Министерство образования
и науки Российской Федерации

Уральский
федеральный
университет
имени первого Президента
России Б.Н.Ельцина

Е.Б. Солонин

СОВРЕМЕННЫЕ МЕТОДИКИ РАЗРАБОТКИ ИНФОРМАЦИОННЫХ СИСТЕМ

Подготовлено кафедрой вычислительной техники

Методические рекомендации к самостоятельным работам
по курсу «Методы и средства проектирования информационных
систем и технологий» для студентов всех форм обучения.

Ориентированы на обучение основам современных информационных технологий. Предназначены для общего ознакомления с современными методами разработки информационных систем: RAD, MSF, Agile, XP, RUP, DSDM, Scrum.

Екатеринбург
2015

ОГЛАВЛЕНИЕ

1. ОСНОВНЫЕ ПОНЯТИЯ И ТЕРМИНЫ

Информационные системы (ИС) предприятий являются одним из важнейших ресурсов, без которых невозможно осуществлять производство товаров и услуг на уровне, достаточном для поддержания конкурентоспособности. Разработкой и эксплуатацией ИС занимается множество людей и организаций, затрачиваются огромные ресурсы. Поэтому так важны методики, позволяющие повысить эффективность систем, а также снизить стоимость их создания и использования.

Эксплуатации ИС на предприятии предшествует создание ее *проекта*. Под проектом ИС будем понимать документацию, в которой представлено описание проектных решений по созданию и эксплуатации ИС. Проектные решения определяют архитектуру системы, структуру хранения информации, состав и функциональные характеристики программных компонентов, характеристики технических средств.

Под *проектированием* ИС понимается процесс преобразования входной информации – сведений об объекте автоматизации и требований заказчика – в проект ИС. При этом существенным образом используются знания о методах проектирования ИС и системах-аналогах. *Объектами проектирования* являются отдельные элементы системы или их комплексы, относящиеся к функциональным или обеспечивающим подсистемам. Функциональные подсистемы реализуют основные функции системы (бизнес-функции), а обеспечивающие подсистемы поддерживают сервисные функции (архивирование данных, авторизацию пользователей и пр.).

В основе *метода проектирования* (также употребляются термины *методика* или *методология*) лежит алгоритм, который определяет проектные действия, их последовательность, состав исполнителей, средства и ресурсы, требуемые для выполнения этих действий. Процесс проектирования ИС делится на совокупность взаимосвязанных действий, каждое из которых может иметь свой объект.

Действия могут быть

- *проектировочными*, формирующими или изменяющими текущий проект;
- *оценочными*, вырабатывающими по установленным критериям оценку результатов проектирования.

Совокупность состояний, которые проходит ИС в своем развитии, от момента принятия решения о создании системы до момента прекращения ее функционирования, называется *жизненным циклом* (ЖЦ) информационной системы.

К основным требованиям, предъявляемым к выбираемой технологии проектирования, относятся следующие:

- созданный с помощью этой технологии проект должен максимально соответствовать требованиям заказчика, причем требования могут меняться уже в ходе создания ИС;
- технология должна максимально отражать все этапы жизненного цикла проекта и служить основой связи между проектированием и сопровождением системы в процессе ее эксплуатации;
- технология должна обеспечивать минимальные затраты времени и средств на проектирование и сопровождение системы при условии обеспечения должного качества конечного продукта.

Методы проектирования ИС можно классифицировать по уровню автоматизации и степени использования типовых проектных решений.

По степени автоматизации методы проектирования разделяются на:

- методы *ручного проектирования*, при котором проектирование ИС осуществляется без использования специальных инструментальных средств;
- методы *автоматизированного проектирования*, при котором производится генерация или настройка проектных решений на основе использования специальных инструментальных средств.

По степени использования типовых проектных решений различают следующие методы проектирования:

- *индивидуального проектирования*, когда проектные решения разрабатываются «с нуля» в соответствии с требованиями к ИС;
- *типового проектирования*, предполагающего сборку или конфигурацию ИС из готовых типовых компонентов.

Сочетание различных признаков классификации методов проектирования определяет характер метода проектирования, в который выделяют два основных класса: канонические и индустриальные методы [1]. Канонический метод основывается на технологии ручного индивидуального проектирования. Индустриальные методы базируются на технологии автоматизированного типового проектирования.

2. МОДЕЛИ ЖИЗНЕННОГО ЦИКЛА ИС

Технологии проектирования, применяемые в настоящее время, предполагают разработку системы в несколько стадий [1]. Типичное содержание жизненного цикла информационной системы сводится к реализации следующих стадий (в скобках приведены синонимы):

1. Планирование и анализ требований (предпроектная стадия). Включает исследование и анализ объекта и существующей информационной системы, определение требований к ИС, оформление технико-экономического обоснования (ТЭО) и технического задания (ТЗ) на разработку системы. В ТЭО должны быть представлены экономические расчеты, подтверждающие целесообразность разработки ИС. В ТЗ отражаются назначение ИС, требования к ИС, ее подсистемам и видам обеспечения, а также ограничения на ресурсы проектирования.

2. Проектирование (техническое проектирование, логическое проектирование). Разработка в соответствии со сформулированными требованиями состава автоматизируемых функций и состава обеспечивающих подсистем, структуры хранения информации, оформление технического проекта ИС.

3. Реализация (рабочее проектирование, физическое проектирование). Включает разработку программ, информационное наполнение баз данных, создание рабочих инструкций для персонала, оформление рабочего проекта. Реализация основывается на техническом проекте ИС.

4. Внедрение (тестирование, опытная эксплуатация). Комплексная отладка подсистем ИС, обучение персонала, поэтапное внедрение ИС по подразделениям предприятия, проведение приемо-сдаточных испытаний, передача ИС в эксплуатацию.

5. Эксплуатация (сопровождение, модернизация). Сбор рекламаций и статистики о функционировании ИС, исправление ошибок и недоработок,

адаптация системы к изменившимся условиям функционирования, формулирование требований к следующей версии ИС.

С точки зрения реализации этапов модели ЖЦ претерпели определенную эволюцию. Среди известных моделей ЖЦ можно выделить следующие:

- каскадная модель;
- итерационная (итеративная) модель;
- спиральная модель.

Хронология появления этих моделей соответствует их позиции в списке: каскадная модель датируется периодом до 70-х годов XX века, итерационная – 70–80 гг., спиральная – начиная с 80-х годов. И по сей день в разных проектах может использоваться любая из трех моделей, но по частоте применения преобладает спиральная модель.

Каскадная модель (англ. *waterfall model* – «модель водопада») подразумевает строго последовательное выполнение стадий без возвратов к предыдущим (рис. 1):

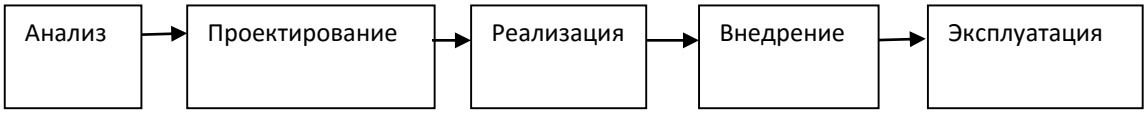

Рис. 1. Каскадная модель ЖЦ

Автором модели считается У.У. Ройс, опубликовавший свою статью в 1970 году. Каскадная модель подразумевает полное и успешное завершение каждого этапа перед переходом к следующему. Например, стадия «Анализ» должна быть проведена настолько полно, чтобы у проектировщиков не возникло никаких проблем, связанных с информацией о предметной области, требованиях заказчика, особых условиях создания системы и пр.

Применение каскадной модели к сложным проектам вследствие большой длительности процесса проектирования возможно лишь тогда, когда изменчивость требований к системе за это время невелика либо вообще отсутствует. В тех же случаях, когда требования меняются, а также если необходимо исправить ошибки и упущения, сделанные на более ранних

стадиях ЖЦ, необходимо повторное выполнение всех или некоторых работ этих стадий. Этот подход и реализован в итерационной модели.

Использование итерационной модели ЖЦ призвано минимизировать риски ошибок, совершенных на ранних стадиях разработки, а также облегчить взаимодействие с заказчиками системы за счет использования предварительных версий (прототипов). Итерационная модель предполагает возможность возврата к предыдущим стадиям жизненного цикла, если выявлена необходимость дополнительных работ по этим стадиям, или произошло изменение требований к системе (рис. 2). Как правило, осуществляется возврат к предыдущей стадии, хотя возможен вариант и более далекого возврата.

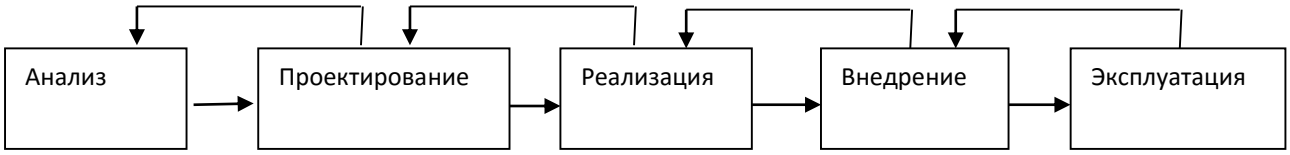

Рис. 2. Итерационная модель ЖЦ

Таким образом, каждая стадия жизненного цикла может простираться на весь процесс разработки и внедрения ИС.

В основе спиральной модели, предложенной Б. Боэмом в 1986 году, лежит технология создания ряда последовательных прототипов системы, все более точно отражающих требования заказчика. В отличие от итерационной модели, где возврат к предыдущим стадиям ЖЦ происходит только при возникновении проблем, в спиральной модели многократное прохождение стадий предусматривается изначально. Последовательность стадий от анализа да внедрения проходится «по спирали», и на каждом витке спирали создается очередная, более совершенная версия продукта (рис. 3).

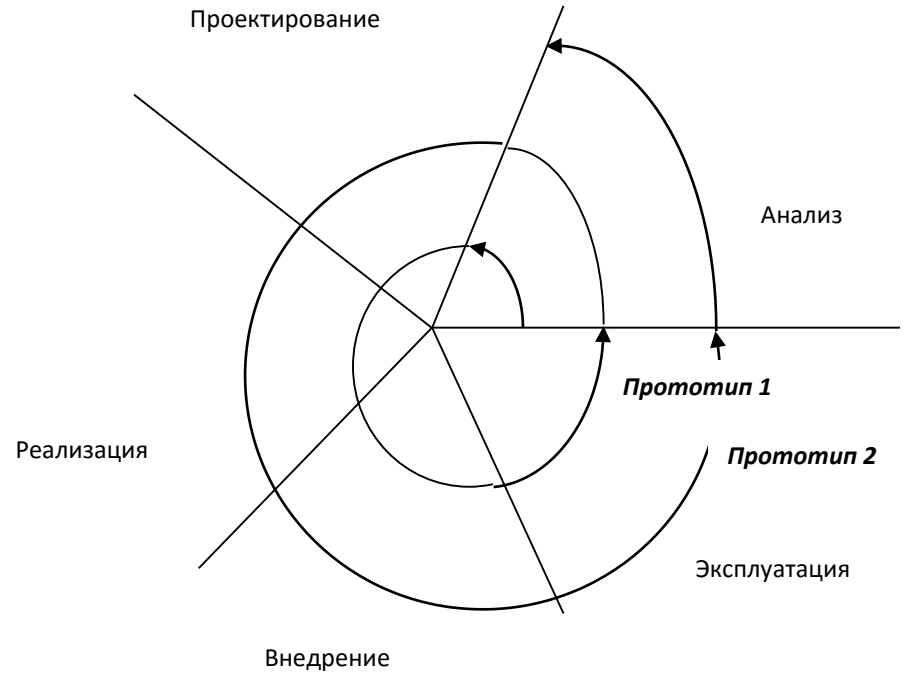

Рис. 3. Спиральная модель ЖЦ

Цель состоит в том, чтобы как можно быстрее предъявить заказчикам системы работоспособный продукт для оценки ими направления и хода работы, внесения уточнений и дополнений в свои требования.

Для реализации спиральной модели нужно правильно определить время перехода к очередному витку – к разработке следующей версии системы. Переход осуществляется в соответствии с планом, который составляется на основе статистических данных, полученных в предыдущих проектах, а также личного опыта разработчиков.

Многие современные технологии проектирования ИС либо используют саму спиральную модель, либо включают ее идеи и компоненты. Некоторые из таких технологий описаны в последующих разделах.

3. ТЕХНОЛОГИЯ RAD

Технология RAD (англ. Rapid Application Development) основана на спиральной модели жизненного цикла и обеспечивает ускорение разработки ИС благодаря широкому привлечению к процессу проектирования будущих пользователей [2]. Для данной технологии характерно перенесение основных объемов работ с предпроектной стадии на стадию проектирования. Представители заказчика получают возможность контролировать весь процесс создания системы, оперативно влиять на состав и реализацию ее функций.

Основателем RAD считается сотрудник IBM Дж. Мартин, который в 1980-х годах сформулировал основные принципы RAD, основываясь на идеях Б. Бема и С. Шульца. В 1991 году Дж. Мартин опубликовал книгу, в которой детально изложил концепцию RAD и возможности ее применения.

3.1. Особенности технологии RAD

Применение технологии RAD целесообразно в тех случаях, когда:

1. Требования к программному обеспечению (ПО) определены нечетко или не полностью. Во многих случаях заказчик весьма приблизительно представляет себе работу будущей системы и не может четко сформулировать все требования к ней.
2. Интерфейс пользователя является для заказчика главным фактором. RAD-технология дает возможность продемонстрировать этот интерфейс в прототипе почти сразу после начала проекта.
3. Требуется выполнение проекта в сжатые сроки. Быстрое выполнение проекта позволяет создать систему, отвечающую требованиям сегодняшнего дня. Если система проектируется долго, то высока вероятность того, что за это время существенно изменятся условия деятельности организации, т. е. система морально устареет еще до завершения ее проектирования.
4. Проект выполняется в условиях ограниченности бюджета. Разработка ведется небольшими RAD-группами в короткие сроки, что

обеспечивает минимум трудозатрат и позволяет вписаться в бюджетные ограничения.

5. ПО не обладает большой вычислительной сложностью.

RAD применима для систем средней сложности, обладающих элементами новизны. Если проектируемая система велика, то она должна допускать разбиение на более мелкие функциональные компоненты. Они могут выпускаться последовательно или параллельно.

К основным приемам RAD относятся следующие.

1. Использование прототипирования, позволяющего полнее выяснить потребности пользователей.
2. Вовлечение пользователей в процесс разработки системы.
3. Разработка приложений итерациями, многократное возвращение к более ранним этапам ЖЦ.
4. Необязательность полного завершения работ на одном этапе жизненного цикла для начала работ на следующем этапе. При итеративном подходе пропущенные работы можно выполнить впоследствии. Переход к следующему этапу ЖЦ осуществляется в соответствии с планом, даже если не вся запланированная работа закончена. План составляется на основе статистических данных, полученных в предыдущих проектах.
5. Высокая степень параллельности работ.
6. Повторное использование частей проекта.
7. Применение CASE-средств (CASE – Computer Aided System Engineering), обеспечивающих техническую целостность проекта на всех этапах проектирования, в том числе использование генераторов (мастеров).
8. Применение средств управления конфигурациями, облегчающее внесение изменений в проект и сопровождение готовой системы.

Как уже отмечалось, технология RAD является примером использования спиральной модели жизненного цикла ИС. Жизненный цикл ИС состоит из многократно повторяемых четырех стадий:

1. Анализ требований и планирование.
2. Проектирование.
3. Реализация.
4. Внедрение версии.

Работа над проектом ведется группами. Типичный состав группы – руководитель, аналитик, два-три программиста, технический писатель. Если проект сложный, то для него может быть выделено несколько RAD-групп. При этом системы разбиваются на подсистемы, и каждая подсистема разрабатывается независимой группой. Проект выполняется в условиях тесного взаимодействия между разработчиками и заказчиком.

RAD-группа всегда работает только над одним прототипом. Это обеспечивает единство целей, лучшую наблюдаемость и управляемость процессом разработки, что в итоге повышает качество конечного продукта. Используемые инструментальные средства должны обеспечивать групповую разработку и конфигурационное управление проектом. Ключевой фактор успеха здесь – правильное разбиение системы на подсистемы. Все группы должны использовать общие стандарты проектирования. Обязательно проводится финальное тестирование всей системы.

Создание прототипов ИС делает требования более реальными, приближает варианты использования к жизни и закрывает пробелы в понимании требований. Прототипы предоставляют пользователям экспериментальную модель новой системы, стимулируя их мышление и активизируя обсуждение требований. Обсуждение прототипов на ранних стадиях процесса разработки помогает заинтересованным в проекте лицам прийти к общему пониманию требований к системе, что уменьшает риск неудачи проекта.

Традиционно для проектов среднего уровня сложности разрабатываются три прототипа. Первый содержит весь пользовательский интерфейс с нулевой функциональностью. Он дает возможность собрать замечания заказчика и после их устранения утвердить экранные формы и документы. Второй прототип содержит реализованную на 70–80 % функциональность системы, третий – полностью реализованную функциональность.

RAD-технология не является универсальной, ее применение целесообразно не всегда. Например, в проектах, где требования к программному продукту четко определены и не должны меняться, вовлечение заказчика в процесс разработки не требуется, и более эффективным может быть каскадный метод. То же касается проектов, сложность которых определяется необходимостью реализации сложных алгоритмов, причем роль и объем пользовательского интерфейса невелики.

3.2. Виды прототипов

Прототип ПО – это частичная реализация предлагаемого нового продукта. Прототипы позволяют решать следующие три основные задачи.

1. *Прояснение и завершение процесса формулировки требований***.** Используемый в качестве инструмента формулировки требований прототип представляет собой предварительную версию той части системы, понимание которой вызывает затруднения. Оценка прототипа пользователями указывает на ошибки в формулировке требований, которые можно исправить без больших затрат еще до создания реального продукта.
2. *Исследование альтернативных решений.* Прототип как инструмент конструирования позволяет заинтересованным в проекте лицам исследовать различные варианты реализации функций системы, оптимизировать удобство работы и оценить возможные технические приемы, в том числе влияющие на быстродействие ИС. Прототипы позволяют на рабочих образцах показать, насколько осуществимы требования.

3. *Создание конечного продукта.* Прототип, используемый в качестве инструмента разработки, позволяет превратить его в готовый продукт, осуществляя последовательную цепочку коротких циклов разработки (итераций).

Основная цель создания прототипа – устранение неясностей на ранних стадиях процесса разработки. Исходя из этого следует решить, для каких частей системы необходим прототип и что можно выяснить, создав его. Прототип полезен для выявления и устранения неясных и неполных утверждений в требованиях.

Различают следующие разновидности прототипов:

- горизонтальные;
- вертикальные;
- одноразовые;
- эволюционные.

Горизонтальные прототипы

Когда говорят о «прототипе ПО», обычно имеют в виду *горизонтальный прототип* (horizontal prototype) предполагаемого интерфейса пользователя. Его также именуют *поведенческим прототипом* (behavioral prototype). Горизонтальным прототип называется потому, что в нем не реализуются все возможности системы, но воплощаются особенности интерфейса пользователя. Он позволяет исследовать поведение предполагаемой системы в тех или иных ситуациях для уточнения требований, а также выяснить, смогут ли пользователи с помощью системы, основанной на прототипе, выполнять свою работу.

Горизонтальный прототип создает видимость функциональности, не обеспечивая ее в действительности. Он показывает вид экранов пользовательского интерфейса и позволяет осуществлять частичную навигацию между ними. Горизонтальные прототипы демонстрируют то, что будет доступно пользователю и в готовой системе: внешний вид пользовательского интерфейса и структуру доступа к информации (структуру навигации).

Перемещение между объектами интерфейса возможно, но вместо некоторых элементов пользователь увидит лишь краткое сообщение о том, что будет здесь находиться в окончательной версии. Информация, появляющаяся в ответ на запрос к базе данных, может быть случайной или постоянной, а содержание отчетов – жестко зафиксированным. В образцах отчетов, диаграмм и таблиц лучше использовать реальные данные – это увеличит достоверность прототипа как модели реальной системы.

Горизонтальный прототип почти не выполняет полезной работы, однако зачастую его вполне достаточно, чтобы пользователи могли решить, нет ли каких-либо упущений, неверных или ненужных функций. Оценивая прототип, пользователи могут указать на альтернативные возможности реализации функций, пропущенные шаги взаимодействия или дополнительные условия.

Вертикальные прототипы

Вертикальный прототип (vertical prototype), также называемый *структурным прототипом* (structural prototype) или *проверкой концепции,* воплощает срез функциональности приложения от интерфейса пользователя до сервисных функций. Вертикальный прототип действует как настоящая система, поскольку затрагивает все уровни ее реализации. Вертикальный прототип разрабатывается в случае, когда есть сомнения в осуществимости предполагаемого подхода к архитектуре системы или когда необходимо оптимизировать алгоритмы, оценить предлагаемую схему базы данных или проверить критически важные временные требования. Чтобы получить значимые результаты, вертикальные прототипы следует создавать в той же среде разработки, что и окончательную версию системы. Вертикальные прототипы используются также и для сокращения рисков при проектировании системы.

Одноразовые прототипы

Прежде чем создавать прототип, нужно принять решение, будет ли он использоваться и далее, постепенно превращаясь в конечный продукт, либо нужда в нем закончится, как только будет решен какой-либо конкретный

вопрос. *Одноразовый прототип* (throwaway prototype), или *исследовательский прототип* (exploratory prototype) создается, чтобы изучить проблему, разрешить неясности и улучшить требования к ПО. Если в дальнейшем он использоваться не будет, его создание должно быть как можно более быстрым и дешевым.

Создавая одноразовый прототип, разработчики пренебрегают большинством методов конструирования качественного ПО. В одноразовом прототипе предпочтение отдается быстроте реализации, а не качеству. Нужно следить, чтобы низкокачественный код одноразового прототипа не попал в окончательный продукт. В противном случае пользователи и персонал технического обслуживания будут страдать от последствий этого в течение всего срока эксплуатации продукта.

Эволюционные прототипы

В отличие от одноразового прототипа *эволюционный прототип* (evolutionary prototype) представляет собой «фундамент» для постепенного создания окончательного продукта – по мере прояснения требований. Эволюционное прототипирование – один из компонентов модели спирального цикла разработки ПО и некоторых процессов разработки объектно-ориентированного ПО. И если одноразовые прототипы создаваются быстро и без внимания к качеству, то для построения эволюционного прототипа необходимо с самого начала использовать качественный код.

Поэтому на конструирование эволюционного прототипа требуется больше времени, чем на одноразовый прототип, моделирующий те же свойства системы. Эволюционный прототип следует создавать в расчете на его рост и частое расширение, поэтому разработчики должны уделять большое внимание архитектуре и принципам проектирования. При создании такого прототипа не стоит экономить на качестве.

4. МЕТОДОЛОГИЯ MSF

В 1994 году компания Microsoft выпустила пакет руководств MSF (Microsoft Solutions Framework) по эффективному проектированию, разработке, внедрению и сопровождению решений, построенных на основе своих технологий. Целью было достижение максимальной отдачи от IT-проектов компании. Руководства отражают опыт, полученный Microsoft при работе над проектами по разработке и сопровождению программного обеспечения. Вторая версия методологии (MSF 2.0) датируется 1998 годом, а версия MSF 3.0 была представлена в 2001 году. Текущая версия – MSF 4.0 была представлена в 2005 году. В этой версии произошло разделение методологии на два направления: MSF for Agile Software Development и MSF for CMMI Process Improvement. Дальнейшее изложение касается первого из этих направлений [3].

В MSF 4.0 впервые появилась инструментальная поддержка – среда разработки Microsoft Visual Studio 2005 Team System. Она может выступать теперь в качестве интегрирующего средства, при помощи которого можно работать со всеми инструментами, обеспечивающими все стадии процесса разработки: от планирования проекта до проведения тестирования, включая создание и выполнение тестовых сценариев.

Текущая версия MSF for Agile Software Development включает две модели и три дисциплины:

Модели

1. Модель процессов.
2. Модель проектной группы.

Дисциплины

1. Дисциплина «управление проектами».
2. Дисциплина «управление рисками».
3. Дисциплина «управление подготовкой».

4.1. Модель процессов MSF

Модель процессов MSF (MSF process model) описывает общие подходы к разработке и внедрению ПО. Благодаря своей гибкости она может быть

применена при разработке весьма широкого круга проектов. Эта модель сочетает в себе свойства двух стандартных моделей: каскадной и спиральной. Модель процессов MSF охватывает весь жизненный цикл создания решения, начиная с его отправной точки и заканчивая внедрением.

Модель процессов MSF опирается на следующие базовые принципы:

- подход, основанный на фазах и вехах;
- итеративный подход;
- интегрированный подход к созданию и внедрению решений.

Модель процессов включает такие основные фазы процесса разработки как:

- выработка концепции;
- планирование;
- разработка;
- стабилизация;
- внедрение.

Процесс MSF ориентирован на «вехи» (milestones) – ключевые точки проекта, характеризующие достижение в его рамках какого-либо существенного (промежуточного либо конечного) результата, причем этот результат может быть оценен и проанализирован. Критерии оценки результата должны быть сформулированы еще до начала проекта.

Для декомпозиции больших этапов работы может быть введено также большое количество промежуточных вех.

Модель процессов MSF учитывает постоянные изменения проектных требований. Она исходит из того, что разработка решения должна состоять из коротких циклов, реализующих поступательное движение от начальных версий системы к ее окончательному виду.

В рамках MSF программный код, документация, дизайн, планы и другие рабочие материалы создаются, как правило, итеративными методами. MSF рекомендует начинать разработку решения с построения, тестирования и внедрения его базовой функциональности. Затем к решению добавляются все новые и новые возможности, т. е. с каждой новой версией эволюционирует

функциональность решения. Для малых проектов может быть достаточным выпуск одной версии.

Итеративный подход к процессу разработки требует использования гибкого способа ведения документации. Документация должна изменяться по мере реализации проекта вместе с изменениями требований к конечному продукту. В рамках MSF предлагается ряд шаблонов стандартных документов для каждой стадии разработки продукта, которые могут быть использованы для планирования и контроля процесса разработки.

Решение не представляет ценности, пока оно не внедрено. Именно по этой причине модель процессов MSF содержит весь жизненный цикл создания решения, включая его внедрение, вплоть до момента, когда решение начинает давать отдачу.

4.2. Модель проектной группы MSF

Модель проектной группы MSF (MSF Team Model) описывает подход Microsoft к организации работающего над проектом персонала и его деятельности в целях обеспечения успешности проекта. В методологии MSF разработки ПО все лица, участвующие в производстве, использовании и сопровождении продукта, обладают равными полномочиями. Участники команды имеют разные роли, связанные с их функциями, при этом ни одна из ролей не считается важнее другой.

Члены команды могут выступать в одной или нескольких ролях. Данная модель определяет *ролевые кластеры*, области их компетенции и зоны ответственности, а также рекомендации членам проектной группы, позволяющие им успешно осуществлять свои функции в рамках проекта.

В соответствии с моделью MSF проектные группы строятся как небольшие универсальные команды, члены которых распределяют между собой ответственность и дополняют области компетенций друг друга. Это дает возможность четко сфокусировать внимание на нуждах проекта. Проектную группу связывает единое видение проекта, стремление к воплощению его в жизнь, высокие требования к качеству работы и желание

самосовершенствоваться. Ниже описываются основные принципы, ключевые идеи и испытанные методики MSF в применении к модели проектной группы.

Успешное использование модели проектной группы MSF основывается на ряде ключевых концепций:

- сотрудничество внутри команды;
- сфокусированность на нуждах заказчика;
- нацеленность на конечный результат;
- установка на отсутствие дефектов;
- стремление к самосовершенствованию;
- заинтересованность команды как фактор эффективной работы.

В проектную группу входят такие ролевые кластеры:

- бизнес-аналитик;
- менеджер проекта;
- архитектор;
- разработчик;
- тестировщик;
- релиз-менеджер;
- администратор баз данных;
- разработчик баз данных.

Они ответственны за различные области компетенции и связанные с ними цели и задачи. Иногда ролевые кластеры называются просто ролями.

Основная задача *бизнес-аналитика* – разобраться в возможностях системы, относящихся к бизнесу, и раскрыть их команде. Он работает с пользователями и другими заинтересованными лицами, чтобы понимать их потребности и задачи, трансформировать их в конкретные определения, сценарии и требования к качеству, которые команда разработчиков будет использовать для построения приложения. Кроме того, бизнес-аналитик определяет ожидания от функциональных возможностей системы и управляет ими. В проекте он представляет пользователей и участвует в управлении продуктом в том смысле, что постоянно отслеживает интересы пользователей и

заказчиков проекта. Бизнес-аналитики отвечают и за обеспечение взаимодействия между разработчиками и пользователями.

Основная задача *менеджера проекта* – добиваться выполнения поставленных перед командой задач в соответствии с графиком и в рамках бюджета. На менеджере проекта лежат обязательства по планированию и составлению графика работ, включающие разработку проекта и планов итераций, отслеживание состояния дел и составление отчетов, а также определению рисков и выработке мер по их уменьшению. Менеджер проекта также проводит консультации с бизнес-аналитиками по планированию сценариев и выработке требований к качеству для каждой итерации, консультируется с архитекторами и разработчиками для оценки объемов работ, советуется с тестировщиками, чтобы спланировать тестирование.

Архитектор отвечает за архитектуру проекта. Его основная задача – обеспечить успех проекта путем разработки основных принципов приложения, которые включают в себя как организационную конфигурацию системы, так и ее физическую структуру. При этом архитектор должен стремиться к снижению сложности путем разделения системы на понятные и простые части. Архитектура приложения чрезвычайно важна, поскольку она не просто устанавливает этапы построения системы, а определяет, будет ли приложение обладать свойствами, присущими успешным проектам. К ним относятся: удобство использования, надежность, практичность сопровождения, производительность и безопасность, а также возможности модификации в случае изменения требований.

В рамках командной модели MSF *разработчик* выполняет разработку приложения. Его основная задача – реализовать приложение согласно спецификациям и в установленные сроки. Разработчик также помогает уточнять физический дизайн, оценивать время и усилия для выполнения конкретных элементов, выполняет реализацию функций или руководит ею, подготавливает продукт к внедрению и является экспертом команды в технологических областях.

В командной модели MSF *тестировщик* выполняет задачи тестирования продукта. Основной задачей тестировщика является выявление проблем в продукте, которые могут неблагоприятно повлиять на его качество. Тестировщик обязан понимать контекст проекта и помогать остальным членам команды понимать решения, основанные на этом контексте. Ключевая цель тестировщика – поиск серьезных дефектов в продукте путем его тестирования и последующее их описание. Каждый найденный дефект тестировщик должен сопроводить точным описанием вредного воздействия и предложить способ обойти дефект, чтобы уменьшить это воздействие. Он должен как можно проще описать дефект и последовательность, в которой его можно воспроизвести.

Релиз-менеджер отвечает за операции по выпуску продукта. Основная цель релиз-менеджера – обеспечение выпуска готового продукта. Он координирует выпуск продукта со специалистами по эксплуатации, создает план выпуска продукта и сертифицирует подготовленные к выпуску версии для поставки или развертывания.

Основная задача *администратора баз данных* в контексте разработки базы данных – поддержка создания проектов баз данных, а также внесение изменений проекта в рабочую базу данных. В дополнение к этому он должен выполнять традиционные задачи, такие как ежедневное администрирование и поддержка серверов баз данных.

Основная задача разработчика баз данных – выполнение комплекса задач по разработке базы данных в установленные сроки. Кроме того, он отвечает за оценку стоимости, контроль над реализацией функций и помощь остальным участникам команды по вопросам, связанным с базами данных. Разработчик баз данных совместно с администратором баз данных и прикладными разработчиками участвует в итеративном жизненном цикле разработки базы данных.

Наличие ролевых кластеров не означает, что количество членов команды должно быть кратным их количеству. Один человек может совмещать несколько ролей, а ролевой кластер может состоять из нескольких лиц в

зависимости от размера проекта, его сложности и профессиональных навыков, требуемых для реализации всех областей компетенции кластера. Минимальный коллектив по MSF может состоять всего из трех человек. Модель не требует назначения отдельного сотрудника на каждый ролевой кластер. Обычно выделение как минимум одного человека на каждый ролевой кластер обеспечивает полноценное внимание к интересам каждой из ролей, но это экономически оправданно не для всех проектов. Зачастую члены проектной группы могут объединять роли.

В малых проектных группах объединение ролей является необходимым. При этом должны соблюдаться два принципа:

1. Роль команды разработчиков не может быть объединена ни с какой другой ролью.
2. Избегание сочетания ролей, имеющих предопределенные конфликты интересов.

Как и в любой другой командной деятельности, подходящая комбинация ролей зависит от самих членов команды, их опыта и профессиональных навыков. На практике совмещение ролей встречается нередко, и если оно проводится обдуманно, возникающие проблемы будут минимальными.

MSF не предоставляет конкретных рецептов управления проектами и не содержит объяснений разнообразных методов работы, которые применяют опытные менеджеры. Принципы MSF формируют такой подход к управлению проектами, при котором:

- ответственность за управление проектом распределенная между лидерами ролевых кластеров внутри команды – каждый член проектной группы отвечает за общий успех проекта и качество создаваемого продукта;
- профессиональные менеджеры выступают в качестве консультантов и наставников команды, а не выполняют функции контроля над ней – в эффективно работающей команде каждый ее член имеет необходимые полномочия для выполнения своих обязанностей.

Модель проектной группы MSF предлагает разбиение больших команд (более 10 человек) на малые группы направлений. Эти малые коллективы работают параллельно, регулярно синхронизируя свои усилия. Кроме того, когда ролевому кластеру требуется много ресурсов, формируются так называемые функциональные группы, которые затем объединяются в ролевые кластеры.

Использование ролевых кластеров не подразумевает и не навязывает никакой специальной структуры организации или обязательных должностей. Административный состав ролей может широко варьироваться в разных организациях и проектных группах. Ключевым моментом является четкое определение работников, ответственных за каждый ролевой кластер, их функций, ответственности и ожидаемого вклада в конечный результат.

Модель проектной группы MSF не обеспечивает успех сама по себе. Есть много других факторов, определяющих успех или неудачу проекта, но структура проектной группы, безусловно, вносит существенный вклад.

Подходящая структура команды является фундаментом успеха, и реализация модели MSF с использованием лежащих в ее основе принципов поможет сделать проектные группы более эффективными и, как следствие, более успешными.

4.3. Дисциплина «Управление проектами»

Проект (project) – ограниченная временными рамками деятельность, цель которой состоит в создании уникального продукта или услуги. Управление проектами (project management) – это область знаний, навыков, инструментария и приемов, используемых для достижения целей проектов в рамках согласованных параметров качества, бюджета, сроков и прочих ограничений.

Хорошо известна взаимозависимость между ресурсами проекта (людскими и финансовыми), его календарным графиком и реализуемыми возможностями. Эти три переменные образуют так называемый «треугольник компромиссов». Нахождение верного баланса между ресурсами, временем разработки и возможностями – ключевой момент в построении решения, должным образом отвечающего нуждам заказчика.

Другое весьма полезное средство для управления проектными компромиссами – матрица компромиссов проекта (project tradeoff matrix). Она отражает достигнутое на ранних этапах проекта соглашение между проектной группой и заказчиком о выборе приоритетов в возможных в будущем компромиссных решениях. В определенных случаях могут делаться исключения, но в целом следование матрице компромиссов облегчает достижение соглашений по спорным вопросам.

Для лидеров групп и ролевого кластера «Управление программой» инструментом управления проектом, облегчающим создание планов и календарных графиков, является иерархическая структура работ (WBS, Work Breakdown Structure). WBS – это структуризация работ проекта, отражающая его основные результаты и определяющая его рамки. Работа, не описанная в WBS, находится вне границ проекта. В MSF создание WBS является коллективной деятельностью, в которую вовлекаются все ролевые кластеры. Каждая роль ответственна за предоставление детального описания собственной работы.

4.4. Дисциплина «Управление рисками»

Управление рисками (Risk management) – это одна из ключевых дисциплин MSF. Риск – это возможная потеря, в том числе падение качества продукта, рост затрат на разработку, отставание от графика или неудача в достижении целей проекта. Управление рисками принимает риски как неотъемлемую часть жизненного цикла информационных технологий. Борются с рисками либо превентивно, либо по факту их проявления. Управление рисками в MSF подразумевает превентивные меры по предотвращению рисков на всех этапах разработки. Данная дисциплина предлагает принципы, идеи и рекомендации, подкрепленные описанием пошагового процесса для успешного активного управления рисками.

Этот процесс включает в себя:

- выявление риска;
- определение степени его влияния на проект;

– определение вероятности его возникновения;
– понимание того, как риск может проявиться в проекте;
– принятие превентивных мер по его предотвращению;
– выработка плана на случай реализации риска.

Использование этой модели помогает проектной группе принимать верные решения и лучше подготовиться к возможному возникновению проблем. Если применять управление рисками с самого начала работы над проектом, вероятность их осуществления на поздних стадиях значительно сократится.

4.5. Дисциплина «Управление подготовкой»

Управление подготовкой – также одна из ключевых дисциплин MSF. Она посвящена управлению знаниями, профессиональными умениями и способностями, необходимыми для планирования, создания и сопровождения успешных решений. Готовность к обучению включает в себя постоянное самосовершенствование участника команды путем накопления знаний и обмена ими с другими. В графике проекта предусматривается время для обучения членов команды, анализа текущего состояния дел и проделанной работы. Кроме того, важным требованием к каждому участнику команды должно быть стремление к обмену знаниями с другими участниками.

Дисциплина управления подготовкой MSF описывает фундаментальные принципы MSF и дает рекомендации по применению превентивного подхода к управлению знаниями на протяжении всего жизненного цикла информационных технологий. Эта дисциплина также рассматривает планирование процесса управления подготовкой. Будучи подкрепленной испытанными на практике методиками, дисциплина управления подготовкой предоставляет проектным группам и отдельным специалистам базу для осуществления этого процесса.

5. AGILE-МЕТОДИКИ

Agile-методики (англ. *Agile software development* – гибкая методология разработки) – семейство подходов к разработке программного обеспечения, ориентированных на использование итеративной разработки, динамическое формирование требований и обеспечение их реализации в результате постоянного взаимодействия внутри самоорганизующихся рабочих групп, состоящих из специалистов различного профиля [4]. Содержание Agile определяется документом, известным как Agile Manifesto. Документ был разработан и принят в 2001 году.

Существует несколько методик, относящихся к классу гибких методологий разработки, в частности, экстремальное программирование (см. раздел 6), DSDM (см. раздел 8).

Большинство гибких методологий нацелены на минимизацию рисков проекта путем использования серии коротких циклов разработки, называемых итерациями. Каждая итерация обычно длится две-три недели и выглядит как программный проект в миниатюре, включая все этапы жизненного цикла системы:

- планирование;
- анализ требований;
- проектирование;
- программирование;
- тестирование;
- документирование.

Хотя отдельная итерация, как правило, недостаточна для выпуска новой версии продукта, подразумевается, что гибкий программный проект готов к работе в конце каждой итерации. По окончании каждой итерации команда выполняет переоценку приоритетов разработки.

Принципы Agile

Agile не включает практических рекомендаций, а определяет ценности и принципы, которыми руководствуются успешные команды.

Agile Manifesto содержит четыре основные идеи и двенадцать принципов.

Основные идеи Agile Manifesto:

1. Люди и взаимодействие важнее процессов и инструментов.
2. Работающий продукт важнее исчерпывающей документации.
3. Сотрудничество с заказчиком важнее согласования условий контракта.
4. Готовность к изменениям важнее следования первоначальному плану.

Принципы, содержащиеся в Agile Manifesto:

1. Наивысшим приоритетом является удовлетворение потребностей заказчика благодаря регулярной и ранней поставке ценного программного обеспечения.
2. Изменение требований приветствуется даже на поздних стадиях разработки. Agile-процессы позволяют использовать изменения для обеспечения заказчику конкурентного преимущества.
3. Работающий продукт следует выпускать как можно чаще, с периодичностью от пары недель до пары месяцев.
4. На протяжении всего проекта разработчики и представители бизнеса должны ежедневно работать вместе.
5. Над проектом должны работать мотивированные профессионалы. Чтобы работа была сделана, создайте условия, обеспечьте поддержку и полностью доверьтесь им.
6. Непосредственное общение является наиболее практичным и эффективным способом обмена информацией как с самой командой, так и внутри команды.
7. Работающий продукт – основной показатель прогресса.
8. Инвесторы, разработчики и пользователи должны иметь возможность поддерживать постоянный ритм бесконечно. Agile помогает наладить такой устойчивый процесс разработки.
9. Постоянное внимание к техническому совершенству и качеству проектирования повышает гибкость проекта.

10. Простота – искусство минимизации лишней работы – крайне необходима.
11. Лучшие требования, архитектурные и технические решения рождаются у самоорганизующихся команд.
12. Команда должна систематически анализировать возможные способы улучшения эффективности и соответственно корректировать стиль своей работы.

Оценка методики

Agile-методы делают упор на непосредственное общение внутри команды-разработчика. Команда включает «заказчика», определяющего требования к системе (эту роль может выполнять менеджер проекта, бизнес-аналитик или клиент), проектировщиков, программистов, тестировщиков, технических писателей и менеджеров. Отдавая предпочтение непосредственному общению, agile-методы уменьшают объем письменной документации по сравнению с другими методами.

Гибкий подход к управлению требованиями не подразумевает долгосрочных планов (по сути, управления требованиями просто не существует в данной методологии), но предусматривает возможность для заказчика в конце каждой итерации выдвигать новые требования, часто без оглядки на созданный продукт.

Считается, что работа в Agile мотивирует разработчиков решать все проектные задачи простейшим из возможных способом, при этом зачастую не обращая внимания на правильность кода с точки зрения требований используемой программной платформы.

6. ТЕХНОЛОГИЯ XP

Технология XP (англ. Extreme Programming), как и RAD, базируется на спиральной модели жизненного цикла [5]. Авторами технологии являются К. Бек, У. Каннингем, М. Фаулер. Название технологии связано со стремлением авторов поднять существующие методы разработки ИС (и программного обеспечения в целом) на новый, «экстремальный» уровень.

Для оценки проектов с точки зрения применимости XP применяются два показателя – критичность и масштаб. Критичность определяется последствиями, вызываемыми дефектами ПО, и может иметь один из четырех уровней:

1. C – дефекты вызывают потерю удобства.
2. D – дефекты вызывают потерю возместимых ресурсов.
3. E – дефекты вызывают потерю невозместимых ресурсов.
4. L – дефекты могут создавать угрозу для человеческой жизни.

Масштаб определяется количеством разработчиков, участвующих в проекте:

- от 1 до 6 человек – малый масштаб;
- от 6 до 20 человек – средний масштаб;
- свыше 20 человек – большой масштаб.

По оценке специалиста по разработке ПО А. Коберна, XP применима в проектах малого и среднего масштаба с низкой критичностью (C или D).

Разработка в XP ведется небольшими трехнедельными итерациями, в течение которых уточняются и реализуются требования к системе. При разработке применяется *рефакторинг* – методика улучшения кода без изменения его функциональности. Программный код в процессе работы над проектом неоднократно переделывается, в том числе и на поздних стадиях проекта.

Для того чтобы эти переделки не привели к неработоспособности системы, используется методика TDD (*Test-Driven Development* – разработка через тестирование). Технология XP предполагает написание автоматических

тестов – специальных программ, написанных для тестирования других программ. Ручная прогонка тестов здесь невозможна, т. к. количество тестов слишком велико. Тесты пишутся еще до того, как начинается создание системы, поэтому риск получения неработоспособной версии из-за постоянно вносимых изменений существенно снижается – любую новую версию тут же можно протестировать.

В процессе создания системы применяются такие характерные для XP методы, как парное программирование, непрерывная интеграция, упрощенное проектирование.

Парное программирование предполагает, что программы создается парами программистов, работающих за одним компьютером. Один из них пишет непосредственно текст программы, другой оценивает его работу, благодаря чему становится возможной постоянная проверка программного кода. В течение работы над проектом пары не фиксируются: это делается с той целью, чтобы каждый программист в команде имел хорошее представление обо всей системе. Повышение эффективности при работе парой программистов подтверждено специальными исследованиями.

Непрерывная интеграция (сборка) системы позволяет поддерживать ее целостность в течение всего процесса разработки. В традиционных методиках интеграция выполняется в самом конце работы над продуктом, когда все составные части разрабатываемой системы полностью готовы. Интеграционные проблемы обладают способностью накапливаться и наслаиваться друг на друга, что может даже привести к провалу проекта. В XP интеграция системы выполняется несколько раз в день, после того, как все модули прошли положенные для них тесты. Это позволяет выявить проблемы интеграции на возможно более ранней стадии разработки и заблаговременно принять необходимые шаги к их преодолению.

Упрощенное проектирование применяется в XP из-за того, что в процессе работы требования к системе могут неоднократно меняться, что снижает ценность проекта, выполненного целиком в самом начале разработки. Для XP

характерно непрерывное проектирование, выполняемое в течение всего времени работы над проектом. Проектирование должно выполняться небольшими этапами, с учетом постоянно изменяющихся требований. В каждый момент времени следует использовать наиболее простые решения, которые подходят для решения текущей задачи, и менять его по мере того, как условия задачи меняются. Согласно К. Беку, упрощенное проектирование обеспечивает корректное выполнение всех тестов, не порождает дублирующего кода, включает наименьшее количество классов и методов, ясно выражает цель программиста.

Помимо перечисленного, все члены команды в ходе работы должны соблюдать *общие требования стандартов программирования*, что существенно облегчает рефакторинг и снижает риски проекта, связанные с текучкой кадров. В идеале соблюдение стандартов программирования должно полностью исключить индивидуальные черты стиля разработки – программный продукт должен выглядеть как результат работы одного человека.

Коллективное владение означает, что каждый член команды несет ответственность за весь исходный код. Каждый вправе вносить изменения в любой участок программы. Сопутствующие риски от вносимых изменений устраняются мощной системой тестирования. Однако это не порождает безответственности, поскольку существует требование, согласно которому каждый программист должен сам исправить сделанные им ошибки. Важное преимущество коллективного владения кодом состоит в том, что оно ускоряет процесс разработки, поскольку при необходимости любой программист может оперативно внести изменения в любую часть кода.

Существенным считается и наличие *метафоры системы* – простой аналогии, понятной всем участникам проекта, которая с достаточной точностью описывает функционирование и внутреннюю структуру ИС.

7. МЕТОДОЛОГИЯ RUP

Rational Unified Process (RUP) – методология разработки программного обеспечения [6], созданная компанией Rational Software, с 2003 года входящей в корпорацию IBM. Методология RUP основана на спиральной модели жизненного цикла ИС. В качестве языка моделирования в RUP используется язык Unified Modelling Language (UML).

Наибольшее внимание RUP уделяет начальным стадиям разработки проекта – анализу и моделированию, – что призвано снизить риски проекта за счет раннего обнаружения возможных ошибок. Последовательный выпуск версий организован таким образом, чтобы наиболее существенные риски устранялись в первую очередь.

Методология RUP широко использует так называемые *прецеденты*, или сценарии использования – описание последовательностей действий, которые может осуществлять система, взаимодействуя с внешними действующими факторами. Прецеденты создаются при помощи UML и включают варианты как правильных, так и ошибочных последовательностей (исключений). Прецеденты служат для документирования требований заказчика к проектируемой информационной системе. Прецедент описывает целостный фрагмент поведения системы в виде последовательности сообщений, которыми система обменивается с действующими лицами.

Другие существенные черты методологии RUP:

- заранее предусматриваются изменения в требованиях и проектных решениях в течение всего процесса разработки,
- постоянное обеспечение качества на всех этапах разработки ИС,
- компонентная архитектура используется начиная с ранних стадий проекта.

Процессы и стадии RUP

RUP использует итеративную модель разработки. В конце каждой итерации (продолжительностью в несколько недель) команда разработчиков должна получить функционирующую версию конечного продукта,

позволяющую достичь запланированных на данную итерацию целей. Итеративная разработка позволяет быстро реагировать на меняющиеся требования, обнаруживать и устранять риски на ранних стадиях проекта, а также эффективно контролировать качество создаваемого продукта.

Полный жизненный цикл разработки ИС состоит из четырех фаз, описываемых ниже. Каждая фаза может включать в себя одну или несколько итераций процесса создания системы.

Начальная стадия

В фазе начальной стадии:

- формируется единая точка зрения на проект у заказчиков и разработчиков, а также определяются границы проекта,
- создается экономическое обоснование разработки,
- определяются основные требования, ограничения и функциональность системы,
- создается базовая версия модели прецедентов,
- оцениваются риски.

Уточнение

В фазе «Уточнение» производится анализ исходных данных для проектирования и выбор архитектуры ИС. Фаза включает в себя:

- анализ требований заказчика, включая детальное описание для большинства прецедентов,
- проектирование архитектуры ИС, спецификация функций и пользовательского интерфейса,
- планирование работ по проекту и всех необходимых ресурсов.

Построение

В фазе «Построение» происходит итеративная реализация требуемых функций ИС. Это основная фаза проектирования и создания программного кода. Фаза завершается выпуском бета-версии системы.

Внедрение

В фазе «Внедрение» финальная версия системы внедряется у ее заказчика. Фаза включает проведение испытаний системы, обучение пользователей, а также оценку качества ИС. В том случае, если качество системы не соответствует требованиям заказчика, фаза «Внедрение» выполняется повторно. Выполнение всех требований и достижение целей проекта означает завершение полного цикла разработки.

Приведем список наиболее распространенных программных продуктов, поддерживающих RUP.

Rational Rose – CASE-средство визуального моделирования информационных систем, имеющее возможности генерирования элементов кода. Специальная редакция продукта – Rational Rose RealTime – позволяет на выходе получить исполняемый модуль.

Rational Requisite Pro – средство управления требованиями, позволяющее создавать, структурировать, устанавливать приоритеты, контролировать изменения требований, возникающие на любом этапе разработки компонентов приложения.

Rational ClearQuest – продукт для управления изменениями и отслеживания дефектов в проекте, тесно интегрирующийся со средствами тестирования и управления требованиями и представляющий собой единую среду для связывания всех ошибок и документов между собой.

Rational SoDA – продукт для автоматического генерирования проектной документации, позволяющий установить корпоративный стандарт на внутрифирменные документы.

8. МЕТОД DSDM

Метод разработки динамических систем (англ. Dynamic Systems Development Method, DSDM) основан на концепции быстрой разработки приложений (RAD) [7]. Метод DSDM – это итеративный и инкрементный подход разработки программного обеспечения, который придает особое значение продолжительному участию в процессе заказчика системы.

Метод DSDM был разработан в Великобритании в 1990-х Консорциумом DSDM. Консорциум DSDM – это ассоциация разработчиков и экспертов в области программного обеспечения, созданная с целью использования лучшего практического опыта участников ассоциации. Все, кто распространяет DSDM, должны быть членами этого некоммерческого консорциума.

Последняя версия DSDM называется DSDM Atern. Предыдущая версия DSDM 4.2, выпущенная в мае 2003 года, все еще действует. Расширенная версия содержит руководство по тому, как использовать DSDM совместно с XP (eXtreme Programming).

Цель метода DSDM – соблюдение сроков и бюджета проекта при допущении изменений в требованиях к системе во время ее разработки. Как представитель RAD-технологии DSDM фокусируется на проектах информационных систем, характеризующихся сжатыми сроками и бюджетами. DSDM входит в семейство гибкой методологии разработки программного обеспечения, а также может применяться для разработок, не входящих в сферу информационных технологий.

Существует возможность включения в DSDM частей других методик, таких как Rational Unified Process (RUP) или XP. Другой гибкий метод, похожий на DSDM по процессу и концепции, – Scrum.

DSDM содержит указания на типичные ошибки проектов информационных систем, такие как превышение бюджета, несоблюдение сроков сдачи, недостаточное вовлечение пользователей и менеджеров организации-заказчика в работу над проектом.

В DSDM существует девять принципов, четыре из которых относятся к основным.

1. Вовлечение пользователя – это основа ведения эффективного проекта, где разработчики делят с пользователями рабочее пространство и поэтому принимаемые решения будут более точными.
2. Команда должна быть уполномочена принимать важные для проекта решения без согласования с начальством.
3. Частая поставка версий результата с учетом такого правила, что «поставить что-то хорошее раньше – это всегда лучше, чем поставить все идеально сделанное в конце». Анализ поставок версий с предыдущей итерации учитывается на последующей.
4. Главный критерий – как можно более быстрая поставка программного обеспечения, которое удовлетворяет текущим потребностям рынка. Но в то же время поставка продукта, который удовлетворяет потребностям рынка, менее важна, чем решение критических проблем в функционале продукта.
5. Разработка – итеративная и инкрементная. Она основывается на обратной связи с пользователем, чтобы достичь оптимального с экономической точки зрения решения.
6. Любые изменения во время разработки – обратимы.
7. Требования устанавливаются на высоком уровне прежде, чем начнется проект.
8. Тестирование интегрировано в жизненный цикл разработки.
9. Взаимодействие и сотрудничество между всеми участниками необходимо для его эффективности.

Предпосылки для использования DSDM

Чтобы успешно использовать DSDM, необходимо, чтобы был выполнен ряд предпосылок. Во-первых, необходимо организовать взаимодействие между проектной командой, будущими пользователями и высшим руководством.

Во-вторых, должна присутствовать возможность разбиения проекта на меньшие части, что позволит использовать итеративный подход.

Можно привести примеры проектов, для которых использование DSDM не рекомендуется:

- проекты, критичные по безопасности (расширенное тестирование и утверждение в таких проектах конфликтуют с целью метода DSDM уложиться в сроки и в бюджет);
- проекты, чья цель – произвести компоненты многоразового использования (требования в таких проектах слишком высоки).

Жизненный цикл проекта

Согласно DSDM, жизненный цикл ИС состоит из трех последовательных стадий:

- предпроектной стадии;
- стадии проекта;
- постпроектной стадии.

Предпроектная стадия

На этой стадии определяются риски проекта, происходит выделение средств и определение проектной команды. Решение задач на этой стадии поможет избежать проблем на более поздних стадиях проекта.

Стадия проекта

Это самая детально разработанная стадия DSDM. Она состоит из пяти этапов, которые формируют итеративный, инкрементный подход к разработке информационных систем:

1. Исследование реализуемости.
2. Исследование экономической целесообразности.
3. Создание функциональной модели.
4. Проектирование и разработка.
5. Этап реализации.

Первые два этапа выполняются последовательно и дополняют друг друга. После их завершения происходит итеративная и инкрементная разработка

системы на основе этапов 3–5, выполняемых циклически, вплоть до выпуска готового продукта.

Постпроектная стадия

На этой стадии обеспечивается внедрение и эксплуатация системы. Это достигается за счет поддержания проекта, его улучшения и исправления ошибок согласно принципам DSDM. Поддержка проекта осуществляется как продолжение разработки, основанной на итеративной и инкрементной природе DSDM. Вместо того чтобы закончить проект за один цикл, обычно возвращаются к предыдущим стадиям или этапам, чтобы улучшить продукт.

Факторы, необходимые для успеха метода DSDM

В рамках DSDM существуют следующие факторы, которые влияют на успех проекта:

- принятие методики DSDM руководством проекта и всеми его участниками, что обеспечивает мотивацию членов команды с момента запуска проекта и до его окончания;
- готовность руководства обеспечить вовлеченность конечных пользователей в работу над проектом, включая тестирование и оценивание функциональных прототипов;
- проектная команда должна в итоге стать постоянной, что обеспечивает доверие и взаимопонимание внутри нее. Команда обладает правом и возможностью принимать важные решения о проекте без формального согласования с руководством, что могло бы отнять много времени;
- DSDM выступает за постоянные продуктивные отношения между разработчиком и заказчиком. Это касается как проектов, разрабатываемых внутри самих компаний, так и проектов с привлечением сторонних подрядчиков.

9. МЕТОДОЛОГИЯ SCRUM

Scrum (от англ. *scrum* – «толкучка») – методология управления проектами, активно применяющаяся при разработке информационных систем для гибкой разработки программного обеспечения [8]. Scrum делает акцент на качественном контроле процесса разработки. Подход впервые описали Хиротака Такэути и Икудзиро Нонака в 1986 году.

В основе управления процессами в Scrum заложены три главных принципа: *прозрачность, инспекция и адаптация*. Прозрачность означает, что все значимые аспекты процесса должны быть доступны тем, кто за него отвечает. Прозрачность требует, чтобы эти аспекты определялись общими стандартами, а все члены команды должны пользоваться общими понятиями и терминологией. Инспекции проводятся достаточно часто для своевременного выявления нежелательных отклонений от установленных целей. Однако инспектирование не должно быть настолько частым, чтобы мешать работе. Если по результатам инспекции выявлены отклонения, то необходимо как можно быстрее внести коррекцию в продукт во избежание распространения проблемы на другие части системы.

Методология Scrum – это набор принципов, на которых строится процесс разработки, позволяющий в жестко фиксированные и небольшие по времени итерации, называемые спринтами (sprints), предоставлять конечному пользователю работающее ПО с новыми возможностями, для которых определен наибольший приоритет. Возможности ПО к реализации в очередном спринте определяются в начале спринта на этапе планирования и не могут изменяться на всем его протяжении. При этом строго фиксированная небольшая длительность спринта придает процессу разработки предсказуемость и гибкость.

Спринт

Спринт – это итерация, в ходе которой создается прирост функциональности программного обеспечения. Спринт жестко фиксирован по времени. Длительность одного спринта составляет от 2 до 6 недель. Считается,

что чем короче спринт, тем более гибким является процесс разработки: версии выходят чаще, быстрее поступают отзывы от потребителя. С другой стороны, при более длительных спринтах команда имеет больше времени на решение возникших в процессе проблем, а владелец проекта уменьшает издержки на совещания, демонстрации продукта и др. Разные команды подбирают длину спринта согласно специфике своей работы, составу команд и требований часто методом проб и ошибок. Для оценки объема работ в спринте можно использовать предварительную оценку. Предварительная оценка фиксируется в журнале проекта. На протяжении спринта никто не имеет права менять список требований к работе, внесенных в журнал спринта.

Журнал продукта

Журнал продукта – это список требований к функциональности, упорядоченный по их степени важности, подлежащих реализации. Элементы этого списка называются «пожеланиями пользователя», или элементами. Журнал продукта открыт для редактирования для всех участников процесса.

Журнал спринта

Журнал спринта – это набор элементов Журнала продукта, выбранных для реализации в текущем спринте. Журнал спринта содержит прогноз возрастания функциональности продукта в данном спринте и план по ее достижению.

Диаграмма сгорания задач

Это диаграмма, показывающая количество сделанной и оставшейся работы. Обновляется ежедневно, с тем чтобы в простой форме показать подвижки в работе над спринтом.

Существуют разные виды диаграммы:

- диаграмма сгорания работ для спринта – показывает, сколько задач уже сделано и сколько еще остается сделать в текущем спринте;
- диаграмма сгорания работ для проекта – показывает, сколько задач уже сделано и сколько еще остается сделать до выпуска продукта.

Основные роли в методологии Scrum

Проект выполняется группой, в которую входят специалисты, выполняющие следующие роли:

- скрам-мастер (Scrum Master) – проводит совещания, следит за соблюдением всех принципов Scrum, разрешает противоречия и защищает команду от отвлекающих факторов;
- владелец продукта (Product Owner) – представляет интересы конечных пользователей и других заинтересованных в продукте сторон;
- скрам-команда (Scrum Team) – кросс-функциональная команда разработчиков проекта, состоящая из специалистов разных профилей: тестировщиков, архитекторов, аналитиков, программистов и т. д; размер команды в идеале составляет от 5 до 9 человек.

ЗАКЛЮЧЕНИЕ

Большое разнообразие методик разработки ИС позволяет подобрать наиболее подходящую из них, используя в качестве критериев размер и сложность проекта, численность и квалификацию команды разработчиков, требования заказчика системы. Описанные в настоящей работе методики используются фирмами-разработчиками ПО для создания самых разных информационных систем во всех областях автоматизации человеческой деятельности. Их особенности, также как и черты сходства, становятся ясными только в процессе практической деятельности по созданию информационных систем.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Грекул, В.И. Проектирование информационных систем / В.И. Грекул, Г.Н. Денищенко, Н.Л. Коровкина. – М. : Бином, 2010. – С. 300.
2. Коберн, А. Быстрая разработка программного обеспечения / А. Коберн. – М. : Лори, 2013. – С. 314.
3. Кон, М. Пользовательские истории. Гибкая разработка программного обеспечения / М. Кон. – М. : Вильямс, 2012. – С. 256.
4. Кон, М. Scrum: гибкая разработка ПО / М. Кон. – М. : Вильямс, 2011. – С. 576.
5. Ларман, К. Применение UML 2.0 и шаблонов проектирования / К. Ларман. – М. : Вильямс, 2013. – С. 736.
6. Фаулер, М. Рефакторинг. Улучшение существующего кода / М. Фаулер, К. Бек, Дж. Брант [и др.]. СПб. : Символ-Плюс, 2013 – С. 432.
7. DSDM Atern Handbook [Электронный ресурс]. – Режим доступа: http://www.dsdm.org.
8. Microsoft Corporation. Microsoft Solutions Framework: Гибкая методология разработки программного обеспечения. Справочное руководство. – М. : Русская редакция, 2010. – С. 127.

Учебное электронное текстовое издание

Солонин Евгений Борисович

СОВРЕМЕННЫЕ МЕТОДИКИ РАЗРАБОТКИ ИНФОРМАЦИОННЫХ СИСТЕМ

Выпускающий редактор	*Н.В. Лутова*
Редактор	*А.В. Ерофеева*
Компьютерная верстка	*авторская*

Рекомендовано Методическим советом ФГАОУ ВПО УрФУ

Разрешено к публикации 07.10.2015

Электронный формат – .pdf

Объем 2,46 уч.-изд. л.

620002, Екатеринбург, ул. Мира, 19

Информационный портал УрФУ

http://study.urfu.ru