# **Методологии и принципы моделирования жизненного цикла IT-проекта**

Перед началом любой работы каждый человек пытается максимально подробно разобраться в деталях, мелочах, с которыми ему предстоит столкнуться, разобраться в том, с чем придется иметь дело, изучить способы и стратегии исследования предмета деятельности. Этим и занимается такое учение, как методология. И как работа в данном случае рассматривается жизненный цикл IT-проекта.

При работе с IT-проектами использование специализированных методов управления проектами будет очень полезным для непрерывного прогресса, приводящего к большей успешности в результате. Планирование и выполнение IT-проектов может быть трудным по разным причинам, что делает умение успешно их завершать очень ценным качеством для любого работодателя. В наше время в интернете можно получить достаточно информации по каждой из методологий, но для особо любознательных IT-компании проводят специальные тренинги по изучению различных методологий жизненного цикла. Так, например, компания "Проектный офис" часто проводит тренинги по Agile, Scrum в Минске продолжительностью 2-3 дня.

Как известно, нет единого способа решения различных проблем. И наш проект здесь не будет исключением. Эффективность каждого из проектов достигается не всегда использованием одной и той же методологии. Здесь и возникает проблема в ее выборе.

Выбор определенной модели жизненного цикла зависит, в основном, от содержания и целей проекта, а также от размера его финансирования. Рассмотрим некоторые из них.

## *Waterfall*

Одна из самых "старых", и уже традиционных, и, как следствие, популярных моделей, которую сейчас принято считать каскадной. Была упомянута в статье американского информатика Уинстона У. Ройса в 1970 году. Модель "водопада" определена как последовательная модель разработки с явно выраженным конечным результатом для каждой фазы. Стандартные фазы "водопада" представляют собой:

1) Requirements (Определение требований)

2) Design (Проектирование)

3) Development (Разработка)

4) Testing (Тестирование и отладка)

5) Maintenance (Инсталляция/ поддержка)

Переход от одной фазы к другой происходит только после полного и успешного завершения предыдущей. Переходов назад либо вперёд или перекрытия фаз — не происходит.

Преимущества:

* Такая жёсткая последовательность делает процесс разработки чрезвычайно прозрачным, а, следовательно, максимально удобным для заказчика (благодаря последовательному переходу от этапа к этапу управление масштабными проектами осуществляется гораздо проще, а команда, в свою очередь, работает слаженнее);
* Жёсткая последовательность позволяет дать точную оценку стоимости разработки и ее сроков, что позволяет точно спрогнозировать эффект, полученный от запуска приложения;
* Удобство управления проектом и возможностью полного контроля над каждым этапом его создания;
* Эта модель представляет базовый подход, который может применяться в любом проекте (универсальная модель; Другие модели жизненного цикла могут оказаться более результативными и эффективными в зависимости от характеристик проекта. Например, если устанавливается пакет программного обеспечения, то пропускаются фазы проектирования и реализации.).

(Классический «водопадный» подход – это модель жизненного цикла, которую любой обыватель может применить, практически ничего не зная о методологии и планируя проект «с чистого листа».

Что может быть проще? Даже если у обычного человека очень маленький проект, всё равно он проходит эти базовые шаги, хотя бы даже проделывая некоторые из них в голове. К примеру, если у него 40-часовой (на одну рабочую неделю) проект разработки или улучшения документа, может показаться что он сразу же бросается в фазу "Разработка". Но так ли это? Наиболее вероятно, что он получил какого-либо рода поручение с требованиями или пожеланиями, которые придется осмыслить (Определение требований) и трансформировать в замысел будущего содержания (Проектирование). Затем он воплощает замысел (Разработка), проверяет результат (Тестирование) и передает для использования (Инсталляция).)

Недостатки:

* При необходимости внесения поправок в документацию разработка продукта останавливается вплоть до момента повторного согласования документов (следовательно, при недостаточном уровне проработки требований существует риск увеличить сроки разработки до абсолютно неприемлемых величин);
* Накопление возможных на ранних этапах ошибок к моменту окончания проекта и, как следствие, возрастание риска провала проекта, увеличение стоимости проекта.

Забавно, что сам Ройс использовал итеративную модель разработки и даже не использовал термин "водопад". Оказалось, что он предусматривал обратные связи между этапами на одном уровне. И в отличие от привычного метода, здесь речь уже шла о параллельных работах по двум последовательным этапам, что дает возможность на более ранних стадиях выявить ошибки предыдущего этапа жизненного цикла и решить один из весомых недостатков классического "водопада" — невозможность возврата на предыдущий этап. К примеру, если рассмотреть параллельные работы по двум последовательным фазам - Coding и Testing, становится очевидным, что часть программы тестируется, в то время как другая часть все еще находится на стадии разработки. То есть получается, что речь действительно идёт об итеративной методологии. В таком случае, остается вопросом, почему методология в широких кругах разработчиков и тестировщиков воспринимается ошибочно.

Водопадный подход хорошо работает в проектах, где требования к программному продукту чётко определены и не должны меняться, вовлечение заказчика в процесс разработки не требуется. Каскадная модель подходит для разработки сложных и больших проектов и систем со строго определённой функциональностью (например, разработка транзакционной системы, где есть чётко определённый результат). Также удобно использовать её при разработке больших государственных заказов или научных разработках. А вот для разработки бизнес или веб-приложений крайне нежелательно, так это относительно креативный проект, ход работы которого может часто меняться (например, по просьбе заказчика), в то время как в "водопадной" методологии всё чётко и последовательно.

## *Iterative, Incremental*

Итеративный подход (англ. iteration — повторение) — выполнение работ параллельно с непрерывным анализом полученных результатов и корректировкой предыдущих этапов работы. Проект при этом подходе в каждой фазе развития проходит повторяющийся цикл: Планирование — Реализация — Проверка — Оценка (англ. plan-do-check-act cycle). Этот метод является альтернативой последовательной модели.

Модель предполагает разбиение жизненного цикла проекта на последовательность итераций, каждая из которых напоминает "мини-проект", представляющий собой небольшие фрагменты функциональности, по сравнению с проектом в целом. Цель каждой итерации заключается в том, чтобы, постепенно добавляя больше опций, получить работающую версию программной системы, включающей функциональность, основанную на содержании всех предыдущих и текущей итерации. Результат финальной итерации содержит всю требуемую функциональность продукта. Таким образом, с завершением каждой итерации продукт получает приращение — инкремент — к его возможностям, которые, следовательно, развиваются эволюционно. По сравнению с водопадом, итеративный метод является более гибким.

Основные преимущества итеративного подхода:

* Снижение воздействия серьёзных рисков на ранних стадиях проекта, что ведет к минимизации затрат на их устранение;
* Акцент усилий на наиболее важные и критичные направления проекта;
* Непрерывное итеративное тестирование, позволяющее оценить успешность всего проекта в целом;
* Раннее обнаружение конфликтов между требованиями, моделями и реализацией проекта;
* Более равномерная загрузка участников проекта;
* Реальная оценка текущего состояния проекта и, как следствие, большая уверенность заказчиков и непосредственных участников в его успешном завершении.

Этот метод имеет и свои отрицательные стороны:

* Общее представление о возможностях и ограничениях проекта очень долгое время отсутствует;
* При итерациях приходится отбрасывать часть сделанной ранее работы;
* Добросовестность специалистов при выполнении работ всё же снижается, что психологически объяснимо тем, что у них появляется ощущение, что "всё равно всё можно будет переделать и улучшить позже".

Различные варианты итерационного подхода реализованы в большинстве современных методологий разработки (RUP, MSF, XP).

Пример реализации итеративного подхода — методология разработки программного обеспечения, созданная компанией Rational Software.

Итеративную либо быструю (Agile development) разработку часто используют при реализации проектов в области информационных технологий.

## *Spiral*

Спиральная модель была разработана в 1986-ом году Барри Боэмом. Она основана на классическом цикле Деминга PDCA (plan-do-check-act). При использовании этой модели создается в несколько итераций (витков спирали) методом прототипирования (процесс создания прототипа программы — макета (черновой, пробной версии) программы, обычно — с целью проверки пригодности предлагаемых для применения концепций, архитектурных и/или технологических решений, а также для представления программы заказчику на ранних стадиях процесса разработки).

Каждый виток спирали соответствует созданию фрагмента или версии ПО, на нем уточняются цели и характеристики проекта, определяется его качество и планируются работы следующего витка спирали. Таким образом углубляются и последовательно конкретизируются детали проекта и в результате выбирается обоснованный вариант, который доводится до реализации.

Неполное завершение работ на каждом этапе позволяет переходить на следующий этап, не дожидаясь полного завершения работы на текущем, в то время, как при итеративной методологии после каждой итерации чётко видно "приращение". При итеративном способе разработки недостающую работу можно будет выполнить на следующей итерации. Главная же задача - как можно быстрее показать пользователям системы работоспособный продукт, тем самым активизируя процесс уточнения и дополнения требований.

Основная проблема спирального цикла - определение момента перехода на следующий этап.

## *Agile*

Зачастую бывает необходимо модернизировать IT-решения или разрабатывать новые программные продукты в самые сжатые сроки. При этом требования к функциональности программного обеспечения не оговорены и постоянно пересматриваются. В таких условиях выбирают для реализации проектов модель гибкой разработки программного обеспечения – Agile.

Agile (и ее вариации – методологии Scrum, Extreme Programming и т.д.) подразумевает разработку программного обеспечения короткими циклами. Каждый этап – это уменьшенный вариант ИТ-проекта: проводится планирование, анализ требований, проектирование, кодирование, тестирование и документирование. По окончании этапа заказчик получает работающую версию ИТ-системы, если требуется, пересматривает дальнейшие приоритеты проекта, и цикл разработки запускается снова. В итоге создается решение, которое на 100% соответствует ожиданиям компании.

На практике методология Agile может использоваться в нескольких интерпретациях. В проектах для заказчиков наиболее часто применяют: Scrum, Extreme Programming, Lean Software Development (LSD), Dynamic Systems Development Method (DSDM), Open Unified Process (OpenUP), Agile Project Management (APM), Microsoft Solutions Framework для Agile (MSF) и др.

Можно выделить следующие преимущества Agile:

* Быстрая и постоянная обратная связь команды разработчиков с заказчиком (изменения в требованиях пользователей оперативно учитываются);
* Гибкий график реализации функциональности;
* Отсутствие затрат на составление и согласование подробной технической документации (Agile позволяет избежать затрат на работы, которые не всегда нужны (например, составление и согласование подробной технической документации));
* Возможность остановки проекта без ущерба для осуществленных вложений в разработку ИТ-системы (каждая стадия проекта, за исключением начальных, заканчивается созданием вполне готовой версии ИТ-системы; есть возможность (например, при замораживании бюджета) остановить проект и при этом получить работоспособное решение).

Рассмотрим несколько разновидностей Agile-метода.

### *Scrum*

Одна из самых популярных методологий гибкой разработки. Одна из причин ее популярности - простота.

Sprint — итерация в scrum, в ходе которой создаётся функциональный рост программного обеспечения. Ее длительность составляет 1 месяц (30 дней), но может и варьироваться. Результатом Sprint является готовый продукт (build), который можно передавать (deliver) заказчику (по крайней мере, система должна быть готова к показу заказчику).

В методологии Scrum всего три роли:

* Scrum Master
* Product Owner
* Team

Scrum Master - самая важная роль в методологии. Scrum Master отвечает за успех Scrum в проекте. По сути, Scrum Master является интерфейсом между менеджментом и командой. Как правило, эту роль в проекте играет менеджер проекта или тимлид. Важно подчеркнуть, что Scrum Master не раздает задачи членам команды, он проводит совещания (Scrum meetings) следит за соблюдением всех принципов Scrum, разрешает противоречия и защищает команду от отвлекающих факторов. Данная роль не предполагает ничего иного, кроме корректного ведения Scrum-процесса. Руководитель проекта скорее относится к владельцу проекта и не должен фигурировать в качестве Scrum-мастера.

Product Owner - это человек, отвечающий за разработку продукта. Представляет интересы конечных пользователей и других заинтересованных в продукте сторон. Product Owner - это единая точка принятия окончательных решений для команды в проекте, именно поэтому это всегда один человек, а не группа или комитет. Product Owner ставит задачи команде, но он не вправе ставить задачи конкретному члену проектной команды в течении спринта.

В методологии Scrum (Team) команда является самоорганизующейся и самоуправляемой. Команда берет на себя обязательства по выполнению объема работ на спринт перед Product Owner. Работа команды оценивается как работа единой группы. Размер команды ограничивается размером группы людей, способных эффективно взаимодействовать лицом к лицу. Типичные размер команды варьируются от 5 до 9 человек.

Команда в Scrum кроссфункциональна. В нее входят люди с различными навыками - разработчики, аналитики, тестировщики. Нет заранее определенных и поделенных ролей в команде, ограничивающих область действий членов команды.

Для облегчения коммуникаций команда должна находиться в одном месте. Предпочтительно размещать команду в одной общей комнате для того, чтобы уменьшить препятствия для свободного общения. Команде необходимо предоставить все необходимое для комфортной работы.

Заказчик получает возможность гибко управлять рамками системы, оценивая результат спринта и предлагая улучшения к созданной функциональности. Такие улучшения попадают в список Product Backlog, где они выстраиваются по приоритету наравне с прочими требованиями и могут быть запланированы на следующий (или на один из следующих) спринтов.

Каждый спринт представляет собой маленький "водопад". В течение спринта делаются все работы по сбору требований, дизайну, кодированию и тестированию продукта. Границы спринта должен быть фиксированными. Это позволяет команде давать обязательства на тот объем работ, который должен быть сделан в спринте.

В начале каждого спринта проводится планирование спринта. В планировании спринта участвуют заказчики, пользователи, менеджмент, Product Owner, Scrum Master и команда.

Остановка спринта производится в исключительных ситуациях. Спринт может быть остановлен до того, как закончатся отведенные 30 дней. Спринт может остановить команда, если понимает, что не может достичь цели спринта в отведенное время. Спринт может остановить Product Owner, если необходимость в достижении цели спринта исчезла. После остановки спринта проводится встреча с командой, где обсуждаются причины остановки спринта. После этого начинается новый спринт: производится его планирование и начинаются работы. Команда демонстрирует изменения продукта, созданные за последний спринт. Product Owner, менеджмент, заказчики, пользователи, в свою очередь, его оценивают. Команда рассказывает о поставленных задачах, о том, как они были решены, какие препятствия были у них на пути, какие были приняты решения, какие проблемы остались нерешенными. Аналогично происходить переход между спринтами.

Срыв спринта грозит в большей степени только финансовой стороне.

### *XP (экстремальное программирование)*

Название методологии исходит из идеи применить полезные традиционные методы и практики разработки программного обеспечения, подняв их на новый "экстремальный" уровень. Так, например, практика выполнения исследования кода, заключающая в проверке одним программистом кода, написанного другим программистом, в "экстремальном" варианте представляет собой "парное программирование", когда один программист занимается кодированием, а его напарник в это же время непрерывно просматривает только что написанный кoд.

Экстремальное программирование предлагает готовое решение: делать всё максимально просто и быстро, держать заказчика при себе (в основе экстремального программирования лежат не конкретные методики, как принято считать, а лишь четыре базовых принципа: общение, простота, обратная связь и храбрость.), позволяя ему активно следить за процессом разработки.

Двенадцать основных приёмов экстремального программирования могут быть объединены в четыре группы:

* Короткий цикл обратной связи (Fine-scale feedback)
* Разработка через тестирование (Test-driven development)
* Игра в планирование (Planning game)
* Заказчик всегда рядом (Whole team, Onsite customer)
* Парное программирование (Pair programming)
* Непрерывный, а не пакетный процесс
* Непрерывная интеграция (Continuous integration)
* Рефакторинг (Design improvement, Refactoring)
* Частые небольшие релизы (Small releases)
* Понимание, разделяемое всеми
* Простота (Simple design)
* Метафора системы (System metaphor)
* Коллективное владение кодом (Collective code ownership) или выбранными шаблонами проектирования (Collective patterns ownership)
* Стандарт кодирования (Coding standard or Coding conventions)
* Социальная защищенность программиста (Programmer welfare):
* 40-часовая рабочая неделя (Sustainable pace, Forty-hour week)

### *Lean*

Известна как бережливая разработка программного обеспечения, так как использует методы концепции бережливого производства.

Основными Lean-принципами являются:

• Исключение потерь. Потерями считается всё, что не добавляет ценности для потребителя. В частности, излишняя функциональность, ожидание (паузы) в процессе разработки, нечёткие требования, медленное внутреннее сообщение.

• Акцент на обучении. Короткие циклы разработки, раннее тестирование, частая обратная связь с заказчиком.

• Предельно отсроченное принятие решений. Решение следует принимать не на основе предположений и прогнозов, а после открытия существенных фактов.

• Предельно быстрая доставка заказчику. Короткие итерации.

• Мотивация команды. Нельзя рассматривать людей исключительно как ресурс. Людям нужно нечто большее, чем просто список заданий.

• Интегрирование. Передать целостную информацию заказчику. Стремиться к целостной архитектуре. Рефакторинг.

• Целостное видение. Стандартизация, установление отношений между разработчиками. Разделение разработчиками принципов бережливости.

Таким образом, нет универсальной модели. Её выбор, как правило, зависит от нескольких факторов, таких как характер работы (есть ли чётко определенный результат? Разработка веб-сайтов является творческим делом и поэтому подходит для гибкого метода. Разработка транзакционной системы, где есть четко определенный результат, больше подходит для водопадного метода.), тип клиента (согласен ли он работать итеративно, есть ли у него время на проверку и комментирование регулярных итераций?), мнение вашего руководителя, внутренний или внешний клиент (водопадный метод подходит для внешних клиентов, от которых можно потребовать соблюдения подписанного договора, в отличие от внутренних клиентов, способных вынудить вас внести изменения, получив поддержку руководства). Но есть ряд методологий, которые являются излюбленными у крупных компаний. Так, например, Microsoft нередко использует MSF (Microsoft Solutions Framework), которую сама и предложила. А Apple отдаёет предпочтение Agile, Scrum, Kanban и Lean.

Каждому проекту своя методология!