ТЕХНОЛОГИЯ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

На сегодняшний день в мире существует огромное количество различных процессов для создания ПО. Тем не менее, именно технологий, рассматривающих полный жизненный цикл проекта разработки ПО, сочетающих в себе научный подход, серьезную базу исследований и имеющих историю реального использования и адаптации, относительно немного. Особое место в этом списке занимает технология компании Rational Software.

Д.Л. Безуглый

В современном, перегруженном информацией обществе сложно найти область деятельности человека, в которой бы не использовались средства вычислительной техники. За несколько десятилетий эволюции аппаратное обеспечение (hardware) достигло небывалого прогресса — та вычислительная мощь, которую десять-пятнадцать лет назад могли позволить себе приобрести лишь немногие научные учреждения, и обслуживание которой требовало целого штата специалистов, сегодня доступна практически каждому инженеру. Однако невозможно использовать эти вычислительные мощности без программного обеспечения (software). И именно в этой области, несмотря на значительный рост доступности аппаратных ресурсов, наблюдаются значительные проблемы.

Так, по данным американских исследователей, в 80-е годы только 14% проектов по созданию ПО завершались успешно[1]. Сегодня, после нескольких десятилетий эволюции языков программирования, инструментальных средств разработки, практически неограниченной доступности машинного времени (по сравнению с 70- и 80-ми годами) процент успешно завершенных проектов составляет всего 26%[2].

В СССР в производстве ПО результаты были значительно лучше и объективными предпосылками этого были следующие моменты:

- плановая организация оптимально сочеталась с каскадной моделью;
- контроль успешности проекта был ориентирован не на удовлетворение требований заказчика, а на удовлетворение изначально согласованного ТЗ;
- разработкой ПО занимались, как правило, исключительно высококвалифицированные специалисты специализированных институтов;
- в силу выполнения в основном ВПК-ориентированных проектов, бюджеты были фактически не ограничены (по сегодняшним меркам).

По ряду причин советская школа разработки ПО прекратила свое развитие, и многие достижения были утрачены. В рыночных условиях (быстро меняющиеся требования, ограниченные бюджеты, ориентация на результат, высокая конкуренция за высококвалифицированный персонал) использование старых наработок советской школы ограничено очень узкими областями.

В любом производстве используемая технология определяет лучшие достижимые показатели. В силу специфичности производства ПО (практически нулевая стоимость тиражирования, очень быстрый процесс устаревания и т. д.) технология его создания очень сильно завязана на человеческий ресурс и поэтому должна включать в себя организационный и управленческий аспекты.

На сегодняшний день в мире существует огромное количество различных процессов для создания ПО. Тем не менее, именно *технологий,* рассматривающих полный жизненный цикл проекта разработки ПО, сочетающих в себе научный подход, серьезную базу исследований и имеющих историю реального использования и адаптации, относительно немного. Из методологий и технологий, получивших определенное признание на данный момент, можно назвать следующие[3]: Datarun, CMM, Microsoft Solution Framework (MSF), Oracle Method, Rational Unified Process (RUP), SADT (IDEFx).

Особое место в этом списке занимает технология компании Rational Software. В методологии[4] применен наиболее современный *процессно-ориентированный* подход: так как разработка ПО является производством, то, как и на всяком производстве, при выявлении проблем в продукции (симптомов) необходимо в обязательном порядке корректировать процесс (устранять причины). Особенностью технологии является то, что в ее создании участвуют ведущие методисты в области

---

1 *Подразумевается не только удовлетворение требований заказчика, но и завершение в срок с соблюдением бюджета (Chaos Report, www.standishgroup.com).*
2 *Данные были приведены на одном из семинаров Г. Бучем.*
3 *В алфавитном порядке.*
4 *Методология является составной частью технологии Rational.*

разработки ПО, такие как Г. Буч (ООАП), Дж. Рамбо (OMT), А. Джекобсон (Objectory), внесшие весомый вклад в теорию и практику разработки современного ПО. Кроме того, эта технология развивалась и проходила проверку с участием военного ведомства США.

СИМПТОМЫ И ПРИЧИНЫ ПРОБЛЕМ

В процессе формирования методологии и ее совершенствования, Rational было проведено исследование процессов разработки ПО в более чем тысяче организаций. В результате исследования были определены наиболее часто встречающиеся симптомы и выявлены причины проблем, приводящих к срыву выполнения проекта (см. Табл. 1).

По мнению автора, отделение симптомов от причин проблем, возникающих при разработке ПО, очень существенно. Как показывает практика, перечисленные симптомы чаще трактуются как особенности, свойственные разработке ПО в целом, а не как следствие неадекватности процесса разработки, так как при устранении симптомов эффект может быть только временным. К примеру «позднее определение недостатков» — это только симптом более значительной проблемы, такой как «неадекватное тестирование». А «неадекватное тестирование» связано с использованием модели «Водопада», в которой решение этой проблемы структурно исключено (тестирование происходит на слишком поздней стадии, когда коррекция ошибок неосуществима без серьезного изменения сроков реализации).

Перечислим некоторые причины проблем:

- Недостаточное управление требованиями делает неконтролируемым изменение объемов работы и ставит под угрозу возможность реализации проекта в срок.
- Хрупкая архитектура, которая легко ломается под воздействием изменений в требованиях или технологиях. Одним из примеров является архитектура, которая строится на одноразовых компонентах в рамках технологии, не предусматривающей повторного использования, что делает саму архитектуру негибкой, так как существенное изменение программы можно рассматривать как сборку новой программы с повторным использованием ранее созданных компонентов.
- Субъективное определение статуса проекта известно, как правило «90% сделано, и еще 90% осталось». Другими словами, разработчики полагают, что сделано 90% работы, но ее завершение требует такого же количества ресурсов и времени для достижения 100% готовности.

Таблица 1. Проблемы при разработке ПО

Симптомы	Причины (Root Causes)
Неточное понимание потребностей конечного пользователя (Inaccurate understanding of end-user needs)	Недостаточное управление требованиями (Insufficient requirements management)
Неспособность иметь дело с изменяющимися требованиями (Inability to deal with changing requirements)	Неопределенные и неточные методы взаимодействия (Ambiguous and imprecise communications)
Модули, которые не соответствуют друг другу (Modules that don't fit together)	Хрупкая архитектура (Brittle architectures)
Программное обеспечение, с ограниченными возможностями поддержки и расширения (Software that's hard to maintain or extend)	Чрезмерная сложность (Overwhelming complexity)
Позднее определение серьезных недостатков в проекте (Late discovery of serious project flaws)	Невыявленные несоответствия между требованиями, дизайном и реализацией (Undetected inconsistencies among requirements, designs, and implementations)
Плохое качество программного обеспечения (Poor software quality)	Недостаточное тестирование (Insufficient testing)
Неприемлемая производительность ПО (Unacceptable software performance)	Субъективное определение статуса проекта (Subjective project status assessment)
Участники разработки не имеют возможности определить, кто и что изменил, когда, где и почему (Team members in each other's way, unable to reconstruct who changed what, when, where, why)	Замедленное управление рисками из-за водопадной разработки (Delayed risk reduction due to waterfall development)
Ненадежный процесс построения и выпуска (An untrustworthy build-and-release process)	Неконтролируемое распространение изменений (Uncontrolled change propagation)
	Недостаточная автоматизация (Insufficient automation)

ПРИЧИНЫ ПРОБЛЕМ И ЛУЧШИЕ ПРАКТИКИ

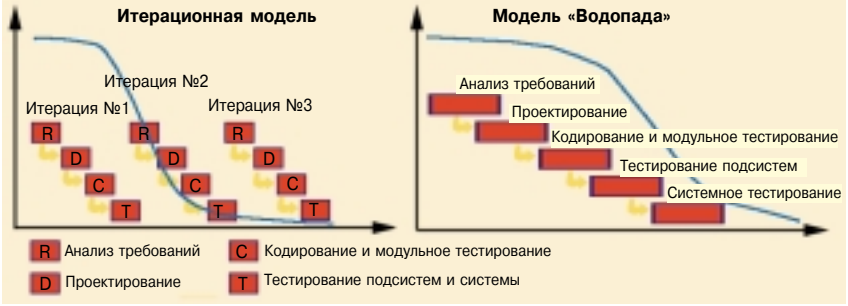
Как было указано ранее, только устранение причины может надолго устранить связанный с ней риск. И только устранив источники проблем, можно достичь стабильного улучшения ситуации в эффективности разработки ПО. В процессе исследования были также собраны и обобщены лучшие практики по устранению источников проблем в процессах разработки ПО (рис. 1).

Лучшие практики — это комплекс подходов к разработке ПО, проверенных на многих реальных проектах, которые при совместном использовании направлены на устранение причин возникновения проблем при разработке ПО. Интересен тот факт, что «лучшие практики» названы так не потому, что легко можно идентифицировать их вклад в успех проекта, а скорее по той причине, что они применяются в каждой организации достигшей стабильного успеха в области разработки ПО.

1 Лучшие практики

Итеративный процесс разработки (Develop Iteratively)			
Управление требованиями (Manage Requirements)	Компонентная архитектура (Use Component Architectures)	Визуальное моделирование (Model Visually)	Контроль качества (Verify Quality)
Управления изменениями (Control Changes)			

2 Жизненные циклы разработки



Практика № 1. Итерационный процесс разработки

Суть итеративной практики может быть лучше всего пояснена визуально (рис. 2). Как видно, в результате продукт проходит как бы несколько жизненных циклов разработки (обработки требований, разработки, кодирования и тестирования) с применением модели «Водопада». Синей линией показана величина риска провала проекта. Разбиение на подсистемы давно использовалось в процессе разработки для снижения сложности отдельно разрабатываемого модуля. Однако в данном случае суть состоит в том, чтобы не только разбить задачу структурно на модули, но и сам процесс проектирования во времени. При этом необходимо выделять работы, связанные с наиболее значительными рисками в начальные итерации, до того как в разработку вложены существенные средства.

Безусловно, с точки зрения управления такой подход значительно сложнее, но дает реальную основу для управления рисками и сложностью в проекте. Поэтапный подход в модели «Водопада» значительно проще с точки зрения формального выполнения функций управления проектом, но не обеспечивает достаточной управляемости проектом, а все проблемы, связанные с ошибками или недочетами начальных этапов, выясняются только в процессе интеграции.

Практика № 2. Управление требованиями

Управление требованиями — это систематический подход к извлечению, организации и документированию требований к системе, установлению и поддержке соглашения между пользователями/заказчиками с командой разработчиков в отношении изменений требований к системе.

Точная модель требований важна именно потому, что она соединяет вместе все остальные элементы проекта. Требования определяют что должно быть сделано, что будет проектироваться, и что должно быть протестировано. Изменения в требованиях, как правило, затрагивают практически все элементы, являющиеся результатами процесса разработки. Требования являются отправной точкой при планировании итерации, в том числе, для определения усилий, которые необходимо затратить для получения результата.

Говоря об управлении требованиями в подходе Rational нельзя не упомянуть о сценариях использования (Use Case). Сценарий является основным структурирующим элементом *текстового* описания проектируемой системы, организованного в соответствии с целями, преследуемыми пользователем при использовании этой системы.

Наличие задокументированных требований в проекте делает предметным общение внутри проекта; помогает преодолевать сложности с помощью расстановки приоритетов, назначения других атрибутов требований, фильтрации и отслеживания зависимостей между требованиями (traceability); позволяет осуществлять объективный контроль состояния проекта на основании реализованной функциональности.

Практика № 3. Использование компонентной архитектуры

Архитектура в ПО является одним из самых сложных понятий. Архитектура приложения совместно с требованиями представляет собой мостик между областью использования системы (конечного пользователя) и областью решения. Она определяет структуру, поведение и контекст системы. От нее зависят такие важные особенности системы, как удобство использования (usability), функциональность, повторное использование компонентов, сложность реализации отдельных модулей, гибкость, производительность и т. п.

Согласно авторам Software Architecture in Practice: «архитектура ПО — это такая составляющая результатов разработки, которая дает наивысшую отдачу от вложений».

Слово «компонент» используется в современном мире в различных контекстах. В данном случае подразумевается практически независимая и заменяемая часть системы, которая соответствует конкретной функции в контексте хорошо определенной архитектуры. Существенной особенностью данного подхода является обязательность использования и соответствия документации интерфейсов при взаимодействии как внутри, так и извне системы. В результате компоненты становятся элементами повторного использования и внутренней эволюции, что делает их естественным объектом для управления конфигурацией ПО.

Наиболее распространенными платформами для построения компонентной архитектуры являются: Microsoft COM, Sun Enterprise Java Beans, CORBA.

Практика № 4. Визуальное моделирование

Цель визуального моделирования — предоставить наиболее эффективное средство коммуникации для отображения структуры и поведения архитектуры и компонентов; отображения соотношения между элементами системы; сокрытия (или наоборот, раскрытия) деталей реализации — в зависимости от задачи, стоящей перед разработчиком.

Использование стандартного языка моделирования (UML) дает возможность всем членам команды использовать в процессе обмена

информацией сложные и, тем не менее, достаточно точные и краткие элементы визуального моделирования. В методологии Rational UML используется для специфицирования, визуализации, конструирования и документирования всех артефактов[5] процесса разработки ПО.

Совместно с итерационным подходом визуальное моделирование позволяет эффективно отслеживать и распространять внутри коллектива разработчиков «межитерационные» изменения в проектируемом приложении. Многие коллективы пытались осуществить подобное и ранее, но объемы работы по отображению в моделях неизбежных изменений в архитектуре и дизайне (вносимых на этапах реализации и тестирования) оказывались слишком велики. Только использование инструмента, осуществляющего прямое и обратное преобразование *модель<->код*, позволяет снизить объем трудозатрат при этих операциях.

Использование диаграмм прецедентов (use-cases) позволяет в наглядной форме отобразить требования к поведению системы в терминах актеров и целей. В процессе моделирования могут быть легко выявлены такие недостатки архитектуры системы, как отсутствие модульности и гибкости.

Практика № 5. Контроль качества

Качество — достаточно сложный термин и встречается во многих областях деятельности человека. В терминологии Rational качество (quality) определяется как «характеристика демонстрируемого достижения производимого продукта, которая удовлетворяет или превышает требования, согласно ранее установленным параметрам и критериям, и произведенным по согласованному ранее процессу». Согласно этому определению, качество — это не только соответствие требованиям, оно также должно включать параметры и критерии их оценки — для демонстрации достижения необходимого качества, а также реализацию процесса — для осуществления контроля того факта, что результирующий продукт соответствует необходимой степени качества (и является повторяемым и управляемым).

Во многих организациях тестирование отнимает 30–50% ресурсов на разработку. Тем не менее, количество ошибок в ПО, выявляемых заказчиком, свидетельствует о том, что ПО недостаточно тестируется до его поставки.

Многие считают, что непрерывное тестирование является идеалом, вспоминая золотую эру больших ЭВМ с пакетным тестированием, но до тех пор, пока адекватный современным задачам инструмент автоматического тестирования не будет доступен, это не осуществимо. Непрерывное тестирование становится осуществимым только с уменьшением времени и трудозатрат с помощью автоматизации.

Данные, полученные Rational в результате автоматизации процесса тестирования у реального заказчика, следующие: до автоматизации выполнялось 13 000 тестов с помощью шести человек за две недели, после автоматизации были выполнены те же тесты за 6 часов с помощью одного человека.

Верификация функциональности системы в технологии Rational тесно интегрирована с использованием сценариев. Каждый сценарий использования является отражением какого-либо аспекта функционирования системы и является идеальной основой для тестирования.

Таким образом, автоматизированное непрерывное тестирование[6] предоставляет объективную оценку степени выполнения проекта; позволяет находить ошибки и неточности по мере их возникновения; может быть сфокусировано на областях высокого риска; дефекты, обнаруженные на ранней стадии, значительно дешевле для исправления. Кроме того, автоматизированный инструментарий позволяет тестировать не только

[5] *Артефакт — какой-либо результат действия (документ, модель, исходные коды).*
[6] *Совместно с управлением требованиями.*

Таблица 2. Лучшие практики и причины

Причины проблем возникающих при разработке ПО	Итеративный подход	Управление требованиями	Использование компонентной архитектуры	Визуальное моделирование	Контроль качества	Контролирование изменений
Недостаточное управление требованиями	+	+		+		+
Неопределенные и неточные методы взаимодействия	+	+		+		+
Хрупкая архитектура			+	+		
Непреодолимая сложность	+	+	+	+		+
Необнаруженные несоответствия между требованиями, дизайном и реализацией	+	+			+	+
Недостаточное тестирование	+	+		+	+	+
Субъективное определение статуса проекта	+	+		+	+	
Замедленное управление рисками вследствие «водопадной» разработки	+					
Неконтролируемое распространение изменений			+			+
Недостаточная автоматизация		+	+	+	+	+

функциональность, но также надежность и производительность, практически нереализуемые для ручных методов тестирования.

Практика № 6. Управление изменениями

Одним из ключевых и сложных понятий в области разработки ПО является парадигма «Управления изменениями». Rational выделяет следующие основные проблемы:

- одновременное изменение (когда два или более процессов изменяют один тот же артефакт – последний, вносящий изменение, разрушает изменения другого);
- ограниченное уведомление (недостаточная гибкость уведомления о произошедших изменениях);
- множество версий (при разработке достаточно часто существует множество версий одного и того же артефакта с разной степенью готовности).

Можно выделить следующие основные компоненты управления изменениями.

Управление запросами на изменение (Change Request Management) соответствует инфраструктуре, необходимой для контроля над влиянием на стоимость и сроки от запрошенных изменений для существующего продукта.

Управление конфигурацией (Configuration Management) представляет собой деятельность по определению конфигураций, построению, маркировке и хранению версий артефактов.

Измерение (Measurement) описывает состояние продукта, опираясь на типы, количество и значительность обнаруженных недостатков на протяжении разработки.

Безусловно, столь краткое рассмотрение не может осветить все аспекты применения лучших практик и показать все взаимосвязи между ними. Как можно увидеть из таблицы 2, для устранения одной причины проблемы задействуется несколько практик. Лучшие практики формируют основу для по-дисциплинного рассмотрения технологии разработки ПО, представленного в Rational Unified Process.

Таблица 3. Стадии жизненного цикла системы

Начальная стадия (Inception)	Определение границ проекта. Осуществляется при помощи определения всех заинтересованных лиц и пользователей, а также всех сценариев использования, с черновым описанием наиболее важных. На этой стадии разрабатывается бизнес-план для определения ресурсов, которые будут привлечены в проект
Стадия Разработки (Elaboration)	Планирование, определение возможностей и архитектуры. Две основные задачи рассматриваются на этой стадии: определение требований (около 90%) и установление архитектуры системы. Могут быть установлены конкретные ожидания (цена/ресурсы) от реализации проекта
Стадия Конструирования (Construction)	Построение продукта за несколько итераций вплоть до получения бета-версии продукта
Стадия перехода (Transition)	Передача продукта конечному пользователю. На этой стадии основной упор делается на обучение пользователей, инсталляцию и поддержку

ПРОЦЕСС РАЗРАБОТКИ

Любой практик, прочитав первую часть статьи, скажет: «Теория это хорошо, но от теории к практике еще ой как далеко. Необходимо развивать и внедрять процессы, связанные с перечисленными практиками, разрабатывать поддерживающие элементы и т. д. И это все огромная работа, которую тоже необходимо делать...».

Безусловно, технология не состоит только из теоретических исследований.

Для успешного применения методологии необходимо установление адекватного процесса разработки. И именно развернутый шаблон процесса вместе с достаточным для его реализации комплексом инструментария предлагает Rational. Описанные шесть лучших практик легли в основу Rational Unified Process (RUP).

Естественно, что шаблон имеет определенную направленность. В силу упомянутых связей с военными заказами, RUP наиболее близок к системе ценностей советской научной школы, так как он в первую очередь ориентирован на разработку уникальных систем высокой сложности. Но как показала мировая практика применения, уменьшение сроков разработки, повышение качества получаемых продуктов с увеличением удельного соотношения функциональность/стоимость приводят к коммер-

ческой рентабельности внедрения и использования этого процесса.

ОПИСАНИЕ RUP

Сам процесс поставляется Rational в виде гипертекстовой базы знаний с качественно выполненным классификатором и средством поиска.

На высоком уровне абстракции RUP содержит три аспекта: фазы, дисциплины, и лучшие практики.

Первый аспект представляет собой динамический вид системы и описывает процесс с точки зрения фаз, итераций и вех разработки, второй аспект представляет собой статический вид процесса и описывает процесс с точки зрения ролей, артефактов и дисциплин, третий аспект представляет собой косой срез концепций, которые легли в основу процесса (лучших практик).

При более близком рассмотрении каждый аспект, будь то фаза или дисциплина, имеет полное детальное описание с точки зрения концепций, необходимых артефактов и руководств по их созданию.

Кроме того, в нем содержатся дополнительные элементы, такие как рекомендации по использованию инструментария, глоссарий, список внешних источников (практически все серьезные работы в области инженерии ПО), примеры и рекомендации по конфигурированию процесса в организации для различных типов деятельности.

Фазы

Жизненный цикл системы разбивается на циклы, каждый из которых представляет собой новую версию системы, устанавливаемую заказчику. RUP определяет четыре стадии отдельного цикла (см. Табл. 3).

Кого-то может удивить отсутствие стадии поддержки системы, однако это естественно, так как по сути она совпадает со стадией Transition.

В зависимости от типа и сложности продукта стадии могут отличаться по времени и числу итераций.

Модели

Они играют существенную роль в RUP, практически в каждой дисциплине рассматривается несколько основных для нее моделей. Перечислим их:

- бизнес-модель — модель бизнес-процессов и бизнес-окружения, она может быть использована для определения требований для поддерживающей информационной системы;
- модель сценариев — отображает то, что должна делать система и ее окружение;
- дизайн-модель — модель объектов, описывающая реализацию сценариев использования системы. Она может быть рассмотрена как абстракция модели реализации;
- модель реализации — коллекция компонентов и реализуемых подсистем, которые их содержат;
- модель тестов — содержит все тестовые варианты и процедуры, необходимые для тестирования системы.

В подходе Rational для отображения моделей используется Unified Modeling Language (UML). С точки зрения процесса UML представляет собой стандарт описания для артефактов разработки и моделирования (семантических моделей, синтаксических нотаций и диаграмм), то есть той информации, которая необходима для обеспечения взаимодействия в процессе разработки и поддержки программного обеспечения. UML *не определяет*, каким образом использовать его возможности для получения результата, какие модели должны быть построены и в какой последовательности.

Дисциплины

Бизнес-моделирование. В ходе бизнес-моделирования выясняется структура и динамика организации заказчика, предпринимаются определенные шаги для определения целей и возможностей усовершенствования бизнес-процессов с использованием информационных технологий, устанавливается взаимопонимание относительно решаемой задачи между разработчиком и заказчиком.

Требования. Основной задачей, решаемой в этой дисциплине является установление соглашения с пользователями и другими лицами, заинтересованными в проекте, относительно того, что система должна делать, определяются границы и ограничения, накладываемые на разработку системы и саму систему. При

решении этой задачи обеспечивается базис для планирования технического содержания итераций расчета затрат (средств и времени), лучшего понимания разработчиками требований к системе. Кроме того, может быть осуществлено определение пользовательского интерфейса системы, с фиксацией внимания на потребностях и целях пользователей.

Анализ и дизайн. В рамках этой дисциплины производится трансформирование требований в дизайн будущей системы, разрабатывается архитектура и адаптируется дизайн к используемой среде разработки.

Реализация. В рамках этой дисциплины производится определение организации кода в терминах реализуемых подсистем, организованных в слои (layers), реализуются классы и объекты, производится объединение разработанных компонентов в рабочие элементы.

Тестирование. В рамках тестирования рассматривается верификация взаимодействия объектов, проверяется корректная работа интегрируемых компонентов, контроль выполнения всех требований к системе, выявление и точная идентификация дефектов, касающихся разработки ПО.

Поставка (Deployment). В рамках этой дисциплины объединены все действия, связанные с передачей системы пользователям.

Управление конфигурациями и изменениями. Эта дисциплина контролирует изменение и целостность всех артефактов проекта. Для этой цели осуществляется идентификация элементов конфигурации, осуществляется ограничение и аудирование изменений для этих элементов, определяются и управляются конфигурации этих элементов.

Управление проектом. Дисциплина управления проектом в программной инженерии — это искусство балансирования конкурирующих целей, управления рисками и преодоления препятствий для успешного завершения продукта, удовлетворяющего требованиям как заказчиков (оплачивающих поставку), так и пользователей продукта.

Для решения этой задачи в RUP представлены готовые к использованию шаблоны основных документов (MS Project, MS Word) для управления проектом, а также комплекс руководств для планирования, кадрового обеспечения, выполнения и контролирования проекта.

Среда (Environment). Эта дисциплина фиксирует внимание на действиях, необходимых для построения процесса разработки в рамках проекта и организации, с учетом таких параметров среды, как процессы и инструменты. Именно с изучения этой дисциплины необходимо начинать планирование внедрения тех или иных элементов RUP. Кроме того, в последней версии RUP появился отдельный комплекс руководств для построения индивидуальных процессов разработки ПО, основанных на RUP.

Люди

Несмотря на полноту и инструментальную поддержку RUP, для достижения успеха внедрения этой технологии необходимы люди, которые смогут правильно применить и использовать эти инструменты и знания. Для достижения этой цели доступны следующие возможности:

- бесплатное ознакомление с пробными версиями продуктов;
- большое количество книг, посвященных использованию и внедрению процесса;
- сообщество разработчиков Rational Developer Network (для зарегистрированных пользователей);
- обучение персонала;
- институт консультантов (менторов), предоставляющих свои услуги через сеть партнеров.

Говоря о людях, нельзя не упомянуть, что предлагаемая линейка продуктов предназначена и оптимизирована не только для использования на уровне организации и команды разработчиков, но и для отдельно взятого разработчика.

В заключение следует отметить, что RUP не является панацеей, однако, по личному мнению автора (и мнению ведущих методистов, пытающихся заглянуть еще глубже в основы программной инженерии), это лучший старт, который может сделать организация на пути к успешному процессно-ориентированному ведению проектов по разработке ПО.

Безуглый Дмитрий Леонидович — Аналитик, АО «Банкомсвязь».