UML-технологии для проектирования программного обеспечения

1. Унифицированный процесс разработки программного обеспечения

Владимир Юрьевич Романов,
Московский Государственный Университет им. М.В.Ломоносова
Факультет Вычислительной Математики и Кибернетики
vromanov@cs.msu.su,
vladimir.romanov@gmail.com

Предыстория унифицированного процесса

Rational Unified Process (**RUP**)

- **RUP** был создан фирмой **Rational** – разработчиком популярного CASE-инструмента **Rational Rose** и ряда других инструментов для разработки ПО

- В 2003 году фирма **Rational** вошла в фирму **IBM** как структурное подразделение

- Rational Unified Process был предложен для стандартизации в консорциум фирм **O**bject **M**anagement **G**roup (**OMG**).

  Цель стандартизации – возможность поддержки *унифицированного процесса разработки ПО* различными инструментами на основе унифицированной *метамодели процесса* разработанной консорциумом OMG.

Стандартизация унифицированного процесса

1. Каталог стандартов консорциума Object Management Group связанных со стандартом на язык UML:

   http://www.omg.org/technology/documents/modeling_spec_catalog.htm#UML

2. *Метамодель* процесса разработки программного обеспечения
   Software **P**rocess **E**ngineering **M**etamodel (**SPEM**)

   http://www.omg.org/spec/SPEM/2.0/

Стандарт SPEM 2.0 консорциума OMG

Software & Systems Process Engineering Metamodel Specification (SPEM) Version 2.0

Release Date: April 2008

Normative

OMG document number	Explanation	Format	URL
formal/2008-04-01	Formal specification	PDF	http://www.omg.org/spec/SPEM/2.0/PDF
formal/2008-04-01	Formal specification	PS	http://www.omg.org/spec/SPEM/2.0/PS
pto/07-08-08 pto/07-08-09	collection of schema files	CMOF XMI	
			http://www.omg.org/spec/SPEM/20070801/SPEM2-Process-Behavior-Content.merged.cmof
			http://www.omg.org/spec/SPEM/20070801/Infrastructure.cmof
			http://www.omg.org/spec/SPEM/20070801/LM.cmof
			http://www.omg.org/spec/SPEM/20070801/SoftwareProcessEngineeringMetamodel.cmof
			http://www.omg.org/spec/SPEM/20070801/SPEM2-Method-Content.cmof
			http://www.omg.org/spec/SPEM/20070801/SPEM2-Process-Behavior-Content.cmof
			http://www.omg.org/spec/SPEM/20070801/SPEM2.cmof
			http://www.omg.org/spec/SPEM/20070801/SPEM2.merged.cmof
			http://www.omg.org/spec/SPEM/20070801/SPEM2-Method-Content.merged.cmof
			http://www.omg.org/spec/SPEM/20070801/SPEM 2.0 Base Plugin Profile.xmi
			http://www.omg.org/spec/SPEM/20070801/SPEM 2.0 UML 2 Profile.xmi

Состав стандарта SPEM 2.0

OMG document number	Explanation	Format	URL
formal/2008-04-01	Formal specification	PDF	http://www.omg.org/spec/SPEM/2.0/PDF
formal/2008-04-01	Formal specification	PS	http://www.omg.org/spec/SPEM/2.0/PS
ptc/07-08-08 ptc/07-08-09	collection of schema files	CMOF XMI	
			http://www.omg.org/spec/SPEM/20070801/SPEM2-Process-Behavior-Content.merged.cmof
			http://www.omg.org/spec/SPEM/20070801/Infrastructure.cmof
			http://www.omg.org/spec/SPEM/20070801/LM.cmof
			http://www.omg.org/spec/SPEM/20070801/SoftwareProcessEngineeringMetamodel.cmof
			http://www.omg.org/spec/SPEM/20070801/SPEM2-Method-Content.cmof
			http://www.omg.org/spec/SPEM/20070801/SPEM2-Process-Behavior-Content.cmof
			http://www.omg.org/spec/SPEM/20070801/SPEM2.cmof
			http://www.omg.org/spec/SPEM/20070801/SPEM2.merged.cmof
			http://www.omg.org/spec/SPEM/20070801/SPEM2-Method-Content.merged.cmof
			http://www.omg.org/spec/SPEM/20070801/SPEM 2.0 Base Plugin Profile.xmi
			http://www.omg.org/spec/SPEM/20070801/SPEM 2.0 UML 2 Profile.xmi

Реализация унифицированного процесса как подпроекта Eclipse

Eclipse Process Framework Project (EPF)

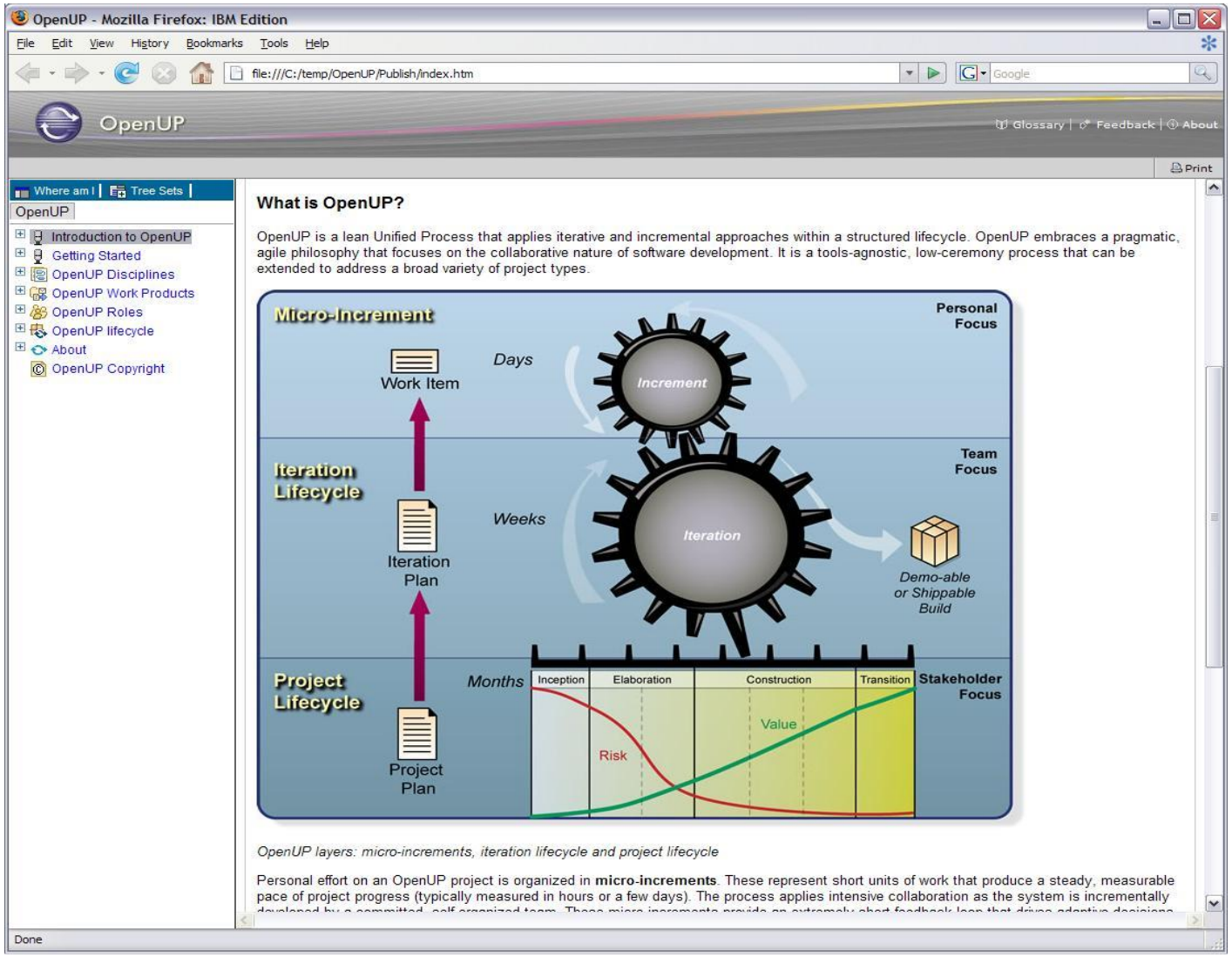
http://www.eclipse.org/epf/

Цель проекта:
способствовать созданию настраиваемого
фреймворка разработки программного обеспечения (SDE)

с примерным содержимым и инструментами процесса

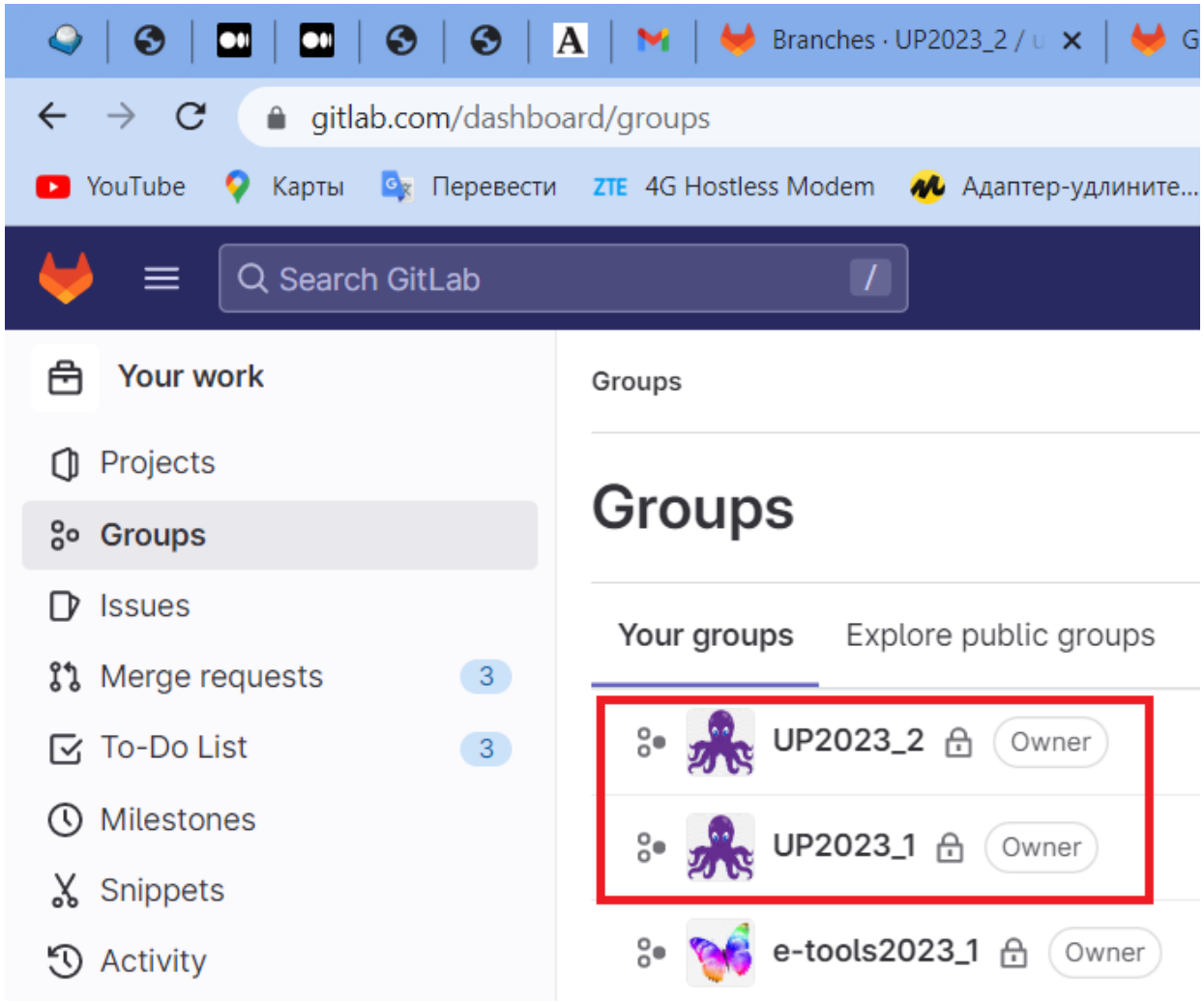
поддерживающим широкий спектр проектов и стилей разработки

Пример – Open UP



Группы практикума с использованием CASE-инструмента ***Papyrus***

МГУ им. М.В.Ломоносова. Факультет ВМК. Романов Владимир Юрьевич ©2024



Назначение унифицированного процесса

1. Предоставляет руководство по упорядочению деятельности команд разработчиков ПО
2. Управляет задачами решаемыми конкретными разработчиками ПО и командой в целом
3. Описывает какие артефакты (UML-модели, программы, документы) должны быть разработаны
4. Предоставляет критерии для оценки и измерения продуктов проекта и деятельностей проекта
5. Вся разработка ведется **на основе UML-моделей** разрабатываемого программного обеспечения

Назначение унифицированного процесса

1. Предоставляет руководство по упорядочению деятельности команд разработчиков ПО
2. Управляет задачами решаемыми конкретными разработчиками ПО и командой в целом
3. Описывает какие артефакты (UML-модели, программы, документы) должны быть разработаны
4. Предоставляет критерии для оценки и измерения продуктов проекта и деятельностей проекта
5. Вся разработка ведется **на основе UML-моделей** разрабатываемого программного обеспечения

Применимость Унифицированного процесса

Унифицированный процесс – это ***типовой процесс*** для:

- Для широкого класса проектируемых программных систем
- Для различных областей приложений
- Для различных типов организаций
- Для различных уровней компетентности
- Для различных размеров проектов

Отличительные особенности Унифицированного процесса

1. Управляется прецедентами (use case driven)
2. Ориентирован на раннюю разработку архитектуры
3. Итеративный и инкрементальный

Управление прецедентами в Унифицированном процессе (1)

- Для построения программной системы необходимо знание потребностей *пользователя*
- *Пользователь* (*user*) – человек или другая программная система расположенные вне разрабатываемой системы
- *Прецедент* (*use case*) – *взаимодействие* пользователя и системы
- Все прецеденты вместе – это *модель прецедентов* (***U**se **C**ase **M**odel*)
- Отличие UCM от функциональной спецификации – привязка функциональности к актерам

Управление прецедентами в Унифицированном процессе (2)

- Модель прецедентов (UCM) порождает серию других ***UML-моделей*** соответствующих UCM:
  - ***Модель анализа*** *(Analysis Model)*
  - ***Модель проектирования*** *(Design Model)*
  - ***Модель реализации*** *(Implementation Model)*
  - ***Модель тестирования*** *(Testing Model)*

Управление прецедентами в Унифицированном процессе (3)

- Процесс разработки следует *потоку* (*flow*)
- Процесс разработки проходит через *потоки работ* (*workflows*)
- Прецеденты ***влияют на создание*** архитектуры программной системы
- Архитектура программной системы затем ***влияет на выбор*** прецедентов использования при проектировании и реализации этой системы

Унифицированный процесс ориентирован на разработку архитектуры системы (1)

- Существует *аналогия архитектуры* в *строительстве* и *разработке программного обеспечения*

- Архитектура в строительстве *(точки зрения на здание*
  - *Структура*
  - *Электрификация*
  - *Водоснабжение*
  - *Обогрев*
  - *Сервис*

- *Архитектура в строительстве позволяет видеть <u>все</u> здание **до** начала строительства*

Унифицированный процесс ориентирован на разработку архитектуры системы (2)

Архитектура в разработке программного обеспечения

- *Порождается из нужд пользователя – модели прецедентов (Use Case Model)*
- *Зависит также от:*
  - Платформы:
    - архитектуры компьютера;
    - операционной системы;
    - СУБД;
    - связанности компьютеров в сети
  - Переиспользуемых компонент*:*
    - framework для построения интерфейса пользователя
    - соглашению по внедрению системы (deployment)
    - нефункциональным требованиям (производительность, память, …)

Унифицированный процесс ориентирован на разработку архитектуры системы (3)

Архитектура в разработке программного обеспечения

- *Это точка зрения (view) на программную систему в целом, в которой:*
  - Существенные *свойства системы сделаны видимыми*
  - *Несущественные свойства системы скрыты*
  - *Архитектор обладает достаточным опытом, что бы определить что существенно, а что не существенно*

Унифицированный процесс ориентирован на разработку архитектуры системы (5)

Архитектура в разработке программного обеспечения

- *Архитектор создает черновик (draft) архитектуры связанный не с прецедентами, а с платформой*
- *Архитектор далее работает с набором ключевых прецедентов.*
- *Каждый ключевой прецедент описывается и реализуется с использование понятий: подсистем, классов, компонент*
- *Разработанная архитектура затем используется для выбора и реализации оставшихся прецедентов*

Унифицированный процесс ориентирован на разработку архитектуры системы (4)

Архитектура в разработке программного обеспечения

- *Унифицированный процесс позволяет **архитектору** сконцентрироваться на правильных целях:*
  - *Понимаемости системы*
  - *Устойчивости системы к последующим изменениям*
  - *Переиспользуемости компонент системы*
- *Архитектор находит 5-10% ключевых прецедентов*

Унифицированный процесс итеративный и инкрементальный (1)

- *Разработка ПО может продолжаться от нескольких месяцев до нескольких лет*
- *По этой причине разработка ПО разделяется на несколько мини-проектов*
- *Каждый мини-проект называется итерацией*
- *Итерация выбирается и выполняется по плану*
- *Итерация проходит через потоки работ (workflows)*
- *Результат итерации - инкремент*

Унифицированный процесс итеративный и инкрементальный (2)

- *Необходимость итерации*
  - *Итерация имеет дело с группой прецедентов расширяющих используемость системы*
  - *Итерация имеет дело с наиболее важными рисками*

- *Итерация проходит через более мелкие шаги - потоки работ:*

  - *Поток работ анализа*
  - *Поток работ проектирования*
  - *Поток работ реализации*
  - *Поток работ тестирования*

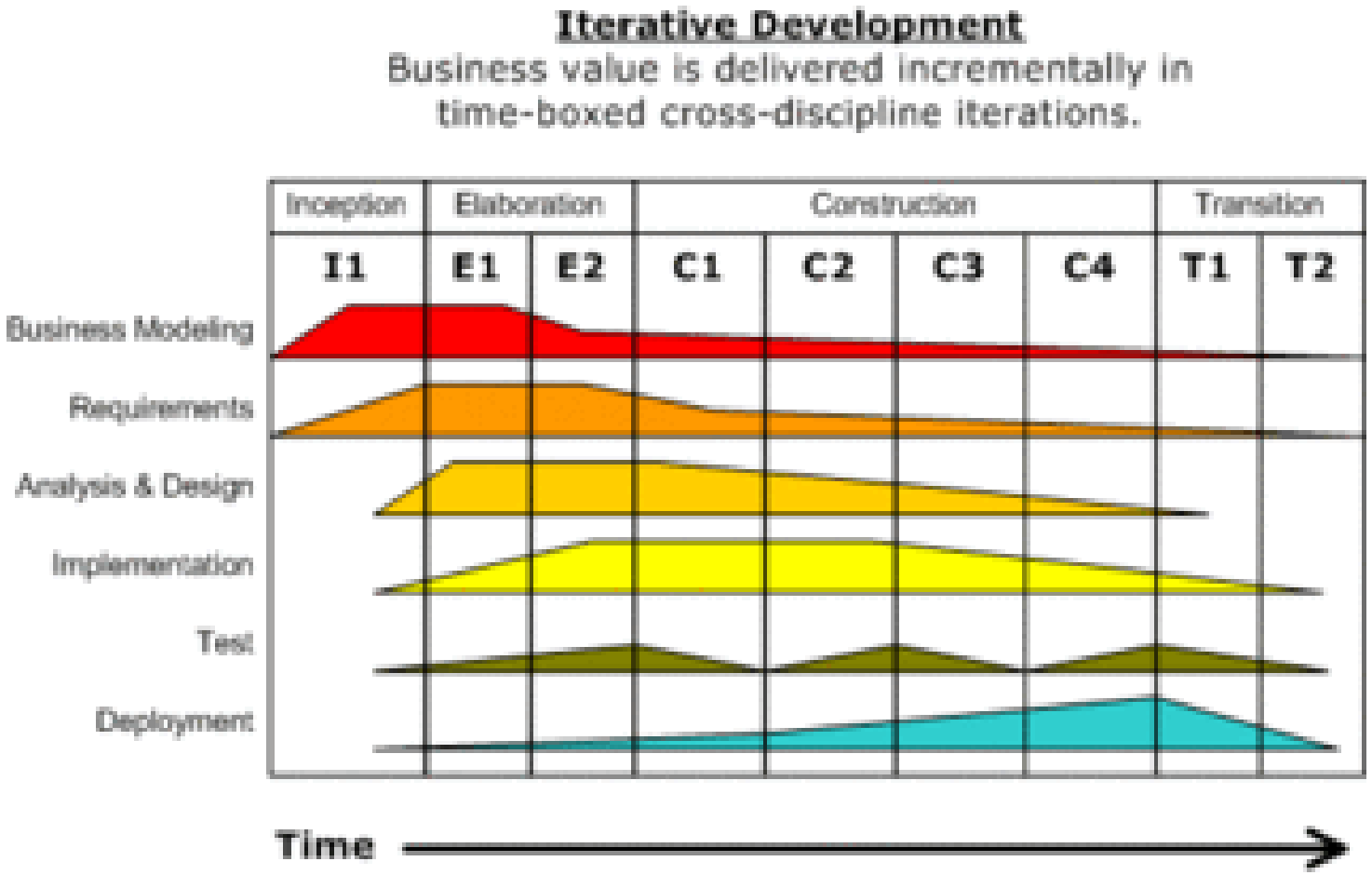
Унифицированный процесс итеративный и инкрементальный (3)

Достоинства итерактивного процесса

- *Уменьшается стоимость риска для одного инкремента. В случает неудачи теряется только одна итерация*
- *Уменьшается риск не поставки системы в заданные сроки. Риски определяются в начале проекта*
- *Ускоряется разработка ПО. Разработчик имеет ясную цель на короткий период времени*
- *По мере реализации итераций уточняются требования к системе*

Унифицированный процесс итеративный и инкрементальный (3)

Фазы итерактивного процесса



Процесс разработки, ориентированный на управление сценариями использования системы.

Процесс управляемый прецедентами
Что определяют прецеденты

- *Поиск и описание классов*
- *Поиск и описание подсистем и интерфейсов*
- *Поиск и описание прецедентов тестирования*
- *Планирование итераций разработки*
- *Системную интеграцию*

Процесс управляемый прецедентами. Цели сбора требований к системе

1. *Определение существенных требований.*

   Существенные те, которые принесут пользователю ощутимый и значимый результат.

2. *Представление в форме удобной для:*
   - *пользователей*
   - *заказчиков*
   - *разработчиков*

   с использование понятного для всех языка

Процесс управляемый прецедентами. Построение модели прецедентов

1. *Выявление категорий пользователей.*

2. *Представление категорий с помощью актеров языка UML*

3. *Для каждого актера определяются прецеденты – последовательности действий выполняемых системой для получения актером необходимого результата*

4. *Модель прецедентов – полный набор актеров и прецедентов*

Процесс управляемый прецедентами. Построение модели анализа (1)

1. *Модель анализа - описание структуры системы с использованием концептуальных классификаторов*

2. *Отдельная модель сопровождаемая для больших систем*

3. *Первая модель в понятиях привычных разработчику системы (классификаторы, пакеты, …)*

4. *Используется также и для сбора требований*

Процесс управляемый прецедентами. Построение модели анализа (2)

5. *Нет привязки к особенностям реализации*

   - *к языку программирования*

   - *к операционной системе*

   - *системе управления базами данных*

   - *библиотекам используемым для построения интерфейса пользователя*

6. *Составляет 20% от объема модели проектирования (Design Model)*

Процесс управляемый прецедентами. Построение модели проектирования (1)

1. *Для модели проектирования (design model) строится иерархия классов и интерфейсов*

2. *Описываются детально отношения ассоциации*

3. *Описываются отношения зависимости*

4. *Детально с использованием классификаторов и с помощью диаграмм поведения языка UML описываются реализации прецедентов*

5. *Затем модель проектирования отображается в модель реализации (Implementation Model)*

Процесс управляемый прецедентами. Построение модели проектирования (2)

4. *Выполняется привязка к конкретным инструментам разработки:*
   - языку программирования
   - операционной системе
   - системе управления базами данных
   - библиотекам для построения интерфейса пользователя

5. *Классы группируются в подсистемы, между которыми определяются интерфейсы*

6. *Строится модель внедрения (deployment model) описывающая распределение системы по вычислительным узлам*

Процесс управляемый прецедентами. Построение модели реализации

1. *Разбиение системы на компоненты*
   - файлы исходных текстов
   - jar-файлы
   - war-файлы
   - exe-файлы
   - DLL-файлы

2. *Распределение компонент по вычислительным узлам и директориям файловой системы*

3. *Определение отношений зависимости между компонентами*

Процесс управляемый прецедентами. Построение модели тестирования (1)

1. *Модель тестирования (Testing Model)*
   - для проверки реализации функциональности определенной в модели прецедентов
   - для проверки нефункциональных требований к системе
2. *Модель тестирования содержит прецеденты тестирования (Test Cases). Из прецедентов получаются:*

   - входные данные
   - условия выполнения прецедентов
   - результаты

Процесс управляемый прецедентами. Построение модели тестирования (2)

4. *Прецеденты тестирования (Test Cases) получаются сразу после получения прецедентов использования (Use Cases).*

   Следовательно прецеденты использования возможно планирование тестирования «черного ящика»

5. *После получения модели проектирования возможно тестирование «белого ящика».*

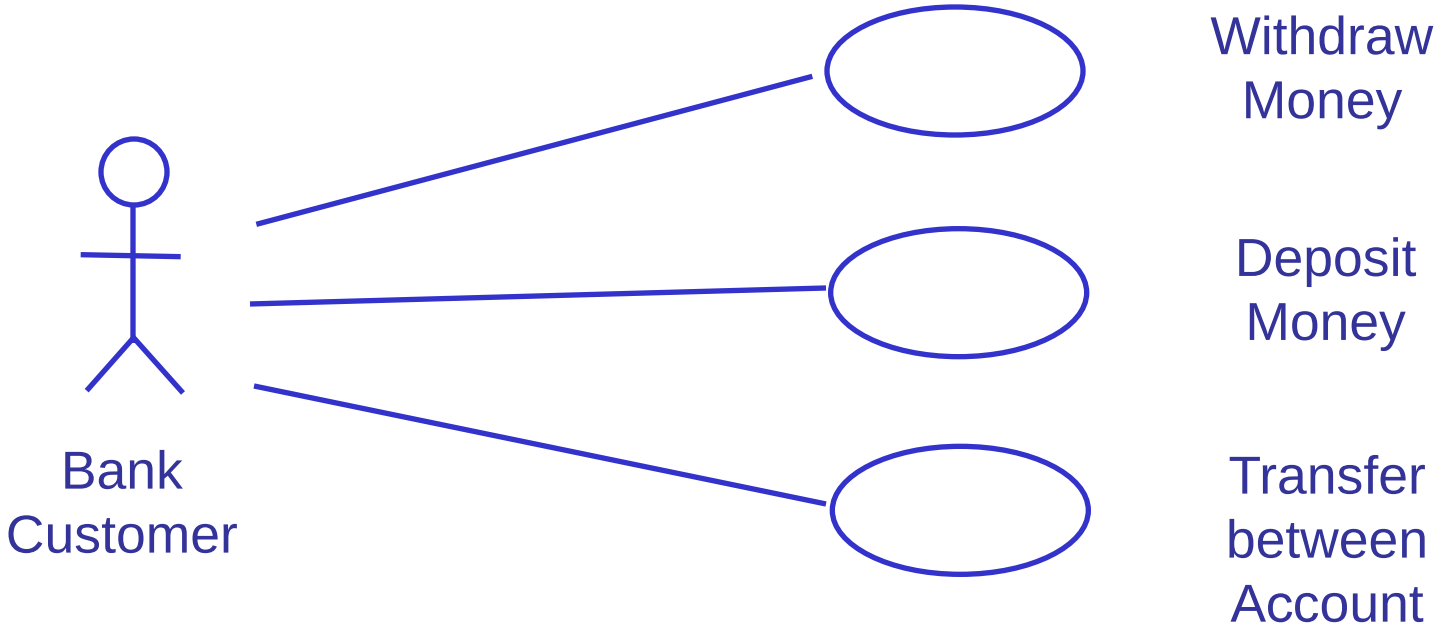
   Каждая ветвь в реализации прецедента использования определяет отдельный прецедент тестирования белого ящика.

Отличительные особенности Унифицированного процесса

1. Управляется прецедентами (use case driven)
2. Ориентирован на раннюю разработку архитектуры
3. Итеративный и инкрементальный

Построение модели прецедентов. Пример

Функциональные требования:



Нефункциональные требования к *WithdrawMoney*:

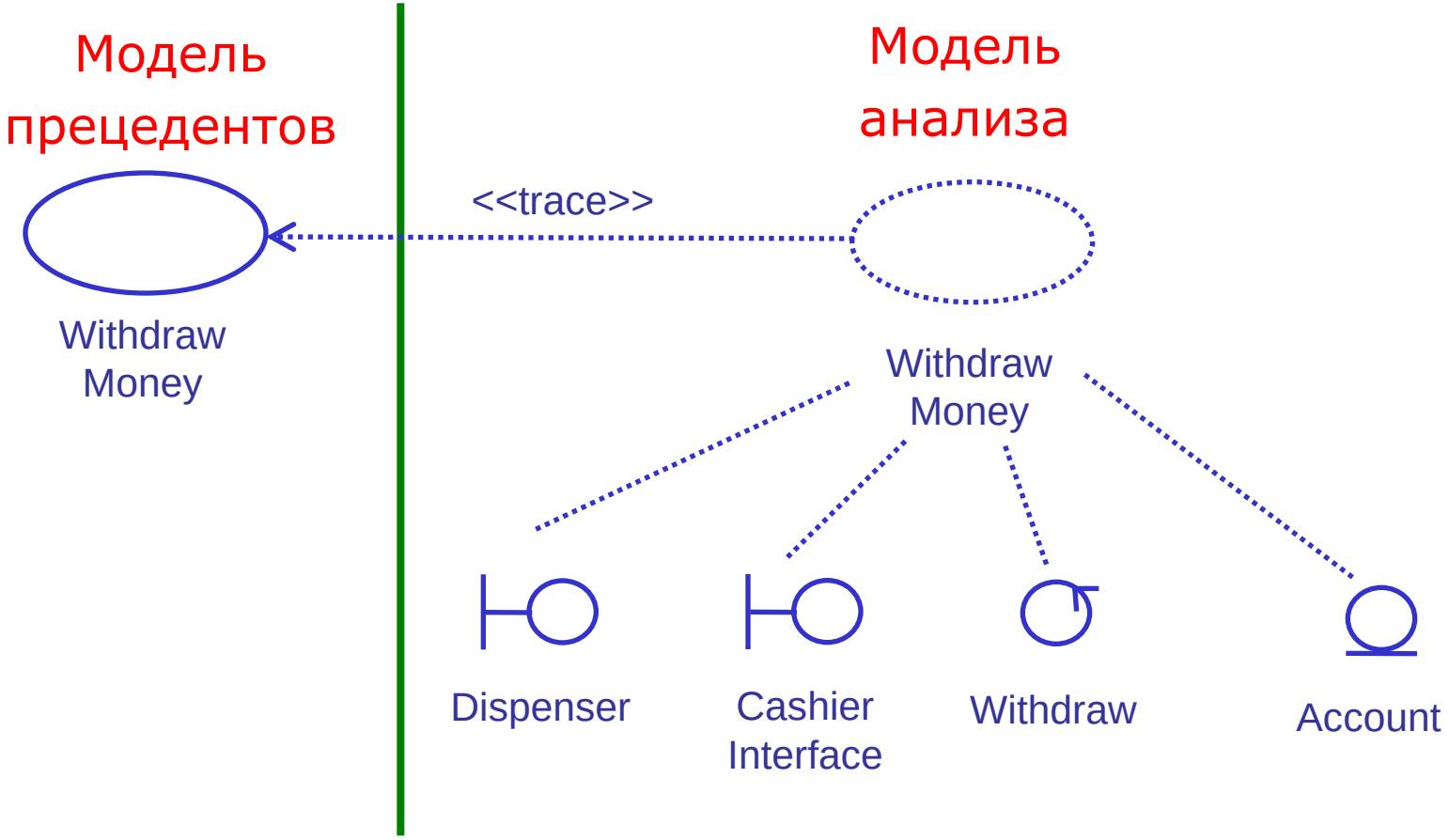
время отклика для *BankCustomer* не более 30 сек в 95% случаев

Построение модели анализа. Пример

Для всех прецедентов текущей итерации строится система классификаторов и отношений между ними

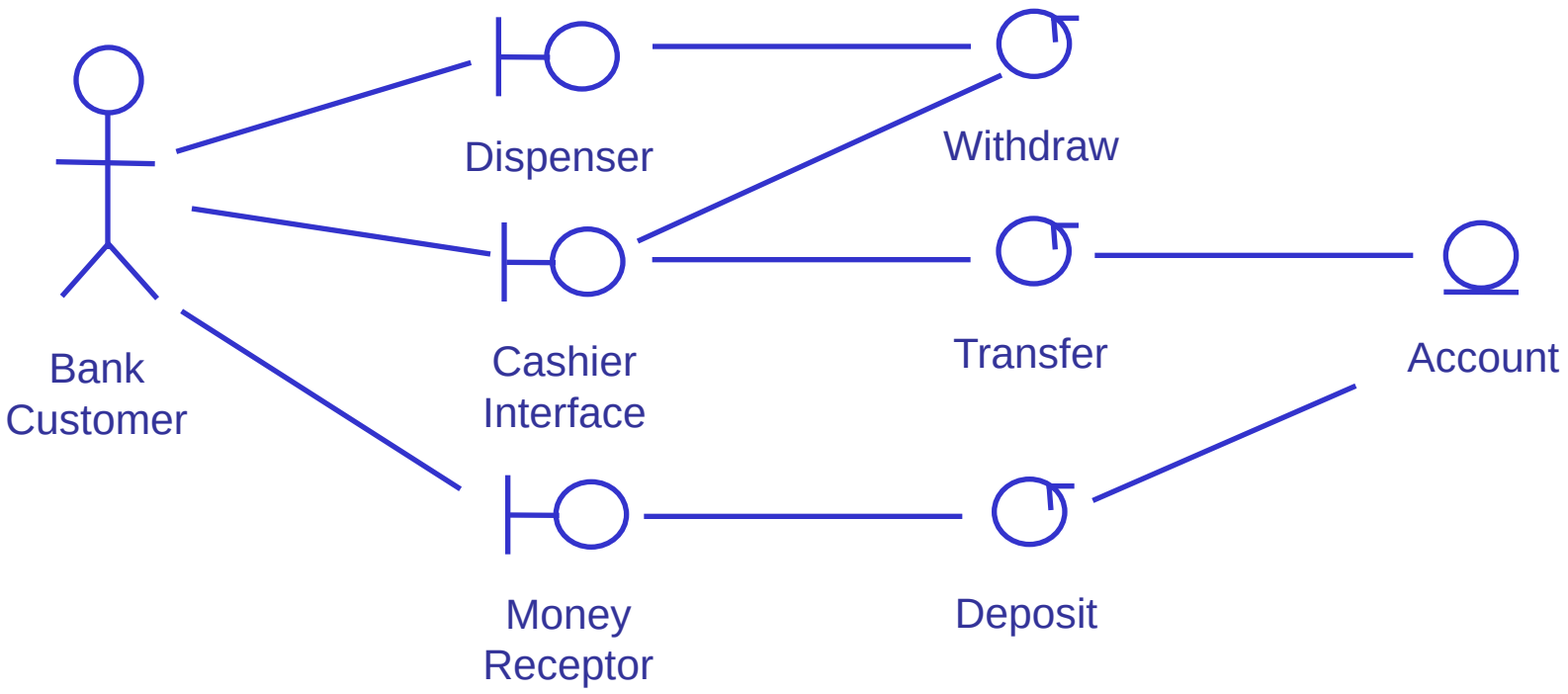
- *Какие классификаторы необходимы для реализации прецедентов*
- *Какие отношения ассоциации между классификаторами необходимы для реализации прецедента*

Построение модели анализа. Выбор классификаторов.



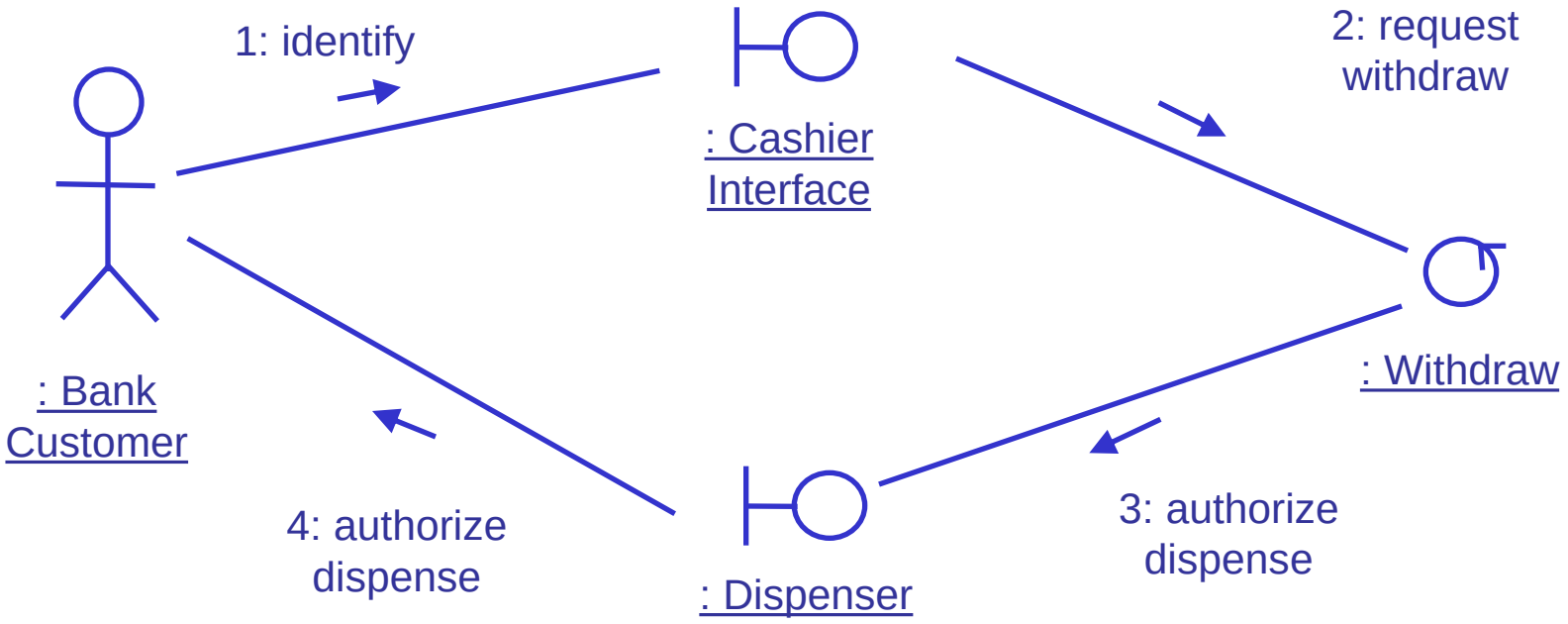
Построение модели анализа. Определение отношений ассоциации

Модель анализа

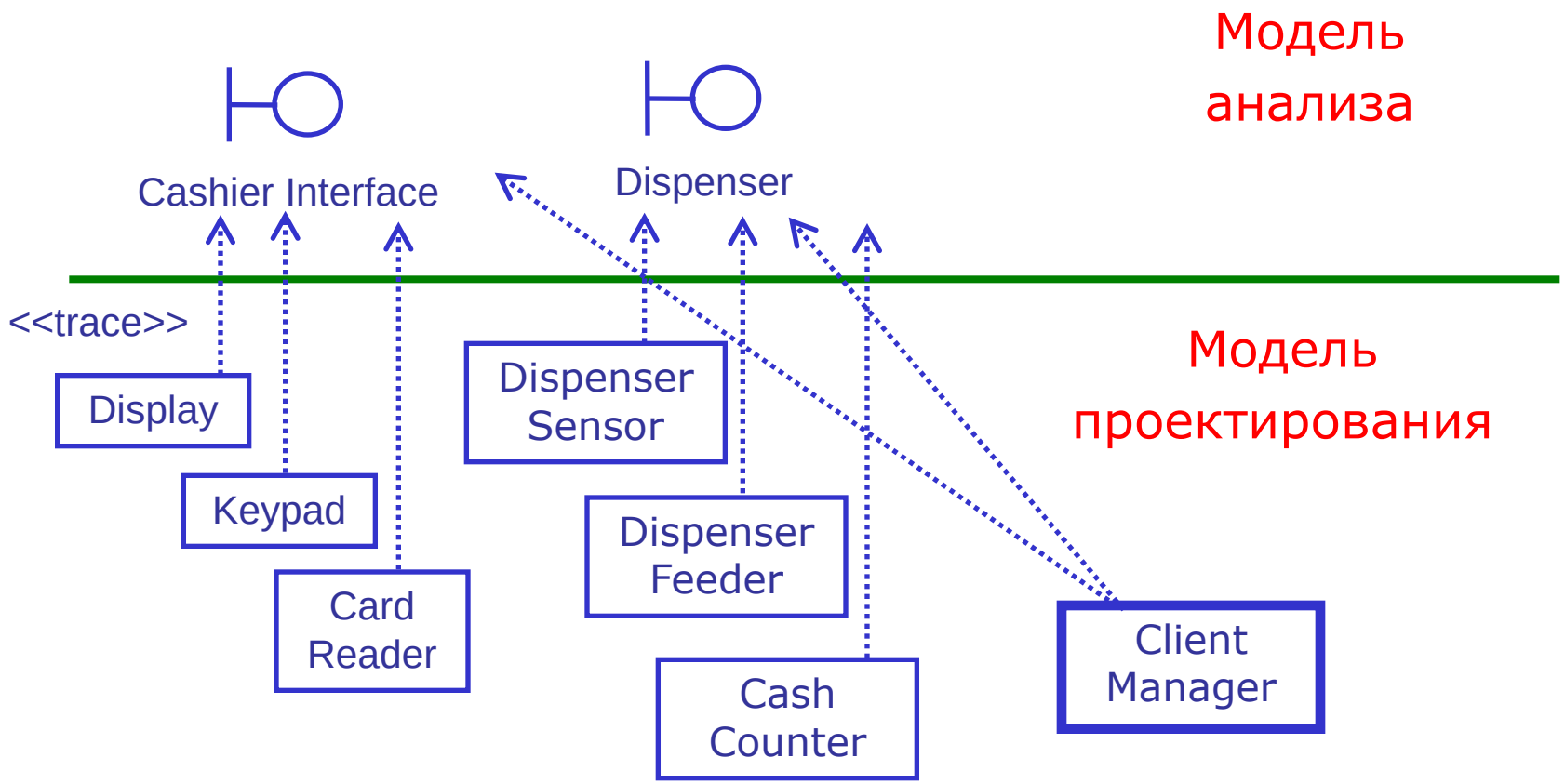


Построение модели анализа. Описание реализации прецедента

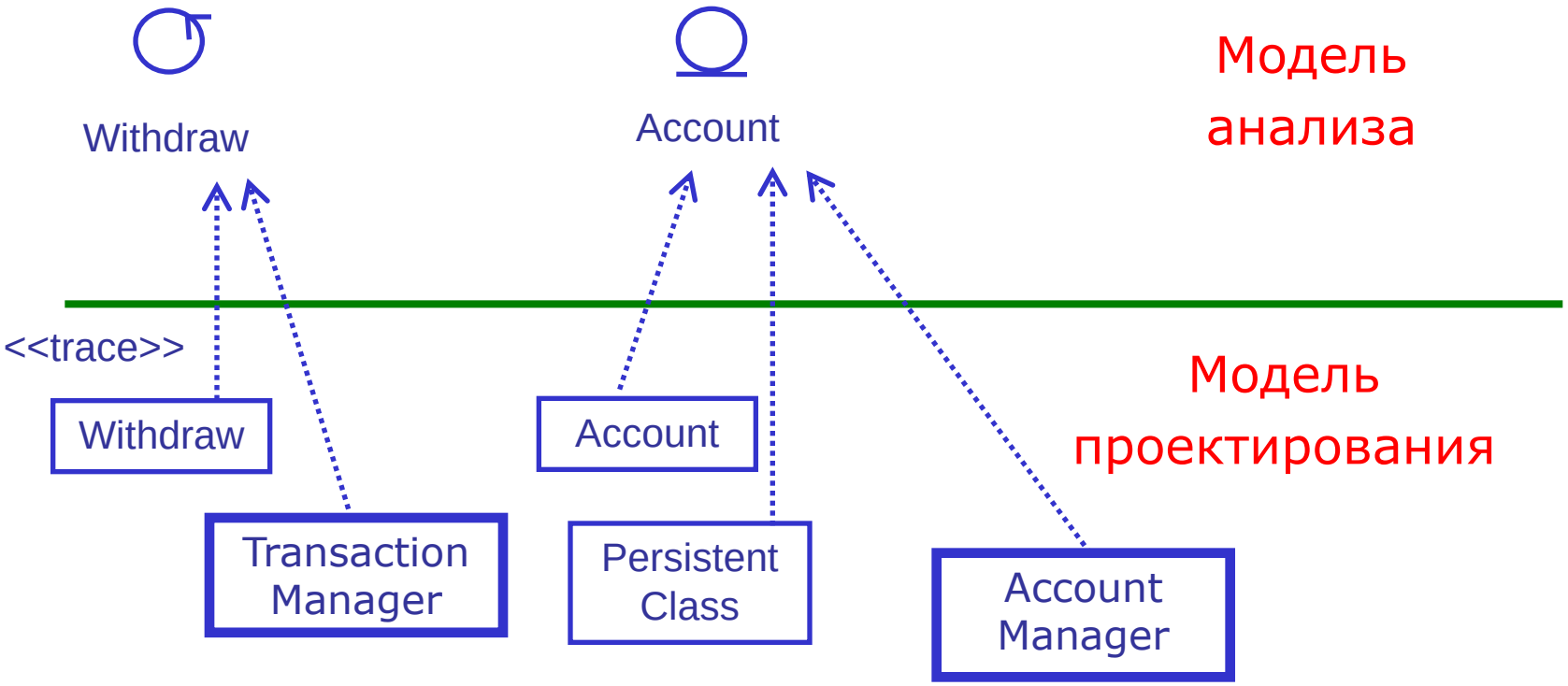
Модель анализа



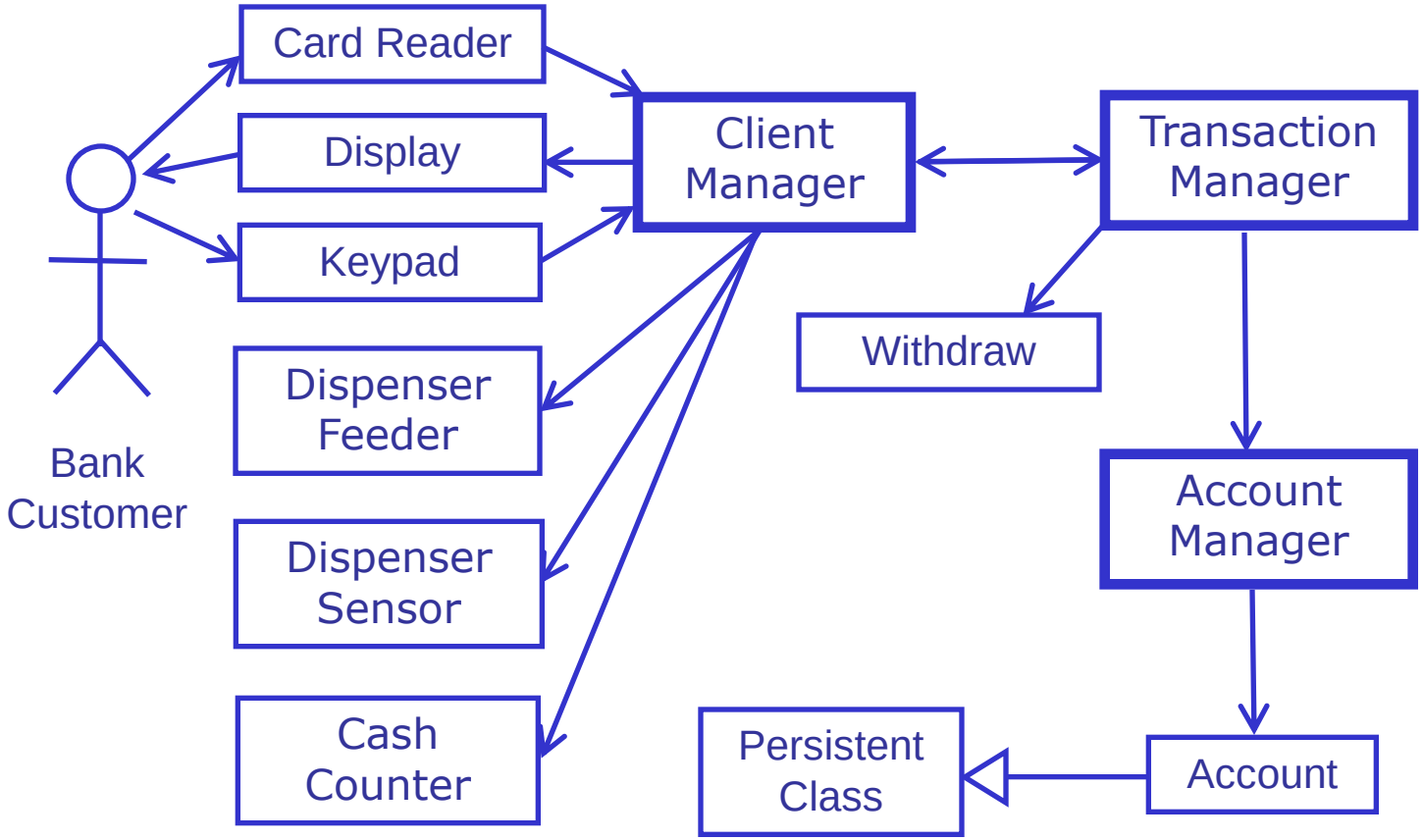
Построение модели проектирования.
Определение классов модели проектирования (1)



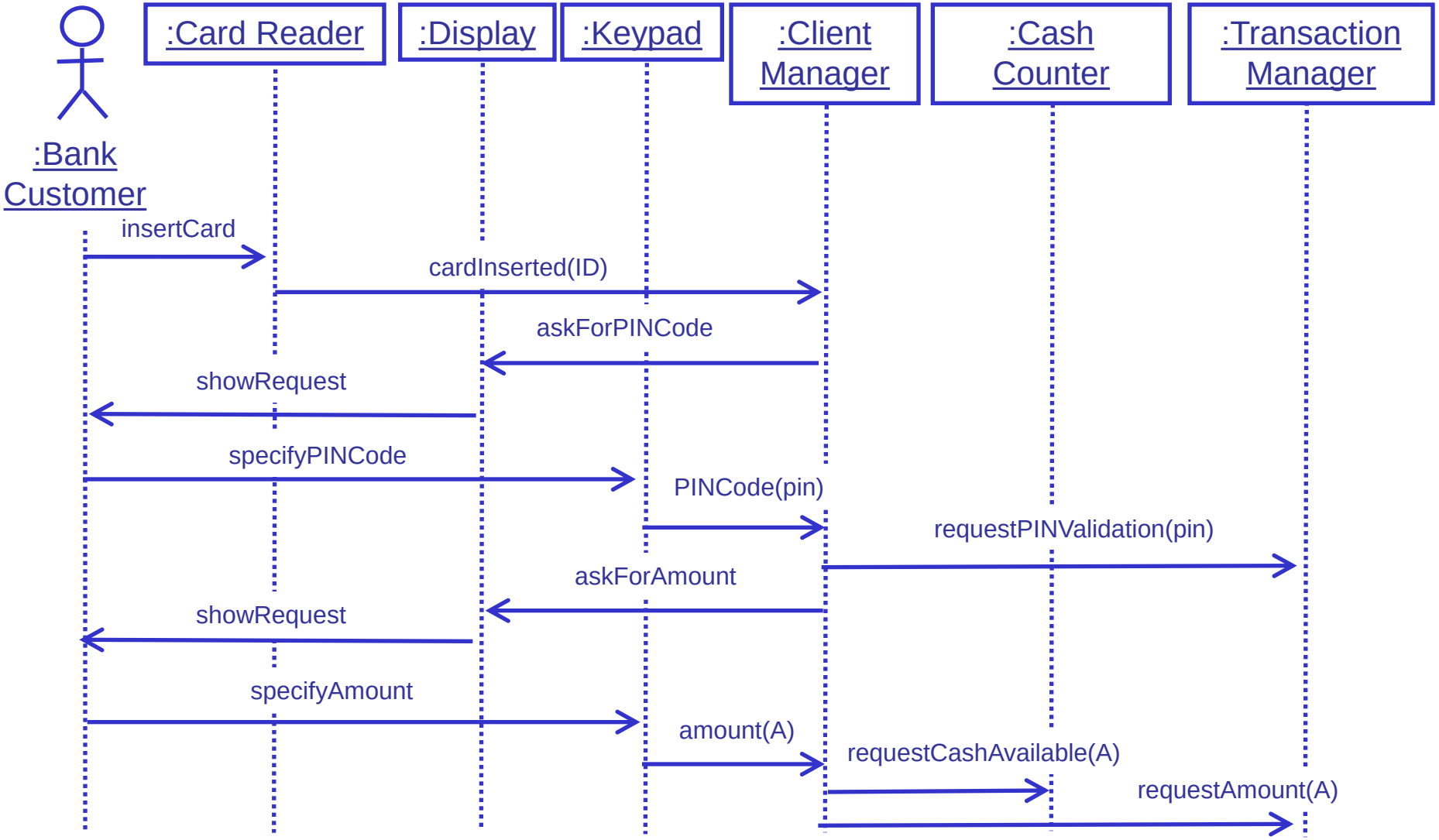
Построение модели проектирования.
Определение классов модели проектирования (2)



Построение модели проектирования. Определение отношений ассоциации



Построение модели проектирования. Описание реализации прецедента

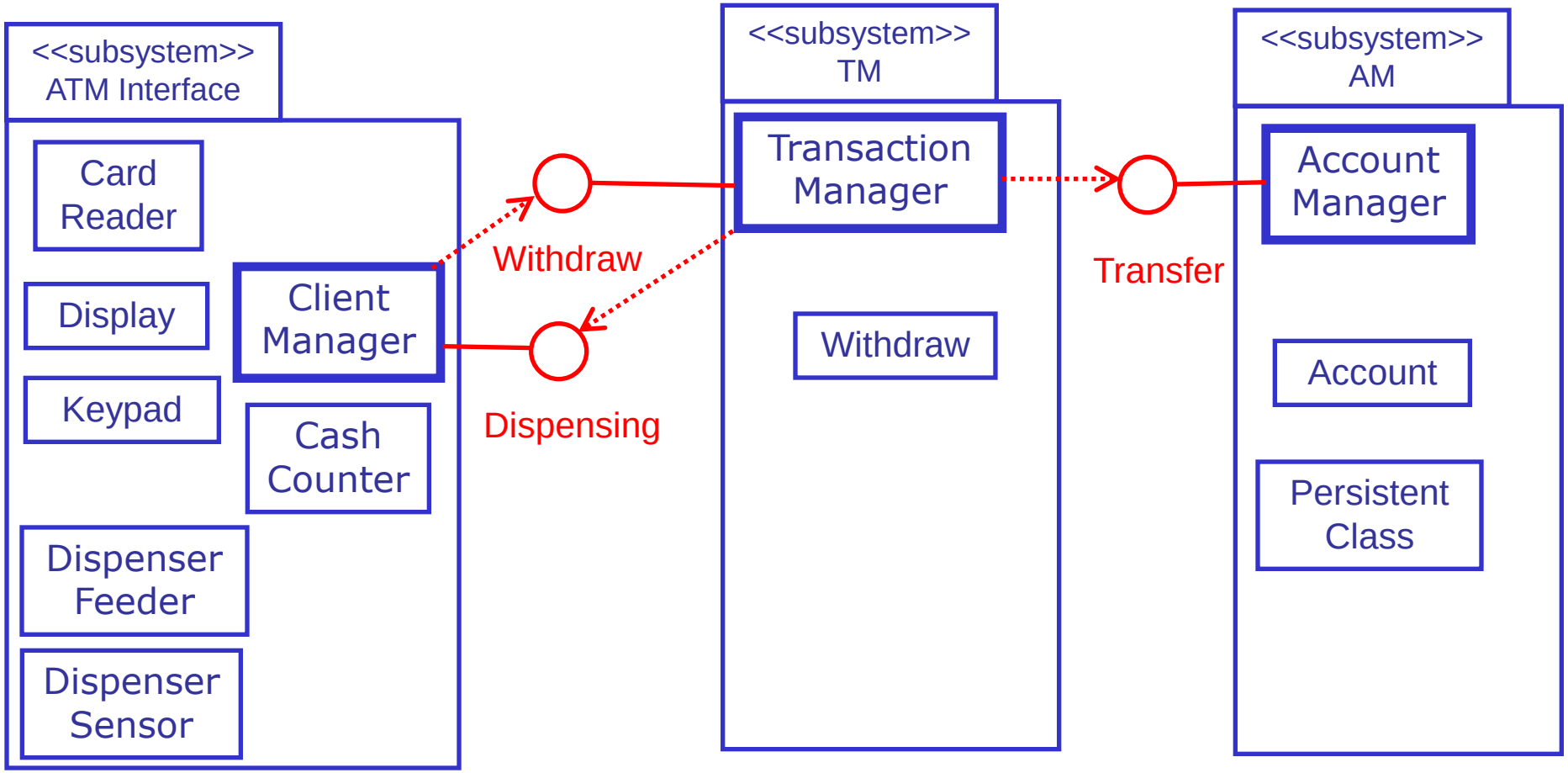


Построение модели проектирования. Пример

Системы с сотнями классов не управляемы =>

1. *Классы группируются в подсистемы*
2. *Подсистемы группируются в подсистемы*
3. *Подсистемы нижнего уровня – сервисные подсистемы*
4. *В подсистемы группируются классы реализующие необязательную (optional) функциональность*
5. *В подсистемы группируются взаимосвязанные классы (имеют тенденцию к общему изменению)*
6. *Группирование на основе уже найденных классов (снизу вверх)*
7. *Подсистемы и их интерфейсы определяет архитектор (сверху вниз)*

Построение модели проектирования. Определение отношений ассоциации



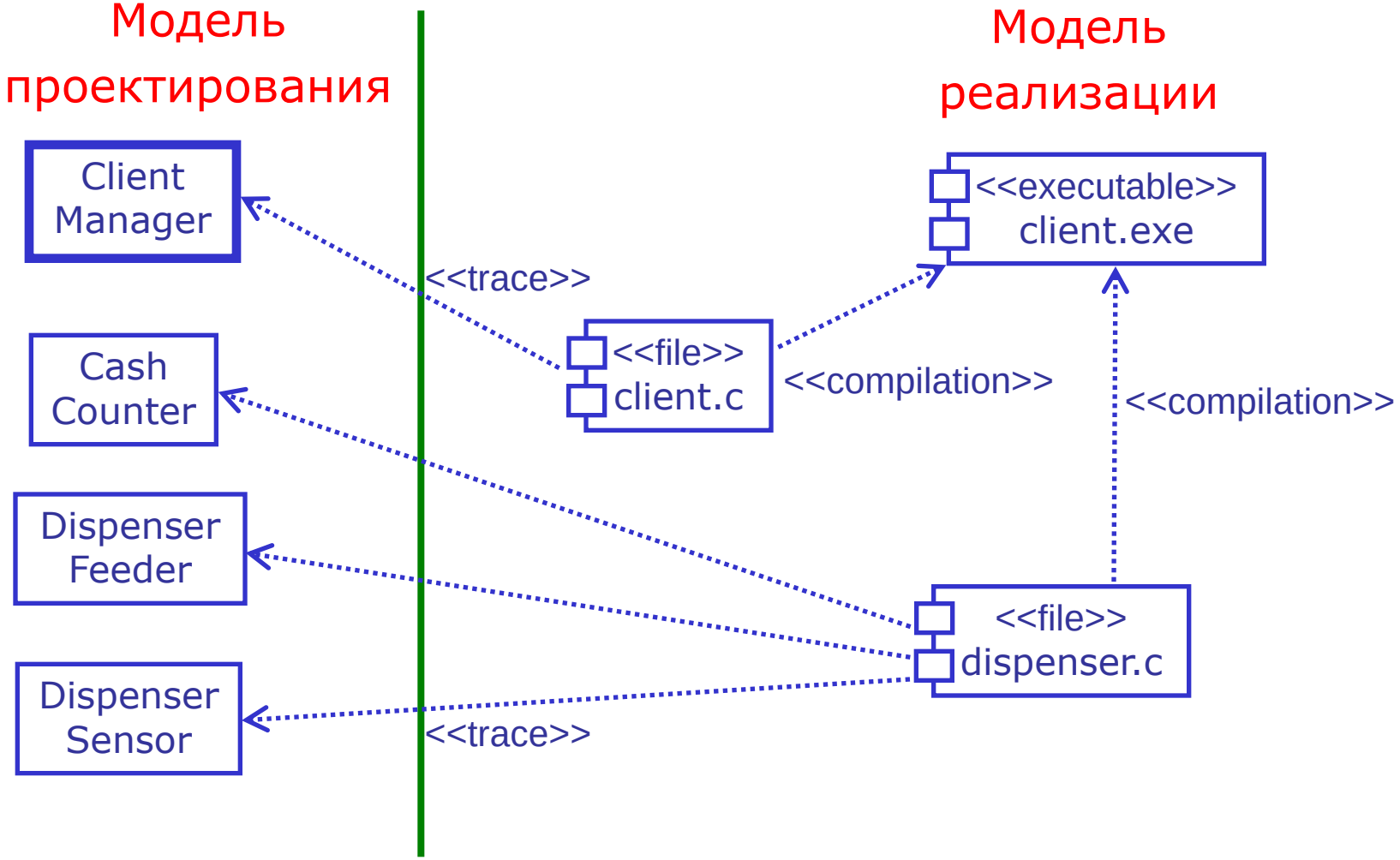
Построение модели реализации. Компоненты

1. *Компонента*
   – физическая и заменяемая часть системы
   - реализует интерфейсы

2. *Для разработки исполняемой компоненты*
   - исходные файлы
   - таблицы баз данных
   - другие исполняемые компоненты (DLL или JAR)

3. *Сервисные компоненты*
   - для размещения на отдельном вычислительном узле
   - может быть несколько реализаций для каждого типа вычислительного узла

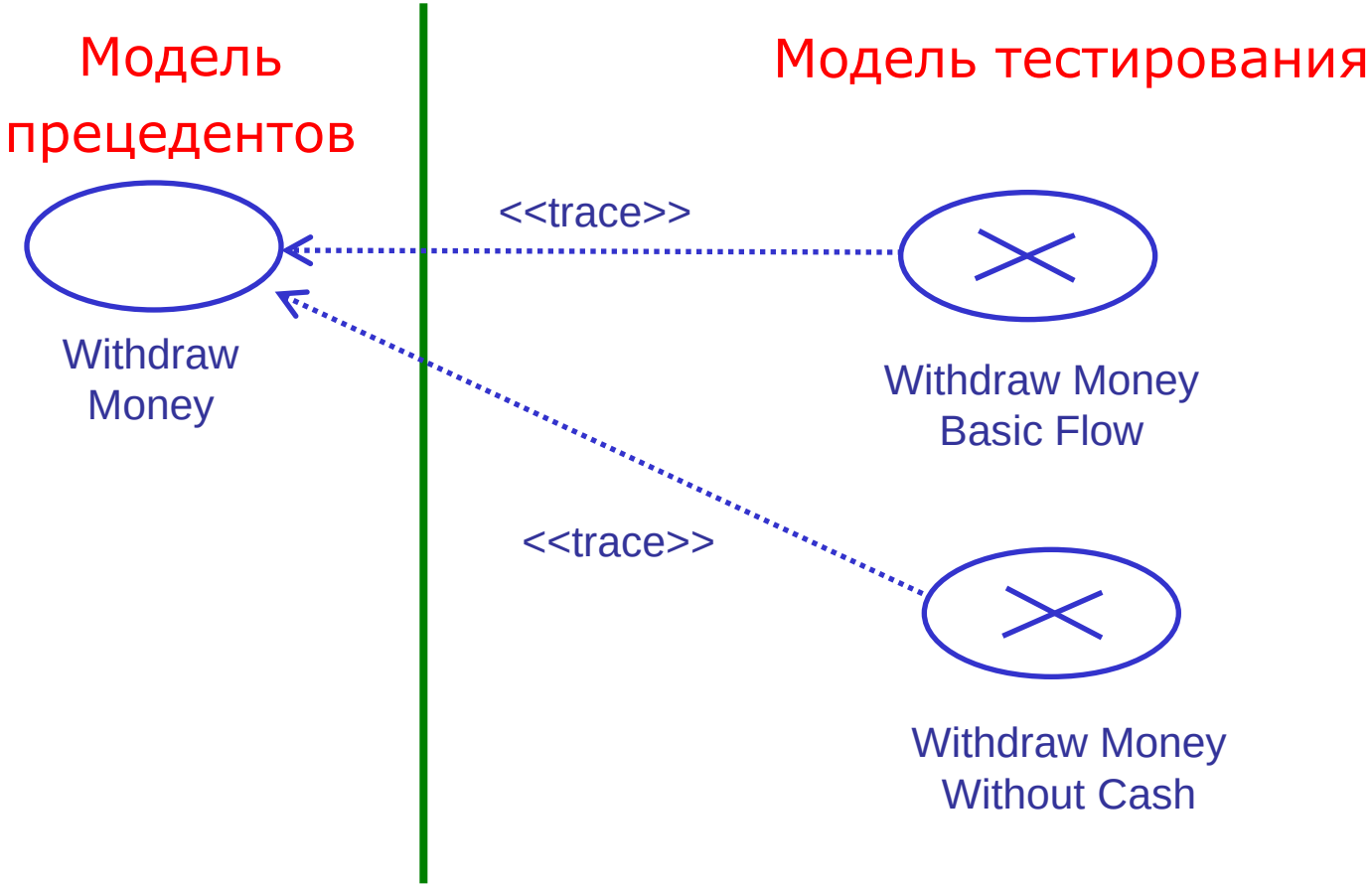
Построение модели реализации из модели проектирования



Построение модели тестирования. Прецеденты и процедуры тестирования

1. *Прецедент тестирования (Test Case)*
   - Входные данные
   - Условия тестирования
   - Предполагаемые результаты
   для
   - конкретной реализации прецедента (черный ящик)
   - конкретной ветви прохода через реализацию прецедента (белый ящик)
   - конкретному не функциональному требованию
2. *Процедура тестирования (Test Procedure)*
   описывает как:
   - выполнить инициализацию системы
   - выполнить тест
   - оценить результаты тестирования

Построение модели тестирования из модели прецедентов



Построение модели тестирования. Пример теста

1. *Ввод*
   - Клиент на счету 11-11-1111 имеет 350$
   - Клиент правильно идентифицировал
   - Клиент запросил снять 200$ со счета 11-11-1111
   - денег в банкомате достаточно

2. *Результат*
   - Баланс клиента уменьшился на 200$
   - Клиент получил 200$

Отличительные особенности Унифицированного процесса

1. Управляется прецедентами (use case driven)
2. Ориентирован на раннюю разработку архитектуры
3. Итеративный и инкрементальный

Процесс ориентированный на разработку архитектуры системы.

Процесс ориентированный на разработку архитектуры системы (1)

Архитектура – общий взгляд на систему, с которым согласны все разработчики системы

Архитектура – ясный взгляд на систему в целом, дающий возможность управлять ее разработкой

Архитектура – описывает наиболее важные элементы модели, которые

- *управляют работой в каждом конкретной итерации*
- *управляют работой через весь жизненный цикл*

Процесс ориентированный на разработку архитектуры системы (2)

*Архитектура – содержит **некоторые** из:*

- *Подсистем*
- *Интерфейсов*
- *Вычислительных узлов*
- *Зависимостей*
- *Взаимодействий*
- *Активных классов*

которые позволяют:

- *Понять систему*
- *Разработать систему*
- *Оценить стоимость разработки системы*

Процесс ориентированный на разработку архитектуры системы (3)

Архитектор проходит несколько итераций в фазах:

- *Inception (начальная, анализа и планирования)*
- *Elaboration (исследование) цель фазы исследования – разработать архитектуры в виде выполняемого фундамента (architecture baseline)*
- *Construction (конструирование) – на фазе конструирования есть прочный фундамент для построения полной системы*
- *Deployment (внедрение)*

Процесс ориентированный на разработку архитектуры системы (4)

Архитектор – специалист по интеграции частей проекта, но не эксперт по каждой части проекта.

Описание архитектуры используется разработчиками системы, но и них есть и более детальное описание

Архитектор дает взаимосвязанные виды, проекции (view) системы с помощью UML-диаграмм

Архитектор принимает решения о

- *организации программной системы*
- *Структурных элементах и их интерфейсах, о взаимодействии элементов (collaborations)*

Процесс ориентированный на разработку архитектуры системы (5)

- *Композиции элементов структуры и поведения в пакеты*
- *Структурных элементах и их интерфейсах, о взаимодействии элементов (collaborations)*
- *Описание архитектуры:*
  для каждой модели 4+1 вида

Модель представления архитектуры **4+1**

Аналитики
Поведение системы

Конечный пользователь
Функциональные возможности

Программисты
Управление прогр.обеспечением

Логическое представление

Реализационное представление

Прецеденты

Процедурное представление

Представление внедрения

Системный интегратор
Функциональные возможности
Модульное наращивание
Производительность

Системотехника
Топология системы
Внедрение
Установка
Связи

Модель представления архитектуры **4+1**

- *Прецеденты*
  Ключевые сценарии и прецеденты. Используются для задания направления поиска других представлений

- *Логическое представление*
  - *Функциональные требования системы (что нужно пользователю)*
  - *Абстракция модели проектирования*
    - пакеты
    - подсистемы
    - классы

Модель представления архитектуры **4+1**

- *Реализационное представление*
  Организация статических программных кодов
  - тексты
  - данные
  - компоненты
  - исполняемые коды

  в терминах

  - пакетов
  - уровней
  - управления конфигурациями

Модель представления архитектуры **4+1**

- *Процедурное представление*
  - *Параллельность*
    - задач
    - потоков
    - процессов
    - их взаимодействие
  - *Распределенность объектов*
  - *Отказоустойчивость*
- *Представление внедрения*
  Распределение компонент на платформы и вычислительные узлы

Описание архитектуры.

- *<u>Сложность</u> создания описания архитектуры – <u>извлечение только самого важного</u> без добавления в описание нового*
- *Первая версия описания архитектуры появляется в конце фазы Elaboration*
- *Сложно сохранить описание системы небольшим*

Описание архитектуры. Зачем нужна архитектура

1. *Для понимания системы*
2. *Для организации разработки системы*
3. *Для стимулирования повторного использования уже написанного кода и компонент системы*
4. *Для развития системы в дальнейшем*

Описание архитектуры. Понимание системы

1. *Сложное поведение системы*
2. *Работа в сложном окружении – сложно интегрировать во множество компонент разработанных сторонними разработчиками*
3. *Используются сложные технологии (EJB или COM)*
4. *Используют и индивидуальные пользователи и группы пользователей*
5. *Разрабатываются распределенно (географически) и поэтому необходимо координировать работу*

Описание архитектуры. Организация разработки

1. *Организация разработки существенно упрощается, если может выполняться распределенно и параллельно*

2. *Назначение групп ответственных за подсистемы дает возможность распределенности и независимости*

3. *Стабильные интерфейсы – позволяют независимо развивать программу по обе стороны интерфейса*

4. *Правильная архитектура и шаблоны проектирования позволяют найти правильные интерфейсы между подсистемами*

Описание архитектуры. Шаблоны архитектуры (1)

Facade – предназначен для создания упрощенного интерфейса для группы подсистем или сложной подсистемы.
Область применения:

- *Необходимо предоставить более простой вариант стандартного интерфейса*
- *Уменьшить зависимость клиента от подсистемы*
- *Нужно создать слои подсистемы*

Описание архитектуры. Шаблоны архитектуры (2)

Decorator – предоставление механизма для добавления или удаления функциональности компонентов без изменения их внешнего представления или функции.
Назначение:

- *Необходимо динамически менять свойства классов*
  - незаметно для пользователя класса
  - не испытывая ограничения механизма наследования классов

- *Свойства компонента динамически включаются и отключаются*

- *Есть несколько независимых функций, применяемых:*
  - динамически
  - в разной комбинации

Описание архитектуры. Шаблоны архитектуры (2)

Layers – Способ организации модели проектирования на уровни.

Компоненты одного уровня могут ссылаться только на компоненты нижнего уровня

=> Упрощение:

- *Компоненты нижнего уровня не зависят от:*

  - интерфейсов верхнего уровня

  - реализации верхнего уровня

Описание архитектуры. Поощрение повторного использования

1. *Архитектор может определить повторно используемые подсистемы тщательно проектируя их*
2. *Архитектор может определить повторно используемые компоненты*
3. *UML ускоряет создание повторно используемых подсистем и компонент*
4. *Должно предполагаться использование подсистем вне текущего контекста*
5. *Архитектор должен решать более общую задачу*

Описание архитектуры. Развитие системы

1. *Развитие стимулируется изменением окружения системы*

2. *Может потребоваться новая функциональность системы*

3. *Необходима устойчивость системы при ее изменении, а не ее <u>деградация</u>*

Описание архитектуры. Развитие системы. Принципы разработки системы (1)

1. *Функциональная модульность. Классы группируются в optional сервисные подсистемы SSS.*
   SSS имеют только внутреннее сцепление (cohesion)
   => SSS независимы

2. *Отделение проектирования интерфейсов от проектирования SSS*

   1. *=> несколько SSS могут поддерживать тот же интерфейс*

   2. *=> возможна замена подсистем, поскольку зависимость клиента только от интерфейса*

Описание архитектуры. Развитие системы. Принципы разработки системы (2)

3. *SSS – на этапе проектирования помещается в отдельную компоненту*

   => компоненты могут быть размещены на разные вычислительные узлы

4. *Необходимо уменьшение сцепления между SSS.*

   => единственный способ общения между подсистемами через асинхронные сигналы

   => поощряется не только инкапсуляция, но и распределенность

Взаимосвязь прецедентов и архитектуры.

Прецеденты

- Системное ПО
- Middleware
- Frameworks
- Стандарты
- Нефункциональные требования
- Особенности внедрения

Архитектура

Опыт

- шаблоны архитектуры
- предшествующие архитектуры

Шаги к архитектуре. (1)

- *Результат фазы Elaboration – фундамент архитектуры (скелет системы)*
- *Выбор прецедентов для построения архитектуры*
  - *те, которые позволяют оценить наиболее серьезные риски*
  - *Наиболее важная функциональность для пользователя*
- *Реализация, интеграция и тестирование фундамента архитектуры приводит к демонстрационной версии системы от которой идет обратная связь*

Шаги к архитектуре. (2)

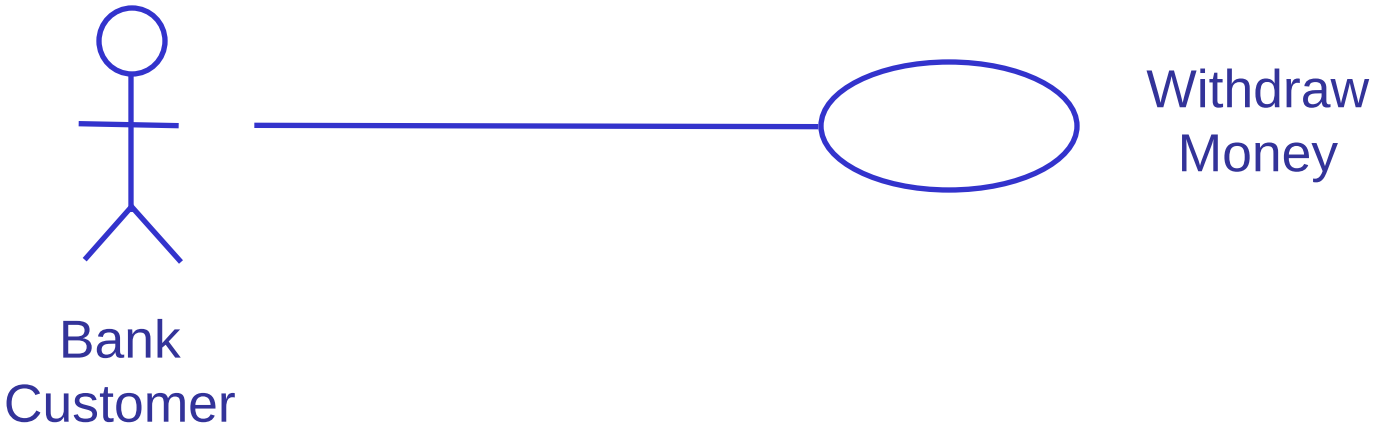
- *Фундамент архитектуры – небольшая тощая система которая имеет все модели которые имеет полная система в конце фазы конструирования*

- *Фундамент имеет скелет для:*
  - *Подсистем*
  - *Компонент*
  - *Вычислительных узлов*
    Но и имеет также
  - *Поведение*
  - *Исполняемый код*

- *После стабилизации фундамента архитектуры заканчивается фаза уточнения*

Описание архитектуры. Модель прецедентов

- *В модель прецедентов включаются наиболее важные актеры и прецеденты*
- *В нашей модели (условно) наиболее важным будем считать прецедент WithdrawMoney*



Описание архитектуры. Модель проектирования

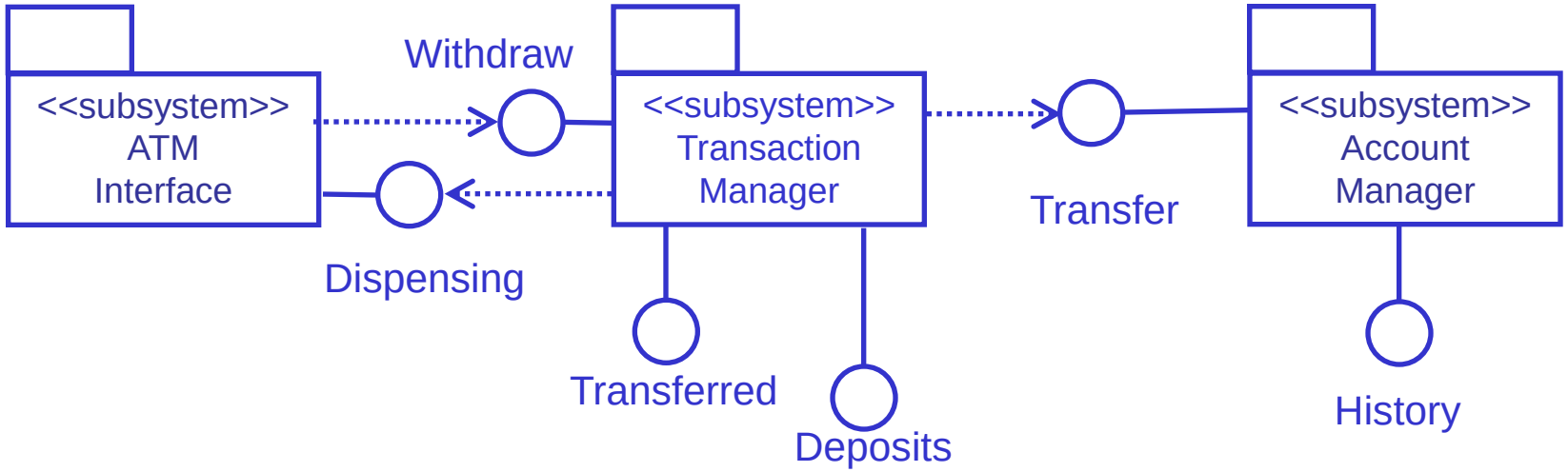
- *Наиболее важные подсистемы*
- *Интерфейсы*
- *Несколько особенно важных классов*
- *Активные классы*
- *Реализация наиболее важных прецедентов с помощью описанных в архитектуре классификаторов*
- *Распределение активных классов по вычислительным узлам*

Модель проектирования (архитектура). Активные классы

- *Все активные классы включаются в описание модели проектирования архитектуры*
- *Классы необходимые для понимания реализации прецедента WithdrawMoney*

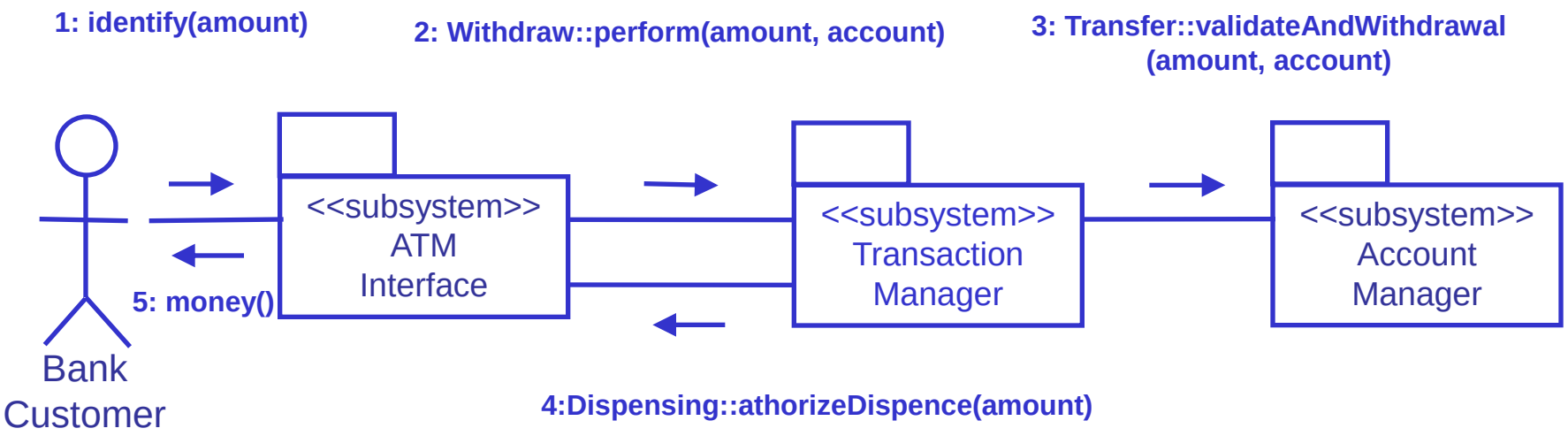
Модель проектирования (архитектура). Подсистемы и их интерфейсы

- *Подсистемы и интерфейсы необходимые для понимания реализации прецедента WithdrawMoney*



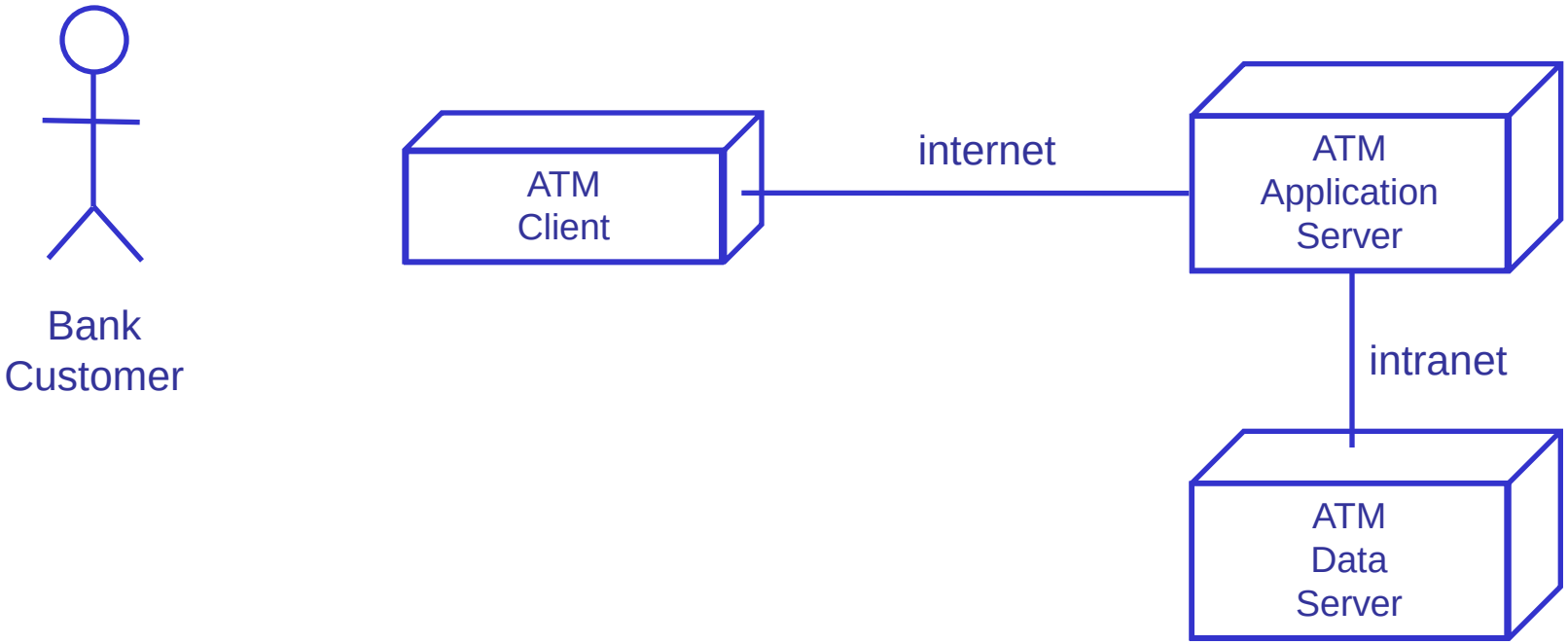
Модель проектирования архитектуры. Реализация прецедента WithdrawMoney

- *Взаимодействие подсистем для реализации прецедента WithdrawMoney*



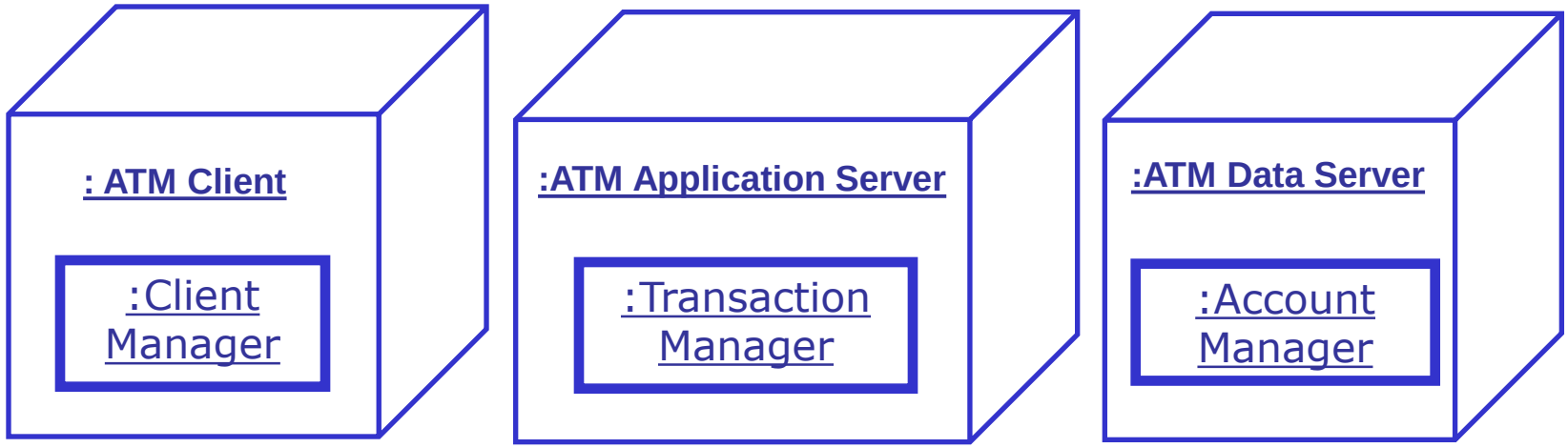
Модель внедрения (архитектура). (Deployment Model)

- *Классификаторы – вычислительные узлы*



Модель внедрения (архитектура). Активные объекты

- *Активные объекты распределенные на экземпляры вычислительных узлов*



Модель внедрения (архитектура). (Deployment Model)

- *Определяется архитектура физической системы в терминах вычислительных узлов*
- *Модель внедрения может быть смоделирована на этапе сбора требований*
- *Для активных классов определяется:*
  - *Что делают активные объекты (экземпляры классов)*
  - *Жизненный цикл активных объектов: кто, когда и как создает и уничтожает активные объекты*
  - *Как активные объекты: общаются, синхронизируются, передают информацию*
- *Оцениваются возможности выч. узлов и их память*
- *Оценивается пропускная способность и доступность линий связи*

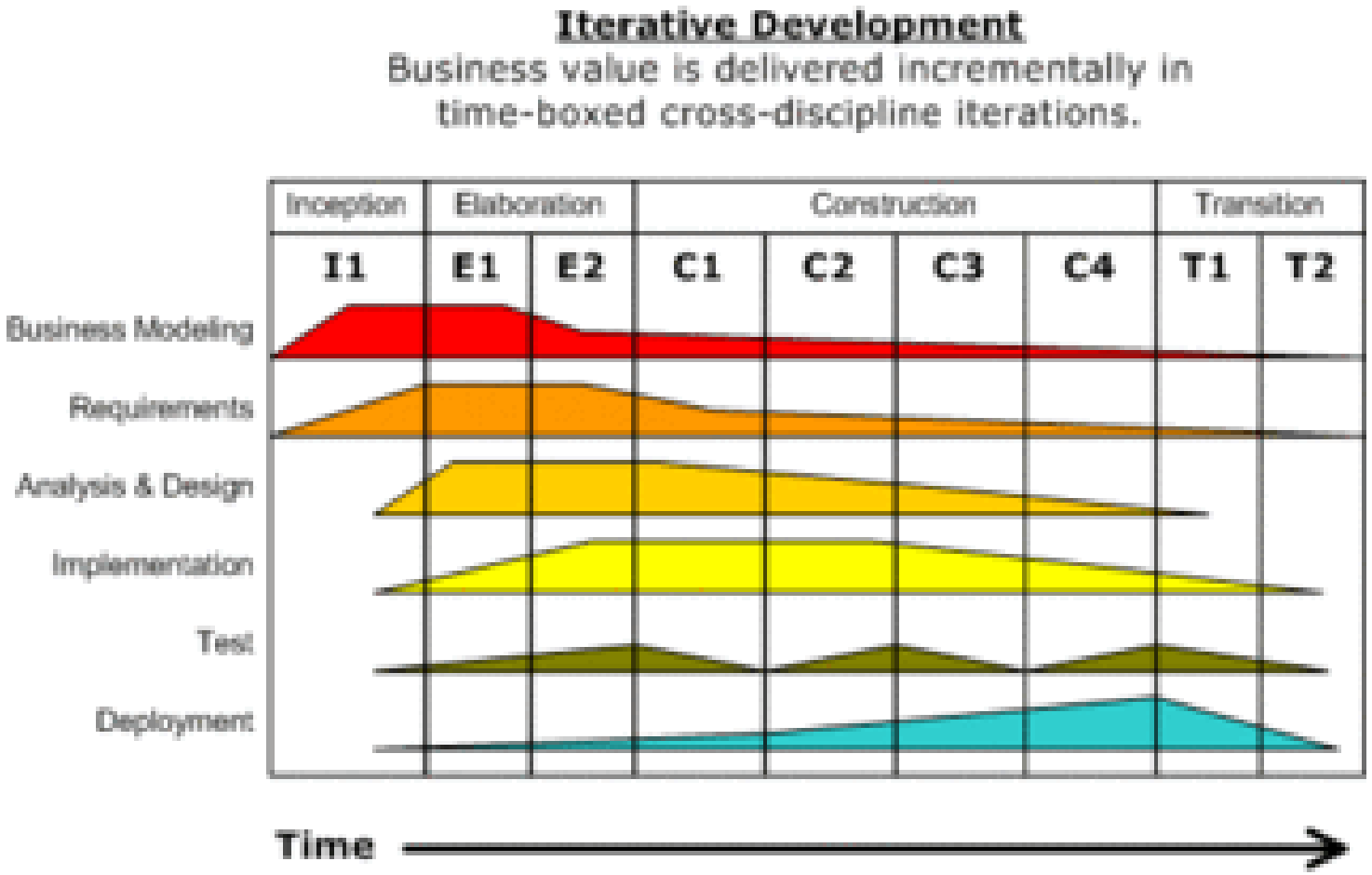
Отличительные особенности Унифицированного процесса

1. Управляется прецедентами (use case driven)
2. Ориентирован на раннюю разработку архитектуры
3. Итеративный и инкрементальный

Итеративность и инкрементальность процесса.

Унифицированный процесс итеративный и инкрементальный (3)

Фазы итерактивного процесса



Итерактивность и инкрементальность унифицированного процесса

- *Для эффективности унифицированного процесса необходимо иметь последовательность ясно сформулированных контрольных точек (milestones)*

  которые дают команде критерии окончания фазы для перехода от одной фазы к другой фазе

- *Внутри каждой фазы процесс проходит через итерации и приращения (increment)*

Фаза анализа и планирования требований

Фаза анализа и планирования требований (inception)

На этой фазе критерий – определение жизнеспособности проекта:

1. *Определение и уменьшение рисков критичных для жизнеспособности системы*
2. *Выделение ключевых требований –> моделирование прецедентов -> получение архитектуры-кандидата*
3. *Получение оценки: стоимости, сроков разработки, качества разрабатываемого продукта*
4. *Составление бизнес плана подтверждающего возможность выполнимость проекта*

Фаза проектирования

Фаза проектирования (Elaboration)

На этой фазе критерий – возможность построить систему

1. *Определение и снижение рисков существенно влияющих на конструирование системы*
2. *Определение большинства прецедентов, описывающих функциональность системы*
3. *Переход от архитектуры-кандидата к исполняемому фундаменту архитектуры*
4. *Подготовка детального плана проекта позволяющего выполнить фазу построения (construction)*
5. *Уточнение предварительной оценки стоимости*
6. *Законченный бизнес план – проект выгодно делать*

Фаза построения

Фаза построения (construction)

Критерий – способность системы начать работать в среде Пользователя

- *Достигается выполнением итераций и выпуском сборки (build) в конце каждой итерации*
- *В конце каждой сборки работоспособная функционально законченная версия системы*

Фаза построения

Фаза построения (construction)

Критерий – способность системы начать работать в среде Пользователя

- *Достигается выполнением итераций и выпуском сборки (build) в конце каждой итерации*
- *В конце каждой сборки работоспособная функционально законченная версия системы*

Фаза внедрения

Фаза внедрения (transition)

Критерий – система готова к эксплуатации

- *Достигается модификацией системы в результате получения требований от пользователя не определенных на других фазах разработки*
- *Исправлением дефектов замеченных пользователем*

Итерактивность и инкрементальность. Разработка небольшими шагами

- *У команды небольшие планы*
- *Спецификация, проектирование и реализация также небольшие*
- *Сборка, тестирование и исполнение кода в каждой итерации для каждого инкремента также небольшие*

При этом

- *Маленький шаг реализует функциональность необходимую пользователю*
- *Команда получает обратную связь от пользователя*
- *Снижаются риски вызывающие беспокойство*

Достоинства итеративной и инкрементальной разработки (1)

- *Ранняя обработка критических и опасных рисков*
- *Ускорение разработки устойчивой архитектуры с помощью которой будет управляться дальнейшая разработка*
- *Появляется скелет системы упрощающий сбор дополнительных требований к системе. Пользователю проще понять систему не виде документации, а в виде прототипа*
- *Система разрабатывается постепенно, а не на последнем этапе разработки когда изменения в требованиях дороги. Пользователь проще оценить прогресс в разработке*
- *Процесс разработки строго регламентирован – увеличивается эффективность разработки*

Достоинства итеративной и инкрементальной разработки (2)

- *Ранняя обработка критических и опасных рисков*
- *Ускорение разработки устойчивой архитектуры с помощью которой будет управляться разработка*
- *Появляется скелет системы упрощающий сбор дополнительных требований к системе*
- *Система разрабатывается постепенно, а не на последнем этапе разработки когда изменения в требованиях дороги*
- *Процесс разработки строго регламентирован – увеличивается эффективность разработки*

Достоинства итеративной и инкрементальной разработки (3)

- *Возможность раннего обучения*
  - *После нескольких итераций каждый в команде понимает смысл различных потоков работ (workflows)*
  - *Легче обучать новых людей. Не требуется специальных «пилотных проектов» для новичков, которые затем выбрасываются*
  - *Ошибки новичков еще не критичны*
  - *Упрощается обучение новым технологиям. Риск их использования на ранних итерациях невелик => стимулируется использование этих технологий*

Достоинства итеративной и инкрементальной разработки (3)

- *Возможность раннего обучения*
  - *После нескольких итераций каждый в команде понимает смысл различных потоков работ (workflows)*
  - *Легче обучать новых людей. Не требуется специальных «пилотных проектов» для новичков, которые затем выбрасываются*
  - *Ошибки новичков **еще** не критичны*
  - *Упрощается обучение новым технологиям. Риск их использования на ранних итерациях невелик => стимулируется использование этих технологий*

Итеративный процесс управления рисками. Риски представления системы

Риск – опасность для разработки, степень которой еще неизвестна

Риски:

- *Идентифицируются*
- *Получают приоритеты*
- *Отслеживаются*

Примеры рисков представления системы

- *Скорость работы*
- *Память используемая системой*
- *Точность вычислений*
- *Надежность*
- *Доступность*

Итеративный процесс управления рисками. Организационные (не технические) риски

- *Организация не имеет людей опытных в уникальных областях необходимых для построения системы*
- *Предлагается новый язык реализации, но не все его знают*
- *Заказчик требует слишком быстрого появления системы*
- *Необходимо включать в разработку внешние фирмы*

Итеративный процесс управления рисками. Работа с рисками

После идентификации рисков и задания их приоритетов

- *Уход от риска, перепроектировав проект или изменив требования к системе*
- *Ограничить риск, что бы при этом затрагивалась по возможности меньшая часть системы или проекта*
- *Материализация риска, проверив оправдались ли опасения. Затем уход от риска или ограничение риска*
- *Риск оправдался. Если исправить невозможно, то возможен отказ о проекта с минимальными затратами времени и денег, до вовлечения в проект большого числа людей*

Типовая итерация

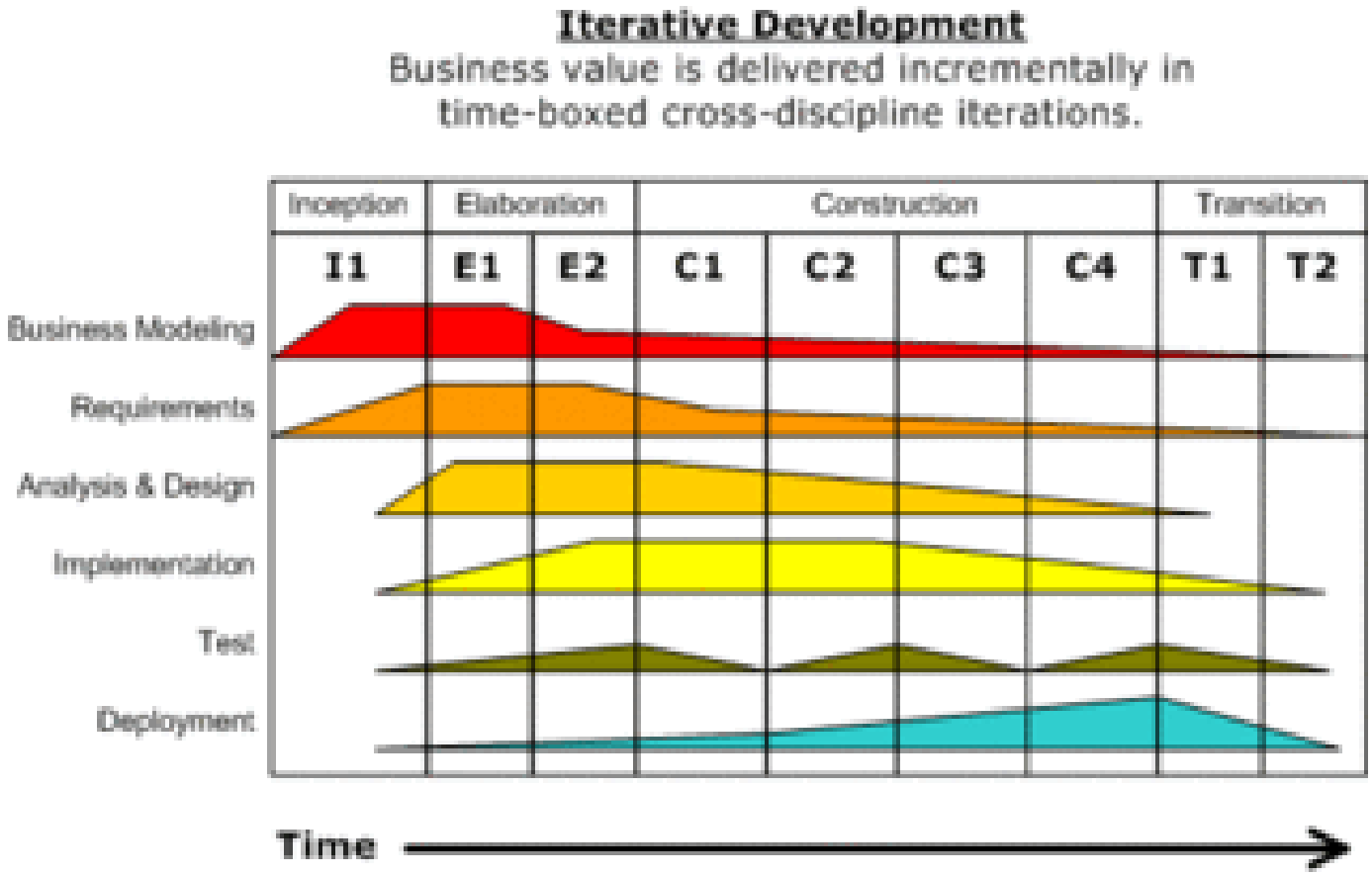
- *Итерация – мини-проект при полном проходе по всем потокам работ (workflows)*
- *Поток работ - кооперация между исполнителями (workers) производящими и использующими артефакты*
- *Пять основных потоков работ:*
  - *Определение требований*
  - *Анализ*
  - *Проектирование*
  - *Реализация*
  - *Тестирование*

Отличие итеративного процесса от модели водопада (1)

- *Между итерациям существует перекрытие*

Распределение сбора артефактов по различным фазам унифицированного процесса

Сдвиг важности каждого потока работ в течении итераций и фаз процесса разработки



Проверка команды с помощью итеративного подхода

- *Тенденция переходить к написанию кода.*
  Строки кода – то, что менеджеры способны сосчитать
- *Сопротивление изменениям*
  Изменения замедляют написание кода
- *Нет заинтересованности в многократном использовании анализа, проекта или кода*
  Возможно потому что работу оценивают по написанию нового кода

Изменение

- *Теперь основное:*
  - *Уменьшение рисков*
  - *Создание фундамента архитектуры*
  - *Реализация функциональности*
- *Прогресс изменяется в рисках, прецедентах, и компонентах реализующих прецедент*

Последствия итеративного подхода

- *Для создание бизнес-плана в фазах анализа и определения требований, необходимо снижать риски и создание демонстрационной версии системы*
- *Для создания законченного бизнес плана в конце фазы Elaboration организация должна быть уверена что не осталось невыясненных рисков и спланировать дальнейшие действия исходя из существующего фундамента архитектуры и требований к системе*
- *Для минимизации затрат организация должна использовать многократно используемые компоненты полученные в результате ранней разработки архитектуры*

Последствия итеративного подхода

- *Для избегания задержки разработки и роста цены разработки организация должна делать сначала сложные вещи*

- *Для избегания устаревания продукта к моменту выхода на рынок организация должна соглашаться на внесение изменений в систему.*

  Разбитый на фазы Итеративный подход позволяет вносить изменения в течение всего времени разработки системы

Типовая итерация

- *Исполнители и артефакты могут участвовать более чем в одном потоке работ.*

  Например, разработчик компонент (component engineer) участвует в потоках работ по анализу, проектированию и реализации

- *После завершения итерации*
  - *Регрессионное тестирование*
  - *Оценка изменений требований*
  - *Оценка появления новых требований*

Поток работ для получения требований к системе как сценариев использования системы.

Сбор требований к системе как построение модели прецедентов использования системы

- Основная задача потока работ по определению требований – построение модели *прецедентов использования* системы
- Функциональные требования структурируются на прецеденты использования
- Нефункциональные требования привязываются к конкретным прецедентам использования системы
- Общие для нескольких прецедентов нефункциональные требования выделяются в отдельный документ называемый *дополнительные требования* (supplementary requirements)

Построение модели прецедентов

- Построение модели прецедентов требует от *системного аналитика* привязывать функциональность системы к конкретному *пользователю системы* – *актеру* (*actor*)
- Требует от системного аналитика выявлять категории *пользователей* системы - *актеров*
- Какие задачи (*прецеденты использования*) должна выполнить система для конкретных *категорий пользователей*
- Затем построенная *модель прецедентов* управляет всей разработкой

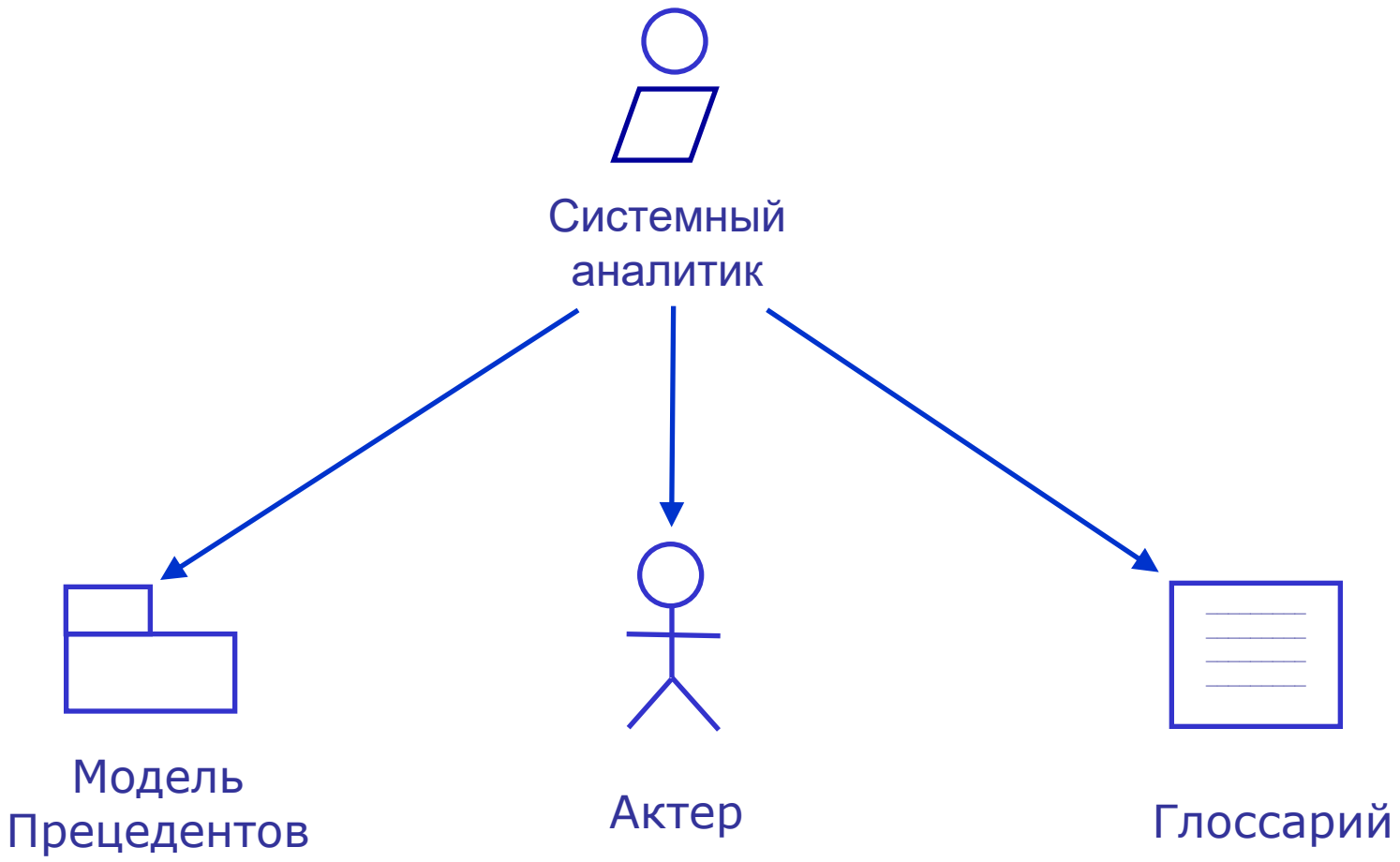
Построение модели прецедентов
Схема описания потока работ

1. Описание артефакта (artifact). Артефакт – общий термин для любого вида информации (создаваемой, изменяемой, используемой) исполнителями (workers) при работе по созданию системы

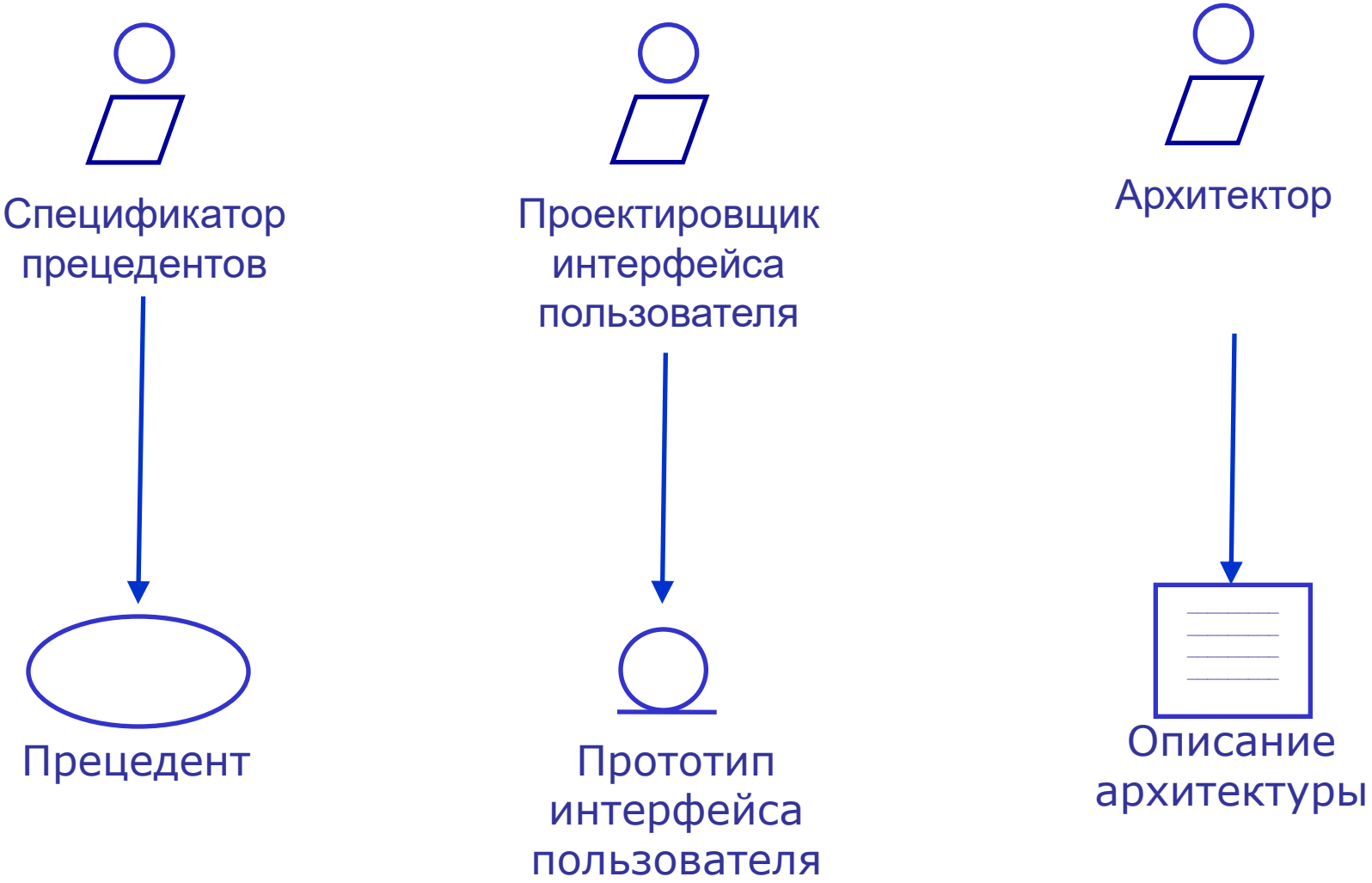
2. Описание исполнителя (worker) – сотрудника фирмы, участник потока работ. Обозначение роли которая может быть поручена одному человеку или группе лиц имеющих для этого соответствующие навыки

3. Описание деятельности (activity) – часть работы за выполнение которой отвечает исполнитель в потоке работ и имеет результат в виде набора артефактов

Построение модели прецедентов
Исполнители и артефакты (1)



Построение модели прецедентов
Исполнители и артефакты (2)



Артефакты в потоке работ по построению модели прецедентов (1)

Модель прецедентов

- Позволяет разработчикам системы *договориться* с заказчиками и пользователями системы о требованиях к разрабатываемой системе

- Позволяет *заключить соглашение* (*контракт*) на разработку системы

- Структурировать требования на пакеты содержащие прецеденты и диаграммы прецедентов

- Модель прецедентов - исходная модель для построения моделей анализа, проектирования, реализации, тестирования

Артефакты в потоке работ по построению модели прецедентов (2)

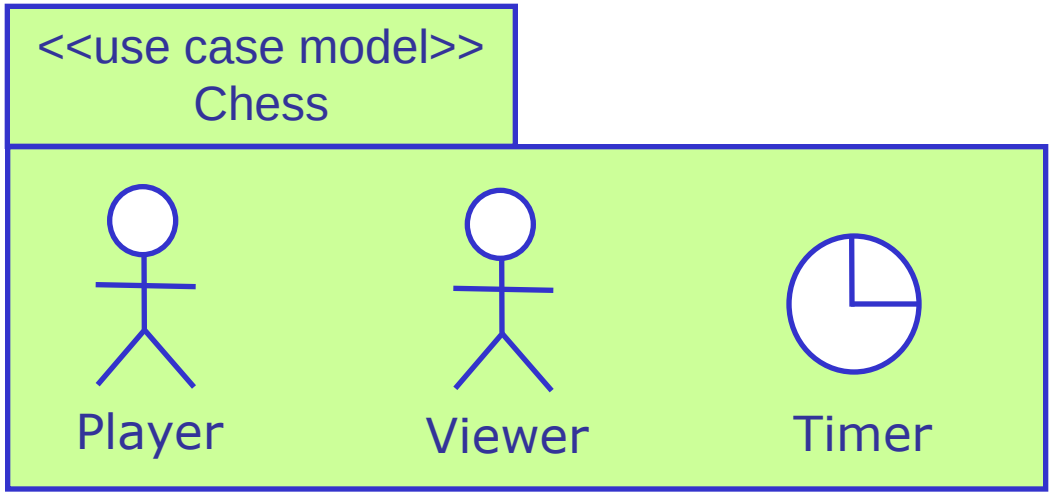
Актер

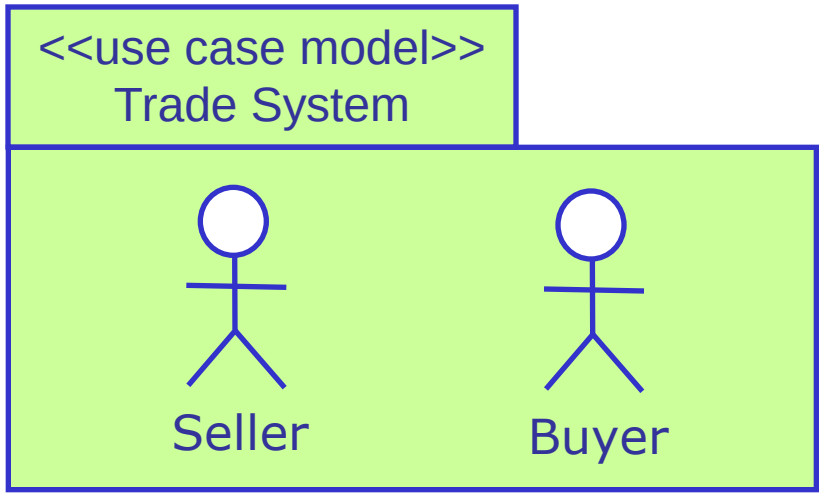
- Модель прецедентов описывает действия системы

для каждой *категории пользователей* – *актеров*

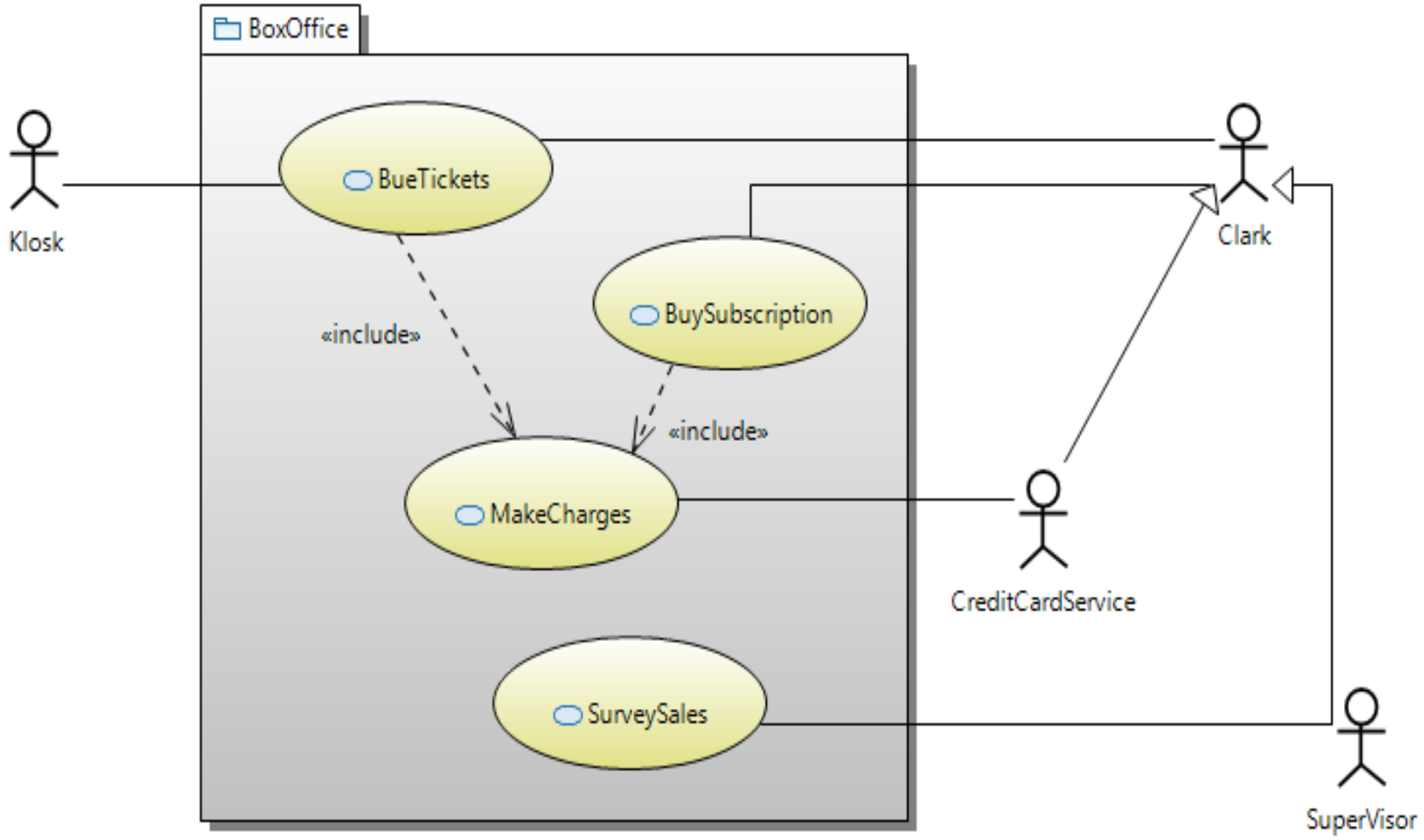
- Актером может быть внешняя программная система
- Актером может быть внешнее устройство: часы, телефонная карта, …
- Актерами могут быть сотрудники или клиенты фирмы для которой разрабатывается система.
  И сотрудники, и клиенты могут быть взяты из бизнес модели

Построение модели прецедентов. Примеры артефактов - актеров

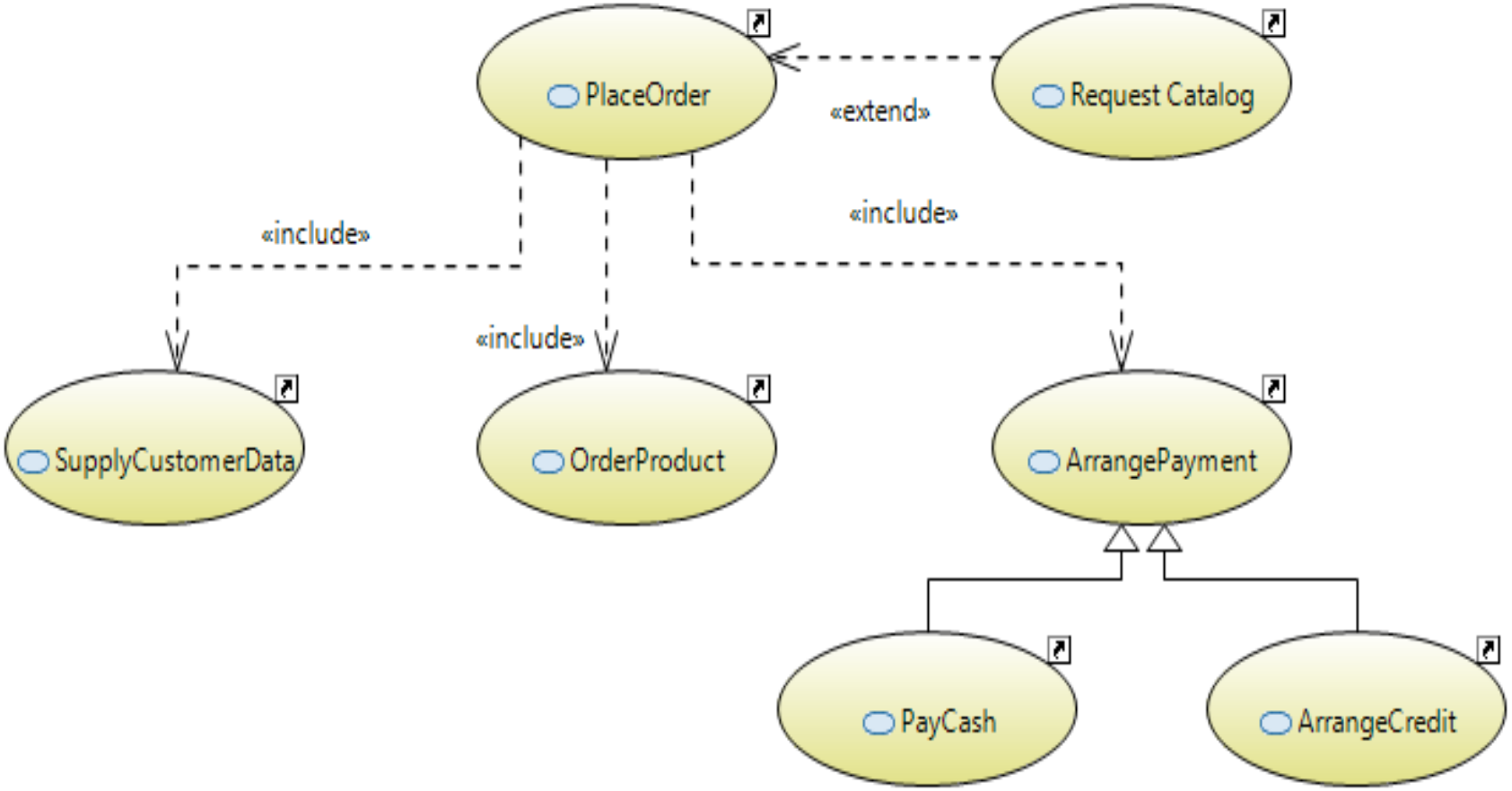




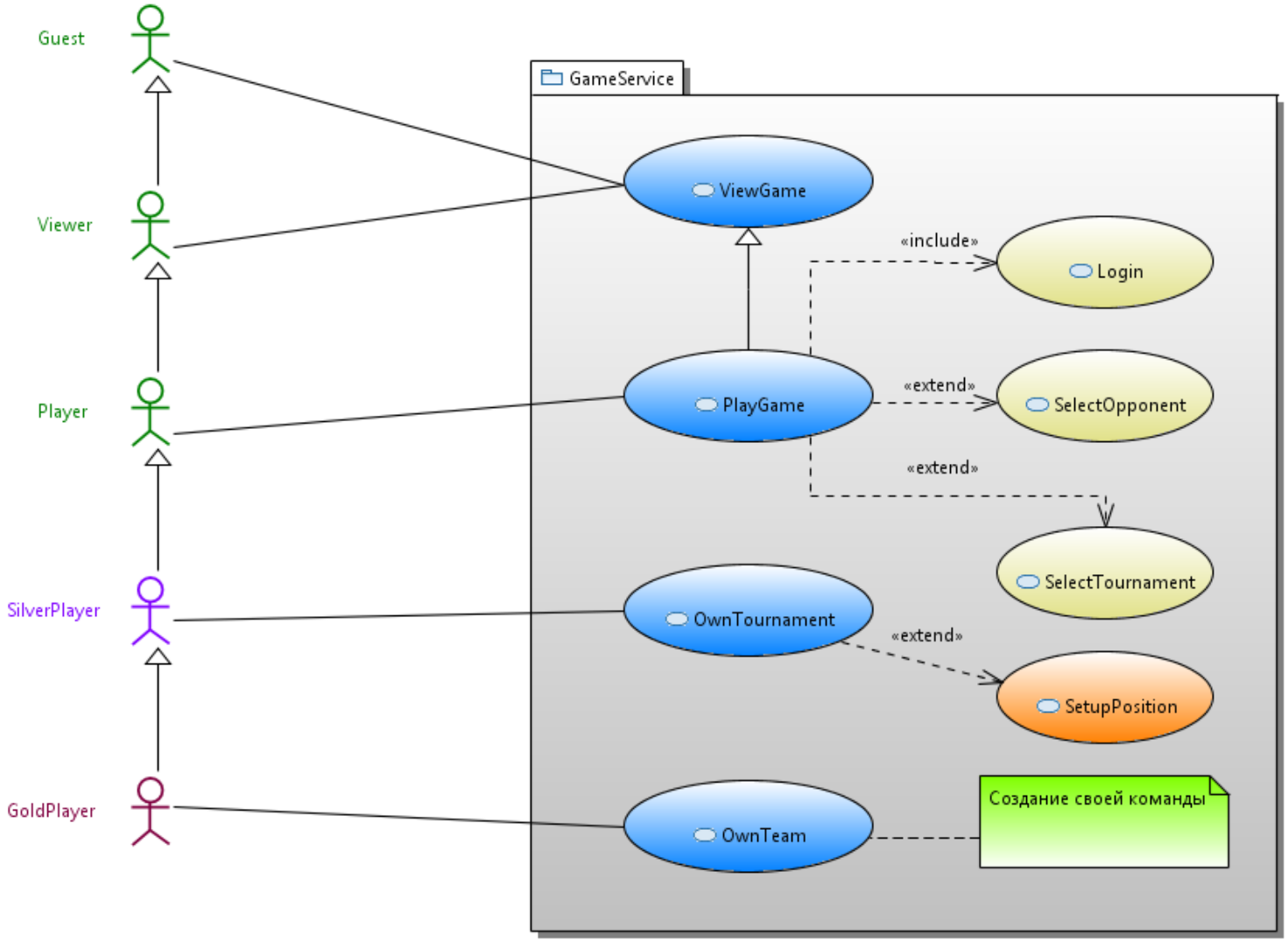
Построение модели прецедентов.
Примеры артефактов – актеров и прецедентов



Построение модели прецедентов. Примеры отношений прецедентов



Построение модели прецедентов. Сервис настольных игр



Артефакты в потоке работ по построению модели прецедентов (1)

Прецедент

- Прецедент – способ использования системы.
- Описывает *последовательность действий* с альтернативами, которые система может выполнить взаимодействуя с актерами
- Прецедент – классификатор для описания поведения => имеет атрибуты и операции
- Прецедент может включать :
  - диаграммы переходов и состояний,
  - диаграммы деятельности,
  - диаграммы взаимодействия
  - диаграммы последовательности взаимодействия

Артефакты в потоке работ по построению модели прецедентов. Прецедент (2)

Прецедент

- *State Transition Diagram* – описывает жизненный цикл прецедентов в терминах состояний и переходов
- *Activity Diagram* – более подробное описание, описывая деятельности выполняемые при переходе на диаграмм состояний и переходов
- *Collaboration Diagram* и *Sequence Diagram* – используются для описания взаимодействия между типовым экземпляром актера и типовым экземпляром прецедента

Артефакты в потоке работ по построению модели прецедентов. Прецедент (3)

Типовой жизненный цикл прецедента

- Инициализация прецедента и установка прецедента в начальное состояние
- Активизация прецедента внешним сообщением
- Переход в другое состояние, с выполнением некоторой последовательности действий. Эта последовательность может содержать вычисления, выбор пути на диаграмме и посылку сообщения актеру
- Ждет в новом состоянии прихода нового сообщения от актера
- Активизация новым сообщением и повторение цикла.

Артефакты в потоке работ по построению модели прецедентов. Прецедент (3)

Особенности прецедента

- Атрибуты прецедента локальны для экземпляра прецедента (не могут быть использованы другими экземплярами прецедентов). Взаимодействие прецедента только с актерами.
- Нет конфликта между прецедентами из-за данных.
- Не требуется синхронизация с другими прецедентами
- Прецедент выполняется полностью или не выполняется совсем.
- Цель – сделать описание простым, что бы модель можно было обсуждать с пользователем и заказчиком.

Артефакты в потоке работ по построению модели прецедентов. Описание архитектуры

Описание архитектуры (модель прецедентов)

- Важная и критичная функциональность
- Важные требования, которые должны быть реализованы в начале жизненного цикла разработки системы
- Описание архитектуры используется для задания приоритетов в реализации прецедентов.

  Использование будет описано при рассмотрении *деятельностей* потока работ по построению модели прецедентов

Артефакты в потоке работ по построению модели прецедентов. Глоссарий

Глоссарий

- Определение важных или общих для всего проекта понятий
- Важен для получения согласованных определений приемлемых пользователями, заказчиками, аналитиками и разработчиками системы
- Может возникать из бизнес модели

Артефакты в потоке работ по построению модели прецедентов. Прототип интерфейса

Прототип интерфейса пользователя

- Прототип интерфейса помогает сбору требований к системе
- Позволяет построить лучший интерфейс системы, с которой пользователь будет работать основную часть рабочего времени
- Способствует лучшему пониманию взаимодействия актеров и прецедентов при построении модели прецедентов
- При построении прототипа интерфейса могут использоваться артефакты:
  модели интерфейса пользователя и скриншоты

Построение модели прецедентов. Прототип интерфейса пользователя

Latvian gambit

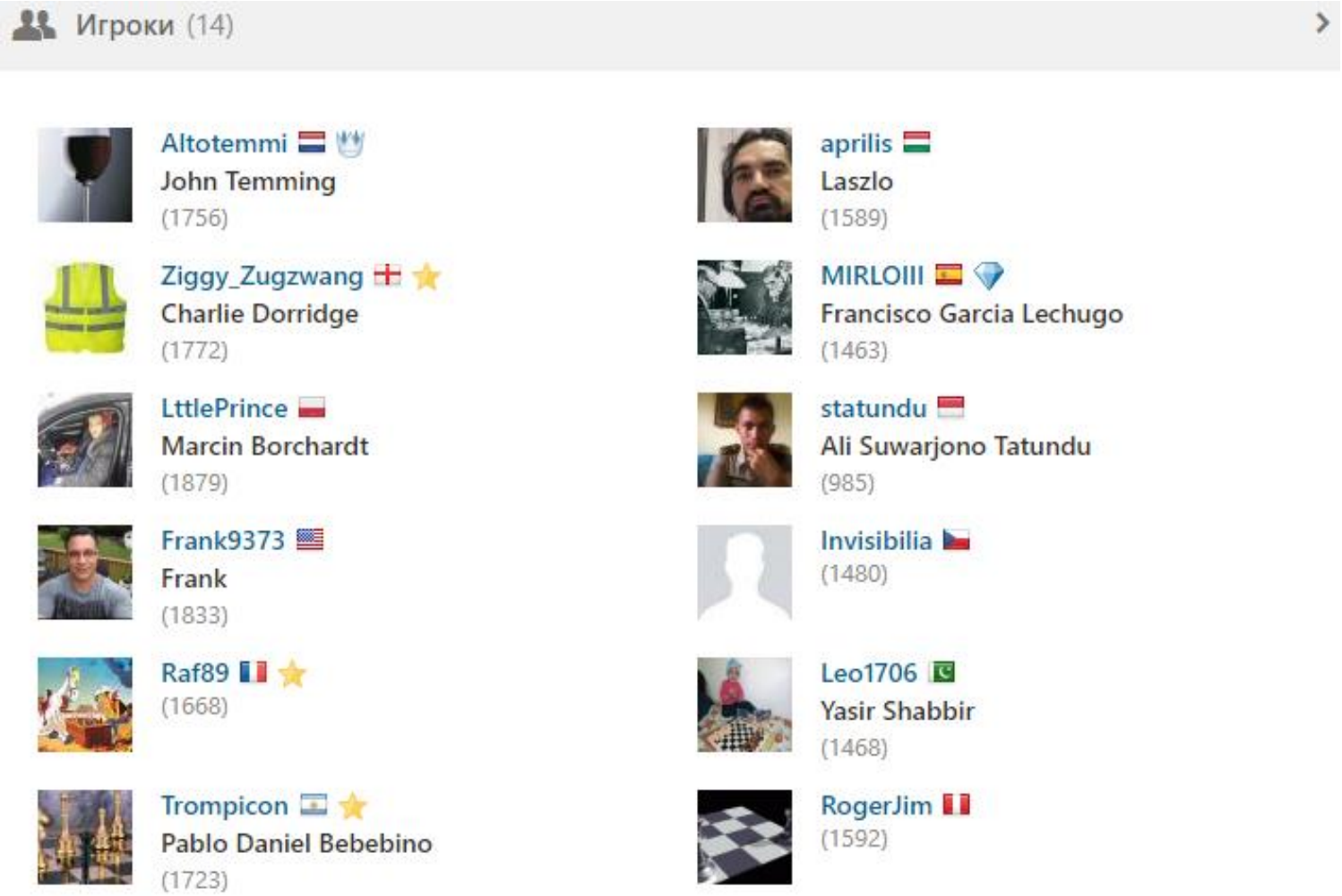
Директор турнира: CM Leo_C

Присоединиться

Контроль времени	**3 дня**	Игроки	**14/60**
Рейтинговые партии	**Да**	Средний рейтинг	**1605**
Диапазон рейтинга	**> 1400**	Баллов доступно	**539**
Макс. размер группы	**6**	Завершено	**0%**
# В следующий этап	**2**	раунд	**0**
Партий одновременно	**10**	Завершенные партии	**0**
КБ	**Да**	Оставшиеся партии	**0 (текущий раунд)**
Макс. ср. время на ход	∞	Макс. поражений по времени %	**20%**

The latvian gambit is a fascinating opening. Although it is not entirely sound it creates a very fun game!

Построение модели прецедентов. Прототип интерфейса пользователя



Построение модели прецедентов. Прототип интерфейса пользователя

https://www.chess.com/club/matches/team-russia/908450

https://www.chess.com/club/matches/917504

Исполнители в потоке работ по построению модели прецедентов.

Исполнитель (worker) – абстракция человека с некоторыми способностями и знаниями

Это не конкретный человек (один человек – много исполнителей)

Это не должность. Должность одна – исполнителей много

Для исполнителя выполняется:

- Описание ответственности исполнителя
- Описание предполагаемого поведения исполнителя

Системный аналитик (System analyst)

Спецификатор прецедентов (Use Case Specifier)

Проектировщик интерфейса пользователя (UI Designer)

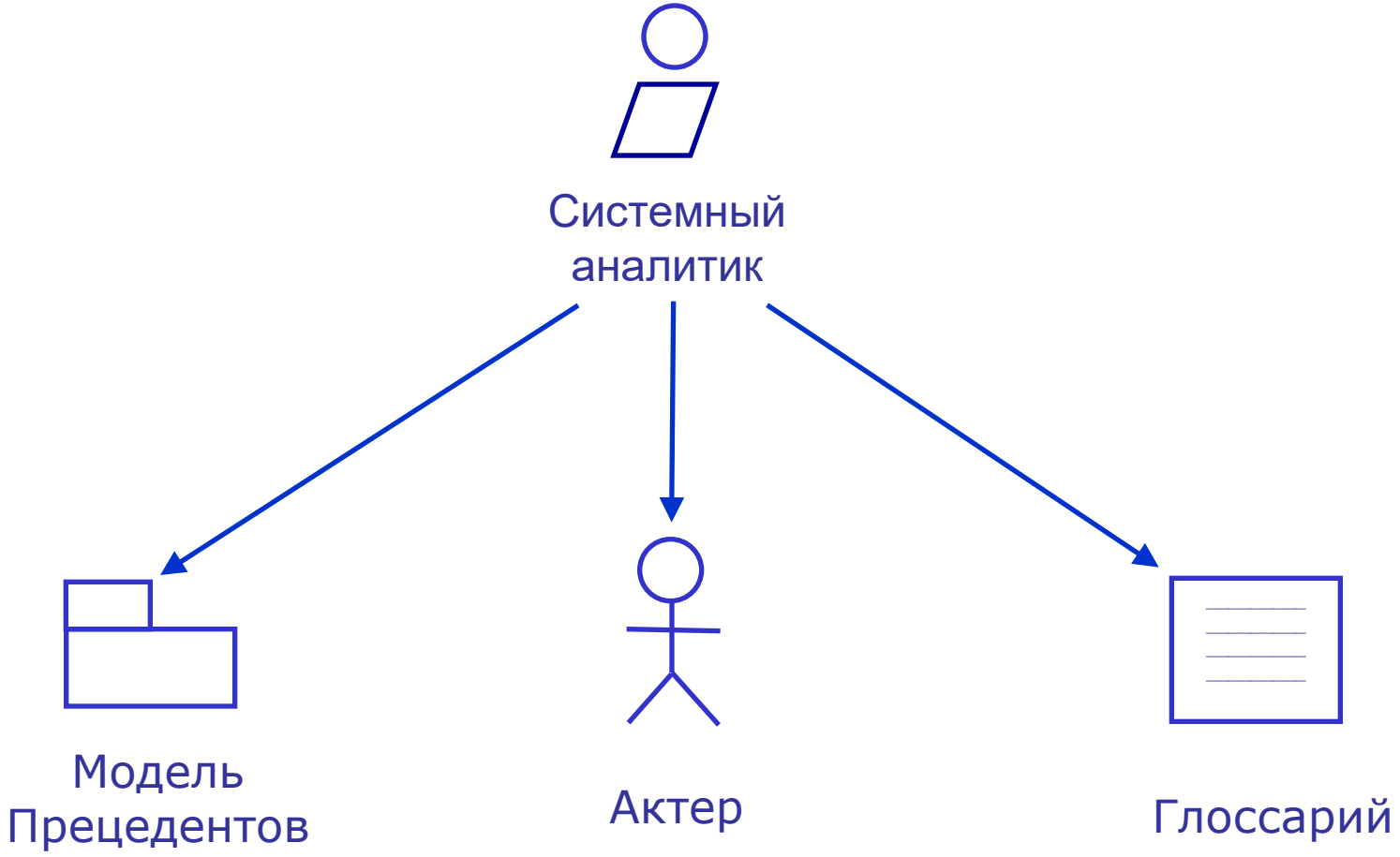
Исполнители в потоке работ по построению модели прецедентов. Системный аналитик

Системный аналитик

- Отвечает за все множество требований, которые моделируются как прецеденты, включая:
  - функциональные требования и
  - нефункциональные требования относящиеся к прецеденту
- Отвечает за ограничение системы
- Отвечает за поиск актеров и прецедентов
- Обеспечивает полноту и согласованность модели прецедентов

- Не отвечает за каждый конкретный прецедент (для этого есть спецификатор прецедента)

Построение модели прецедентов
Исполнители и артефакты (1)



Исполнители в потоке работ по построению модели прецедентов. Системный аналитик

Системный аналитик

- Отвечает за все множество требований, которые моделируются как прецеденты, включая:
  - функциональные требования и
  - нефункциональные требования относящиеся к прецеденту
- Отвечает за ограничение системы
- Отвечает за поиск актеров и прецедентов
- Обеспечивает полноту и согласованность модели прецедентов

- Не отвечает за каждый конкретный прецедент (для этого есть спецификатор прецедента)

Исполнители в потоке работ по построению модели прецедентов. Спецификатор прецедентов

Спецификатор прецедентов

- Отвечает за детальное описание одного и более прецедентов
- Спецификатор прецедентов должен работать с реальными пользователями и заказчиками системы
- Детально описывает на языке UML взаимодействие актеров и прецедентов

Исполнители в потоке работ по построению модели прецедентов. Проектировщик интерфейса пользователя

Проектировщик интерфейса пользователя

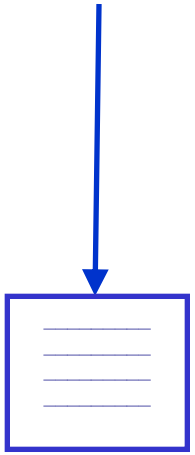
- Обычно разрабатывает по одному прототипу интерфейса пользователя для каждого актера

Исполнители в потоке работ по построению модели прецедентов. Архитектор

Архитектор

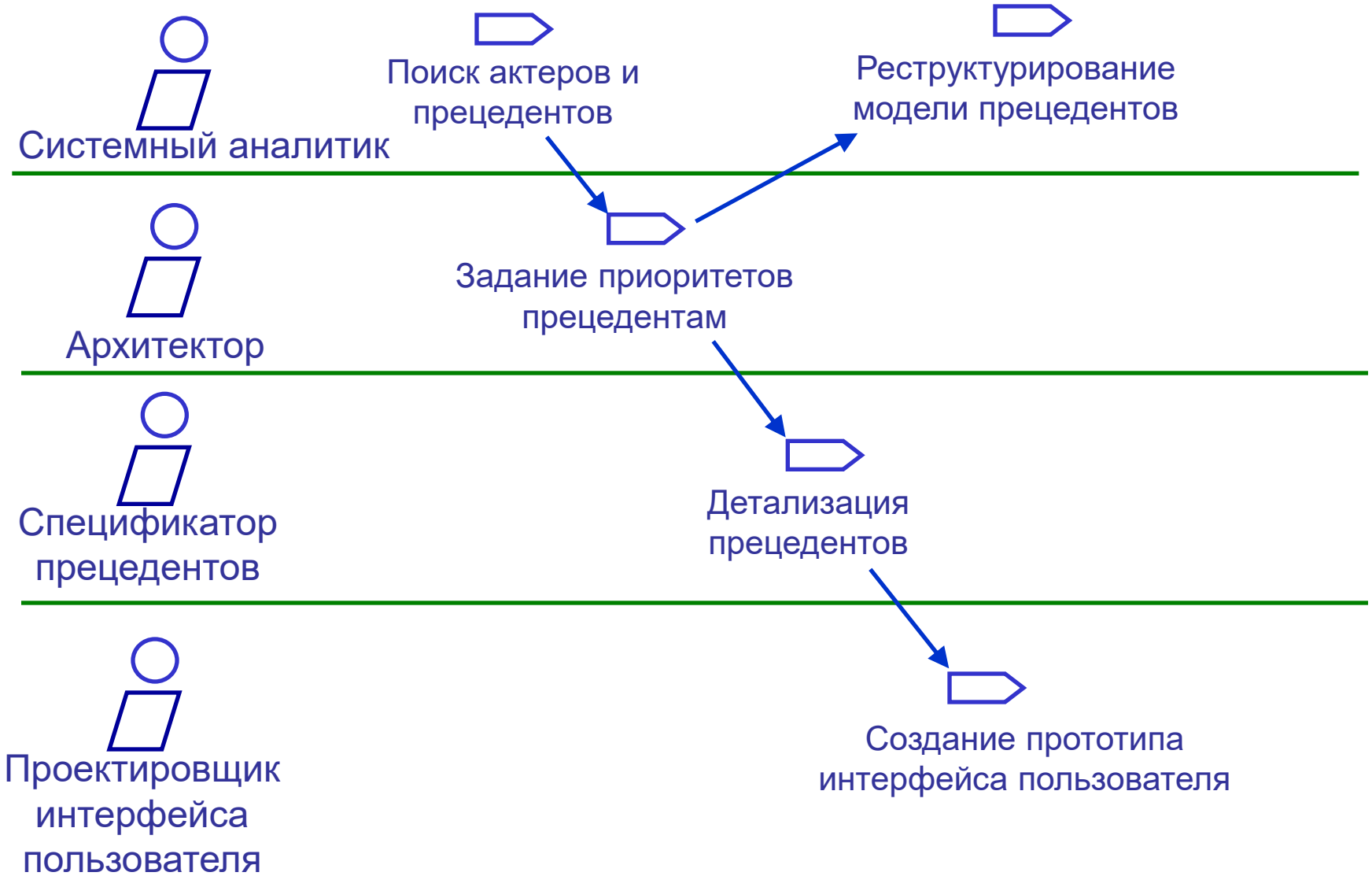
Архитектор

- Разрабатывает описание архитектуры в виде модели прецедентов



Описание архитектуры

Построение модели прецедентов
Поток работ (workflow)



Обзор потока работ по построению модели прецедентов (1)

Каждая деятельность может быть пройдена несколько раз. Каждый раз при этом будет выполнена лишь часть работы текущей итерации.

1. Системный аналитик выполняет *поиск актеров и прецедентов* для подготовки новой версии модели прецедентов. На вход аналитику поступает:
   - Бизнес модель и модель предметной области (domain model)
   - Список возможностей (feature-list)
   - Дополнительные возможности (не функциональные)

2. Архитектор после этого *задает приоритеты прецедентам*, определяя прецеденты важные для создания архитектуры

Обзор потока работ по построению модели прецедентов (2)

3. Проектировщики интерфейса затем *создают прототип интерфейса пользователя* для каждого выявленного аналитиком актера на основе существующих прецедентов

4. Спецификаторы прецедентов выполняют *детализацию прецедентов* в соответствии с заданными архитектором приоритетами прецедентов

5. Системный аналитик выполняет *реструктурирование модели прецедентов*, определяя отношения обобщения и включения между прецедентами

Обзор потока работ по построению модели прецедентов (2)

Результат первого прохода через поток работ – 1-я версия артефактов:

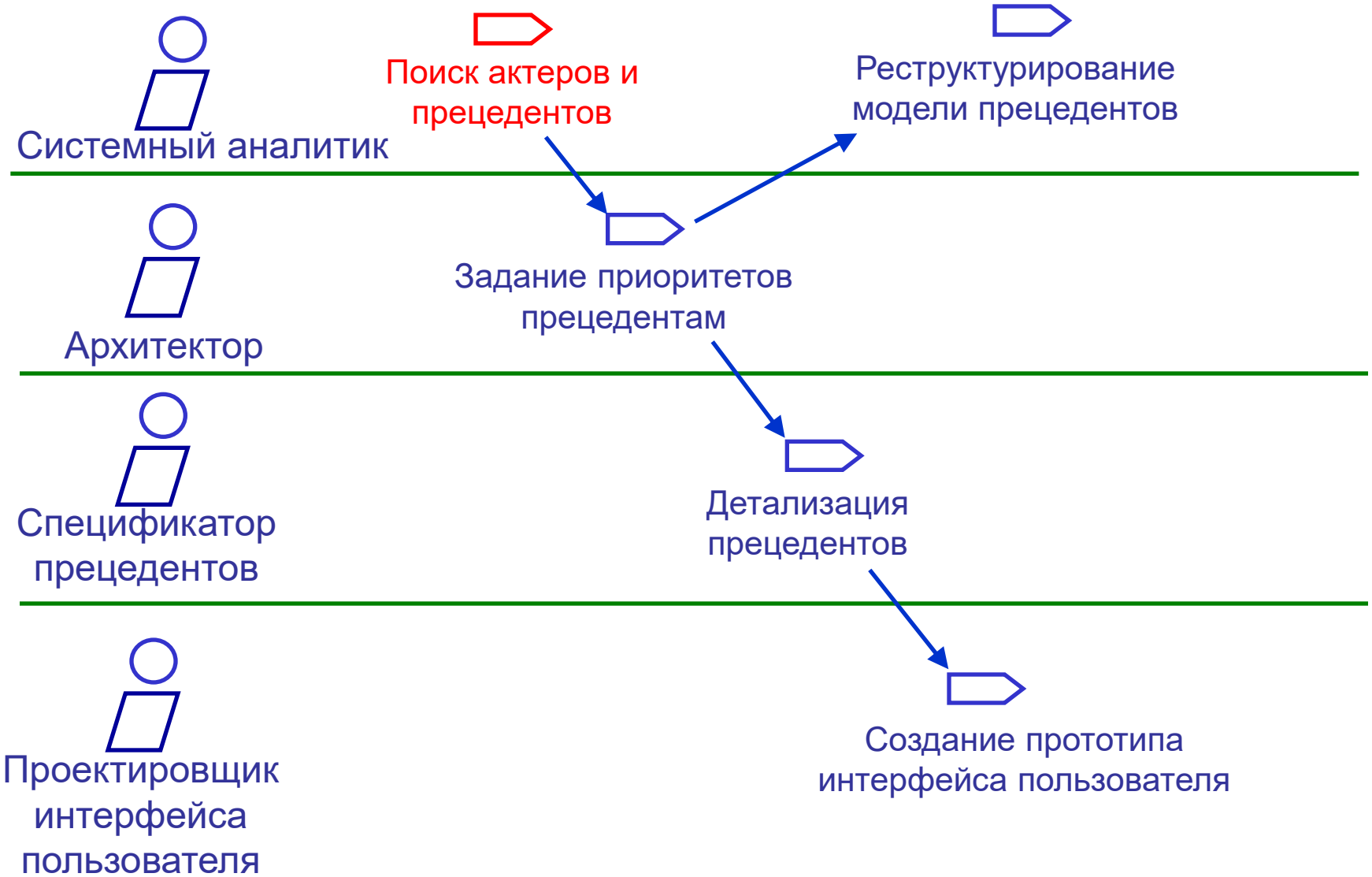
- Модели прецедентов
- Прецеденты
- Прототипы пользователя

Следующие проходы через этот поток работ уточняют эти артефакты.

Различия в проходах по *потоку работ* на различных *фазах*

- В фазах Inception и Elaboration системный аналитик находит много актеров
- В фазе Construction изменения в составе актеров и прецедентов минимальны, но уточняются диаграммы прецедентов и взаимодействие прецедентов и актеров

Построение модели прецедентов
Поток работ (workflow)



Деятельность по поиску актеров и прецедентов

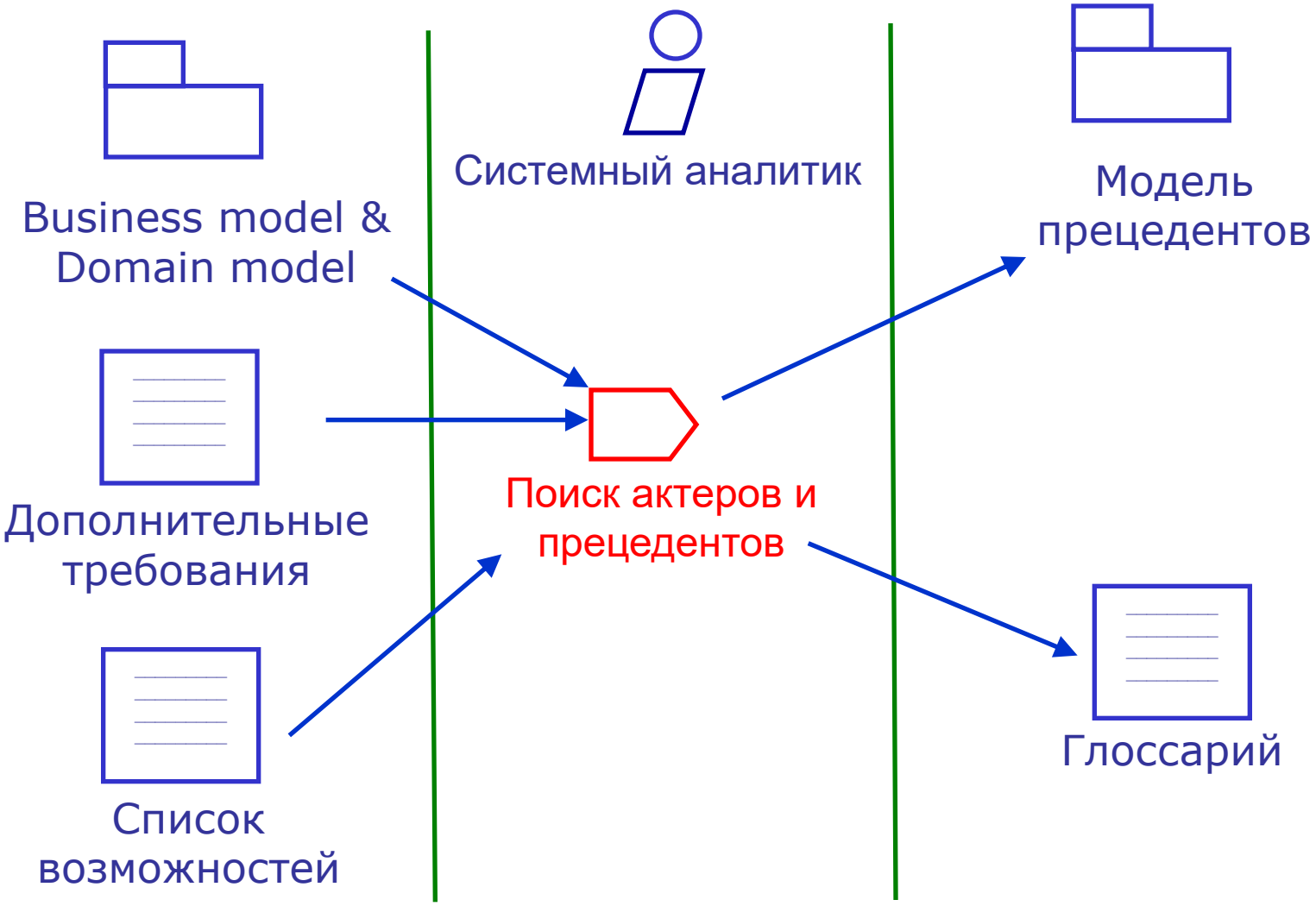
Цели:

1. Отделить систему от окружения
2. Кто или что взаимодействует с системой (актеры)
3. Какая функциональность ожидается от системы
4. Определить глоссарий понятий используемых при детальном описании прецедентов

Поддеятельности:

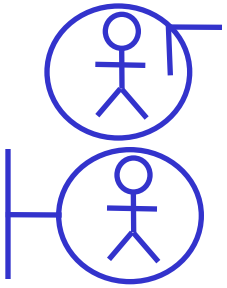
1. Поиск актеров
2. Поиск прецедентов
3. Краткое описание каждого прецедента
4. Описание модели прецедентов в целом

Деятельность по поиску актеров и прецедентов и ее артефакты



Поиск актеров

1. По одному актеру для каждого сотрудника (внутреннего актера) в бизнес-модели
2. По одному актеру для каждого бизнес актера (внешнего актера) в бизнес-модели



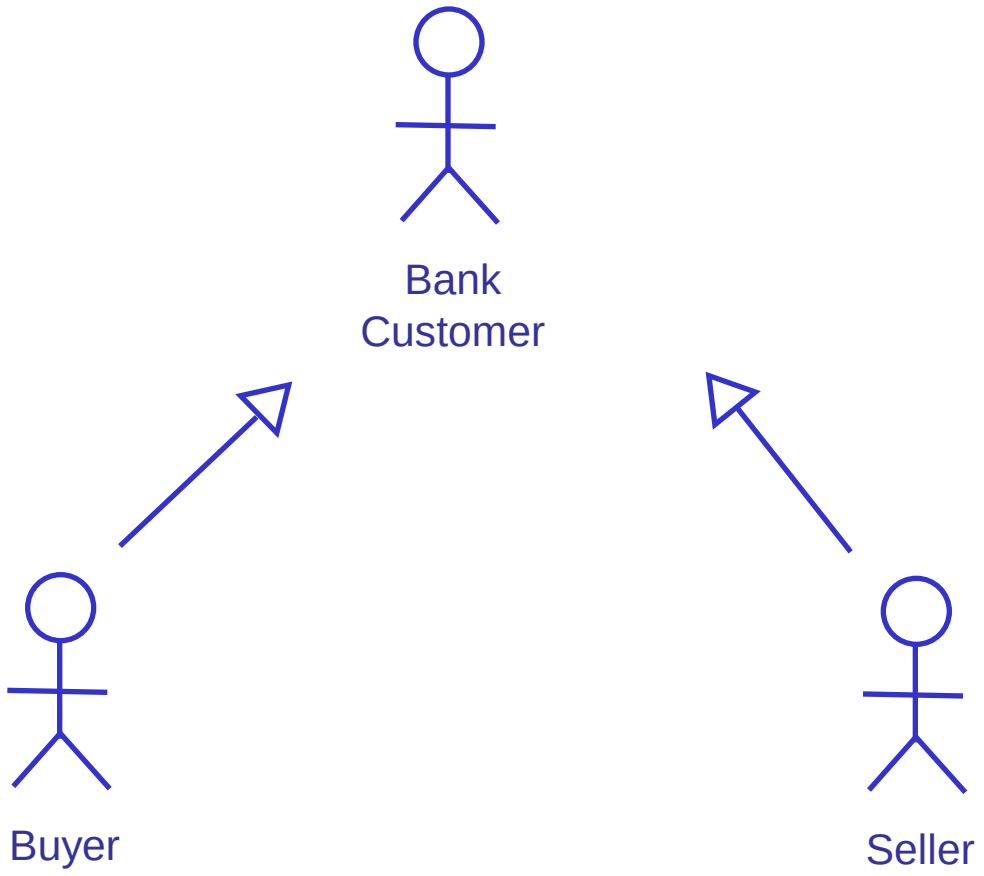
Критерии отбора:

1. Идентификация пользователей и их обобщение в актеров
2. Избегать пересечения ролей у актеров.
   Если роли актеров пересекаются то ввести отношение обобщения у актеров

Результат поддеятельности:

1. Описание роли актера
2. Описание задач, для которых актер использует систему

Поиск актеров. Пример



Поиск прецедентов

1. Прецедент предлагается для каждой роли каждого сотрудника участвующего в реализации бизнес-прецедента, если бизнес модель существует.

2. Системный аналитик предлагает для каждого актера *список прецедентов-кандидатов.* Каждый актер:
   - Создает, изменяет, отслеживает, удаляет бизнес-объекты. Например, заказы и счета.
   - Информирует системы о событиях. Например, если актер – это таймер или календарь, а событие – счет просрочен.
   - Информирует об истечении интервала времени
   - Актеры для:
     - Запуска системы
     - Остановки системы
     - Техобслуживание

Краткое описание каждого прецедента

1. Текстовое описание назначения каждого прецедента
2. Пошаговое описание деятельностей прецедента

Пример

1. Прецедент «оплатить счет» используется актером «покупатель» для пометки счетов к оплате.

2. Затем прецедент «оплатить счет» выполняет платеж в назначенный день

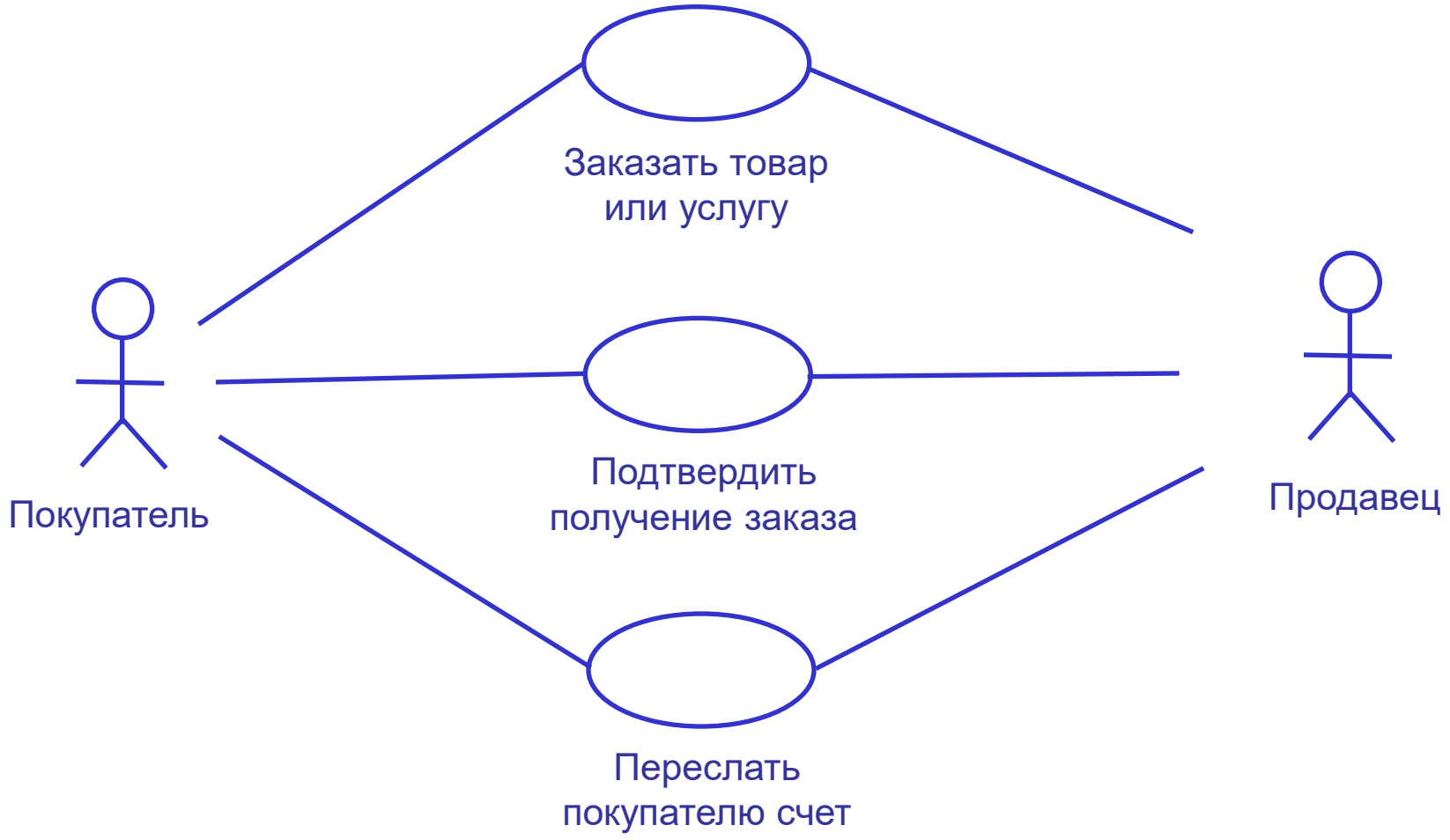
Описание модели прецедентов в целом

Связь прецедентов друг с другом и с актерами.

Диаграммы прецедентов представляющих модель прецедентов

- Диаграмму прецедентов на каждый бизнес-прецедент с описанием его реализации
- Прецеденты для каждого актера
- Группирование прецедентов в пакеты
  - Диаграмма прецедентов пакета
  - Диаграмма связей пакета

Пример диаграммы прецедентов. Отношения ассоциации



Пример диаграммы прецедентов. Отношения обобщения и расширения

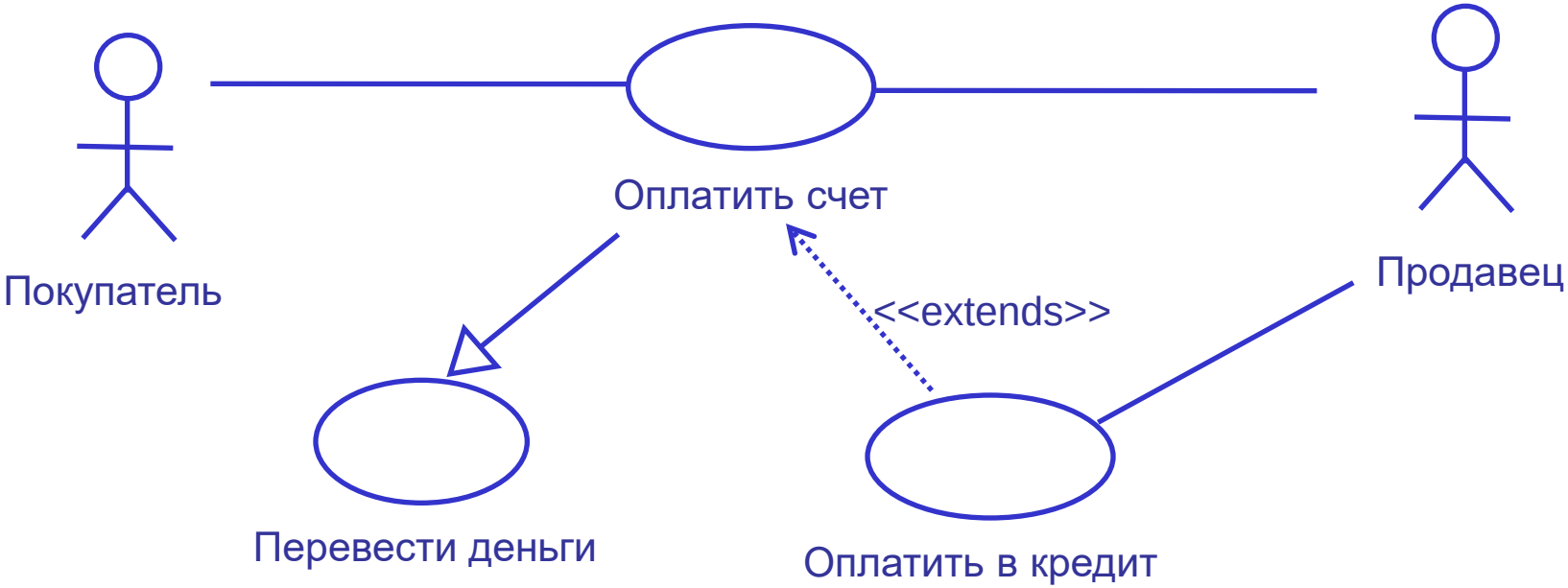
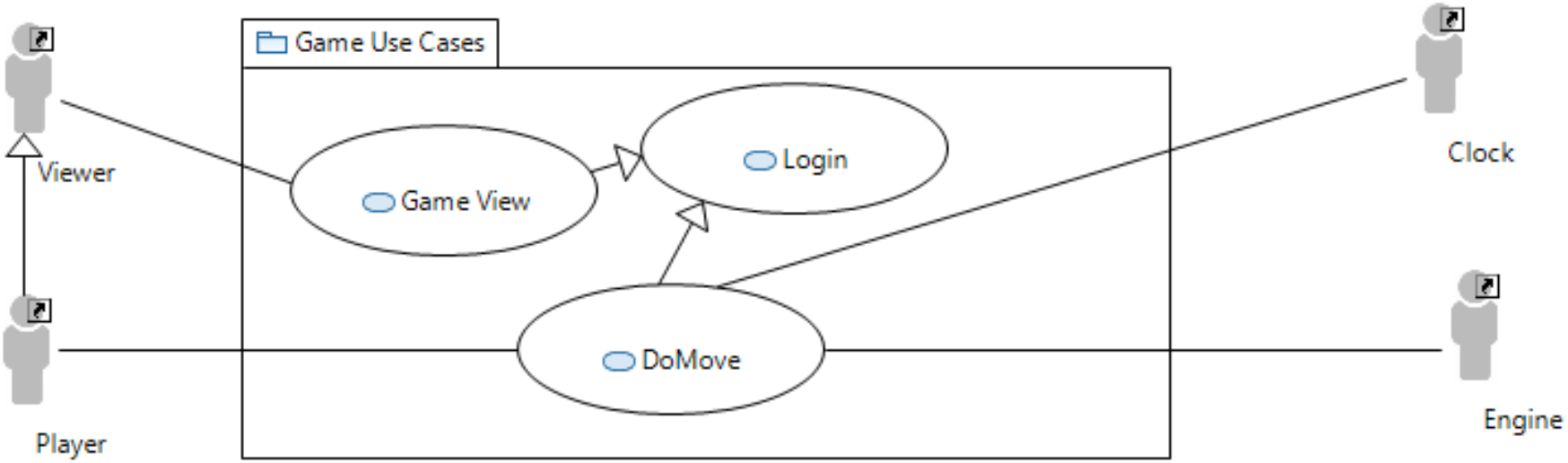
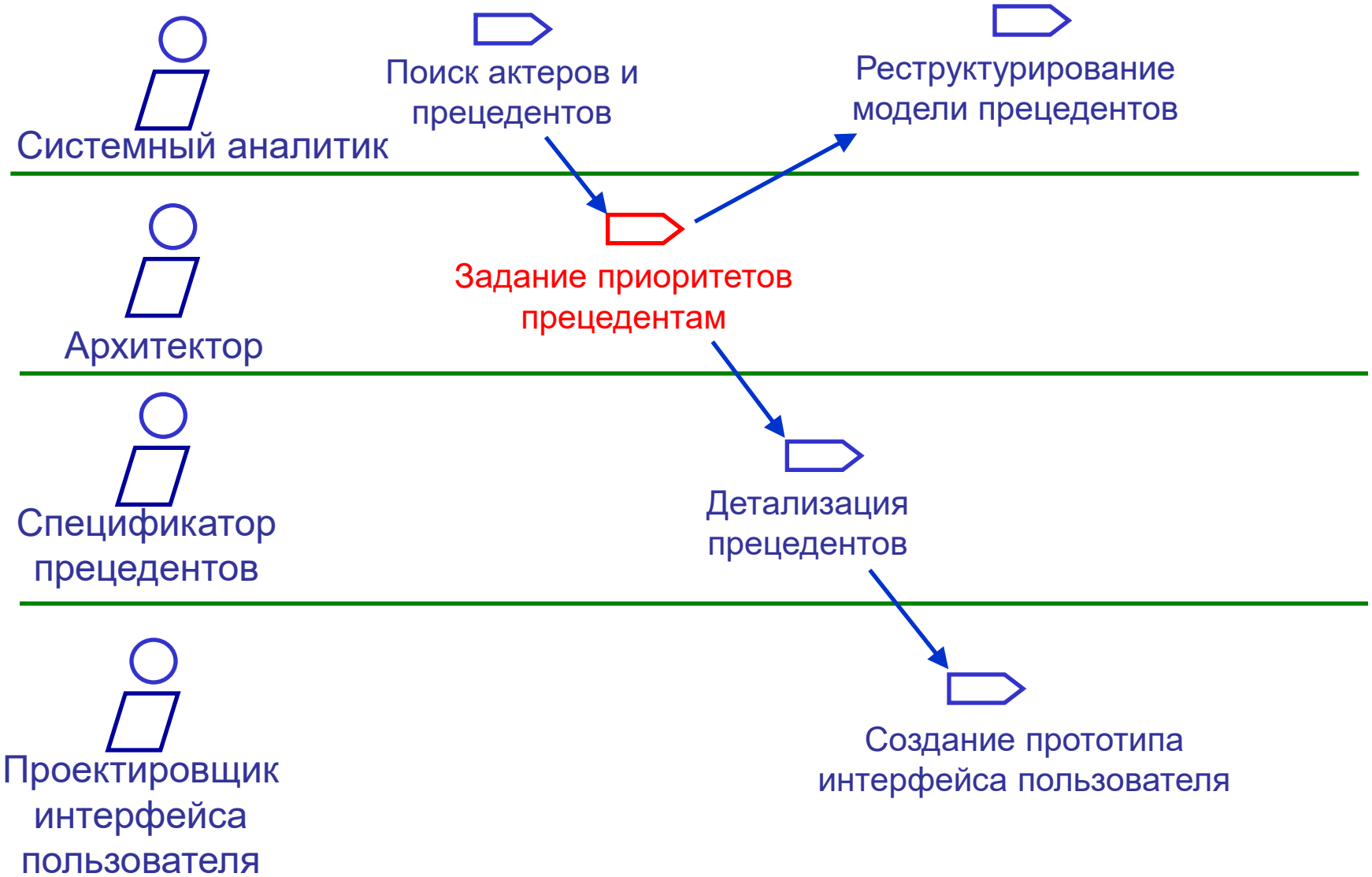


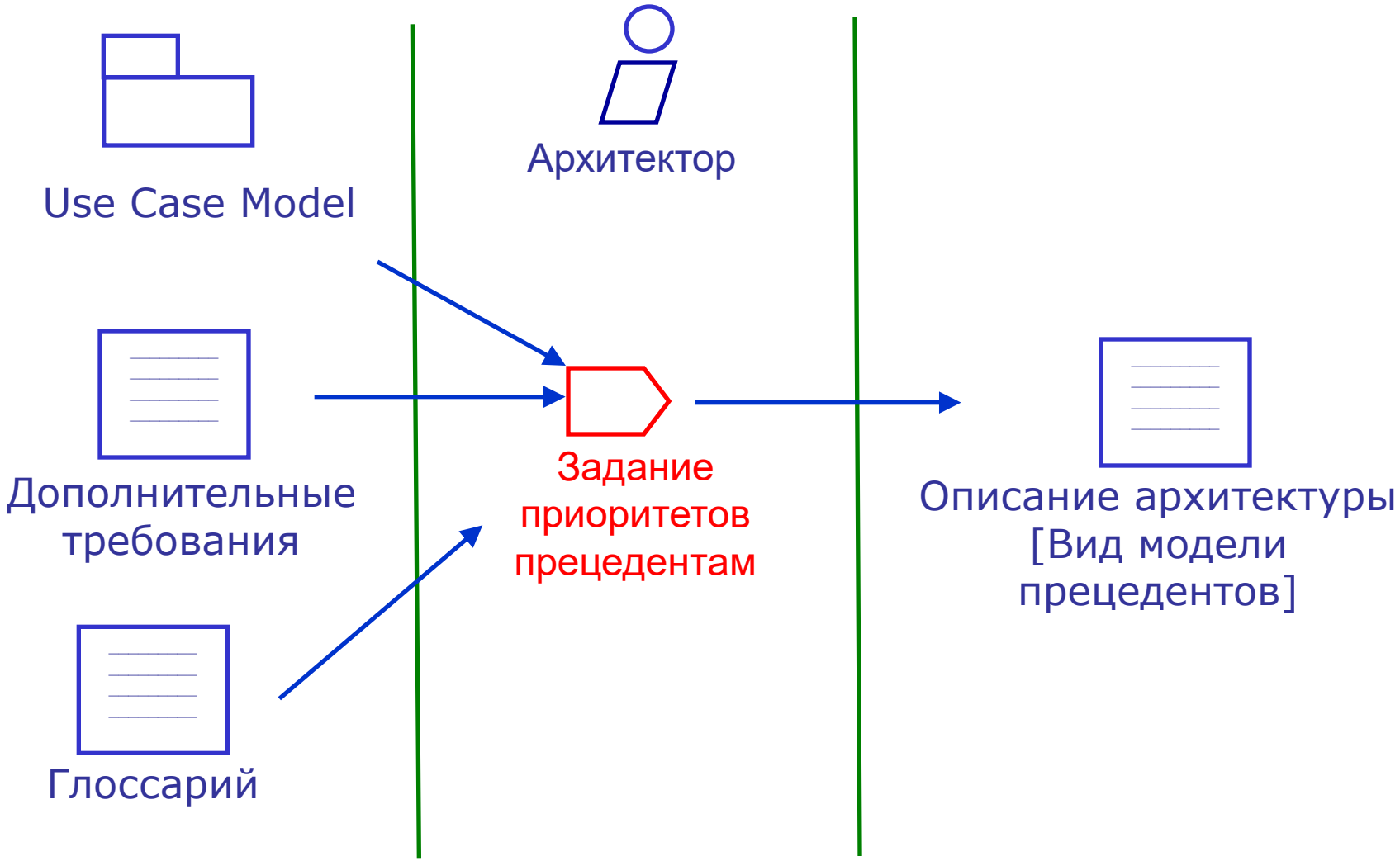
Диаграмма прецедентов. Настольные игры



Построение модели прецедентов
Поток работ (workflow)



Деятельность по заданию приоритетов прецедентам



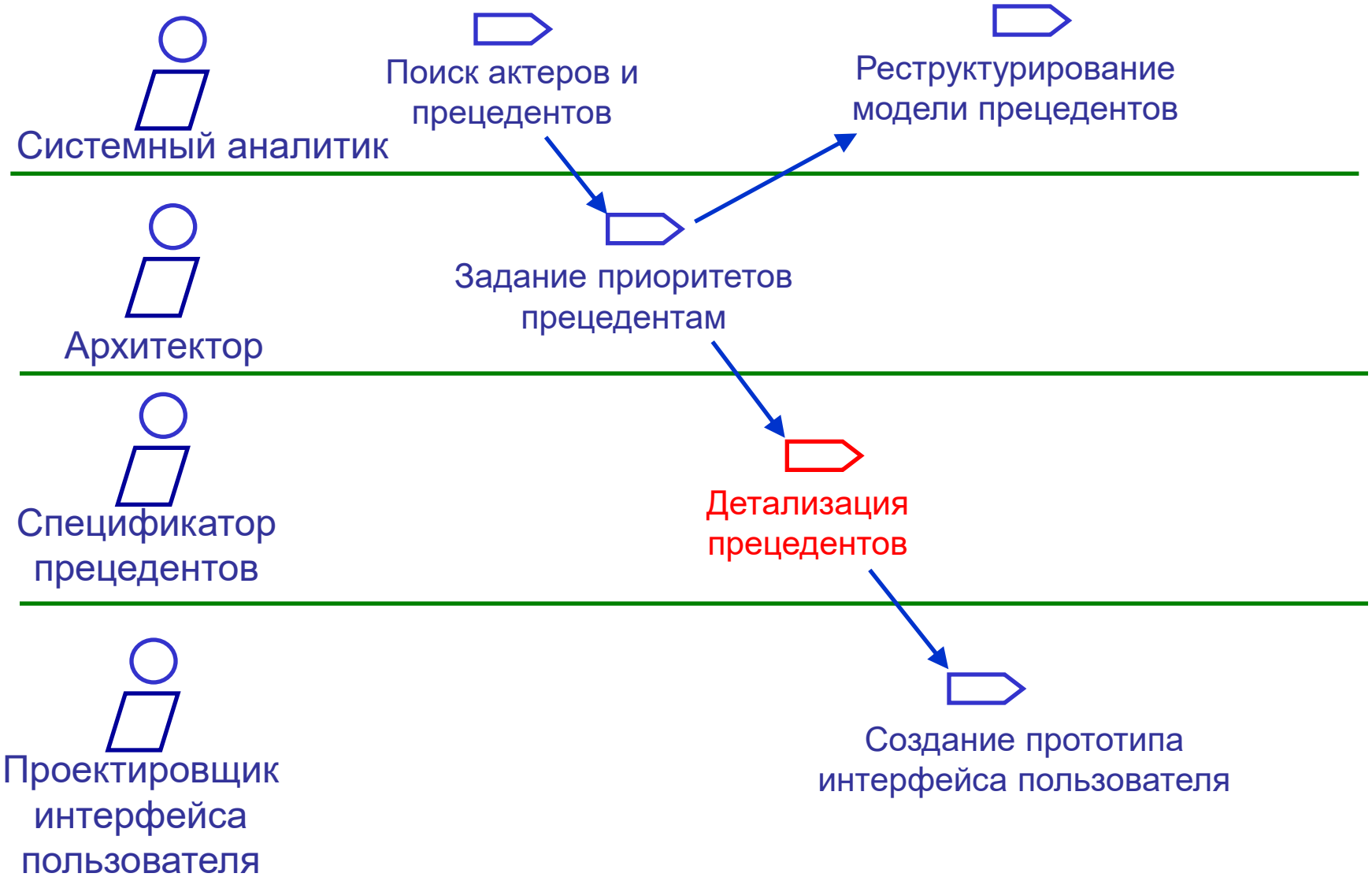
Деятельность по заданию приоритетов прецедентам (1)

1. На начальных фазах – определение прецедентов необходимых для раннего построения архитектуры.

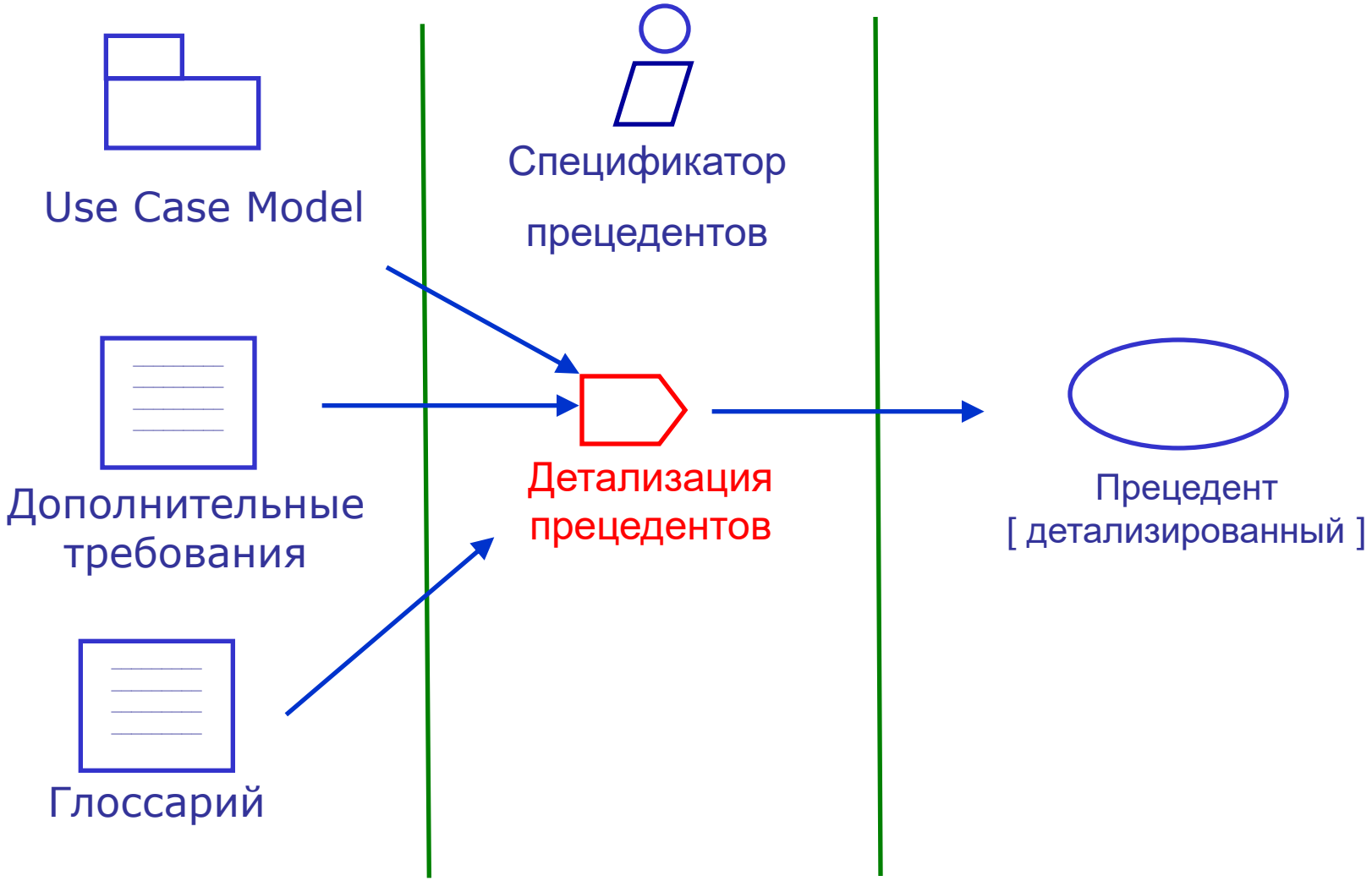
2. На поздних фазах - выбор прецедентов определяется очередностью и важностью прецедентов. Выбирают архитектор и менеджер проектов

3. Результат фиксируется в описании архитектуры и используется менеджером проекта для планирования итераций

Построение модели прецедентов
Поток работ (workflow)



Деятельность по заданию детализации прецедентов



Деятельность по детализации прецедентов

1. Пошагово описывается последовательность действий

2. Детализация описания прецедента

3. Формализация описания прецедента

Спецификатор прецедентов непосредственно работает с будущими пользователями прецедента.
Должна быть обратная связь от пользователя.

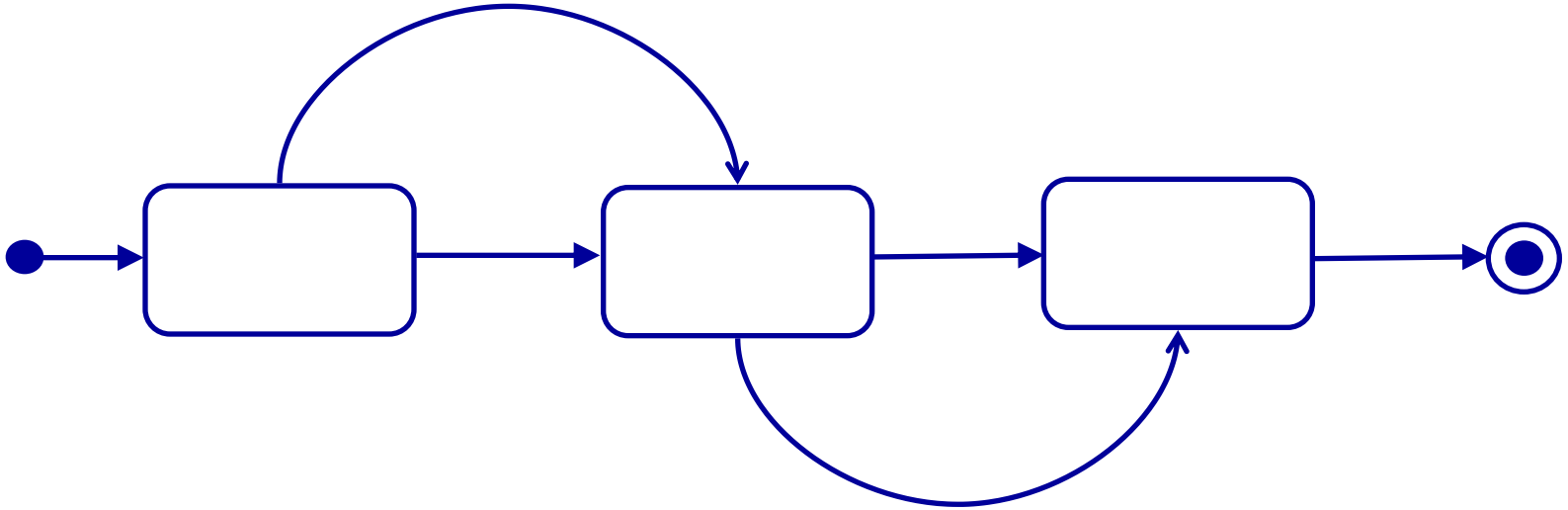
Деятельность по детализации прецедентов. Структурирование описание прецедента

Определяются состояния экземпляров прецедента и переходов между ними

1. Выбирается полный основной путь от начального состояния до конечного состояния

2. Описываются альтернативные пути
   - Отказы актера
   - Вмешательство других актеров
   - Ошибочный ввод от актеров
   - Недостаток ресурсов

Деятельность по детализации прецедентов Пример диаграммы



Переход – последовательность действий, которая выполняется экземпляром прецедента в случае некоторого события

Состояния – например, состояния документа

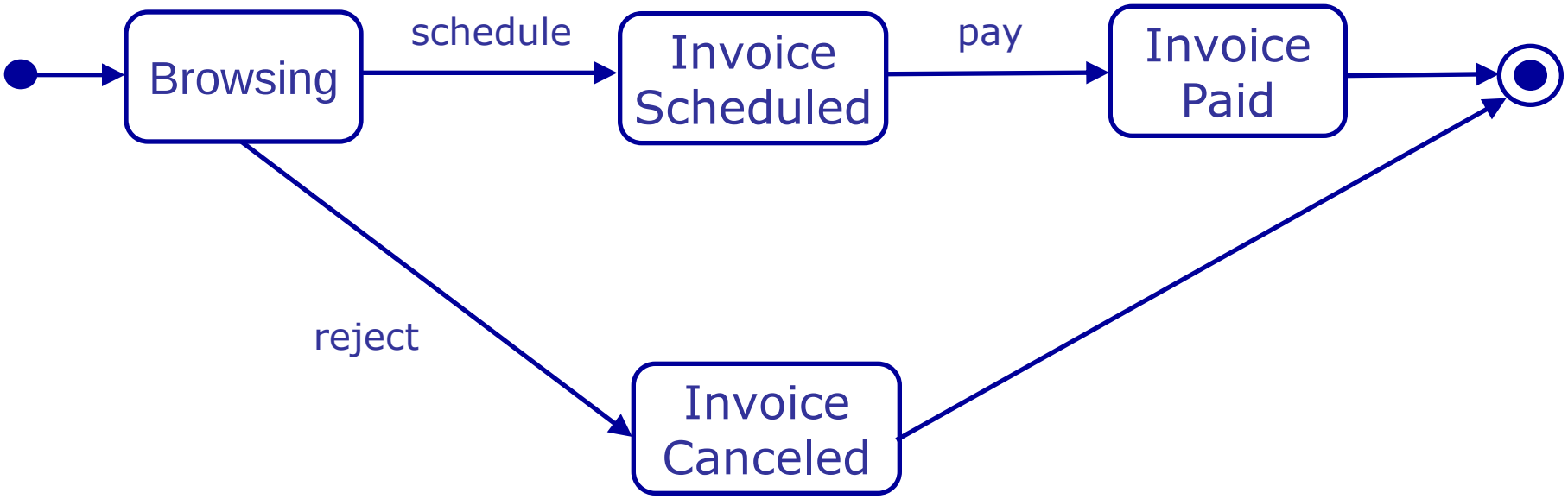
Деятельность по детализации прецедентов. Формализованное описание прецедента

1. State transition diagram (*диаграмма переходов и состояний*) – для описания состояния прецедента

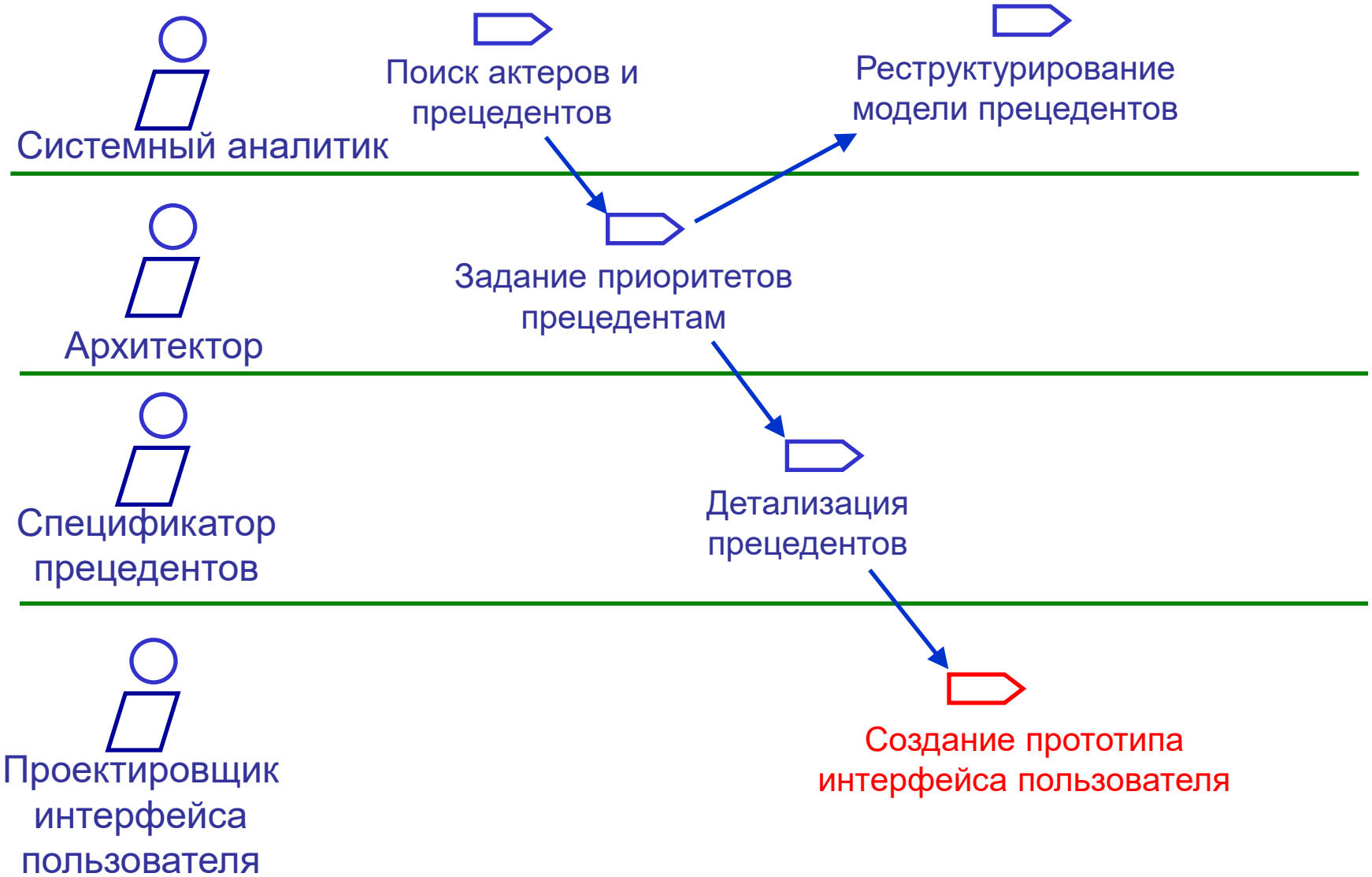
2. Activity diagram (*диаграмма деятельности*) – для детального описания переходов как последовательности действий

3. Collaboration diagram & Sequence diagram (*диаграмма кооперации объектов* и *диаграмма последовательности взаимодействия объектов*) – для описания взаимодействия экземпляра актера и экземпляра прецедента

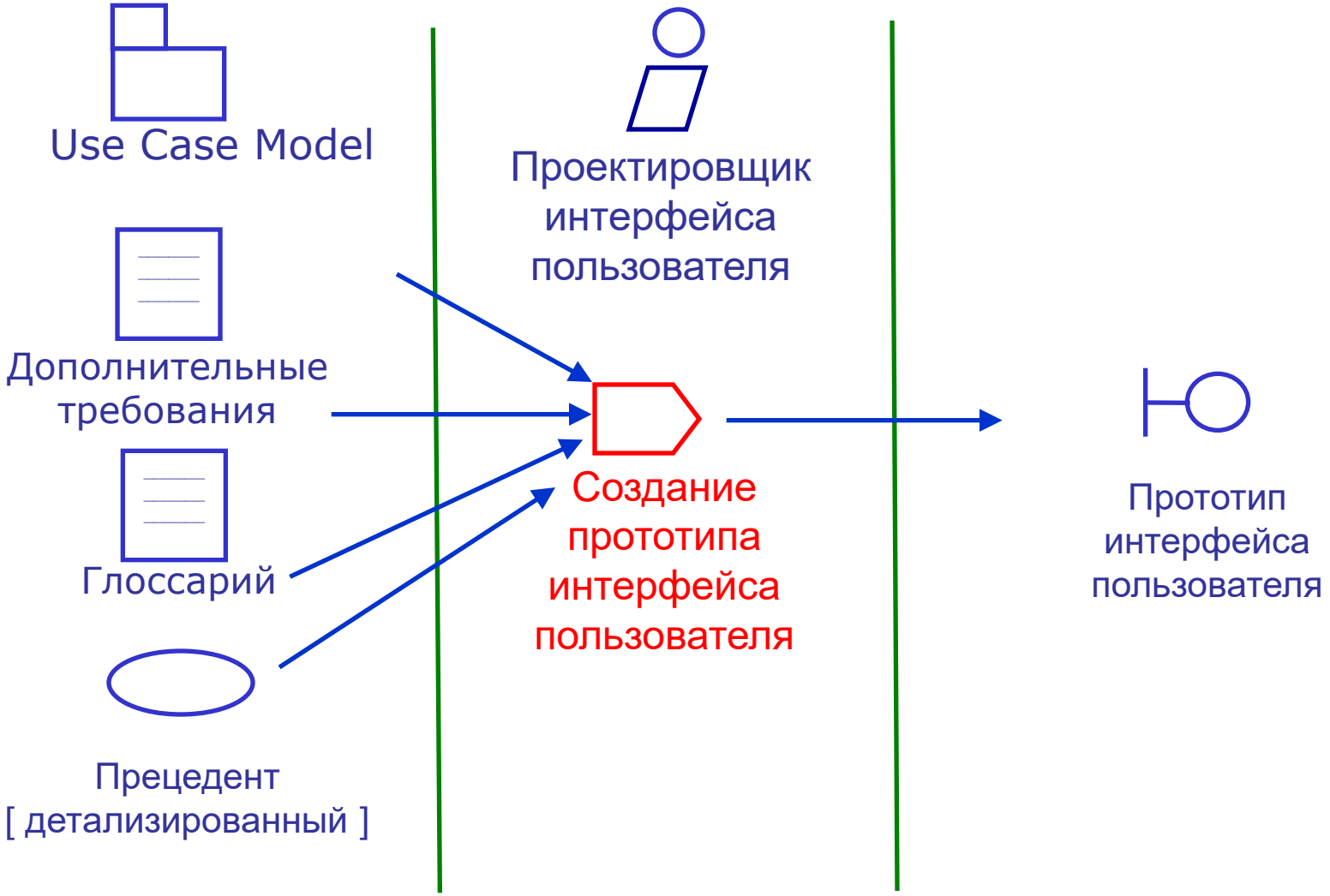
Деятельность по детализации прецедентов. Пример формализованного описания



Построение модели прецедентов
Поток работ (workflow)



Создание прототипа интерфейса пользователя



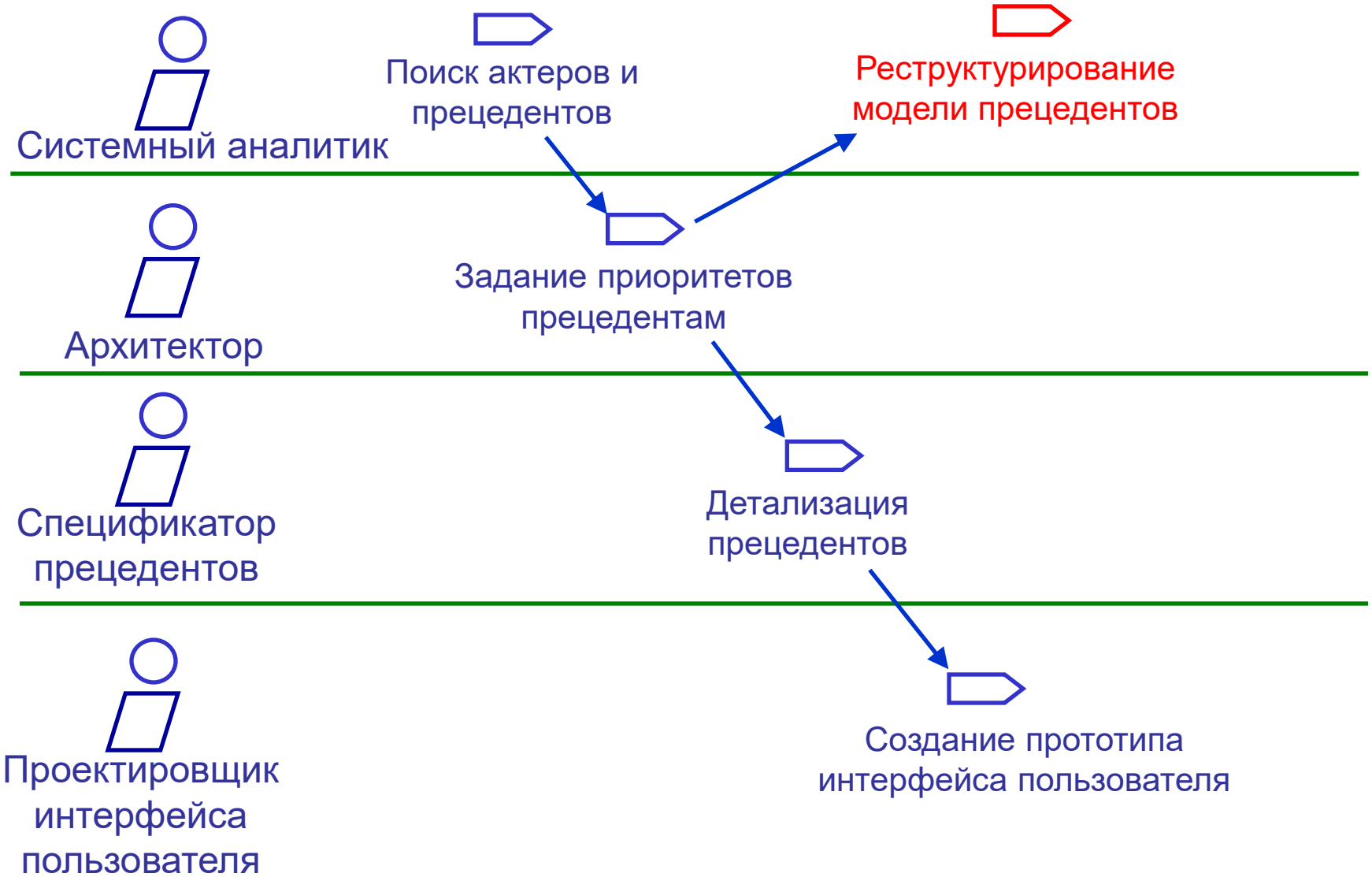
Создание прототипа интерфейса пользователя. Создание логического экрана

- Определяет что необходимо от интерфейса пользователя, что бы сделать доступным прецедент для актера.
- Элементы интерфейса пользователя – это атрибуты прецедента.
- Актер манипулирует атрибутами прецедента.
- Часто атрибуты – это термины из глоссария.
- Атрибуты представляются пиктограммами, списками, папками, объектами на карте

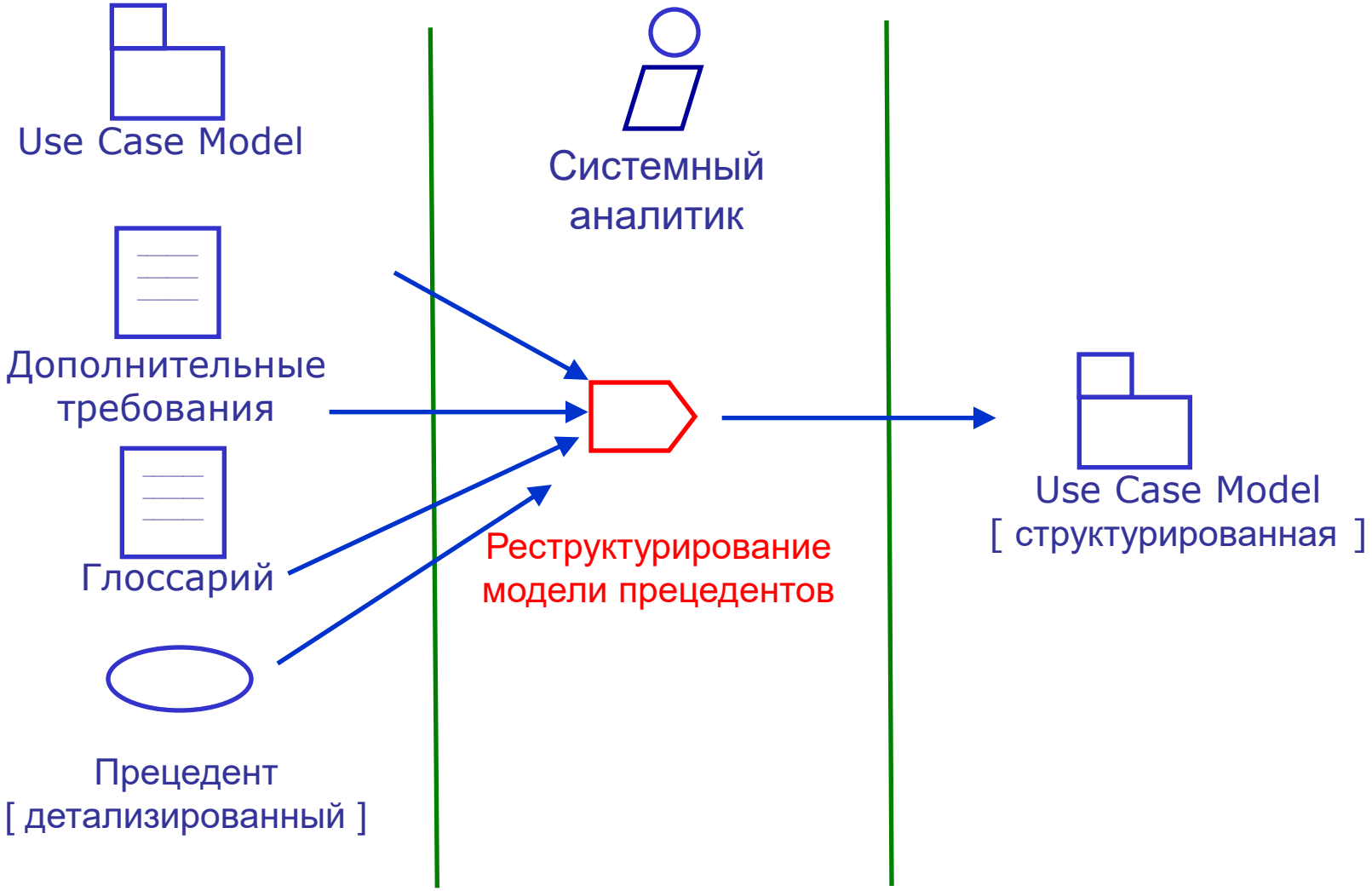
Создание прототипа интерфейса пользователя. Создание физического экрана

- Взаимное расположение управляющих элементов
- Для навигации и структурирования – папки, окна, инструменты (tools)
- Табуляция, ускорители
- Цвета, размеры

Построение модели прецедентов
Поток работ (workflow)



Реструктурирование модели прецедентов



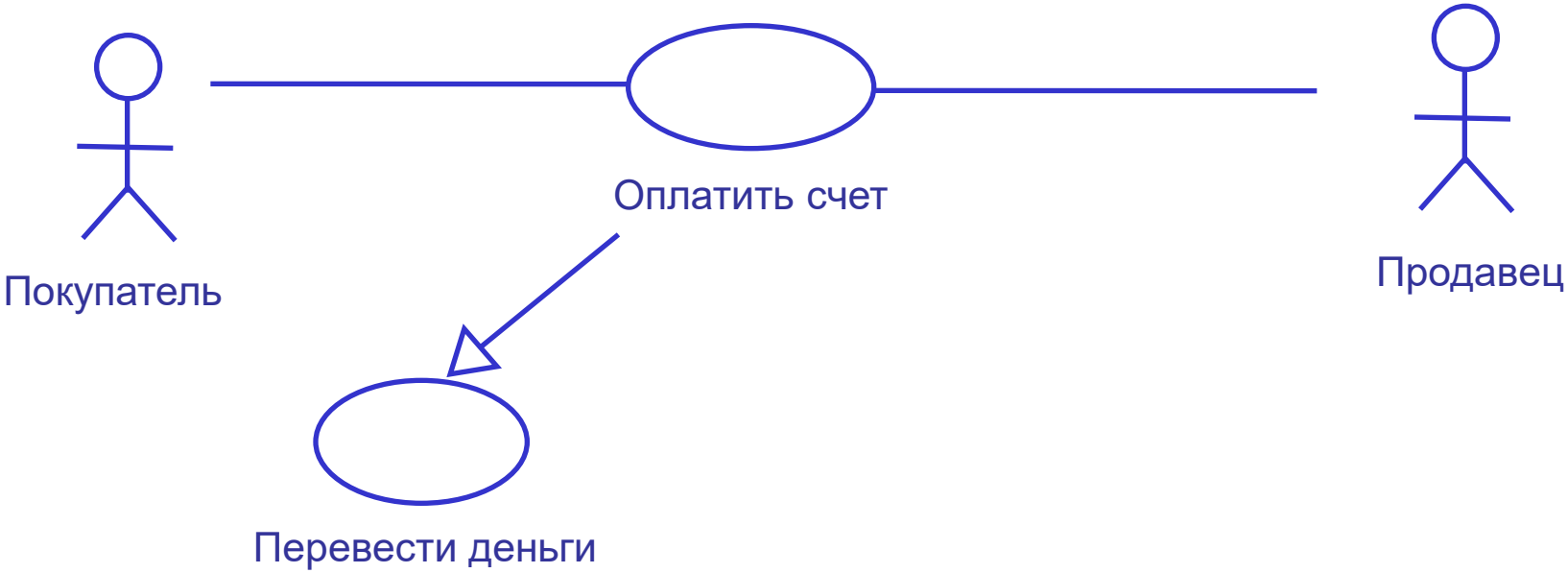
Реструктурирование модели прецедентов.
Разделяемые описания функциональности.

1. Выделить общие и совместно используемые прецеденты (описания функциональности которые могут быть использованы более специализированно)

2. Экземпляры в обобщенных прецедентах могут выполнить все поведение, описанное в обобщающем прецеденте

3. Экземпляр A будет включать поведение B

Реструктурирование модели прецедентов.
Разделяемые описания функциональности. Пример



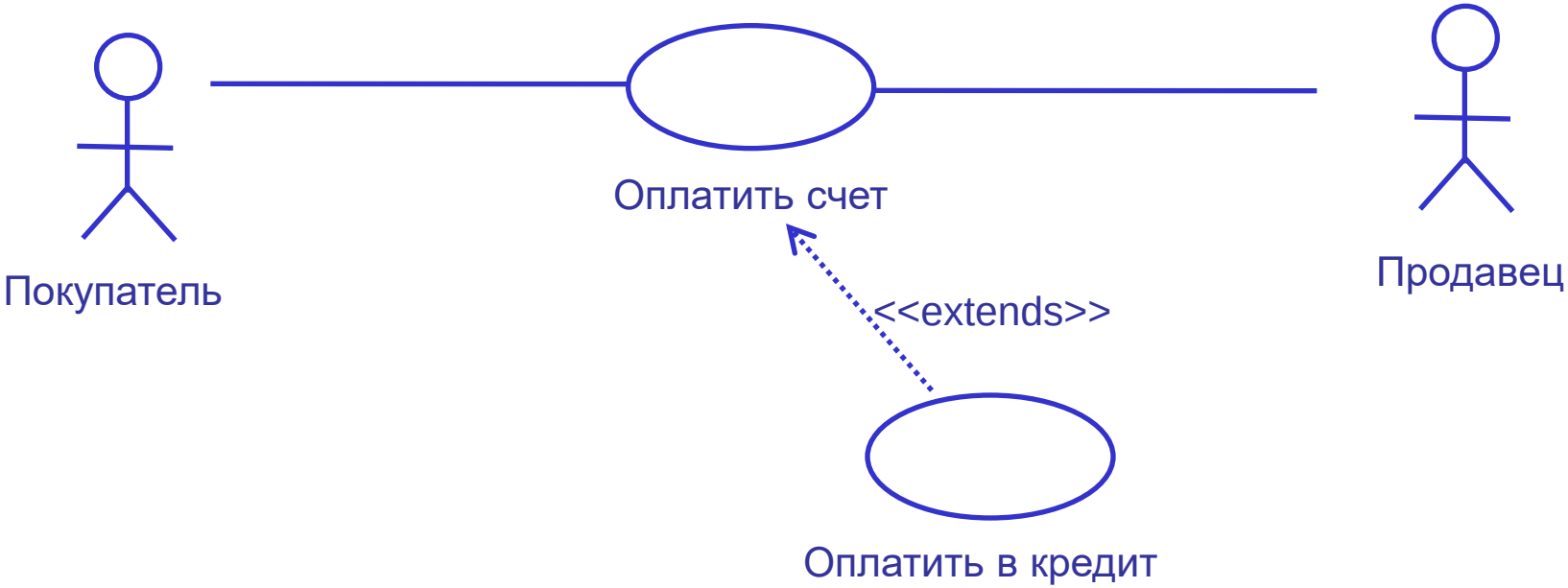
Реструктурирование модели прецедентов.
Дополнительные и условные описания функциональности

1. Выделить дополнительные и условные прецеденты которые могут расширить более специфичные прецеденты
2. Отношение зависимости со стереотипом <<extends>> моделируют дополнение к последовательности действий прецедента. Отношение включает условие расширения и точку расширения в целевом прецеденте.

3. Экземпляр A может *при некоторых условиях* включать поведение B

Реструктурирование модели прецедентов.

Дополнительные и условные описания функциональности



Поток работ по анализу системы.

Построение модели анализа

На этом этапе анализируются требования, полученные при построении модели прецедентов, путем их уточнения и структурирования.

Цели:

- Получить более полное понимание требований
- Получить описание требований, которые легко сопровождать
- Что поможет определить структуру системы в целом, включая ее архитектуру

Структура требований в построении модели анализа основана на классах и пакетах =>

Ортогональна структуре требований на основе модели прецедентов

Построение модели анализа
Сравнение моделей прецедентов и анализа

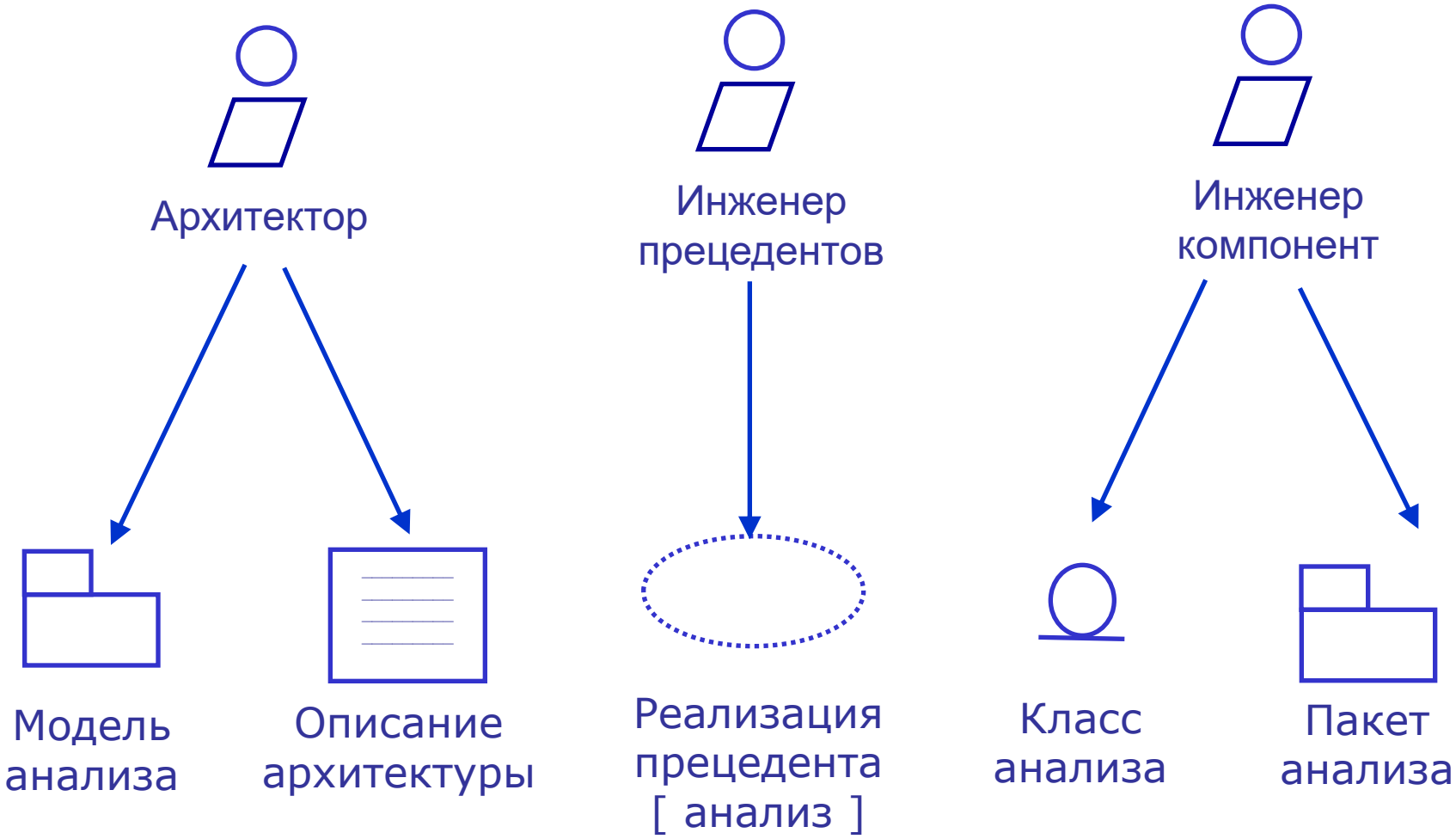
Use Case Model

- Язык заказчика
- Внешний вид системы
- Внешний вид структурируется с помощью прецедентов и актеров
- Используется как контракт заказчика и разработчика (что делать и не делать)
- Среди требований дублирование и несогласованность
- Получает функциональность системы (+архитектурно важную)

Analysis Model

- Язык разработчика
- Внутренний вид системы
- Внутренний вид структурируется с помощью стерео-классификаторов и пакетов
- Использует разработчик для понимания как система должна выглядеть
- Избыточности и несогласованности нет
- Черновик для реализации архитектурно важной функциональности

Построение модели анализа
Исполнители и артефакты



Артефакты в потоке работ по построению модели анализа (1)

Артефакт - Модель анализа

1. Пакеты анализа (*Analysis Packages*). Эти пакеты будут представлять уровни в *модели проектирования* (*Design model*).

2. Классы анализа (Analysis Classes). Эти классы будут представлять один и более классов в модели проектирования.

3. Реализации прецедентов (Use Case Realization). Эти реализации прецедентов будут описывать реализацию прецедентов как взаимодействие экземпляров классов анализа.

Артефакты в потоке работ по построению модели анализа (2)

Артефакт - Класс анализа

Абстракция одного и более классов и/или подсистем модели проектирования. Имеет особенности:

1. Класс анализа ориентирован на сбор функциональных требований. Не функциональные требования обрабатываются на этапе построения моделей проектирования и реализации.

2. Класс анализа редко определяет и предоставляет интерфейсы в терминах операций и их сигнатур.

   Вместо этого поведение определяется в терминах *ответственностей* (*responsibility*) на высоком уровне.

Артефакты в потоке работ по построению модели анализа (3)

Ответственность (*responsibility*) - текстовое описание связанное с подмножеством поведения определяемого классом анализа.

3. Класс анализа определяет атрибуты на высоком уровне. Типы атрибутов концептуальны и получаются из проблемной области. Это не типы языка программирования, как в модели проектирования и модели реализации. Возможно типы атрибутов станут классами в модели проектирования.

4. Класс анализа может включать отношения ассоциации и обобщения. Свойство *navigability* (*навигация*) отношения ассоциации не имеет такого значения в модели анализа, как имеет в модели проектирования

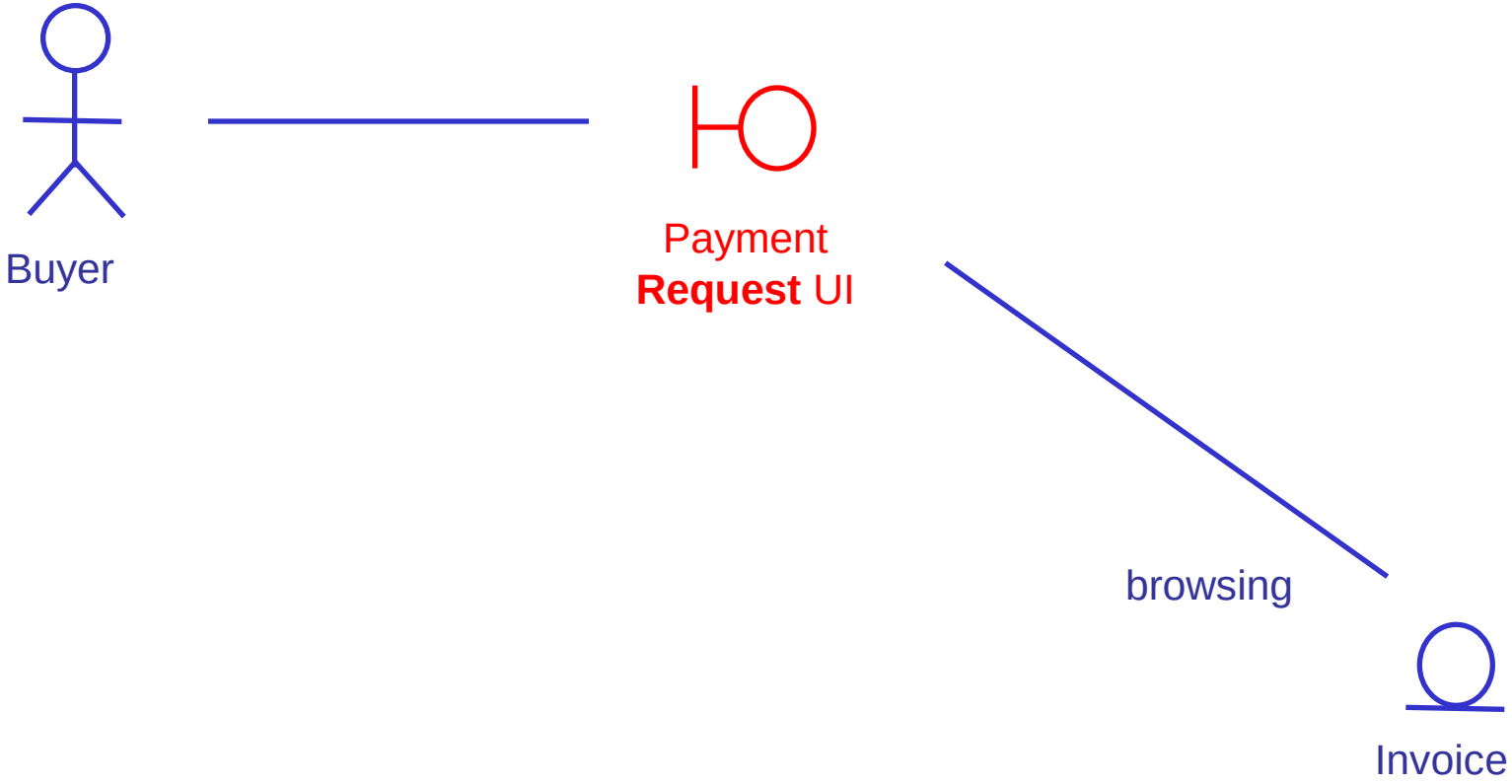
Артефакты в потоке работ по построению модели анализа (4)

1. *Граничный* (*boundary*) *класс* – используется для моделирования взаимодействия системы и актеров.

2. Граничные классы используются для сбора требований к системе на границе системы =>
   Изменение интерфейса пользователя изолируется 1..* граничных классов.

3. Граничные классы представляют: окна, формы, терминалы
4. Граничные классы должны быть высокого уровня => должны описывать взаимодействие с актером, а не его реализацию

Артефакты в потоке работ по построению модели анализа (5)



Артефакты в потоке работ по построению модели анализа (6)

1. *Класс-сущность* (*Entity*)– используется для моделирования долгоживущей и часто сохраняемой информации.

2. Класс-сущность используется для моделирования объектов и событий

3. Класс-сущность может быть взят из *Business model* или *Domain model*

4. Класс-сущность не обязательно пассивен
5. Класс-сущность:
   - показывает логическую структуру данных
   - показывает от какой информации зависит система

Артефакты в потоке работ по построению модели анализа (7)

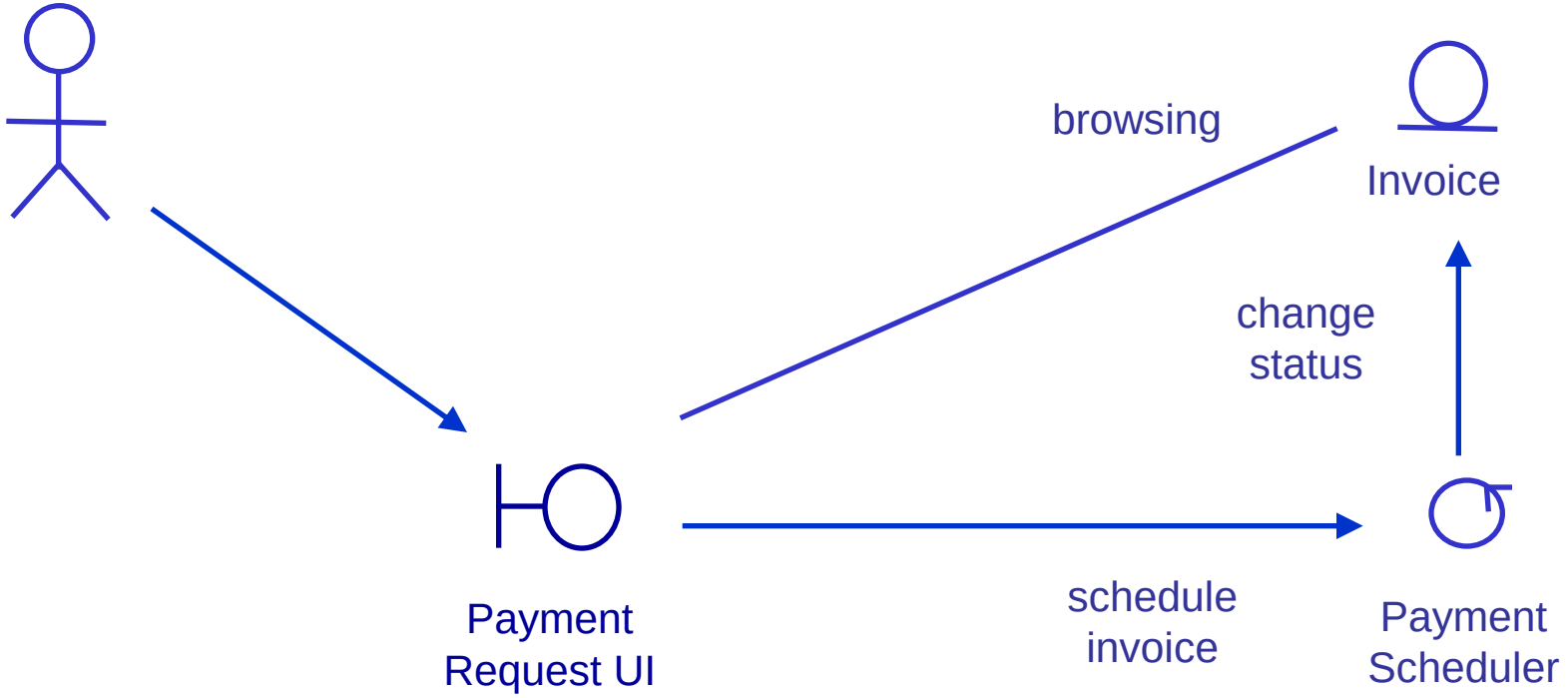
1. *Управляющий класс* (*Control*) – в нем сложная логика работы, но нет долгоживущей информации.

2. Управляющие классы не связаны с актерами

3. Управляющие классы делегирует работу в граничные классы и классы-сущности =>
имеют много связей с другими классами

4. Управляющие классы инкапсулируют
   - Изменения логики алгоритма
   - Изменения в координации классов

Артефакты в потоке работ по построению модели анализа (8)



Артефакты в потоке работ по построению модели анализа (9)

Артефакт - Реализация прецедента [анализ]

– описывает реализацию прецедента в терминах классов анализа и взаимодействия объектов анализа (экземпляров классов)

Реализация прецедента включает:

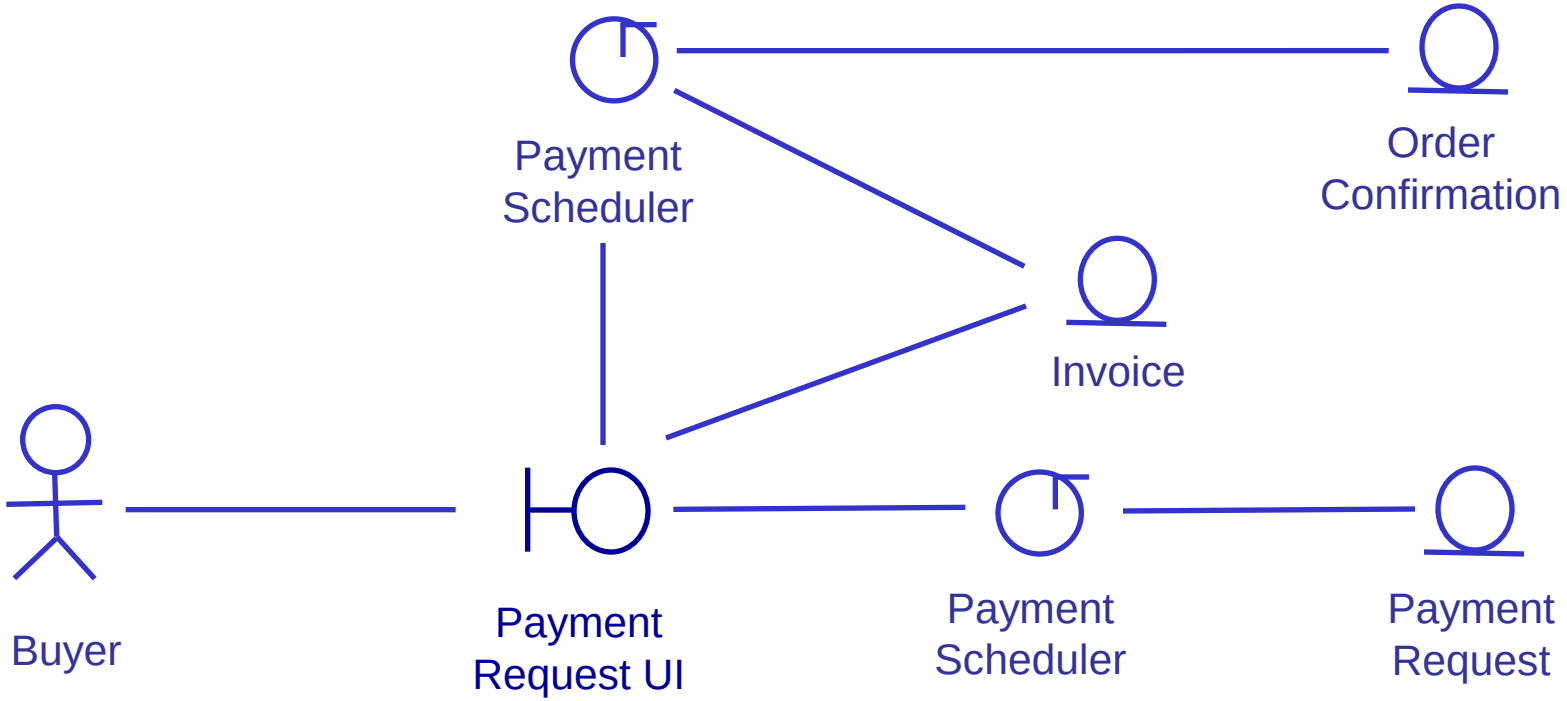
- текстовое описание потока событий
- диаграмму классов анализа, участвующих в реализации прецедента
- диаграммы взаимодействия объектов
- специальные требования

ориентированные на описание функциональных требований

Артефакты в потоке работ по построению модели анализа (9)

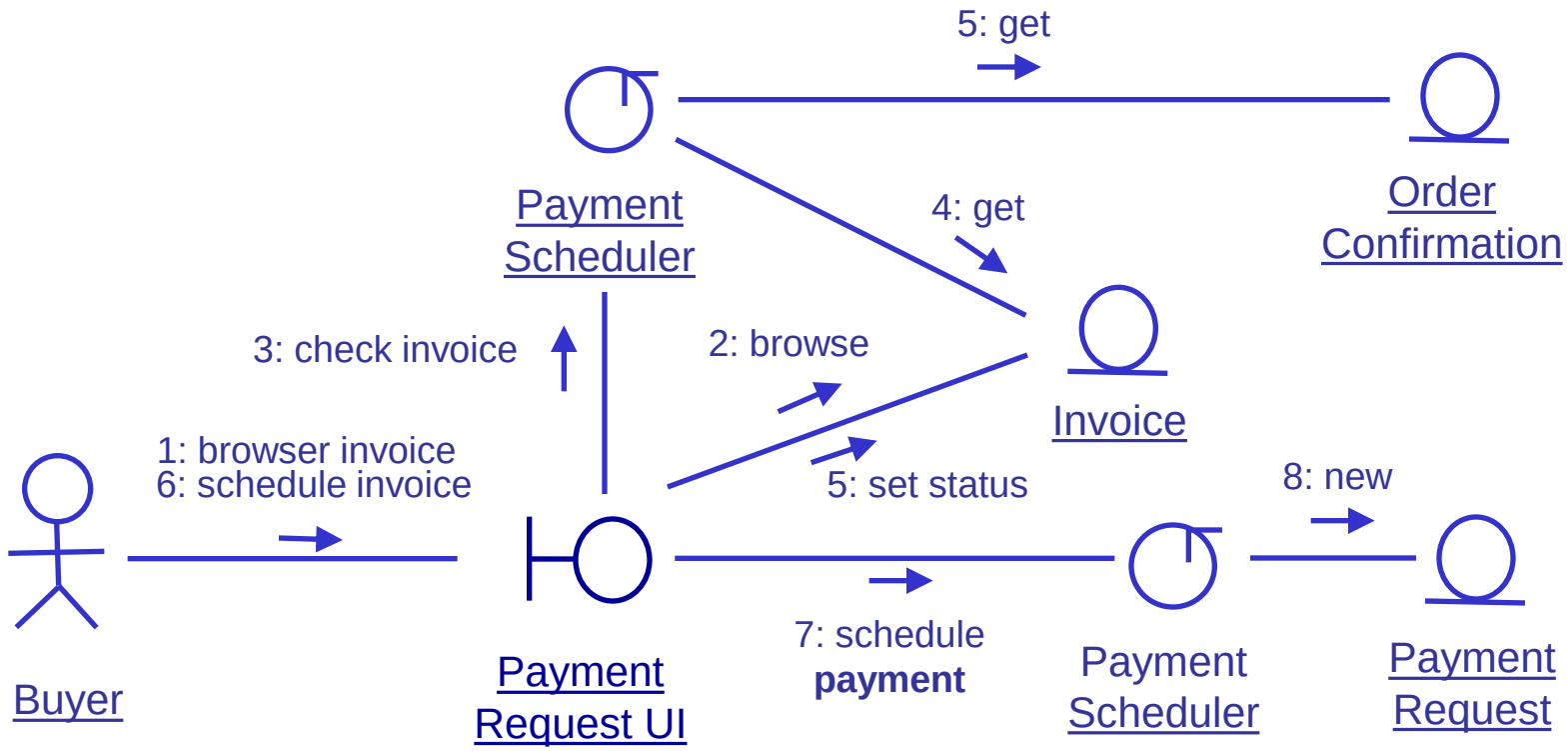
Диаграмма классов - – включает шаблон для рассылки сообщений:

- классы участвующие в реализации прецедента
- некоторые ответственности, атрибуты, ассоциации

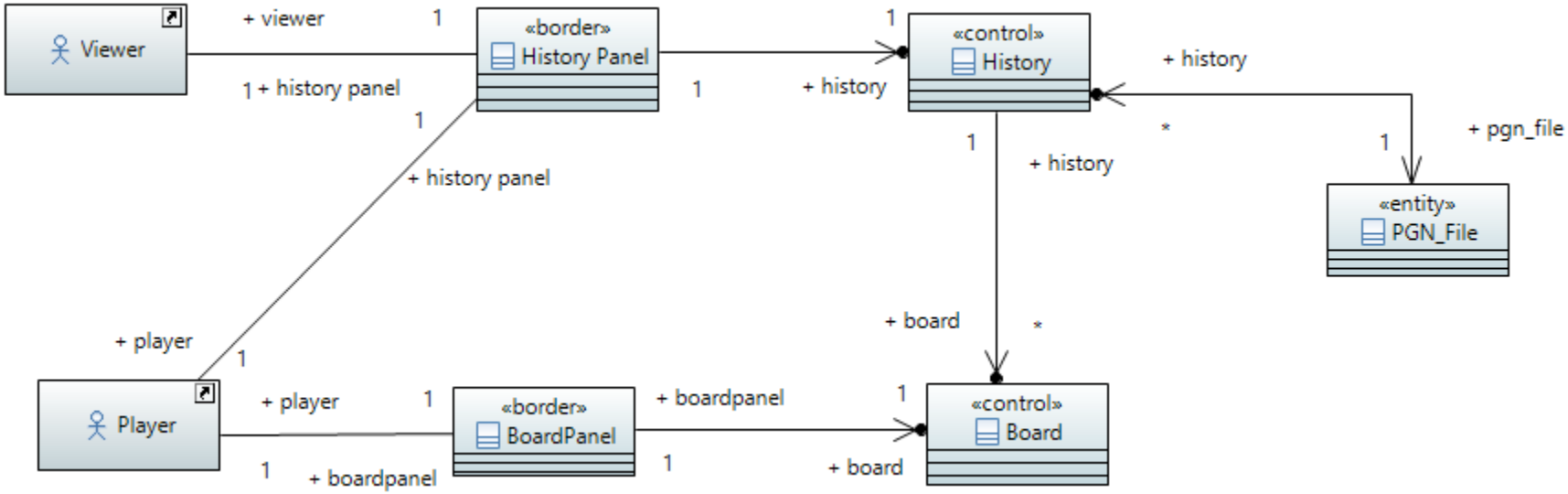


Артефакты в потоке работ по построению модели анализа (9)

Диаграмма взаимодействия



Артефакты в потоке работ по построению модели анализа (8)



Артефакты в потоке работ по построению модели анализа (9)

Поток событий

- Часто бывает необходим, в дополнение к диаграммам, упорядоченный текст облегчающий понимание сложных диаграмм
- Должен быть написан в терминах управляющих объектов, которые взаимодействуют друг с другом.
- Не должны упоминаться меняющиеся атрибуты, ассоциации, роли

Специальные требования

- Текстовое описание нефункциональных требований
- Пример:

Если покупатель хочет посмотреть счета, то они должны появиться не более чем через 0.5 сек.

Артефакты в потоке работ по построению модели анализа. Пакеты модели анализа

Артефакт - Пакеты модели анализа

1. Средство организации модели в управляемые фрагменты.
2. Содержит классы анализа, реализации прецедентов и другие пакеты
3. Зависимости между пакетами анализа должны быть минимизированы, а классы внутри пакета должны быть связаны
4. Свойства пакетов анализа:
   - Позволяют анализировать параллельно и раздельно различными разработчиками с опытом в разных предметных областях
   - Должны создаваться на основе функциональных требований и предметных областях
   - Пакеты станут 2-я верхними уровнями модели проектирования

Артефакты в потоке работ по построению модели анализа. Пакеты модели анализа

Сервисные пакеты

1. Система предоставляет заказчику множество сервисов
2. Сервис предоставляет неделимое множество функциональности (все или ничего)
3. Прецеденты для пользователей, сервисы для заказчика
4. Сервисный пакет – множество функционально связанных классов
5. Неделим. Заказчик получает все классы или ничего.
6. Используется для реализации многих прецедентов
7. Имеет ограниченную зависимость от других сервисных пакетов
8. С одним или небольшим количеством актеров
9. Позволяет структурировать модели проектирования и реализации на бинарные и исполняемые файлы

Артефакты в потоке работ по построению модели анализа. Описание архитектуры

Артефакт – Описание архитектуры

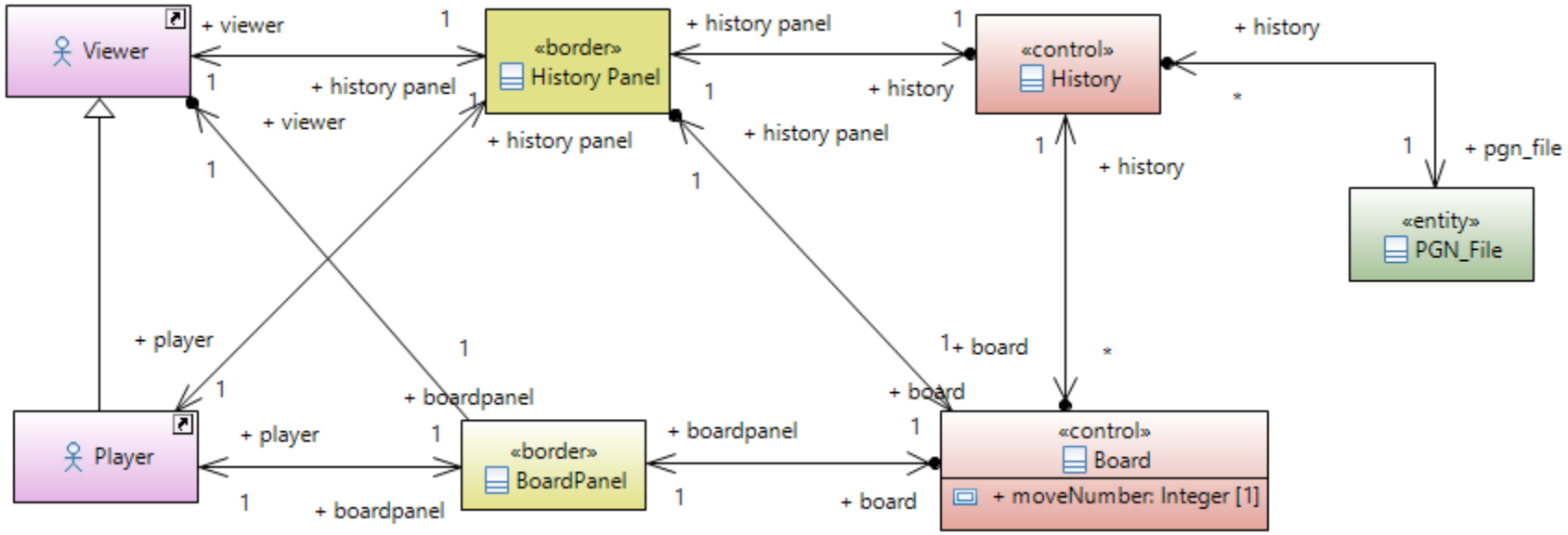
1. Разбиение модели анализа на пакеты анализа – важная информация для создания архитектуры. Часто это подсистемы верхнего уровня в модели проектирования

2. Включает классы:
   - Классы-сущности описывающие предметную область
   - Граничные классы – наиболее важные пользовательские интерфейсы
   - Граничные классы – реализующие общие механизмы интерфейса пользователя
   - Управляющие классы – координирующие взаимодействие классов в важных прецедентах
   - Классы анализа: общие, центральные, имеющие много отношений с другими классами, абстрактные классы

Артефакты в потоке работ по построению модели анализа. Описание архитектуры

3. Реализации прецедентов:
   - Реализующие важную и критичную функциональность
   - Реализацию прецедентов, используемых многими пакетами анализа (utility)
   - Которые включают много классов анализа
   - Которые должны быть реализованы как можно раньше, поскольку их реализация используется реализациями других прецедентов

Построение модели анализа
Классы в модели анализа для шахмат



Артефакты в потоке работ по построению модели анализа. Классы

Portable **G**ame **N**otation (**PGN**) – текстовый формат для хранения информации о шахматной партии (об игроках и сделанных в партии ходах)

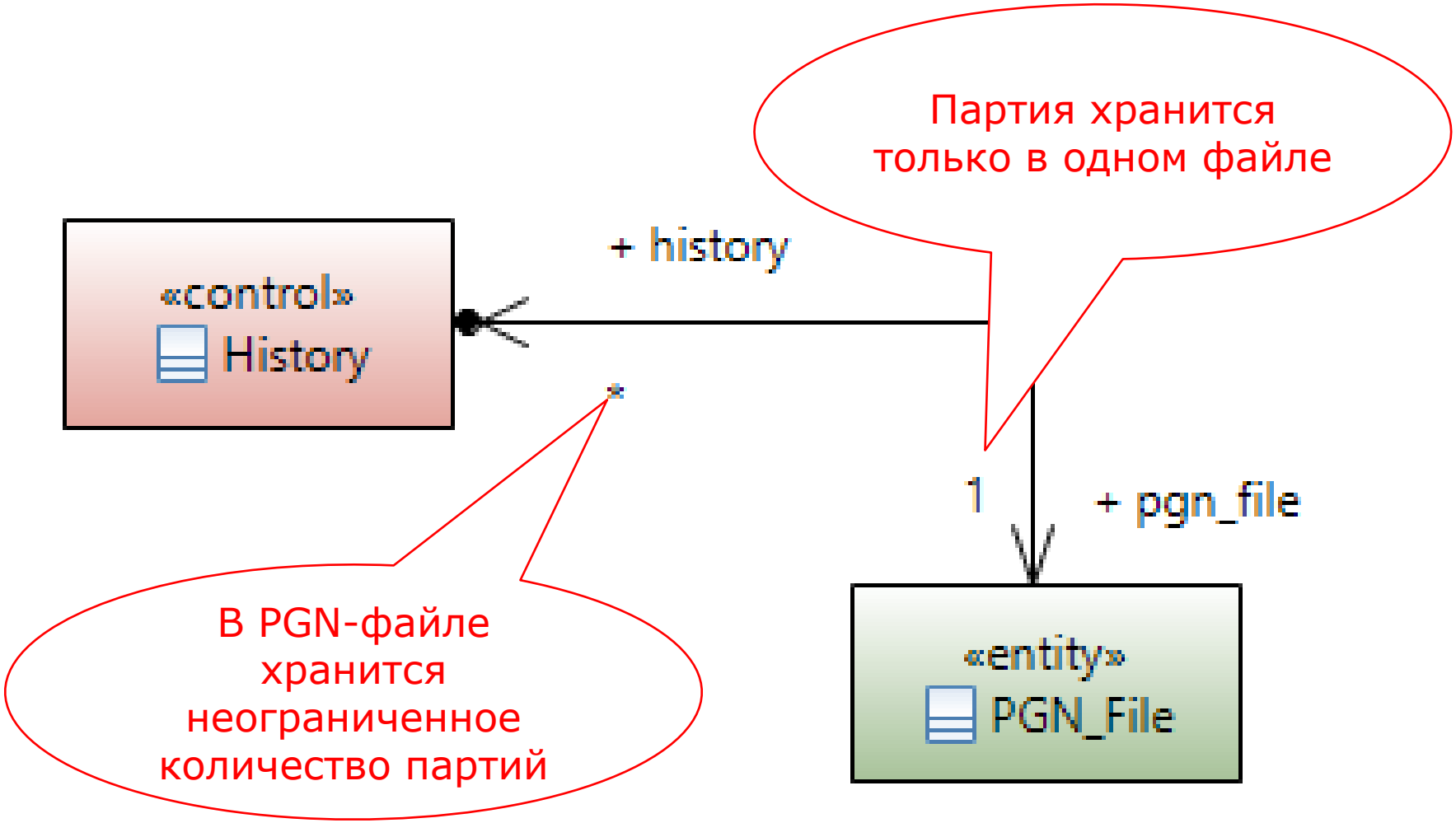
PGN_File - файл с множеством сохраненных партий игр

History – история ходов игры (партия)

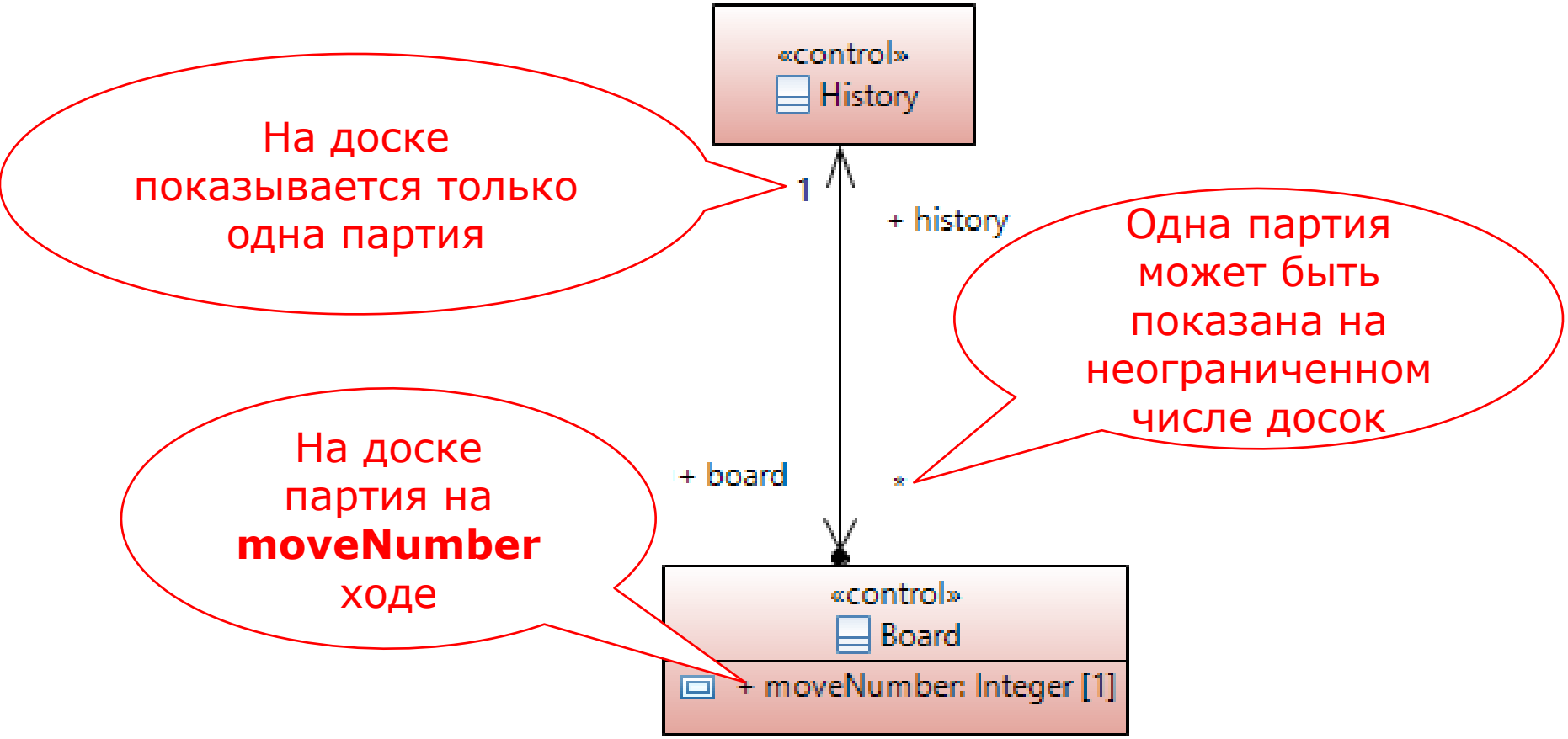
Viewer – обозреватель. Пользователь наблюдающий за игрой и имеющий возможность просмотреть историю игры.

Player – игрок. Пользователь имеющий возможность передвигать фигуры на доске.

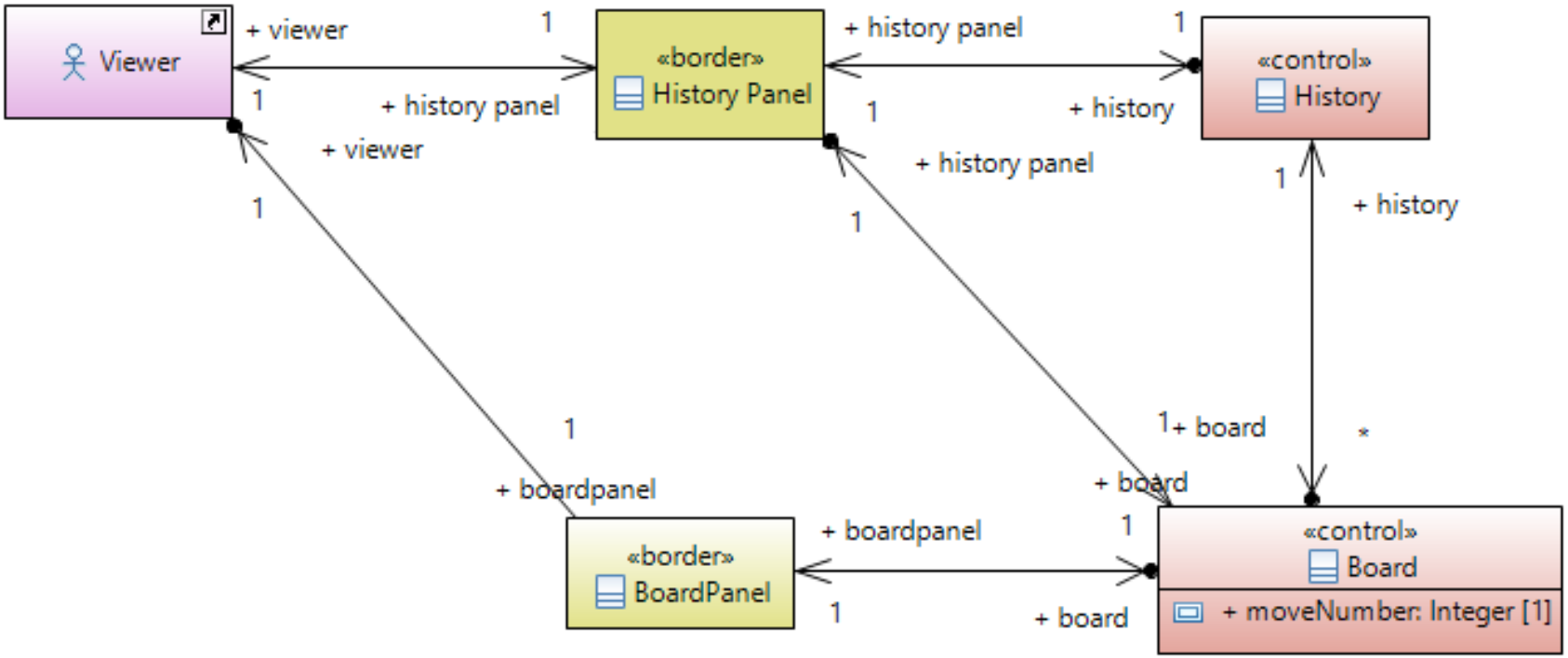
Построение модели анализа. Отношение классов *History* и *PGN_File*



Построение модели анализа. Отношение классов *History* и *Board*



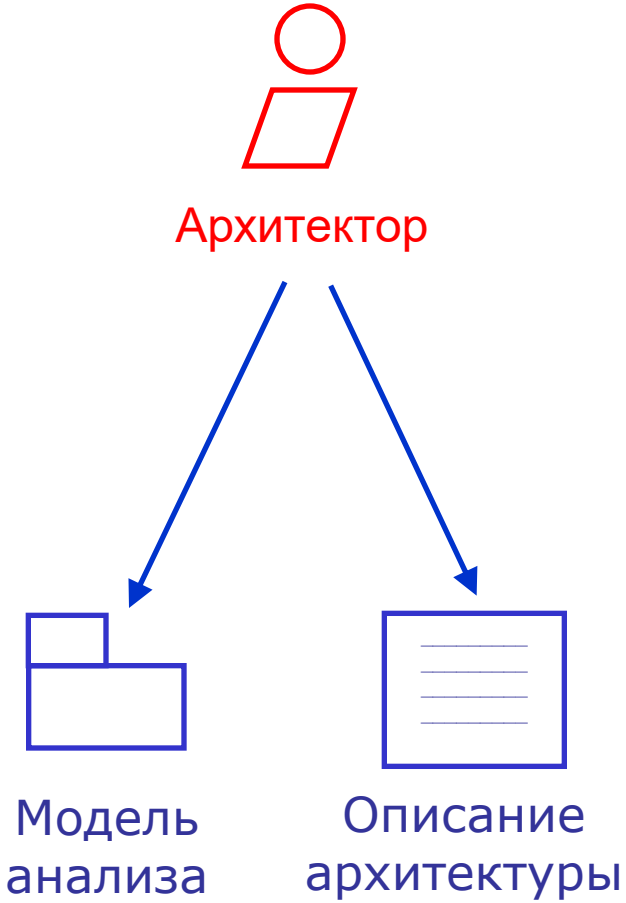
Построение модели анализа. Отношение класса Viewer



Обозреватель может смотреть и воздействовать (выбирая номер хода) на истории партии.

Обозреватель может смотреть на панель доски с выбранным ходом.

Построение модели анализа. Исполнители Архитектор

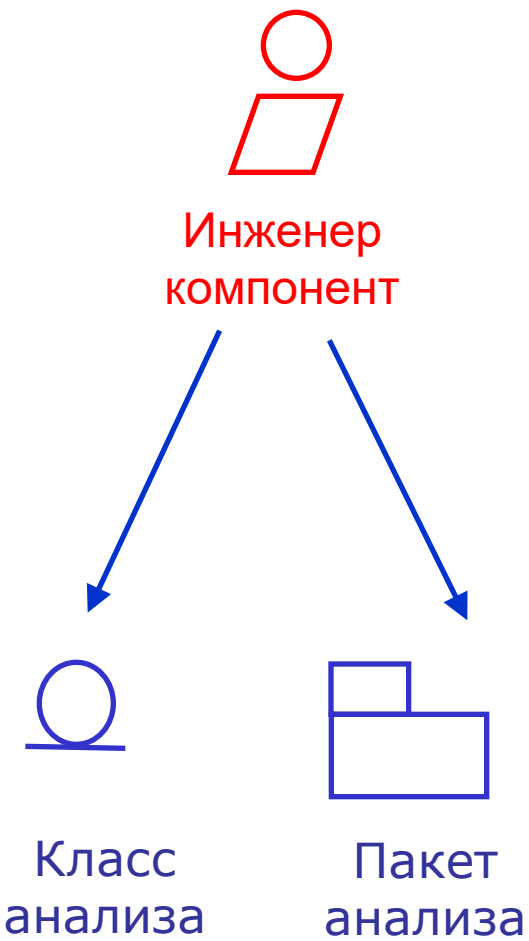


1. Отвечает за правильность, согласованность и читаемость модели анализа
2. Полностью отвечает за описание архитектуры
3. Другие сотрудники отвечают за согласование с архитектурой пакетов верхнего уровня
4. Модель анализа верна, если реализует функциональность только из прецедентов
5. Архитектор не отвечает за сопровождение артефактов

Построение модели анализа. Исполнители. Инженер прецедентов

1. Отвечает за целостность одной и более реализаций прецедентов
2. Прецедент полностью реализует функциональность и только ее
3. Обеспечивает читаемость всех диаграмм и текстов описывающих реализацию
4. Инженер прецедентов НЕ отвечает за классы модели анализа и их отношения. За это отвечают только инженеры компонент

Построение модели анализа. Исполнители. Инженер компонентов

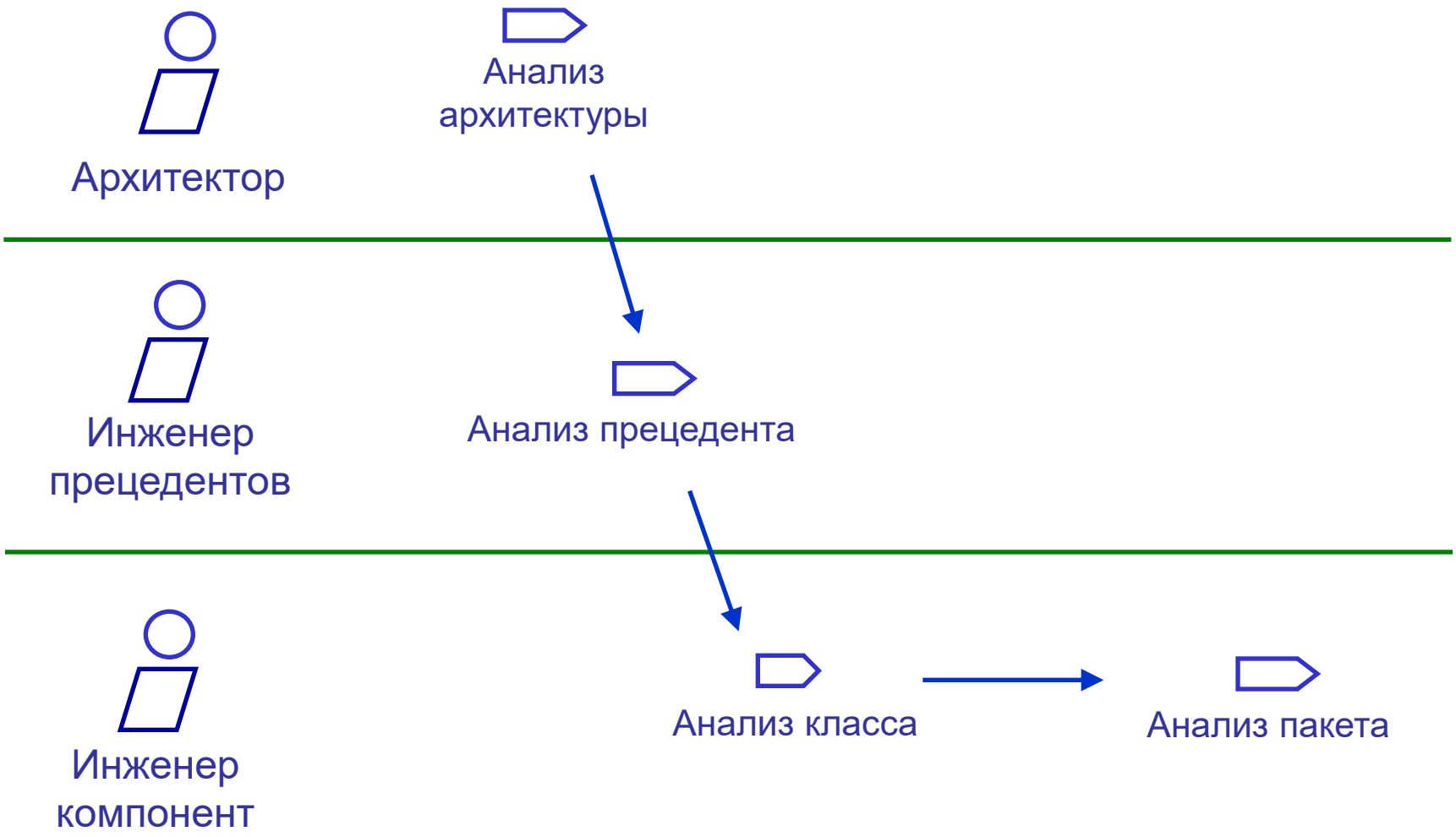


1. Для одного и более классов определяет и поддерживает ответственности, атрибуты, отношения, специальные требования

2. Поддерживает целостность одного и более пакетов.
   - Классы пакета и их отношения корректны.
   - Отношения с классами из других пакетов корректны и минимальны

3. Инженер компонент отвечает за пакеты и подсистемы моделей проектирования и реализации, соответствующие пакетам анализа

Построение модели анализа
Поток работ (workflow)



Построение модели анализа
Обзор потока работ (workflow)

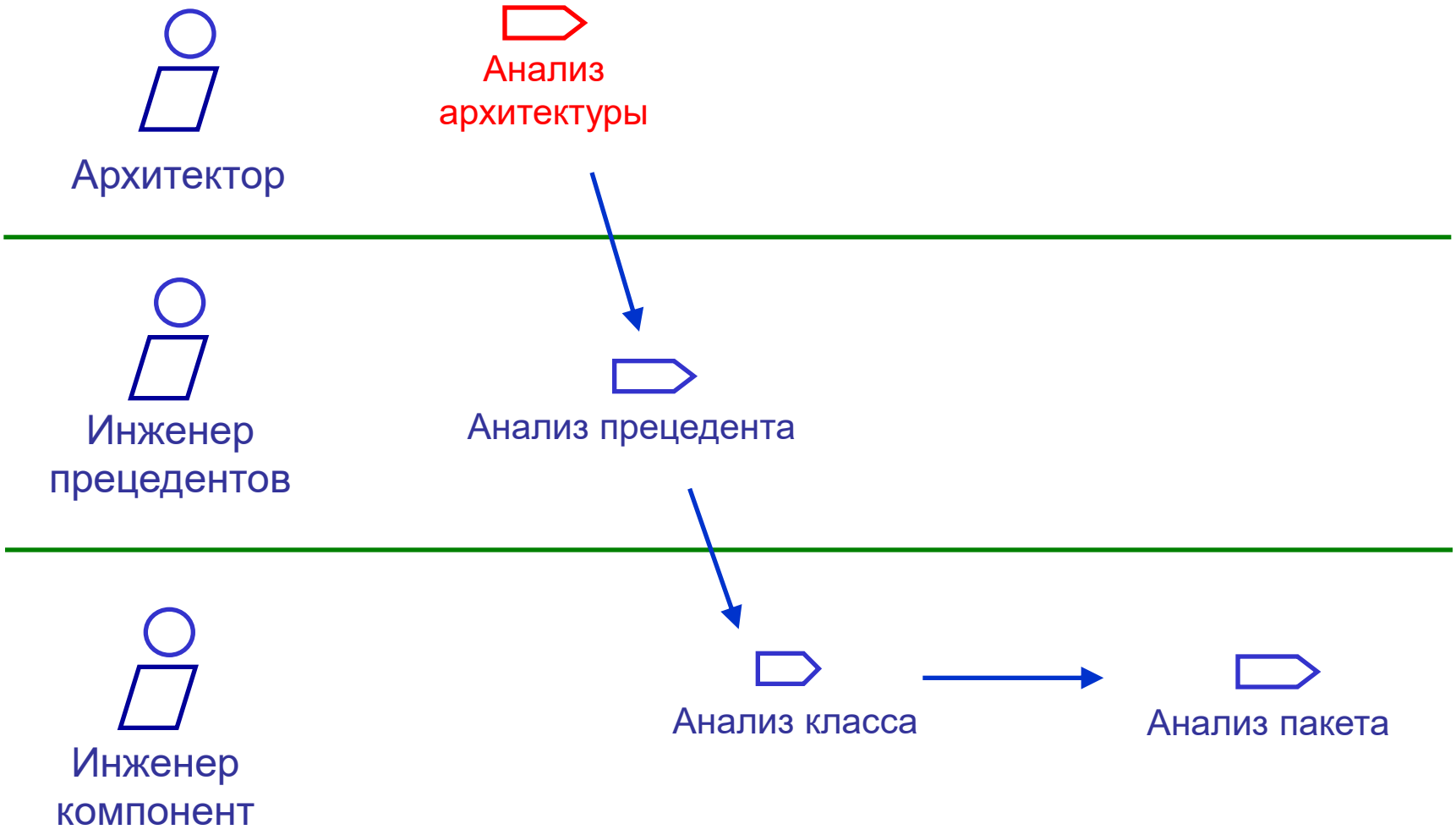
1. Поток работ инициируется архитектором, определяющим:
   - Главные пакеты анализа
   - Очевидные классы сущности
   - Общие требования

2. Инженер прецедентов реализует каждый прецедент используя классы анализа, задавая требования к поведению каждого класса

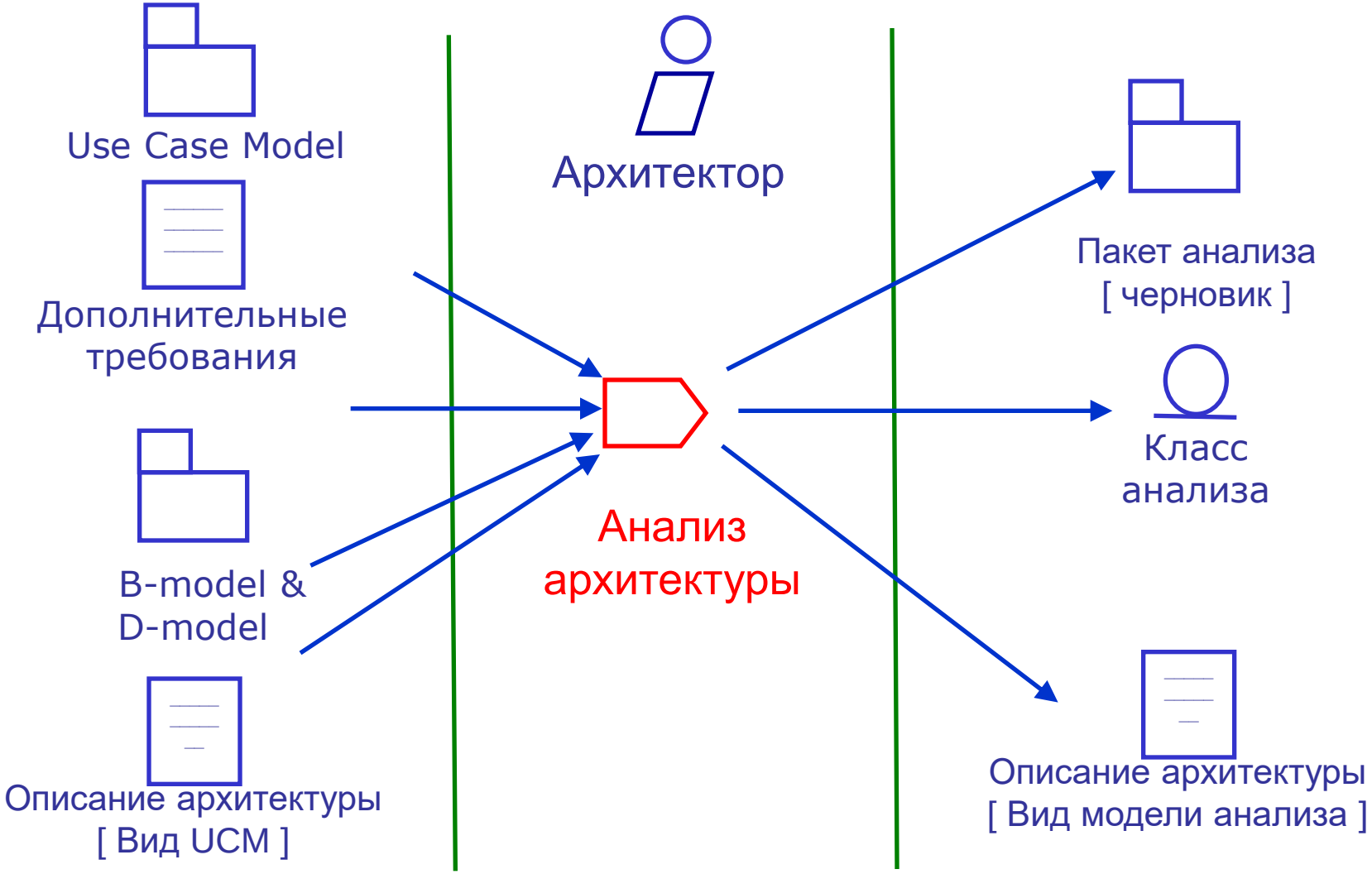
3. Затем эти требования к поведению классов инженер компонент интегрирует в классы, создавая:
   - Согласованные ответственности класса
   - Атрибуты классов
   - Отношения классов

4. Ответственный за пакеты инженер компонентов согласовывает содержимое пакета и его отношения

Построение модели анализа
Поток работ (workflow)



Анализ архитектуры



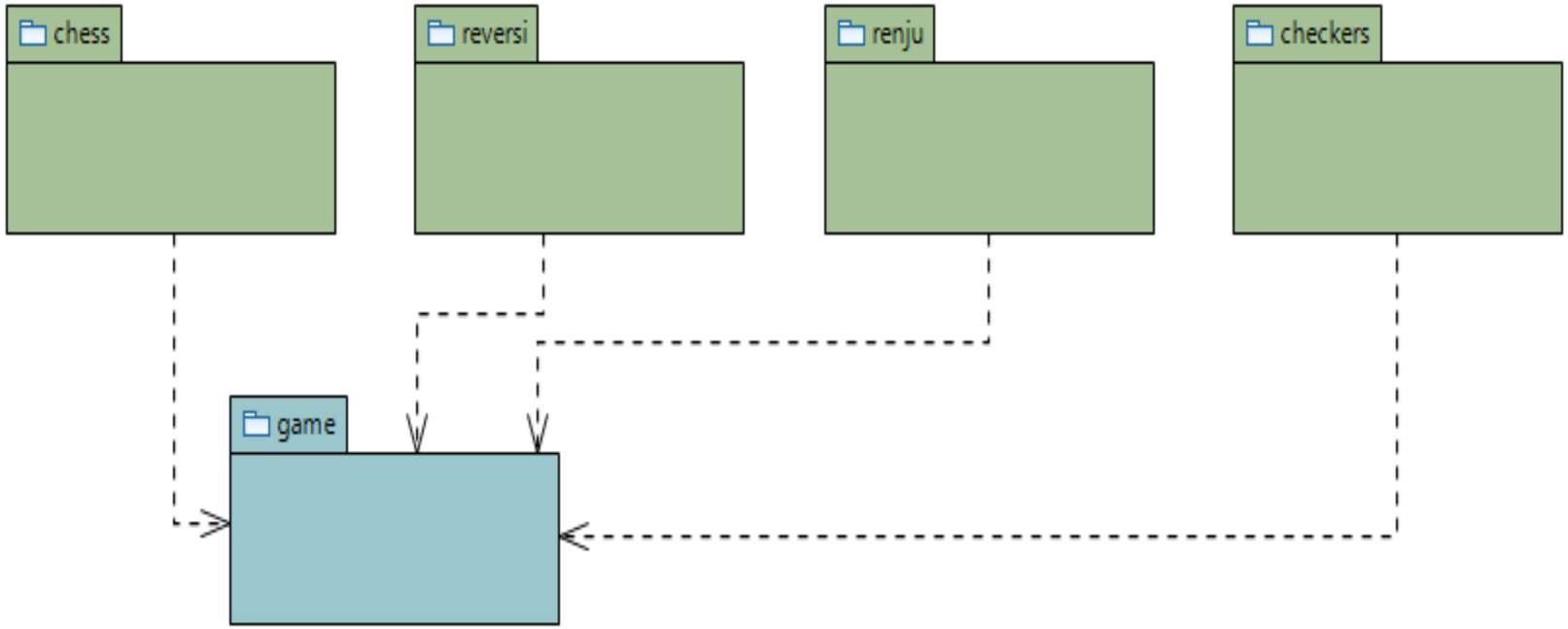
Анализ архитектуры
Идентификация пакетов модели анализа

1. Пакеты анализа разделяют модель на более легко управляемые части. Пакеты организуются:
   - По организации работ
   - Путем декомпозиции большой структуры

2. Первичная идентификация пакетов делается на основе функциональных требований и модели предметной области (domain model)
   - В отдельный пакет прецеденты реализующие конкретный бизнес процесс
   - В отдельный пакет прецеденты поддерживающие конкретного актера
   - В один пакет прецеденты связанные отношения наследования и расширения

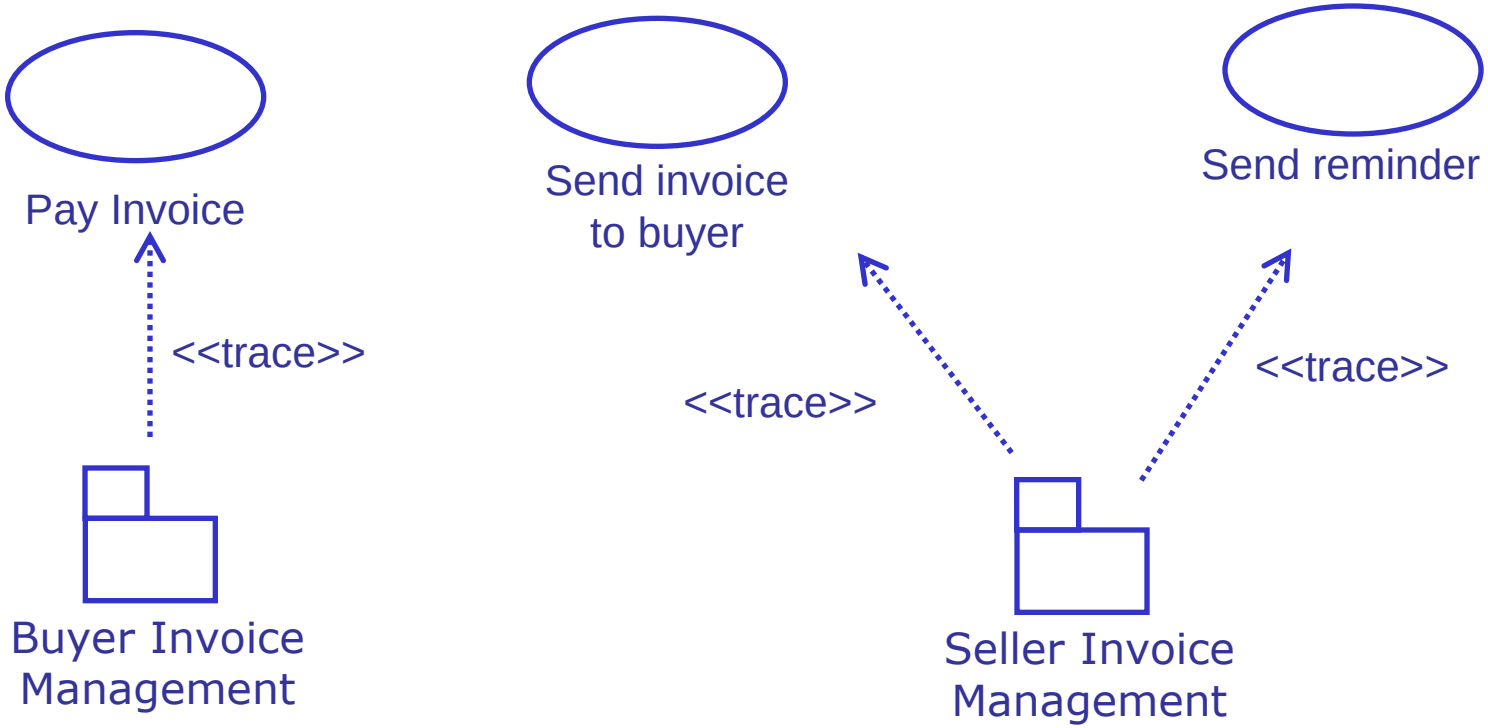
Такие пакеты локализуют изменения в бизнес процессе

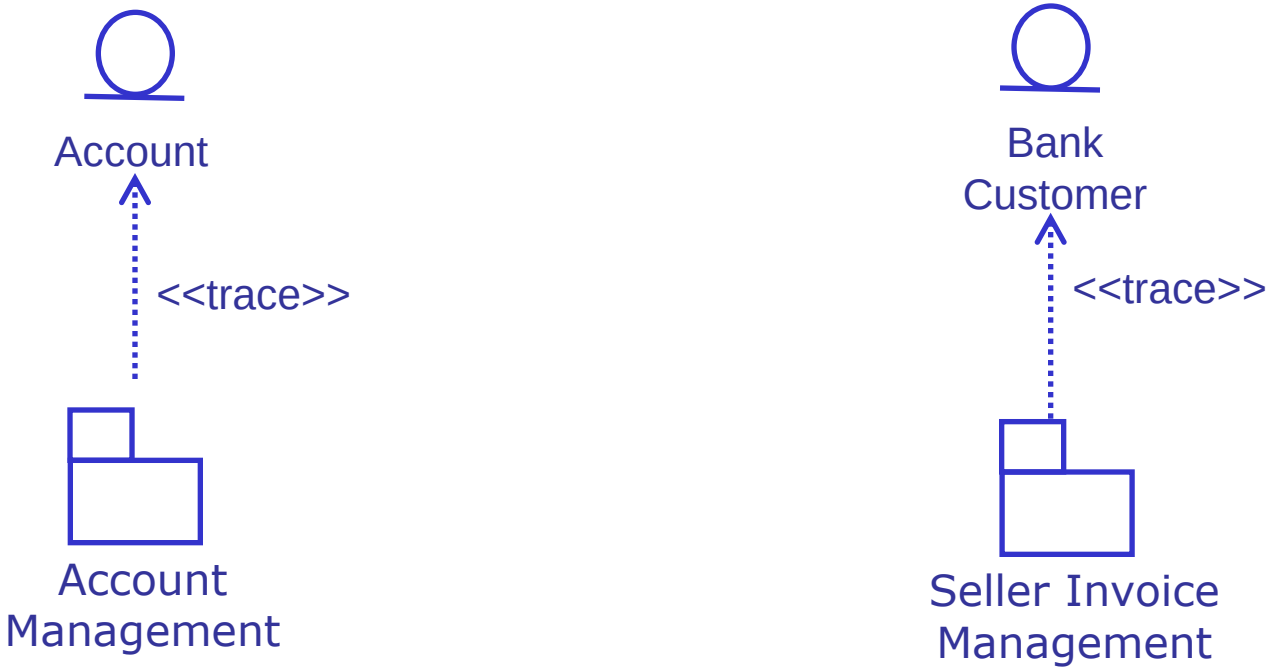
Анализ архитектуры
Идентификация пакетов в модели анализа



Анализ архитектуры
Идентификация пакетов модели анализа

3. При реализации прецедентов как диаграмм поведения с помощью классов происходит уточнение структуры пакетов



Анализ архитектуры
Идентификация пакетов модели анализа

4. Обработка общности среди пакетов анализа. Если пакеты разделяют общий класс, то этот класс должен быть вынесен в отдельный пакет или быть вне разделяющих его пакетов.