Раздел 3. Цикл создания программного обеспечения

Следует начать с определения, **Жизненный цикл программного обеспечения** (Software Life Cycle Model) — это период времени, который начинается с момента принятия решения о создании программного продукта и заканчивается в момент его полного изъятия из эксплуатации. Этот цикл — процесс построения и развития ПО.

3.1 Модели Жизненного цикла программного обеспечения

Жизненный цикл можно представить в виде моделей. В настоящее время наиболее распространенными являются: *каскадная*, *инкрементная* (*поэтапная модель с промежуточным контролем*) и *спиральная* модели жизненного цикла.

Каскадная модель (англ. *waterfall model*) — модель процесса разработки программного обеспечения, жизненный цикл которой выглядит как поток, последовательно проходящий фазы анализа требований, проектирования. реализации, тестирования, интеграции и поддержки.

Процесс разработки реализуется с помощью упорядоченной последовательности независимых шагов. Модель предусматривает, что каждый последующий шаг начинается после полного завершения выполнения предыдущего шага. На всех шагах модели выполняются вспомогательные и организационные процессы и работы, включающие управление проектом, оценку и управление качеством, верификацию и аттестацию, менеджмент конфигурации, разработку документации. В

результате завершения шагов формируются промежуточные продукты, которые не могут изменяться на последующих шагах.

Жизненный цикл традиционно разделяют на следующие основные *этапы* (рис. 3.1):

1. Анализ требований,
2. Проектирование,
3. Кодирование (программирование),
4. Тестирование и отладка,
5. Эксплуатация и сопровождение.

Рис. 3.1 Каскадная модель Жизненного цикла

Достоинства модели:

- стабильность требований в течение всего жизненного цикла разработки;
- на каждой стадии формируется законченный набор проектной документации, отвечающий критериям полноты и согласованности;
- определенность и понятность шагов модели и простота её применения;
- выполняемые в логической последовательности этапы работ позволяют планировать сроки завершения всех работ и соответствующие ресурсы (денежные. материальные и людские).

Каскадная модель хорошо зарекомендовала себя при построении относительно простых ПО, когда в самом начале разработки можно достаточно точно и полно сформулировать все требования к продукту.

Недостатки модели:

- сложность чёткого формулирования требований и невозможность их динамического изменения на протяжении пока идет полный жизненный цикл;
- низкая гибкость в управлении проектом;
- 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 уникальных систем;
- позднее обнаружение проблем, связанных со сборкой, в связи с одновременной интеграцией всех результатов в конце разработки;
- недостаточное участие пользователя в создании системы — в самом начале (при разработке требований) и в конце (во время приёмочных испытаний);
- пользователи не могут убедиться в качестве разрабатываемого продукта до окончания всего процесса разработки. Они не имеют возможности оценить качество, т.к.нельзя увидеть готовый продукт разработки;
- у пользователя нет возможности постепенно привыкнуть к системе. Процесс обучения происходит в конце жизненного цикла, когда ПО уже запущено в эксплуатацию;
- каждая фаза является предпосылкой для выполнения последующих действий, что превращает такой метод в рискованный выбор для систем, не имеющих аналогов, т.к. он не поддается гибкому моделированию.

Реализовать Каскадную модель жизненного цикла затруднительно ввиду сложности разработки ПС без возвратов к предыдущим шагам и изменения их результатов для устранения возникающих проблем.

3.2 Область применения Каскадной модели

Ограничение области применения каскадной модели определяется её недостатками. Её использование наиболее эффективно в следующих случаях:

1. при разработке проектов с четкими, неизменяемыми в течение *жизненного цикла* требованиями, понятными реализацией и техническими методиками;
2. при разработке проекта, ориентированного на построение системы или продукта такого же типа, как уже разрабатывались разработчиками ранее;
3. при разработке проекта, связанного с созданием и выпуском новой версии уже существующего продукта или системы;
4. при разработке проекта, связанного с переносом уже существующего продукта или системы на новую платформу;
5. при выполнении больших проектов, в которых задействовано несколько больших команд разработчиков.

Инкрементная модель (англ. *increment* — увеличение, приращение) подразумевает разработку программного обеспечения с линейной последовательностью стадий, но в несколько инкрементов (версий), т.е. с запланированным улучшением продукта за все время пока Жизненный цикл разработки ПО не подойдет к окончанию (рис. 3.2)..

Рис. 3.2. Поэтапная модель с промежуточным контролем

Разработка программного обеспечения ведется итерациями с циклами обратной связи между этапами. Межэтапные корректировки позволяют учитывать реально существующее взаимовлияние результатов разработки на различных этапах, время жизни каждого из этапов растягивается на весь период разработки.

В начале работы над проектом определяются все основные требования к системе, подразделяются на более и менее важные. После чего выполняется разработка системы по принципу приращений, так, чтобы разработчик мог использовать данные, полученные в ходе разработки ПО. Каждый инкремент должен добавлять системе определенную функциональность. При этом выпуск начинают с компонентов с наивысшим приоритетом. Когда части системы определены, берут первую часть и начинают её детализировать, используя для этого наиболее подходящий процесс. В то же время можно уточнять требования и для других частей, которые в текущей совокупности требований данной работы были заморожены. Если есть необходимость, можно вернуться позже к этой части. Если часть готова, она поставляется клиенту, который может использовать её в работе. Это позволит клиенту уточнить требования для следующих компонентов. Затем занимаются разработкой следующей части системы. Ключевые этапы этого процесса — простая реализация подмножества требований к программе и

совершенствование модели в серии последовательных релизов до тех пор, пока не будет реализовано ПО во всей полноте.

Жизненный цикл данной модели характерен при разработке сложных и комплексных систем, для которых имеется четкое видение (как со стороны заказчика, так и со стороны разработчика) того, что собой должен представлять конечный результат. Разработка версиями ведется в силу разного рода причин:

- отсутствия у заказчика возможности сразу профинансировать весь дорогостоящий проект;
- отсутствия у разработчика необходимых ресурсов для реализации сложного проекта в сжатые сроки;
- требований поэтапного внедрения и освоения продукта конечными пользователями. Внедрение всей системы сразу может вызвать у её пользователей неприятие и только “затормозить” процесс перехода на новые технологии. Образно говоря, они могут просто “не переварить большой кусок, поэтому его надо измельчить и давать по частям”.

Достоинства и **недостатки** этой модели (стратегии) такие же, как и у каскадной (классической модели жизненного цикла). Но в отличие от классической стратегии заказчик может раньше увидеть результаты. Уже по результатам разработки и внедрения первой версии он может незначительно изменить требования к разработке, отказаться от нее или предложить разработку более совершенного продукта с заключением нового договора.

Достоинства:

- затраты, которые получаются в связи с изменением требований пользователей, уменьшаются, повторный анализ и совокупность документации значительно сокращаются по сравнению с каскадной моделью;
- легче получить отзывы от клиента о проделанной работе — клиенты могут озвучить свои комментарии в отношении готовых частей и могут

видеть, что уже сделано. Т.к. первые части системы являются прототипом системы в целом.

- у клиента есть возможность быстро получить и освоить программное обеспечение — клиенты могут получить реальные преимущества от системы раньше, чем это было бы возможно с каскадной моделью.

Недостатки модели:

- менеджеры должны постоянно измерять прогресс процесса. в случае быстрой разработки не стоит создавать документы для каждого минимального изменения версии;
- структура системы имеет тенденцию к ухудшению при добавлении новых компонентов — постоянные изменения нарушают структуру системы. Чтобы избежать этого требуется дополнительное время и деньги на рефакторинг. Плохая структура делает программное обеспечение сложным и дорогостоящим для последующих изменений. А прерванный Жизненный цикл ПО приводит еще к большим потерям.

Схема не позволяет оперативно учитывать возникающие изменения и уточнения требований к ПО. Согласование результатов разработки с пользователями производится только в точках, планируемых после завершения каждого этапа работ, а общие требования к ПО зафиксированы в виде технического задания на всё время её создания. Таким образом, пользователи зачастую получаю ПП, не удовлетворяющий их реальным потребностям.

3.3 Спиральная модель

Спиральная модель: Жизненный цикл — на каждом витке спирали выполняется создание очередной версии продукта, уточняются требования проекта, определяется его качество и планируются работы следующего витка. Особое внимание уделяется начальным этапам разработки — анализу

и проектированию, где реализуемость тех или иных технических решений проверяется и обосновывается посредством создания прототипов (рис. 3.3).

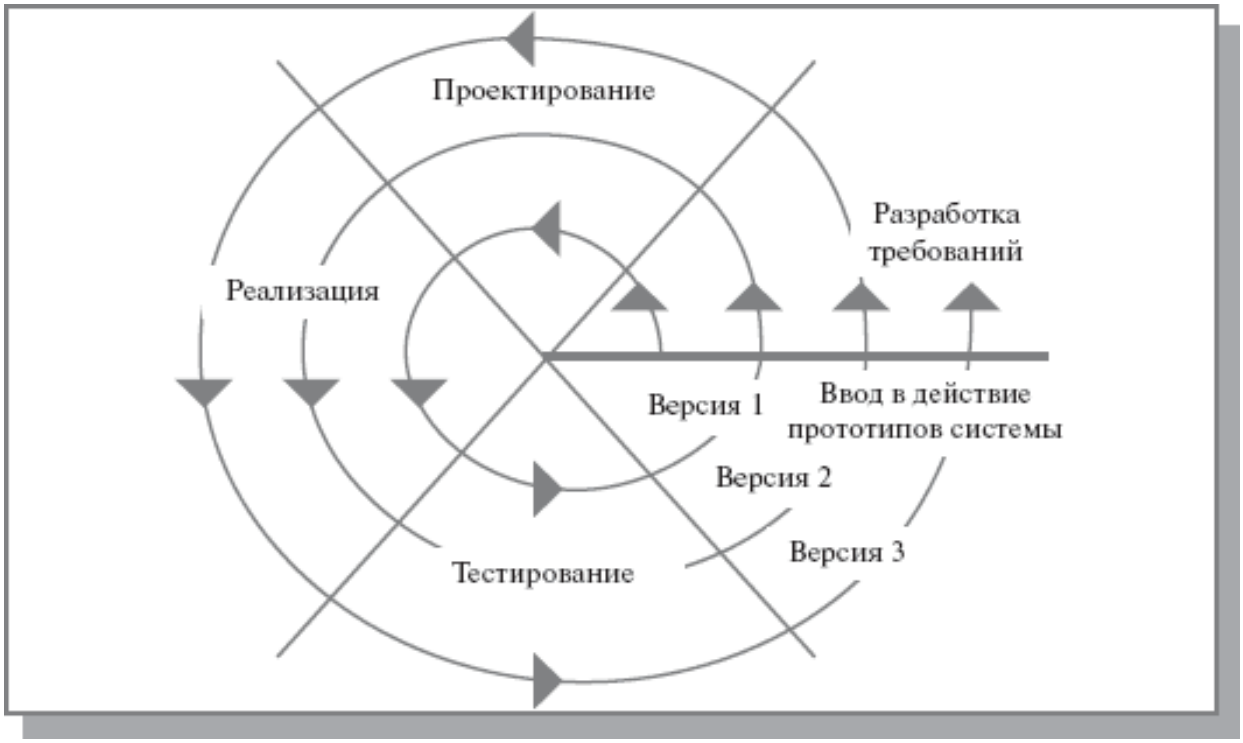

Рис. 3.3. Спиральная модель жизненного цикла

Данная модель представляет собой процесс разработки программного обеспечения, сочетающий в себе как проектирование, так и постадийное прототипировнаие 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.

На этапах анализа и проектирования реализуемость технических решений и степень удовлетворения потребностей заказчика проверяется путем создания прототипов. Каждый виток спирали соответствует созданию работоспособного фрагмента или версии системы. Это позволяет уточнить требования, цели и характеристики проекта, определить качество разработки, спланировать работы следующего витка спирали. Таким образом углубляются и последовательно конкретизируются детали проекта и в

результате выбирается обоснованный вариант, который удовлетворяет действительным требованиям заказчика и доводится до реализации.

Жизненный цикл 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. 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. Главная задача — как можно быстрее показать пользователям системы работоспособный продукт, тем самым активизируя процесс уточнения и дополнения требований.

Достоинства модели:

- позволяет быстрее показать пользователям системы работоспособный продукт, тем самым, активизируя процесс уточнения и дополнения требований;
- допускает изменение требований при разработке программного обеспечения, что характерно для большинства разработок, в том числе и типовых;
- 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;
- позволяет получить более надежную и устойчивую систему. По мере развития программного обеспечения ошибки и слабые места обнаруживаются и исправляются на каждой итерации;
- эта модель разрешает пользователям активно принимать участие при планировании, анализе рисков, разработке, а также при выполнении оценочных действий;
- уменьшаются риски заказчика. Заказчик может с минимальными для себя финансовыми потерями завершить развитие неперспективного проекта;

- обратная связь по направлению от пользователей к разработчикам выполняется с высокой частотой и на ранних этапах модели, что обеспечивает создание нужного продукта высокого качества.

Недостатки модели:

- если проект имеет низкую степень риска или небольшие размеры, модель может оказаться дорогостоящей. Оценка рисков после прохождения каждой спирали связана с большими затратами;
- Жизненный цикл модели имеет усложненную структуру, поэтому может быть затруднено её применение разработчиками, менеджерами и заказчиками;
- 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 циклов может привести к необходимости в обработке дополнительной документации;
- использование модели может оказаться дорогостоящим и даже недопустимым по средствам, т.к. время. затраченное на планирование, повторное определение целей, выполнение анализа рисков и прототипирование, может быть чрезмерным;
- могут возникнуть затруднения при определении целей и стадий, указывающих на готовность продолжать процесс разработки на следующей и

Основная проблема спирального цикла — определение момента перехода на следующий этап. Для её решения вводятся временные ограничения на каждый из этапов *жизненного цикла* и переход осуществляется в соответствии с планом, даже если не вся запланированная работа закончена. *Планирование* производится на основе статистических данных, полученных в предыдущих проектах и личного опыта разработчиков.

3.4 Область применения спиральной модели

Применение спиральной модели целесообразно в следующих случаях:

- при разработке проектов, использующих новые технологии;
- при разработке новой серии продуктов или систем;
- при разработке проектов с ожидаемыми существенными изменениями или дополнениями требований;
- для выполнения долгосрочных проектов;
- при разработке проектов, требующих демонстрации качества и версий системы или продукта через короткий период времени;
- при разработке проектов. для которых необходим подсчет затрат, связанных с оценкой и разрешением рисков.