свою ценность. Другими словами, если система остается неизменной, польза от нее постепенно убывает.

Общие вопросы сопровождения систем

Программное обеспечение в составе информационных систем является одним из наиболее гибких видов продукции, которое часто подвергается изменениям в течение всего времени его использования. Иногда достаточно при корректировке внести одну-две ошибки для того, чтобы резко снизилась надежность ПО или его корректность при некоторых входных данных. Для сохранения и повышения качества систем необходимо регламентировать процесс модификаций и поддерживать его соответствующим тестированием и контролем качества. В результате информационная система со временем чаще всего улучшается и по функциональным возможностям, и по качеству решения отдельных задач [4].

Беннет [6] упорядочил проблемы, связанные с сопровождением программ, разделив их на три категории.

1. Проблемы управления:

- трудность выявления прибыли на инвестированных капитал.

2. Проблемы обработки:

- для облуживания потока запросов на сопровождение требуется жесткая координация.

3. Проблемы технического характера:

- трудность учета всех результатов изменений;
- высокая стоимость тестирования по сравнению с пользой от отдельных изменений: идеальным было бы сосредоточенное тестирование, но оно стоит дорого. Невозможно полностью исключить регрессионное тестирование.

Руководство обычно гораздо сильнее интересуется процессом поставки приложений. Кроме того, затраты на сопровождение программы трудно классифицировать как прибыль на инвестированный капитал.

Если бы мы имели дело с одним конкретным изменением, проблемы были бы еще не так велики. Обычно организации сталкиваются с непрерывным потоком запросов на сопровождение. Большое количество изменений позволяет снизить стоимость каждого из них, но поток запросов создает повышенные требования к системе обработки. Усилия программистов, тестеров и разработчиков документации нужно координировать [1].

Работы, обеспечивающие контроль и повышение качества, а также развитие функциональных возможностей информационных систем, составляют процесс сопровождения, который обычно включает:

- исправление ошибок – корректировка программ, выдающих неверные результаты в условиях, ограниченных техническим заданием и документацией (в процессе сопровождения требуют около 20% затрат);
- регламентированная документами адаптация к условиям конкретного использования, обусловленным характеристиками внешней среды или конфигурацией аппаратных средств, на которой предстоит функционировать программам (около 20 % общих затрат);
- модернизация – расширение функциональных возможностей или улучшение характеристик решения отдельных задач в соответствии с новым или дополненным техническим заданием на информационную систему (до 60 % общих затрат).

Первый вид изменений является непредсказуемым и его трудно регламентировать заранее. Остальные виды корректировок носят вполне упорядоченный характер и проводятся в соответствии с заранее подготавливаемыми планами и документами. Эти корректировки в наибольшей степени изменяют компоненты системы и требуют наибольших затрат ресурсов.

Со временем, иногда через десятки лет, сопровождение системы прекращается. Это может быть обусловлено разработкой более совершенных систем, прекращением использования сопровождаемой системы или нерентабельным возрастанием затрат на ее сопровождение. Для того, чтобы со временем прийти к обоснованному решению о прекращении сопровождения, необходимо периодически оценивать эффективность эксплуатации и возможный ущерб от отмены сопровождения (в отдельных случаях решение о прекращении сопровождения принимается при противодействии со стороны отдельных пользователей).

Организация процесса сопровождения

Некорректные изменения используемых систем могут вызвать значительный ущерб; поэтому необходимо их тщательно проверять. На завершающих стадиях комплексной отладки в процессе эксплуатации и сопровождения сложных ИС применяются методы конфигурационного управления, которое необходимо и особенно эффективно при сопровождении широко тиражируемых сложных или сильно распределенных ИС, используемых одновременно в нескольких версиях.

История сопровождаемого программного продукта может быть очень непростой. Многие существующие приложения часто оказываются снабжены скудной или противоречивой документацией. Первым шагом в организации рациональных работ по сопровождению таких приложений должно быть создание подробной согласованной

документации. Для этого часто требуется восстановление архитектуры по исходному коду – *обратное проектирование* (reverse engineering). Существует несколько программных средств, помогающих выполнить эту задачу. Например, пакеты Rational Rose (Rational Corporation) и Together (Object International) позволяют создавать модели объектов из исходного кода на C++ и Java. Получающиеся в результате диаграммы можно использовать для анализа хода мыслей разработчика. Из-за неоднозначности трактовки диаграмм на UML и механичности процесса обращения, обратное проектирование не позволяет полностью понять намерения разработчика.

В 1980 году в книге Хаммера и Чампи о реинжиниринге бизнес-процесса (BPR – Business Process Reengineering) компаниям предлагалось заново взглянуть на процесс производства ценностей для потребителей, после чего спроектировать его заново. Хаммер и Чампи основное внимание уделяли процессу как целому, а не отдельным его частям. Процесс начинается с входных данных, например заказов, заканчивается окончательной отгрузкой товаров или предоставлением услуг и включает все способствующие достижению цели элементы, такие как поддержка программ и вклад сотрудников. Реинжиниринг заставляет взглянуть на требования к программным приложениям как на часть требований к предприятию (компании, организации и т.п.) в целом. Получающиеся приложения иногда называются корпоративными приложениями, хотя обсуждаемая концепция используется и вне контекста реинжиниринга бизнес процесса. Реинжиниринг обычно заставляет ответить на вопрос о том, как должны работать существующие приложения, а не о том, как они работают.

Реинжиниринг относят к сопровождению, потому что он требует перепланирования приложений [1].

Информационная система университета

В Комсомольском-на-Амуре государственном техническом университете (КнАГТУ) с лета 2012 года проектируется и разрабатывается электронная система. Изначально основной идеей разработки была идея минимальной зависимости от разработчика, поэтому в основу системы легли конструкторы ролей, видов деятельности, отчетов. Конструктор ролей обеспечивает возможность редактировать субъекты и объекты доступа, группируя последние по принадлежности к модулям. Задачи модуля «Конструктор видов деятельности» – это предоставление комфортной возможности пользователю системы создать и описать новый вид деятельности для информационной системы; поддержка системы шаблонов, позволяющих на их основе конструировать новые виды деятельности. Задачи «Конструктора отчетов» – это предоставление интерфейсов получения значений показателей генерирования отчетных значений; генерация файлов отчетов на основе шаблонов.

В данный момент активно ведется разработка и началось внедрение следующих модулей: «Унифицированная система личных кабинетов», «Дополнительные образовательные услуги», «Система мониторинга текущей успеваемости», «Система для проведения on-line олимпиад», «Система сбора и анализа показателей деятельности подразделений ВУЗа».

В процессе эксплуатации текущей версии ИС у пользователей выявляются некоторые претензии к ее функционированию, которые пользователями обычно квалифицируются как ошибки эталонной или собственной версии. Ряд таких «ошибок» обусловлен недостаточной квалификацией пользователя (или из-за недостатков документации на ИС, или вследствие сбоев в аппаратуре). Для установления достоверности сообщений о выявленных ошибках производилась регистрация условий, при которых проявлялись подобные «аномалии», и предварительное тестирование версии программ для выявления не подтверждающихся ошибок.

От пользователей университета также поступали предложения по внесению изменений

в текущую версию для улучшения эксплуатационных характеристик и расширения функциональных возможностей ИС. Такие же предложения могут поступать от разработчиков системы. На базе предложений группой разработчиков создается документ – исходные данные для планирования доработок и тестирования системы в процессе сопровождения.

При разработке ИС в КнАГТУ используются две разновидности тестирования: тестирование модулей (unit testing) и приемочное тестирование (acceptance testing).

Тесты модулей разрабатываются программистами по мере того, как они пишут код. Тесты функциональности разрабатываются заказчиком после того, как он формулирует пожелания. Тесты модулей в любой момент времени сообщают разработчику о том, что разработанный код функционирует корректно. Приемочные тесты сообщают всей команде о том, что разрабатываемый продукт делает именно то, чего хочет пользователь.

Для разработки команда использует объектно-ориентированный язык C#, поэтому разработчики пишут тесты для каждого метода, корректность работы которого может быть нарушена. Разработка тестов для метода выполняется до того, как будет написан рабочий код этого метода. После того как разработаны все возможные тесты, разрабатывается код метода. Для многих такой подход кажется весьма странным, однако он вполне оправдан. Благодаря тому, что код тестов пишется до того, как разрабатывается код метода, мы имеем:

- наиболее полный набор всевозможных тестов;
- наиболее простой код, который (скорее всего) реализует заданную функциональность;
- четкое представление о том, какую задачу решает код.

Тесты модулей должны выполняться автоматически, в результате их работы разработчик должен получить четкий, недвусмысленный ответ: «код работоспособен» или «код неработоспособен».

Заказчик отвечает за разработку приемочных тестов для каждого из сформулированных им пожеланий. Заказчик может написать приемочные тесты самостоятельно, однако это вовсе не обязательно. Он может привлечь для этой цели других сотрудников организации (например, сотрудников отдела контроля качества, бизнес-аналитиков и др.). В идеале заказчик завершает разработку приемочных тестов для некоторой итерации (iteration) еще до того, как программисты завершают работу над этой итерацией. Приемочные тесты должны запускаться автоматически. Разработчики должны запускать приемочные тесты достаточно часто, чтобы убедиться в том, что в ходе реализации новых функций системы никоим образом не нарушена корректность работы существующих механизмов.

Вовсе не обязательно каждый раз выполнять абсолютно все приемочные тесты. Приемочные тесты позволяют заказчику получить представление о том, насколько успешно продвигается работа над проектом. Приемочные тесты также позволяют заказчикам принимать обоснованное решение о том, готова ли очередная версия (release) продукта к использованию или нет.

Частое усовершенствование программы требует не просто изменения отдельных строк кода, а приложения гораздо больших усилий, не достигающих, однако, масштабов полного реинжиниринга. Иногда этот процесс называют рефакторингм.

Рефакторинг – это методика улучшения кода без изменения его функциональности. Существуют две возможности выполнить рефакторинг: до реализации некоторой функциональности и после реализации некоторой функциональности. Разработчики из команды КнАГТУ пытаются определить, можно ли модифицировать существующий код таким

образом, чтобы упростить реализацию новой функциональности. Разработчики смотрят на только что написанный ими код, чтобы понять, существует ли способ еще более упростить его. Например, если они приходят к выводу, что можно абстрагировать код, они выполняют рефакторинг, удаляя дублирующийся код из конкретных реализаций.

Состояние разрабатываемого продукта постоянно меняется. Рефакторинг позволяет нам модифицировать код в соответствии с новыми полученными нами знаниями, и при этом не нарушать работу тестов. Благодаря этому код сохраняется чистым и аккуратным. А это в свою очередь означает, что код будет служить нам дольше, количество связанных с ним проблем резко снизится, а разработчики, которые будут работать с ним в будущем, без труда поймут его предназначение.

Проблемы с тестированием и рефакторингом привели команду разработчиков ИС КнАГТУ к пониманию необходимости частой интеграции разрабатываемой системы.

Если мы будем выполнять интеграцию разрабатываемой системы достаточно часто, то мы сможем избежать большей части связанных с этим проблем. В традиционных методиках интеграция, как правило, выполняется в самом конце работы над продуктом, когда считается, что все составные части разрабатываемой системы полностью готовы.

Если интеграция выполняется довольно часто, то причины возникающих при этом проблем становятся более очевидными. Предположим, нам надо добавить в систему новый, только что разработанный код. Интегрировав код в систему, мы обнаруживаем, что система работает неправильно. До интеграции все тесты корректно срабатывали, значит, причина проблемы связана с только что добавленным кодом. Благодаря этому команда работает с максимальной скоростью. В рамках традиционных методик разработчики длительное время по отдельности занимаются разработкой отдельных частей системы. Закончив работу каждый на своем участке, они собирают код воедино, выполняя интеграцию, а затем тратят значительное время на устранение возникающих при этом проблем. Система долго разрабатывается как набор не связанных между собой частей, из-за этого устранение потенциальных интеграционных проблем длительное время откладывается на потом. Проблемы накапливаются, благодаря чему усложняется их устранение. Постоянная интеграция делает процедуру устранения интеграционных проблем частью ежедневного процесса разработки.

Членами команды разработчиков было выработано правило: версии продукта должны поступать в эксплуатацию как можно чаще. Работа над каждой версией должна занимать как можно меньше времени. При этом каждая версия должна быть достаточно осмысленной с точки зрения полезности для заказчика.

Уже на начальном этапе разработки ИС, участниками проекта было принято решение, что все члены команды в ходе работы должны соблюдать требования общих стандартов кодирования.

Если в команде не используются единые стандарты кодирования, то разработчикам становится сложнее выполнять рефакторинг, то есть переделывать общий код. Команда сформировала набор правил кодирования и структуры баз данных, а затем каждый член команды стал следовать этим правилам в процессе работы.

Таким образом, несколько практик экстремального программирования (XP) было принято нашей командой разработчиков и для проектирования, и для разработки ИС.

Заключение

Многие авторы [2] указывают ряд проблем, связанных с внедрением в вузе типовых решений таких, как, например ERP-системы. Одна из главных проблем – это сложность самого объекта управления – университета. Ведь сегодня он является многопрофильным

Главный редактор - д.э.н., профессор К.А. Кирсанов
тел. для справок: +7 (925) 853-04-57 (с 11^{00} – до 18^{00})
Опубликовать статью в журнале - http://publ.naukovedenie.ru

предприятием с большим количеством специфических бизнес-процессов. В то время как ERP-системы, внедряемые на основе типовых решений, изначально ориентированы на стандартизацию бизнес-процессов. К сказанному следует добавить и длительное время внедрения (чаще 1–3 года на основе готового программного решения). При разработке собственной ИС, первая проблема решается уже на первых этапах ее жизненного цикла (сбор требований и описание). Вторая же проблема, согласно выбранной методологии разработки ИС [5], будет решена значительным снижением срока внедрения.

В заключение следует отметить два немаловажных плюса в создании собственных информационных систем, которые обычно не учитываются сторонниками приобретения готовых решений. Дело в том, что любой подобный проект может стать основой для развития дальнейшей инновационной деятельности вуза. Так, в процессе выполнения проекта формируется коллектив, отрабатываются современные технологии и приемы, осваиваются наукоемкие методы и алгоритмы аналитической обработки данных, возникают новые программные решения [2]. А сбор и анализ требований на первой стадии разработки информационной системы может послужить стандартизации бизнес-процессов в университете, выявлении слабых сторон в управлении и, в конце концов, улучшению системы управления.

Главный редактор - д.э.н., профессор К.А. Кирсанов
тел. для справок: +7 (925) 853-04-57 (с 11^{00} – до 18^{00})
Опубликовать статью в журнале - http://publ.naukovedenie.ru

ЛИТЕРАТУРА

1. Брауде Э. Технология разработки программного обеспечения. – СПб.: Питер, 2004. – 655 с.: ил.
2. Глухих И.Н. Корпоративная информационная система университета на базе интернет / интернет-портала [Электронный ресурс] – Режим доступа: http://quality.petrsu.ru/file/71/2005-5-10.pdf (дата обращения 11.05.2013).
3. Попов А.В., Григорьева А.Л., Лошманов А.Ю. Объектно-ориентированных анализ, проектирование и программирование информационной системы университета // Современные проблемы науки и образования. 2012. № 6. С. 605-605.
4. Организация работ по сопровождению информационных систем [Электронный ресурс] – Режим доступа: http://referatplus.ru/economik/ste23.php (дата обращения 11.05.2013).
5. Петрова А.Н., Еськова А.В., Лошманов А.Ю. Проблема выбора методологии разработки информационной системы вуза // Современные проблемы науки и образования. – 2013. – № 2; URL: www.science-education.ru/108-8605 (дата обращения: 11.05.2013).
6. Bennett, K. «Software Maintenance: A Tutorial», Software Engineering, IEEE Computer Society, November, 1999.
7. Lechman, M. «Programs, Life Cycles, and the Laws of Software Evolution», Proc. IEEE, vol. 19, 1980, pp. 1060–1076.
8. Lechman, M. «Program Evolution Information Processing Management», Proc. IEEE, vol. 20, 1984, pp. 19–36.

Рецензент: Хромов Александр Игоревич, доктор физико-математических наук, профессор, главный научный сотрудник Института машиноведения и металлургии ДВО РАН.