1. Этапы жизненного цикла программного обеспечения

Стратегия – данный этап предполагает обследование системы т.е. оценка реального объема проекта, его целей и задач, а также получение определений сущностей и функций на высоком уровне. На этом этапе к работе привлекаются бизнес-аналитики, также предполагается тесное взаимодействие с основными пользователями системы и бизнес-экспертами. Основная задача такого взаимодействия — получить как можно более полную информацию о системе, однозначно понять требования заказчика и передать данную информацию в формализованном виде системным аналитикам. Итогом этапа определения стратегии становится документ, где четко сформулировано следующее: что именно причитается заказчику, если он согласится финансировать проект; когда он сможет получить готовый продукт (график выполнения работ); во сколько это ему обойдется (график финансирования этапов работ для крупных проектов). В документе должны быть отражены не только затраты, но и выгода, например время окупаемости проекта, ожидаемый экономический эффект (если его удается оценить).

Анализ - предполагает подробное исследование бизнес-процессов (функций, определенных на предыдущем этапе) и информации, необходимой для их выполнения (сущностей, их атрибутов и связей (отношений)). Данный этап дает информационную модель. Вся информация о системе, собранная на этапе определения стратегии, формализуется и уточняется на этапе анализа. На этом же этапе определяются необходимые компоненты плана тестирования.
Аналитики собирают и фиксируют информацию в двух взаимосвязанных формах:

• функции — информация о событиях и процессах, которые происходят в бизнесе;
• сущности — информация о вещах, которые имеют значение для организации и о которых что-либо известно.
В настоящее время существует способ формализации проекта — Unified Modelling Language (UML), позволяющий формально описать различные стороны жизнедеятельности разрабатываемого проекта.

Проектирование - На этапе проектирования формируется модель данных. Проектировщики получают входные данные анализа. Конечным продуктом этапа проектирования являются схема базы данных (если таковая существует в проекте) или схема хранилища данных (ER-модель) и набор спецификаций модулей системы (модель функций).

Задачами проектирования являются:
• рассмотрение результатов анализа и проверка их полноты;
• семинары с заказчиком;
• определение критических участков проекта и оценка ограничений проекта;
• определение архитектуры системы;
• принятие решения об использовании продуктов сторонних разработчиков, а также о способах интеграции и механизмах обмена информации с этими продуктами;
• проектирование хранилища данных: модель базы данных, бета-версия базы данных;
• проектирование процессов и кода: окончательный выбор средств разработки, определение интерфейсов программ, отображение функций системы на ее модули и определение спецификаций модулей;
• определение требований к процессу тестирования;
• определение требований безопасности системы.

Реализация - При реализации проекта важно координировать группу (группы) разработчиков. Все разработчики должны подчиняться жестким правилам контроля исходных тестов. Группа разработчиков, получив технический проект, начинает

писать код модулей. Основная их задача состоит в том, чтобы уяснить спецификацию: проектировщик написал, что надо сделать, разработчик определяет, как это сделать. На этапе разработки осуществляется тесное взаимодействие проектировщиков, разработчиков и групп тестировщиков. В случае интенсивной разработки тестировщик буквально неразлучен с разработчиком, фактически становясь членом группы разработки. Следует отметить исключительную важность обмена информацией между проектировщиками, разработчиками, тестировщиками: ошибки должны быть классифицированы согласно приоритетам; для каждого класса ошибок должна быть определена четкая структура действий: «что делать», «как срочно», «кто ответственен за результат»; каждая проблема должна отслеживаться проектировщиком/разработчиком/тестировщиком, отвечающим за ее устранение.

Тестирование - Группы тестирования могут привлекаться к сотрудничеству уже на ранних стадиях разработки проекта. Строго говоря, комплексное тестирование следует выделить в отдельный этап разработки. В зависимости от сложности проекта тестирование и исправление ошибок может занимать треть, половину общего времени работы над проектом и даже больше.

Чем сложнее проект, тем больше будет потребность в автоматизации системы хранения ошибок — bug tracking, которая обеспечивает следующие функции:

• хранение сообщения об ошибке (к какому компоненту системы относится ошибка, кто ее нашел, как ее воспроизвести, кто отвечает за ее исправление, когда она должна быть исправлена);

• система уведомления о появлении новых ошибок, об изменении статуса известных в системе ошибок (уведомления по электронной почте);

• отчеты об актуальных ошибках по компонентам системы;

• информация об ошибке и ее история;

• правила доступа к ошибкам тех или иных категорий;

• интерфейс ограниченного доступа к системе bug tracking для конечного пользователя.

Собственно тесты систем можно разделить на несколько категорий:
• автономные тесты модулей; они используются уже на этапе разработки компонентов системы и позволяют отслеживать ошибки отдельных компонентов;
• тесты связей компонентов системы; эти тесты также используются и на этапе разработки, и на этапе тестирования, они позволяют отслеживать правильность взаимодействия и обмена информацией компонентов системы;
• системный тест; он является основным критерием приемки системы; как правило, это группа тестов, включающая и автономные тесты, и тесты связей и модели; данный тест должен воспроизводить работу всех компонентов и функций системы; основная цель данного теста — внутренняя приемка системы и оценка ее качества;
• приемосдаточный тест; основное его назначение — сдать систему заказчику; здесь разработчики часто занижают требования к системе по сравнению с системным тестом, и причины этого вполне очевидны;
• тесты производительности и нагрузки; данная группа тестов входит в системный тест, но достойна отдельного упоминания, поскольку именно эта группа тестов является основной для оценки надежности системы.
Еще одним важным аспектом программы тестирования информационных систем является наличие генераторов тестовых данных. Они используются для проведения тестов функциональности системы, тестов надежности системы и тестов производительности системы. Задачу оценки характеристик зависимости производительности информационной системы от роста объемов обрабатываемой информации без генераторов данных решить невозможно.

Внедрение - Система редко вводится полностью. Как правило, это процесс постепенный или итерационный — в случае циклического жизненного цикла.
Ввод в эксплуатацию проходит как минимум три стадии:
• первоначальная загрузка информации;
• накопление информации;

• выход на проектную мощность (то есть собственно переход к этапу эксплуатации).
Первоначальная загрузка информации инициирует довольно узкий спектр ошибок: в основном речь идет о проблемах рассогласования данных при загрузке и о собственных ошибках загрузчиков.
В период накопления информации из информационной системы выявляется наибольшее количество ошибок, связанных с многопользовательским доступом. Вторая категория исправлений связана с тем, что пользователя не устраивает интерфейс. Здесь не всегда нужно выполнять абсолютно все пожелания пользователя, иначе процесс ввода в эксплуатацию будет бесконечным.
Выход системы на проектную мощность в хорошем варианте — это доводка мелких ошибок и редкие серьезные ошибки.

Эксплуатация и тех. Поддержка - Здесь последним документом, от которого зависят разработчики, является документ технической приемки.
Ошибки, с которыми заказчик согласился, описаны в документации, также определяет необходимый персонал и требуемое оборудование для поддержки работоспособности системы, а также условия нарушения эксплуатации продукта и ответственность сторон. Помимо этого, если документ заключается между сторонами, он содержит условия технической поддержки.

2. Agile-манифест

- Люди и взаимодействие важнее процессов и инструментов
- Работающий продукт важнее исчерпывающей документации
- Сотрудничество с заказчиком важнее согласования условий контракта
- Готовность к изменениям важнее следования первоначальному плану

То есть, не отрицая важности того, что справа,
мы всё-таки больше ценим то, что слева.

3. Scrum Framework (Роли, артефакты, мероприятия)

Скрам – это фреймворк, в рамках которого возможно решать сложные адаптивные проблемы и в то же время продуктивно и креативно разрабатывать продукты наивысшего качества. Фреймворк Скрама состоит из Скрам Команд и связанных с ними ролей, мероприятий, артефактов и правил.

SCRUM команда:

Владелец продукта

- Определяет функциональность продукта.

- Определяет даты релиза и содержимое.
- Приоритизирует задачи.
- Регулярно общается с пользователями.
- Анализирует обратную связь и меняет направление разработки при необходимости.
- Принимает или отклоняет результаты работы
- Ответственнен за прибыль продукта.

SCRUM-мастер

- Следит за корректным применением принципов Agile и процессов (ритуалов) Scrum
- Организует работу команды и обеспечивает её всем необходимым
- Защищает команду, несёт ответственность за её эффективность
- Только один человек.

Команда

- Обычно 6-9 человек.
- Кросс-функциональная: разработчики, тестировщики, дизайнеры, etc.
- Взаимозаменяемая
- Самоорганизующаяся
- Фиксированный состав (в ходе спринта)

Артефакты SCRUM:

• Product Backlog — это приоритезированный список имеющихся на данный момент бизнес требований и технических требований к системе.

Варианты содержимого:
• User Story
• Technical Story
• Bug

• Sprint Backlog — содержит функциональность, выбранную Product Owner из Product Backlog. Все функции разбиты по задачам, каждая из которых оценивается командой.

• Increment — сумма всех выполненных задач из Product Backlog в рамках спринта и значение инкрементов всех предыдущих спринтов.

• Очки истории (Story points)
    Оценка задачи,которая:
    • Не привязана ко времени
    • Не привязана к конкретному участнику команды

Обязательные мероприятия:

• Планирование спринта (Sprint planning)
• Ежедневное Scrum-совещание (Daily Scrum) — 3 вопроса за 15минут:
    • Что я сделал с момента прошлой встречи?
    • Что я сделаю сегодня?
    • Вижу ли я препятствия для себя и команды, которые могли бы затруднить достижение цели спринта?
• Обзор итогов спринта (Sprint Review)
• Ретроспективное совещание (Sprint Retrospective)
• Обсуждение беклога (Baclog refinement meeting)

4. Kanban Method (Диаграмма выполнения работ и ограничение незавершённой работы)

Канбан доска - должна состоять как минимум из трех колонок: «сделать» (to-do), «в процессе» (in progress), «сделано»(done). При разработке ПО SCRUM канбан-доска обычно включает следующие колонки в соответствии со статусом задач: обсуждается (backlog), согласовано (ready), кодируется (coding), тестируется (testing), подтверждается (approval) и сделано (done). На доску в соответствующий столбец прикрепляются канбан-карточки. Каждый из этих объектов представляет собой этап производственного процесса и движется по доске, по мере прогресса. Такое движение соответствует движению SCRUM-процесса производства по Burndown Chart сверху вниз. Часто используется электронная Канбан-доска.

Требования к канбан-доске и канбан-карточкам:

- Стирание надписей происходит легко даже спустя несколько недель
- Карточка легко магнитится к любым металлическим доскам и поверхностям
- Края карточки не расходятся при многократном снятии с доски
- Для надписей используется маркер на водной основе

5. Системы управления версиями. Subversion и Git

SVN или Subversion является свободной централизованной системой управления версиями. Такие системы используются для облегчения работы с информацией, которая постоянно изменяется. Это может быть исходный код программы, скрипта, веб-страницы, сайта, текстового документа и других. Позволяет хранить несколько версий одного и того же документа, с возможностью возврата к более ранним версиям и просмотра информации о том, кто и какие изменения вносил.

VN используют для автоматизации процесса разработки и избавления от рутинной работы. Прежде всего, эта программа полезна для веб-специалистов:

- вам не нужно помнить про все модификации и ошибки, все эти функции исполняет SVN;
- нет необходимости сохранять ревизии на жестком диске, это уже размещено в репозитории системы;
- происходит автоматический процесс создания папок с небольшими отличиями;
- вам не нужно переносить проекты руками, можно получить их с репозитория в новом месте.

Кроме всего этого, система не допускает ошибок, которые веб-разработчик мог бы допустить, выполняя все эти действия вручную.

Среди возможностей данного ПО стоит выделить следующие ***полезные функции***:

- хранения истории изменений в централизованном хранилищи;
- копирование объектов с разветвлением истории;
- обеспечение переноса изменений между копиями объектов, даже полного слияния копий;
- обеспечение ветвления – создание и слияние ветвей;
- обеспечение меток;
- история изменений и копии объектов хранятся в виде связанных простых копий;
- с помощью атомарных транзакций осуществляется фиксация изменений в хранилище;
- эффективная работа с текстовыми и двоичными файлами;
- много вариантов доступа к хранилищу;
- допустимость зеркалирования хранилища;
- возможные внутренние форматы хранилища: база данных или набор обычных файлов;
- наличие библиотек для популярных языков и возможность их подключения и использования;
- многоуровневая архитектура библиотек.

В отличии от систем распределения таких как Git, Subversion хранит информацию в одном едином хранилище, которое может быть или на локальном диске или на сетевом сервере. Рассмотрим еще одну популярную систему контроля версий **Git**. Данная система уже считается распределенной, в отличии от предыдущей. ***Для чего используют git***: для синхронизации работы с сайтом и хранения/обновлений версий сайтов. Эта система удобна, когда над одним проектом работают одновременно несколько разработчиков. Гит позволяет обновлять и править файлы сайта учитывая изменения, которые были внесенные другими.

Преимущества использования Git:

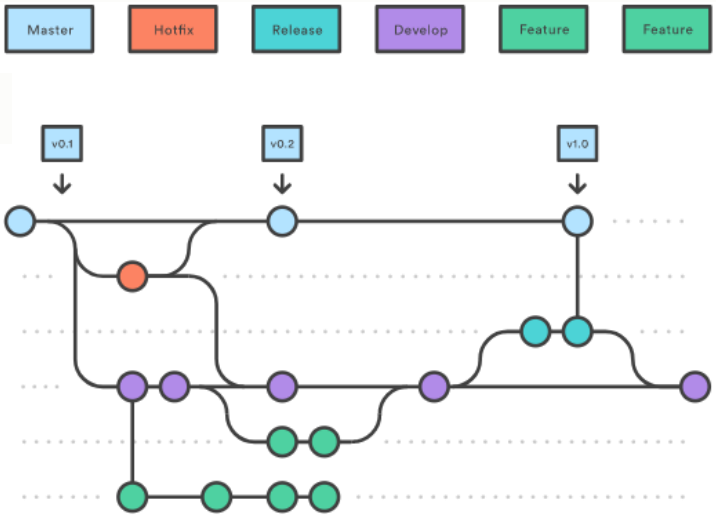
- Git имеет все плюсы использования VCS, что и в Subversion;
- имеет легкий процес шифрования “из коробки”;
- в случае приостановки сервера с главным репозиторием, можно делать коммиты в локальный сервер и ждать восстановления работы сервера;
- служебная информация хранится в корне репозитория;
- разработано много полезных утилит: Qgit, gitk, gitweb и другие. «Из коробки» есть импорт и экспорт в/из Subversion/CVS;
- система пользуется огромной популярностью и с ней работают много веб-разработчиков, поэтому помощь можно будет легко найти в любом случае, например, на специализированном форуме;
- с помощью Git можно размещать свой код на GitHub, BitBucket и Google Code.

Git отличается высокой производительностью, имеет развитые средства интеграции с другими VCS. Также включает продуманную систему команд и приятный для работы веб-интерфейс “из коробки”.

Работая с Git вы получаете абсолютный контроль над репозиторием и быстроту работы.

6. Модели ветвления в Git

Git flow



• Ветка develop для полной истории
• Ветка master для урезанной истории
• Каждая новая функциональность разрабатывается в своей feature ветке

GitHub flow
• Никих develop, release и hotfix веток
• Любая работа (новая функциональность или исправление ошибок) выполняется в отдельной ветке, порождённая от master

GitLab flow, ветки окружения
• Любая работа (новая функциональность или исправление ошибок) выполняется в отдельной ветке
• Дополнительные ветки для среды тестирования

Комментарии:
• Staging — для внутреннего тестирования
• Production — для поставки решения потребителю
• Дополнительные ветки для версий
• Правило “Upstream first”: сначала мерж фикса в master, а после вытягивание исправления в нужные версии

7. Заинтересованные лица / Стейкхолдеры

Стейкхолдеры (или заинтересованные лица) – это группы людей или отдельные люди, которых проект как-то затрагивает (как в хорошем, так и в плохом смысле) либо (и это важно, про это почему-то часто забывают!) те, кого проект не затрагивает, но они сами могут его «затронуть» или как-то на него повлиять, используя имеющиеся у них возможности.

В первую группу входят как непосредственные участники проекта (команда, спонсор, подрядчики, менеджеры, которым придется выделять свои ресурсы для работы на проекте и проч.), так и потребители результата проекта (заказчик, конечные пользователи, сотрудники, которых как-то заденет изменение процессов работы, и проч.).

Во вторую – те, кого проект напрямую не касается, но кто может на его очень ощутимо повлиять.

8. Способы спецификации требований

Спецификация требований программного обеспечения (англ. Software Requirements Specification, SRS) является полным описанием поведения системы, которая будет создана. Она включает ряд сценариев использования, которые описывают все виды взаимодействия пользователей с программным обеспечением. Сценарии использования также известны как функциональные требования. В дополнении к сценариям использования, спецификация программного обеспечения также содержат нефункциональные (или дополнительные) требования. Нефункциональные требования — требования, которые налагают дополнительные ограничения на систему (такие как требования эффективности работы, стандарты качества, или проектные ограничения).

Требования систематизируются несколькими способами. Ниже представлены общие классификации требований, которые касаются технического управления.

Клиенты, это те, кто выполняет основные функции системного проектирования, со специальным акцентом на пользователе системы как ключевом клиенте. Пользовательские требования определят главную цель системы и, как минимум, ответят на следующие вопросы:

- Требования эксплуатации или развёртывания: Где система будет использоваться?
- Профиль миссии или сценарий: Как система достигнет целей миссии?
- Требования производительности: Какие параметры системы являются критическими для достижения миссии?
- Сценарии использования: Как различные компоненты системы должны использоваться?
- Требования эффективности: Насколько эффективной должна быть система для выполнения миссии?
- Эксплуатационный жизненный цикл: Как долго система будет использоваться?
- Окружающая среда: Каким окружением система должна будет эффективно управлять?

Функциональные требования объясняют, что должно быть сделано. Они идентифицируют задачи или действия, которые должны быть выполнены. Функциональные требования определяют действия, которые система должна быть способной выполнить, связь входа/выхода в поведении системы.

Нефункциональные требования — требования, которые определяют критерии работы системы в целом, а не отдельные сценарии поведения.
Нефункциональные требования определяют системные свойства такие как производительность, удобство сопровождения, расширяемость, надежность, средовые факторы эксплуатации.

Требования, которые подразумеваются или преобразованы из высокоуровневого требования. Например, требование для

большего радиуса действия или высокой скорости может привести к требованию низкого веса.

9. Пользовательские истории (User story)

Пользовательские истории — способ описания требований к разрабатываемой системе, сформулированных как одно или более предложений на повседневном или деловом языке пользователя. Пользовательские истории используются гибкими методологиями разработки программного обеспечения для спецификации требований (вместе с приёмочными испытаниями). Каждая пользовательская история ограничена в размере и сложности. Часто история пишется на маленькой бумажной карточке. Это гарантирует, что она не станет слишком большой.
В Экстремальном программировании пользовательские истории пишутся пользователями (заказчиками) системы. В методологии SCRUM — пишутся либо одобряются ролью владельца продукта. Для заказчиков (пользователей) пользовательские истории являются основным инструментом влияния на разработку программного обеспечения.

Пользовательские истории — быстрый способ документировать требования клиента, без необходимости разрабатывать обширные формализованные документы и впоследствии тратить ресурсы на их поддержание. Цель пользовательских историй состоит в том, чтобы быть в состоянии оперативно и без накладных затрат реагировать на быстро изменяющиеся требования реального мира.

Пользовательская история остается неофициальным определением требований, пока отсутствует процедура приемочного тестирования. Прежде чем реализовывать пользовательскую историю, клиент должен определить соответствующую приемную процедуру, чтобы гарантировать что цели пользовательской истории были достигнуты.

В методологии экстремального программирования пользовательские истории являются результатом планирования, и определяют то, что должно быть реализовано в программном проекте. Пользовательские истории приоретизируются клиентом по важности для системы, разбиваются на серию задач и оцениваются разработчиками.

Непосредственно перед реализацией разработчики могут обсудить историю с заказчиком. Истории могут быть сложными для понимания, могут требовать специфические знания, или требования, возможно, могли измениться со времени написания.

К каждой пользовательской истории в какой-то момент должно быть прикреплено одно или более приемочное тестирование. Это позволяет разработчику узнать, когда пользовательская история готова и как клиенту проверить это. Без точных формулировок требований в момент поставки продукта могут возникнуть длительные неконструктивные разногласия.

10. Очки истории (Story points)

Стори поинтами измеряют усилия, которые нужны, чтобы выполнить элемент Бэклога Продукта или любой другой отрезок работы.

Пользуясь стори поинтами, мы присваиваем каждому элементу (работы) некое количественное значение. Сами по себе эти количественные оценки не важны. Важно то, как оценки разных элементов соотносятся друг с другом. История, которой присвоено значение 2, должна быть вдвое больше истории со значением 1 и соответствовать двум третям истории со значением 3.

Вместо единицы, двойки и тройки команда могла бы использовать цифры 100, 200 и 300. Или 1 миллион, 2 миллиона, 3 миллиона. Важно соотношение, а не цифры как таковые.

Поскольку стори поинты выражают усилия для выполнения истории, команда должна оценить все, что повлияет на эти усилия. Это может быть:

1. Объем работы для выполнения.
2. Сложность работы.
3. Риски или неопределенность при выполнении работы.

Чем больше работы необходимо выполнить, тем, очевидно, больше должен быть условный показатель усилий. Возьмем разработку

двух веб-страниц. На одной должно быть только одно поле и просьба ввести имя. На второй странице полей для текста должно быть 100.

Вторая страница не сложнее первой. Она не предусматривает взаимодействия между полями, и заполнять их не нужно ничем, кроме текста. Ее разработка не связана ни с какими рисками. Единственное отличие двух страниц — на второй сделать нужно больше.

Второй странице стоит присвоить больше стори поинтов. Может, не в 100 раз больше, хотя на ней и во 100 раз больше полей. В конце концов, существует эффект масштаба, так что в реальности мы можем затратить на вторую страницу всего в 2, 3 или в 10 раз больше усилий, чем на первую.

Нужно включать в оценку и риски или неопределенные моменты в работе. Например, команде бывает нужно оценить элемент бэклога, о котором стейкхолдер не может сказать ничего конкретного. Похожая ситуация: нужно давать наценку по сложности, когда внедрение фичи требует переписать старый ненадежный код без автотестов.

Неопределенность как таковая отражена уже в самой последовательности чисел для стори поинтов, которая напоминает ряд Фибоначчи: 1, 2, 3, 5, 8, 13, 20, 40, 100. Какое бы число вы не выбрали, усредненная неопределенность уже заложена в него — если,, конечно, вы используете именно такое соотношение чисел.

Также нужно учитывать сложность. Вернемся к нашей странице со 100 полями, между которыми нет взаимодействий.

А теперь представим себе другую страницу со 100 полями. Часть из них — поля с датами и, соответственно, календарными виджетами. В другой части можно вводить только ограниченное количество символов: например, номера телефонов. Есть и поля, в которых проверяется контрольная сумма (например, с номерами банковских

карт). Кроме того, есть взаимодействия между полями. Для карты Visa страница должна выдавать поле с трехзначным CVV-кодом, а для карты American Express — с четырехзначным.

Хотя на экране по-прежнему 100 полей, разработка будет сложнее, а значит, займет больше времени. Растет вероятность, что разработчик ошибется и ему придется откатить какие-то из изменений и переделать работу.

Может показаться, что невозможно отобразить три фактора в одном числе, но это не так. Объединяющим фактором должны стать необходимые усилия. Сначала оценивающие учитывают, какими могут быть общие трудозатраты по каждому элементу бэклога.

Потом нужно оценить, как на трудозатраты повлияет риск. Для этого стоит озвучить риски и оценить их влияние. Например, больше стори поинтов стоит присвоить элементу с более высокой степенью риска, особенно если он потребует выполнить больший объем работы, а не элементу, с которым проблемы менее вероятны.

И наконец, необходимо оценить сложность. Она может потребовать больше осмысления, проб и ошибок, коммуникации с заказчиком, проверки и времени на исправление ошибок.

11. Диаграмма сценариев использования в UML

Диаграмма вариантов использования (use case) в UML - диаграмма, отражающая отношения между акторами и прецедентами и являющаяся составной частью ***модели прецедентов***, позволяющей описать систему на концептуальном уровне.

Прецедент — возможность моделируемой системы (часть её функциональности), благодаря которой пользователь может получить конкретный, измеримый и нужный ему результат. Прецедент соответствует отдельному сервису системы, определяет

один из вариантов её использования и описывает типичный способ взаимодействия пользователя с системой. Варианты использования обычно применяются для спецификации внешних требований к системе.

Основное назначение диаграммы — описание функциональности и поведения, позволяющее заказчику, конечному пользователю и разработчику совместно обсуждать проектируемую или существующую систему.

При моделировании системы с помощью диаграммы прецедентов системный аналитик стремится:

- чётко отделить систему от её окружения;
- определить действующих лиц (актёров), их взаимодействие с системой и ожидаемую функциональность системы;
- определить в глоссарии предметной области понятия, относящиеся к детальному описанию функциональности системы (то есть прецедентов).

Работа над диаграммой может начаться с текстового описания, полученного при работе с заказчиком. При этом нефункциональные требования (например, конкретный язык или система программирования) при составлении модели прецедентов опускаются (для них составляется другой документ).

Основное назначение диаграммы — описание функциональности и поведения, позволяющее заказчику, конечному пользователю и разработчику совместно обсуждать проектируемую или существующую систему.

При моделировании системы с помощью диаграммы прецедентов системный аналитик стремится:

- чётко отделить систему от её окружения;
- определить действующих лиц (актёров), их взаимодействие с системой и ожидаемую функциональность системы;
- определить в глоссарии предметной области понятия, относящиеся к детальному описанию функциональности системы (то есть прецедентов).

Часть дублирующейся информации в модели прецедентов можно устранить указанием связей между прецедентами[1]:

- ***обобщение прецедента*** — стрелка с незакрашенным треугольником (треугольник ставится у более общего прецедента),
- ***включение прецедента*** — пунктирная стрелка со стереотипом «include»,
- ***расширение прецедента*** — пунктирная стрелка со стереотипом «extend» (стрелка входит в расширяемый прецедент, в дополнительном разделе которого может быть указана *точка расширения* и, возможно в виде комментария, условие расширения)