ПРОГРАММНАЯ ИНЖЕНЕРИЯ

РЕАЛИЗАЦИЯ ПРЕЦЕДЕНТОВ

АНАЛИЗ ПРЕЦЕДЕНТА

- Аналитическая модель классов – это статическая структура системы, а **реализация прецедентов** показывает, как взаимодействуют экземпляры классов анализа для осуществления функциональности системы.
- Цели разработчика:
  - Выяснить, взаимодействие каких классов анализа обеспечивает поведение прецедента;
  - Выяснить, какими сообщениями и в какой последовательности должны обмениваться экземпляры данных классов

РЕАЛИЗАЦИЯ ПРЕЦЕДЕНТОВ

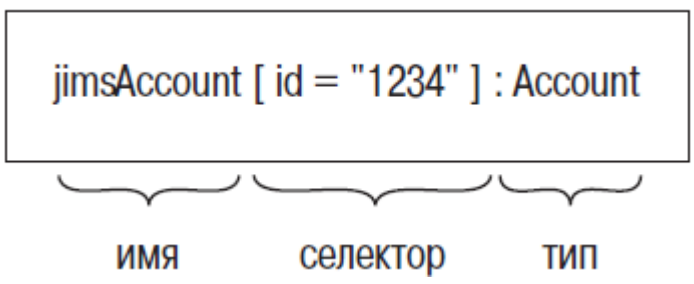
- Реализации прецедентов показывают, как взаимодействуют классы, чтобы реализовать функциональность системы.

Элемент	Назначение
Диаграммы классов анализа	Показывают классы анализа, взаимодействующие для реализации прецедента.
Диаграммы взаимодействий	Показывают взаимодействия определенных экземпляров, реализующих прецедент; это «моментальные снимки» работающей системы.
Особые требования	Процесс реализации прецедентов может выявить новые характерные для прецедента требования; они должны быть зафиксированы.
Уточнение прецедентов	Во время реализации может быть обнаружена новая информация, т. е. происходит обновление исходного прецедента.

ДИАГРАММЫ ПОСЛЕДОВАТЕЛЬНОСТЕЙ

ЛИНИЯ ЖИЗНИ

- **Линия жизни (lifeline)** представляет одного участника взаимодействия, т. е. она представляет, как экземпляр классификатора (объекта, класса, актера и др.) участвует во взаимодействии.



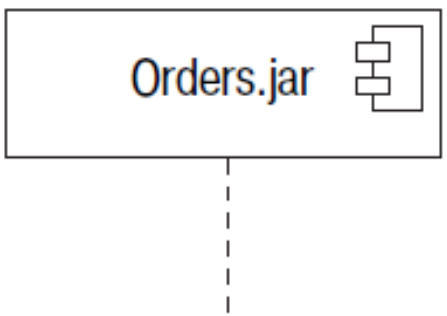
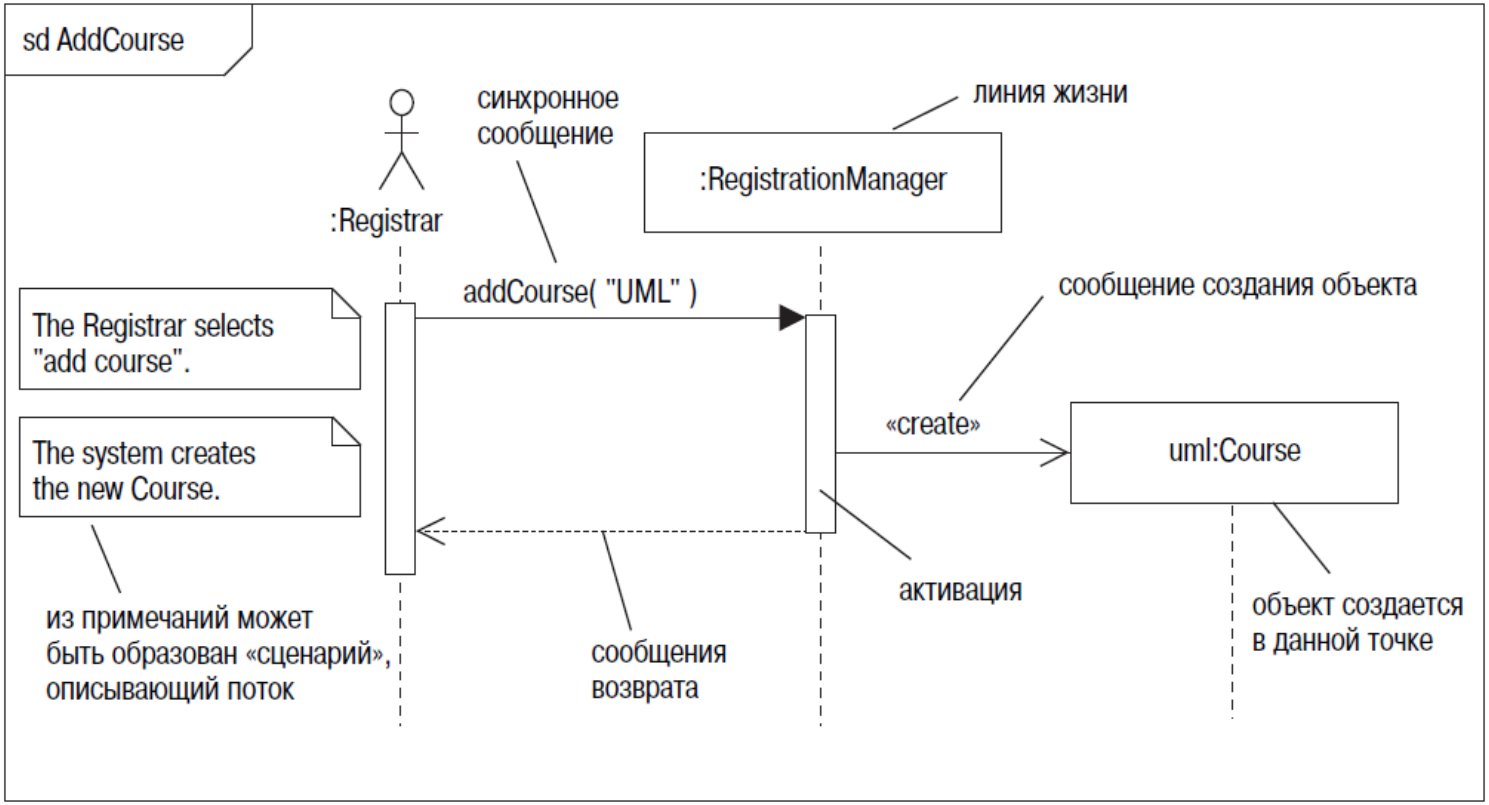


ДИАГРАММА ПОСЛЕДОВАТЕЛЬНОСТИ

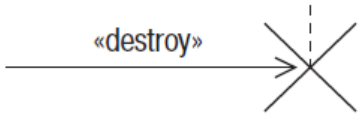
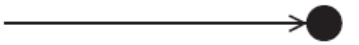
- Диаграммы последовательностей представляют взаимодействия между линиями жизни как упорядоченную последовательность событий.



СООБЩЕНИЯ

- Сообщение – это особый вид коммуникации между линиями жизни:
  - вызов операции – сообщение вызова;
  - создание или уничтожение экземпляра – сообщение создания или уничтожения;
  - отправку сигнала.
- Сообщения отображаются в виде стрелок между линиями жизни.

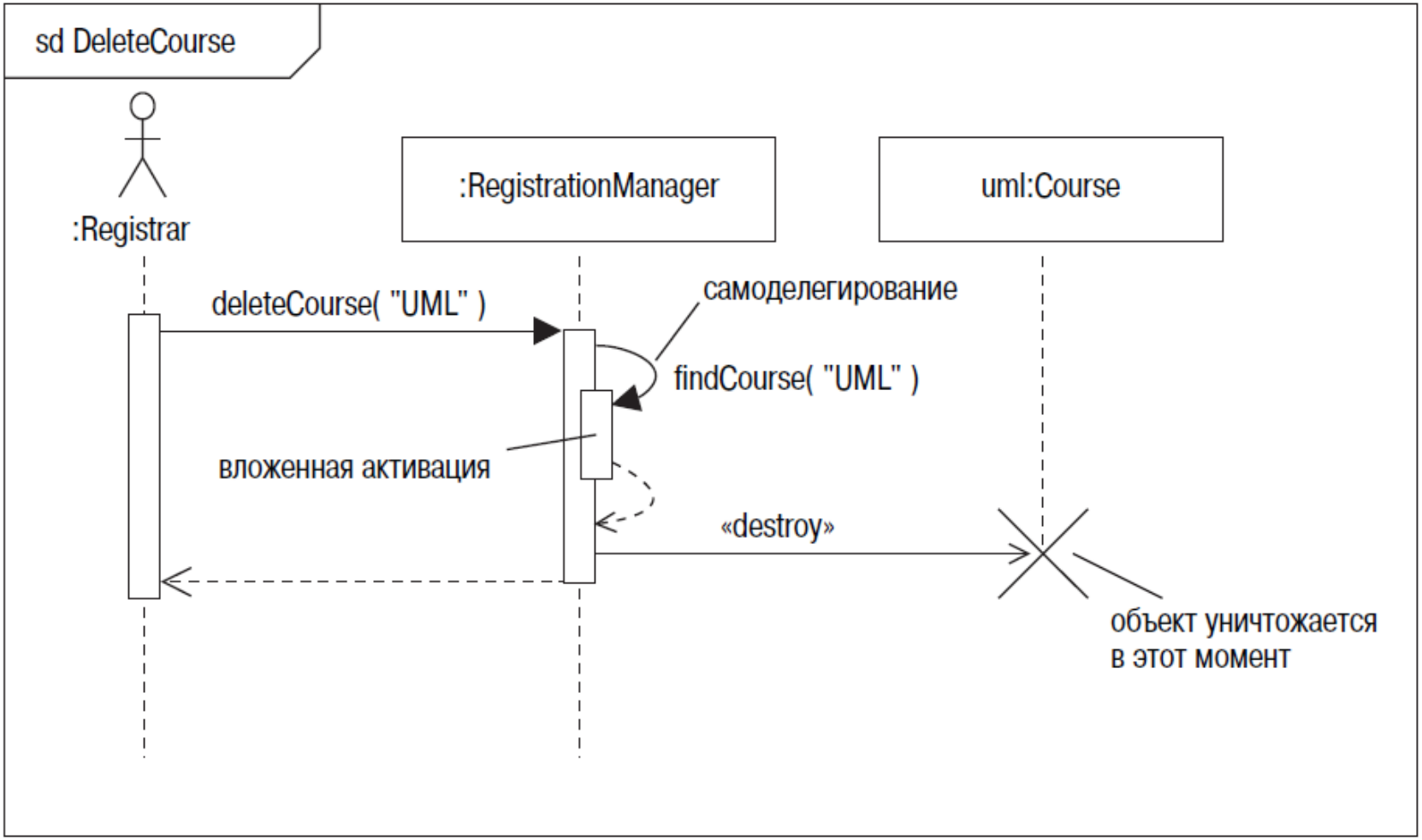
ТИПЫ СООБЩЕНИЙ

Синтаксис	Имя	Семантика
	Синхронное сообщение	Отправитель ожидает завершения выполнения сообщения получателем.
	Асинхронное сообщение	Отправитель посылает сообщение и продолжает исполнение – он *не* ожидает возврата от получателя.
	Возврат	Получатель сообщения возвращает фокус управления отправителю этого сообщения.
	Создание объекта	Отправитель создает экземпляр классификатора, определенного получателем.
	Уничтожение объекта	Отправитель уничтожает получателя. Если у линии жизни есть «хвост», он завершается символом X.
	Найденное сообщение	Отправитель сообщения находится вне области видимости взаимодействия. Используется, когда необходимо показать получение сообщения без указания его источника.
	Потерянное сообщение	Сообщение никогда не достигает точки своего назначения. Может использоваться для обозначения состояний ошибки, при которых пропадают сообщения.

ПРИМЕР

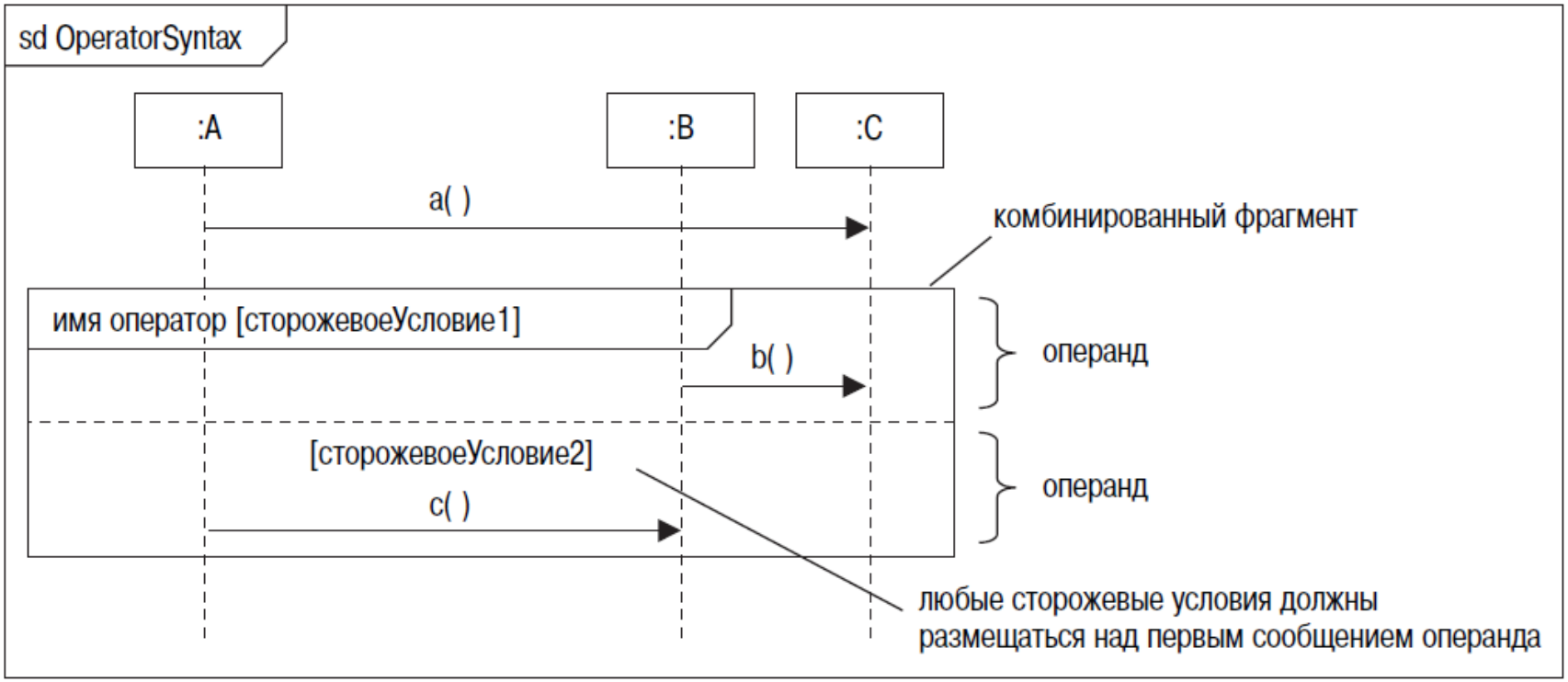
Прецедент: DeleteCourse
ID: 8
Краткое описание: Удаляет курс из системы.
Главные актеры: Registrar
Второстепенные актеры: Нет.
Предусловия: 1. Registrar вошел в систему.
Основной поток: 1. Registrar выбирает «delete cource». 2. Registrar вводит имя курса. 3. Система удаляет курс.
Постусловия: 1. Курс удален из системы.
Альтернативные потоки: CourseDoesNotExist

ПРИМЕР


sd DeleteCourse
:Registrar
:RegistrationManager
uml:Course
deleteCourse("UML")
самоделегирование
findCourse("UML")
вложенная активация
«destroy»
объект уничтожается
в этот момент

КОМБИНИРОВАННЫЕ ФРАГМЕНТЫ

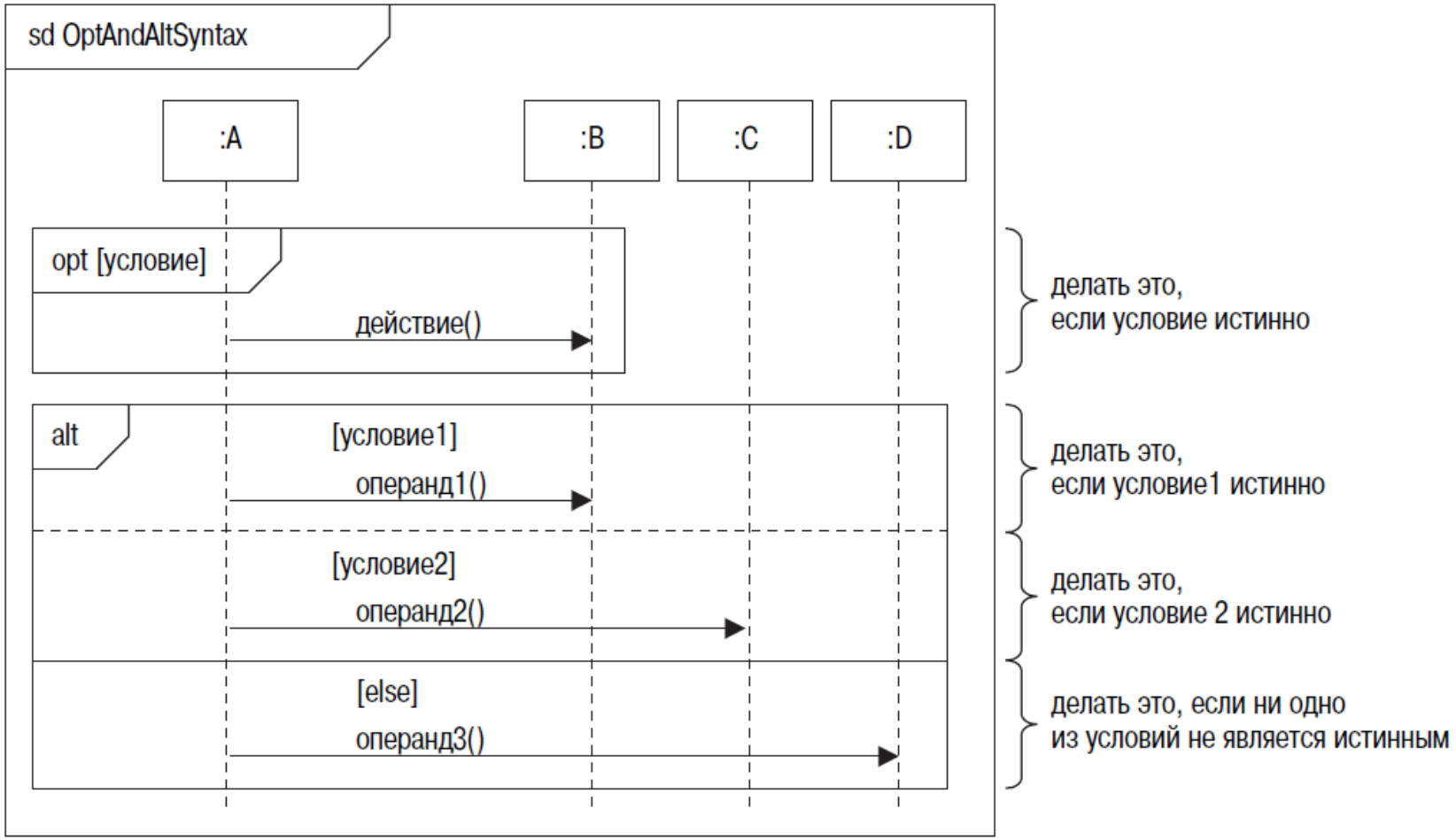
- Комбинированные фрагменты разделяют диаграмму последовательностей на области с различным поведением.



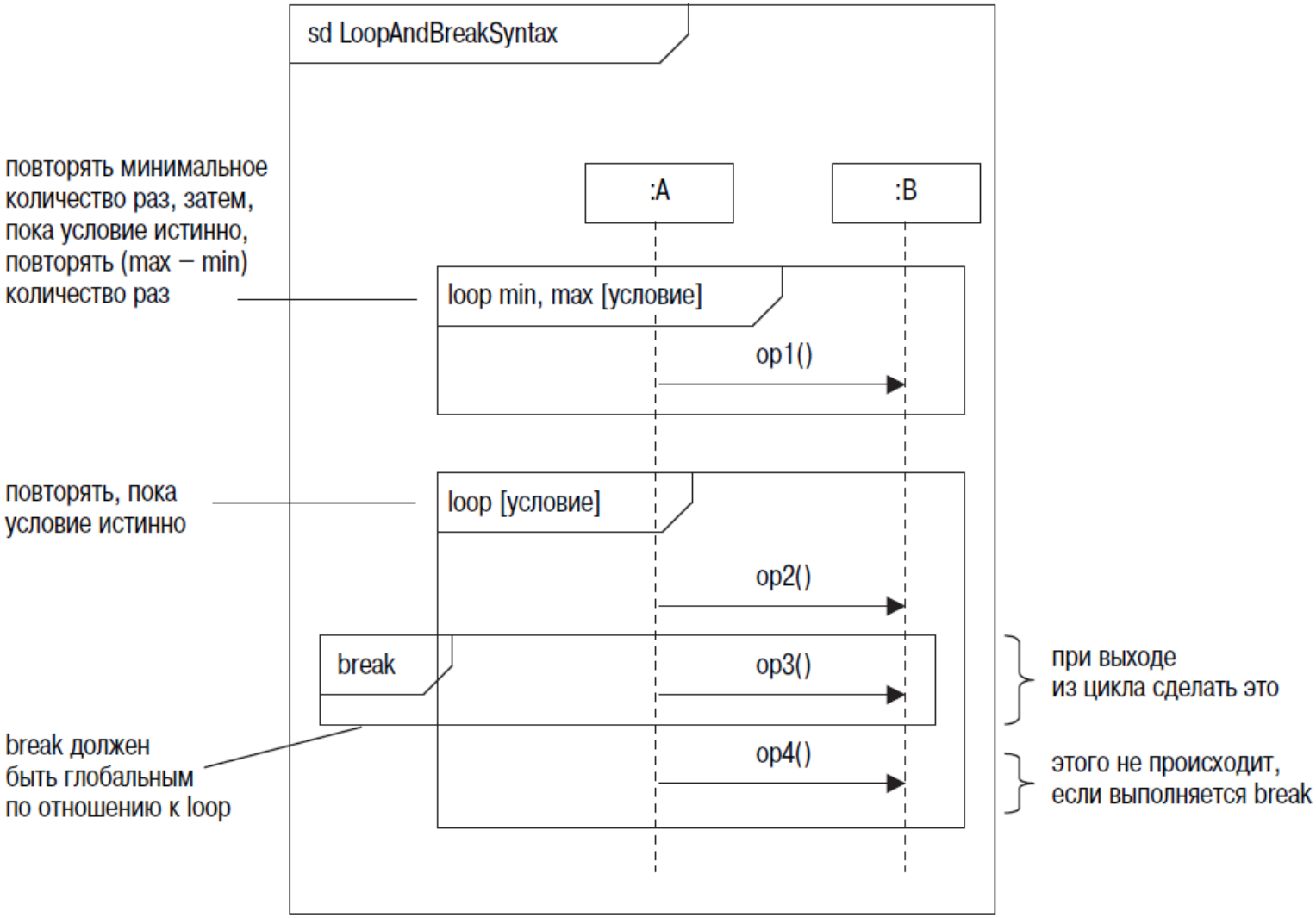
ОСНОВНЫЕ ОПЕРАТОРЫ

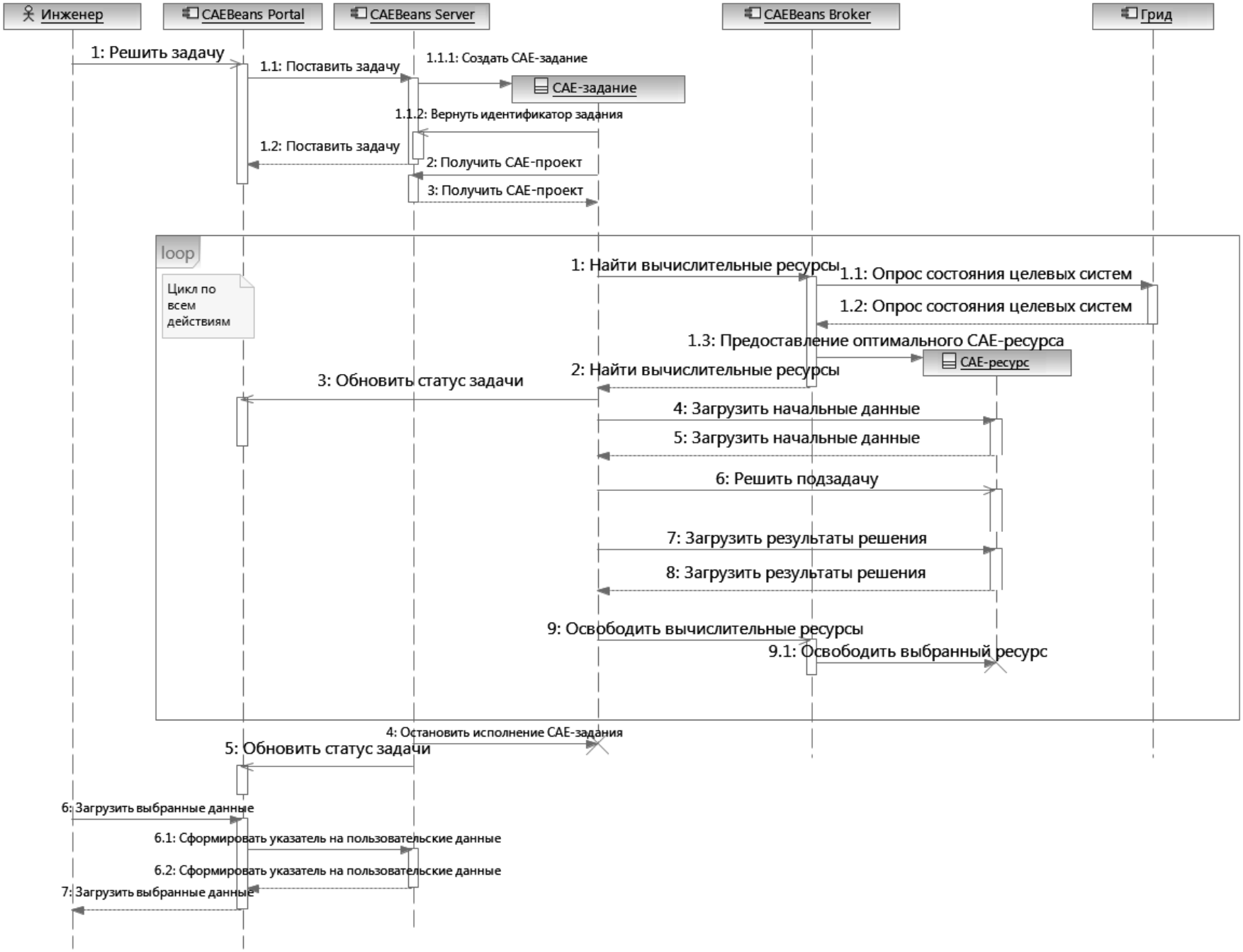
- **opt** (option) - Единственный операнд выполняется, если условие истинно (как if … then).
- **alt** (alternatives) - Выполняется тот операнд, условие которого истинно. Вместо логического выражения может использоваться ключевое слово else (как select … case).
- **loop** - loop min, max [condition] повторять минимальное количество раз, затем, пока условие истинно, повторять еще (max – min) число раз.

ПРИМЕНЕНИЕ OPT И ALT



ПРИМЕНЕНИЕ LOOP




Инженер
CAEBeans Portal
CAEBeans Server
CAEBeans Broker
Грид
1: Решить задачу
1.1: Поставить задачу
1.1.1: Создать CAE-задание
CAE-задание
1.1.2: Вернуть идентификатор задания
1.2: Поставить задачу
2: Получить CAE-проект
3: Получить CAE-проект
loop
Цикл по всем действиям
1: Найти вычислительные ресурсы
1.1: Опрос состояния целевых систем
1.2: Опрос состояния целевых систем
1.3: Предоставление оптимального CAE-ресурса
CAE-ресурс
2: Найти вычислительные ресурсы
3: Обновить статус задачи
4: Загрузить начальные данные
5: Загрузить начальные данные
6: Решить подзадачу
7: Загрузить результаты решения
8: Загрузить результаты решения
9: Освободить вычислительные ресурсы
9.1: Освободить выбранный ресурс
4: Остановить исполнение CAE-задания
5: Обновить статус задачи
6: Загрузить выбранные данные
6.1: Сформировать указатель на пользовательские данные
6.2: Сформировать указатель на пользовательские данные
7: Загрузить выбранные данные

ДИАГРАММЫ ДЕЯТЕЛЬНОСТИ

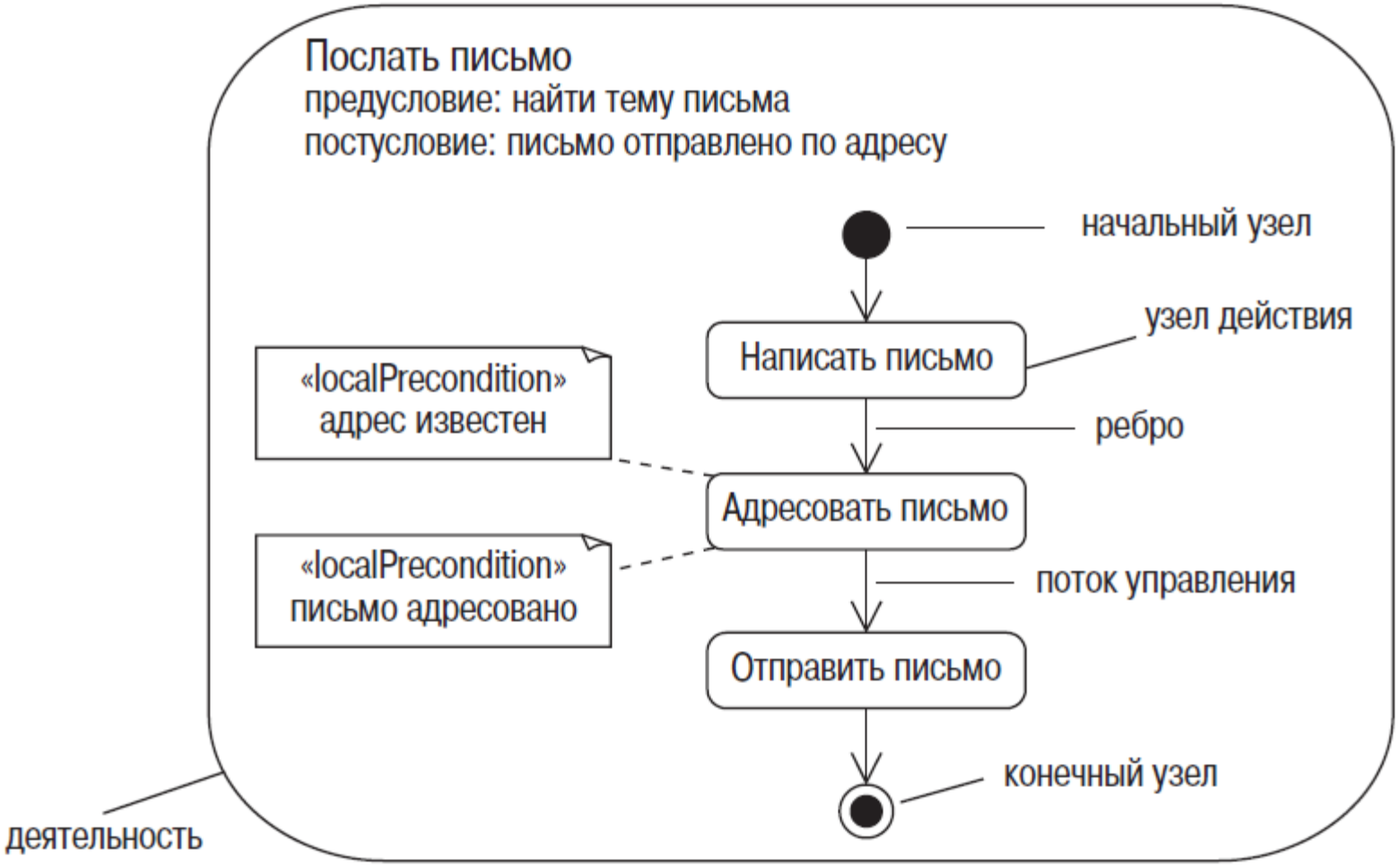
ДИАГРАММЫ ДЕЯТЕЛЬНОСТИ

- **Диаграммы деятельности** – это ОО блоксхемы.
- Они позволяют моделировать процесс как деятельность, состоящую из коллекции соединенных ребрами узлов.
- Деятельность может быть добавлена к **любому элементу модели** с целью моделирования его поведения. *Обычно: прецеденты, классы, интерфейсы, компоненты, операции.*

ДЕЯТЕЛЬНОСТИ

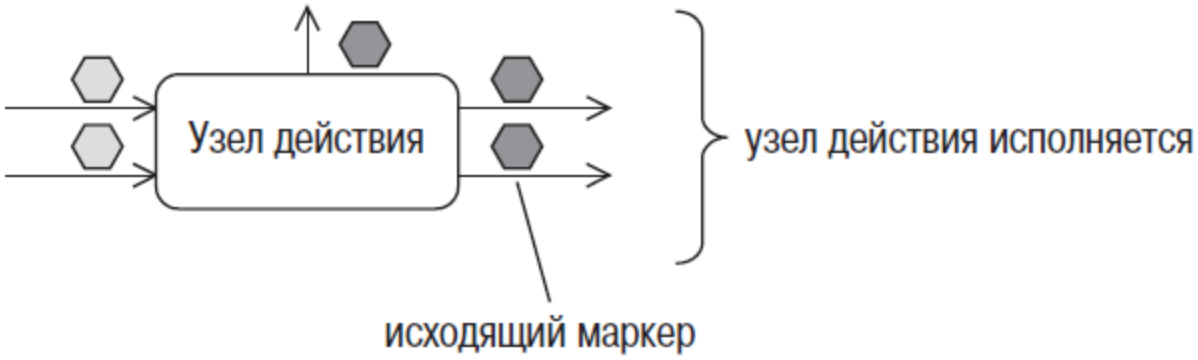
- **Деятельности** – это системы *узлов, соединенных ребрами.* Существует три категории узлов:
  - **Узлы действия** (action nodes) – представляют отдельные единицы работы, элементарные *в рамках деятельности;*
  - **Узлы управления** (control nodes) – управляют потоком деятельности;
  - **Объектные узлы** (object nodes) – представляют объекты, используемые в деятельности.

ПРИМЕР ДИАГРАММЫ ДЕЯТЕЛЬНОСТИ



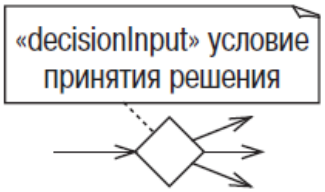
УЗЕЛ ДЕЙСТВИЯ

- Узел действия может инициировать деятельность, поведение или операцию (имена – глагольные группы)
- Узлы действия осуществляют операцию логическое И над своими входящими маркерами – узел не готов к исполнению до тех пор, пока маркеры не будут присутствовать на *всех входящих ребрах.*

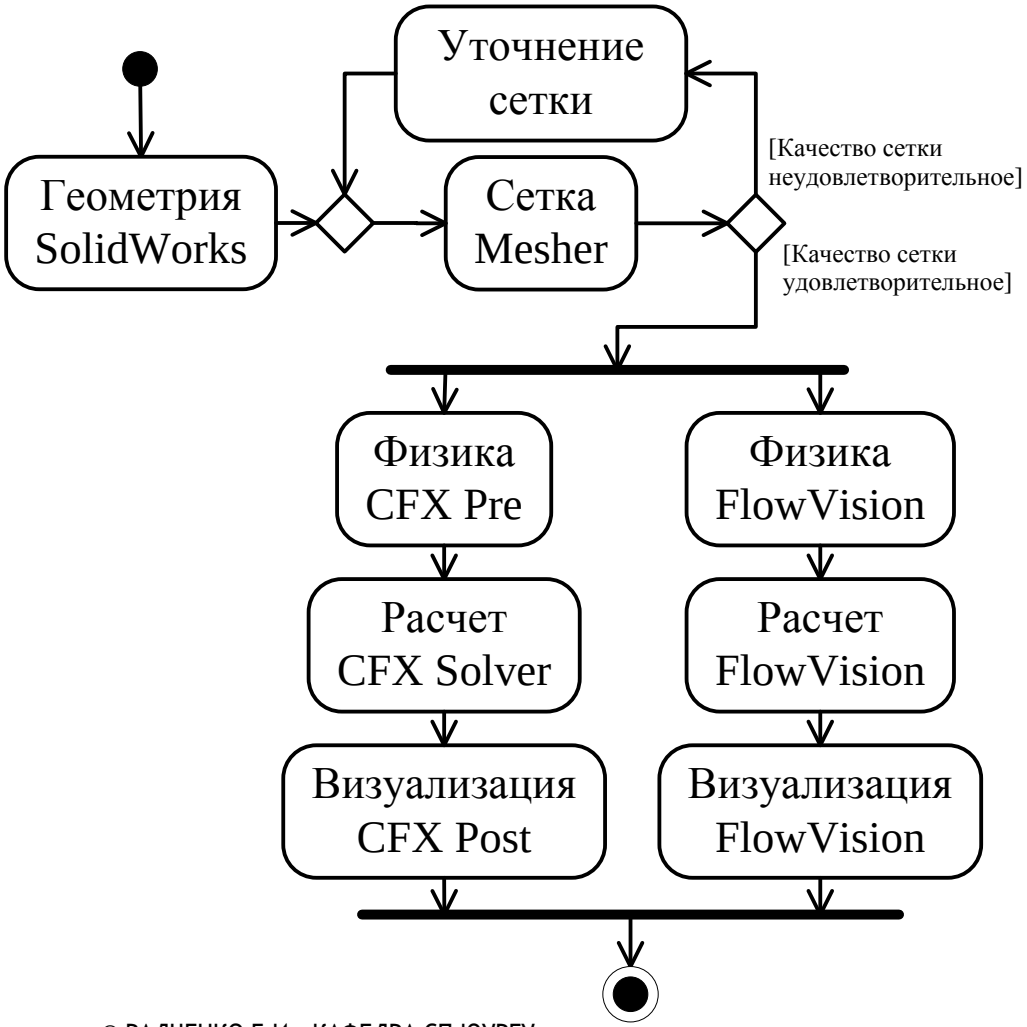


ПРИМЕРЫ ПРИМЕНЕНИЯ УЗЛА ДЕЙСТВИЯ

УЗЛЫ УПРАВЛЕНИЯ

Синтаксис	Имя	Семантика	
	Начальный узел	Указывает, где начинается поток при вызове деятельности.	
	Конечный узел деятельности	Завершает деятельность.	Конечные узлы
	Конечный узел потока	Завершает определенный поток деятельности – другие потоки не затрагиваются.	
	Узел решения	Поток проходит по исходящему ребру, сторожевое условие которого истинно. Может иметь входные данные (необязательно).	
	Узел слияния	Копирует входные маркеры в единственное выходное ребро.	
	Узел ветвления	Разделяет поток на несколько параллельных потоков.	
	Узел объединения	Синхронизирует несколько параллельных потоков. Может иметь описание объединения (не обязательно) для изменения его семантики.	

ПРИМЕР ДИАГРАММЫ ДЕЯТЕЛЬНОСТИ



Кодирование: Метрики разработки ПО

Мера и Метрика

- *Мера* - количественный показатель степени, количества, или размеров некоторых атрибутов продукта или процесса.
  - Например, количество ошибок

- *Метрика* - количественная мера позволяющая оценить, в какой степени система, компоненты или процесс обладают заданным атрибутом. “Предположение о значении данного атрибута.”
  - Например, количество обнаруженных ошибок на затраченный человеко-час

Зачем измерять ПО?

- Определение качества существующего продукта или процесса
- Прогнозирование качества продукта / процесса
- Улучшение качества продукта / процесса

Мотивация для метрик

- Оценка стоимости и графика будущих проектов
- Оценка производительности применения новых средств и методов
- Определение тенденций производительности с течением времени
- Улучшение качества программного обеспечения
- Прогноз будущих потребностей в персонале
- Предвидеть и сокращения будущих потребностей в техническом обслуживании

Классификация метрик

- Продукты
  - Результаты деятельности разработки программного обеспечения
  - Конечная продукция, документация по продуктам
- Процессы
  - Деятельность, связанная с производством программного обеспечения
- Ресурсы
  - База для процесса разработки программного обеспечения
  - Оборудование, знания, люди

Метрики сложности программного кода

Метрики сложности программ принято разделять на три основные группы:

1. метрики размера программ;
2. метрики сложности потока управления программ;
3. метрики сложности потока данных программ.

Размерно-ориентированные метрики

Размерно-ориентированные метрики

LOC-оценка (Lines Of Code) может включать в себя:

- общие трудозатраты (в человеко-месяцах, человеко-часах);
- объем программы (в тысячах строках исходного кода - LOC);
- стоимость разработки;
- объем документации;
- ошибки, обнаруженные в течение года эксплуатации;
- количество людей, работавших над изделием;
- срок разработки.

Физические и логические строки кода

- «**Физические**» строки кода – SLOC (используемые аббревиатуры **LOC, SLOC, KLOC, KSLOC, DSLOC**) – определяется как общее число строк исходного кода, включая комментарии и пустые строки.
- «**Логические**» строки кода – SLOC (используемые аббревиатуры **LSI, DSI, KDSI, где «SI»** - source instructions) – определяется как количество команд и зависит от используемого языка программирования.

Недостатки SLOC

- Неправильно сводить оценку работы человека к нескольким числовым параметрам. Менеджер может назначить наиболее талантливых программистов на сложнейший участок работы.
- Метрика не учитывает опыт сотрудников и их другие качества.
- Искажение: процесс измерения может быть искажён за счёт того, что сотрудники знают об измеряемых показателях и стремятся оптимизировать эти показатели, а не свою работу.
- Неточность: нет метрик, которые были бы одновременно и значимы и достаточно точны. Количество строк кода — это просто количество строк, этот показатель не даёт представления о сложности решаемой проблемы.

Метрики сложности

Метрики сложности

- **Объектно-ориентированные**: оценка сложности объектно-ориентированных проектов
- **Метрики Холстеда**: вычисляются на основании анализа числа строк и синтаксических элементов исходного кода программы
- **Цикломатическая сложность** : один из наиболее распространенных показателей оценки сложности программных проектов на основе графа управляющей логики
- **Метрика Чепина**: оценка информационной прочности

Объектно-ориентированные метрики

Метрика	Описание
Взвешенная насыщенность класса 1 (Weighted Methods Per Class (WMC)	Отражает относительную меру сложности класса на основе цикломатической сложности каждого его метода. Класс с более сложными методами и большим количеством методов считается более сложным. При вычислении метрики родительские классы не учитываются.
Взвешенная насыщенность класса 2 (Weighted Methods Per Class (WMC2))	Мера сложности класса, основанная на том, что класс с большим числом методов, является более сложным, и что метод с большим количеством параметров также является более сложным. При вычислении метрики родительские классы не учитываются.
Глубина дерева наследования (Depth of inheritance tree)	Длина самого длинного пути наследования, заканчивающегося на данном модуле. Чем глубже дерево наследования модуля, тем может оказаться сложнее предсказать его поведение. С другой стороны, увеличение глубины даёт больший потенциал повторного использования данным модулем поведения, определённого для классов-предков.
Количество детей (Number of children)	Число модулей, непосредственно наследующих данный модуль. Большие значения этой метрики указывают на широкие возможности повторного использования; при этом слишком большое значение может свидетельствовать о плохо выбранной абстракции.
Связность объектов (Coupling between objects)	Количество модулей, связанных с данным модулем в роли клиента или поставщика. Чрезмерная связность говорит о слабости модульной инкапсуляции и может препятствовать повторному использованию кода.
Отклик на класс (Response For Class)	Количество методов, которые могут вызываться экземплярами класса; вычисляется как сумма количества локальных методов, так и количества удаленных методов

Метрики Холстеда

Основу метрики Холстеда составляют четыре измеряемые характеристики программы:

- **NUOprtr** (Number of Unique Operators) — число уникальных операторов программы, включая символы-разделители, имена процедур и знаки операций (словарь операторов);
- **NUOprnd** (Number of Unique Operands) — число уникальных операндов программы (словарь операндов);
- **Noprtr** (Number of Operators) — общее число операторов в программе;
- **Noprnd** (Number of Operands) — общее число операндов в программе.

Оценки на основе Метрики Холстеда

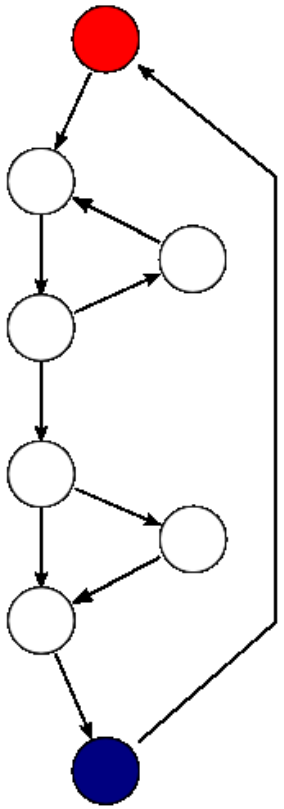
- **Словарь программы** (Halstead Program Vocabulary):
  - HPVoc = NUOprtr + NUOprnd;
- **Длина программы** (Halstead Program Length):
  - HPLen = Noprtr + Noprnd;
- **Объем программы** (Halstead Program Volume):
  - HPVol = HPLen log2 HPVoc;
- **Сложность программы** (Halstead Difficulty, HDiff):
  - HDiff = (NUOprtr/2) × (NOprnd / NUOprnd);
- На основе показателя HDiff предлагается оценивать **усилия программиста** при разработке при помощи показателя HEff (Halstead Effort):
  - HEff = HDiff × HPVol.

Цикломатическая сложность

- Цикломатическая сложность части программного кода — счётное число линейно независимых маршрутов через программный код.
- Если исходный код не содержит никаких точек решений, таких, как указания **IF** или циклы **FOR**, то сложность равна единице, поскольку, есть только единственный маршрут через код.

ЦС: Граф потока управления программы

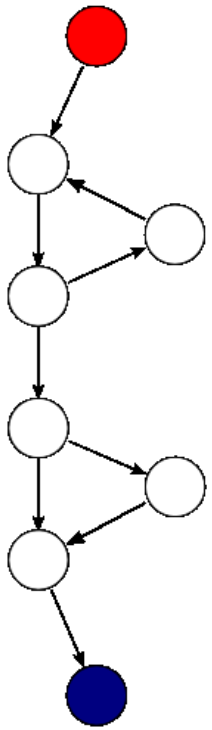
- Графом потока управления программы называется ориентированный граф, в узлах которого находятся неделимые группы команд, а ребрами соединяются такие блоки, которые могут быть выполнены сразу друг за другом.



(2) Цикломатическая сложность

- Показатель цикломатической сложности позволяет не только **произвести оценку** трудоемкости реализации отдельных элементов программного проекта и скорректировать общие показатели оценки длительности и стоимости проекта, но и **оценить связанные риски** и принять необходимые управленческие решения.
- **C = e – n + 2p**, где e – число ребер, n – число узлов, а p – число компонентов связности на графе управляющей логики.

Пример цикломатической сложности



Программа начинает выполняться с красного узла,
затем идут циклы (после красного узла идут две группы по три узла).
Выход из цикла осуществляется через условный оператор (нижняя группа узлов)
и конечный выход из программы в синем узле. Для этого графа E = 9, N = 8 и P = 1,
цикломатическая сложность программы равна 3.

Метрики Чепина

Все множество переменных, составляющих список ввода-вывода, разбивается на четыре функциональные группы.

- **Множество «P»** – вводимые переменные для расчетов и для обеспечения вывода. Примером может служить используемая в программах лексического анализатора переменная, содержащая строку исходного текста программы, то есть сама переменная не модифицируется, а только содержит исходную информацию.

- **Множество «M»** – модифицируемые или создаваемые внутри программы переменные.

- **Множество «C»** – переменные, участвующие в управлении работой программного модуля (управляющие переменные).

- **Множество «T»** – не используемые в программе (“паразитные”) переменные.

Поскольку каждая переменная может выполнять одновременно несколько функций, необходимо учитывать ее в каждой соответствующей функциональной группе.

Вычисление метрики Чепина

В общем виде

- Q = a1P + a2M + a3C + a4T,

где a1, a2, a3, a4 – весовые коэффициенты.

Автор предлагает следующие значения коэффициентов:

- **Q = P + 2M + 3C + 0.5T**

Единственно-правильная метрика качества программного кода