Программное обеспечение (ISO/IEC 12207) - набор компьютерных программ, процедур и связанных с ними документации и конфигурационных данных, необходимых для работы программы. Специалисты по программной инженерии разрабатывают конечные продукты, которые могут быть проданы конечному потребителю. В зависимости от того, для кого разрабатывается программный продукт (конкретного заказчика программного продукта) выделяют два типа продуктов:
1) коробочные продукты;
2) заказные;
в зависимости от того, кто определяет и специфицирует требования:
1) разработчик;
2) заказчик.
Программная инженерия - это инженерная дисциплина, связанная со всеми аспектами производства программного обеспечения, от начальных стадий создания спецификаций до поддержки системы после сдачи в эксплуатацию.
Методы программной инженерии - это структурные решения, которые предназначены для разработки программного обеспечения и которые включают системные модели, формализованные нотации, правила проектирования, а также способы управления процессом создания программного обеспечения.
Информатика занимается теорией и методами вычислительных и программных систем, в то время как программная инженерия занимается практическими проблемами создания ПО. При этом программная инженерия должна быть поддержана некоторыми теориями информатики. Таким образом, программные инженеры чаще всего используют приёмы, применимые в конкретных условиях и не могут быть обобщены на другие случаи, а теории информатики не всегда могут быть применимы к реальным большим системам.
Отличия программной инженерии:
1) компьютерная программа - это не материальный объект;
2) программа - искусственный объект;
3) программная инженерия - молодая дисциплина.
Профессиональные и этические требования к специалистам по программной инженерии:
1) конфиденциальность: не разглашать никакие сведения о работодателе и клиентах;
2) компетентность: специалист не должен скрывать свой уровень компетенции и не должен браться за работу, которая не соответствует его уровню;
3) защита прав на интеллектуальную собственность: специалист не должен нарушать права интеллектуальную собственность;
4) злоупотребление компьютером: специалист не должен наносить вред компьютерам других людей.
Кодекс этики (согласно стандарту IEEE-CS/ACM): программные инженеры должны руководствоваться следующими восемью принципами:
1) общество: программные инженеры будут действовать в соответствии с общественными интересами;
2) клиент и работодатель: программные инженеры будут действовать в интересах клиентов и работодателей согласно общественным интересам;
3) продукт: программные инженеры будут добиваться, чтобы сделанные ими продукты и их модификации соответствовали высоким профессиональным стандартам;
4) суждения: программные инженеры будут добиваться честности и независимости в своих профессиональных суждениях;
5) менеджмент: менеджеры и лидеры программных инженеров будут руководствоваться этическим подходом к руководству разработкой и сопровождением программного обеспечения, а также будут продвигать и развивать этот подход;
6) профессия: программные инженеры будут улучшать целостность и репутацию своей профессии в соответствии с интересами общества;
7) коллеги: программные инженеры будут честными по отношению к своим коллегам и будут всячески их поддерживать;
8) личность: программные инженеры на протяжении всей своей жизни будут учиться практике своей профессии и будут продвигать этический подход к практике своей профессии.
Технология - это подробное описание того, как нужно производить то или иное изделие и наука о составлении таких описаний. Стандарт - это утверждённый компетентными органами нормативно-технический документ, устанавливающий комплекс норм и правил относительно предмета стандартизации. Сертификация считается основным достоверным способом подтверждения соответствия продукции, процесса или услуги заданным требованиям.
Типы стандартов:
1) корпоративный стандарт: разрабатываются крупными фирмами (корпорациями) с целью повышения качества продукции на основе собственного опыта и с учётом требований мировых стандартов. Корпоративные стандарты не сертифицируются, но обязательны для применения в рамках заданной корпорации;
2) отраслевые стандарты: действуют в пределах организации, определённой отрасли или министерства; разрабатываются с учётом требований мирового опыта и специфики отрасли; являются обязательными для отрасли и подлежат сертификации;
3) государственные стандарты (ГОСТ): принимаются государственными органами и имеют силу закона; разрабатываются с учётом мирового опыта или на основе отраслевых стандартов; могут иметь как рекомендательный, так и обязательный характер; для сертификации создаются государственные и лицензированные органы спецификации;
4) международные стандарты: разрабатываются специальными международными организациями на основе мирового опыта и лучших корпоративных стандартов; имеют сугубо рекомендательный характер; право сертификации получают организации, государственные или частные, прошедшие лицензирование в международных организациях.
Основные разработчики стандартов в области программной инженерии:
1) International Organization for Standartization;
2) Association for Computer Machinery;
3) Software Engineering Institute;
4) Project Management Institute;
5) Institute of Electrical and Electronic Engineers

Принципы создания программных систем:
1) абстракция и уточнение: один из базовых принципов описания сложных систем: чтобы работать со сложными системами, мы используем возможности абстрагироваться, то есть отвлекаться от всего, что несущественно поставленной на данный момент конкретной цели и не влияет на те аспекты рассматриваемого предмета, которые для этой цели важны. Чтобы прийти от абстрактного представления к более конкретному, используется обратный процесс последовательного уточнения. Рассмотрев систему в каждом аспекте отдельно, мы пытаемся объединить результаты анализа. Добавляя аспекты по одному, и обращая при этом внимание на возможные взаимные влияния и возникающие связи между элементами, обнаруженными при анализе отдельных аспектов;
2) модульность: принцип организации больших систем в виде наборов подсистем, модулей или компонентов, взаимодействующих друг с другом через чётко определённые интерфейсы. При этом каждая задача, решаемая всей системой, разбивается на простые подзадачи, решаемые отдельными модулями. Их решения, скомбинированные определённым образом, дают в итоге решение исходной задачи. Хорошо спроектированные интерфейсы должны обладать четырьмя требованиями:
1) адекватность: интерфейс модуля даёт возможность решать именно те задачи, которые нужны пользователям этого модуля;
2) минимальность интерфейса: у предоставляемой интерфейсом операции решает различные по содержанию задачи и ни одну из них невозможно реализовать с помощью всех других;
3) полнота интерфейса: интерфейс позволяет решать все значимые задачи в рамках функциональности модуля;
4) простота интерфейса: интерфейсные операции достаточно элементарны и не могут быть представлены как композиция некоторых более простых операций на том же уровне абстракции и при том же понимании функциональности модуля;
3) разделение ответственности: основной принцип выделения модулей - это создание отдельных модулей под каждую задачу, решаемую системой, или необходимую как составляющая для решения её основных задач;
4) слабая связность модулей и сильное сцепление функций в одном модуле. Слабая связность модулей требует, чтобы зависимостей между модулями было меньше. Модуль, который зависит от большинства других модулей в системе, скорее всего нужно перепроектировать, потому что он решает слишком много задач, и наоборот, сцепление функций, выполняемых одним модулем, должно быть как можно выше;
5) принцип повторного использования: требует избегать повторений описаний одинаковых знаний в виде структур данных, действий, алгоритмов, кодов в разных частях системы. Вместо этого в хорошо спроектированной системе выделяется один источник, одно место фиксации для каждого элемента знаний и организуется его повторное использование во всех местах, где нужно использовать этот элемент знаний. Подобная организация позволяет при необходимости удобным образом модифицировать код и документы системы в соответствии с новым содержанием элементов знаний.

Методологии проектирования:

1) структурное проектирование (декомпозиция сверху вниз). Основная идея этой методики заключается в том, что мы движемся от общего утверждения типа "что делает данная программа" к более детальным описаниям конкретных выполняемых задач: "разделяй и властвуй", "проектирование сверху вниз", "пошаговое уточнение";
2) структурное проектирование (композиция снизу вверх). Основной вопрос: что будет точно необходимо в этой системе; выделяется несколько низкоуровневых задач. Сформулировав список потребностей для исследуемой системы, можно снова вернуться к самому верху и начать движение вниз или пробовать группировать имеющиеся на низком уровне задачи в определённые общие группы и тем самым продвинуться на один уровень вверх;
3) объектно-ориентированное проектирование: ставит своей целью идентификацию и реализацию объектов, принимающих участие в процессе, работает на более высоком уровне абстракции, чем структурное проектирование, поскольку объекты включают в себя как данные, так и методы, позволяющие работать над ними, то появляется возможность работать с более объёмными наборами информации, не увеличивая существенное количество деталей, которые видны на самом высоком уровне.
Принципы объектно-ориентированного подхода:
1) абстракция - это характеристика сущности, которая отличает её от других сущностей. Абстракция определяет границу представления соответствующего элемента модели, применяется для определения фундаментальных понятий объектно-ориентированного подхода. Класс - абстракция совокупности реальных объектов, которые имеют общий набор свойств и общее поведение. Объект в контексте объектно-ориентированного подхода рассматривается как экземпляр соответствующего класса. Наследование тесно связано с иерархией классов, которая определяет, какие классы следует считать наиболее абстрактными и общими по отношению к другим классам. При этом в то время как общий или родительский класс (предок) имеет фиксированный набор свойств и поведение, то производный от него класс (потомок) должен содержать этот же набор свойств и подобное поведение, а также дополнительные, которые будут характеризовать уникальность полученного класса (производный класс наследует свойства и поведение родительского класса);
2) инкапсуляция: характеризует сокрытие отдельных деталей внутреннего устройства класса от внешних по отношению к нему объектов или пользователей. Отдельные свойства этого класса могут быть невидимы за его пределами. Это относится к базовой идее введения различных категорий видимости для элементов класса;
3) полиморфизм - свойство объектов принимать различные внешние формы в зависимости от обстоятельств. Применительно к ООП полиморфизм означает, что действия, выполняемые одноимёнными методами, могут различаться в зависимости от того, к какому из классов относится тот или иной метод.
Требования к программному обеспечению

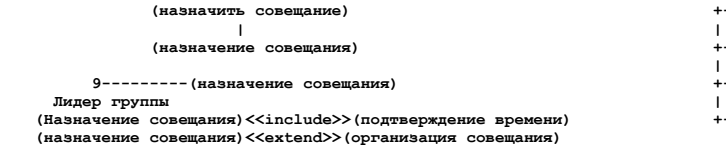
Требование - это желаемое свойство, характеристика или условие, которому должна удовлетворять система в процессе своей эксплуатации. IEEE Standard Glossary Software Engineering Terminology трактует требования:
1) условия или возможности, необходимые пользователю для решения проблем или достижения целей;
2) условия или возможности, которыми должна обладать система или системные компоненты, чтобы выполнить контракт, или удовлетворять стандартам, спецификациям или другим формальным документам;
3) документированное предоставление условий или возможностей для пунктов 1) и 2).
Уровни требований:
1) бизнес-требования;
2) требования пользователей;
3) функциональные требования.
Системные требования и требования к программному обеспечению:
1) Карл Биггерс формулирует термин как "высокоуровневые требования к продукту, содержащему несколько подсистем". При этом под системой понимают программную, программно-аппаратную или человеко-машинную систему;
2) INCOSE (International Council on System Engineering) даёт детальное определение системы: это комбинация взаимодействующих элементов, которая создана для достижения определённых целей и может включать аппаратные средства, программное обеспечение, встроенное программное обеспечение, другие средства, людей, информацию, техники, службы и другие поддерживающие элементы;
3) в практике компьютерной инженерии под системными требованиями в узком смысле понимают требования, предъявляемые прикладной программной системой, в частности, информационной, к среде своего функционирования (системного или аппаратного).
Функциональные требования регламентируют функционирование или поведение системы (behavioral requirements) и отвечают на вопрос "что должна делать система", при этом не должно отвечать на вопрос "как должна работать система". Функциональные требования устанавливают цели, задачи и сервисы, предоставляемые системой заказчику, и определяют основной "фронт работ разработчика". Формы функциональных требований:
1) правила (что и для кого должна делать система);
2) варианты использования.
Нефункциональные требования регламентируют внутренние и внешние условия или аттрибуты функционирования систем. Выделяют следующие основные группы нефункциональных требований:
1) внешние интерфейсы: наиболее важным является интерфейс пользователя (UI), кроме того, выделяют интерфейсы с внешними устройствами (аппаратные интерфейсы), программные интерфейсы, интерфейсы передачи информации;
2) аттрибуты качества: применимость, надёжность, производительность,

эксплуатационная пригодность;
 3) ограничения: формулировка условий, которые модифицируют требования или наборы требований, сужают выбор возможных решений по их реализации, выбор платформы реализации и/или развёртывания: протоколы, серверы приложений, серверы баз данных.
Features - набор логически связанных функциональных требований, обеспечивающих возможности пользователя и удовлетворяющих бизнес-цели. С т. з. инженерии требований, features являются самостоятельным артефактом характеристики продукта, который может быть соотнесён как с функциональными, так и не с функциональными требованиями.
В спецификации Rational Unified Process используется модель FURPS+ (Functionality, Usability, Reliability, Performance, Supportability + проектные ограничения, требования управления системой, требования к графическому интерфейсу пользователя, физические требования, юридические требования).
Разработка требования - это процесс формирования, анализа, документирования и проверки функциональных возможностей и ограничений на программную систему. Этот процесс завершается созданием и утверждением документа, содержащего спецификацию системных требований. Т. о., спецификация требований - официальное изложение системных требований.
Пользователи спецификации требований:
 1) заказчик системы: определяет требования и их изменения;
 2) менеджер: использует спецификацию требований для определения цены системы и планирования процесса её разработки;
 3) системный инженер: использует требования, чтобы понять, какую систему следует разработать;
 4) инженер по тестированию систем: использует требования для разработки системных тестов;
 5) инженер по сопровождению системы: использует требования, чтобы понять систему и взаимосвязи между её частями.
Шаблон документа описания требований, составленный Карлом Виггерсом на основе стандарта IEEE:
 1) введение:
  1.1) назначение документа;
  1.2) поддерживаемые соглашения;
  1.3) предполагаемая аудитория и рекомендации по последовательности работы с документом для каждого класса читателей;
  1.4) границы проекта: краткое резюме проекта или ссылка на документ-концепцию;
  1.5) ссылки;
 2) общее описание:
  2.1) общий взгляд на продукт;
  2.2) особенности продукта (вставляется контекстная диаграмма (вариантов использования, потоков данных));
  2.3) классы и характеристики пользователей: документируется процесс поиска актёров, всех возможных пользователей системы;
  2.4) операционная среда;
  2.5) ограничения проектирования и реализации:
   2.5.1) технологии и средства, языки программирования, базы данных;
   2.5.2) ограничения, налагаемые операционной средой продукта;
   2.5.3) обязательные соглашения или стандарты разработки;
   2.5.4) совместимость с продуктами, выпущенными ранее;
   2.5.5) ограничения, налагаемые бизнес-правилами;
   2.5.6) ограничения, связанные с оборудованием;
   2.5.7) соглашения, связанные с интерфейсом пользователя;
   2.5.8) форматы и протоколы обмена данных;
  2.6) документация для пользователей;
2.7) предположения и зависимости.
 3) функции системы (для каждой i-й функции системы):
  3.i) название i-й функции системы:
   3.i.1) описание и приоритеты;
   3.i.2) последовательности "влияния-реакции";
   3.i.3) функциональные требования;
 4) требования к внешнему интерфейсу:
  4.1) пользовательские интерфейсы;
  4.2) интерфейсы оборудования;
  4.3) интерфейсы программного обеспечения, службы;
  4.4) интерфейсы передачи информации;
 5) другие нефункциональные требования:
  5.1) требования к производительности;
 приложение А): словарь терминов, глоссарий;
 приложение Б): модели анализа;
 приложение В): список вопросов.

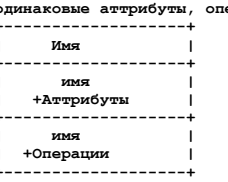
Анализ требования к программному обеспечению.
Варианты использования и проектирование, ориентированное на пользователя
Вариант использования (use case) - это определённый сценарий действия пользователя системы, который обеспечивает ощутимый и значимый для пользователя результат. На практике в виде одного варианта использования оформляют вариант действий пользователя системы, который будет неоднократно возникать во время её работы и имеет достаточно чётко определённые условия начала и завершения выполнения. Проектируемая программная система представляется в форме вариантов использования, с которыми взаимодействуют внешние сущности или актёры. Актёром или действующим лицом называется любой объект, субъект или система, взаимодействующая с моделируемой бизнес-системой извне: это может быть человек, техническое устройство, программа или любая другая система, которая служит источником влияния на моделируемую систему так, как определит разработчик - ВИ служит для описания сервисов, которые система предоставляет актёру.
Диаграмма ВИ (use case diagram) описывает функциональное назначение системы или то, что бизнес-система должна делать в процессе своего функционирования. Также это диаграмма, на которой изображаются отношения между актёрами и вариантами использования в виде графа специального вида.
Содержание варианта использования может быть представлено в форме дополнительного пояснительного текста, который раскрывает содержание или семантику действий при выполнении этого варианта использования. Такой пояснительный текст получил название текста-сценария или просто сценария.
Актёр - согласованное множество ролей, которые играют внешние сущности при совершении вариантов использования при взаимодействии с ними.
Отношения (relationship) - семантическая связь между отдельными элементами модели. Язык UML обеспечивает несколько стандартных видов отношений: ассоциации, включения, расширения и обобщения.
Ассоциация специфицирует семантические особенности взаимодействия актёров и вариантов использования в графической модели системы.

Включения в UML - это отношение зависимости между базовым вариантом использования и его специальным случаем. При этом отношение зависимости является таким отношением между двумя элементами модели, при котором изменение одного элемента (независимого) приводит к изменению другого элемента (зависимого). <<include>>
Отношение расширения определяет взаимосвязь базового варианта использования с другим вариантом использования, функциональное поведение которого требуется базовому не всегда, а лишь при выполнении дополнительных условий. <<extend>>
Отношение обобщения возникает, когда два или более актёра имеют общие свойства, то есть одинаково взаимодействуют с одним подмножеством ВИ. Такая общность свойств изображается как отношение обобщения с другим, возможно, астрактным, актёром. который моделирует соотвествующую общность ролей. <-----------
Отношение обобщения между ВИ применяется в том случае, когда необходимо отметить, что дочерние варианты использования имеют все особбнности поведения родительских вариантов, при этом дочерние варианты использования учавствуют во всех отношениях родительских вариантов. В свою очередь, дочерние варианты могут иметь свойства, отсутствующие у родительских, а также уточнять или модифицировать существующие у них свойства поведения.

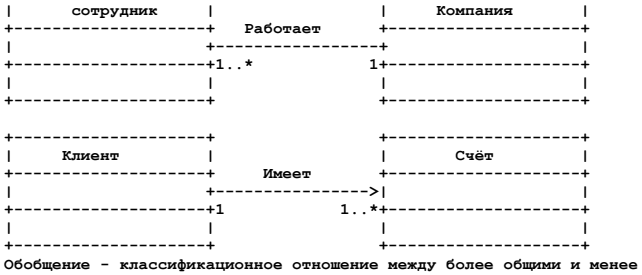
главный раздел	порядок выполенения	исключения	примечания
имя варианта использования	типовый ход		
актёры	событий,		
цель	который приводит		
краткое описание	к успешному		
уровень цели	исполнению		
ссылки на другие варианты использования	варианта использования		



Моделирование предметной области
Предметная область - это часть реального мира, которая имеет существенное значение или непосредственное отношение к процессу функционирования программы.
Объектно-ориентированный анализ и проектирование - это технология разработки программных систем, в основу которой положена объектно-ориентированная методология представления предметной области в виде объектов, которые являются экземплярами соответствующих классов.
Основное место в этой методологии занимает логическая модель системы, которую принято называть диаграммой классов.
Диаграмма классов - это диаграмма UML, на которой представлена совокупность декларативных и статических элементов модели, таких как классы с аттрибутами и операциями, а также отношения, которые их объединяют. Т.о,, диаграмма классов - это изображение статической структуры модели системы в терминологии классов объектно-ориентированного программирования.
Класс - это абстрактное описание множества однородных объектов, имеющих одинаковые аттрибуты, операции и отношения с объектами других классов.

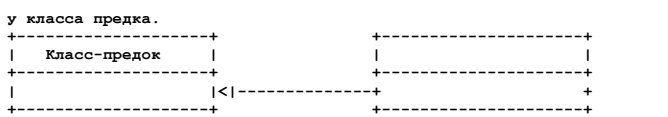


Аттрибут класса - это содержательная характеристика класса, описывающая множество значений, которые могут принимать отдельные объекты этого класса.
Имя аттрибута - это некоторая строка текста, являющаяся уникальным идентификатором соответствующего аттрибута, и уникальна в пределах этого класса.
Квантор видимости - это качественная характеристика описания элементов класса, характеризующая потенциальную возможность других объектов модели влиять на отдельные аспекты поведения этого класса. В стандарте UML существует три способа изображения квантора видимости: + обозначает аттрибут с областью видимости типа "общедоступный" (public): аттрибут с такой областью видимости доступен из любого другого класса пакета, для которого создаётся эта диаграмма классов; # обозначает аттрибут с областью видимости типа "защищённый" (protected): аттрибут с такой областью видимости недоступен или невидим для всех классов, за исключением подклассов этого класса; - обозначает аттрибут с областью видимости типа "закрытый" (private): аттрибут с такой областью видимости недоступен или невидим для всех классов без исключения.
Операция класса - это сервис, который предоставляется каждым экземпляром или объектом класса по требованию своих клиентов, которыми могут выступать другие объекты, в т.ч. экземпляры этого класса. Имя является единственным обязательным элементом, но полезно указывать список параметров и тип возвращаемого значения. Параметр - это спецификация переменной, которая может быть изменена, передана или возвращена операции. Кроме того, перед операцией также может ставиться квантор видимости.
Интерфейс - это именованное множество операций, которое характеризует поведение отдельного элемента модели; в контексте UML это специальный класс, у которого есть операции, но отсутствуют аттрибуты.
Базовые отношения: ассоциации, обобщения, агрегации, композиции.
Ассоциация - это семантическое отношение между двумя и более классами, которое специфицирует характер связи между соответствующими экземплярами этих классов.
Конец ассоциации - это конечная точка ассоциации, которая связывает ассоциацию с классификатором. Свойства ассоциации указываются рядом с элементами ассоциации: роль (специфическое поведение определённой сущности, которая имеет имя и рассматривается в определённом контексте): может быть статической или динамической; кратность ассоциации (имеет отношение к концам ассоциации и обозначается как интервал целых чисел; этот интервал записывается рядом с концом соответствующей ассоциации и означает потенциальное количество экземпляров класса, которые могут иметь место, когда другие экземпляры или объекты класса фиксированы.



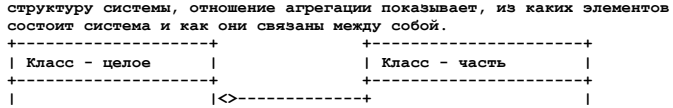
Обобщение - классификационное отношение между более общими и менее общими понятиями, т. е. что класс потомок имеет все свойства и поведение класса-предка, а также свойства и поведение, которые могут отсутствовать

у класса предка.

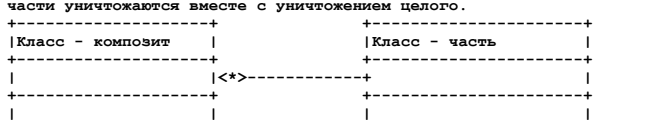


При этом от одного класса-предка могут наследоваться несколько классов-потомков.

Отношение агрегации - это специальная форма ассоциации, которая служит для изображения отношения типа "часть - целое" между агрегатом (целое) и его составной частью. Отношение агрегации имеет место между несколькими классами в том случае, когда один из классов есть сущность, которая содержит в себе как составляющие другие сущности. Раскрывая внутреннюю структуру системы, отношение агрегации показывает, из каких элементов состоит система и как они связаны между собой.



Отношение композиции - разновидность отношения агрегации, при котором составные части целого имеют такое же время жизни, как и целое. Эти части уничтожаются вместе с уничтожением целого.



Моделирование поведения программного обеспечения.

Диаграммы взаимодействия (interaction diagrams) - описывают взаимодействия, состоящие из множества объектов и отношения между ними, включая сообщения, которыми они обмениваются.

Диаграммы кооперации в языке UML

Кооперация (collaboration) - это спецификация множества объектов отдельных взаимодействующих классов с целью реализации отдельных вариантов использования в общем контексте моделируемой системы.

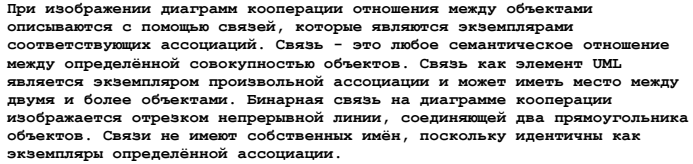
Объект - это сущность с хорошо определёнными границами и индивидуальностью, которая инкапсулирует состояние и поведение. В контексте UML любой объект является экземпляром класса, описанного в модели и изображённого на диаграмме классов. Объект создаётся на этапе реализации модели или выполнения программы: <собственное имя объекта>:<Имя класса>. Вся запись имени объекта подчёркивается, что является визуальным признаком объекта на разных диаграммах UML.

manager:TeamMember

:TeamMember - анонимный

manager - сирота

При изображении диаграмм кооперации отношения между объектами описываются с помощью связей, которые являются экземплярами соответствующих ассоциаций. Связь - это любое семантическое отношение между определённой совокупностью объектов. Связь как элемент UML является экземпляром произвольной ассоциации и может иметь место между двумя и более объектами. Бинарная связь на диаграмме кооперации изображается отрезком непрерывной линии, соединяющей два прямоугольника объектов. Связи не имеют собственных имён, поскольку идентичны как экземпляры определённой ассоциации.



Каждое взаимодействие между множеством объектов описывается совокупностью сообщений, которыми объекты, участвующие в ней, обмениваются между собой.

Сообщение - это спецификация передачи информации от одного элемента модели к другому с ожиданием выполнения определённых действий со стороны принимающего элемента. Сообщения на диаграмме кооперации изображаются дополнительными стрелками рядом с соответствующей связью.

----------|*> обозначает вызов процедуры или операции или передачу потока управления. Сообщения этого типа могут быть использованы параллельно активными объектами, когда один из них передаёт сообщения этого типа и ожидает, пока не закончится определённая последовательность действий, выполняемая вторым объектом. Обычно все такие сообщения синхронные, то есть инициируются по завершению деятельности или при выполнении определённого условия.

--------------> обозначает асинхронное сообщение в простом потоке управления. В этом случае клиент передаёт асинхронное сообщение и продолжает выполнять свою деятельность, не ожидая ответа от клиента.

- - - - - - - -> обозначает возвращение из вызова процедуры

Сообщение - строка текста, в которой указывается порядковый номер сообщения в процедурной последовательности, информация об условия сообщения и имя сообщения. Условие указывается после порядкового номера в [].

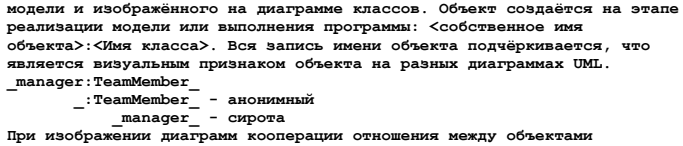
Диаграмма последовательностей в языке UML - это диаграммы, на которых показаны взаимодействия объектов, упорядоченные по времени их появляения. На диаграмме последовательностей неявно присутствует ось времени, которая позволяет визуализировать временные отношения между переданными сообщениями. С помощью диаграммы последовательности можно представить взаимодействие элементов модели как своеобразный временной график жизни всей совокупности объектов, связанных между собой для реализации варианта использования программной системы, достижения бизнес-целей или выполнения любой задачи.

Процесс взаимодействия объектов реализуется с помощью сообщений, которые одни объекты посылают другим. Сообщения изображаются в виде горизонтальных стрелок с именем сообщения, которые образуют определённый порядок своего запуска по времени (инициализации).

Масштаб на оси времени не указывается, поскольку диаграмма последовательности моделирует лишь временную упорядоченность взаимодействия типа "раньше - позже".

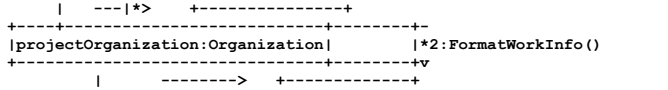
Линия жизни объекта (Object lifeline) - это вертикальная линия на диаграмме последовательности, изображающая существование объекта в течение определённого периода времени.

Фокус управления - это специальный символ на диаграмме последовательности, указывающий период времени, в течение которого объект выполняет определённое действие, находясь в активном состоянии.

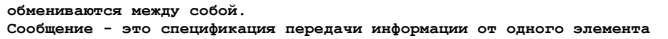


Сообщения изображаются в виде стрелок. Сообщения на диаграмме последовательности упорядочены по времени своей передачи в моделируемой системе, хотя номера в них могут и не указываться. Сообщения обычно изображаются горизонтальными стрелками, которые сочетают линии жизни или фокусы управления двух объектов на диаграмме последовательности. При этом неявно предполагается, что время передачи сообщения достаточно мало по сравнению с процессами выполнения действий объектами.

Каждое сообщение на диаграмме последовательности ассоциируется с определённой операцией, которая должна быть выполнена объектом, который его принял. при этом операция может иметь аргументы или параметры, значения которых влияют на получение различных результатов. Также значения параметров отдельных сообщений могут содержать условные выражения, образуя ветвления или альтернативные пути основного потока управления.



Оператор управления - некая прямоугольная рамка с тегом: opt - условие оператора выполняется, если защитное условие истинно на его входе; alt - условное управление; par -параллельное управление; loop - повторяемое управление.

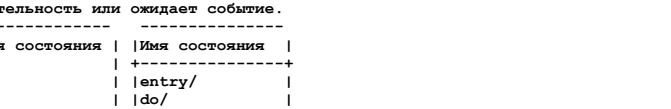


Моделирование динамики функционирования программного обеспечения

Диаграммы состояний в языке UML

Диаграмма состояний (statechart) - это диаграмма, которая представляет собой конечный автомат. Главное её назначение - описать возможные последовательности состояний и переходов, которые в совокупности характеризуют поведение моделируемой системы в течение всего её жизненного цикла. Диаграмма состояний изображает динамическое поведение сущностей на основе спецификации их реакций на восприятие некоторых конкретных событий.

Состояние - это ситуация в ходе жизненного цикла объекта, в течение которой он удовлетворяет логическому условию, выполняет определённую деятельность или ожидает событие.



Действие (action) - это спецификация выполняемого утверждения, которая образует абстракцию вычислительной процедуры. Действие обычно приводит к изменению состояния системы и может быть реализовано с помощью передачи сообщения объекту, модификации связи или значения аттрибута.

Виды действий:

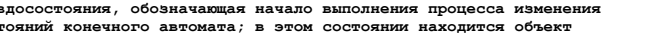
1) входное (entry action) - действие, которое выполняется в момент перехода к данному состоянию;

2) действие выхода (exit action) - действие, которое выполняется при выходе из состояния;

3) внутренняя деятельность (do activity) - выполнение объектом операций или процедур, которые требуют определённого времени.

Псевдосостояния (pseudo state) - вершины в конечном автомате, которые имеют форму состояния, но не имеют поведения:

1) начальное состояние (start state) - это разновидность псевдосостояния, обозначающая начало выполнения процесса изменения состояний конечного автомата; в этом состоянии находится объект устойчиво в начальный момент времени;



2) конечное состояние (final state) - это также разновидность псевдосостояний, которая обозначает прекращение процесса изменения состояний конечного автомата; в этом состоянии должен находиться объекти или система устойчиво после завершения работы конечного автомата.

Переход (transition) - это отношение между двумя состояниями, который указывает на то, что объект в первом состоянии должен выполнить определённые действия, перейти к другому состоянию. Переход осуществляется при осуществлении определённого события, при получении объектом сообщения или при получении (приёме) сигнала.

<Имя события>(список параметров)[сторожевое условие]<выражение действия>

Событие (event) - это спецификация существенных явлений в поведении системы, которые имеют место во времени и пространстве.

Сторожевое условие (guard condition) - это логическое условие, которое записывается в квадратных скобках и является булевым выражением.

Выражение действия (action expression) является вызовом операции или передачей сообщения, имеет атомарный характер и выполняется сразу после срабатывания соответствующего перехода к началу действий в целевом состоянии.

Диаграммы деятельности

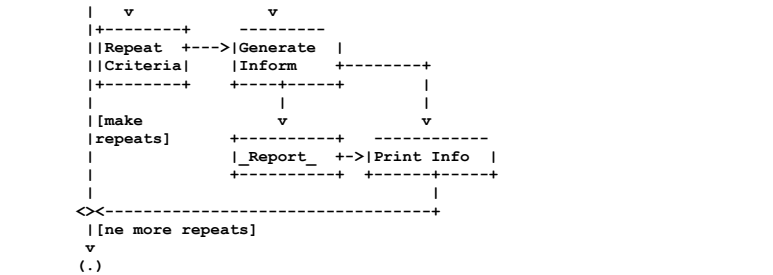


Диаграмма деятельности является частным случаем диаграммы состояний. Основным направлением использования является визуализация особенностей реализации операций класса, когда необходимо представить алгоритмы их выполнения. При этом каждое состояние может быть выполнением операции определённого класса или её части, позволяя использовать диаграмму деятельности для описания реакций на внутренние события системы. Также диаграммы деятельности используются для моделирования потока работ, то есть деятельность с точки зрения действующих лиц, взаимодействующих с системой.

Основные элементы:

1) состояние деятельности (activity state) - это состояние в графе деятельности, которое служит для изображения процедурной последовательности действий, требующих определённого времени; деятельность, описанная в состоянии деятельности, не может быть прервана каким-либо внешним событием, и обычно используется для моделирования выполнения. Обычно состояние деятельности сводится к выполнению определённых алгоритмов или процедур;

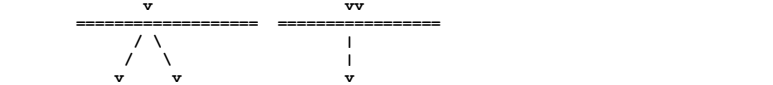
2) состояние действия (action state) - специальный случай состояния с определённым входным действием и по крайней мере одним переходом, который выходит из состояния; обычно используются для моделирования шага выполнения алгоритма или процедуры в рамках одного потока управления. Допускается наличие нескольких конечных состояний, при этом все они эквивалентны друг другу.

Ветвление (decision) <> В ромб может входить только одна стрелка из того состояния действия, после которого поток управления должен быть продолжен по одной из ветвей, которые являются взаимоисключающими. При наличии нескольких стрелок выхода каждой указывается сторожевое условие в виде булевого выражения.

merge <> (соединение) используется для графического объединения альтернативных ветвей.

Возможно объединения символа соединения с символом решения.

Разделение (forking) и слияние (joining) - графическое изображение параллельных потоков на диаграмме деятельности.



Диаграммы деятельности могут быть использованы не только для спецификации алгоритмов вычислений или потоков управления в программных системах, а также для моделирования бизнес-процессов. Для бизнес-процессов желательно выполнение каждого действия ассоциировать с конкретным исполнителем. В этом случае исполнитель будет отвечать за реализацию определённых действий, а сам бизнес-процесс представляется в виде переходов действий от одного исполнителя к другому.

Дорожка (swimlane) - это графический участок диаграммы деятельности, который содержит элементы модели, ответственность за выполнение которых принадлежит отдельным исполнителям или подсистемам.

Базовые принципы создания программных систем