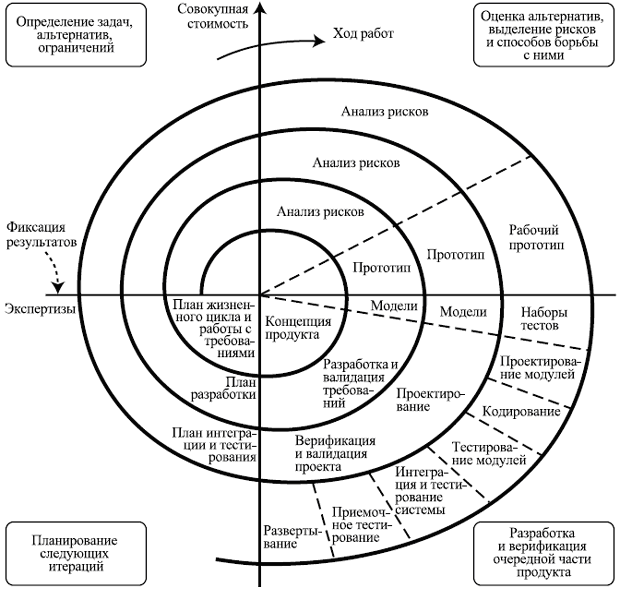
**Спиральная модель**.  
Спиральная модель представляет собой процесс разработки программного обеспечения сочетающий в себе как проектирование так и постадийное прототипирование с целью сочетания преимуществ восходящей и нисходящей концепции, делающая упор на начальные этапы жизненного цикла: анализ и проектирование. Отличительной особенностью этой модели является специальное внимание рискам, влияющим на организацию жизненного цикла. Боэм формулирует десять наиболее распространённых (по приоритетам) рисков:

1. Дефицит специалистов.
2. Нереалистичные сроки и бюджет.
3. Реализация несоответствующей функциональности.
4. Разработка неправильного пользовательского интерфейса.
5. ненужная оптимизация и оттачивание деталей.
6. Непрекращающийся поток изменений.
7. Нехватка информации о внешних компонентах, определяющих окружение системы или вовлечённых в интеграцию.
8. Недостатки в работах, выполняемых внешними (по отношению к проекту) ресурсами.
9. Недостаточная производительность получаемой системы.
10. «Разрыв» в квалификации специалистов разных областей знаний.

Большая часть этих рисков связана с организационными и процессными аспектами взаимодействия специалистов в проектной команде.



Спиральная модель Боэма

Каждый виток спирали соответствует созданию фрагмента или версии программного обеспечения, 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. Каждый виток разбит на 4 сектора:

1. **Постановка задач** (Objective setting) - определяются цели этой фазы, то есть витка, ограничения процесса, результаты, план управления, потенциальные риски и альтернативные стратегии, исходя из рисков.

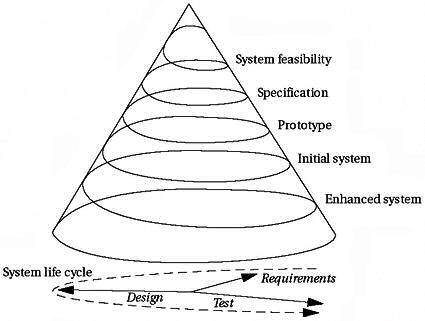
2. **Оценивание и сокращение рисков** (Risk assessment and reduction) - для каждого найденного риска делается анализ, предпринимаются некоторые действия для сокращения рисков (например, риск, чьи требования не являются адекватными: изготавливается прототип).

3. **Разработка и проверка достоверности** (Development and validation) - выбирается модель разработки, исходящая из оцененных рисков (модель должна быть такой, чтобы помочь снизить риски). Например, если в пользовательском интерфейсе имеется самый большой риск, то тогда может помочь прототипирование.

4. **Планирования** (Planning) - проект рассматривается и делается решение о том, переходить ли на следующий виток, если решают продолжить, делается план для следующей фазы.

Пример спиральной модели изображен на рисунке.

На каждом витке спирали могут применяться разные модели процесса разработки ПО. В конечном итоге на выходе получается готовый продукт. Модель сочетает в себе возможности модели прототипирования и [водопадной модели](http://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C_%D0%B2%D0%BE%D0%B4%D0%BE%D0%BF%D0%B0%D0%B4%D0%B0). Разработка итерациями отражает объективно существующий спиральный цикл создания системы. Неполное завершение работ на каждом этапе позволяет переходить на следующий этап, не дожидаясь полного завершения работы на текущем. При итеративном способе разработки недостающую работу можно будет выполнить на следующей итерации. Главная 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. Для ее решения необходимо ввести временные ограничения на каждый из этапов жизненного цикла. Переход осуществляется в соответствии с планом, даже если не вся запланированная работа закончена. План составляется на основе статистических данных, полученных в предыдущих проектах, и личного опыта разработчиков.



**Extreme Programming – Экстремальное программирование**

Экстремальное программирование (XP) – это упрощенная методология организации разработки программ для небольших и средних по размеру команд разработчиков, занимающихся созданием программного продукта в условиях неясных или быстро меняющихся требований.

Основными целями XP являются ***повышение доверия заказчика*** к программному продукту путем предоставления реальных доказательств успешности развития процесса разработки и ***резкое сокращение сроков разработки продукта***. При этом XP сосредоточено на минимизации ошибок на ранних стадиях разработки. Это позволяет добиться максимальной скорости выпуска готового продукта и даёт возможность говорить о прогнозируемости работы. Практически все приемы XP направлены на повышение качества программного продукта.

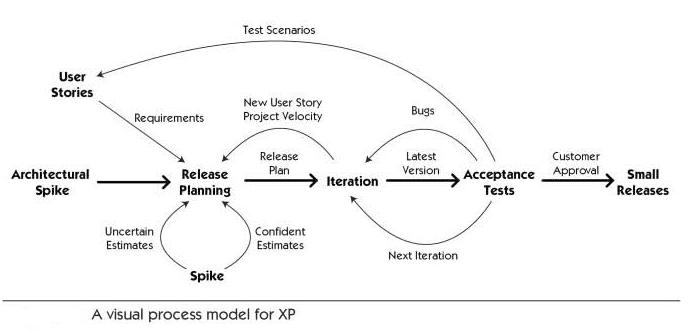
Основными принципами являются:

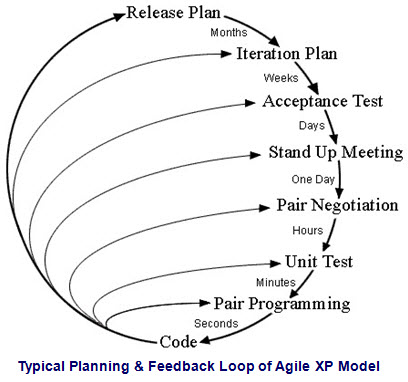
* **Итеративность.** Разработка ведется короткими итерациями при наличии активной взаимосвязи с заказчиком. Итерации как таковые предлагается делать короткими, рекомендуемая длительность – 2-3 недели и не более 1 месяца. За одну итерацию группа программистов обязана реализовать несколько свойств системы, каждое из которых описывается в пользовательской истории. Пользовательские истории (ПИ) в данном случае являются начальной информацией, на основании которой создается модуль. Они отличаются от вариантов использования (ВИ). Описание ПИ короткое – 1-2 абзаца, тогда как ВИ обычно описываются достаточно подробно, с основным и альтернативными потоками, и дополняются моделью. ПИ пишутся самими пользователями, которые в XP являются частью команды, в отличие от ВИ, которые описывает системный аналитик. Отсутствие формализации описания входных данных проекта в XP стремятся компенсировать за счет активного включения в процесс разработки заказчика как полноправного члена команды.
* **Простота решений.** Принимается первое простейшее рабочее решение. Экстремальность метода связана с высокой степенью риска решения, обусловленного поверхностностью анализа и жестким временным графиком. Реализуется минимальный набор главных функций системы на первой и каждой последующей итерации; функциональность расширяется на каждой итерации.
* **Интенсивная разработка малыми группами** (не больше 10 человек) **и парное программирование** (когда два программиста вместе создают код на одном общем рабочем месте), активное общение в группе и между группами. Все это нацелено на как можно более раннее обнаружение проблем (как ошибок, так и срыва сроков). Парное программирование направлено на решение задачи стабилизации проекта. При применении XP методологии высок риск потери кода по причине ухода программиста, не выдержавшего интенсивного графика работы. В этом случае второй программист из пары играет роль «наследника» кода. Немаловажно и то, как именно распределены группы в рабочем пространстве – в XP используется открытое рабочее пространство, которое предполагает быстрый и свободный доступ всех ко всем.
* **Обратная связь с заказчиком**, представитель которого фактически вовлечен в процесс разработки.
* **Достаточная степень смелости** и желание идти на риск.

Обычно XP характеризуют набором из 12 правил (практик), которые необходимо выполнять для достижения хорошего результата. Ни одна из практик не является принципиально новой, но в XP они собраны вместе.

1. **Планирование процесса.** Вся команда разработчиков собирается вместе, принимается коллективное решение о том, какие свойства системы будут реализованы в ближайшей итерации. Трудоемкость реализации каждого свойства определяется самими программистами.
2. Тесное взаимодействие с заказчиком. Представитель заказчика должен быть членом XP-команды. Он пишет ПИ, выбирает истории, которые будут реализованы в конкретной итерации, и отвечает на вопросы, касающиеся бизнеса. Представитель заказчика должен быть экспертом в автоматизируемой предметной области. Необходимо наличие постоянное обратной связи с представителем заказчика. «Заказчик» в XP — это не тот кто оплачивает счета, а тот кто на самом деле использует систему. XP утверждает, что заказчик должен быть всё время на связи и доступен для вопросов.
3. **Общесистемные правила именования.** Хорошие системные правила именования предполагают ***простоту именования*** классов и переменных. Команда разработчиков должна иметь единые правила именования.
4. **Простая архитектура.** Любое свойство системы должно быть реализовано как можно проще. Программисты в XP-команде работают под девизом: «Ничего лишнего!». Принимается первое простейшее работающее решение, реализуется необходимый уровень функциональности на данный момент. Тем самым экономится время программиста.
5. **Рефакторинг.** Это оптимизация существующего кода с целью его упрощения, Такая работа должна вестись постоянно. Сохраняя код прозрачным и определяя его элементы всего один раз, программисты сокращают число ошибок, которые впоследствии придется устранять. При реализации каждого нового свойства системы программист должен подумать над тем, можно ли упростить существующий код и как это поможет реализовать новое свойство. Кроме того, нельзя совмещать рефакторинг с дизайном: если создается новый код, рефакторинг следует отложить.
6. **Парное программирование**. предполагает, что весь код создается парами программистов, работающих за одним компьютером. Один из них работает непосредственно с текстом программы, другой просматривает его работу и следит за общей картиной происходящего. При необходимости клавиатура свободно передается от одного к другому. В течение работы над проектом пары не фиксированы: рекомендуется их перемешивать, чтобы каждый программист в команде имел хорошее представление о всей системе. Таким образом, парное программирование усиливает взаимодействие внутри команды.
7. **40-часовая рабочая неделя.** Программист не должен работать более 8 часов в день. Необходимость сверхурочной работы – это четкий индикатор проблемы на данном конкретном направлении разработки. Поиск причин сверхурочной работы и их скорейшее устранение – одно из основных правил.
8. **Коллективное владение кодом.** Каждый программист в коллективе должен иметь доступ к коду любой части системы и право вносить изменения в любой код. Обязательное правило: если программист внес изменения и система после этого работает некорректно, то именно этот программист должен исправить ошибки.
9. **Единые стандарты кодирования.** Стандарты кодирования нужны для обеспечения других практик: коллективного владения кодом, парного программирования и рефакторинга. Без единого стандарта выполнять эти практики как минимум сложнее, а в реальности вообще невозможно: группа будет работать в режиме постоянной нехватки времени. Команда работает над проектом продолжительное время. Люди приходят и уходят. Никто не кодирует в одиночку и код принадлежит всем. Всегда будут моменты, когда необходимо будет понять и скорректировать чужой код. Разработчики будут удалять дублирующий код, анализировать и улучшать чужие классы и т. п. Со временем нельзя будет сказать, кто автор конкретного класса. Следовательно, все должны подчиняться общим стандартам кодирования – форматирование кода, именование классов, переменных, констант, стиль комментариев. Вышесказанное означает, что все члены команды должны договориться об общих стандартах кодирования. Неважно каких, но все обязаны им подчиняются.
10. **Небольшие релизы.** Минимальная итерация – один день, максимальная – месяц; чем чаще осуществляются релизы, тем больше недостатков системы будет выявлено. Первые релизы помогают выявить недостатки на самых ранних стадиях, далее функциональность системы расширяется на основании ПИ. Поскольку пользователь включается в процесс разработки начиная с первого релиза, то он оценивает систему и выдает пользовательскую историю и замечания. На основании этого определяется следующая итерация, то есть, каким будет новый релиз. В XP все направлено на обеспечение непрерывной обратной связи с пользователями.
11. **Непрерывная интеграция.** Интеграция новых частей системы должна происходить как можно чаще, как минимум раз в несколько часов. Основное правило интеграции следующее: интеграцию можно производить, если все тесты проходят успешно. Если тесты не проходят, то программист должен либо внести исправления и тогда интегрировать составные части системы, либо вообще не интегрировать их. Правило это – жесткое и однозначное. Если в созданной части системы имеется хотя бы одна ошибка, то интеграцию производить нельзя. Частая интеграция позволяет быстрее получить готовую систему, вместо того чтобы тратить на сборку неделю.
12. **Тестирование.** В отличие от большинства остальных методологий тестирование в XP – одно из важнейших составляющих. Экстремальный подход предполагает, что ***тесты пишутся до написания кода***. Каждый модуль обязан иметь unit test – тест данного модуля. Таким образом, в XP осуществляется регрессионное тестирование, «неухудшение качества» при добавлении функциональности. Большинство ошибок исправляются на стадии кодирования. Тесты пишут сами программисты, любой из них имеет право написать тест для любого модуля. Еще один важный принцип: тест определяет код, а не наоборот (test-driven development), то есть кусок кода кладется в хранилище тогда и только тогда, когда все тесты прошли успешно, в противном случае данное изменение кода отвергается.

Процесс XP является неформальным, но требует высокого уровня самодисциплины. Если это правило не выполняется, то XP мгновенно превращается в хаотичный и неконтролируемый процесс. XP не требует от программистов написания множества отчетов и построения массы моделей. В XP каждый программист считается квалифицированным работником, который профессионально и с большой ответственностью относится к своим обязанностям. Если в команде этого нет, то внедрять XP абсолютно бессмысленно – лучше для начала заняться перестройкой команды. Риск разработки снижается только в команде, которой XP подходит идеально, во всех остальных случаях XP – это процесс разработки с наиболее высокой степенью риска, поскольку другие методы снижения коммерческих рисков, кроме человеческого фактора, в XP просто отсутствуют.





Сравнительная таблица моделей жизненного цикла ПО

|  |  |  |
| --- | --- | --- |
| Модель жизненного цикла | **Спиральная** | **Экстренное программирование** |
| Новизна разработки и обеспеченность ресурсами | Нетипичный.  Ресурсы разработчика | Нетипичный.  Группа разработчиков и заказчик |
| Длина цикла | Разработка одной версии системы - средний | Разработка одной истории - короткий |
| Сроки выполнения проекта | До нескольких лет. Разработка одной версии может занимать срок от нескольких недель до года | До года |
| Верификация и внесение изменений | В конце разработки каждого из этапов версии системы - средний | Периодически изменяемые итерации, резко меняемые в пределах версии |
| Интеграция отдельных компонентов системы | Периодически изменяемые интерфейсы, редко в пределах версии | Часто изменяемые интерфейсы |
| Масштаб проекта | Любые проекты | Средние |
| Заключение отдельных договоров на отдельные версии | На отдельную версию или несколько последовательных версий обычно заключается отдельный договор | Заключается один договор |
| Определение основных требований в начале проекта | Нет | Нет |
| Изменение требований по мере развития проекта | Да | Да |
| Разработка итерациями | Да | Да |
| Распространение промежуточного ПО | Да | Да |