Введение в программную инженерию. Этапы разработки программного обеспечения

План

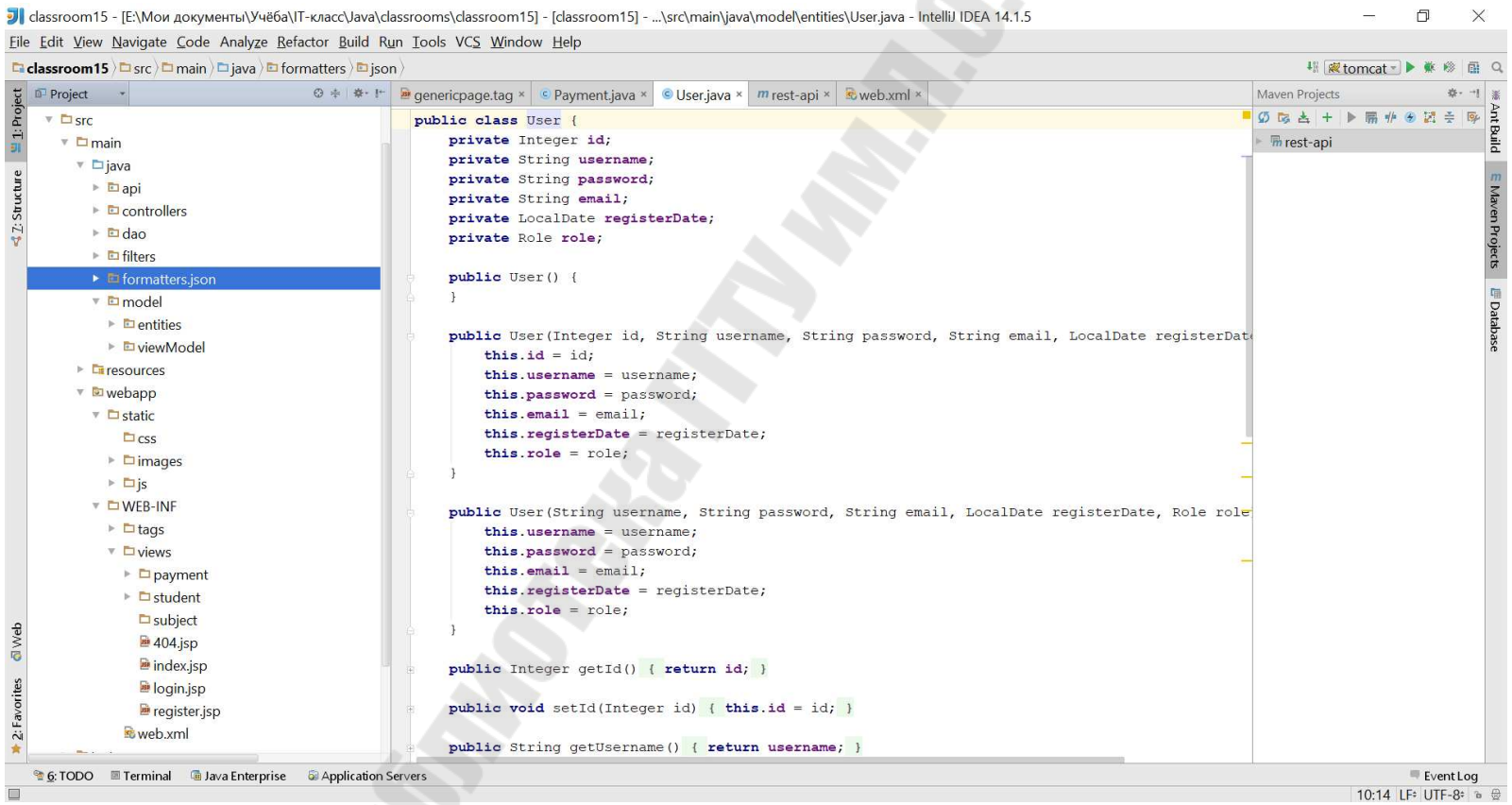
- Введение в программную инженерию
- Основы жизненного цикла программных средств
- Системные основы современных технологий программной инженерии
- Этапы разработки программного обеспечения
- Управление требованиями к программному обеспечению

Введение в программную инженерию

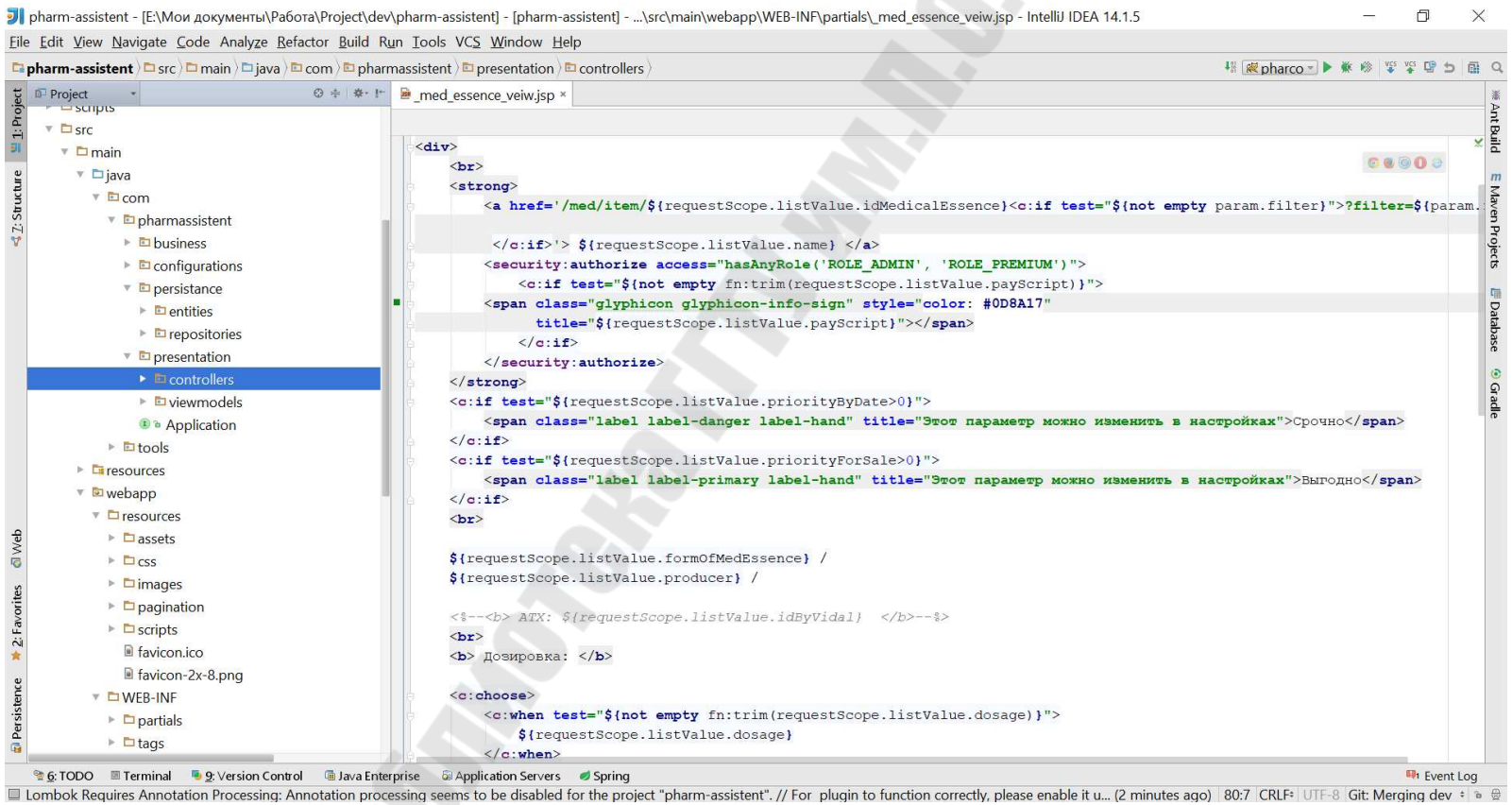
Программная инженерия — приложение систематического, дисциплинированного, измеримого подхода к разработке, функционированию и сопровождению программного обеспечения, а также исследованию этих подходов.

Программная инженерия — это деятельность, связанная с производством и поддержанием ПО.

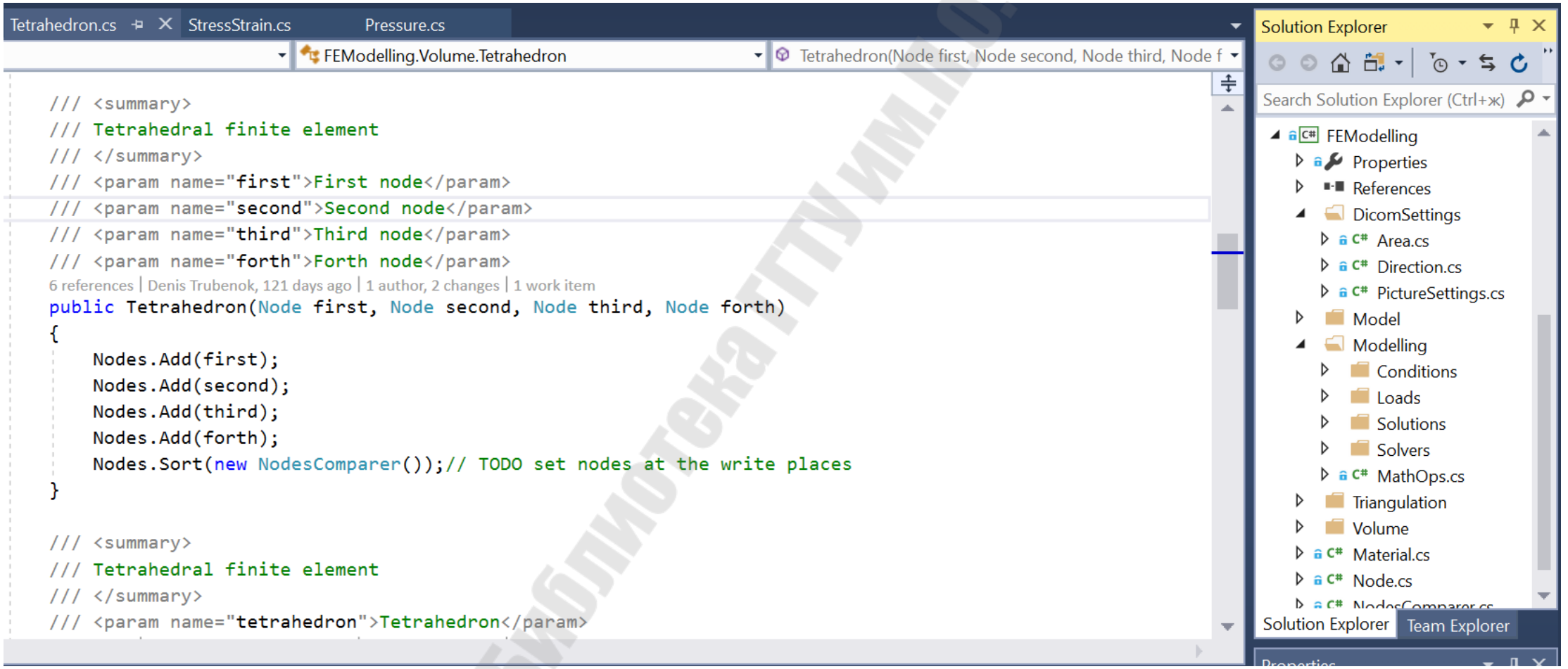
Введение в программную инженерию



Введение в программную инженерию



Введение в программную инженерию



Основы жизненного цикла программных средств

Жизненный цикл программного обеспечения — период времени, который начинается с момента принятия решения о необходимости создания программного продукта и заканчивается в момент его полного изъятия из эксплуатации.

Основы жизненного цикла программных средств

Общую модель жизненного цикла сложной системы обычно разделяют на следующие основные этапы:

- определение потребностей;
- исследование и описание основных концепций;
- проектирование и разработка;
- испытания системы;
- создание и производство;

Основы жизненного цикла программных средств

- распространение и продажа;
- эксплуатация;
- сопровождение и мониторинг;
- снятие с эксплуатации (утилизация).

Основы жизненного цикла программных средств

По особенностям и свойствам жизненного цикла программ их разделить на два крупных класса – большие и малые проекты.

Первый класс составляют относительно небольшие программы, создаваемые одиночками или небольшими командами 3 – 5 человек:

- создаются преимущественно для получения конкретных результатов автоматизации научных исследований или для анализа относительно простых процессов самими разработчиками программ;

Основы жизненного цикла программных средств

- не предназначены для массового тиражирования и распространения как программного продукта на рынке;
- не имеют конкретного независимого заказчика-потребителя, определяющего требования к программам и их финансирование;
- не ограничиваются заказчиком допустимой стоимостью, трудоемкостью и сроками их создания, требованиями заданного качества и документирования;
- не подлежат независимому тестированию, гарантированию качества и/или сертификации.

Основы жизненного цикла программных средств

Второй класс составляют крупномасштабные комплексы программ для сложных систем управления и обработки информации, оформляемые в виде ПО с гарантированным качеством, и отличаются следующими особенностями и свойствами их жизненного цикла:

- большая размерность, высокая трудоемкость и стоимость создания таких комплексов программ определяют необходимость тщательного анализа экономической эффективности всего их жизненного цикла и возможной конкурентоспособности на рынке;

Основы жизненного цикла программных средств

- от заказчика, финансирующего проект программного средства и/или базы данных, разработчикам необходимо получать квалифицированные конкретные требования к функциям и характеристикам проекта и продукта, соответствующие выделенному финансированию и квалификации исполнителей проекта;
- для организации и координации деятельности специалистов-разработчиков при наличии единой, крупной целевой задачи, создания и совершенствования программного продукта, необходимы квалифицированные менеджеры проектов;

Основы жизненного цикла программных средств

- от разработчиков проектов требуются гарантии высокого качества, надежности функционирования и безопасности применения компонентов и поставляемых программных продуктов, в которые не допустимо прямое вмешательство заказчика и пользователей для изменений, не предусмотренных эксплуатационной документацией разработчиков;
- необходимо применять индустриальные, регламентированные стандартами процессы, этапы и документы, а также методы, методики и комплексы средства автоматизации, технологии обеспечения жизненного цикла комплексов программ.

Основы жизненного цикла программных средств

Существует множество моделей процессов жизненного цикла систем и программных средств, но три из них в международных стандартах обычно квалифицируются как фундаментальные: каскадная; инкрементная; эволюционная.

Основы жизненного цикла программных средств

Каскадная модель:

Основы жизненного цикла программных средств

Инкрементная модель:

Основы жизненного цикла программных средств

Эволюционная модель:

Системные основы современных технологий программной инженерии

Основная цель современных технологий программной инженерии состоит в обеспечении эффективности всего жизненного цикла комплексов программ для ЭВМ в различных проблемно-ориентированных областях. В понятие современной технологии включается совокупность методов и инструментальных средств автоматизации, а также технологические процессы, обеспечивающие жизненный цикл сложных ПС с заданными функциональными и конструктивными характеристиками качества.

Системные основы современных технологий программной инженерии

Эта деятельность регламентируется рядом методов и стандартов, которые являются компонентами технологического обеспечения сложных ПС в течение их жизненного цикла. Их применение предполагает высокую дисциплину коллектива специалистов, использование им методик, стандартов, типовых нормативных документов и средств автоматизации разработки, которые регламентируют порядок организации и проведения работ по выполнению технологических операций, направленных на получение, в имеющихся организационно-технических условиях, готового программного продукта с заданными функциями и качеством.

Системные основы современных технологий программной инженерии

Методической основой технологии, регламентирующей деятельность специалистов, является типовой технологический процесс. Он отражается набором этапов и операций в последовательности их выполнения и взаимосвязи, обеспечивающих ведения работ на всех стадиях от инициирования проекта и подготовки технического задания до завершения испытаний или применения версии ПС. В современных технологиях объединяются методы непосредственной разработки программ и данных с методами обеспечения качества и организации управления их созданием с учетом технологических и человеческих факторов.

Системные основы современных технологий программной инженерии

Индустриализация технологий программной инженерии ***базируется на стандартизации процессов*** разработки программ, их структурного построения и интерфейсов с операционной и внешней средой. Для этого с самого начала разработки должны определяться состав и этапы работ, необходимые для достижения конечной цели, а также требуемые для их выполнения ресурсы.

Системные основы современных технологий программной инженерии

Значительные достижения в развитии и применении современных методов и технологии обеспечения ***крупномасштабных проектов ПС*** сосредоточены в методологии **CMM** (Capability Maturity Model –***система и модель для оценки зрелости***) комплекса технологических процессов жизненного цикла ПС, а также в её последующем развитии в **CMMI:2003.**

Системные основы современных технологий программной инженерии

Концептуальные и организационные основы административного управления жизненным циклом и качеством ПС в системе **CMM**, а также **CMMI:2003**, определены в ***восьми базовых принципах***, которые декларированы в стандартах **ISO 9000:2000** и **ISO 15504:1-9**.

Принцип 1: ориентация предприятия-разработчика на потребителя-заказчика . Предприятия зависят от своих потребителей и, таким образом, должны понимать текущие и будущие потребности потребителей-заказчиков, удовлетворять их требования и стремиться превзойти их ожидания.

Системные основы современных технологий программной инженерии

Принцип 2: лидерство-руководство. Лидеры обеспечивают единство назначения и направления деятельности предприятия. Они должны создавать и поддерживать внутреннюю окружающую среду, в которой специалисты могут в полной мере участвовать в достижении стратегических целей предприятия.

Принцип 3: вовлечение персонала. Люди составляют сущность предприятия на всех уровнях, и их полноценное участие в деятельности способствует применению их способностей на благо целей предприятия.

Системные основы современных технологий программной инженерии

Принцип 4: процессный подход. Желаемый результат достигается более эффективно, когда требуемые ресурсы и деятельность специалистов предприятия управляются как единый связанный процесс.

Принцип 5: системный подход к административному управлению. Выявление и понимание задач и административное управление системой взаимосвязанных процессов для заданной стратегической цели, повышает эффективность и результативность предприятия.

Системные основы современных технологий программной инженерии

Принцип 6: постоянное усовершенствование. Непрерывное усовершенствование процессов и повышение качества продукции должно быть постоянной стратегической целью предприятия и его специалистов.

Принцип 7: подход к принятию решений основанный на фактах. Эффективные решения должны базироваться на анализе только реальных данных и достоверной информации.

Системные основы современных технологий программной инженерии

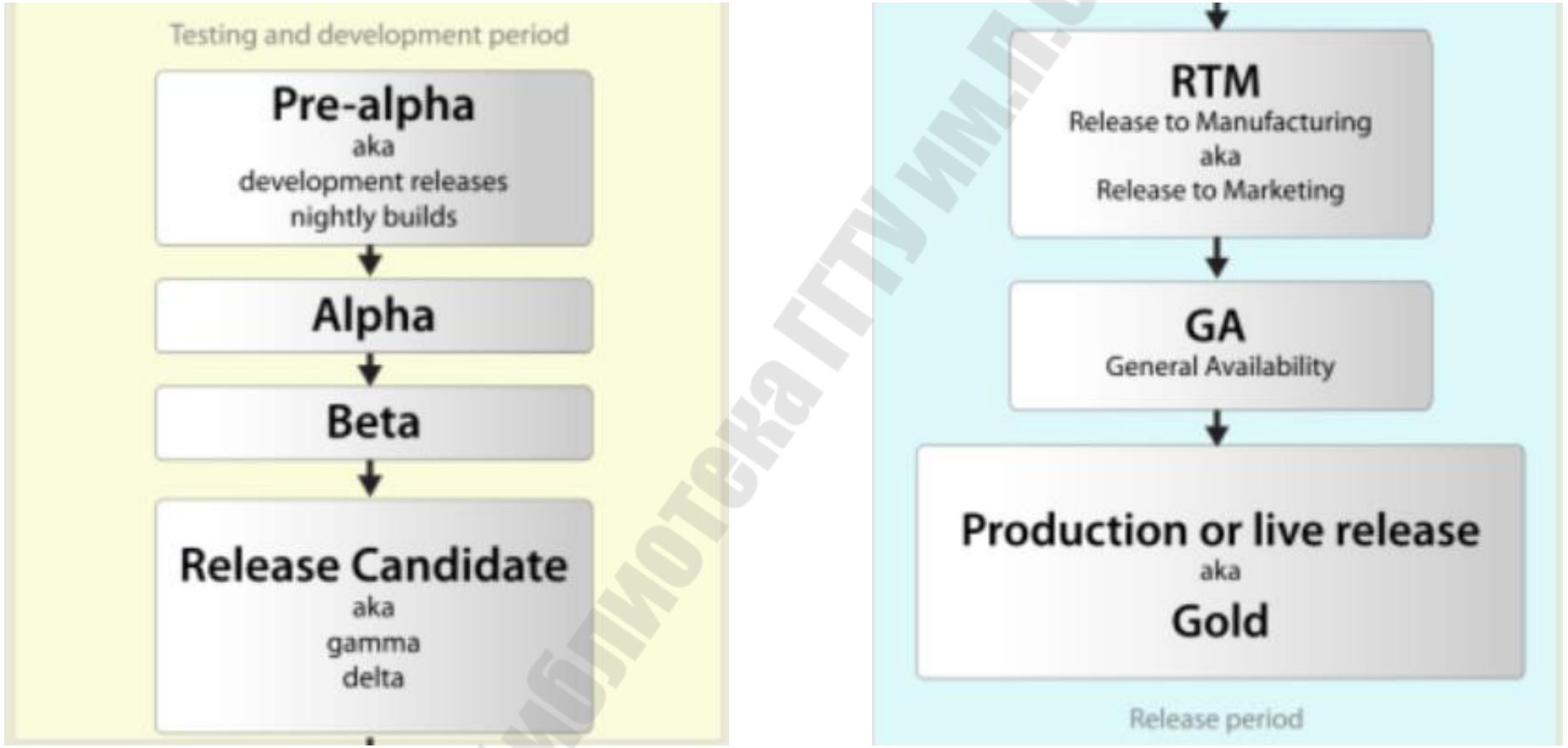
Принцип 8: взаимовыгодные отношения с поставщиками. Предприятие-пользователь и его поставщики-разработчики взаимозависимы, и взаимовыгодные отношения между ними повышают способность обоих производить качественную продукцию.

Этапы разработки программного обеспечения

Типовой проект включает в себя следующие этапы разработки программного обеспечения:

- анализ требований к проекту;
- проектирование;
- реализация;
- тестирование продукта;
- внедрение и поддержка.

Этапы разработки программного обеспечения



Управление требованиями к программному обеспечению

Управление требованиями к программному обеспечению (англ. *software requirements management*) — процесс, включающий идентификацию, выявление, документирование, анализ, отслеживание, приоритизацию требований, достижение соглашения по требованиям и затем управление изменениями и уведомление соответствующих заинтересованных лиц. Управление требованиями — непрерывный процесс на протяжении всего проекта разработки программного обеспечения.

Управление требованиями к программному обеспечению

Цель управления требованиями состоит в том, чтобы гарантировать, что организация документирует, проверяет и удовлетворяет потребности и ожидания её клиентов и внутренних или внешних заинтересованных лиц. Управление требованиями начинается с выявления и анализа целей и ограничений клиента. Управление требованиями, далее, включает поддержку требований, интеграцию требований и организацию работы с требованиями и сопутствующей информацией, поставляющейся вместе с требованиями.

Управление требованиями к программному обеспечению

Требование — это любое условие, которому должна соответствовать разрабатываемая система или программное средство. Требованием может быть возможность, которой система должна обладать и ограничение, которому система должна удовлетворять.

Требование должно обладать следующими характеристиками:

- Единичность — требование описывает одну и только одну вещь.
- Завершенность — требование полностью определено в одном месте и вся необходимая информация присутствует.

Управление требованиями к программному обеспечению

- Последовательность — требование не противоречит другим требованиям и полностью соответствует документации.
- Атомарность — требование нельзя разделить на более мелкие.
- Отслеживаемость — требование полностью или частично соответствует деловым нуждам как заявлено заинтересованными лицами и задокументировано.
- Актуальность — требование не стало устаревшим с течением времени.
- Выполнимость — требование может быть реализовано в рамках проекта.

Управление требованиями к программному обеспечению

- Недвусмысленность — требование определено без обращения к техническому жаргону, акронимам и другим скрытым формулировкам.
- Обязательность — требование представляет собой определенную заинтересованным лицом характеристику, отсутствие которой ведет к неполноценности решения, которая не может быть проигнорирована.
- Проверяемость — реализованность требования может быть проверена.

Управление требованиями к программному обеспечению

Все требования в проекте можно разделить на следующие группы:

- Функциональные (Functional) — реализуют саму бизнес-функцию.
- Управленческие (Manageability) — требования к доступным и безопасным сервисам; относятся к размещению системы, администрированию и безопасности.
- Эргономические (Usability) — к удобству работы конечных пользователей.

Управление требованиями к программному обеспечению

- Архитектурные (Architectural) — требования к архитектуре системы.
- Взаимодействия (Interface) — к взаимосвязям между существующими приложениями и программным средствами и новым приложением.
- Сервисного уровня (Service Level) — описывают поведение сервиса, качество его выходных данных и другие качественные аспекты, измеряемые заказчиком.

Проектирование программного обеспечения и тестирование

План лекции

- Цели и принципы системного проектирования сложных систем
- Структурное проектирование сложных программных средств
- Проектирование программных модулей и компонентов
- Тестирование

Цели и принципы системного проектирования сложных систем

Основная цель системного проектирования **-** подготовить и обосновать замыслы и решения заказчика (потребителя) и разработчика (поставщика) о необходимости, направлениях и концепции создания или модернизации существующих ПС и БД.

Результатом этих работ должны быть ***системный проект, техническое задание и контракт*** на продолжение проектирования или решение о его нецелесообразности и прекращении.

Цели и принципы системного проектирования сложных систем

Методологической базой целевого планирования и управления проектами ПС является системный анализ, который предполагает:

- обследование объектов и среды проектирования для предварительной формализации целей, назначения и задач проекта ПС;
- исследование и сопоставление альтернативных действий, которые должны приводить к достижению поставленных целей проектирования;
- сравнение альтернатив по величине достигаемого эффекта проекта в зависимости от затрат на его достижение (желательно, по показателю «эффективность/стоимость»);
- учет и анализ влияния неопределенностей характеристик альтернатив, определяющих их выбор, на эффект проекта.

Цели и принципы системного проектирования сложных систем

В последовательности выработки и подготовки к реализации этих требований далее рассматриваются в основном **три крупных этапа:**

- **обследование,** системный анализ существующей системы и выявление ее недостатков;
- обобщение результатов системного анализа и создание предварительной **концепции** новой или модернизированной системы и ее программных средств;
- разработка **системного проекта** комплекса программ и базы данных, определяющего и конкретизирующего цель, назначение и методы дальнейшего детального проектирования и всего жизненного цикла ПС.

Структурное проектирование сложных программных средств

При разработке структуры программного средства в процессе системного проектирования, прежде всего, ***необходимо сформулировать критерии*** её формирования. Важнейшими критериями могут быть: модифицируемость, отлаживаемость и удобство управления разработкой ПС, обеспечение возможности контролируемого изменения конфигурации, состава и функций компонентов с сохранением целостности структурного построения базовых версий ПС. В зависимости от особенностей проблемной области, критериями при выборе архитектуры ПС могут также применяться: эффективное использование памяти или производительности реализующей ЭВМ, трудоемкость или длительность разработки, надежность, безопасность и защищенность.

Структурное проектирование сложных программных средств

Гибкость модификации и расширяемость комплексов программ при их совершенствовании обеспечивается рядом принципов и правил структурного построения ПС и их компонентов, а также взаимодействия между ними.

Структурное проектирование сложных программных средств

Основные принципы и правила структурирования ПС и БД можно объединить в группы, которые отражают:

- стандартизированную структуру целостного построения ПС и/или БД определенного класса;
- унифицированные правила структурного построения функциональных программных компонентов и модулей;
- стандартизированную структуру базы данных, обрабатываемых программами;

Структурное проектирование сложных программных средств

- унифицированные правила структурного построения информационных модулей, заполняющих базу данных;
- унифицированные правила организации и структурного построения межмодульных интерфейсов программных компонентов;
- унифицированные правила внешнего интерфейса и взаимодействия компонентов ПС и БД с внешней средой, с операционной системой и другими типовыми средствами организации вычислительного процесса, защиты и контроля системы.

Структурное проектирование сложных программных средств

Методология структурного анализа и предварительного структурного проектирования ПС начинается с общего обзора функций системы. Далее функции должны детализироваться ***сверху вниз*** в виде иерархической структуры таким образом, чтобы процедуры сбора, хранения и переработки информации, рассматриваемые сначала как нечто единое целое, расчленялись на отдельные элементы данных, компонентов и действия, совершаемые над этими данными.

Структурное проектирование сложных программных средств

Учет возможности изменений - это принцип, который более всего отличает программное средство от большинства других типов промышленных продуктов. Во многих случаях структура ПС разрабатывается, когда требования заказчика к нему осознаны не полностью. Позже, при детализации и после поставки, ПС должно развиваться на основе откликов заказчика и пользователей, поскольку обнаруживаются новые требования, а старые уточняются.

Структурное проектирование сложных программных средств

Повторная применимость - является еще одним качеством структуры программного продукта, на которое заметно воздействует принцип возможности изменений. Компонент является многократно применимым, если он может быть непосредственно использован для производства нового продукта или версии.

Проектирование программных модулей и компонентов

Сложная система обычно может быть разделена на более простые части - **модули**. Модульность является важным качеством инженерных процессов и продуктов.

Главное преимущество модульности заключается в том, что она позволяет применять принцип разделения задач на ***двух этапах***:

- при работе с элементами каждого модуля отдельно (игнорируя элементы других модулей);
- при работе с общими характеристиками групп модулей и отношениями между ними с целью объединить, их в конкретный, более крупный и сложный компонент.

Проектирование программных модулей и компонентов

По принципам построения, языку описания, размеру и другим характеристикам компонентов в структуре ПС **можно выделить иерархические уровни**:

- программных модулей, оформляемых как законченные компоненты текста программ;
- функциональных групп (компонентов) или пакетов программ;
- комплексов программ, оформляемых как законченные программные продукты определенного целевого назначения.

Проектирование программных модулей и компонентов

Программные модули решают относительно небольшие функциональные задачи, и каждый реализуется 10-100 операторами языка программирования. Каждый модуль может использовать на входе около десятка типов переменных.

Функциональные группы программ(компоненты) формируются на базе нескольких или десятков модулей и решают достаточно сложные автономные задачи. На их реализацию целесообразно использовать до десятка тысяч строк текста программы.

Проектирование программных модулей и компонентов

Комплексы программ – программные продукты создаются для решения сложных задач управления и обработки информации. В комплексы объединяются несколько или десятки функциональных групп программ для решения общей целевой задачи системы.

Тестирование

Тестирование программного продукта – процесс исследования, испытания программного продукта.

Цели тестирования ПО:

- продемонстрировать разработчикам и заказчикам, что программа соответствует требованиям;
- выявить ситуации, в которых поведение программы является неправильным, нежелательным или не соответствующим спецификации.

Тестирование

Классификация тестирования:

По объекту тестирования

- Функциональное тестирование
- Тестирование производительности (нагрузочное, стресс-тестирование, тестирование стабильности)
- Юзабилити-тестирование
- Тестирование интерфейса пользователя
- Тестирование безопасности
- Тестирование локализации
- Тестирование совместимости

Тестирование

По знанию системы

- Тестирование черного ящика
- Тестирование белого ящика
- Тестирование серого ящика

По степени автоматизации

- Ручное тестирование
- Автоматизированное тестирование

Тестирование

По степени изолированности

- Тестирование компонентов
- Интеграционное тестирование
- Системное тестирование

Сопровождение программного обеспечения

План лекции

- Организация и методы сопровождения программных средств
- Этапы и процедуры при сопровождении программных средств
- Задачи и процессы переноса программ и данных на другие платформы
- Ресурсы, для обеспечения сопровождения и мониторинга программных средств

Организация и методы сопровождения программных средств

При организации сопровождения крупных ПС следует учитывать важные психологические факторы, усложняющие привлечение и деятельность менеджеров и квалифицированных специалистов в этой области:

- эта деятельность требует очень высокой квалификации и больших умственных затрат, связанных, прежде всего, с необходимостью одновременного, широкого охвата и анализа множества компонентов ПС и их взаимосвязей, находящихся в различных состояниях завершенности модификаций;
- корректируемые компоненты зачастую разрабатывались в прошлом в разное время, различными специалистами, в различном стиле и с неодинаковой полнотой документирования, что усложняет освоение их содержания при внесении изменений и устранении дефектов;

Организация и методы сопровождения программных средств

- сложная, творческая сторона работ при сопровождении вуалируется тем, что приходится овладевать и анализировать программы, разработанные ранее другими специалистами, которые зачастую может быть проще не корректировать, а разработать заново;
- комплексы программ, прошедшие широкие испытания и эксплуатацию у заказчиков гарантируют достигнутое качество результатов функционирования, и любые в них изменения имеют высокий риск внесения дополнительных ошибок и ухудшения этого качества, что ограничивает возможность коренных модификаций;
- выполняемые работы требуют особой, скоординированной тщательности корректировок и четкого регламентированного взаимодействия ряда специалистов, различающихся квалификацией и уровнем ответственности.

Организация и методы сопровождения программных средств

Целью сопровождения является выявление и устранение обнаруженных дефектов и ошибок в программах и данных, введение новых функций и компонентов в ПС, анализ состояния и корректировка документации, тиражирование и контроль распространения версий ПС, актуализация и обеспечение сохранности документации и физических носителей.

Основная задача - изменить и улучшить существующий программный продукт, сохраняя его целостность и функциональную пригодность.

Организация и методы сопровождения программных средств

Сопровождаемость – возможность регламентированной модификации, является важной характеристикой ПС для заказчиков, поставщиков и пользователей, отражающей возможность и простоту внесения изменений в программный продукт после его ввода в эксплуатацию.

Сопровождаемость должна быть определена до начала первичной разработки проекта ПС соответствующим соглашением между заказчиком и разработчиком-поставщиком.

Организация и методы сопровождения программных средств

Сопроводители иногда встречаются с необходимостью сопровождать программный продукт с минимальным набором документов или даже при отсутствии их.
В подобной ситуации, сопроводитель при подготовке к сопровождению должен:

- определить проблемную область (тип программного продукта);
- изучить любые доступные документы, по возможности обсудить программный продукт с разработчиками и поработать с данным продуктом;
- изучить структуру и организацию программного продукта;
- установить приоритеты предложений о модификации.

Организация и методы сопровождения программных средств

Описание концепции сопровождения должно быть первым шагом при разработке политики сопровождения ПС. Она должна быть разработана, при первом выпуске исходного программного продукта и отражать:

- область сопровождения и изменений программного продукта;
- практическое применение (адаптацию) данного процесса;
- определение предприятия и лиц, ответственных за сопровождение;
- оценку стоимости и длительности сопровождения.

Организация и методы сопровождения программных средств

Область сопровождения должна определять обязанности сопроводителя и какую поддержку программному продукту он обязан обеспечить. Она зачастую определяется наличием соответствующих ресурсных и бюджетных ограничений и должна охватывать:

- типы допустимых изменений и процедур сопровождения;
- уровень качества сопровождаемых документов;
- реакцию (чувствительность) пользователей на сопровождение;
- обеспечиваемый уровень обучения персонала сопровождения;

Организация и методы сопровождения программных средств

- обеспечение поставки модифицированных версий программного продукта;
- возможность организации справочной службы.

Этапы и процедуры при сопровождении программных средств

Работы, обеспечивающие сопровождение програм-мных продуктов, включают:

- подготовку процесса;
- анализ проблем и изменений;
- внесение изменений;
- проверку и приемку при сопровождении;
- перенос;
- снятие с эксплуатации.

Этапы и процедуры при сопровождении программных средств

Стратегия сопровождения должна быть ориентирована на людские и материальные ресурсы, необходимые и доступные для обеспечения развития и модификаций программного продукта.

Политика сопровождения ПС должна охватывать следующие основные компоненты: концепцию сопровождения; план сопровождения; анализ ресурсов.

Этапы и процедуры при сопровождении программных средств

Целью планирования сопровождения является подготовка плана работ по сопровождению и обеспечение ресурсов, необходимых для проведения этих работ после передачи программного продукта на сопровождение.

Общий план сопровождения должен определять:

- причины необходимости сопровождения;
- состав исполнителей работ по сопровождению;
- роли и обязанности каждого субъекта, вовлеченного в сопровождение;

Этапы и процедуры при сопровождении программных средств

- как должны быть выполнены основные процессы и работы;
- какие имеются и необходимы ресурсы для сопровождения;
- методы и средства организации работ по управлению, выпуску продукта и синхронизации работ;
- перечень всех проектных результатов и продуктов, подлежащих поставке заказчику;
- критерии завершения соответствующей деятельности, работ и задач;
- состав отчетных материалов по этапам, затратам и графикам проведения работ;
- периодичность и способы выдачи отчетных материалов;
- время начала и длительность сопровождения.

Этапы и процедуры при сопровождении программных средств

Анализ дефектов и модификаций рекомендуется реализовать в следующем порядке:

- анализируются предложения о модификации и отчеты о дефектах;
- дублируется или проверяется реальность каждого дефекта;
- разрабатываются варианты реализации изменения;
- документально оформляются: предложения о модификации и отчеты о дефектах;
- проводится согласование выбранного варианта реализации изменения с заказчиком.

Этапы и процедуры при сопровождении программных средств

При внесении изменений в программный продукт сопроводитель разрабатывает и тестирует конкретные изменения программного продукта. Исходными данными для проведения работы при внесении изменений должны быть: базовая версия программного продукта; согласованные с заказчиком предложения о модификации; согласованные документы на реализацию корректировки; отчет о влиянии корректировки и выходные результаты работы по анализу изменений.

Этапы и процедуры при сопровождении программных средств

Проверка и приемка модификаций при эксплуатации обеспечивает подтверждение корректности изменений, внесенных в систему, в соответствии с принятыми стандартами и по установленной методологии. Исходными данными для проведения работы по проверке и приемке при сопровождении являются: измененный программный продукт; результаты квалификационного тестирования внесенных изменений.

Проверки проводятся для гарантирования правильности изменений и их согласованности с точки зрения выполнения установленных требований заказчика к программному продукту.

Этапы и процедуры при сопровождении программных средств

Перенос и внедрение новой версии программного продукта осуществляется, как правило, в два этапа; силами разработчиков модификаций в целях обкатки, проверки и выявления ошибок в изменениях на стадии опытной эксплуатации, и посредством использования специализированных коллективов сопровождения для тиражирования и распространения.

Применение версий программного продукта у пользователей регламентируется установленными правилами и закрепляется соответствующими договорами. Эти договоры определяют порядок поставки, инсталляции, ввода в строй и сопровождения версий ПС, а также порядок обучения пользователей.

Этапы и процедуры при сопровождении программных средств

Снятие программного средства с эксплуатации и сопровождения должно быть подготовлено анализом, обосновывающим это решение. В анализе следует определить и экономически обосновать: возможность сохранения устаревшей версии комплекса программ, а также необходимость создания и применения новой версии программного продукта.

При снятии программного продукта с сопровождения следует определить необходимые для этого действия, а затем разработать и документально оформить этапы работ, обеспечивающие их эффективное выполнение.

Задачи и процессы переноса программ и данных на другие платформы

Многочисленное дублирование, по существу, одних и тех же программных средств и информации баз данных на подобных или разных платформах сопряжено со значительными нерациональными затратами на их разработку и с увеличением длительностей создания информационных систем. Для их сокращения необходима организация, технология и инструментарий, обеспечивающие эффективный перенос готовых программ и данных в пределах одной операционной и аппаратной среды или с иных платформ.

Задачи и процессы переноса программ и данных на другие платформы

Под **переносимостью** понимается:

- процессы переноса программ и данных из одной аппаратной, операционной и пользовательской среды в иную по архитектуре и характеристикам среду с сохранением их целостности или небольшими изменениями функций системы;
- процессы повторного использования готовых программных компонентов и средств, а также информации баз данных, возможно, в пределах одной архитектуры аппаратной и операционной среды для расширения и изменения функций системы и программного продукта.

Задачи и процессы переноса программ и данных на другие платформы

Задачи повторного использования и переноса программ и данных охватывают:

- встраивание готового программного средства и информации базы данных в создаваемую новую систему при условии, что их поставщики гарантируют функционирование на выбранной платформе;
- перенос программ и данных с платформ, в среде которых они были ранее реализованы, на выбранную для системы новую платформу;
- обеспечение доступа к информационным ресурсам других распределенных систем и сетей.

Задачи и процессы переноса программ и данных на другие платформы

Объектами анализа переносимости являются:

- программные модули и функциональные компоненты ПС;
- готовые (покупные) программные продукты и пакеты прикладных программ;
- крупные программные комплексы определенного функционального назначения;
- системы управления базами данных;
- файлы и информационные массивы баз данных;
- электронные документы на программы и данные.

Задачи и процессы переноса программ и данных на другие платформы

Задачи и объекты, связанные с мобильностью ПС и БД в системах, и подлежащие рассмотрению при выборе методов и средств обеспечения переносимости, включают:

- унифицированные протоколы и интерфейсы взаимодействия ПС между собой, с пользователями, с внешней средой, интерфейсы операционных систем, сетевые протоколы, спецификации служб организации процессов, функционирующих поверх операционных систем;
- языки программирования и инструментальные средства, поддерживающие создание переносимых ПС и БД систем и средства программной инженерии CASE-системы;
- языки баз данных и системы управления базами данных;
- форматы данных, форматы внешних электронных сообщений;
- форматы переносимых электронных документов.

Задачи и процессы переноса программ и данных на другие платформы

Для плавного перехода в новую среду, пользователями параллельно могут выполняться работы в прежней и новой среде с соответствующими версиями ПС и БД. Сопроводитель должен выполнить следующие работы по обучению персонала:

- определить требования по обучению для реализации переноса;
- запланировать реализацию требований по обучению переноса;
- выполнить проверку результатов обучения после выполнения переноса;
- обновить и откорректировать планы обучения.

Ресурсы, для обеспечения сопровождения и мониторинга программных средств

При анализе затрат на сопровождение и мониторинг программных средств целесообразно рассматривать следующие сценарии:

- определение размера отдельных локальных модификаций программ и данных практически без учета взаимодействий с остальной частью версии программного продукта;
- совокупные затраты ресурсов на реализацию каждой модификации, при которой необходимо учитывать не только размер конкретного изменения, но и величину влияния на весь комплекс программ;
- оценивание интегральных затрат и совокупных размеров изменений при сопровождении и управлении конфигурацией ПС и БД в течение некоторого интервала времени (месяц, год) с учетом всего множества изменений.

Ресурсы, для обеспечения сопровождения и мониторинга программных средств

При обосновании необходимых ресурсов для сопровождения сложных ПС наибольшее значение имеют три ключевых фактора:

- размер – масштаб, подлежащих разработке полностью новых или модификаций программных компонентов;
- размер и относительная доля готовых программных компонентов, которые могут быть заимствованы из предшествовавших проектов и повторно использованы для модификаций в очередной версии программного продукта;
- относительные затраты ресурсов на создание модификаций и новых компонентов ПС с оцененным масштабом изменений: труда специалистов, времени, бюджета на единицу размера (на строку текста программ) или полные затраты на разработку всей новой версии ПС.

Ресурсы, для обеспечения сопровождения и мониторинга программных средств

При технико-экономическом обосновании сопровождения проекта ПС целесообразно применять методы и методики адекватные целям и этапам его реализации.

Базовый уровень изменений масштаба ПС должен обеспечивать:

- приемлемый для заказчика минимум дополнительных функций и требований к версии программного продукта;
- разумную вероятность успеха с точки зрения возможностей коллектива сопровождения и разработчиков модификаций за требуемое время.

Ресурсы, для обеспечения сопровождения и мониторинга программных средств

При системном анализе затраты на сопровождение можно считать аддитивными и включающими составляющие:

- затраты на обнаружение и устранение ошибок и дефектов в каждой версии ПС;
- затраты на доработку и совершенствование программ, формирование и испытание новых модернизированных версий ПС;
- затраты на тиражирование каждой новой версии и её внедрение в эксплуатируемых и новых системах.

Конфигурационное управление

План лекции

- Процессы управления конфигурацией программных средств
- Этапы и процедуры при управлении конфигурацией программных средств
- Технологическое обеспечение при сопровождении и управлении конфигурацией программных средств

Процессы управления конфигурацией программных средств

Концепция организации конфигурационного управления содержит:

- ожидаемую длительность поддержки развития и модификации конкретного проекта ПС;
- масштаб и уровень предполагаемых изменений и модификаций;
- возможное число и периодичность выпуска базовых версий ПС;
- организационные основы процессов сопровождения и конфигурационного управления ПС;
- требования к документированию изменений и базовых версий ПС;
- кто будет осуществлять управление конфигурацией - покупатель, разработчик или специальный персонал поддержки ЖЦ ПС.

Процессы управления конфигурацией программных средств

Управление конфигурацией включает действия и средства, позволяющие устанавливать категории, статус и личности руководителей, которые правомочны определять целесообразность и эффективность изменений, а также техническую реализуемость корректируемых версий с учетом ограничений бюджетов и сроков.

Процессы управления конфигурацией программных средств

Концепция конфигурационного управления конкретным проектом должна предусматривать возможность ***анализа изменений иерархической структуры - конфигурации,*** программных средств и их компонентов как сверху вниз, так и снизу вверх.

Средства и методы УК должны быть ориентированы на ***координированное развитие множества версий ПС и их компонентов.***

Процессы управления конфигурацией программных средств

Важной целью управления конфигурацией является документальное оформление и обеспечение полной наглядности текущей конфигурации программ и данных и степени выполнения требований к их функциональным характеристикам.

Процессы управления конфигурацией программных средств

Цель работ по идентификации конфигурации заключается в однозначной маркировке каждой единицы конфигурации (и ее последующих версий) для того, чтобы:

- установить базис для управления и ссылок на единицы конфигурации;
- выполнить идентификацию для каждой единицы конфигурации, для каждой отдельно управляемого компонента конфигурации и для комбинаций единиц конфигурации, которые составляют ПС;
- провести идентификация единиц конфигурации до начала реализации контроля изменений и прослеживаемости документов;

Процессы управления конфигурацией программных средств

- обеспечить идентификацию конфигурации для документов жизненного цикла ПС;
- выполнить идентификацию единицы конфигурации прежде, чем она будет использоваться другими процессами жизненного цикла ПС, для производства и для загрузки;
- исполняемый объектный код содержал идентификацию конфигурации, которая должна быть доступна для других компонентов системы.

Процессы управления конфигурацией программных средств

Конфигурационный учет составляют методы и средства регистрации и отслеживания состояния объектов - единиц конфигурации, накопления и классификации отчетов о всех реализованных и отвергнутых изменениях вариантов компонентов и ПС в целом.

В ключевых точках разработки проекта все объекты должны быть подвергнуты ***версионному учету***.

Этапы и процедуры при управлении конфигурацией программных средств

Конфигурационное управление в значительной степени обеспеченно, если ***ПС имеет четкую структуру***, а его компоненты - унифицированные интерфейсы по управлению и информации.

Множество процедур в стандартах сконструировано так, что возможна их ***адаптация в соответствии с характеристиками проектов и внешней среды ПС***.

Этапы и процедуры при управлении конфигурацией программных средств

Этапы и процедуры при управлении конфигурацией программных средств

Для ***установления достоверности*** сообщений пользователей о выявленных дефектах и ошибках необходима детальная регистрация сценариев и условий, при которых проявляются аномалии. Установление разработчиками достоверности ошибок начинается с тестирования эталонной базовой версии ПС при исходных данных m-го пользователя, обнаружившего дефект.

Этапы и процедуры при управлении конфигурацией программных средств

Описание выявленных дефектов и предложений по корректировке ПС и компонентов, содержащий исходные данные для планирования доработок в процессе сопровождения, поделенные в базе данных на крупные разделы:

- выявленные дефекты и ошибки в программах и данных;
- предложения по совершенствованию функций и улучшению качества эксплуатируемых версий программ;
- идеи и предполагаемая экономическая эффективность коренной модернизации и расширения функций ПС или его компонентов.

Этапы и процедуры при управлении конфигурацией программных средств

Приоритетность каждого предлагаемого изменения программ целесообразно оценивать по следующим критериям:

- насколько данное изменение может улучшить эксплуатационные характеристики ПС в целом;
- каковы затраты на выполнение корректировок при создании новой базовой версии и их распространение пользователям;
- возможно ли, и насколько сильно влияние изменений на функциональные характеристики остальных компонентов версии ПС;

Этапы и процедуры при управлении конфигурацией программных средств

- для какого числа пользователей может быть полезно данное изменение;
- как данное изменение отразится на эксплуатации пользователями предыдущих версий;
- насколько подготовка и оперативное внедрение данного изменения может отразиться на сроках создания очередной базовой версии.

Для повышения качества новых версий ***руководитель и совет конфигурационного управления*** анализируют все предлагаемые изменения и выделяют из них целесообразные для анализа возможного использования в очередной версии

Этапы и процедуры при управлении конфигурацией программных средств

Каждое разработанное изменение должно документироваться в ***описании и базе данных предполагаемых корректировок*** с указанием содержания, автора, времени и степени срочности.

Для принятых к внедрению изменений разрабатывается ***план доработок программ*** и определяется конкретный специалист, ответственный за каждую корректировку программы.

Этапы и процедуры при управлении конфигурацией программных средств

После выполнения доработок разработчиками компонентов и корректировки документации, следует окончательная проверка корректности изменений, которая осуществляется ***специалистами по квалификационному тестированию и испытаниям очередной версии ПС.***

Технологическое обеспечение при сопровождении и управлении конфигурацией программных средств

Технологической основой сопровождения и управления конфигурацией ПС являются ***системы управления базами данных*** (СУБД), адекватные целям и функциям проектов, структурированные по целям, назначению и содержанию данных в выделенных подсистемах хранения.

Технологическое обеспечение при сопровождении и управлении конфигурацией программных средств

Первоначально должен быть разработан ***проект архитектуры системы технологического обеспечения управления конфигурацией, а также Руководство по её применению***, настроена выбранная СУБД на управление основными взаимодействующими подсистемами базы данных, с учетом класса и масштаба предполагаемого проекта ПС.

Технологическое обеспечение при сопровождении и управлении конфигурацией программных средств

В жизненном цикле ПС характеристика качества практичности, доступности для понимания и освоения документации, должна использоваться и учитываться специалистами с двух позиций:

- ***разработчиками модификаций*** и версий комплексов программ, для которых необходимо адекватное отражение и восприятие в документах состояния и изменений ПС и их компонентов в процессе сопровождения и управления конфигурацией;
- ***пользователями измененных и утвержденных документов***, поставляемых базовых версий программных продуктов в процессе их применения и эксплуатации.

Технологическое обеспечение при сопровождении и управлении конфигурацией программных средств

Для управления конфигурацией, развитием комплекса программ и испытаний в базе данных документов должны собираться сведения, которые состоят из следующих крупных ***функциональных частей***:

- описания данных об отказах, дефектах и ошибках, условиях их проявления и характеристиках обнаруживающих тестов, а также предложения на изменения программ, подлежащие анализу и селекции для выделения тех из них, для которых будут разрабатываться корректировки программ – ***описания предполагаемых изменений*** ПС;
- разработанные изменения программ, отобранные аналитиками сопровождения и конфигурационного управления для проведения корректировок в очередной версии ПС – ***описания подготовленных и утвержденных корректировок*** компонентов и версий комплексов программ;

Технологическое обеспечение при сопровождении и управлении конфигурацией программных средств

- сборки компонентов, откорректированные версии и наборы изменений, выполненных в каждой из них – ***описания откорректированных и утвержденных базовых версий программного продукта***;
- характеристики и параметры пользователей, которым переданы для использования соответствующие базовые версии и особенности внешней среды эксплуатации у них – ***описания параметров адаптации пользовательских версий программного продукта***.

Технологическое обеспечение при сопровождении и управлении конфигурацией программных средств

Отчеты специалистов о выявленных дефектах и предложениях по корректировке ПС, которые должны содержать:

- подробное описание сценария тестирования и исходных данных, при которых выявлен дефект;
- описание проявления дефекта и документы результатов его регистрации;
- предположение о возможной причине, вызвавшей проявление дефекта;
- предложение о возможной модификации ПС и его компонентов для устранения дефекта или совершенствования качества функционирования комплекса программ.

Технологическое обеспечение при сопровождении и управлении конфигурацией программных средств

Описание выявленных дефектов и предложений по совершенствованию функций комплекса программ, а также результатов анализа предполагаемых корректировок версий ПС должен содержать отчеты пользователей о выявленных дефектах и предложениях и дополнительно:

- оценки сложности, трудоемкости, эффективности и срочности модификаций программ;
- оценки возможного влияния предлагаемых изменений на эксплуатацию версий ПС, имеющихся у пользователей.

Технологическое обеспечение при сопровождении и управлении конфигурацией программных средств

Описание подготовленных и утвержденных корректировок, а также реализованных изменений и обобщенных характеристик модифицированной базовой версии программного продукта должно содержать:

- причину изменения программ и базы данных (ошибка, дефект, совершенствование);
- содержание изменений программ и базы данных, а также документации на версию ПС или компонента;
- результаты квалификационного тестирования базовой версии программного продукта с предполагаемыми изменениями;

Технологическое обеспечение при сопровождении и управлении конфигурацией программных средств

- результаты испытаний и обобщенные характеристики качества базовой версии программного продукта после внесения изменений;
- решение по распространению пользователям проведенной модификации или версии программного продукта;
- адрес хранения корректировок, документов и квалификационных тестов новой базовой версии программного продукта.

Технологическое обеспечение при сопровождении и управлении конфигурацией программных средств

Особое значение при сопровождении и управлении конфигурацией имеет ***документация на реализованные изменения и тесты,*** с помощью которых проверялась корректность версий компонентов и ПС в целом. Эта документация должна позволять восстанавливать ***историю разработки и проверки*** каждого изменения любого компонента.

Введение в шаблоны проектирования

Литература

- Гамма, Э. Приемы объектно-ориентированного проектирования. Паттерны программирования. / Э. Гамма, Р. Хелм, Р. Джонсон, Дж.Влиссидес
- Фримен, Эрик. Паттерны проектирования. / Эрик Фримен, Элизабет Фримен

План лекции

- Понятие шаблона проектирования
- Типы шаблонов проектирования

Понятие шаблона проектирования

Шаблон проектирования или **паттерн** — повторяемая архитектурная конструкция, представляющая собой решение проблемы проектирования в рамках некоторого часто возникающего контекста.

Любой паттерн описывает задачу, которая снова и снова возникает в нашей работе, а также принцип ее решения, причем таким образом, что это решение можно потом использовать миллион раз, ничего не изобретая заново.

Понятие шаблона проектирования

В общем случае паттерн состоит из четырех основных элементов:

- *Имя.* Сославшись на него, мы можем сразу описать проблему проектирования; ее решения и их последствия. Присваивание паттернам имен позволяет проектировать на более высоком уровне абстракции.
- *Задача.* Описание того, когда следует применять паттерн.
- Решение. Описание элементов дизайна, отношений между ними, функций каждого элемента.
- Результаты - это следствия применения паттерна и разного рода компромиссы.

Типы шаблонов проектирования

Уровень\Цель	Порождающие паттерны	Структурные паттерны	Паттерны поведения
Класс	Фабричный метод	Адаптер (класса)	Интерпретатор Шаблонный метод
Объект	Абстрактная фабрика Одиночка Прототип Строитель	Адаптер (объекта) Декоратор Заместитель Компоновщик Мост Приспособленец Фасад	Итератор Команда Наблюдатель Посетитель Посредник Состояние Стратегия Хранитель Цепочка обязанностей

Типы шаблонов проектирования

Назначение	Паттерн проектирования	Аспекты, которые можно изменять
Порождающие паттерны	Абстрактная фабрика	Семейства порождаемых объектов
	Одиночка	Единственный экземпляр класса
	Прототип	Класс, из которого инстанцируется объект
	Строитель	Способ создания составного объекта
	Фабричный метод	Инстанцируемый подкласс объекта

Типы шаблонов проектирования

Назначение	Паттерн проектирования	Аспекты, которые можно изменять
Структурные паттерны	Адаптер	Интерфейс к объекту
	Декоратор	Обязанности объекта без порождения подкласса
	Заместитель	Способ доступа к объекту, его местоположение
	Компоновщик	Структура и состав объекта
	Мост	Реализация объекта
	Приспособленец	Накладные расходы на хранение объектов
	Фасад	Интерфейс к подсистеме

Типы шаблонов проектирования

Назначение	Паттерн проектирования	Аспекты, которые можно изменять
Паттерны поведения	Интерпретатор	Грамматика и интерпретация языка
	Итератор	Способ обхода элементов агрегата
	Команда	Время и способ выполнения запроса
	Наблюдатель	Множество объектов, зависящих от другого объекта;
	Посетитель	Операции, которые можно применить к объекту или объектам
	Посредник	Объекты, взаимодействующие между собой, и способ их кооп-ций
	Состояние	Состояние объекта
	Стратегия	Алгоритм
	Хранитель	Закрытая информация, хранящаяся вне объекта, и время ее сохран.
	Цепочка обязанностей	Объект, выполняющий запрос
	Шаблонный метод	Шаги алгоритма

Основные шаблоны проектирования

План лекции

- Основные шаблоны проектирования
- Шаблон делегирования

Основные шаблоны проектирования

Название	Описание
Шаблон Делегирования	Объект внешне выражает некоторое поведение, но в реальности передаёт ответственность за выполнение этого поведения связанному объекту.
Шаблон функционального дизайна	Гарантирует, что каждый модуль компьютерной программы имеет только одну обязанность и исполняет её с минимумом побочных эффектов на другие части программы.
Неизменяемый интерфейс	Создание неизменяемого объекта.
Интерфейс	Общий метод для структурирования компьютерных программ для того, чтобы их было проще понять.

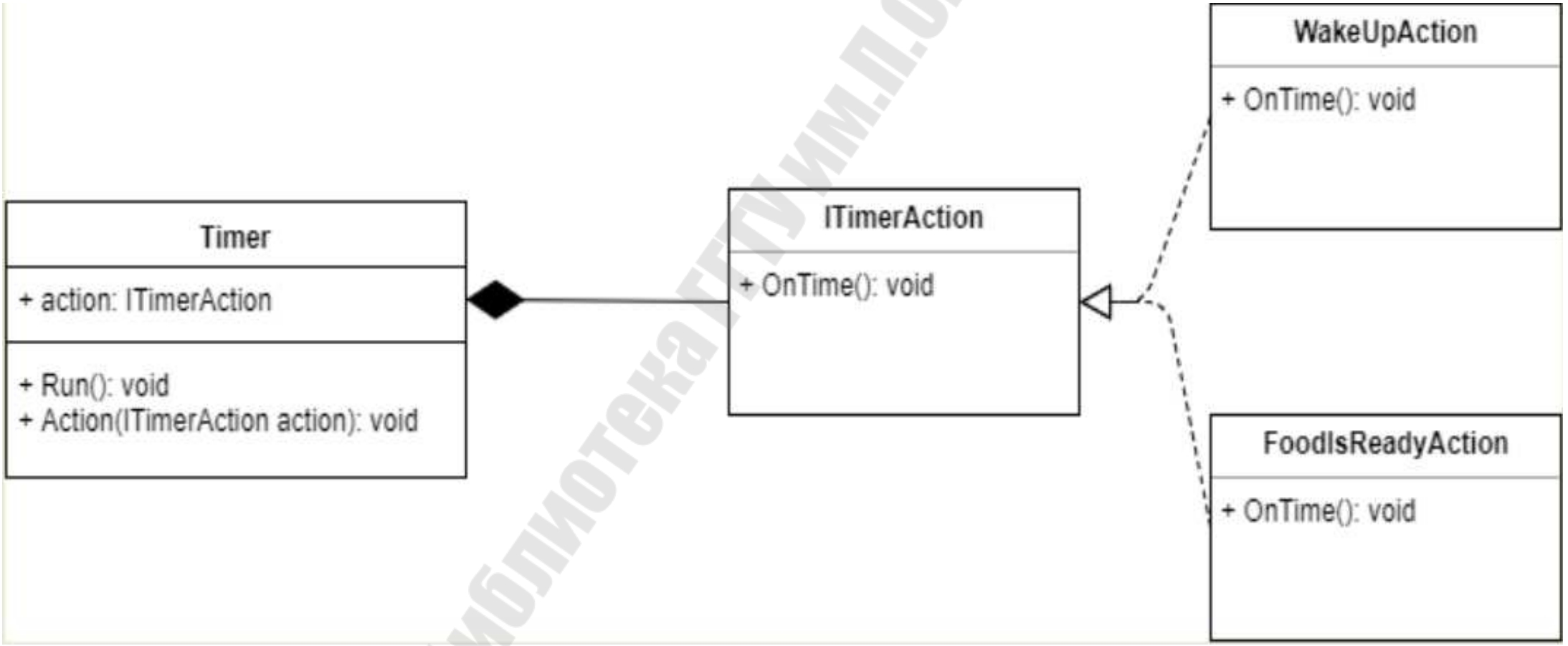
Основные шаблоны проектирования

Название	Описание
Интерфейс-маркер	В качестве атрибута (как пометки объектной сущности) применяется наличие или отсутствие реализации интерфейса-маркера. В современных языках программирования вместо этого могут применяться атрибуты или аннотации.
Контейнер свойств	Позволяет добавлять дополнительные свойства для класса в контейнер (внутри класса), вместо расширения класса новыми свойствами.
Канал событий	Расширяет шаблон Publish/Subscribe, создавая централизованный канал для событий. Использует объект-представитель для подписки и объект-представитель для публикации события в канале.

Шаблон делегирования

Делегирование (англ. *Delegation*) — основной шаблон проектирования, в котором объект внешне выражает некоторое поведение, но в реальности передаёт ответственность за выполнение этого поведения связанному объекту.

Шаблон делегирования



Порождающие шаблоны

План лекции

- Виды порождающих шаблонов
- Строитель (Builder)
- Фабричный метод (Factory method)
- Одиночка (Singleton)
- Абстрактная фабрика (Abstract factory)

Виды порождающих шаблонов

Название	Оригинальное название	Описание
Абстрактная фабрика	Abstract Factory	Класс, который представляет собой интерфейс для создания компонентов системы.
Строитель	Builder	Класс, который представляет собой интерфейс для создания сложного объекта.
Фабричный метод	Factory method	Определяет интерфейс для создания объекта, но оставляет подклассам решение о том, какой класс инстанцировать.
Отложенная инициация	Lazy initialization	Объект, инициализируемый во время первого обращения к нему.
Пул одиночек	Multiton	Гарантирует, что класс имеет поименованные экземпляры объекта и обеспечивает глобальную точку доступа к ним.

Виды порождающих шаблонов

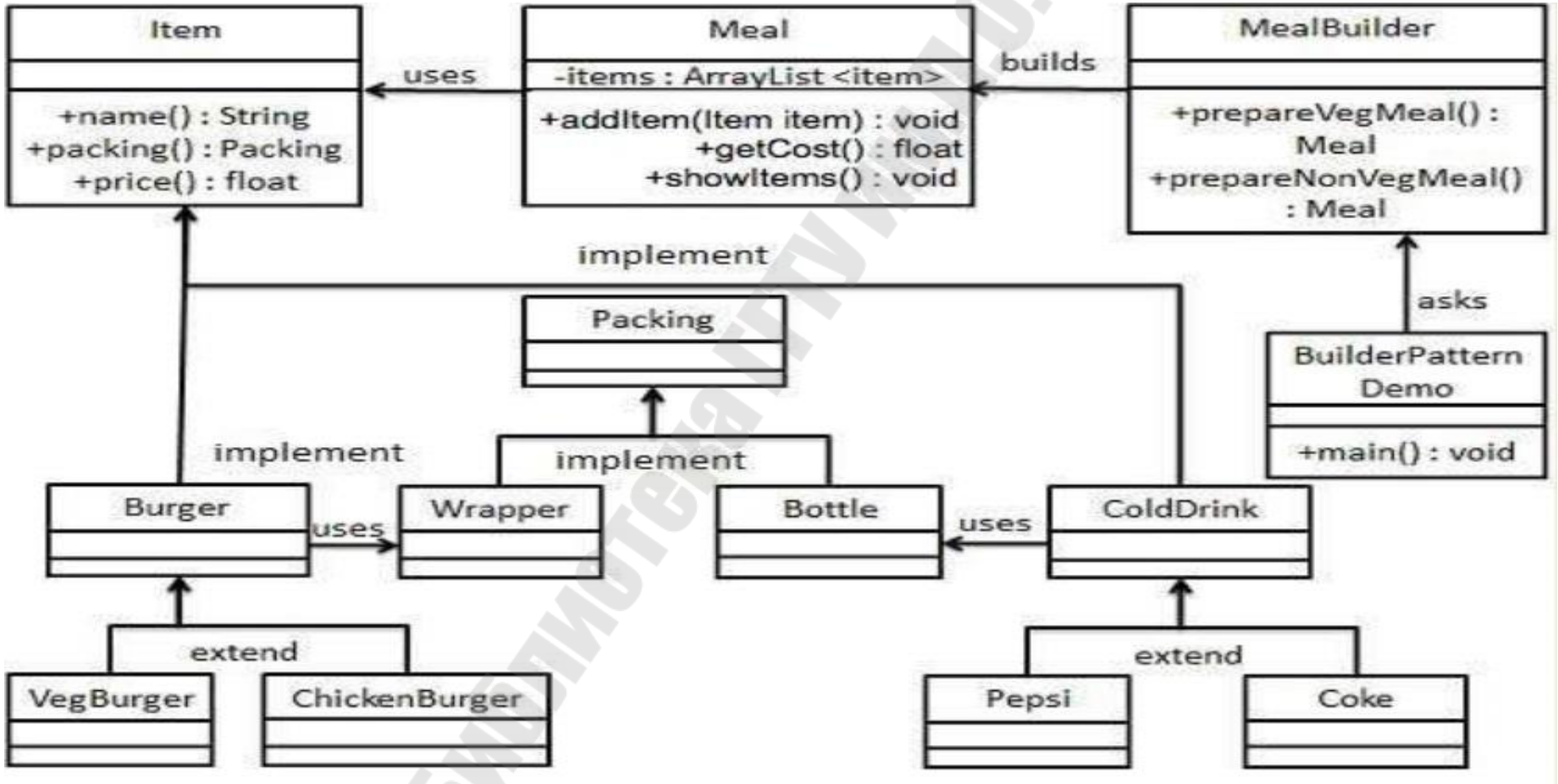
Название	Оригинальное название	Описание
Объектный пул	Object pool	Класс, который представляет собой интерфейс для работы с набором инициализированных и готовых к использованию объектов.
Прототип	Prototype	Определяет интерфейс создания объекта через клонирование другого объекта вместо создания через конструктор.
Получение ресурса есть инициализация	Resource acquisition is initialization (RAII)	Получение некоторого ресурса совмещается с инициализацией, а освобождение — с уничтожением объекта.
Одиночка	Singleton	Класс, который может иметь только один экземпляр.

Строитель (Builder)

Строитель (*Builder*) — порождающий шаблон проектирования предоставляет способ создания составного объекта.

Цель. Отделяет конструирование сложного объекта от его представления, так что в результате одного и того же процесса конструирования могут получаться разные представления.

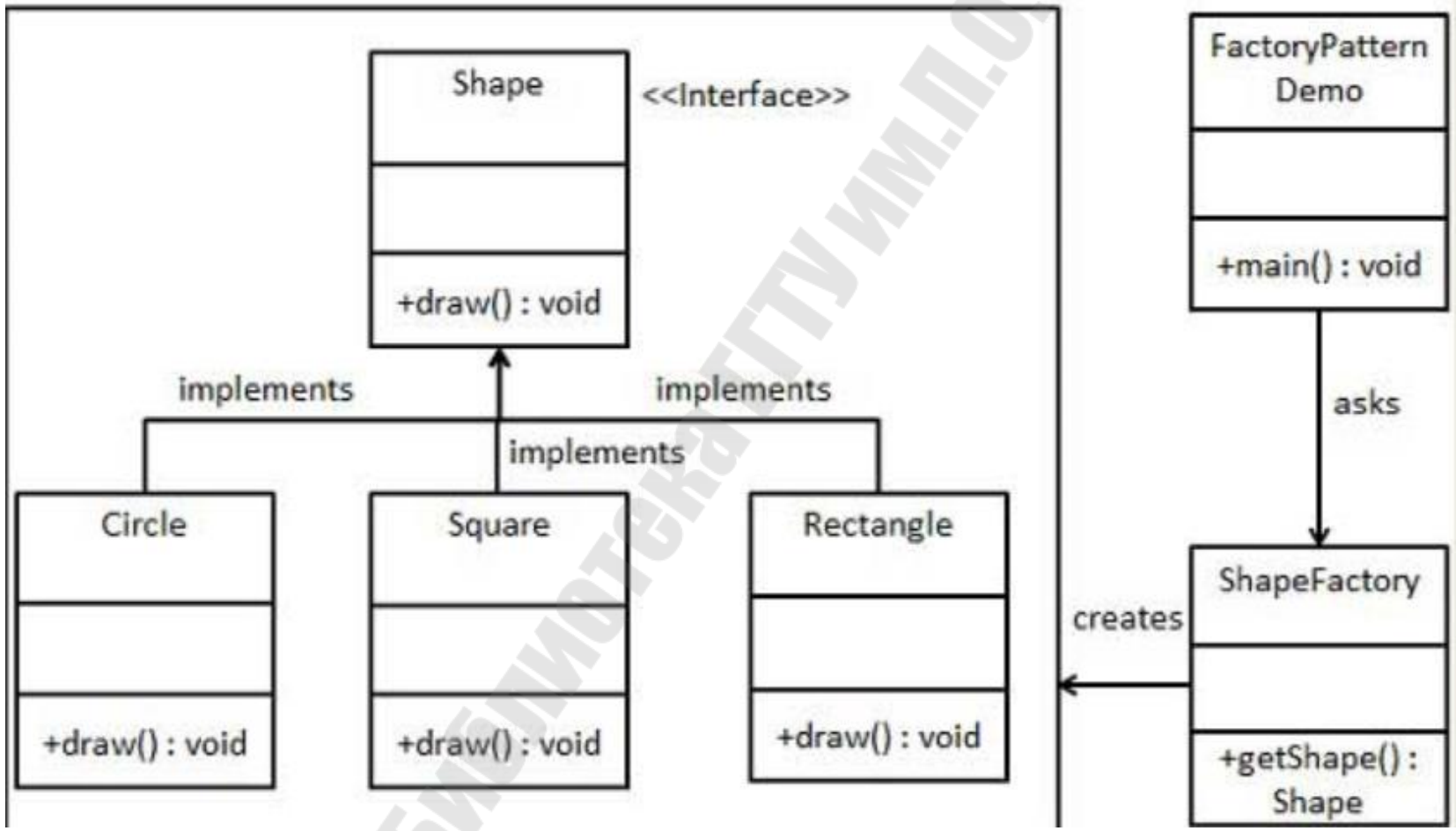
Строитель (Builder)



Фабричный метод (Factory method)

Фабричный метод (англ. *Factory Method*) —порождающий шаблон проектирования, предоставляющий подклассам интерфейс для создания экземпляров некоторого класса. В момент создания наследники могут определить, какой класс создавать. Иными словами, данный шаблон делегирует создание объектов наследникам родительского класса. Это позволяет использовать в коде программы не специфические классы, а манипулировать абстрактными объектами на более высоком уровне.

Фабричный метод (Factory method)

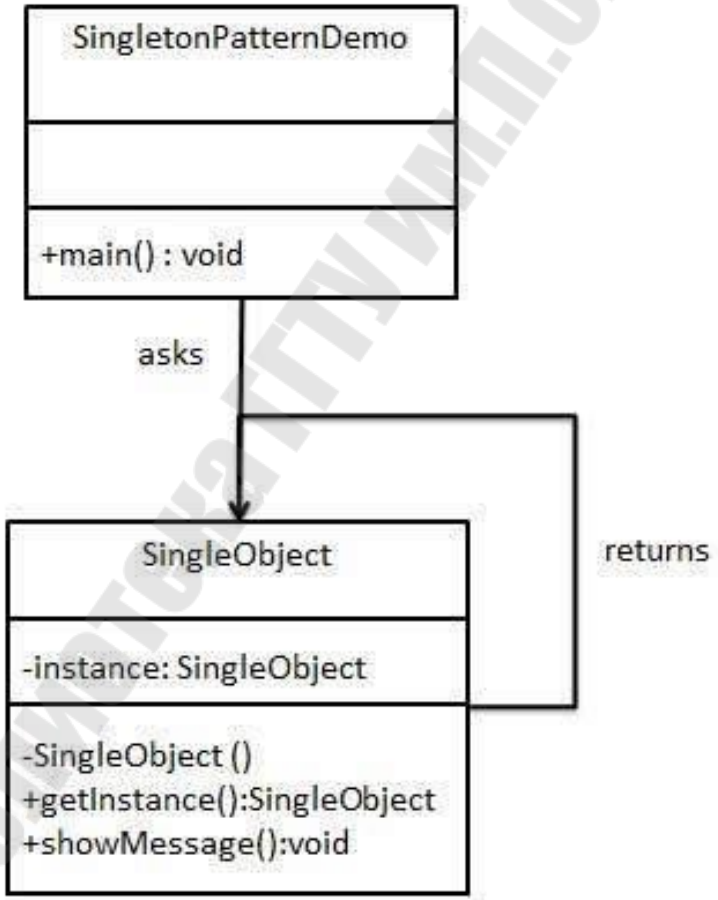


Одиночка (Singleton)

Одиночка (*Singleton*) — порождающий шаблон проектирования, гарантирующий, что в однопроцессном приложении будет единственный экземпляр некоторого класса, и предоставляющий глобальную точку доступа к этому экземпляру.

Цель. У класса есть только один экземпляр, и он предоставляет к нему глобальную точку доступа. Существенно то, что можно пользоваться именно *экземпляром* класса, так как при этом во многих случаях становится доступной более широкая функциональность.

Одиночка (Singleton)



Абстрактная фабрика

Абстрактная фабрика (англ. *Abstract factory*) — порождающий шаблон проектирования, предоставляет интерфейс для создания семейств взаимосвязанных или взаимозависимых объектов, не специфицируя их конкретных классов.

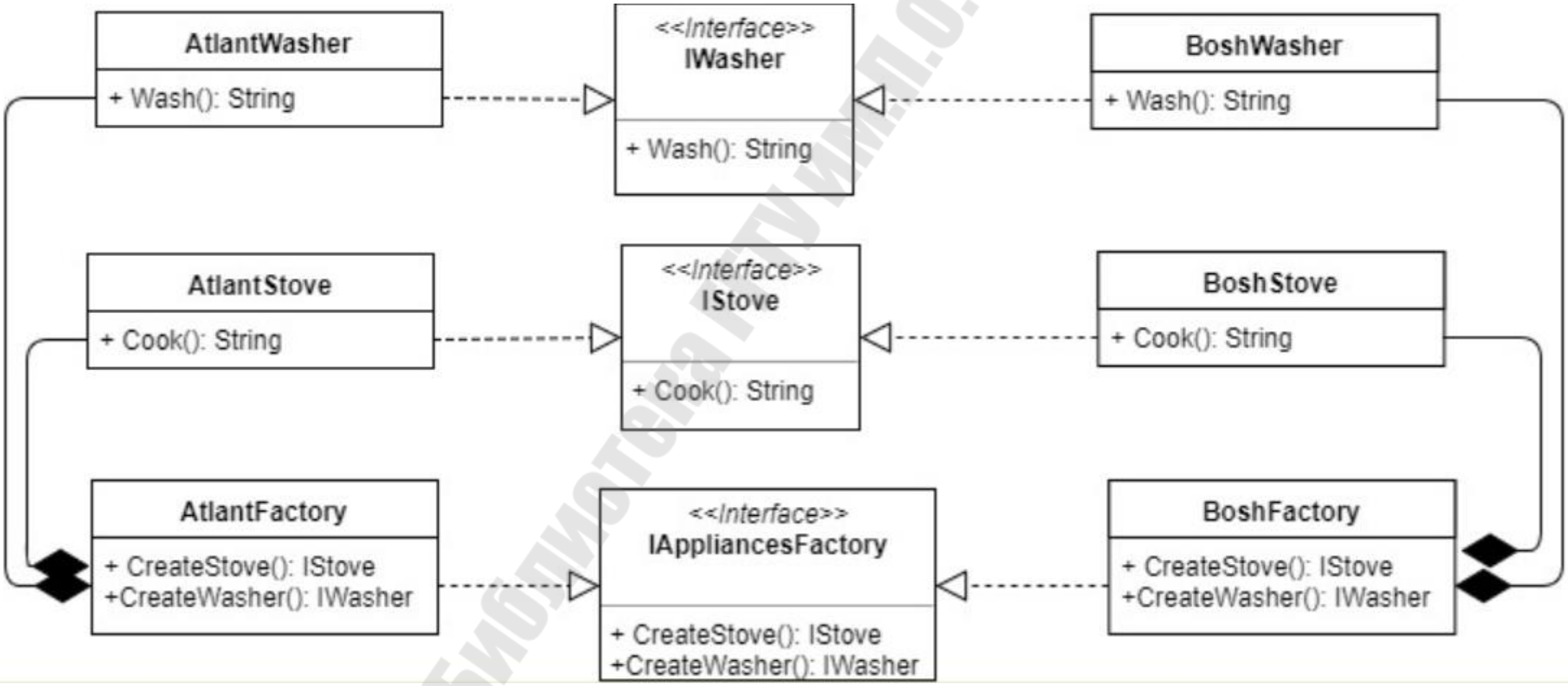
Плюсы

- изолирует конкретные классы;
- упрощает замену семейств продуктов;
- гарантирует сочетаемость продуктов.

Минусы

- сложно добавить поддержку нового вида продуктов.

Абстрактная фабрика



Структурные шаблоны

План лекции

- Основные структурные шаблоны
- Декоратор
- Компоновщик
- Адаптер
- Фасад

Основные структурные шаблоны

Структурные шаблоны (*Structural*) определяют различные сложные структуры, которые изменяют интерфейс уже существующих объектов или его реализацию, позволяя облегчить разработку и оптимизировать программу.

Основные структурные шаблоны

Название	Оригинальное название	Описание
Адаптер	Adapter	Объект, обеспечивающий взаимодействие двух других объектов, один из которых использует, а другой предоставляет несовместимый с первым интерфейс.
Мост	Bridge	Структура, позволяющая изменять интерфейс обращения и интерфейс реализации класса независимо.
Компоновщик	Composite	Объект, который объединяет в себе объекты, подобные ему самому.
Декоратор или Wrapper	Decorator	Класс, расширяющий функциональность другого класса без использования наследования.

Основные структурные шаблоны

Название	Оригинальное название	Описание
Фасад	Facade	Объект, который абстрагирует работу с несколькими классами, объединяя их в единое целое.
Единая точка входа	Front controller	Обеспечивает унифицированный интерфейс для интерфейсов в подсистеме. Front Controller определяет высокоуровневый интерфейс, упрощающий использование подсистемы.
Приспособленец	Flyweight	Это объект, представляющий себя как уникальный экземпляр в разных местах программы, но фактически не являющийся таковым.
Заместитель	Proxy	Объект, который является посредником между двумя другими объектами, и который реализует/ограничивает доступ к объекту, к которому обращаются через него.

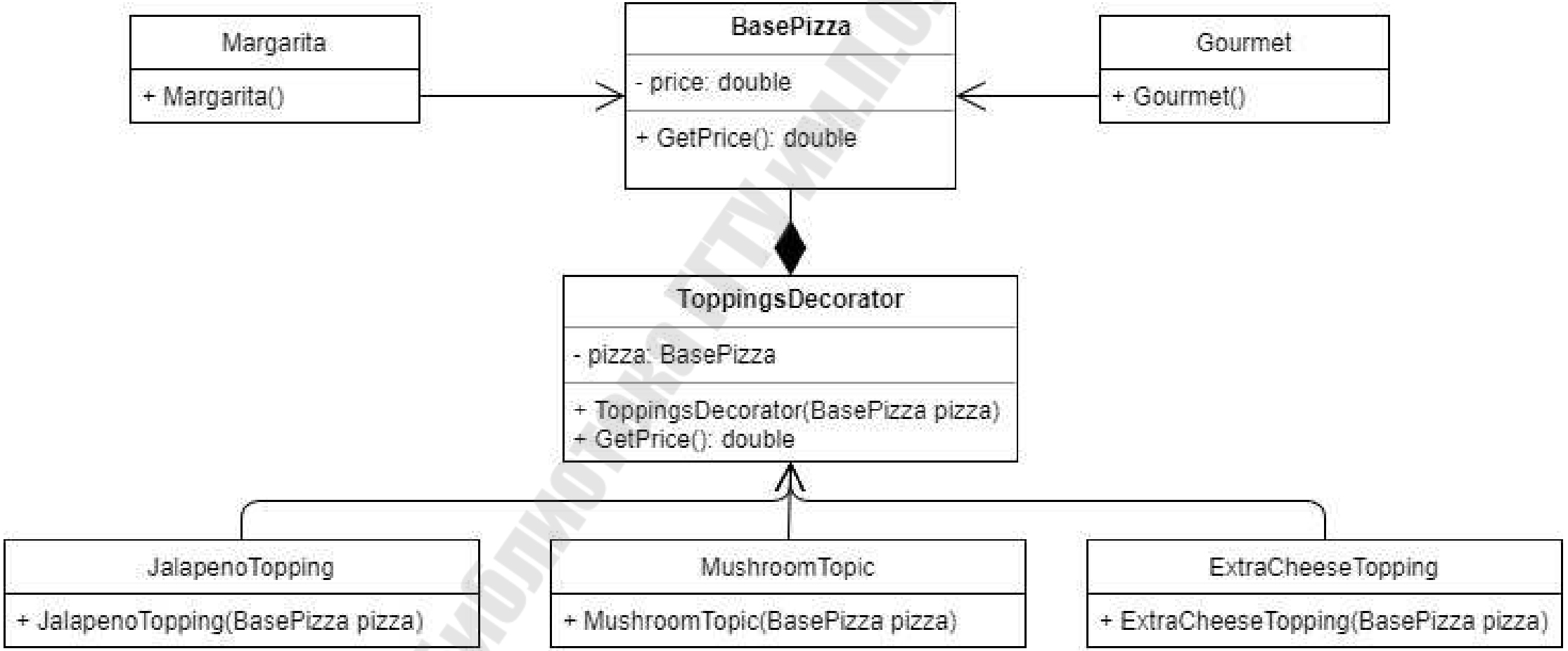
Декоратор

Декоратор (англ. *Decorator*) — структурный шаблон проектирования, предназначенный для динамического подключения дополнительного поведения к объекту.

Цель

Объект, который предполагается использовать, выполняет основные функции. Однако может потребоваться добавить к нему некоторую дополнительную функциональность, которая будет выполняться до, после или даже вместо основной функциональности объекта.

Декоратор



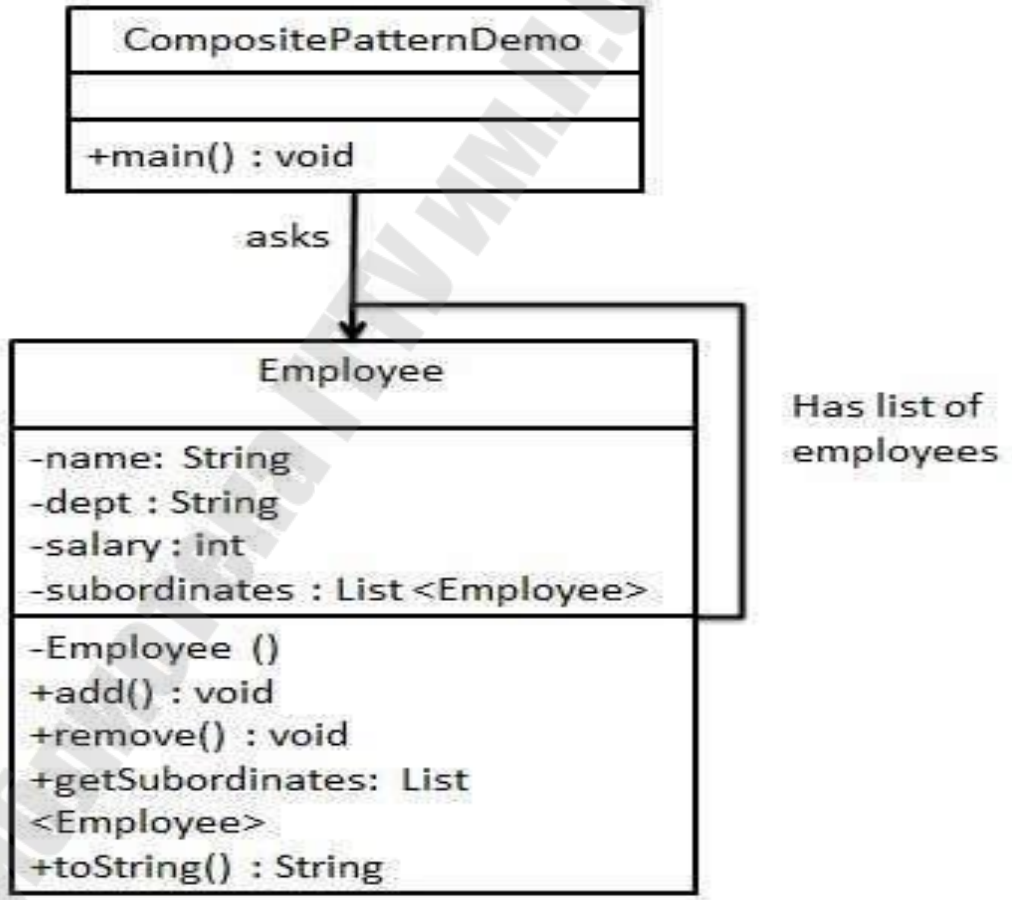
Компоновщик

Компоновщик (англ. *Composite pattern*) — структурный шаблон проектирования, объединяющий объекты в древовидную структуру для представления иерархии от частного к целому. Компоновщик позволяет клиентам обращаться к отдельным объектам и к группам объектов одинаково.

Цель

Паттерн определяет иерархию классов, которые одновременно могут состоять из примитивных и сложных объектов, упрощает архитектуру клиента, делает процесс добавления новых видов объекта более простым.

Компоновщик



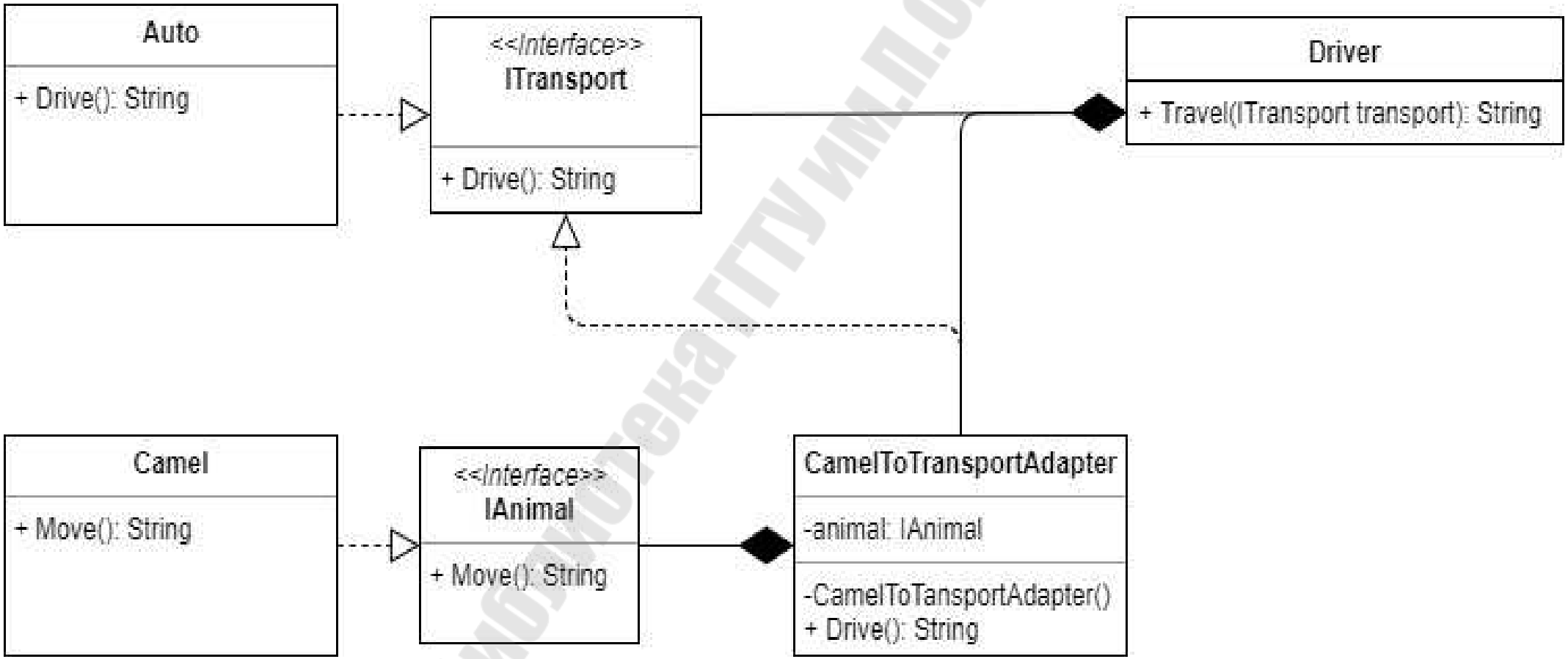
Адаптер

Адаптер (англ. *Adapter*) — структурный шаблон проектирования, предназначенный для организации использования функций объекта, недоступного для модификации, через специально созданный интерфейс.

Задача. Система поддерживает требуемые данные и поведение, но имеет неподходящий интерфейс.

Способ решения. Адаптер предусматривает создание класса-оболочки с требуемым интерфейсом.

Адаптер

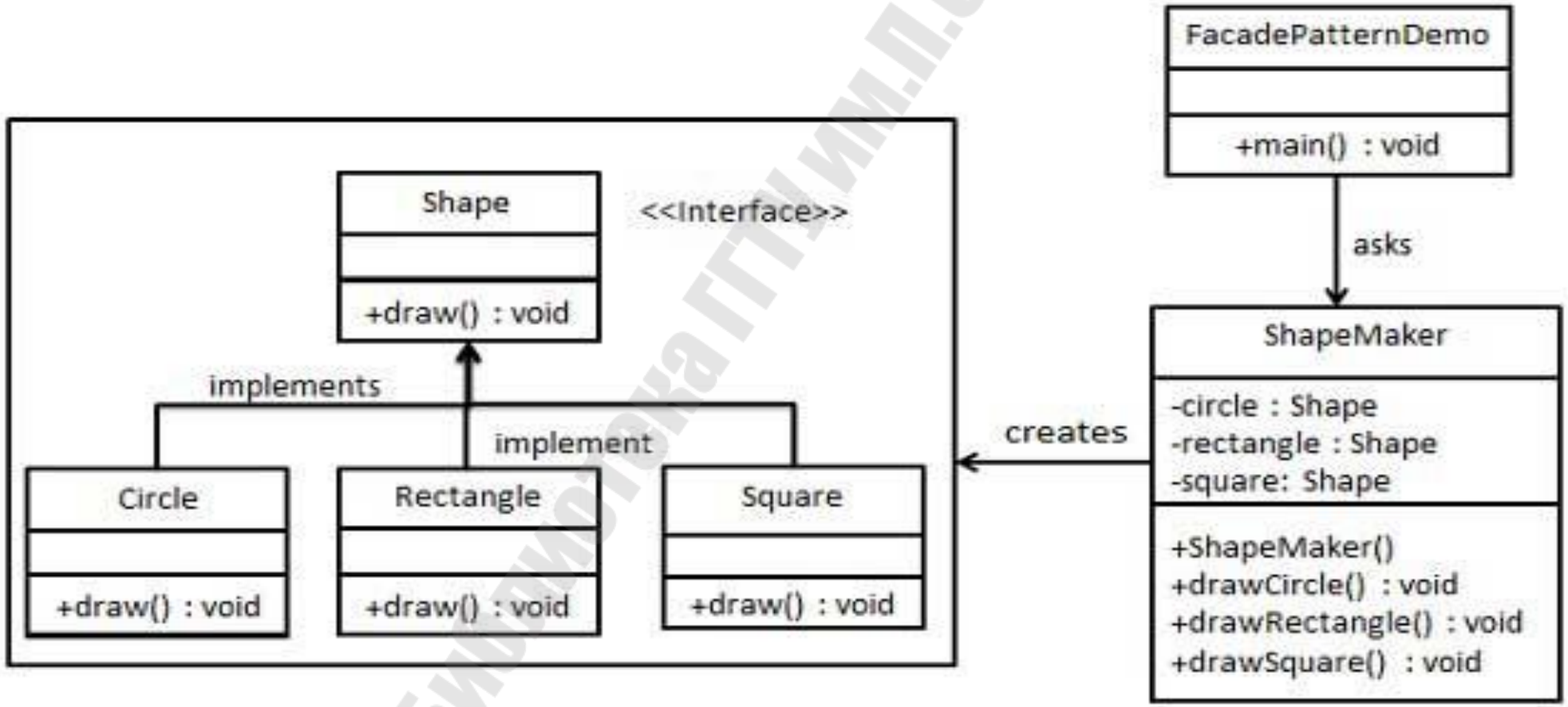


Фасад

Фасад (англ. *Facade*) — структурный шаблон проектирования, позволяющий скрыть сложность системы путём сведения всех возможных внешних вызовов к одному объекту, делегирующему их соответствующим объектам системы.

Особенности применения. Шаблон применяется для установки некоторого рода политики по отношению к другой группе объектов. Если политика должна быть яркой и заметной, следует воспользоваться услугами шаблона Фасад. Если же необходимо обеспечить скрытность и аккуратность (прозрачность), более подходящим выбором является шаблон Заместитель (Proxy).

Фасад



Поведенческие шаблоны

План лекции

- Основные поведенческие шаблоны
- Интерпретатор
- Команда
- Наблюдатель

Основные поведенческие шаблоны

Поведенческие шаблоны (англ. *behavioral patterns*) — шаблоны проектирования, определяющие алгоритмы и способы реализации взаимодействия различных объектов и классов.

Основные поведенческие шаблоны

Название	Оригинальное название	Описание
Цепочка обязанностей	Chain of responsibility	Предназначен для организации в системе уровней ответственности.
Команда	Command	Представляет действие. Объект команды заключает в себе само действие и его параметры
Интерпретатор	Interpreter	Решает часто встречающуюся, но подверженную изменениям, задачу.
Итератор	Iterator	Представляет собой объект, позволяющий получить последовательный доступ к элементам объекта-агрегата без использования описаний каждого из объектов, входящих в состав агрегации.

Основные поведенческие шаблоны

Название	Оригинальное название	Описание
Посредник	Mediator	Обеспечивает взаимодействие множества объектов, формируя при этом слабую связанность и избавляя объекты от необходимости явно ссылаться друг на друга.
Хранитель	Memento	Позволяет не нарушая инкапсуляцию зафиксировать и сохранить внутренние состояния объекта так, чтобы позднее восстановить его в этих состояниях.
Наблюдатель	Observer	Определяет зависимость типа «один ко многим» между объектами таким образом, что при изменении состояния одного объекта все зависящие от него оповещаются об этом событии.
Состояние	State	Используется в тех случаях, когда во время выполнения программы объект должен менять своё поведение в зависимости от своего состояния.

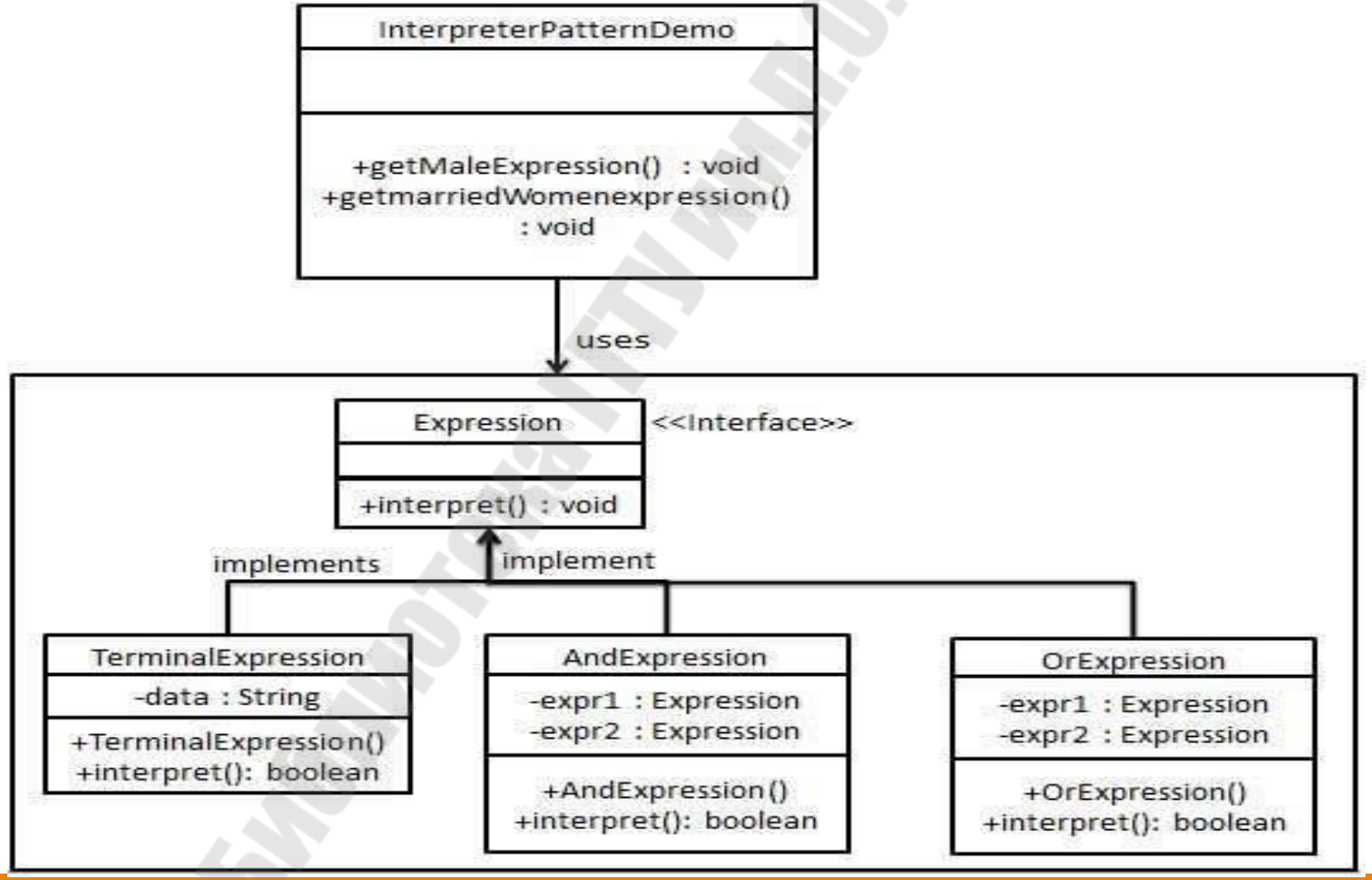
Интерпретатор

Интерпретатор (англ. *Interpreter*) — поведенческий шаблон проектирования, решающий часто встречающуюся, но подверженную изменениям, задачу. Также известен как Little (Small) Language.

Проблема. Имеется часто встречающаяся, подверженная изменениям задача.

Решение. Создать интерпретатор, который решает данную задачу.

Интерпретатор

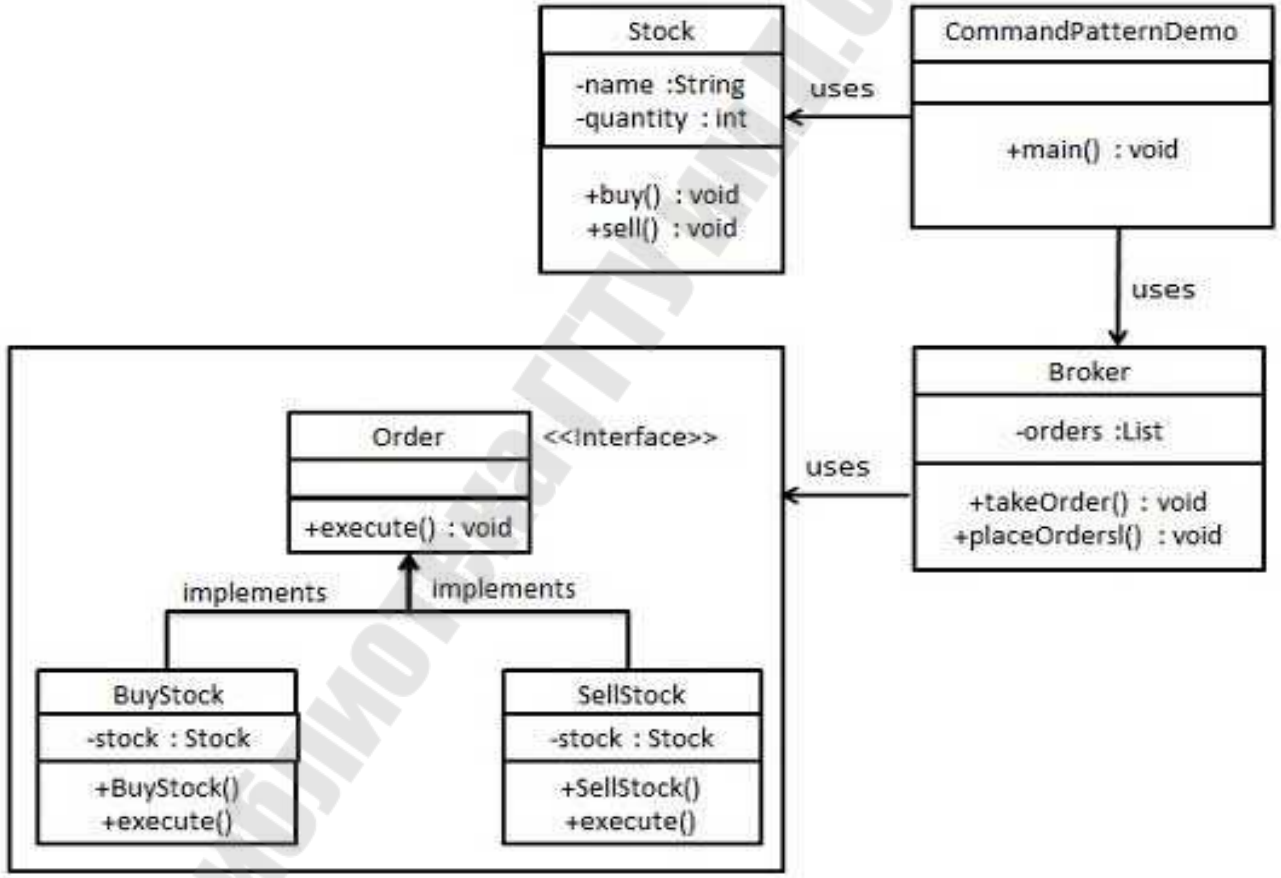


Команда

Команда (англ. *Command*) — поведенческий шаблон проектирования, используемый при объектно-ориентированном программировании, представляющий действие. Объект команды заключает в себе само действие и его параметры.

Цель. Создание структуры, в которой класс-отправитель и класс-получатель не зависят друг от друга напрямую. Организация обратного вызова к классу, который включает в себя класс-отправитель.

Команда

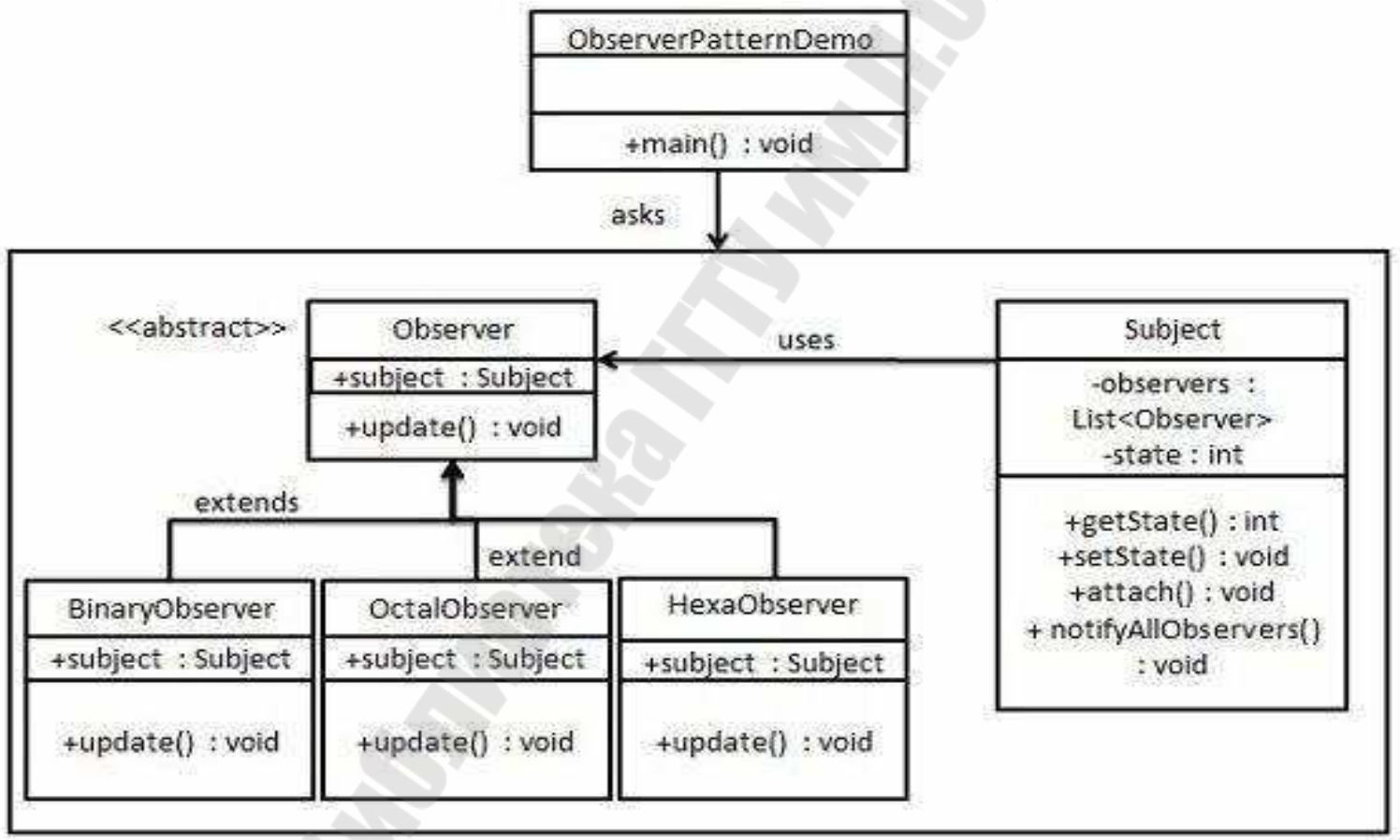


Наблюдатель

Наблюдатель (англ. *Observer*) — поведенческий шаблон проектирования. Также известен как «подчинённые» (Dependents). Создает механизм у класса, который позволяет получать экземпляру объекта этого класса оповещения от других объектов об изменении их состояния, тем самым наблюдая за ними.

Назначение. Определяет зависимость типа «один ко многим» между объектами таким образом, что при изменении состояния одного объекта все зависящие от него оповещаются об этом событии.

Наблюдатель



Архитектурные шаблоны проектирования

План

- Архитектурные шаблоны проектирования
- Шаблон проектирования Model View Controller (MVC)

Архитектурные шаблоны проектирования

Архитектурный шаблон проектирования является общим многоразовым решением общей проблемы в архитектуре программного обеспечения в рамках данного контекста.

Архитектурные шаблоны аналогичны шаблонам проектирования программного обеспечения, но имеют более широкий охват.

Архитектурные шаблоны затрагивают различные проблемы разработки программного обеспечения, такие как ограничения производительности аппаратного обеспечения, высокая доступность и минимизация бизнес-рисков. Некоторые архитектурные шаблоны были реализованы в рамках программного обеспечения.

Архитектурные шаблоны проектирования

Название	Описание
Model View Controller (MVC)	Схема использования нескольких шаблонов проектирования, с помощью которых модель данных приложения, пользовательский интерфейс и взаимодействие с пользователем разделены на три отдельных компонента таким образом, чтобы модификация одного из компонентов оказывала минимальное воздействие на остальные.
Model View View-Model (MVVM)	MVVM удобно использовать вместо классического MVC и ему подобных в тех случаях, когда в платформе, на которой ведётся разработка, присутствует «связывание данных». В шаблонах проектирования MVC/MVP изменения в пользовательском интерфейсе не влияют непосредственно на Модель, а предварительно идут через Контроллер или Presenter.

Архитектурные шаблоны проектирования

Название	Описание
Model View Presenter (MVP)	Шаблон проектирования, производный от MVC, который используется в основном для построения пользовательского интерфейса. Элемент Presenter в данном шаблоне берёт на себя функциональность посредника (аналогично контроллеру в MVC) и отвечает за управление событиями пользовательского интерфейса (например, использование мыши) так же, как в других шаблонах обычно отвечает представление.

Model View Controller

Model View Controller (**MVC**, «Модель-Представление-Контроллер», «Модель-Вид-Контроллер») — схема разделения данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: модель, представление и контроллер — таким образом, что модификация каждого компонента может осуществляться независимо.

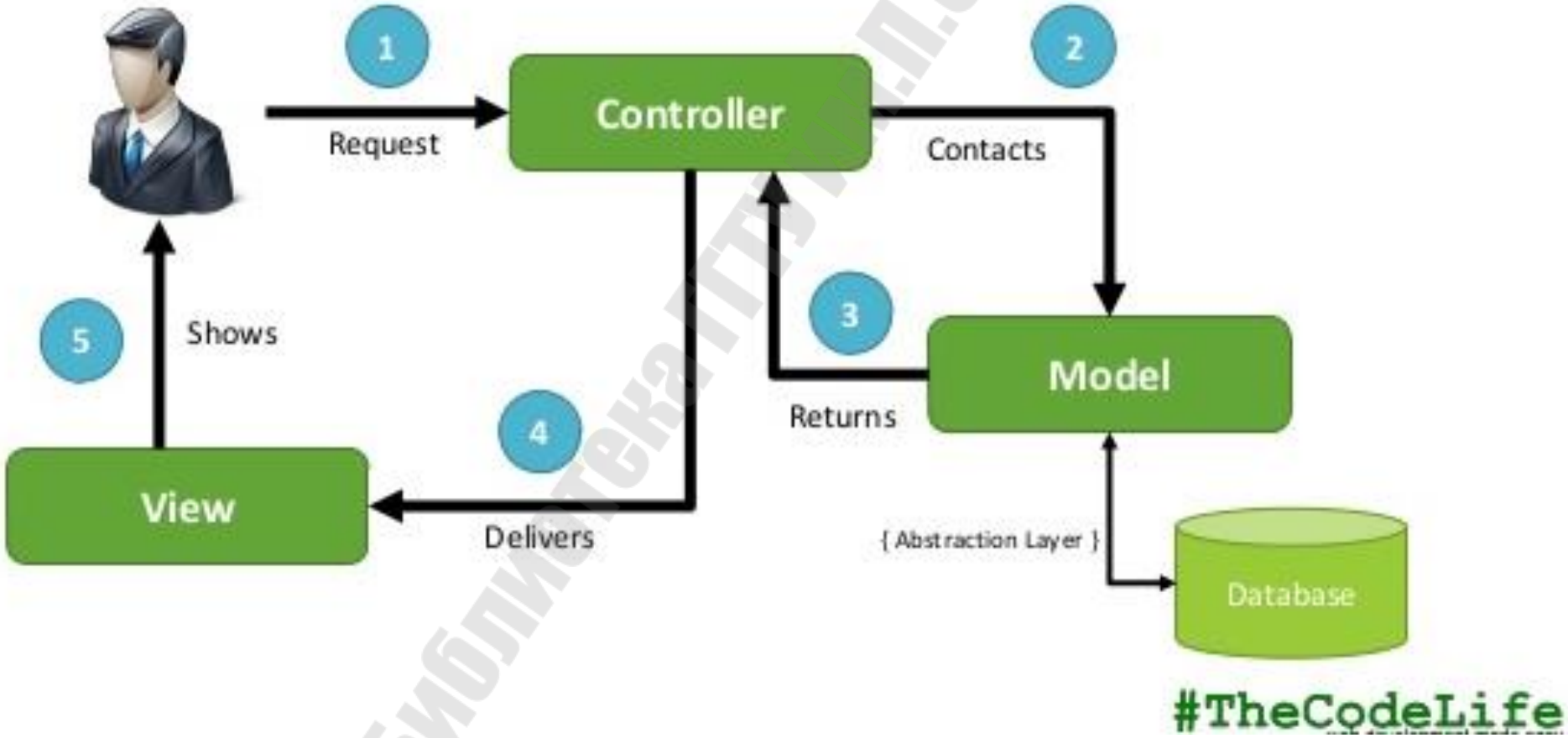
Model View Controller

Модель (*Model*) предоставляет данные и реагирует на команды контроллера, изменяя свое состояние.

Представление (*View*) отвечает за отображение данных модели пользователю, реагируя на изменения модели.

Контроллер (*Controller*) интерпретирует действия пользователя, оповещая модель о необходимости изменений.

Model View Controller



Архитектурный шаблон проектирования. MVVM

MVVM

MVVM (Model-View-ViewModel) позволяет отделить логику приложения от визуальной части (представления).

Model

Модель описывает используемые в приложении данные. Модели могут содержать логику, непосредственно связанную этими данными, например, логику валидации свойств модели. В то же время модель не должна содержать никакой логики, связанной с отображением данных и взаимодействием с визуальными элементами управления.

MVVM

View

View или представление определяет визуальный интерфейс, через который пользователь взаимодействует с приложением. Применительно к WPF представление - это код в xaml, который определяет интерфейс в виде кнопок, текстовых полей и прочих визуальных элементов.

Хотя окно (класс Window) в WPF может содержать как интерфейс в xaml, так и привязанный к нему код C#, однако в идеале код C# не должен содержать какой-то логики, кроме разве что конструктора, который вызывает метод InitializeComponent и выполняет начальную инициализацию окна. Вся же основная логика приложения выносится в компонент ViewModel.

MVVM

ViewModel

ViewModel или модель представления связывает модель и представление через механизм привязки данных. ViewModel также содержит логику по получению данных из модели, которые потом передаются в представление. И также VewModel определяет логику по обновлению данных в модели.

Поскольку элементы представления, то есть визуальные компоненты типа кнопок, не используют события, то представление взаимодействует с ViewModel посредством команд.

Например, пользователь хочет сохранить введенные в текстовое поле данные. Он нажимает на кнопку и тем самым отправляет команду во ViewModel. А ViewModel уже получает переданные данные и в соответствии с ними обновляет модель.

MVVM