Абрамова О. Ф.

Индустриальная разработка программных продуктов

Часть 2

Учебное пособие

Волжский

2021

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
ВОЛЖСКИЙ ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ (ФИЛИАЛ)
ФЕДЕРАЛЬНОГО ГОСУДАРСТВЕННОГО БЮДЖЕТНОГО ОБРАЗОВАТЕЛЬНОГО
УЧРЕЖДЕНИЯ ВЫСШЕГО ОБРАЗОВАНИЯ
«ВОЛГОГРАДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Абрамова О.Ф.

Индустриальная разработка программных продуктов

Часть 2

Электронное учебное пособие

2021

УДК 004.45(07)
ББК 32.81я73
А 161

Рецензенты:
заведующий кафедрой математики, информатики и естественных наук Волжского филиала Волгоградского государственного университета, канд. физ.-мат. наук, доцент
Полковников А. А.;
к.п.н., доцент, кафедра физики, методики преподавания физики и математики, ИКТ Волгоградский государственный социально-педагогический университет
Филиппова Е. М.

Издается по решению редакционно-издательского совета
Волгоградского государственного технического университета

Абрамова, О.Ф.
Индустриальная разработка программных продуктов [Электронный ресурс] : учебное пособие/ О.Ф. Абрамова ; Министерство науки и высшего образования Российской Федерации, ВПИ (филиал) ФГБОУ ВО ВолгГТУ. – Электрон. текстовые дан. (1 файл: 244 КБ). – Волжский, 2021. – Режим доступа: http://lib.volpi.ru. – Загл. с титул. экрана.
ISBN 978-5-9948-4054-2

В пособии собран методический материал (часть 2) к лекционному курсу по дисциплине «Индустриальная разработка программных продуктов».

Предназначено для студентов, обучающихся по направлению подготовки бакалавров 09.03.01 «Информатика и вычислительная техника» и 09.03.04 «Программная инженерия» и рекомендуется как основной источник для изучения дисциплины «Индустриальная разработка программных продуктов».

Ил. 1, библиограф.: 30 назв.

ISBN 978-5-9948-4054-2

Оглавление

ВВЕДЕНИЕ

В настоящее время производство и продажа программных продуктов приобрели черты бизнеса высокорентабельного вида. Это связано как с высокой долей интеллектуального труда создателей программ, так и с низкой материалоемкостью процессов производства. Процесс проектирования и разработки программного обеспечения представляет собой достаточно специфический набор из взаимосвязанных этапов, многие из которых автоматизированы и поддерживаются специальными программными инструментами. Он реализует преобразование исходных требований заказчика о предметной области в готовый программный продукт. В этом процессе участвуют как специалисты, выполняющие определенную работу с помощью специфических инструментальных средств и предметов производства, так и заказчики, оценивающие и впоследствии оплачивающие разработку. На вход этого процесса в виде требований поступают сведения о предметной области. На выходе процесса – разработанное программное обеспечение и техническая документация. Одной из важнейших особенностей процесса можно считать материальную заинтересованность участников, что существенно влияет как на сам процесс, так и на результат. Секрет успеха реализации проектов в области промышленного программирования – это выполнение работы в заданные сроки в рамках определенного предварительно бюджета с заданным качеством. Жесткая конкуренция разработчиков в сфере IT-технологий объективно требует от разработчиков обращать также максимум внимания и на экономику и маркетинг их разработки, и продвижения на рынок. Кроме того, индустриальные способы производства должны основываться на современных инструментальных средствах создания программных продуктов, поддерживающих все этапы жизненного цикла, начиная от выявления требований пользователей и заканчивая поставкой им готового

продукта, снабженного качественной документацией. Очевидно также, что реализация процесса индустриальной разработки программных продуктов невозможна без учета разнообразных стандартов и нормативных документов как корпоративных, так и отраслевых, государственных и международных. Стандарты в области информационных технологий не являются жесткими и требующими неукоснительного исполнения, однако они регламентируют требования к процессам жизненного цикла, качества, документирования программного продукта, а также требования к оценке зрелости организаций-разработчиков, позволяя достичь установленных целей как команде разработчиков, так и участвующим в процессах заказчикам.

Все эти вопросы входят в круг интересов программной инженерии как методологии промышленного создания и продвижения на рынок качественных и экономичных программных продуктов и систем. Студенты, обучающиеся по направлению 09.03.04 «Программная инженерия», должны ознакомиться, изучить и иметь практические навыки работы на каждом из этапов процесса проектирования и разработки программных систем. Это будущие профессионалы, занимающиеся решением конкретных практических задач, отвечающие за реализацию жизненного цикла программного продукта на всех этапах, знающие и осмысленно использующие существующие стандарты и нормативные документы в области индустриальной разработки программных продуктов.

Данное пособие преследует цель ознакомить студентов направления 09.03.04 «Программная инженерия» с этапами, методами и особенностями индустриальной разработки программных продуктов с учетом разного рода стандартов в этой области.

ЛЕКЦИЯ 5

ПРОГНОЗИРУЕМЫЕ МЕТОДОЛОГИИ ПРОЕКТИРОВАНИЯ ПРОГРАММНЫХ ПРОДУКТОВ

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.

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.

Прогнозируемые (промышленные) методологии фокусируются на детальном планировании будущего. Известны запланированные задачи и ресурсы на весь срок проекта. Команда с трудом реагирует на возможные изменения. План оптимизирован исходя из состава работ и существующих

требований. Изменение требований может привести к существенному изменению плана, а также дизайна проекта. Часто создается специальный комитет по «управлению изменениями», чтобы в проекте учитывались только самые важные требования.

К прогнозируемым методологиям относят:

- каскадная (водопадная) модель;
- прототипирование;
- инкрементная модель;
- спиральная модель;
- RUP;
- MSF;
- быстрая разработка приложений (RAD).

Рассмотрим выделенные методологии подробнее.

КАСКАДНАЯ МОДЕЛЬ (WATERFALL)

Одна из самых первых методологий, которая подразумевает последовательное прохождение классических стадий проектирования и разработки программного продукта, каждая из которых должна завершиться полностью до начала следующей. Годом рождения методологии можно считать 1970, в котором вышла статья Уинстона Уолкера Ройса, директора Lockheed Software Technology Center, в которой он смоделировал подход к разработке ПП в форме каскадной модели и описал ее.

Этапы WaterFall

Классический водопадный подход к проектированию ПП предполагает 5 основных этапов плюс дополнительный, в котором идет речь о сопровождении реализованного ПО.

- *Аналитика***.** Самый важный этап. Исполнитель обсуждает продукт с заказчиком, получает требования, прописывает и утверждает планы, цели и бюджет, графики работ, процессы, риски. После этого нужно

составить техническое задание и инструкции. Отступать от них на следующих этапах нельзя.

- *Проектирование***.** На этом этапе создается прототип ПО. Также нужно выбрать платформу для программирования и утвердить роли в команде.
- *Разработка***.** Этап кодинга продукта четко по техническому заданию.
- *Тестирование***.** Основная цель этапа – проверка соответствия кода техническому заданию.
- *Эксплуатация***.** Этап предполагает выпуск и внедрение ПП. В ходе этапа предполагается анализировать результат, собрать фидбэк и указать критичные баги. Если их много, весь процесс придется начать сначала.
- *Поддержка***.** На завершающем этапе исполнителю остается поддерживать работоспособность, устранять сбои и собирать обратную связь от пользователей, чтобы расширить или заменить функционал.

Каскадная разработка ПП предполагает работу по заранее написанному и согласованному техническому заданию с максимально подробным документированием каждого этапа. В модели Waterfall легко управлять проектом. Данный подход не предполагает итераций, здесь не допустим возврат на предыдущий этап для внесения изменений. А для выявления и исправления ошибок предназначен только этап тестирования. Поэтому, благодаря таким жестким ограничениям, разработка проходит быстро, стоимость и срок заранее определены, что очень важно для разработки заказных ПП. Однако нужно понимать, что каскадная модель будет давать отличный результат только в тех проектах, в которых четко и заранее определены требования и способы их реализации. Продукты, разработанные по данной модели без обоснованного ее выбора, могут иметь недочеты (список требований нельзя скорректировать в любой момент), о которых

становится известно лишь в конце процесса из-за строгой последовательности действий. Стоимость внесения изменений высока, так как для ее инициализации приходится ждать завершения всего проекта.

Современная разработка внесла некоторые коррективы в классическую каскадную модель и при сохранении основных преимуществ предлагает несколько вполне жизнеспособных модификаций. С учетом начальной можно выделить следующие

модификации WaterFall:

1. линейная;
2. итерационная (1970, обратная связь после каждого этапа);
3. каскадная (завершение каждого этапа проверкой);
4. строгая каскадная (минимизация возвратов);
5. V-модель.

Преимущества каскадной модели разработки ПП

К основным преимуществам каскадной модели относятся:

- стабильность требований в течение всего жизненного цикла разработки;
- возможность последовательного устранения возникающих сложностей;
- сравнительная легкость внесения изменений и уточнений на первых двух этапах, так как разработка не поддержана в это время кодом;
- определенность и понятность шагов модели и простота ее применения;
- упрощение возможности осуществления планирования, контроля и управления проектом;
- доступность для понимания заказчиками;
- эффективность для проектов с четкими и понятными, но трудно реализуемыми требованиями;
- эффективность для проектов с высокими требованиями к качеству при отсутствии жестких ограничений затрат и графика работ;
- наличие подробной и качественной документации.

Недостатки каскадной модели разработки ПП

При использовании классической каскадной модели для несоответствующего ей проекта выявляются следующие ее недостатки:

- сложность четкого формулирования требований в начале жизненного цикла и невозможность их динамического изменения на его протяжении;
- последовательность линейной структуры процесса разработки, в результате возврат к предыдущим шагам для решения возникающих проблем приводит к увеличению затрат и нарушению графика работ;
- непригодность промежуточного продукта для использования;
- невозможность гибкого моделирования систем, не имеющих аналогов;
- позднее обнаружение проблем, связанных со сборкой, в связи с одновременной интеграцией всех результатов в конце разработки;
- недостаточное участие пользователя в создании системы – только в самом начале (при разработке требований) и в конце (во время приемочных испытаний);
- невозможность предварительной оценки качества системы пользователем;
- проблемность финансирования проекта, связанная со сложностью единовременного распределения больших денежных средств.

Область применения каскадной модели

Ограничение области применения каскадной модели определяется ее недостатками. Ее использование наиболее эффективно в следующих случаях:

- только тогда, когда требования известны, понятны и зафиксированы;
- противоречивых требований не имеется;
- изменять требования не планируется;
- заказчик не планирует участвовать в разработке;
- нет проблем с доступностью программистов нужной квалификации;
- в относительно небольших проектах;

- при разработке проекта, ориентированного на построение системы или продукта такого же типа, как уже разрабатывались разработчиками ранее;
- при разработке проекта, связанного с созданием и выпуском новой версии уже существующего продукта или системы.

ПРОТОТИПИРОВАНИЕ

Сложно назвать прототипирование отдельно стоящей методологией, но большое количество существующих методологий проектирования и разработки программных продуктов используют в своих жизненных циклах его идеи, преимущества и ограничения. Поэтому мы рассмотрим данный метод отдельно от других.

Как правило, на начальной стадии проекта, особенно инновационного, полностью и точно сформулировать все требования к будущей модели невозможно, т.к. пользователи, как это часто бывает, не в состоянии изложить ясно и четко все свои требования. Так же зачастую невозможно предвидеть, как изменятся требования в ходе разработки, и какие изменения могут произойти во внешней среде, способные повлиять на требования к реализуемой системе. Поэтому процесс создания ПО носит скорее итерационный характер, когда результаты очередной стадии разработки могут вызвать необходимость возврата к предыдущим разработкам, и для этого применяется макетирование (прототипирование), т.е. создается модель требуемого программного продукта. *Под прототипом* в данном случае понимается действующий программный компонент, реализующий отдельные функции.

Модель (прототип) может принимать одну из трех форм:

– бумажный макет или макет на основе ПК (изображает или рисует человеко-машинный диалог),

– работающий макет (выполняет некоторую часть требуемых функций),

– программа, характеристики которой затем должны быть улучшены.

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.

Алгоритм прототипирования

1. Сбор и уточнение требований
2. Быстрое проектирование
3. Построение макета
4. Оценка макета заказчиком
    - Заказчик не удовлетворен
        - Уточнение требований

- Переход к п. 2

- Заказчик удовлетворен
  - Переход к п. 5

5. Конструирование продукта

Прототипирование (макетирование) начинается со сбора и уточнения требований к создаваемому ПО. Совместными усилиями разработчик и заказчик определяют все цели ПО, устанавливают, какие требования известны, а какие предстоит доопределить. Для этого важно расставить правильно приоритеты в целях и требованиях. Следующим шагом является быстрое проектирование, внимание в котором сосредотачивается на тех характеристиках ПО, которые должны быть видимы пользователю. Макет (прототип) оценивается заказчиком и используется для уточнения требований к ПО. Итерации повторяются до тех пор, пока макет не выявит все требования заказчика и не даст возможности разработчику понять, что должно быть сделано.

Виды прототипов

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 еще называют *поведенческим прототипом* (behavioral prototype). Название прототипа основано на том, 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.

Раскадровка

Решением промежуточного между электронным и бумажным вариантами прототипов UI класса является презентация, изготовленная при помощи средств электронного офиса (например, комбинации Microsoft Visio и Microsoft PowerPoint). В этом случае пользователь лишен свободы выбора, предоставляемой ему поведенческим прототипом. Но идею пошаговой смены экранов в процессе реализации сценария варианта использования вполне можно реализовать. Данный вид решения определяется как пассивная

раскадровка. Активная раскадровка является дальнейшим развитием понятия пассивной раскадровки, с применением средств анимации и т.п. Третий вид раскадровки – интерактивная, которая представляет собой электронный одноразовый горизонтальный прототип.

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 создаются 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.

Область применения прототипирования

Менеджер проекта может быть уверен в необходимости применения структурной эволюционной модели быстрого прототипирования, если:

- требования не известны заранее;
- требования не постоянны или могут быть неверно истолкованы или неудачно сформулированы;
- следует уточнить требования;
- существует потребность в разработке пользовательских интерфейсов;
- нужна проверка концепции;

- осуществляются временные демонстрации;
- построенное по принципу структурной модели, эволюционное быстрое прототипирование можно успешно использовать в больших системах, в которых некоторые модели подвергаются прототипированию, а некоторые – разрабатываются более традиционным образом;
- выполняется новая, не имеющая аналогов разработка (в отличие от эксплуатации продукта на уже существующей системе);
- требуется уменьшить неточности в определении требований; т.е. уменьшается риск создания системы, которая не имеет никакой ценности для заказчика;
- требования подвержены быстрым изменениям, когда заказчик неохотно соглашается на фиксированный набор требований или если о прикладной программе отсутствует четкое представление;
- разработчики не уверены в том, какую оптимальную архитектуру или алгоритмы следует применять;
- алгоритмы или системные интерфейсы усложнены;
- требуется продемонстрировать техническую осуществимость, когда технический риск высок;
- задействованы высокотехнологические системы с интенсивным применением ПО, где можно лишь обобщенно, но не точно сформулировать требования, лежащие за пределами главной характеристики;
- разрабатывается ПО, особенно в случае программ, когда проявляется средняя и высокая степень риска;
- осуществляется применение в комбинации с каскадной моделью: на начальном этапе проекта используется прототипирование, а на последнем – фазы каскадной модели с целью обеспечения функциональной эффективности системы и качества;

- прототипирование всегда следует использовать вместе с элементами анализа и проектирования, применяемыми при объектно-ориентированной разработке. Быстрое прототипирование особенно хорошо подходит для разработки интенсивно используемых систем пользовательского интерфейса, таких как индикаторные панели для контрольных приборов, интерактивные системы, новые в своем роде продукты, а также системы обеспечения принятия решений, среди которых можно назвать подачу команд, управление или медицинскую диагностику.

ИНКРЕМЕНТНАЯ МОДЕЛЬ

Инкрементная модель (increment в переводе с англ. – приращение) – это метод, в котором программный продукт проектируется, реализуется и тестируется по частям, то есть каждый раз с небольшими добавлениями до самого окончания разработки, включая дальнейшую поддержку продукта. Т.е. возможно ситуация, когда проект с минимальной функциональностью уже выпущен на рынок, а параллельно идет разработка дополнительных модулей, запланированных на начальном этапе, но обладающих более низким приоритетом, либо необходимость в которых возникла по результатам анализа эксплуатации. Проект считается законченным, когда удовлетворяет всем заявленным требованиям к программному продукту. Инкрементная модель представляет из себя симбиоз элементов каскадной модели и прототипирования. Каждый инкремент в процессе проходит через основные фазы жизненного цикла: кодирование, тестирование и поставку.

Спецификация программного обеспечения, проект и реализация делятся на части (increment), которые разрабатываются одна за другой. Сначала выполняется конструирование, тестирование и реализация набора функций, формирующих основу продукта, или требований первостепенной важности, играющих основную роль для успешного выполнения проекта либо снижающих степень риска. Таким образом, уменьшается количество

нуждающихся в переделке частей системы, и клиенты получают возможность в течение более длительного времени обдумать свои желания. Особенностью инкрементной стратегии является большое количество циклов разработки при незначительной продолжительности цикла и небольших различиях между итерациями соседних циклов.

Инкрементные модели используются там, где отдельные запросы на изменение ясны, могут быть легко формализованы и реализованы. Проект можно разложить на несколько составляющих, каждая из которых спроектирована и построена независимо от другой (*билд*). Каждый такой компонент поставляется клиенту по мере готовности, что позволяет сразу начать использовать продукт и избежать длительной разработки. Это предполагает большие инвестиционные затраты, но существенно сокращает время ожидания результата. Модель помогает «сгладить углы», представляя реализацию работоспособными частями, вместо того, чтобы демонстрировать пользователю совершенно новую систему целиком.

Этапы инкрементной модели

В общем случае разработка начинается с начального планирования, в рамках которого определяются все требования к продукту и выполняется разбиение проекта на инкременты. Далее каждый инкремент предполагает выполнение основных этапов разработки ПО.

- Планирование и анализ
- Инкремент 1:
    - Проектирование
    - Реализация
    - Тестирование
    - Поставка
- Инкремент 2
- …
- Инкремент N

Преимущества инкрементной модели

Применяя инкрементную модель при разработке проекта, для которого она подходит в достаточной мере, можно убедиться в следующих ее преимуществах:

- не требуется заранее тратить средства, необходимые для разработки всего проекта (поскольку сначала выполняется разработка и реализация основной функции или функции из группы высокого риска);
- в результате выполнения каждого инкремента получается функциональный продукт;
- заказчик располагает возможностью высказаться по поводу каждой разработанной версии системы;
- правило по принципу "разделяй и властвуй" позволяет разбить возникшую проблему на управляемые части, благодаря чему предотвращается формирование громоздких перечней требований, выдвигаемых перед командой разработчиков;
- существует возможность поддерживать постоянный прогресс в ходе выполнения проекта;
- снижаются затраты на первоначальную поставку программного продукта;
- ускоряется начальный график поставки (что позволяет соответствовать возросшим требованиям рынка);
- снижается риск неудачи и изменения требований;
- заказчики могут распознавать самые важные и полезные функциональные возможности продукта на более ранних этапах разработки;
- риск распределяется на несколько меньших по размеру инкрементов (не сосредоточен в одном большом проекте разработки);

- требования стабилизируются (посредством включения в процесс пользователей) на момент создания определенного инкремента, поскольку не являющиеся особо важными изменения отодвигаются на момент создания последующих инкрементов;
- инкременты функциональных возможностей несут больше пользы и проще при тестировании, чем продукты промежуточного уровня при поуровневой разработке по принципу "сверху-вниз";
- улучшается понимание требований для более поздних инкрементов (что обеспечивается благодаря возможности пользователя получить представление о раннее полученных инкрементов на практическом уровне);
- в конце каждой инкрементной поставки существует возможность пересмотреть риски, связанные с затратами и соблюдением установленного графика;
- использование последовательных инкрементов позволяет объединить полученный пользователями опыт в виде усовершенствованного продукта, затратив при этом намного меньше средств, чем требуется для выполнения повторной разработки;
- в процессе разработки можно ограничить количество персонала таким образом, чтобы над поставкой каждого инкремента последовательно работала одна и та же команда и все задействованные в процессе разработки команды не прекращали работу над проектом (график распределения рабочей силы может выравниваться посредством распределения по времени объема работы над проектом);
- возможность начать построение следующей версии проекта на переходном этапе предыдущей версии сглаживает изменения, вызванные сменой персонала;

- в конце каждой инкрементной поставки существует возможность пересмотреть риски, связанные с затратами и соблюдением установленного графика;
- потребности клиента лучше поддаются управлению, поскольку время разработки каждого инкремента очень незначительно;
- поскольку переход из настоящего в будущее не происходит моментально, заказчик может привыкать к новой технологии постепенно;
- ощутимые признаки прогресса при выполнении проекта помогают поддерживать вызванное соблюдением графика "давление" на управляемом уровне.

Недостатки инкрементной модели

При использовании этой модели относительно проекта, для которого она подходит не в достаточной мере, проявляются следующие недостатки:

- в модели не предусмотрены итерации в рамках каждого инкремента;
- определение полной функциональной системы должно осуществляться в начале жизненного цикла, чтобы обеспечить определение инкрементов;
- формальный критический анализ и проверку намного труднее выполнить для инкрементов, чем для системы в целом;
- заказчик должен осознавать, что общие затраты на выполнение проекта не будут снижены;
- поскольку создание некоторых модулей будет завершено значительно раньше других, возникает необходимость в четко определенных интерфейсах;
- использование на этапе анализа общих целей, вместо полностью сформулированных требований, может оказаться неудобным для руководства;

- для модели необходимы хорошее планирование и проектирование: руководство должно заботиться о распределении работы, а технический персонал должен соблюдать субординацию в отношениях между сотрудниками.
- может возникнуть тенденция к оттягиванию решений трудных проблем на будущее с целью продемонстрировать руководству успех, достигнутый на ранних этапах разработки.

Область применения инкрементной модели

Менеджер проекта может быть уверен в целесообразности применения модели, если для этого имеются следующие причины:

✓ если большинство требований можно сформулировать заранее, но их появление ожидается через определенный период времени;

✓ если рыночное окно слишком "узкое" и существует потребность быстро поставить на рынок продукт, имеющий функциональные базовые свойства;

✓ для проектов, на выполнение которых предусмотрен большой период времени разработки, как правило, один год;

✓ при равномерном распределении свойств различной степени важности;

✓ когда при рассмотрении риска, финансирования, графика выполнения проекта, размера программы, ее сложности или необходимости в реализации на ранних фазах оказывается, что самым оптимальным вариантом является применение принципа пофазовой разработки;

✓ при разработке программ, связанных с низкой или средней степенью риска;

✓ при выполнении проекта с применением новой технологии, что позволяет пользователю адаптироваться к системе путем выполнения более мелких инкрементных шагов, без резкого перехода к применению основного нового продукта;

- ✓ когда однопроходная разработка системы связана с большой степенью риска;
- ✓ когда результативные данные получаются через регулярные интервалы времени

СПИРАЛЬНАЯ МОДЕЛЬ (SPIRAL MODEL)

Спиральная модель проектирования и разработки ПП похожа на инкрементную модель, но предполагает акцент на анализе рисков. Каждый этап спиральной модели начинается с формулирования цели проектирования и заканчивается тем, что клиент анализирует ход выполнения. Последующая стадия основывается на предыдущей, а в конце каждого витка – цикла итераций – предлагается вариант реализации и принимается решение, продолжать ли проект.

Спиральная модель была разработана из-за появления значительной изменчивости на рынке ПП и является клиент-ориентированным решением. Она впервые была упомянута американским инженером-программистом Барри Боемом (Barry Boehm) в 1988 году в его статье «Спиральная модель разработки и совершенствование программного обеспечения», в которой предложил создавать не цельную программу, а выпускать ряд прототипов, каждый из которых содержит дополнительную или расширенную функциональность по сравнению с предыдущим. Пользователь может изучить и попробовать в деле каждый прототип. Получая обратную связь, разработчик дорабатывает приложение, пока заказчик не получит готовый продукт, который полностью его устраивает. Такая модель хорошо работает для решения критически важных бизнес-задач, когда неудача несовместима с деятельностью компании в условиях выпуска новых продуктовых линеек, с необходимостью дополнительных научных исследований и практической апробации.

Этапы спиральной модели

Спиральная модель предполагает 4 этапа для каждого витка:

1. планирование;
2. анализ рисков;
3. конструирование;
4. оценка результата и при удовлетворительном качестве переход к новому витку.

Для каждой итерации следует определить цели, альтернативные варианты и ограничения; установить и разрешить риски; дать оценку альтернативным вариантам, разработать результативные данные для этой итерации и подтвердить их правильность; спланировать следующую итерацию. Затем следует выбрать метод осуществления следующей итерации в случае, если требуется ее выполнять. Следует отметить тот факт, что кодирование в такой модели выполняется значительно позже, чем в других моделях. Смысл заключается в том, чтобы минимизировать риск поставки не соответствующего требованиям ПП посредством последовательных уточнений требований, выдвигаемых пользователем. Предполагается разработка различных вариантов продукта в соответствии с различными вариантами требований в формате прототипов, которые служат средством общения между заказчиком и разработчиком и помогают уточнять требования. Причем возможен вариант не завершения очередного витка, если доказано, что риск реализации данного варианта неоправданно высок.

В каждом "минипроекте" (движении по спирали) рассматривается один или несколько главных факторов риска, начиная с фактора наивысшего риска. Типичные риски включают в себя неправильно истолкованные требования, архитектуру, потенциальные проблемы, связанные с эксплуатацией продукта, проблемы в базовой технологии и т.д.

Спиральная модель также предполагает активное участие заказчика в процессе разработки, в отличие от каскада. Он оценивает риски и принимает решения по продолжению или завершению очередного варианта или всего проекта.

Преимущества спиральной модели

При использовании спиральной модели при выполнении проекта, для которого она в достаточной мере подходит, проявляются следующие преимущества:

- спиральная модель обеспечивает разбиение большого потенциального объема работы по разработке продукта на небольшие части, в которых сначала реализуются решающие функции с высокой степенью риска, позволяющие устранить необходимость продолжения работы над проектом (таким образом, в случае необходимости становится возможным прекратить работу над проектом, и уменьшаются расходы);
- она разрешает пользователям "увидеть" систему на ранних этапах, что обеспечивается посредством использования ускоренного прототипирования в жизненном цикле разработки ПО;
- обеспечивается определение непреодолимых рисков без особых дополнительных затрат;
- эта модель разрешает пользователям активно принимать участие при планировании, анализе рисков, разработке, а также при выполнении оценочных действий;
- в модели предусмотрена возможность гибкого проектирования, поскольку в ней воплощены преимущества каскадной модели, и в то же время разрешены итерации по всем фазам этой же модели;
- в спиральной модели реализованы преимущества инкрементной модели, а именно выпуск инкрементов, сокращение графика посредством перекрывания инкрементов, рассортированных по версиям, и неизменяемость ресурсов при постепенном росте системы;
- отсутствует цель выполнить невозможное – довести конструкцию до совершенства;

- наличествует бесперебойная обратная связь от пользователей к разработчикам, что обеспечивает создание нужного продукта высокого качества;
- происходит усовершенствование административного управления над процессом обеспечения качества, правильностью выполнения процесса разработки, затратами, соблюдением графика и кадровым обеспечением, что достигается путем выполнения обзора в конце каждой итерации;
- повышается продуктивность благодаря использованию пригодных для повторного использования свойств;
- повышается вероятность предсказуемого поведения системы с помощью уточнения поставленных целей;
- при использовании спиральной модели не нужно распределять заранее все необходимые для выполнения проекта финансовые ресурсы;
- можно выполнять частую оценку совокупных затрат, а уменьшение рисков связано с затратами.

Недостатки спиральной модели

При использовании спиральной модели относительно проекта, для которого она *не* подходит в достаточной мере, проявляются следующие недостатки:

- т.к. оценка рисков после прохождения каждой спирали связана с большими затратами, то при сравнительно низкой степени риска проекта или небольших размеров, модель может оказаться дорогостоящей;
- высокая необходимость в высокопрофессиональных знаниях для оценки рисков;
- при неуверенности заказчика в ожидаемом решении спираль может продолжаться до бесконечности, поскольку каждая

ответная реакция заказчика на созданную версию может порождать новый цикл, что отдаляет окончание работы над проектом;

- большое количество промежуточных стадий может привести к необходимости в обработке внутренней дополнительной и внешней документации;
- время, затраченное на планирование, повторное определение целей, выполнение анализа рисков и прототипирование может быть чрезмерным, что может сделать модель недопустимой по средствам;
- при выполнении действий на этапе вне процесса разработки возникает необходимость в переназначении разработчиков;
- могут возникнуть затруднения при определении целей и стадий, указывающих на готовность продолжать процесс разработки на следующей итерации;
- необходимость в хорошем средстве или методе прототипирования.

Область применения спиральной модели

Эта модель не подойдет для малых проектов, она резонна для сложных и дорогих, требующих скрупулезного анализа для оценки последствий больше, чем программирования.

Менеджер проекта может быть уверен в целесообразности применения спиральной модели, если для этого существует хотя бы одна из следующего перечня причин:

- ✓ создание прототипа представляет собой подходящий тип разработки продукта;
- ✓ есть четкий запрос на расчет затрат на реализацию рискованных действий и их разъяснение;
- ✓ организация обладает навыками, требуемыми для адаптации модели;
- ✓ проект обладает средней или высокой степенью риска;

- ✓ пользователи не уверены в своих потребностях;
- ✓ когда нет смысла браться за выполнение долгосрочного проекта из-за потенциальных изменений, которые могут произойти в экономических приоритетах, изучении или исследовательской работе и когда такая неопределенность может вызвать ограничение во времени;
- ✓ необходимость применения новой технологии либо тестирования базовых концепций;
- ✓ требования к ПП слишком сложные;
- ✓ разработка новой функции или новой серии продуктов.

ЛЕКЦИЯ 6

ПРОМЫШЛЕННЫЕ МЕТОДОЛОГИИ РАЗРАБОТКИ ПРОГРАММНЫХ ПРОДУКТОВ

А сейчас поговорим о методологиях, разработанных на конкретных производствах.

УНИФИЦИРОВАННЫЙ ПРОЦЕСС RATIONAL (RUP)

Методология Унифицированный Процесс Rational (RUP), которая была разработана в компании Rational Software Corporation и в 2003 году куплена фирмой IBM, – это универсальная методология распределения задач и сфер ответственности при разработке программного обеспечения. Её цель – создание высококачественного программного обеспечения, отвечающего потребностям и запросам пользователей.

Методология RUP предназначена для крупных проектов разработки, поэтому многие менеджеры уверены, что она не подойдёт для небольших задач, не требующих большого объёма ресурсов. Но есть множество примеров, когда небольшие проекты значительно выигрывали от внедрения RUP. Невероятный успех предложенного в рамках RUP подхода помог многим компаниям понять, насколько важно иметь чётко прописанный и задокументированный процесс разработки.

Наибольшее внимание RUP уделяет начальным стадиям разработки проекта, анализу и моделированию, что призвано снизить риски проекта за

счет раннего обнаружения возможных ошибок. 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 (Unified Modelling Language) и включают варианты как правильных, так и ошибочных последовательностей (исключений), что помогает команде легко донести свои требования к проекту, его архитектуру и план реализации. Последовательный выпуск версий организован таким образом, чтобы наиболее существенные риски устранялись в первую очередь.

RUP основан на трех ключевых идеях:

- Весь ход работ направляется итоговыми целями проекта, выраженными в виде вариантов использования (use cases).
- Основным решением, принимаемым в ходе проекта, является архитектура результирующей программной системы. Архитектура устанавливает набор компонентов, из которых будет построено ПО, ответственность каждого из компонентов (т.е. решаемые им подзадачи в рамках общих задач системы), четко определяет интерфейсы, через которые они могут взаимодействовать, а также способы взаимодействия компонентов друг с другом.
- Основой процесса разработки являются планируемые и управляемые итерации, объем которых (реализуемая в рамках итерации функциональность и набор компонентов) определяется на основе архитектуры.

В основе RUP лежит шесть главных принципов:

1. Итеративная модель разработки – устранение рисков на каждой стадии проекта позволяет лучше понять проблему и вносить необходимые изменения, пока не будет найдено приемлемое решение;
2. Управление требованиями – RUP описывает процесс организации и отслеживания функциональных требований, документации и выбора

оптимальных решений (как в процессе разработки, так и при ведении бизнеса);

3. Компонентная архитектура – архитектура системы разбивается на компоненты, которые можно использовать как в текущем, так и в будущих проектах;
4. Визуальное моделирование ПО – RUP методология разработки показывает, как создать визуальную модель программного обеспечения, чтобы понять структуру и поведение архитектуры и его компонентов;
5. Проверка качества ПО – в процессе разработки программного обеспечения контролируется качество всех действий команды;
6. Контроль внесённых изменений – отслеживание изменений позволяет выстроить непрерывный процесс разработки. Создаётся благоприятная обстановка, в рамках которой команда будет защищена от изменений в рабочем процессе.

RUP оптимизирует командную работу – обеспечивает команде разработчиков свободный доступ к базе знаний с инструкциями для использования программных средств. Это помогает быстрее справиться с критическими проблемами. Благодаря этому команда легко находит общий язык в процессе работы над проектом. И при этом RUP – конфигурируемый процесс, который может использоваться как небольшими группами разработчиков, так и крупными организациями в области проектирования и разработки программных продуктов.

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 в RUP состоит из девяти дисциплин и четырех фаз, предполагающий выполнение поделенными на десятки ролей разработчиками примерно полутора сотен различных задач и разработку почти сотни различных артефактов. В сравнении с этим меркнут все требования ГОСТов и СММ вместе взятых, однако это устоявшаяся и максимально апробированная методология, позволяющая сократить, как это ни парадоксально звучит, сроки и стоимость разработки ПО.

Динамическую структуру процесса составляют *фазы и итерации*.

Фазы RUP:

1. Начальная фаза (Inception)
2. Уточнение (Elaboration)
3. Построение (Construction)
4. Передача (Transition).

Каждая фаза может включать в себя одну или несколько итераций процесса создания системы.

Статическую структуру RUP составляют описания работ и задач (части работы), описания создаваемых артефактов, а также рекомендации по их выполнению, которые группируются в дисциплины.

Дисциплины RUP :

- шесть основных
  - *бизнес-моделирование* (Business Modeling),
  - *управление требованиями* (Requirements),
  - *анализ и проектирование* (Analysis and Design),
  - *реализация* (Implementation),
  - *тестирование* (Test),
  - *внедрение* (Deployment),
- три вспомогательных

- *управление конфигурациями и изменениями* (Configuration and Change Management),
- *управление проектом* (Project Management),
- *поддержка среды разработки* (Environment).

Проект, как правило, делится на четыре фазы: *начало* (Inception), *проработка* (Elaboration), *построение* (Construction) и *передача* (Transition). Фазы, в свою очередь, делятся на итерации. В ходе каждой итерации выполняются работы и задачи из различных дисциплин, соотношение этих работ меняется в зависимости от фазы.

Рассмотрим их подробнее.

Начальная стадия (фаза анализа)

Посвящена сбору требований будущих пользователей к системе. Из рассказов пользователей о своей части работы (сценарий использования системы пользователем) аналитик пытается собрать целый рабочий процесс. Цель этой итерации выйти на бизнес-сценарий использования системы / будущий бизнес-процесс на основе ИС, которых может быть больше чем один. 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.

Передача

В этой фазе финальная версия системы внедряется у ее заказчика. Фаза включает:

- проведение испытаний системы,
- обучение пользователей,
- оценку качества ИС.

В случае, если качество системы не соответствует требованиям заказчика, фаза «Внедрение» выполняется повторно. Выполнение всех требований и достижение целей проекта означает завершение полного цикла разработки.

Методология RUP не ставит условий по количеству итераций и их распределению по фазам. Концепт фаз и итераций введен для снижения рисков, потребности в ресурсах, задания ритма реализации проекта и предоставления возможности внести требуемые изменения при разработке большой информационной системы.

Результат каждой итерации уточняет план следующей и даёт больше понимания о том, понадобятся ли нам дополнительные ресурсы для следующей итерации, нужны ли будут дополнительные итерации для завершения фазы. Начать работать над проектом может минимальная команда из нескольких человек и её расширение будет обосновываться планом следующей итерации.

Результаты завершения фазы уточняют план и экономику проекта. Здесь принимаются решения, продолжать ли проект, нужно ли сузить или

расширить содержание проекта (объем бизнес-требований), чтобы остаться в бизнес рамках: нужного срока доставки результата проекта, его стоимости (бюджета) и доходности (рентабельности вложений).

Принципиальное отличие RUP от многих других итеративных подходов состоит в большом внимании к разработке архитектуры системы. Однако следует понимать, что в RUP понятие архитектуры не ограничивается общепринятыми взглядами, как общее описание системы, а включает, в частности, используемые в проекте типовые решения для доступа к СУБД, реализации параллельных процессов и т.д., а также ключевую часть кода (обычно до 20% общего объема), которая позволяет продемонстрировать соответствие системы основным функциональным и нефункциональным требованиям. Поэтому в RUP часто говорится об *исполняемой архитектуре* (executable architecture).

Артефакты методологии

Артефакты, вырабатываемые в ходе проекта, могут быть представлены в виде баз данных и таблиц с информацией различного типа, разных видов документов, исходного кода и объектных модулей, а также моделей, состоящих из отдельных элементов.

Наиболее важные с точки зрения RUP артефакты проекта – это модели, описывающие различные аспекты будущей системы. Большинство моделей представляют собой наборы диаграмм UML. Основные используемые виды моделей следующие:

- *Модель вариантов использования (Use-Case Model).* Эта модель определяет требования к ПО в виде набора вариантов использования и служит основой для проектирования и оценки готовности системы к внедрению.
- *Модель анализа (Analysis Model).* Она включает основные классы, необходимые для реализации выделенных вариантов использования, а также возможные связи между классами. Выделяемые классы разбиваются на три разновидности интерфейсные, управляющие и классы данных.

- *Модель проектирования* является детализацией и специализацией модели анализа. Она также состоит из классов, но более четко определенных, с более точным и детальным распределением обязанностей, чем классы модели анализа.

- *Модель реализации (Implementation Model).* Под моделью реализации в рамках *RUP* и UML понимают набор компонентов результирующей системы и связей между ними. Под компонентом здесь имеется в виду компонент сборки – минимальный по размерам кусок кода системы, который может участвовать или не участвовать в определенной ее конфигурации, единица сборки и *конфигурационного управления*. Связи между компонентами представляют собой зависимости между ними. Если компонент зависит от другого компонента, он не может быть поставлен отдельно от него.

- *Модель развертывания (Deployment Model).* Модель развертывания представляет собой набор узлов системы, являющихся физически отдельными устройствами, которые способны обрабатывать информацию – серверами, рабочими станциями, принтерами, контроллерами датчиков и пр., со связями между ними, образованными различного рода сетевыми соединениями.

- *Модель тестирования.* В рамках этой модели определяются тестовые варианты или тестовые примеры (testcases) и тестовые процедуры (testscripts). Первые являются определенными сценариями работы одного или нескольких действующих лиц с системой, разворачивающимися в рамках одного из вариантов использования. *Тестовая процедура* представляет собой способ выполнения одного или нескольких тестовых вариантов и их составных элементов (отдельных шагов и проверок).

Моделирование предметной области (бизнес-моделирование, Business Modeling)

Задачи этой деятельности – понять предметную область или бизнес-контекст, в которых должна будет работать система, и убедиться, что все

заинтересованные лица понимают его одинаково, осознать имеющиеся проблемы, оценить их возможные решения и их последствия для бизнеса организации, в которой будет работать система.

Довольно странной была бы ситуация, если бы компания разработала методологию, но не предложила набор специализированных программных продуктов для ее использования. 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 Clear Quest – 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.

Преимущества методологии RUP

- Методология RUP позволяет справляться с изменениями в требованиях независимо от того, исходят они от клиента или возникают в ходе работы над проектом;
- RUP подчёркивает необходимость точной документации.
- Интеграция требований происходит на протяжении всего процесса разработки, и в частности в фазе построения.

Недостатки методологии RUP

- RUP опирается на способность экспертов и профессионалов назначить действия определённым работникам, которые затем обязаны выдать запланированные результаты в виде артефактов.
- Интеграция в процесс разработки может негативно сказаться на другой более фундаментальной деятельности на этапах тестирования.
- Хотя RUP методология разработки показывает отличные результаты, особенно в сфере создания ПО, это довольно сложный метод, который трудно внедрить в свой проект, особенно если у вас небольшая команда или компания.
- Деятельность RUP начинается со стадии сбора и формулирования требований к системе. Создание программного обеспечения без этапа обследования предметной области является оправданным только при разработке небольших информационных систем, чего нельзя сказать о построении социально-ориентированной системы городского масштаба, которая затрагивает интересы широких слоев населения, бизнеса и государственного аппарата. На практике перечисленные процедуры часто даже не предшествуют, а лежат в параллельных плоскостях со стадиями жизненного цикла информационной системы.
- Методика RUP предоставляет возможность для документирования только части из перечисленных процедур, поскольку ориентируются в значительной степени на разработку программных систем.
- Деятельность RUP сразу начинается с моделирования процессов «как должно быть» (планируемое состояние процесса) посредством определения вариантов использования.
- Методология имеет дело с системой целиком, а не ее функциональными частями, в результате чего отсутствует возможность компонентного построения системы. RUP ориентирована на поддержку всего жизненного цикла программного продукта без учета модульности, начиная со стадии сбора и формулирования требований к системе, проектирования и

заканчивая стадиями сопровождения, т.е. без возможности возвратов «как бы» в начало проекта после завершения и ввода в эксплуатацию первой части системы.

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. 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.

Модели и дисциплины MSF

MSF разработана как комплекс отдельных компонентов – моделей и дисциплин. Всего их пять, и они описаны в пяти «белых книгах» (white papers) MSF. 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»

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 методологий проектирования: каскадной и спиральной. От водопадной методологии MSF досталась система вех (milestones) – особых точек в конце каждой фазы процесса, отвечающих заданным критериям завершения фазы. В этих точках команда рассматривает результаты своего труда и отвечает на вопросы «Сделали ли мы все, что планировали?», «Работает ли решение так, как нужно заказчику?», «Готовы ли мы двигаться дальше или необходимо уделить внимание доработкам?». Чтобы перейти на следующий этап, необходимо дать большинство положительных ответов.

От спиральной модели MSF унаследовала фокусировку на уточнениях требований к проекту. Разработчик должен постоянно быть готов к тому, что задачи, а порой и цели клиента могут измениться на любом этапе работы.

Тем не менее MSF – это не просто компиляция двух систем. Инновационность методологии заключается в том, что она охватывает жизненный цикл решения от начала проекта до развертывания в реальном времени. Это помогает проектным группам сосредоточиться на бизнес-ценности приложения, поскольку она не будет реализована до тех пор, пока решение не развернуто и не запущено в эксплуатацию.

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 выделенных 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.

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. Команда проекта в MSF – это коллектив равноправных сотрудников. Они разделяют ответственность и свободно обмениваются опытом и информацией. 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.

MSF стремится избавиться от иерархической структуры, поэтому при управлении проектом нет диктатуры. Считается, что демократичные обсуждения, на которых рассматриваются все точки зрения и достигается консенсус, способствуют выработке наиболее удачных решений. Когда члены команды не могут прийти к соглашению, менеджер проекта выступает арбитром: он обязан принять решение, максимально удовлетворяющее клиента и ориентированное на его бизнес-ценности.

Модель проектной группы 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.

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 распределена 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.

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.

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. Дисциплина помогает выявлять, отслеживать и минимизировать риски, в шесть шагов:

- определение рисков;
- анализ и расстановка приоритетов;
- планирование;
- отслеживание;
- контроль;
- знание.

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. Значительным преимуществом можно считать необходимость фиксации полученной информации об изученном риске в специализированную базу знаний для использования этой информации в будущих проектах.

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.

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.

Процесс управления готовностью включает четыре этапа:

1. Определение. На этом этапе выстраивается структура команды. Для каждой роли определяются уровни квалификации и компетенции, необходимые для успешной работы специалистов. Кроме того, вырабатываются сценарии – типичные виды деятельности, которые потребуются для разработки проекта.

2. Оценка. Здесь внимание сосредоточено на каждом члене команды. Проводится анализ компетенций и навыков, связанных с должностными обязанностями, и определяется, насколько они соответствуют желаемым показателям для каждой конкретной роли. Это позволяет выявить разницу между текущим уровнем знаний и требуемым. В результате можно разработать планы индивидуального обучения для каждого сотрудника, которые позволят ему приобрести нужный уровень компетенций.

3. Изменение. В процессе обучения специалисты совершенствуют знания, чтобы преодолеть разрыв между нынешним и желаемым уровнем квалификации. При этом используются учебные планы со списками ресурсов и учебных материалов – учебников, технических документов. Учебный план может предусматривать и самостоятельное изучение, и под руководством наставника.

4. Подведение итогов. На этом этапе проводится повторная оценка знаний и компетенций, чтобы определить, были ли планы обучения эффективными и не требуются ли дополнительные занятия. Рассматриваются не только теоретические знания, но и способность сотрудника использовать их на практике.

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.

Роли в методологии MSF

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

- сотрудничество внутри команды;
- ориентированность на нужды 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.

Бизнес-аналитик

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.

Менеджер проекта

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 определение рисков и выработку 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

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.

Разработчик

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.

Тестировщик

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

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.

Администратор БД

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.

Разработчик БД

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.

Область применения MSF

Пять «белых книг» описывают MSF в мельчайших деталях, давая точные определения ее компонентам, а также подробные описания процессов.

Тем не менее методология MSF остается гибкой и может легко масштабироваться для использования в корпорациях или стартап-проектах. Она очень демократична по своей сути, но категорически требует одного – отказаться от иерархии и диктатуры в управлении. Любое решение должно быть выработано в коллективе коллегиально, а ответственность распределяется между всеми. Кроме того, MSF поощряет постоянный обмен информацией и накопление коллективного опыта, а также подталкивает каждого члена команды к совершенствованию знаний и повышению квалификации.

Методология не требует применять специализированные средства компании Microsoft. Существуют системы, «заточенные» под MSF, например Visual Studio Team System, и Microsoft прямо призывает организации, использующие ее, следовать MSF. Но ничто не мешает применять MSF с любыми другими средствами организации производства.

Сравнительный анализ RUP и MSF

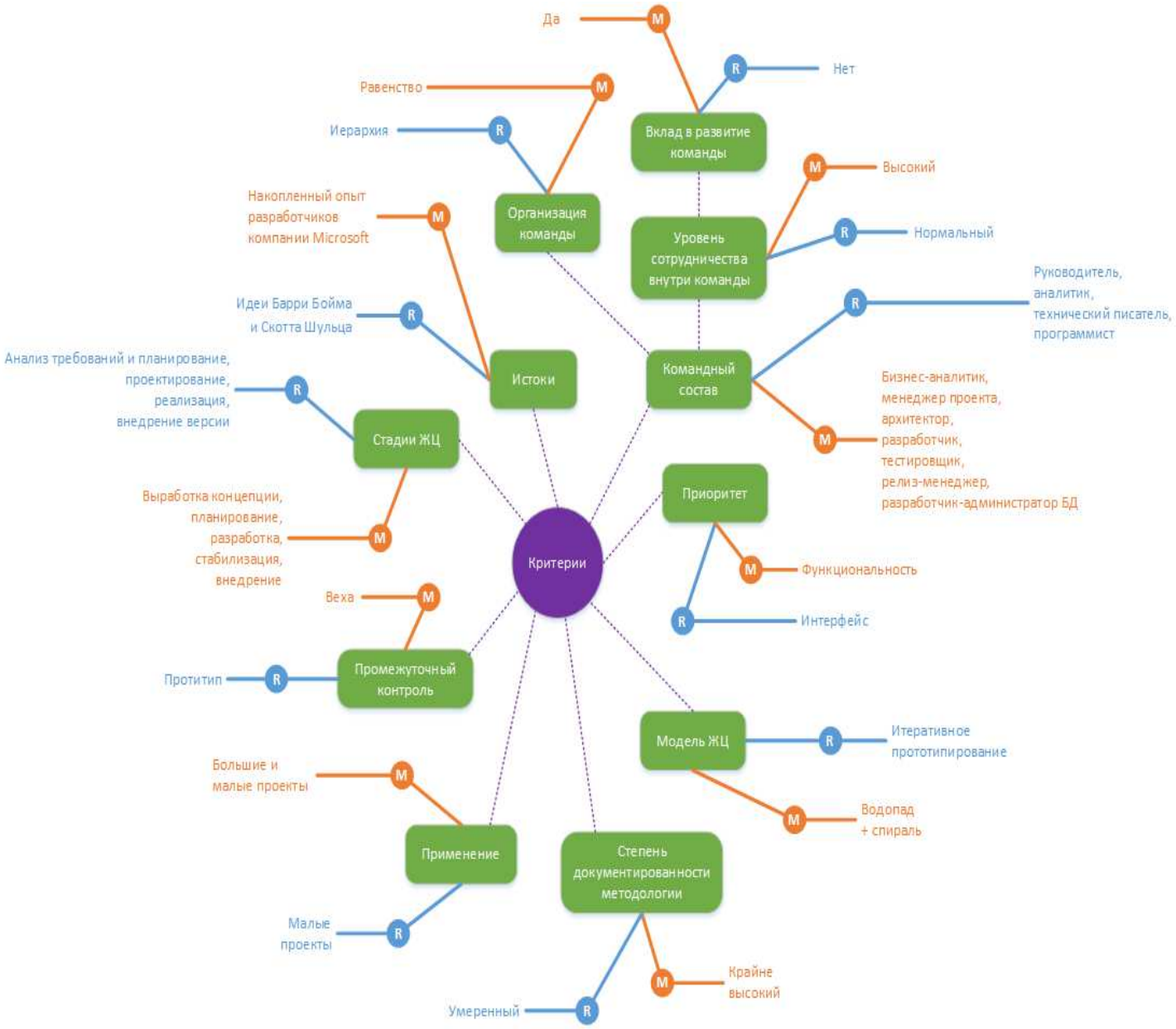

Рисунок 1. Сравнительный анализ RUP и MSF

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. 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.

В 1991 году Дж. Мартин, специалист IBM, опубликовал книгу «Быстрая разработка приложений», в которой детально изложил концепцию RAD и возможности ее применения. Правда не обошлось без путаницы в значении слова «RAD» даже среди IT-специалистов. Ведь речь шла о двух концепциях: RAD как эффективной альтернативе Waterfall и RAD как специфическому методу, разработанному Мартином. Последний был адаптирован к бизнес-системам с интенсивным использованием UI.

Можно сказать, что данная методология является переходной от жестких рамок процесса проектирования программных продуктов в рамках прогнозируемых технологий к более гибкому подходу, характерному agile-философии разработки ПО.

Этапы методологии RAD

К основным приемам RAD относятся следующие.

1. Использование прототипирования, позволяющего точнее 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.

Этапы RAD

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.

Фаза анализа требований и планирования

Определяются требования, функции приложения и их приоритетность, описываются информационные потребности. Фаза выполняется преимущественно пользователями при участии разработчиков. На этой стадии также обозначаются масштаб проекта, временные и финансовые рамки, платформы для запуска ПО.

Фаза проектирования

Часть пользователей участвует в техническом проектировании системы под руководством разработчиков. Группы или подгруппы RAD на этой фазе обычно используют комбинацию техник совместной разработки приложений (JAD) и CASE-инструменты для воплощения потребностей пользователей в рабочих моделях. JAD (Joint Application Development) – концепция совместной разработки приложений, в рамках которой происходит тесное взаимодействие между заказчиком и исполнителями для максимально эффективного решения вопросов, касающихся разрабатываемого ПО.

На фазе проектирования пользователи могут понять, модифицировать и определить рабочую модель системы, которая соответствует их нуждам.

Каждый процесс рассматривается детально и при необходимости создаётся частичный прототип.

В результате фазы создаются:

- общая информационная модель приложения;
- функциональные модели системы и подсистем;
- рабочие прототипы экранов, отчётов и диалогов.

Если в предыдущих моделях средства разработки прототипов не соответствовали реальным приложениям, и они в дальнейшем не использовались, то в RAD каждый прототип становится частью будущей системы.

Фаза реализации

На этой стадии происходит непосредственно быстрая разработка на основе полученных по предыдущим фазам результатов. При этом пользователи продолжают участвовать в развитии системы, предлагая изменения и улучшения приложения. Тестирование приложения тоже происходит во время разработки.

Фаза внедрения

Охватывает обучение пользователей, тестирование и замену старой системы на новую. Подготовка к этой фазе начинается с этапа проектирования.

В отличии от Waterfall, жизненный цикл проекта по методологии RAD не является жёстким. В зависимости от стартовых условий количество фаз может уменьшаться, так же как и их наполнение. Методология RAD требует, чтобы работающие прототипы создавались максимально часто. Продолжительность одного производственного цикла – от выработки требований до демонстрации пользователю (то есть одной итерации) – от одного дня до трех недель.

Роли в методологии RAD

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.

Преимущества методологии RAD

- ✓ высокое качество, достигаемое за счет взаимодействия будущих пользователей с прототипами системы, что повышает

функциональность проектов, выполненных в рамках быстрой разработки приложений;

- ✓ контроль рисков – использование методологии позволяет уже на ранних стадиях разработки сосредоточиться на главных факторах риска и приспособиться к ним;
- ✓ за единицу времени выполняется больше проектов в рамках бюджета за счет использования инкрементную модель разработки. Если в проектах по водопадной системе реализация проекта была возможна после шести и более месяцев анализа и разработки, то в RAD вся необходима информация открывается раньше, во время самого процесса создания приложения.

Недостатки методологии RAD

1. Постоянное участие заказчика (пользователей) в процессе разработки программного продукта;
2. Риск «новизны», необходимость изучения с нуля инструментов и техник приводят к ошибкам во время первых внедрений методологии;
3. Уменьшенный контроль – гибкость, адаптивность процесса можно достигнуть только за счет того, что придётся отдать предпочтение чему-то одному – гибкости или контролю. В последнем случае методика быстрой разработки приложений не будет жизнеспособной.
4. Скудный дизайн – фокусирование на прототипах в некоторых случаях приводит к методике «взлом и тестирование», по которой разработчики постоянно вносят мелкие изменения в отдельные элементы и игнорируют проблемы системной архитектуры.
5. Отсутствие масштабируемости.

Область применения методологии 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.

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.

ЛЕКЦИЯ 7

ГИБКИЕ МЕТОДОЛОГИИ РАЗРАБОТКИ ПРОГРАММНЫХ ПРОДУКТОВ

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. 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, DSDM.

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.

ЭКСТРЕМАЛЬНОЕ ПРОГРАММИРОВАНИЕ (EXP)

Технология eXP (англ. Extreme Programming), как и RAD, базируется на спиральной модели жизненного цикла. 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.

Если говорить о том, в каких случаях применятся экстремальное программирование, то можно привести пример разбитой дороги, где основная цель водителя – преодолеть дистанцию как можно быстрее, испытав при этом минимальные неудобства. Аналогичную цель преследует и методология eXP с единственным отличием, что в качестве неудобств выступают нечеткие и постоянно меняющиеся требования.

Экстремальное программирование – это упрощенная методика организации производства для небольших и средних по размеру команд специалистов, занимающихся разработкой программного продукта в условиях неясных и быстро меняющихся требований. Основные цели ЭП – это:

- повышение доверия заказчика к программному продукту путем предоставления реальных доказательств успешности развития процесса разработки и
- резкое сокращение сроков разработки продукта.

Благодаря этому, достигается то, что практически все приемы XP направлены на повышение качества программного продукта.

Признаки применения:

- короткие циклы;
- планирование по нарастающей;
- гибкий график реализации функциональности;
- базируется на автоматических тестах, разработанных и программистами, и заказчиками;
- обмен сведениями через общение, тесты и исходный код;

- эволюционирующий дизайн.

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).

Принципы eXP

В первой книге Кент Бек сформулировал такие принципы экстремального программирования: простота, коммуникация, обратная связь и смелость. В новом издании книги он добавил пятый принцип – уважение.

1. Простота

В XP разработка начинается с самого простого решения, которое удовлетворит текущую потребность в функциональности. Члены команды учитывают только то, что должно быть сделано сейчас, и не закладывают в код функциональность, которая понадобится завтра, через месяц или никогда.

2. Коммуникация

В XP коммуникация между разработчиками ведется не посредством документации, а вживую. Команда активно общается между собой и с заказчиком.

3. Обратная связь

Обратная связь в XP реализуется сразу в трех направлениях:

- фидбек от системы при постоянном тестировании модулей;
- фидбек от заказчика, т.к. он входит в команду и участвует в написании приемочных тестов;
- фидбек от команды во время планирования касательно времени на разработку.

4. Смелость

Некоторые методики экстремального программирования настолько непривычны, что требует смелости и постоянного контроля над собой.

5. Уважение

В экстремальном программировании уважение рассматривается с точки зрения уважения к команде и самоуважения. Члены команды не должны заливать изменения, которые поломают компиляцию, модульные тесты или замедлят работу коллег. Каждый стремится к высшему качеству кода и дизайна.

В экстремальном программировании планирование – неотъемлемая часть разработки и то, что планы могут поменяться, учитывается с самого начала. Той точкой опоры, методикой, которая позволяет прогнозировать ситуацию и безболезненно мириться с изменениями, является игра в планирование. В ходе такой игры можно быстро собрать известные требования к системе, оценить и запланировать их разработку в соответствии с приоритетностью.

Как и любая другая игра, планирование имеет своих участников и свою цель. Ключевой фигурой является, конечно же, заказчик. Именно он сообщает о необходимости той или иной функциональности. Программисты же дают ориентировочную оценку каждой функциональности. Прелесть игры в планирование заключается в единстве цели и солидарности разработчика и заказчика: в случае победы побеждают все, в случае поражения все проигрывают. Но при этом каждый участник идет к победе своей дорогой:

заказчик выбирает наиболее важные задачи в соответствии с бюджетом, а программист оценивает задачи в соответствии со своими возможностями по их реализации.

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.

Экстремальное программирование предполагает, что разработчики в состоянии сами решить, за какой промежуток времени они справятся со своими задачами и кто из них охотнее бы решил одну задачу, а кто другую. Одним из основных предположений eXP считается следующее: если стоимость внесения в систему изменений со временем растет достаточно медленно, стратегия разработки программы должна быть совершенно другой, отличной от той, которая используется в случае, если стоимость внесения в систему изменений со временем растет экспоненциально. В подобной ситуации вы можете откладывать решение важных задач на более поздние сроки. Вы получаете возможность принимать важные решения настолько поздно, насколько это возможно. Это делается для того, чтобы осуществлять связанные с этим затраты как можно позже. Кроме того, если вы

откладываете решение важных вопросов на более поздний срок, тем самым вы повышаете вероятность того, что выбранное вами решение окажется правильным. Другими словами, сегодня вы должны реализовать только то, без чего сегодня не обойтись, при этом вы можете рассчитывать на то, что проблемы, решение которых вы отложили до завтра, развеются сами собой, то есть перестанут быть актуальными. Вы можете добавлять в дизайн новые элементы только в случае, если эти новые элементы упрощают код или делают написание следующего фрагмента кода более простым.

В качестве фундамента методологии XP вступает понятие живого метода разработки программного обеспечения.

Живые методы разработки – базируются на протесте против чрезмерной бюрократизации разработки ПО, обилия побочных, не являющихся необходимыми для получения конечного результата документов.

На планировании релиза команда программистов встречается с заказчиком, чтобы выяснить, какую функциональность он хочет получить к следующему релизу, то есть через 2-6 месяцев. Так как требования заказчика часто размытые, разработчики конкретизируют их и дробят на части, реализация которых занимает не более одного дня. Важно, чтобы заказчик разбирался в операционной среде, в которой будет работать продукт.

Задачи записываются на карточки, а заказчик определяет их приоритет. Дальше разработчики оценивают, сколько времени уйдет на каждую задачу. Когда задачи описаны и оценены, заказчик просматривает документацию и дает добро на начало работы. Для успеха проекта критично, чтобы заказчик выбирал действительно необходимую функциональность в рамках бюджета, а программисты адекватно сопоставляют требования заказчика со своими возможностями.

В XP если команда не успевает выполнить все задачи к дате релиза, то релиз не отодвигается, а режется часть функционала, наименее важная для заказчика.

Артефакты и методы eXP

- *User Story* – описание того как система должна работать, представляет какой-то кусок функциональности системы, имеющий логический смысл с точки зрения Заказчика.
- *Acceptance Tests* – функциональные тесты по каждой истории, подтверждающие, что данная история корректно реализована.
- *Coding Standards* – форматирование кода, именование классов, переменных, констант, стиль комментариев.
- *Small releases* – версии выпускаются как можно чаще, но каждая версия должна быть достаточно осмысленной.
- *План релизов* – определяет даты релизов и формулировки пользователей, которые будут воплощены в каждом из них. Исходя из этого, можно выбрать формулировки для очередной итерации. В течение итерации изготавливаются тесты приемки, которые выполняются в пределах этой итерации и всех последующих, чтобы обеспечить правильную работу программы. План может быть пересмотрен в случае значительного отставания или опережения по итогам одной из итераций.

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.

Собрание стоя. Каждое утро проводится собрание для обсуждения проблем, их решений и для усиления концентрации команды. Собрание проводится стоя для избежания длительных дискуссий не интересных всем членам команды. В типичном собрании большинство участников ничего не вносят, просто участвуют, чтобы послушать, что скажут другие. Большое количество времени людей тратится, чтобы получить небольшое количество коммуникации. Поэтому участие всех людей в собраниях уводит ресурсы из проекта и создает хаос в планировании.

Для такого рода коммуникаций и нужно собрание стоя. Намного лучше иметь одно короткое обязательное собрание, чем множество длинных, на которых большинство разработчиков должно все равно присутствовать. Если у вас проводятся ежедневные собрания стоя, то все остальные собрания должны посещать только те люди, которые необходимы и будут что-либо привносить. Более того, имеется возможность даже избежать некоторых собраний. С ограничением участников, большинство собраний может быть проведено спонтанно перед монитором, где обмен идеями намного более

интенсивен. Ежедневное утреннее собрание – это не еще одна трата времени. Оно позволит вам избежать многих других собраний и сэкономит больше времени, чем на него затрачено.

Преимущества методологии eXP

- Заказчик получает именно тот продукт, который ему нужен, даже если в начале разработки сам точно не представляет его конечный вид.
- Команда быстро вносит изменения в код и добавляет новую функциональность за счет простого дизайна кода, частого планирования и релизов.
- Код всегда работает за счет постоянного тестирования и непрерывной интеграции.
- Команда легко поддерживает код, т.к. он написан по единому стандарту и постоянно рефакторится.
- Быстрый темп разработки за счет парного программирования, отсутствия переработок, присутствия заказчика в команде.
- Высокое качество кода.
- Снижаются риски, связанные с разработкой, т.к. ответственность за проект распределяется равномерно и уход/приход члена команды не разрушит процесс.
- Затраты на разработку ниже, т.к. команда ориентирована на код, а не на документацию и собрания.

Недостатки методологии eXP

- Успех проекта зависит от вовлеченности заказчика, которой не так просто добиться.
- Трудно предугадать затраты времени на проект, т.к. в начале никто не знает полного списка требований.
- Успех XP сильно зависит от уровня программистов, методология работает только с senior специалистами.

- Менеджмент негативно относится к парному программированию, не понимая, почему он должен оплачивать двух программистов вместо одного.
- Регулярные встречи с программистами дорого обходятся заказчикам.
- Из-за недостатка структуры и документации не подходит для крупных проектов.

Область применения eXP

- Бизнес-культура;
- Обычный стиль работы разработчиков, настроенный на тщательное планирование;
- Крупномасштабные проекты, требующие большой команды программистов;
- Рабочая среда, препятствующая легкости обратной связи.

Процесс ЭП является неформальным, но требует высокого уровня самодисциплины, в связи с чем требуется высокая квалификация всех специалистов. Но, в свою очередь, связанность методик, рассмотренная ранее, способна ввести процесс разработки в интеллектуальный резонанс, заметно повысив качество продукта и ускорив его разработку в разы. А также достигаются прогнозируемость и сведение к минимуму затрат на разработку. Довольный заказчик получает продукт, который он желает получить на момент выпуска. А общение и обучение разработчиков происходит без отрыва от производства, что повышает настроение в коллективе и подтягивает общий профессиональный уровень команды.

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.

Подход DSDM к управлению проектом фиксирует время, стоимость и качество, в то время как непредвиденные обстоятельства управляются путем изменения функций (требований), которые должны быть доставлены. По мере возникновения непредвиденных обстоятельств требования с более низким приоритетом отменяются или откладываются при полном согласии заинтересованных сторон бизнеса в соответствии с приоритетами MoSCoW.

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.

Принципы DSDM

Базовые принципы, на которых строится DSDM:

- активное взаимодействие с пользователями;
- частые выпуски версий;
- самостоятельность разработчиков в принятии решений;
- тестирование в течение всего цикла работ.

Как и большинство других гибких методологий разработки приложений, DSDM использует короткие итерации, продолжительностью от двух до шести недель каждая.

Особый упор делается на высоком качестве работы и адаптируемости к изменениям в требованиях.

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.

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.

Жизненный цикл проекта

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

- предпроектная стадия;
- стадия проекта;
- постпроектная стадия.

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.

Основные роли в методологии DSDM

В среду DSDM введены некоторые роли. Важно, чтобы участники проекта были назначены на разные роли до того, как они приступят к проекту. Каждая роль несет свою ответственность.

- Исполнительный спонсор/продюсер – очень важная роль. У него есть возможность и обязанность распоряжаться фондами и ресурсами. У этой роли также есть полное право принимать решения.
- Провидец – это тот, кто запускает проект в работу и находит первые основные требования. У провидца самое точное понимание коммерческих целей системы и проекта.
- Представительный пользователь – представляет пользователей в проекте. Отвечает за то, чтобы разработчики получали достаточное число отзывов пользователей во время процесса разработки.

- Пользователь-консультант – может быть любой пользователь, который представляет важную точку зрения на проект и привносит в проект знание по некоторой стороне использования продукта.
- Менеджер проекта – может быть из сообщества пользователей или из области информационных технологий. Обеспечивает общее руководство проектом.
- Технический координатор – ответственный за разработку архитектуры системы и контролирует качество проекта.
- Лидер команды – возглавляет команду разработки и обеспечивает её эффективную работу.
- Разработчик – анализирует требования к системе и моделирует её. Это подразумевает написание кода и создание прототипов.
- Тестировщик – проверяет исправность проекта с технической стороны, проводя тесты. Составляет комментарии и документацию.
- Секретарь – отвечает за сбор и запись требований, соглашений и решений, принятых в каждой рабочей группе.
- Посредник – управляет рабочими группами.
- Другие роли: бизнес-архитектор, менеджер по качеству, специалист по системной интеграции и т.д.

Область применения 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.

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).

МЕТОДОЛОГИЯ SCRUM

Scrum (от англ. *scrum* – «схватка», термин пришел из игры в регби) – методология управления проектами, активно применяющаяся при разработке информационных систем для гибкой разработки программного обеспечения. Scrum делает акцент на качественном контроле процесса разработки. Т.е. использование этой методологии предполагает в первую очередь довольно жесткие требования к организации самого процесса разработки, имитирующего интенсивный, целеустремленный, спортивный подход – игру. Основной упор делается именно на профессиональные и личные качества членов команды, способных ответственно, четко и быстро выполнять задачи и развиваться профессионально. Подход впервые описали Хиротака Такэути и Икудзиро Нонака в 1986 году.

Принципы SCRUM

В основе управления процессами в Scrum заложены три главных принципа:

1. *прозрачность,*
2. *инспекция,*
3. *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.

Основой Scrum является спринт, в течение которого выполняется работа над продуктом. По окончании спринта должна быть получена новая рабочая версия продукта. Спринт всегда ограничен по времени (1-4 недели) и имеет одинаковую продолжительность на протяжении всей жизни продукта.

Перед началом каждого спринта производится планирование спринта (Sprint Planning), на котором производится оценка содержимого журнала продукта (Product Backlog) и формирование журнала спринта (Sprint Backlog), который содержит задачи (Story, Bugs, Tasks). Указанные задачи должны быть выполнены в текущем спринте. При этом каждый спринт

должен иметь цель, которая является мотивирующим фактором и достигается с помощью выполнения задач из Sprint Backlog.

Каждый день производится ежедневный скрам-митинг (Daily Scrum Meeting) , на котором каждый член команды отвечает на вопросы «что я сделал вчера?», «что я планирую сделать сегодня?», «какие препятствия в своей работе я встретил?». Задача Daily Scrum Meeting – определение статуса и прогресса работы над спринтом, раннее обнаружение возникших препятствий, выработка решений по изменению стратегии, необходимых для достижения целей спринта.

По окончании спринта производятся его обзор (Sprint Review) и ретроспектива (Sprint Retrospective Meeting), задача которых оценить эффективность (производительность) команды в прошедшем спринте, спрогнозировать ожидаемую эффективность (производительность) в следующем спринте, выявлении имеющихся проблем, оценки вероятности завершения всех необходимых работ по продукту и другое.

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;
- *development Team* (дэвэлопмэнт тим) либо *скрам-команда (Scrum Team)* – команда разработки, кросс-функциональная команда разработчиков проекта, состоящая из специалистов разных профилей: программистов, тестировщиков, аналитиков, архитекторов и т.д. Размер команды составляет от 5 до 9 человек (5 оптимально). Команда является единственным полностью вовлеченным участником разработки и отвечает за результат как единое целое. Данная рабочая единица является самодостаточной,

самоуправляемой и самоорганизующейся, неким единым организмом, состоящим из отдельных элементов.

- *Stakeholders (стэкхолдэрс)* – дословно акционеры, лица, которые инициируют проект (бизнес-заказчики), которым скрам-проект будет приносить выгоду. Они вовлечены в скрам только во время обзорного совещания по спринту (Sprint Review).
- *User* – пользователь продукта.

***Артефакты методологии* SCRUM**

Спринт (Sprint) – 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 (Product Backlog) – 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 (Sprint Backlog) – 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.

Epic (эпик) – одна из нескольких глобальных функций продукта. В эпике могут содержаться User Story, например, пакет пожеланий одного пользователя или список задач (Task) для реализации Эпика.

User Story (юзер стори) – или Story, сюжет, в котором содержится пожелания пользователя.

Task (таск) – задача, фрагмент, который необходимо выполнить для реализации цели проекта.

Диаграмма сгорания задач (Burnup Chart) – диаграмма, показывающая количество сделанной и оставшейся работы. Обновляется ежедневно, 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.

Scrum Poker (скрам покэ) – быстрый и точный способ сбора оценок при помощи колоды карт с числами Фибоначчи (1, 2, 3, 5, 8, 13). Можно использовать мобильные приложения для Scrum Poker. Задачи с оценкой 13 необходимо дробить на более мелкие.

Story Points (стори поинтс) – единица оценки сложности выполнения задачи. Story Points имеет смысл применять, если проект состоит из 3-х и более спринтов, так как у команды накапливается статистика и опыт оценивания задач. На проекте из одного-двух спринтов использовать Story Points нет смысла, если только не для получения практики.

Daily Scrum Meeting (дэйли скрам митин) – ежедневное собрание не более 15 минут, проводимое в одно и то же время. Участвует скрам тим, наблюдать могут все. Проводит скрам-мастер. Цель митинга – оперативный обмен информацией, все в курсе происходящего, нет коммуникационных

разрывов. Задаются три вопроса: что сделал вчера? что будешь делать сегодня? какие препятствия встают на пути к цели?

Sprint Review (спринт ревью) – обзор спринта, участвуют все, встреча открытая. Команда рассказывает, что было сделано, и демонстрирует те части проекта, которые окончательно готовы.

Sprint Retrospective Meeting (спринт рэтроспэктив митин) – ретроспектива, участвует скрам команда. Собрание за «круглым» столом. Обсуждаются вопросы: что прошло хорошо, а что плохо? что можно было сделать лучше? Главное, никого не обличать! Рассматривается рабочий процесс. Цель – совершенствование рабочего процесса, стать «супер» командой.

И другие.

Преимущества методологии SCRUM

- ✓ ориентация на клиента, адаптивность: Scrum дает клиенту возможность делать изменения в требованиях в любой момент времени (но не гарантирует того, что эти изменения будут выполнены);
- ✓ сравнительная простота в изучении, которая позволяет экономить время, за счет исключения не критичных активностей;
- ✓ возможность получить потенциально рабочий продукт в конце каждого спринта;
- ✓ прекрасно подходит для малых компаний и стартапов, так как избавляет от необходимости от найма или обучения специализированного персонала руководителей.

Недостатки методологии SCRUM

- ✓ Использование в работе команды небольшого количества довольно жестких правил вступает в конфликт с идеей клиентоориентированности, т. к. клиенту не важны внутренние правила команды разработки, особенно если они ограничивают клиента;

- ✓ в Scrum не принято создание плана коммуникаций и реагирования на риски;
- ✓ упор на самоорганизующуюся, многофункциональную команду, что приводит к повышению затрат на отбор персонала, его мотивацию, обучение. При определенных условиях рынка труда, формирование полноценной, эффективной Scrum команды может быть невозможным.

Область применения SCRUM

Согласно авторам Scrum, эмпирический опыт является главным источником достоверной информации. Необходимость полного и точного выполнения Scrum указана в The Scrum Guide и обусловлена нетипичной организацией процесса, отсутствием формального лидера и руководителя. Одним из основных принципов Scrum являются самоорганизующиеся, многофункциональные команды. Однако, лишь небольшая часть сотрудников способно эффективно работать в Scrum без существенных изменений в ролях Scrum master и Product Owner, что противоречит идеологии Scrum, и потенциально приводит к неверному или неполному использованию Scrum.

Идеология Scrum утверждает, что заранее невозможно предусмотреть все изменения, таким образом, нет смысла заранее планировать весь проект, ограничившись только just-in-time планированием, т.е. планировать только ту работу, которая должна быть выполнена в текущем Sprint.

KANBAN

Kanban – японский термин, который начали использовать применительно к производству в 60-х годах 20-го века в компании Toyota. В основу данного принципа положен конвейерный метод производства, а также различные скорости выполнения отдельных технологических операций на производстве. Для понимания можно представить реальное предприятие, на котором есть основное производство («главный конвейер») и дополнительное производство («дополнительные конвейеры»). Темп выпуска

конечных изделий задает главный конвейер, при этом дополнительные конвейеры не могут ускорить темп выпуска изделия, однако могут замедлить его, опаздывая с выпуском требуемых деталей. Дополнительно при производстве может произойти смена приоритетов. К примеру, выяснилось что станция, которая производила левые зеркала, произвела 20 шт., а станция, производившая правые зеркала, — 10 шт., в то время как на конвейере находятся 15 автомобилей и необходимо 15 штук зеркал обоих типов. Налицо конфликт метрики – количественно производство не упало (дополнительные конвейеры выпустили 30 изделий в срок), но производство все равно рискует остановиться. Kanban призван помочь с этой проблемой.

Принципы Kanban

Идея заключается в том, чтобы максимально визуализировать процесс разработки решения, его отдельных задач. А также минимизировать перепроизводство путем создания баланса между потоком заданий и работой в процессе.

В упрощенном варианте Kanban включает в себя два простых правила:

- производственная станция имеет план производства деталей («backlog»). План отсортирован по приоритету и может меняться в любой момент;
- количество задач, выполняемых на станции одновременно, ограничено, что необходимо для управления скоростью производства на станции, а также скоростью реагирования на изменения плана.

Данная методология предполагает высокую самоорганизованность и дисциплину команды разработчиков и предъявляет к ней следующие требования:

- максимальная визуализация процесса (используется доска, которая разбивается на столбцы, в соответствии с этапами разработки; в каждом столбце помещаются задачи, распределенные с учетом приоритета);

- минимизация незавершенности (в каждом столбце указывается максимальное количество задач, между которыми команда может переключаться при возникновении проблем с текущей задачей);
- оптимизация процесса (отслеживание среднего времени выполнения задачи и уменьшение его).

Артефакты методологии Kanban

Доска Kanban

Для работы в данной методологии необходима доска, на которой будет вестись учет заданий. Вся доска делится минимум на 3 области:

- бэклог (все задачи),
- задачи в процессе,
- готовые задачи.

Процесс разработки разбивается на максимальное количество мелких задач, которые помещаются в бэклог. Из бэклога и других секций задания могут перемещаться по потоку только вправо, что означает выполнение задач. На области исполняемых в данный момент задач стоит ограничение. Таким образом сокращаются возможные простои задачи и реализуется одна из главных мыслей системы Kanban: меньше незавершённых задач – выше эффективность команды.

Считается, что если необходимо изменить требования или если происходит несостыковка по времени, то лучше все равно доделать задачу, а изменения вносить между выполнением задач.

Лидер

В Kanban нет лидера команды. Ответственность распределяется равномерно на всех членов команды. Если случились непредвиденные обстоятельства, то нет одного виноватого, взаимовыручка – один из главных столпов методологии.

Область применения KANBAN

В последнее время Kanban набирает большую популярность в производстве программного обеспечения. Однако чистый Kanban плохо

работает для продуктовых команд (читай – «основной конвейер»), но отлично работает с командами поддержки, такими как:

- группы поддержки программного обеспечения, где не важен «план», но важна скорость реагирования на изменения;
- группы тестирования, работающие отдельно от групп разработки;
- службы поддержки;
- другие примеры «неосновных производств».

Отдельно необходимо отметить, что Kanban хорошо работает в стартапах, не имеющих четкого плана, но активно работающих над разработкой. При грамотном руководстве Kanban обеспечивает максимально возможную для команды скорость работы, максимальную скорость реагирования на изменения и в то же время помогает сократить «расходы» на поддержку методологии.

К ограничениям Kanban'а при использовании его в продуктовых командах можно отнести:

- данная методология плохо работает с большими командами (больше 5 человек);
- в чистом виде Kanban плохо работает с кросс-функциональными командами. Т.е., в отличие от Scrum, тяжело совместить тестирование и разработку в одной команде. Более удачной мыслью является разбить процесс на «станцию» разработки и «станцию» тестирования с отдельными руководителями и backlog-ами;
- ввиду своей истории и специфики Kanban не предназначен для долгосрочного планирования.
- Kanban – не методология разработки, это метод управления процессами, реализующий принцип «точно в срок», может быть применен с совместно с любыми методологиями, а также в любых сферах, где важно планирование.

ЗАКЛЮЧЕНИЕ

Конечно, мы рассмотрели не все методологии проектирования и разработки программных продуктов, а только некоторую часть. Большое разнообразие методик разработки ИС позволяет подобрать наиболее подходящую из них для собственного проекта, используя в качестве критериев размер и сложность проекта, численность и квалификацию команды разработчиков, требования заказчика системы. Описанные выше методологии проектирования и разработки программных продуктов используются фирмами-разработчиками ПО для создания самых разных информационных систем во всех областях автоматизации человеческой деятельности. Особенности каждой из методологий, так же как и черты сходства, становятся ясными только в процессе практической деятельности по созданию ПП. Сравнение любых подходов по принципу «кто круче» не продуктивно и в корне не правильно. Каждая из методологий имеет свои плюсы, минусы и границы применения. И на практике команды, работающие в рамках единственного подхода, встречаются не так уж и часто. Единственное, что можно сказать определенно, это то, что Agile-методологии предъявляют серьезные требования к сработанности и опыту членов команды.

Библиографические источники

1. Буч Градди Объектно-ориентированный анализ и проектирование с примерами приложений, 3-е изд. / Буч Градди, Максимчук Роберт А., Энгл Майкл У., Янг Бобби Дж., Коналлен Джим, Хьюстон Келли А.: Пер с англ. — М.: ООО «И.Д. Вильямс», 2010. — 720 с.
2. Грекул В. И. Проектирование информационных систем / В. И. Грекул, Н. Г. Денищенко, Н. Л. Коровкина. – Интернет-университет информационных технологий – ИНТУИТ.ру, 2008. – 300 с.
3. Вендров А. М. Проектирование программного обеспечения экономических информационных систем. – М.: Финансы и статистика, 2006. – 544 с.
4. Уинстон Уолкер Ройс Managing the development of large software systems https://web.archive.org/web/20160318002949/
5. http://www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf
6. Солонин Е. Б. Современные методики разработки информационных систем. - Екатеринбург 2015
7. Якобсон А., Буч Г., Рамбо Дж. Унифицированный процесс разработки программного обеспечения. — СПб.: Питер, 2002. — 496 с: ил.
8. Экстремальное программирование (eXtreme Programming): https://www.youtube.com/watch?v=8pe3mVHfLZg
9. Экстремальное программирование: https://intuit.ru/studies/courses/64/64/lecture/1870?page=6
10. Extreme Programming (XP) - Georgia Tech - Software Development Process: https://www.youtube.com/watch?v=hbFOwqYIOcU
11. https://blog.calltouch.ru/chto-takoe-kanban-i-kak-vnedrit-metodologiyu-v-kompanii/
12. https://bootups.ru/2009/07/metodologiya-kanban-v-it-kanban-development/

13. https://scrumtrek.ru/blog/kanban/1360/chto-takoe-kanban-metod-maksimalno-korotko/
14. https://www.unisender.com/ru/support/about/glossary/kanban/#howto
15. https://ru.wikipedia.org/wiki/%D0%9A%D0%B0%D0%BD%D0%B1%D0%B0%D0%BD_(%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0)
16. https://blog.iteam.ru/chto-takoe-kanban-i-chem-on-polezen/#i-2
17. https://geekbrains.ru/posts/kanban_howto
18. https://netology.ru/blog/scrum-kanban
19. https://wm-help.net/lib/b/book/2291208051/118
20. https://ru.wikipedia.org/wiki/DSDM
21. http://www.k-press.ru/cs/2001/4/fauler/fauler.asp#N745
22. http://window.edu.ru/catalog/pdf2txt/373/60373/30320?p_page=12
23. Белые страницы MSF. http://www.microsoft.com/rus/msdn/msf
24. Ken Schwaber, Jeff Sutherland The Scrum Guide. The definitive Guide to Scrum: The Rules of the Game.
25. А. А. Трусь Психология управления, учебное пособие.
26. Jeff Sutherland, Nafis Ahmad How a Traditional Project Manager Transforms to Scrum: PMBOK vs. Scrum.
27. Пихлер Р. Agile Product Management With Scrum: Создание Продуктов, Которые Любят Клиенты/ Addison-Wesley, 2010 – 155с.
28. А. В. Рудаков Технология разработки программных продуктов. Учебник// Academia, 2013 – 208 стр.
29. Аллен Э. Типичные ошибки проектирования / Э.Аллен: Пер. с англ. – СПб.: Питер, 2003. – 224 стр.
30. Брукс Ф. Мифический человеко-месяц или как создаются программные системы / Ф. Брукс. — М.: СПб: Символ-Плюс, 2016. — 304 с.

Электронное учебное издание

Оксана Федоровна **Абрамова**

Индустриальная разработка программных продуктов
Часть 2

Учебное пособие

Электронное издание сетевого распространения

Редактор Матвеева Н.И.

Темплан 2021 г. Поз. № 6.
Подписано к использованию 14.05.2021. Формат 60x84 1/16.
Гарнитура Times. Усл. печ. л. 5,6.

Волгоградский государственный технический университет.
400005, г. Волгоград, пр. Ленина, 28, корп. 1.

ВПИ (филиал) ВолгГТУ.
404121, г. Волжский, ул. Энгельса, 42а.