Учебный курс

МЕТОДЫ И СРЕДСТВА ПРОЕКТИРОВАНИЯ ПО

Лекция 5

История развития языка UML

Базовые принципы и понятия технологии разработки объектно-ориентированных информационных систем на основе UML

UML

История развития языка UML

Основные этапы развития языка UML

- Отдельные языки объектно-ориентированного моделирования начали появляться в середине 1970-х годов, когда различные исследователи и программисты предлагали свои подходы к ООАП. В период между 1989 -1994 гг. общее число наиболее известных языков моделирования возросло с 10 до более чем 50. Многие пользователи испытывали серьезные затруднения при выборе языка ООАП, поскольку ни один из них не удовлетворял всем требованиям, предъявляемым к построению моделей сложных систем. Принятие отдельных методик и графических нотаций в качестве стандартов (IDEF0, IDEF1X) не смогло изменить сложившуюся ситуацию непримиримой конкуренции между ними в начале 90-х годов, которая получила название "войны методов".

История развития языка UML

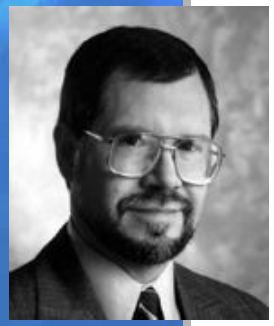
К середине 1990-х некоторые методы были существенно улучшены и приобрели самостоятельное значение при решении различных задач ООАП. Наиболее известными в этот период становятся следующие три, каждый из этих которых ориентирован на поддержку отдельных этапов ООАП:

- Метод Гради Буча (Grady Booch), получивший условное название Booch или Booch'91, Booch Lite (позже - Booch'93), OODA. Сильная сторона - дизайн, а слабая - анализ, нашел широкое применение на этапах проектирования и разработки различных программных систем.
- Метод Джеймса Рамбо (James Rumbaugh), наименованный Object Modeling Technique - OMT (позже - OMT-2). Сильная сторона - анализ, а слабая — дизайн, наиболее подходил для анализа процессов обработки
- Метод Айвара Якобсона (Ivar Jacobson), под названием Object-Oriented Software Engineering – OOSE. Сильная сторона - анализ поведения (behavior analysis), однако в остальных областях он достаточно слаб, содержал средства представления вариантов использования, которые имеют существенное значение на этапе анализа требований в процессе проектирования бизнес-приложений.

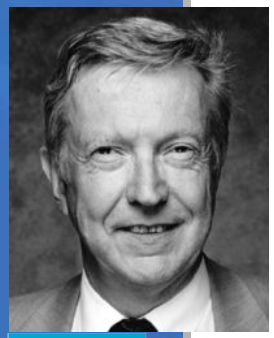
История UML

Three Amigos:

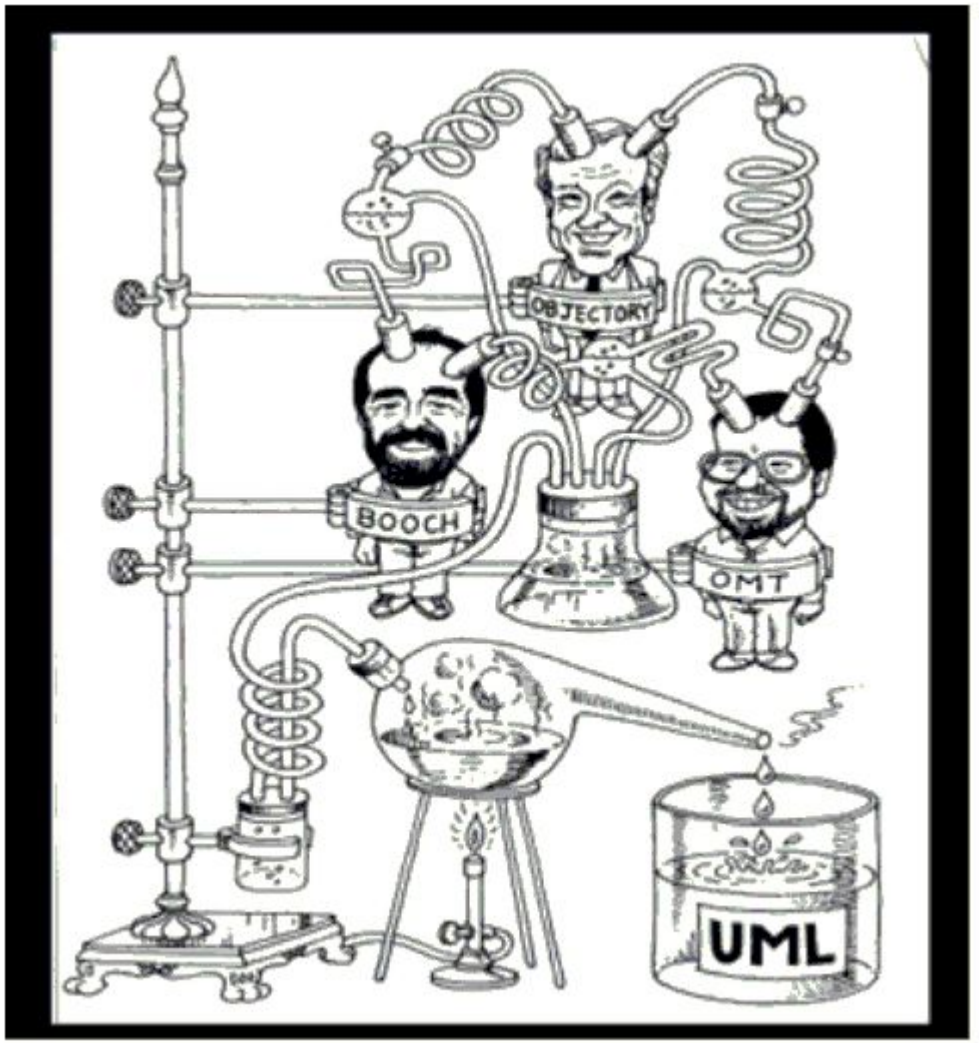
Grady Booch



James Rumbaugh



Ivar Jacobson



Rational. software

История развития языка UML

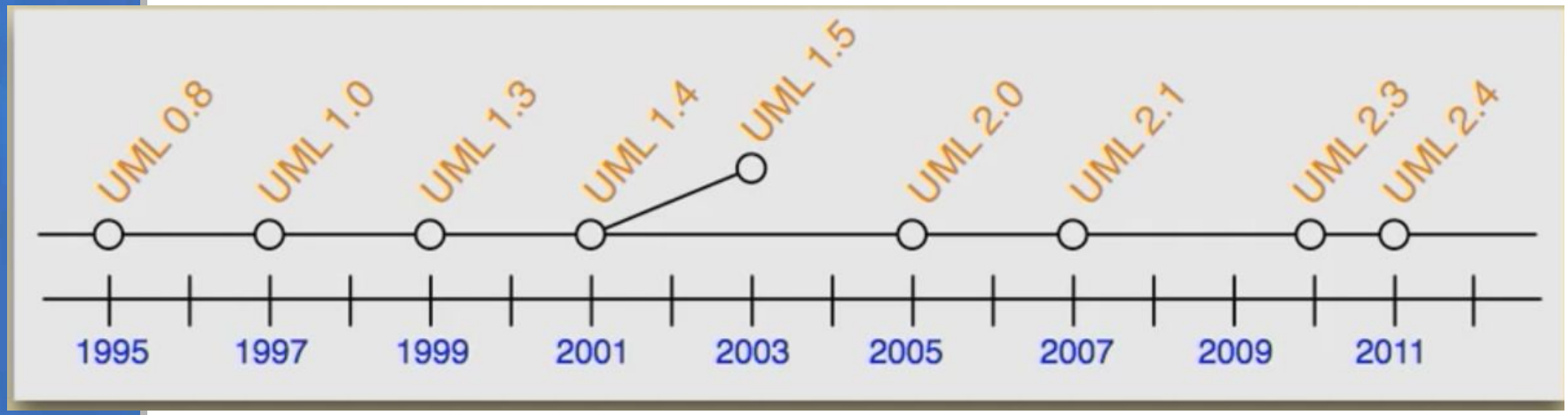
- История развития языка UML берет начало с октября 1994 года, когда Гради Буч и Джеймс Рамбо из компании Rational Software Corporation начали работу по унификации методов Booch и OMT. Несмотря на то, что сами по себе эти методы были достаточно популярны, совместная работа была направлена на изучение всех известных объектно-ориентированных методов с целью объединения их достоинств. При этом Г. Буч и Дж. Рамбо сосредоточили усилия на полной унификации результатов своей работы. Проект так называемого унифицированного метода (Unified Method) версии 0.8 был подготовлен и опубликован в октябре 1995 года. Осенью того же года к ним присоединился А. Якобсон, главный технолог компании Objectory AB (Швеция), с целью интеграции своего метода OOSE с двумя предыдущими.
- В этот период поддержка разработки языка UML становится одной из целей консорциума OMG (Object Management Group), который был образован еще в 1989 году с целью разработки предложений по стандартизации объектных и компонентных технологий CORBA. В то время язык UML приобрел статус второго стратегического направления в работе OMG. Именно в OMG создается команда разработчиков под руководством Р. Соли, которая обеспечила дальнейшую работу по унификации и стандартизации языка UML. Усилия группы разработчиков, в которую входили также Г. Буч, Дж. Рамбо и А. Якобсон, привели к появлению первых документов, содержащих собственно описание языка UML версии 0.9 (июнь 1996 г.) и версии 0.91 (октябрь 1996 г.).

История развития языка UML

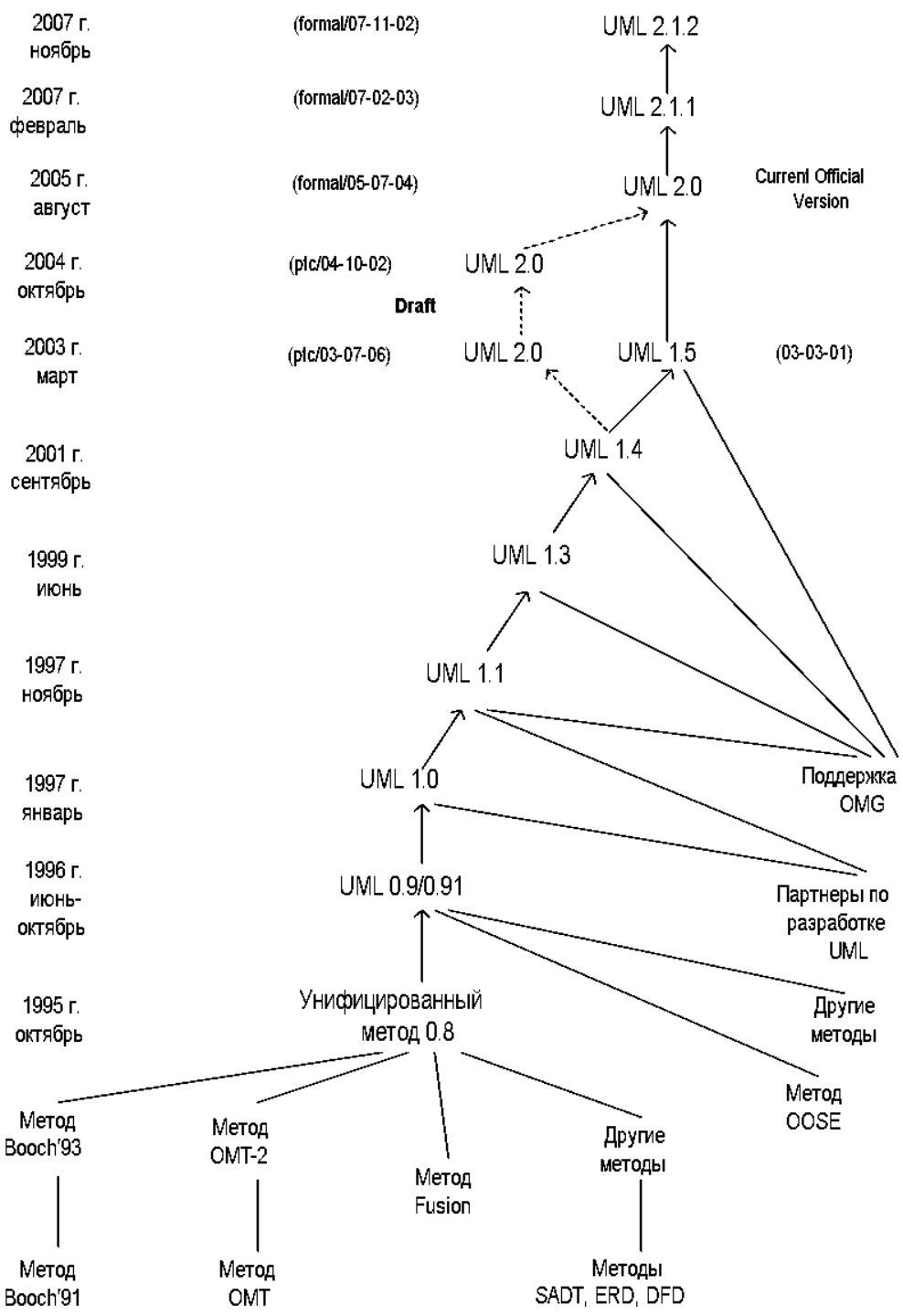
- Тогда же некоторые компании и организации увидели в языке UML стратегический интерес для своего бизнеса. Компания Rational Software вместе с несколькими организациями, изъявившими желание выделить ресурсы для разработки строгого определения версии 1.0 языка UML, учредила консорциум партнеров UML, в который первоначально вошли такие фирмы, как Digital Equipment Corp., HP, i-Logix, Intellicorp, IBM, ICON Computing, MCI Systemhouse, Microsoft, Oracle, Rational Software, TI и Unisys. Эти компании обеспечили поддержку последующей работы по более точному определению нотации.
- В январе 1997 года был опубликован документ с описанием языка UML 1.0, как начальный вариант ответа на запрос предложений RTP. Эта версия языка моделирования была достаточно хорошо определена, обеспечивала требуемую выразительность и мощность, предполагала решение широкого класса задач. В результате работы инициативной группы в составе OMG была предложена пересмотренная версия 1.1 языка UML. Основное внимание при разработке языка UML 1.1 было уделено достижению большей ясности семантики по сравнению с UML 1.0, а также учету предложений новых партнеров. Эта версия языка была представлена на рассмотрение OMG, затем одобрена и принята в качестве стандарта OMG в ноябре 1997 года.

История развития языка UML

- Если проследить историю возникновения и развития элементов UML, как на уровне основополагающих идей и технических деталей, то пришлось бы назвать сотни имен и десятки организаций. Но язык постоянно совершенствуется, обогащается и расширяется
- Линейная упорядоченность версий на рисунке имеет одно отклонение.
- На особом положении оказалась версия 1.5. Она появилась, когда ожидалось появление версии 2.0, и содержит набор элементов, позволяющий применять UML не только, как язык моделирования, но и как язык программирования. Генеральная линия развития инструментальных средств прошла мимо этого явления. Крупные поставщики инструментов предпочли заявить о поддержке версии 2.0, не имея для этого достаточно оснований, и оставили без поддержки версию 1.5.



История развития языка UML



Версии UML

Версия	Дата принятия
1.1	ноябрь 1997
1.3	март 2000
1.4	сентябрь 2001
1.4.2.	июль 2004
1.5	март 2003
2.0	июль 2005
2.1	формально не была принята
2.1.1	август 2007
2.1.2	ноябрь 2007
2.2	февраль 2009
2.3	май 2010
2.4 beta 2	март 2011
2.5	июнь 2015

История развития языка UML

- Текущей версией языка UML является версия 2.5, принятая консорциумом OMG в июне 2015 г. В августе-сентябре 2003 г. был опубликован проект языка UML 2.0, но эта версия официально принята.в июле 2005.
- В настоящее время все вопросы дальнейшей разработки языка UML сконцентрированы в рамках консорциума OMG. При этом статус языка UML определен как открытый для всех предложений по его доработке и совершенствованию. Сам язык UML не является чьей-либо собственностью и не запатентован кем-либо. В тоже время аббревиатура UML, как и некоторые другие (OMG, CORBA, ORB), является торговой маркой их законных владельцев.

История развития языка UML

- На рынке CASE-средств представлены десятки программных инструментов, поддерживающих разные нотации языка UML и обеспечивающих интеграцию, включая прямую и обратную генерацию кода программ, с наиболее распространенными языками и средами программирования, такими как MS Visual C++, Java, Object Pascal/Delphi, Power Builder, MS Visual Basic, Forte, Ada, Smalltalk, MS Visual Studio.
- С каждым годом интерес к языку UML со стороны специалистов неуклонно возрастает. Язык UML повсеместно становится не только основой для разработки и реализации во многих перспективных инструментальных RAD-средствах, но и в CASE-средствах визуального и имитационного моделирования. Более того, заложенные в языке UML потенциальные возможности широко используются как для объектно-ориентированного моделирования систем, так и для документирования бизнес-процессов, а в более широком контексте - для представления знаний в интеллектуальных системах, которыми, по существу, станут перспективные сложные программно-технологические комплексы.

UML

UML - унифицированный язык моделирования

UML - унифицированный **язык** моделирования

UML - унифицированный язык моделирования. Из этих трех слов главным является слово " **язык** ". Что же такое язык?

Язык - система знаков, служащая:

- средством человеческого общения и мыслительной деятельности;
- способом выражения самосознания личности;
- средством хранения и передачи информации.

Язык включает в себя набор знаков (словарь) и правила их употребления и интерпретации (грамматику).

UML - унифицированный **язык** моделирования

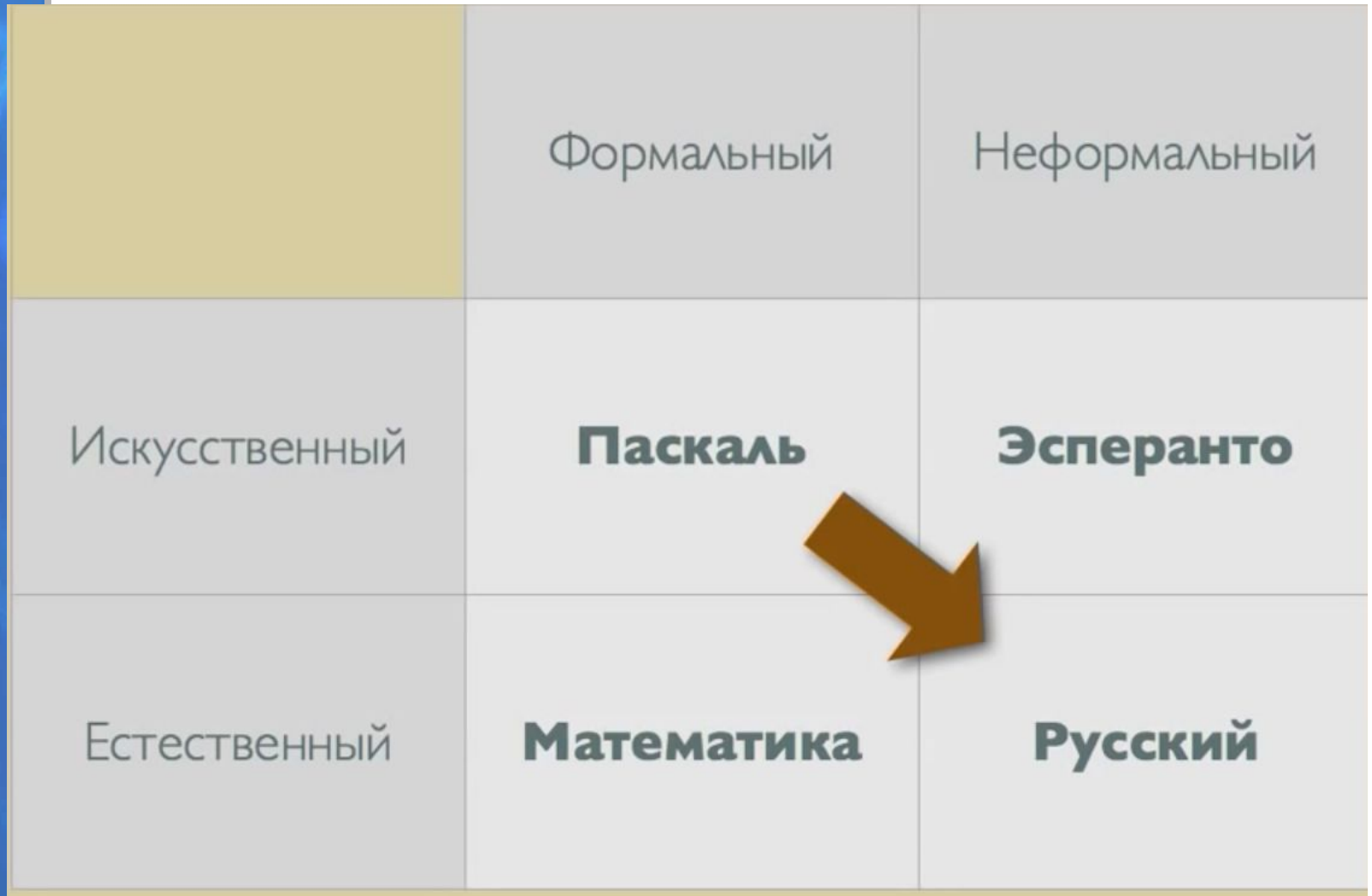
К этому нужно добавить, что языки бывают естественные и искусственные, формальные и неформальные. UML - язык формальный и искусственный.

Искусственный он потому, что у него имеются авторы, в то же время, развитие UML непрерывно продолжается, что ставит его в один ряд с естественными языками.

Формальным его можно назвать, поскольку имеются правила его употребления (правда, описание UML содержит и явно неформальные элементы).

Еще один нюанс: UML - язык графический, что также немного путает ситуацию!

UML - унифицированный **ЯЗЫК** моделирования



UML - унифицированный **язык** моделирования

При описании формального искусственного языка (например языков программирования), как правило, описывают:

- *синтаксис*, то есть определение правил построения конструкций языка;
- *семантика*, то есть определение правил, в соответствии с которыми конструкции языка приобретают смысловое значение;
- *прагматика*, то есть определение правил использования конструкций языка для достижения нужных нам целей.

Естественно, UML включает все эти элементы, хотя в их описании тоже наблюдаются отличия от правил, принятых в языках программирования

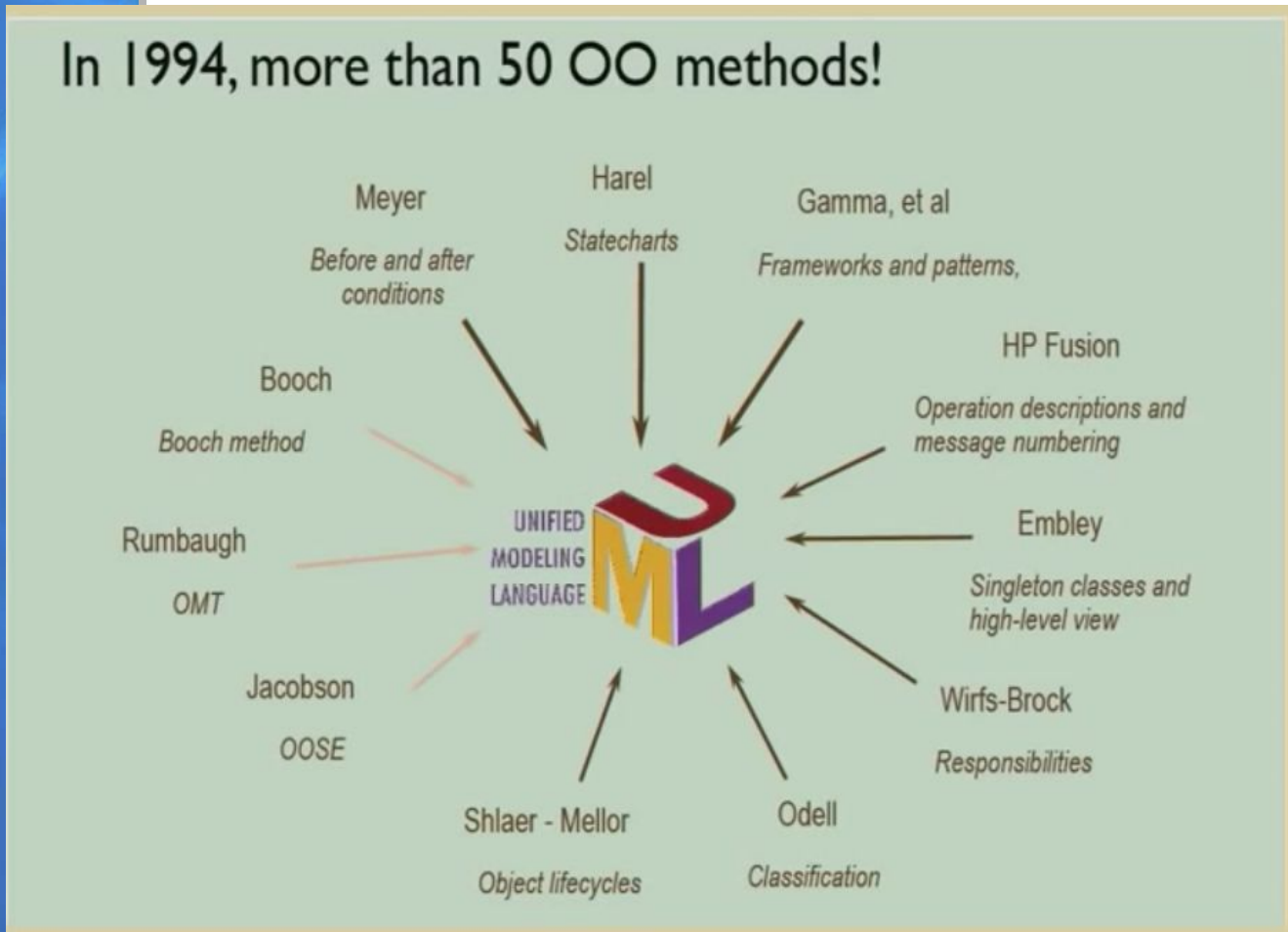
UML - унифицированный язык **МОДЕЛИРОВАНИЯ**

- Второе слово в фразе, которой расшифровывается аббревиатура UML - слово " **моделирование** ".
- UML - это язык моделирования, причем объектно-ориентированного моделирования.
- Слово «моделирование» весьма многозначно.
- В английском языке есть два слова - **modeling** и **simulation**, которые оба переводятся как "моделирование", хотя означают разные понятия. Modeling подразумевает создание модели, лишь описывающей объект, а simulation предполагает получение с помощью созданной модели некоторой дополнительной информации об объекте.
- UML в первую очередь - язык моделирования именно в первом смысле, то есть средство построения описательных моделей. Как средство симулирования его тоже можно использовать, хотя для этой роли он подходит не так хорошо.

UML - **УНИФИЦИРОВАННЫЙ** язык моделирования

- Третье слово в названии UML - слово " **унифицированный**". Его можно понимать тоже неоднозначно. В литературе можно встретить описание эры "до UML" как "войны методов" моделирования, ни один из которых "не дотягивал" до уровня индустриального стандарта. UML как раз и стал таким единым универсальным стандартом для объектно-ориентированного моделирования, которое во времена его создания "вошло в моду". "Единым" языком моделирования UML можно назвать еще и потому, что в его создании объединились усилия авторов трех наиболее популярных методов моделирования (и не только их).

UML - **УНИФИЦИРОВАННЫЙ** язык моделирования



- ...
- Петроглифы
- Блок-схемы
- Р-технология
- Диаграмма потоков данных (DFD)
- Диаграмма "сущность-связь" (ERD)
- Методология структурного анализа и проектирования (SADT)
- SDL
- MSC
- ...

В начале 80-х стартовала "объектно-ориентированная эра". Все началось с появлением семейства языков программирования SmallTalk (они применяли некоторые понятия языка Simula-67, использовавшегося в 60-х годах).

Появление объектно-ориентированного подхода было обусловлено увеличением сложности задач.

Объектно-ориентированный подход

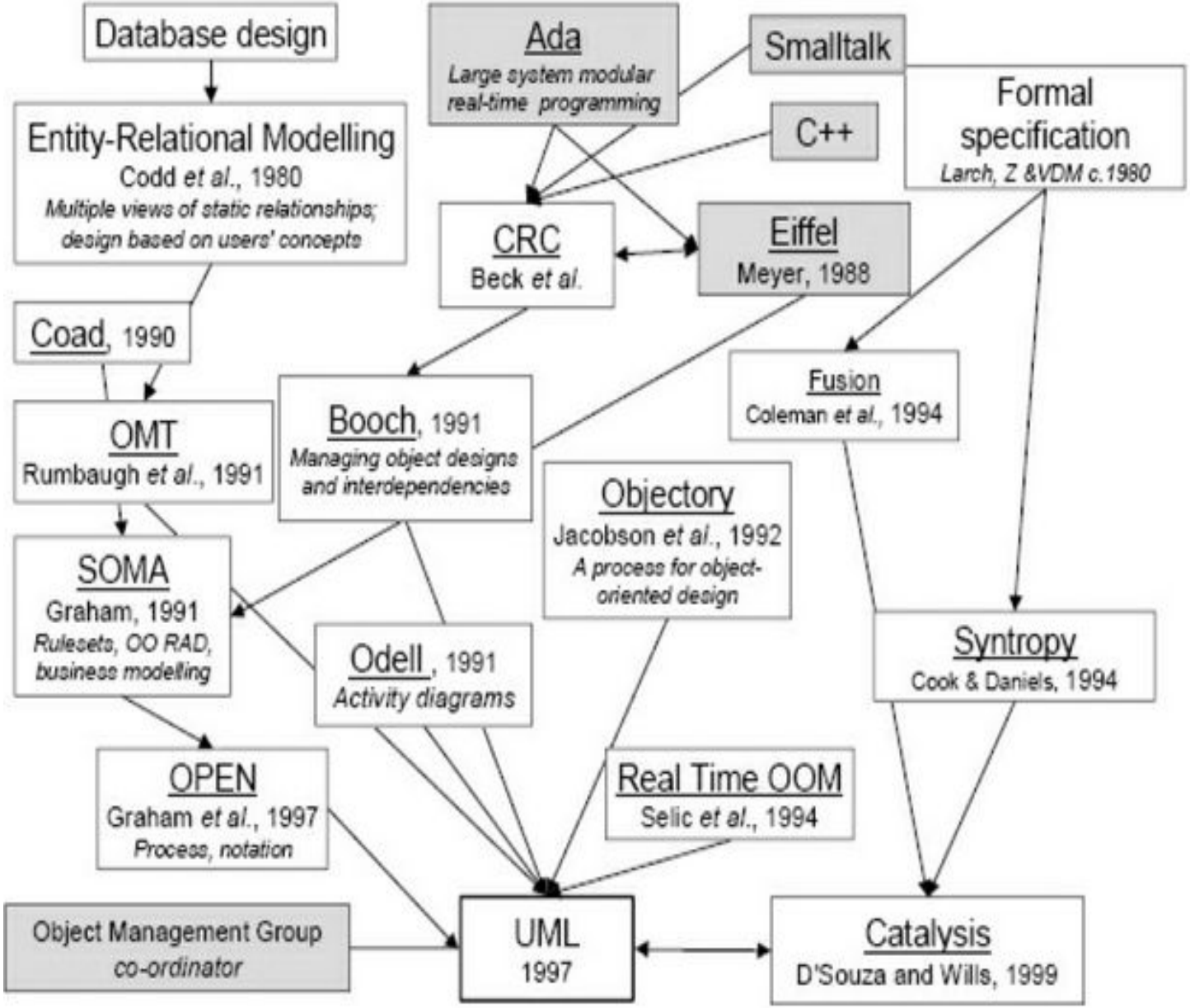
- внес достаточно радикальные изменения в сами принципы создания и функционирования программ
- позволил существенно повысить производительность труда программистов
- по-иному взглянуть на проблемы и методы их решения
- сделать программы более компактными и легко расширяемыми.

Как результат, языки, первоначально ориентированные на традиционный подход к программированию, получили ряд объектноориентированных расширений. Одной из первых, в середине 80-х, была фирма Apple со своим проектом Object Pascal. Кроме этого, объектно-ориентированный подход породил мощную волну и абсолютно новых программных технологий, вершинами которой стали такие общепризнанные сегодня платформы, как Microsoft .NET Framework и Sun Java.

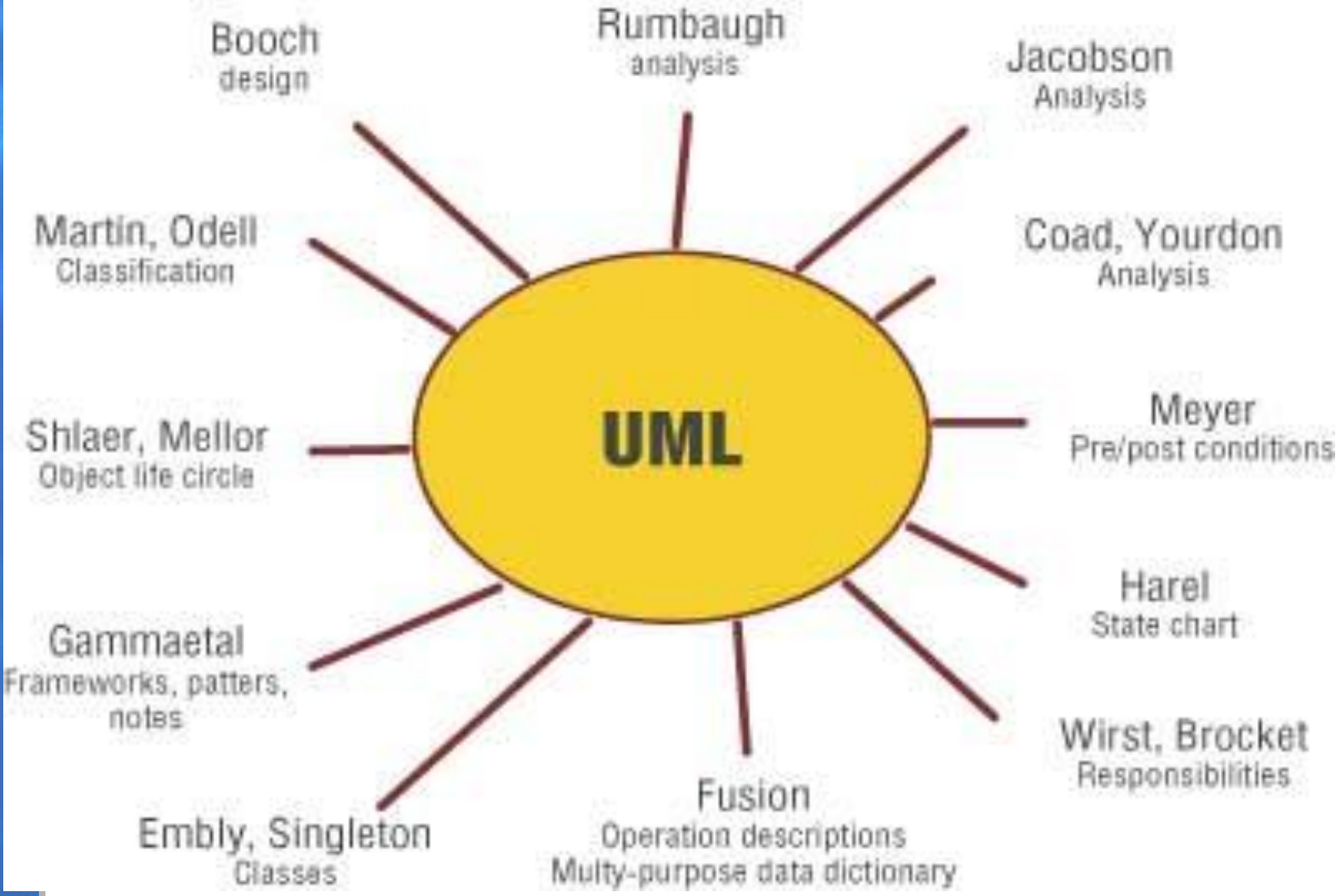
Но самое главное, что появление ООП требовало удобного инструмента для моделирования, единой нотации для описания сложных программных систем.

- И вот "три амиго", три крупнейших специалиста, три автора наиболее популярных методов решили объединить свои разработки. В 1991-м каждый из "трех амиго" начал с написания книги, в которой изложил свой метод ООАП. Каждая методология была по-своему хороша, но каждая имела и недостатки.
- Метод Буча был хорош в **проектировании**, но слабоват в анализе.
- OMT Рамбо был отличным средством **анализа**, но плох в проектировании.
- Objectory Якобсона был действительно хорош с точки зрения user experience, на который ни метод Буча, ни OMT не обращали особого внимания. Основной идеей Objectory было то, что анализ должен начинаться с **прецедентов**, а не с диаграммы классов, которые должны быть производными от них.

UML вобрал в себя черты нотаций многих языков и методов



UML вобрал в себя черты нотаций многих методов



- К 1994-му существовало 72 метода, или частные методики. Многие из них "перекрывались", т. е. использовали похожие идеи, нотации и т. д.
- Чувствовалась острая потребность, "социальный заказ" - закончить "войну методов" и объединить в одном унифицированном средстве все лучшее, что было создано в области моделирования.
- А что сейчас? The UML живет и развивается. Сейчас мы имеем UML 2.5 и десятки CASE-средств, поддерживающих UML
- Вопреки популярному мнению, в наши дни Rational не владеет UML, но продолжает работать над ним. UML же принадлежит OMG, а сама Rational ныне является одним из подразделений IBM и фигурирует во всех документах как IBM Rational.
- UML получил множество пакетов расширений, называемых *профайлами* и позволяющих использовать его для моделирования систем из специфических предметных областей.

- Подводя итоги, кратко можно сказать, что UML - искусственный язык, который имеет некоторые черты естественного языка, и формальный язык, который имеет черты неформального. Это звучит не очень понятно, но это действительно так!

Все по полочкам

Методология	Нотация или язык	Стандарт	Case-инструмент
SADT	IDEF0	IDEF0	IDEF Doctor
IDEF3	IDEF3	IDEF3	MS Visio
DFD	Гейна-Сарсона Йордана-ДеМарко	DFD	BPWin (AllFusion)
IDEF1X	IDEF1X	IDEF1X	ERWin (AllFusion)
ARIS	VAD eEPC/PCD	ARIS	ARIS Toolset MS Visio
OOM	UML	UML 2.0	Rational Rose Magic Draw MS Visio Net Beans Enterprise Architect StarUML

Причины неудачности проектов

- Недостаточно адекватное управление требованиями
- Несогласованность требований, проектных решений и реализации
- Жесткая архитектура ПО
- Нарастающая сложность ПО
- Неточная и противоречивая коммуникация
- Недостаточное тестирование
- Субъективное отношение к приоритетам отдельных артефактов проекта
- Игнорирование рисков и отсутствие процедур управления рисками
- Бесконтрольное внесение изменений в артефакты проекта
- Недостаточное использование CASE-средств и средств поддержки отдельных этапов проекта

Отсутствие моделей при разработке ПО

- Не позволяет справиться с растущей сложностью разрабатываемых программных систем
- Не позволяет эффективно управлять разработкой в условиях изменяющихся требований
- Создает барьеры непонимания: аналитик не понимает руководителя проекта, разработчик – аналитика, тестировщик – разработчика и пр.
- Не позволяет обеспечить контроль изменений в процессе выполнения работ
- Не позволяет избежать субъективности в оценке качества разрабатываемых продуктов

  - **Модель** (model) — абстракция физической системы, рассматриваемая с определенной точки зрения и представленная на некотором языке или в графической форме

Modeling

- **Model is description of the system.**

  It is abstract description of software, in which some aspect the system are hidden for n simple representation of other.

- **Characteristics of the model:**
  - Universality (knowledge domain)
  - Refinement (level of abstraction) (детализация)
  - Criteria of simplification (critical aspects) (упрощение)

Textual model

Graphic model

Uniformly accelerated motion:

Useful formulae:

$$v = u + at$$

$$s = ut + \frac{1}{2}at^2$$

$$v^2 = u^2 + 2as$$

$$s = \left(\frac{u+v}{2}\right)t$$

Mechanics:

Newton's second law: $F = \frac{d(mv)}{dt}$

Power: $P = Fv$

Momentum: $p = mv$

Circular motion:

Angular speed: $\omega = \frac{d\theta}{dt} = \frac{v}{r}$

Angular acceleration: $\alpha = \frac{d\omega}{dt} = \frac{a}{r}$

Centripetal acceleration: $a = \frac{v^2}{r}$

Torque: $\tau = I\alpha$

Work done in rotation: $T\theta = \frac{1}{2}I\omega^2$

Simple harmonic motion:

Displacement: $x = x_o \sin(\omega t + \phi)$

Velocity: $v = \omega x_o \cos(\omega t + \phi)$

Acceleration: $a = -\omega^2 x$

Period: $T = \frac{1}{f} = \frac{2\pi}{\omega}$

Mass on a light spring: $T = 2\pi\sqrt{\frac{m}{k}}$

Ray optics:

Refractive index: $_1n_2 = \frac{\sin\theta_1}{\sin\theta_2} = \frac{v_1}{v_2}$

$_1n_2 = {}_1n_3 \cdot {}_3n_2$

Thin lenses: $\frac{1}{f} = \frac{1}{u} + \frac{1}{v}$ (real is positive)

$\frac{1}{f} = \frac{1}{v} - \frac{1}{u}$ (Cartesian)

Magnification: $m = \frac{v}{u} = \frac{h_i}{h_o}$ (real is positive)

$m = -\frac{v}{u} = -\frac{h_i}{h_o}$ (Cartesian)

Current electricity:

Current: $I = nAve$

Resistors in series: $R_{TOTAL} = R_1 + R_2 + \ldots$

Resistors in parallel: $\frac{1}{R_{TOTAL}} = \frac{1}{R_1} + \frac{1}{R_2} + \ldots$

Power: $P = IV$

Resistivity: $\rho = \frac{R\,A}{l}$

Temperature coefficient: $\alpha = \frac{R_\theta - R_0}{R_0\theta}$

Alternating current:

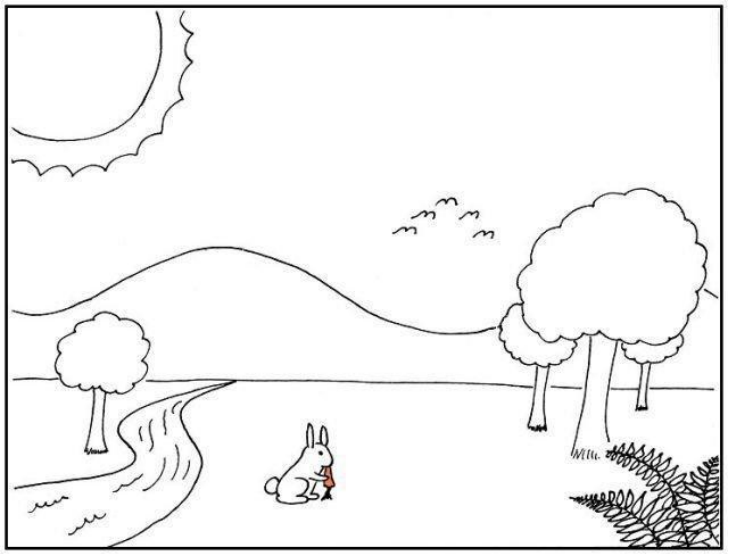
For sinusoidal alternating current:

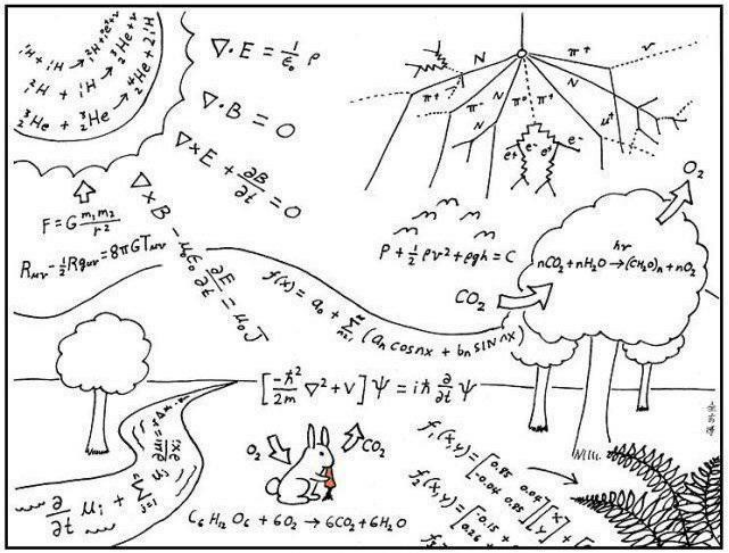
$$I = I_o \sin 2\pi ft$$

Root mean square for sinusoidal alternating current and voltage: $I_{rms} = \frac{I_0}{\sqrt{2}}$; $V_{rms} = \frac{V_0}{\sqrt{2}}$

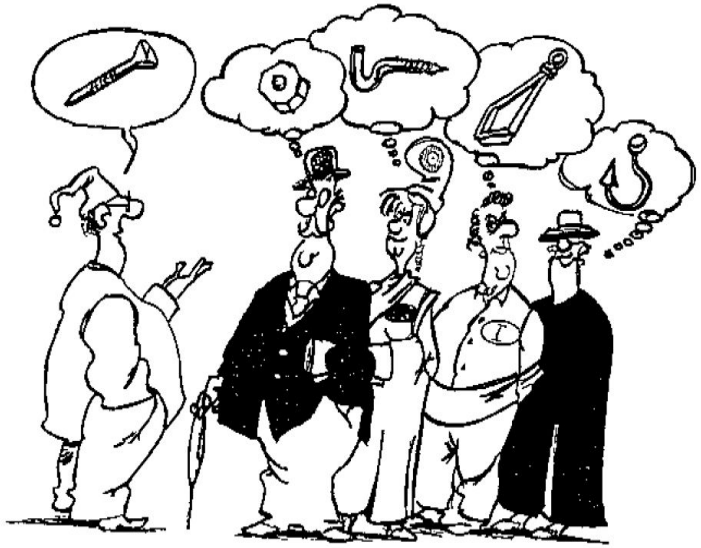
Ideal transformer: $\frac{N_P}{N_S} = \frac{V_P}{V_S}$

$I_P V_P = I_S V_S$





Зачем придумали диаграммы и правила их рисования?



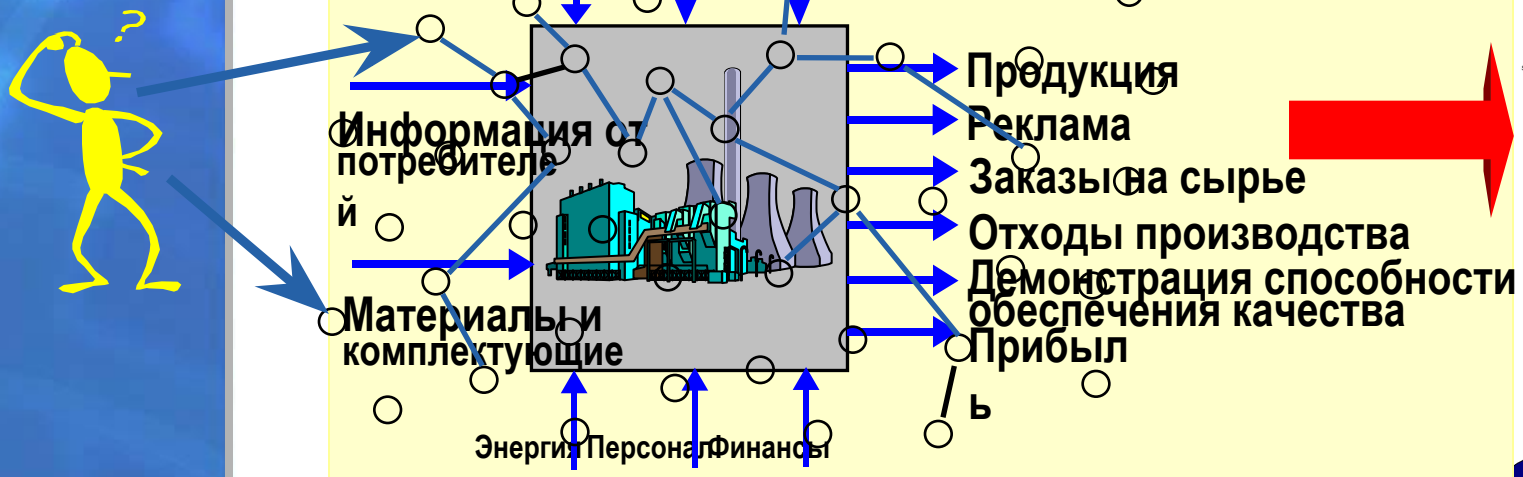
- **Облегчить восприятие текстового описания системы (особенности восприятия человека)**
- **Описать систему с помощью стандартных обозначений, исключающих неоднозначное толкование**

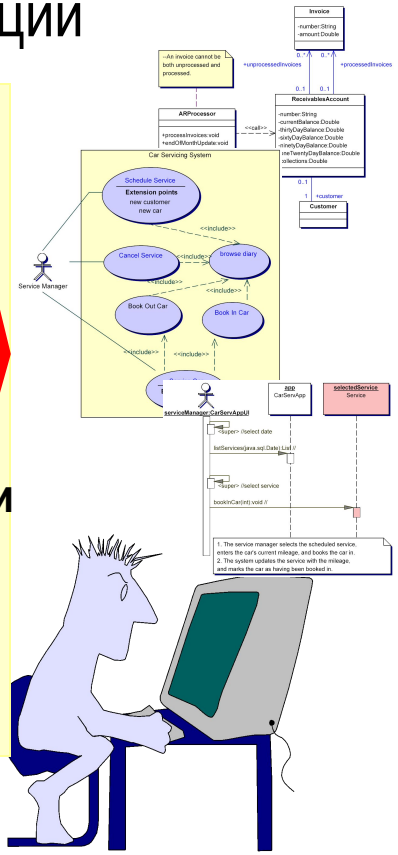
Лучшие практики разработки ПО

- Использование визуальных моделей при разработке ПО
- Итеративная разработка ПО
- Управление требованиями
- Управление изменениями и конфигурацией артефактов ПО
- Использование компонентных архитектур
- Непрерывное тестирование и верификация качества ПО
- Использование паттернов проектирования
- Использование CASE-средств и RAD-средств
- Управление рисками:
  - Технологическими рисками
  - Связанными с требованиями
  - Связанными с квалификацией персонала проекта
  - Политическими рисками

Что такое визуальное моделирование?

Визуальное моделирование есть моделирование с использованием некоторой графической нотации





На входе – Неструктурированная информация

На выходе – Модели ПО и бизнес-процессов

Основные понятия визуального моделирования

- **Нотация** – система условных обозначений для графического представления визуальных моделей
- **Семантика** – система правил и соглашений, определяющая смысл и интерпретацию конструкций некоторого языка
- **Методология** – совокупность принципов моделирования и подходов к логической организации методов и средств разработки моделей
- **CASE (Computer Aided Software Engineering) –** методология разработка программного обеспечения, основанная на комплексном использовании компьютеров не только для написания исходного кода, но и для анализа и моделирования соответствующей предметной области
- **CASE-средства** (CASE-tools) – программное обеспечение, которое предназначено для разработки визуальных моделей программных систем и генерации исходного кода или схемы базы данных на некотором языке

CASE-средства

Разработка визуальных моделей сложных систем, в виду значительного объема решаемых задач, должно опираться на специальные средства программной поддержки

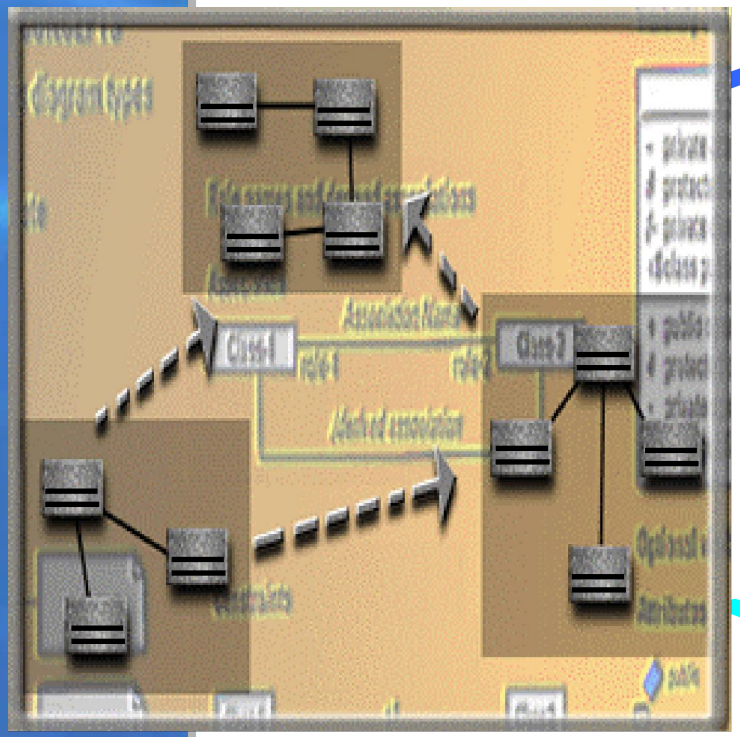
Oracle Designer

BPwin, ERwin

Rational Rose

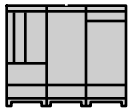
- 1-е поколение: генерация схем БД (Oracle Designer 2000, ERwin)
- 2-е поколение: генерация программного кода (Borland Together Designer 2005)
- 3-е поколение: прямая и обратная кодогенерация (IBM Rational Rose 2002/2003, Borland Together Developer 2005, Sparx Enterprise Architect)
- 4-е поколение: синхронизация программного кода и моделей (IBM Rational Software Architect 6/7, Borland Together Architect 2006 и 2008, Borland Development Studio 2006)

Визуальные модели представляют архитектуру программных систем



Бизнес-логика
(C++, Java)

Интерфейсы
пользователя
(Delphi,
Visual Basic,
Java)



Базы данных
(SQL)

Визуальная модель системы не должна зависеть от языка ее реализации!

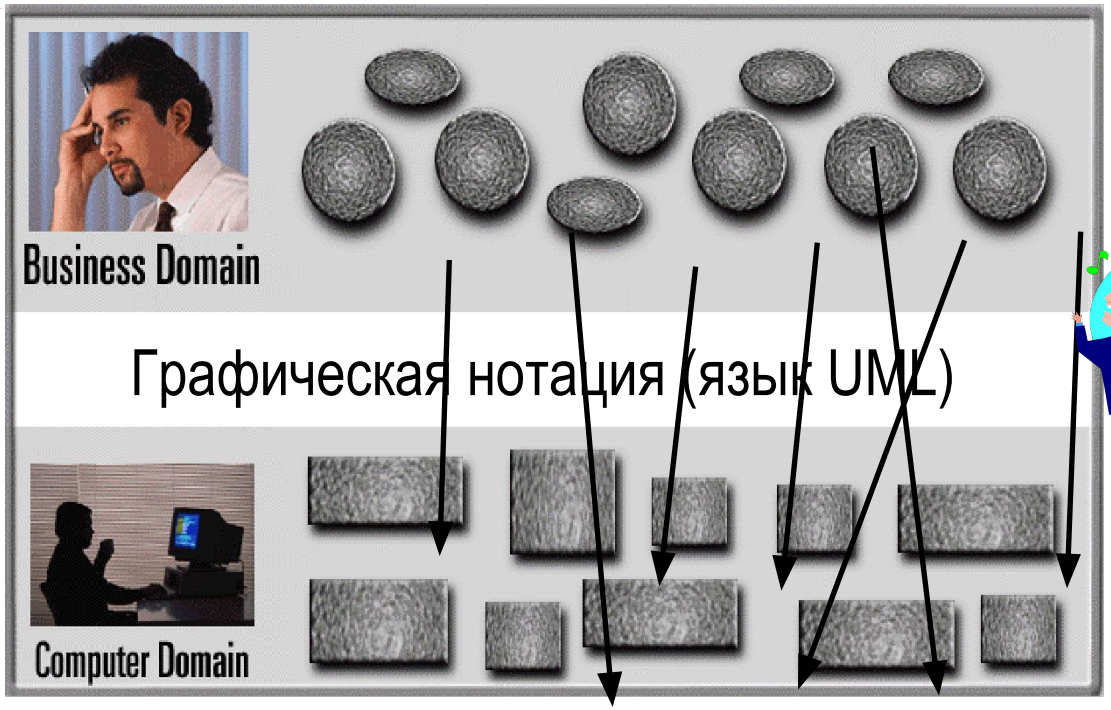
Визуальные модели являются средством коммуникации

Артефакты БП

Бизнес-аналитики, системные аналитики, архитекторы, CIO, MIS, CPO

Визуальные модели описывают бизнес-процессы

Визуальные модели используются для проектирования и разработки программных систем

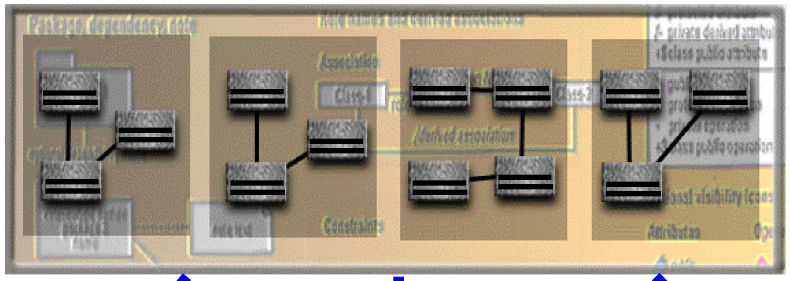


Артефакты ПО

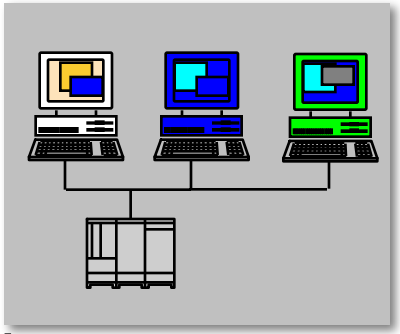
Программисты, тестировщики, менеджеры проектов

Визуальные модели – основа многократного использования кода

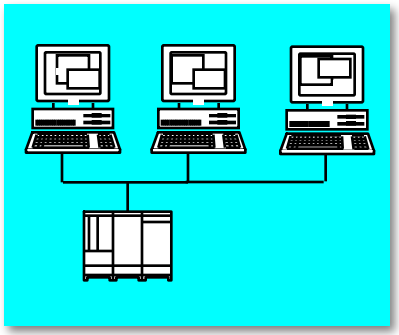
Моделирование охватывает существенные (основные, релевантные) аспекты структуры и поведения системы



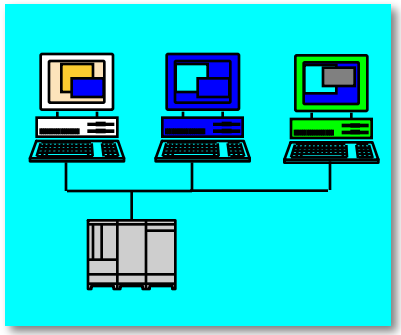
Многократно используемые компоненты (Reusable Components)



Интернет порталы



ERP Системы



Базы данных

ООП – основные понятия

- ***Объектно-ориентированное программирование*** (Object-Oriented Programming) — совокупность принципов, технологии и инструментальных средств для создания программных систем, в основу которых закладывается архитектура взаимодействия объектов
- ***Абстракция*** — характеристика сущности, которая отличает ее от других сущностей
- ***Наследование*** — принцип, в соответствии с которым знание о более общей категории разрешается применять для более частной категории
- ***Инкапсуляция*** — сокрытие отдельных деталей внутреннего устройства классов от внешних по отношению к нему объектов или пользователей
- ***Полиморфизм*** — свойство элементов модели с одинаковыми именами иметь различное поведение

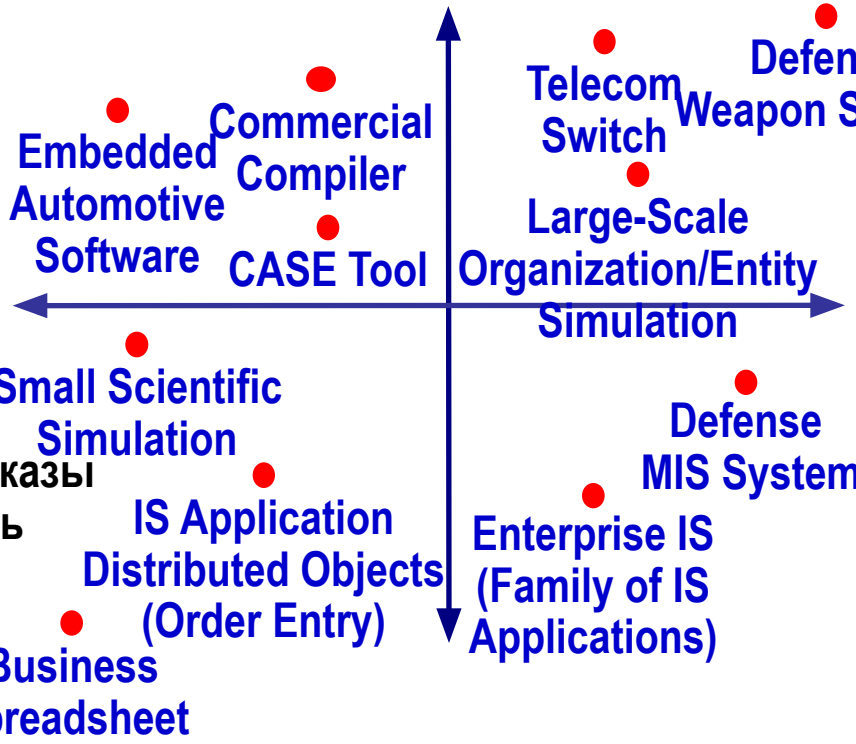
ООАП – основные понятия

- ***Объектно-ориентированный анализ и проектирование*** (Object-Oriented Analysis/Design) — технология разработки программных систем, в основу которых положена объектно-ориентированная методология представления предметной области в виде объектов, являющихся экземплярами соответствующих классов
- ***Предметная область*** (domain) – часть реального мира, которая имеет существенное значение или непосредственное отношение к процессу функционирования программы
- ***Диаграмма*** (diagram) — графическое представление совокупности элементов модели в форме связного графа, вершинам и ребрам (дугам) которого приписывается определенная семантика
- Нотация канонических диаграмм является основным средством разработки моделей на языке UML

Классификация проектов по сложности

Высокая техническая сложность
- **Встроенные системы реального времени**
- **Распределенные высоконадежные системы**
- **Высокопроизводительные системы**

Использование языка UML обязательно!



Низкая сложность управления
- **Малый масштаб**
- **Неформальные заказы**
- **Один пользователь**
- **“Продукты”**

Высокая сложность управления
- **Большой масштаб**
- **Контрактные заказы**
- **Много пользователей**
- **«Проекты»**

Использование языка UML не обязательно

Низкая техническая сложность
- **Использование макроязыков или 4GL**
- **Реинжиниринг приложений баз данных**
- **Разработка учетно-расчетных приложений**

Классификация проектов по типу приложений

Использование языка UML обязательно!

Использование языка UML в проектах по отраслевой принадлежности

- **Банки и инвестиционные фонды**
- **Связь и телекоммуникации**
- **Нефтегазовая промышленность**
- Страховые фонды
- Энергетика
- Машиностроение
- Торговля
- Фармацевтическая промышленность
- Оборонная промышленность
- Федеральная таможенная служба
- Учебные заведения

Средний проект по разработке ПО:

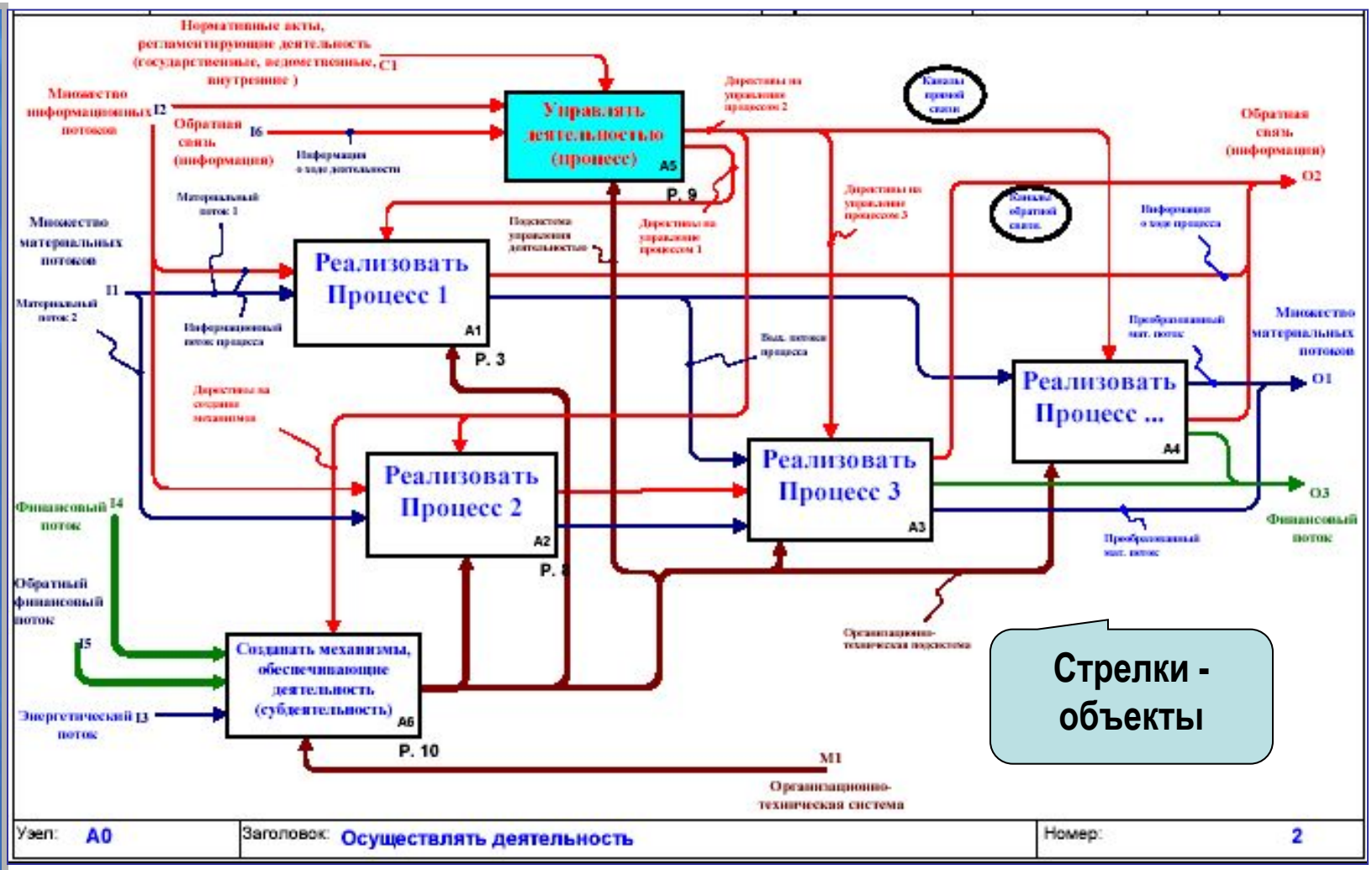
- 5-10 человек
- 10-15 месяцев
- 10-15 внешних интерфейсов
- Незначительная неопределенность и риски

Взаимосвязь нотации, методологии и инструментальных средств

Графические нотации моделирования, используемые в России

- **UML** (Unified Modeling Language) – отраслевой стандарт OMG, поддерживают более 50 CASE-средств, основной инструмент IBM Rational Rose/ IBM RSA (IBM Rational Software Architect)
- **IDEF** – семейство нотаций, стандарт МО США, рекомендован Правительством РФ для применения в государственных учреждениях, основной инструмент AllFusion Pricess Modeller (Computer Associations)
- **ARIS** (*ARchitecture of Integrated Information Systems*) – методология и нотация для профессионального моделирования бизнес-процессов, инструмент ARIS Toolset (IDS Scheer AG)

Пример визуальной модели в нотации IDEF



IDEF не объектно-ориентированная нотация!

Взаимосвязь нотации UML, методологии и инструментальных средств

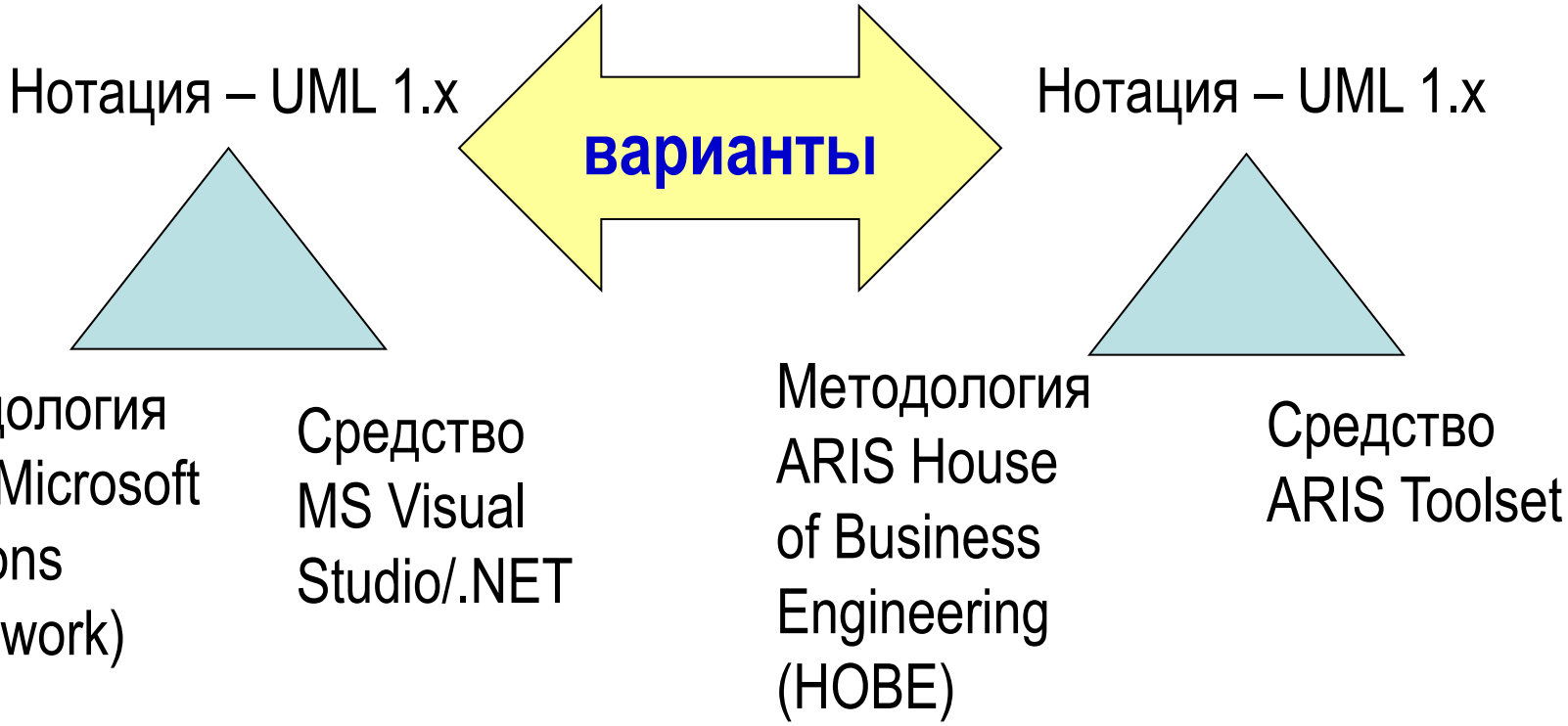
Нотация – UML 1.x

Методология - RUP

Средство – IBM Rational Rose

+ дополнительная интеграция с линейкой продуктов IBM Rational

Взаимосвязь нотации UML, методологии и инструментальных средств



Взаимосвязь нотации UML, методологии и инструментальных средств

Нотация – UML 2.x

Нотация - UML 2.x

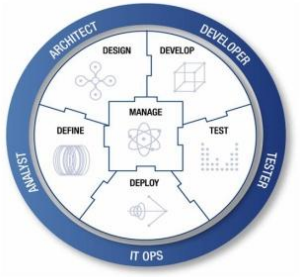
варианты

Методология RUP

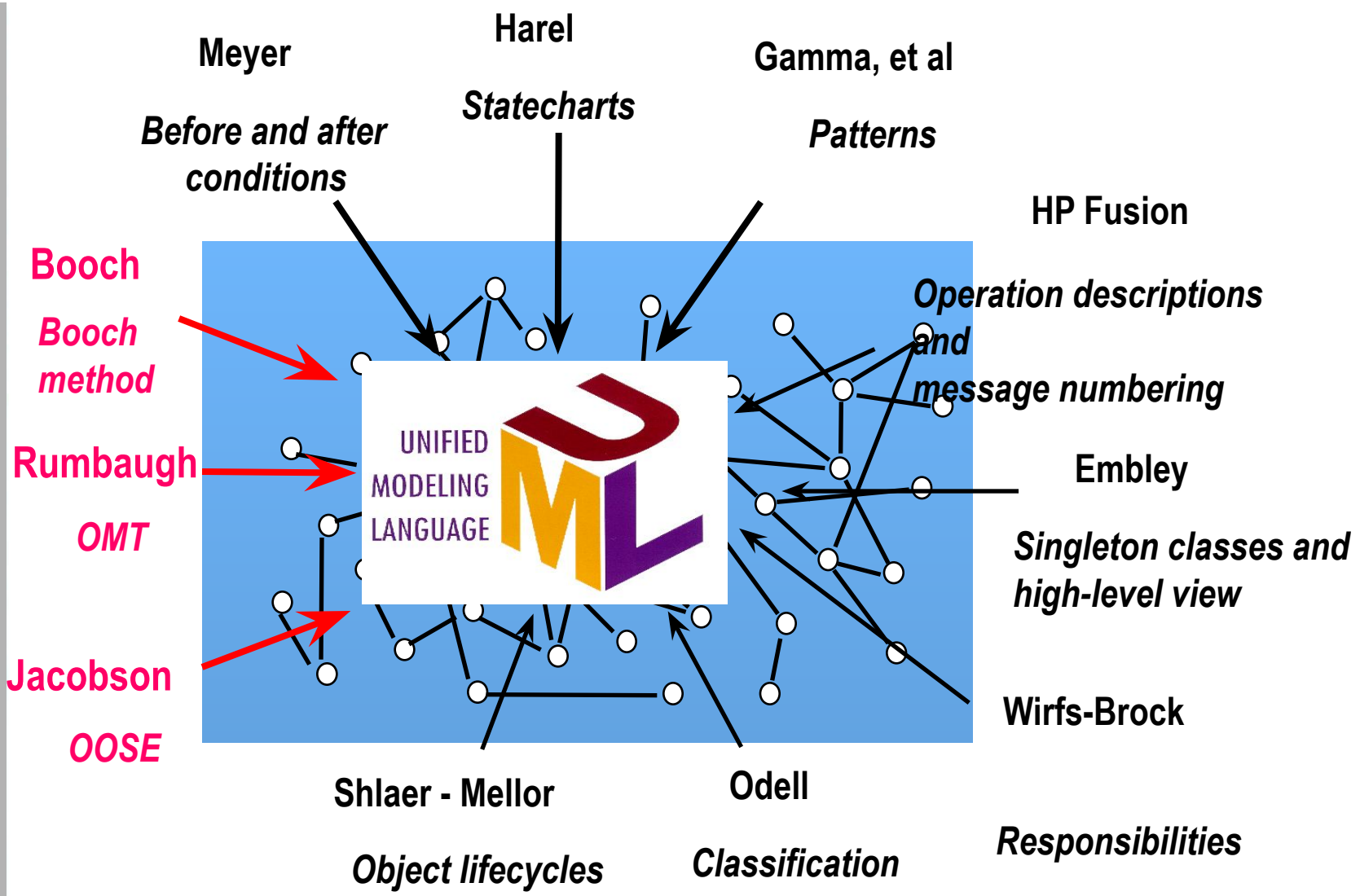
Средство IBM Rational Software Architect

Методология ALM (Application Lifecycle Management)

Средство Borland Together Architect 2006



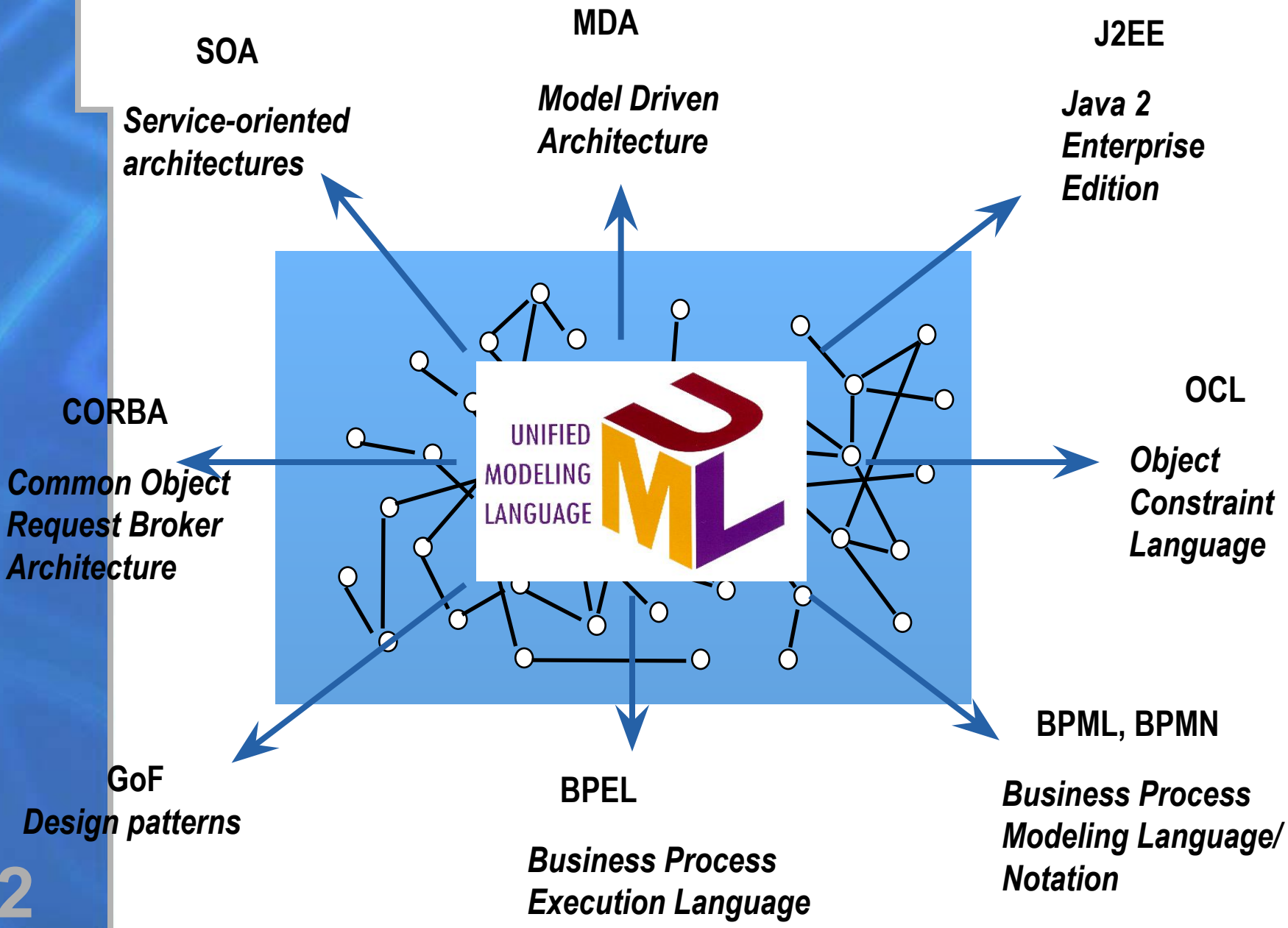
«Война методов» конца 1980 гг.



Популярные графические нотации визуального моделирования (конец 80-х гг.)

- **ERD** (Entity-Relationship Diagrams) – диаграммы «сущность-связь»
- **DFD** (Data Flow Diagrams) – диаграммы потоков данных, обеспечивающих анализ требований и функциональное проектирование информационных систем
- **STD** (State Transition Diagram) – диаграммы перехода состояний для проектирования систем реального времени (для моделирования конечных автоматов)
- **SADT** (Structured Analysis and Design Technique) – технология структурного анализа и проектирования
- **ICAM** (Integrated Computer Aided Manufacturing) – интегрированное компьютерное производство
- **FDD** (Functional Decomposition Diagrams) – диаграммы функциональной декомпозиции
- **Структурные карты** Джексона и Константайна – проектирование межмодульных взаимодействий и внутренней структуры объектов

Язык UML и современные технологии



Основные разработчики языка UML (Three amigos)

Grady Booch
Гради Буч

Dr. James Rumbaugh
Джеймс Рамбо
(Джим Румбах)

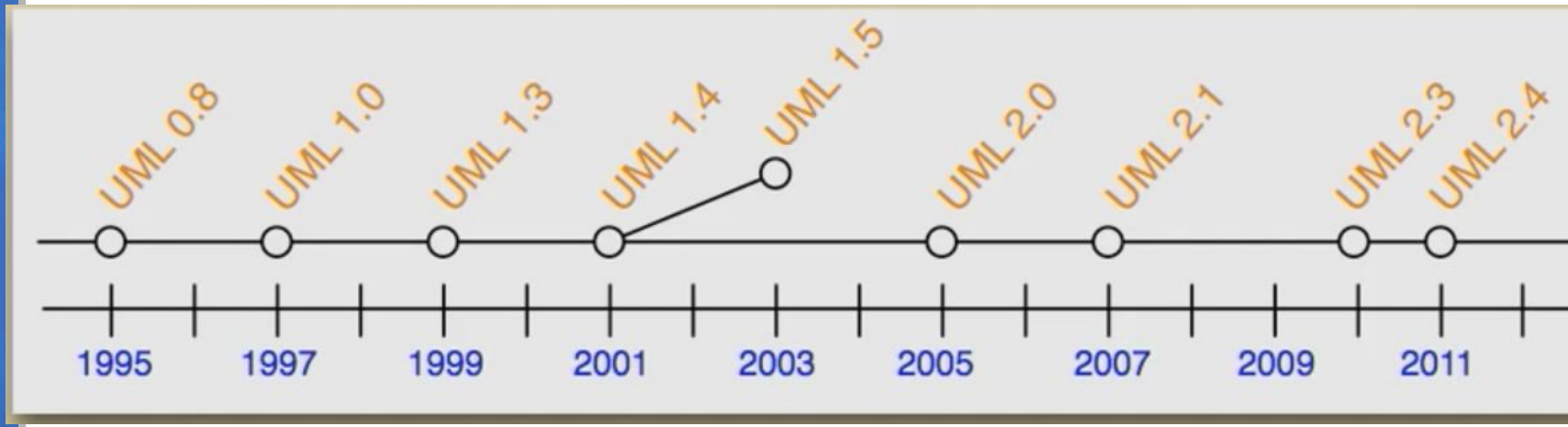
Dr. Ivar Jacobson
Айвар Джекобсон
(Ивар Якобсон)

- **OMG** (Object Management Group) — название консорциума, созданного в 1989 году для разработки индустриальных стандартов с их последующим использованием в процессе создания масштабируемых неоднородных распределенных объектных сред.
- В настоящее время входит более 800 софтверных компаний
- Официальный сайт: www.omg.org

Rational
the software development company

История развития языка UML

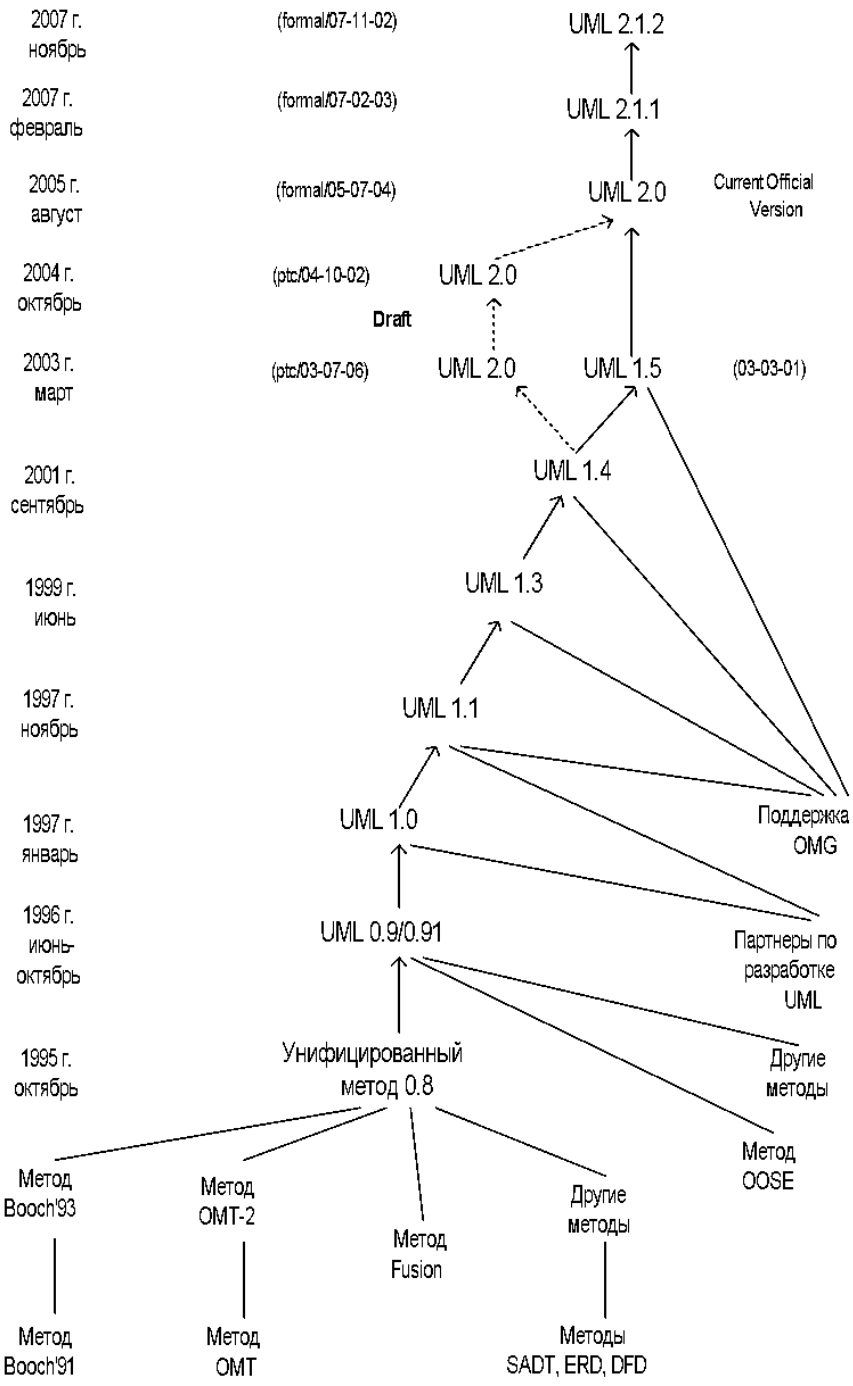
- Если проследить историю возникновения и развития элементов UML, как на уровне основополагающих идей и технических деталей, то пришлось бы назвать сотни имен и десятки организаций. Но язык постоянно совершенствуется, обогащается и расширяется
- Линейная упорядоченность версий на рисунке имеет одно отклонение. На особом положении оказалась версия 1.5. Она появилась, когда ожидалось появление версии 2.0. UML 1.5 содержит набор элементов, позволяющий применять UML не только, как язык моделирования, но и как язык программирования. Генеральная линия развития инструментальных средств прошла мимо этого явления. Крупные поставщики инструментов предпочли заявить о поддержке версии 2.0, не имея для этого достаточно оснований, и оставили без поддержки версию 1.5.



История развития языка UML

Спецификация языка UML 2.1.2:

- Суперструктура: 07-11-02.pdf – 736 стр.
- Инфраструктура: 07-02-04.pdf – 218 стр.
- Object Constrain Language v.2.0: 2005-06-06.pdf – 185 стр.
- Diagram Interchange: 03-07-03.pdf – 34 стр.
- Model Driven Architecture 03-06-01.pdf – 62 стр.



Основные разработчики языка UML 2

- Don Baisley
- Morgan Bjorkander
- Conrad Bock
- Steve Cook
- Philippe Desfray
- Nathan Dykman
- Anders Ek
- David Frankel
- Eran Gery
- Oystein Haugen
- Sridhar Iyengar
- Cris Kobryn
- Birger Moller-Pedersen
- James Odell
- Gunnar Overgaard
- Karin Palmkvist
- Guus Ramackers
- **Jim Rumbaugh**
- Bran Selic
- Thomas Weigert
- Larry Williams

Определение языка UML

***Unified Modeling Language* —** унифицированный язык моделирования для описания, визуализации и документирования объектно-ориентированных систем в процессе их анализа и проектирования

Язык UML предоставляет стандартный способ написания проектной документации на системы, включая концептуальные аспекты, такие как бизнес процессы и функции системы, а также конкретные аспекты, такие как выражения языков программирования, схемы баз данных и повторно используемые компоненты ПО

- Язык UML не является методологией
- Язык UML не является процессом
- Язык UML не является языком программирования
- Язык UML не является формальным языком

UML = нотация + семантика !

Определения языка UML 1.4 и 2.0

UML 1.4 — графический язык моделирования общего назначения предназначенный для спецификации, визуализации, проектирования и документирования всех артефактов, создаваемых при разработке **программных** систем

Г.Буч

UML 2.0 — графический язык моделирования общего назначения предназначенный для спецификации, визуализации, проектирования и документирования всех артефактов, создаваемых при разработке систем

Г.Буч

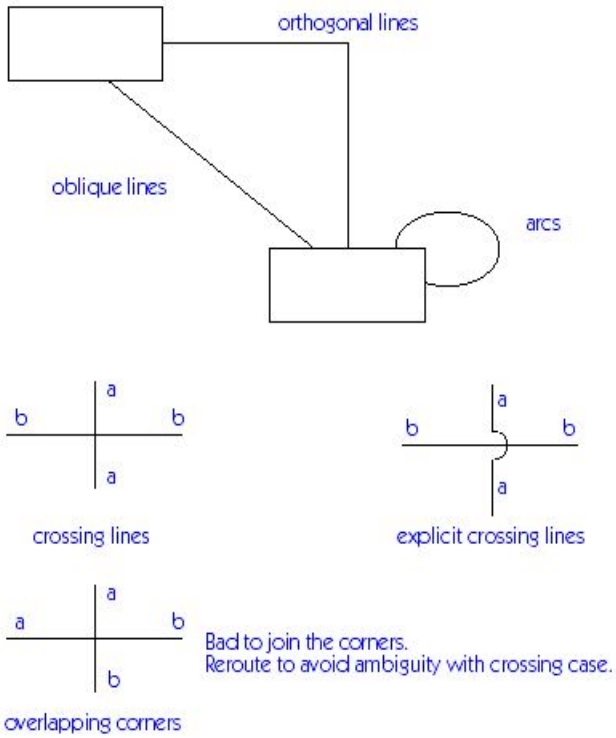
Способы использования UML

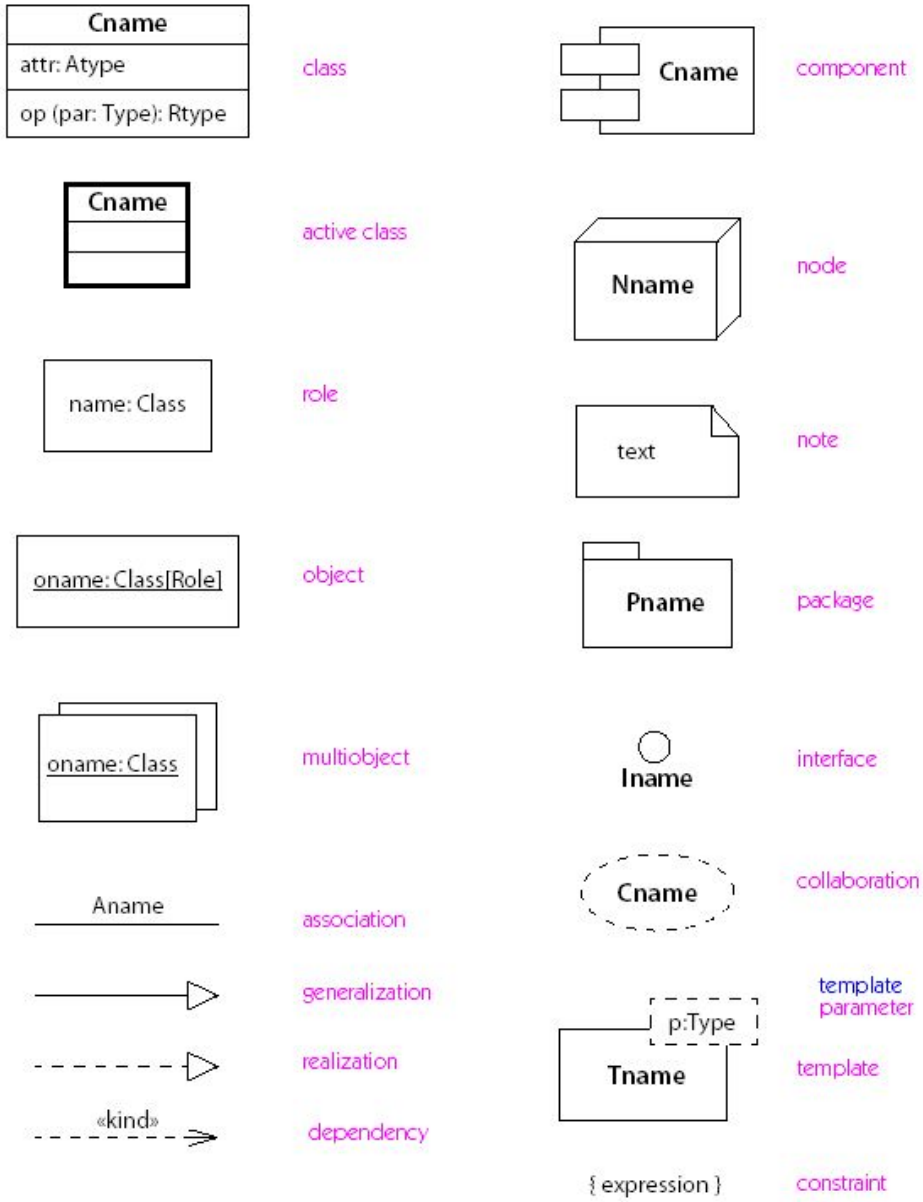
- Рисование картинок
- Обмен информацией
- Спецификация систем
- Повторное использование архитектурных решений
- Генерация кода
- Имитационное моделирование
- Верификация моделей

Назначение языка UML

- Предоставить разработчикам легко воспринимаемый и выразительный язык визуального моделирования, специально предназначенный для разработки и документирования моделей сложных систем различного целевого назначения
- Снабдить исходные понятия языка UML возможностью расширения и специализации для более точного представления моделей систем в конкретной предметной области
- Графическое представление моделей в нотации UML не должно зависеть от конкретных языков программирования и инструментальных средств проектирования
- Описание языка UML должно включать в себя семантический базис для понимания общих особенностей ООАП
- Способствовать распространению объектных технологий и поощрять развитие рынка программных инструментальных средств
- Интегрировать в себя новейшие и наилучшие достижения практики ООАП

Особенности изображения графического элементов диаграмм языка UML





Особенности изображения диаграмм в нотации UML

- **Графические узлы** на плоскости, которые изображаются с помощью геометрических фигур и могут иметь различную высоту и ширину с целью размещения внутри этих фигур других конструкций языка UML
- **Пути**, которые представляют собой последовательности из отрезков линий, соединяющих отдельные графические узлы
- **Значки или пиктограммы**. Значок представляет собой графическую фигуру фиксированного размера и формы, которая не может увеличивать свои размеры, чтобы разместить внутри себя дополнительные символы.
- **Строки текста**. Служат для представления различных видов информации в некоторой грамматической форме.

Общие рекомендации по изображению диаграмм в нотации языка UML

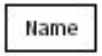
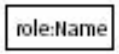
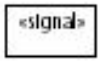
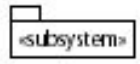
- Каждая диаграмма должна служить законченным представлением соответствующего фрагмента моделируемой предметной области
- Все сущности на диаграмме модели должны быть одного концептуального уровня
- Вся информация о сущностях должна быть явно представлена на диаграммах
- Диаграммы не должны содержать противоречивой информации
- Диаграммы не следует перегружать текстовой информацией
- Каждая диаграмма должна быть само достаточной для правильной интерпретации всех ее элементов и понимания семантики всех используемых графических символов

Противоречивость и адекватность моделей в нотации UML

- Модель, соответствующая правилам нотации или семантики языка UML называется *непротиворечивой* (well-formed model)
- Модель, нарушающая правила нотации или семантики языка UML называется *противоречивой* (ill-formed model)
- Здесь могут быть использованы формальные критерии – соответствие спецификации языка UML!
- Модель, достаточно полно и правильно отражающая предметную область или решаемую проблему называется *адекватной*
- Модель, не достаточно полно или неправильно отражающая предметную область или решаемую проблему называется *не адекватной*
- Здесь могут быть использованы только неформальные критерии – субъективное мнение экспертов!
- *«Моя модель – это не ваша модель, а ваша модель – не моя…» Мартин Фаулер*

Классификаторы – основные элементы языка UML

Прямоугольник – основной символ для графического изображения классификатора

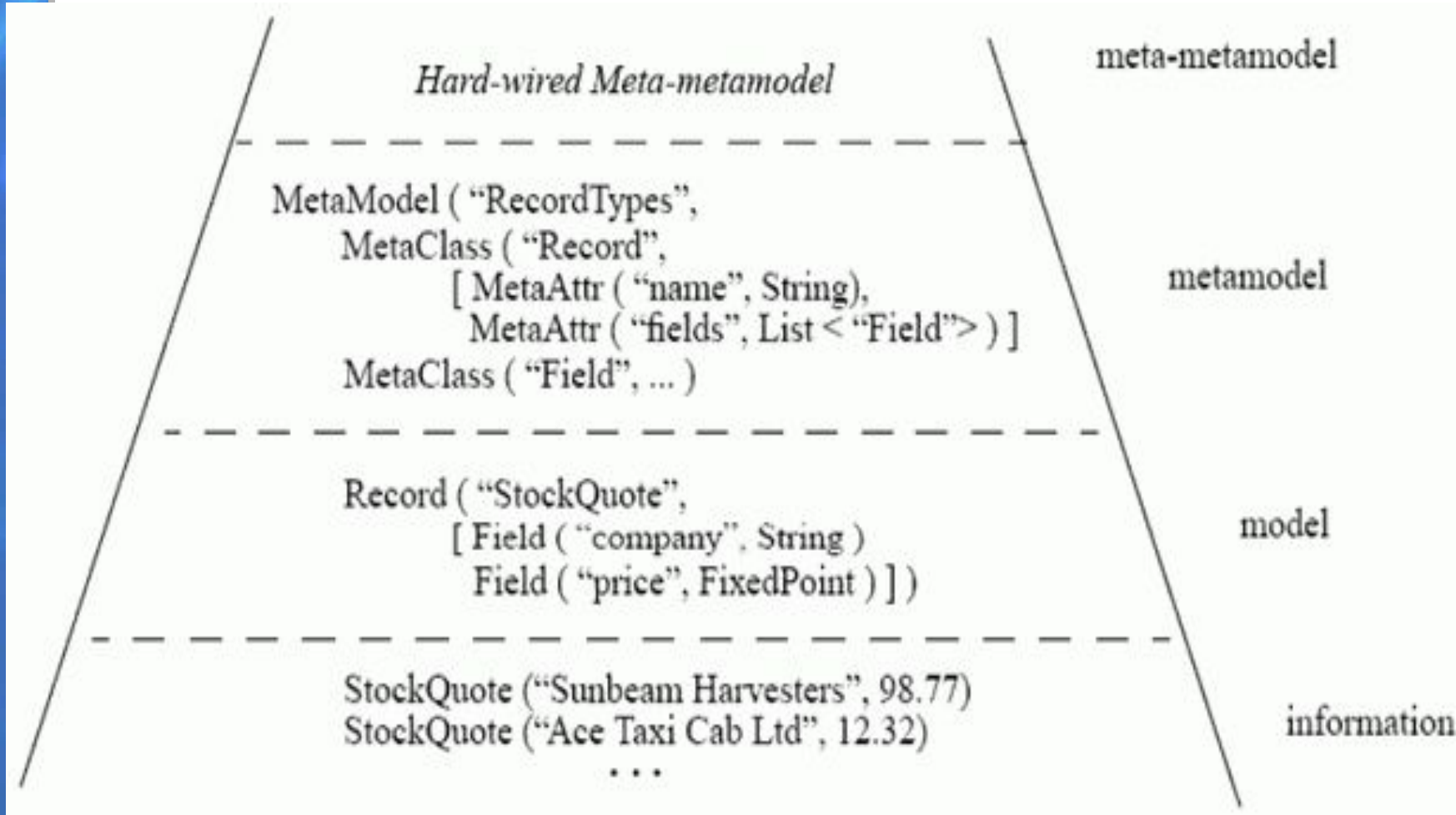
Classifier	*Function*	*Notation*
actor	An outside user of a system	
class	A concept from the modeled system	
class-in-state	A class restricted to being in a given state	
classifier role	A classifier restricted to a particular usage in a collaboration	
component	A physical piece of a system	
data type	A descriptor of a set of primitive values that lack identity	Name
interface	A named set of operations that characterize behavior	
node	A computational resource	
signal	An asynchronous communication among objects	
subsystem	A package that is treated as a unit with a specification, implementation, and identity	
use case	A specification of the behavior of an entity in its interaction with outside agents	

Структура определения языка

- Авторы использовали так называемое четырехуровневое мета-моделирование.
- Первый уровень - это сами данные.
- Второй - это их модель, т. е., например, описание их в программе.
- Третий - метамодель, т. е. описание языка построения модели.
- Четвертый - мета-метамодель, т. е. описание языка, на котором описана метамодель.

Структура определения языка

Пример применения подхода к простым записям о котировках акций (из стандарта UML)



Структура определения языка

- МЕТАМОДЕЛЬ - описание самого языка,
- МЕТА-МЕТАМОДЕЛЬ - описание формализма, с помощью которого производится описание языка.

Все это сопровождается комментариями на естественном языке и примерами моделей. Организованное таким образом описание UML распространяется OMG абсолютно свободно и "лежит" на сайте OMG, по адресу http://www.omg.org/. Этот грандиозный документ насчитывает около тысячи страниц, и неподготовленному читателю имеет смысл ознакомиться в нем лишь с первым и последним разделами (краткий обзор и словарь терминов). Зато, если человек уже знаком с UML, изучение метамодели языка - весьма интересное и полезное занятие.

Терминология

- Вопрос терминологии в программной инженерии, а тем более РУССКОЙ - вопрос сложный. Дело в том, что оригинальная терминология UML не всегда последовательна и довольно запутана. Русская же терминология еще не успела сложиться, ведь UML как технология проектирования сама по себе очень молода, и русскоязычная литература по нему стала появляться, как всегда, с некоторым опозданием. Некоторые авторы пытаются каждый термин передать "осмысленными", "хорошими русскими словами", что не всегда удается. Искать русские аналоги уже привычных английских терминов - занятие ненужное и даже вредное. Поэтому, наверное, проще использовать транскрипцию и не изобретать велосипед!

Нотация

- "Нотация" - это то, что в других языках называют "синтаксисом". Само слово "нотация" подчеркивает, что UML - язык графический и модели (а точнее диаграммы) не "записывают", а рисуют.
- Одна из задач UML - служить средством коммуникации внутри команды и при общении с заказчиком. "В рабочем порядке" диаграммы часто рисуют на бумаге от руки, причем обычно - не слишком аккуратно. Поэтому при выборе элементов нотации основным принципом был отбор значков, которые хорошо смотрелись бы и были бы правильно интерпретированы в любом случае - будь они нарисованы карандашом на салфетке или созданы на компьютере и распечатаны на лазерном принтере.

Нотация

В UML используется четыре вида элементов нотации:

- фигуры,
- линии,
- значки,
- надписи.

Нотация. Фигуры.

- Фигуры используются "плоские" - прямоугольники, эллипсы, ромбы и т. д. Но есть одно исключение - на диаграмме развертывания для обозначения узлов инфраструктуры применяется "трехмерное" изображение параллелепипеда. Это единственное исключение из правил. Внутри любой фигуры могут помещаться другие элементы нотации.

Нотация. Линии.

- О линиях стоит сказать лишь то, что своими концами они должны соединяться с фигурами. На UML диаграммах вы не встретите линий, нарисованных "сами по себе" и не соединяющих фигуры. Применяется два типа линий - сплошная и пунктирная. Линии могут пересекаться, таких случаев следует по возможности избегать, но в этом нет ничего страшного.

Нотация

- UML предоставляет исключительную свободу - можно рисовать что угодно и как вздумается, лишь бы можно было понять смысл созданных диаграмм. В изображении фигур и значков тоже нет каких-то жестких требований, и разработчики CASE-средств для UML-проектирования вовсю используют эту свободу, применяя различные стили рисования, заливку фигур цветом, тени и т. д. Иногда это смотрится весьма симпатично, а иногда даже раздражает.

CASE-средства для построения диаграмм UML

- На данный момент на рынке присутствует огромное количество и полноценных средств UML-моделирования, и программ для рисования диаграмм, в том числе и UML.
- Такие продукты, как Borland Together, Poseidon, StarUML и Dia, могут быть загружены с сайта производителя абсолютно бесплатно.
- StarUML выглядит наиболее функциональным из бесплатных продуктов и может служить полноценной заменой коммерческим программам для UML-моделирования.
- Для использования в качестве справочника идеально подходит Zicom Mentor от Sparx Systems, который также может быть получен абсолютно бесплатно.
- Выбор средства UML-проектирования - вопрос сложный и неоднозначный, и решить его каждый должен для себя сам, исходя из своих потребностей, уровня знаний и т. д.

Наиболее заметные программные инструменты:

- **Sparx Systems Enterprise Architect;**
- **IBM Rational Rose**;
- **Borland Together**;
- Gentleware Poseidon;
- StarUML;
- Dia;
- Microsoft Visio;
- Telelogic TAU G2;
- ...

Наиболее известными из этой пятерки являются Rational Rose и Together.

Приведенные пакеты - очень малая часть всего доступного в Интернете ПО для визуального моделирования с помощью UML. Список другого ПО для создания UML-диаграмм можно найти, например, на http://www.objectsbydesign.com/tools/umltools_byCompany.html.

Sparx Systems Enterprise Architect

Как уверяют разработчики (Sparx Systems), Enterprise Architect - это программа для UML-моделирования и проектирования **нового поколения**. Вот фраза из их рекламных материалов:

WELCOME to the next generation in UML modeling and design software! At Sparx Systems, we realize that because you want to remain competitive, you need to be productive. You need to have your whole team perfectly equipped with the very latest trouble-free UML modeling software. In other words, you need the most reliable, capable and progressive business modeling software, that won't break the budget.

Enterprise Architect существует в вариантах для Windows и Linux и является неплохим средством для UML-моделирования, с возможностью многопользовательской работы и дружественным интерфейсом. В EA есть множество функций, распределенных между несколькими приложениями, включая отличные возможности по генерации документации, поддержку плагинов, генерацию XSD-схем, HTML и поддержку для таких языков программирования, как C++, Java, PHP, Visual Basic, VB.Net, Delphi или C#.

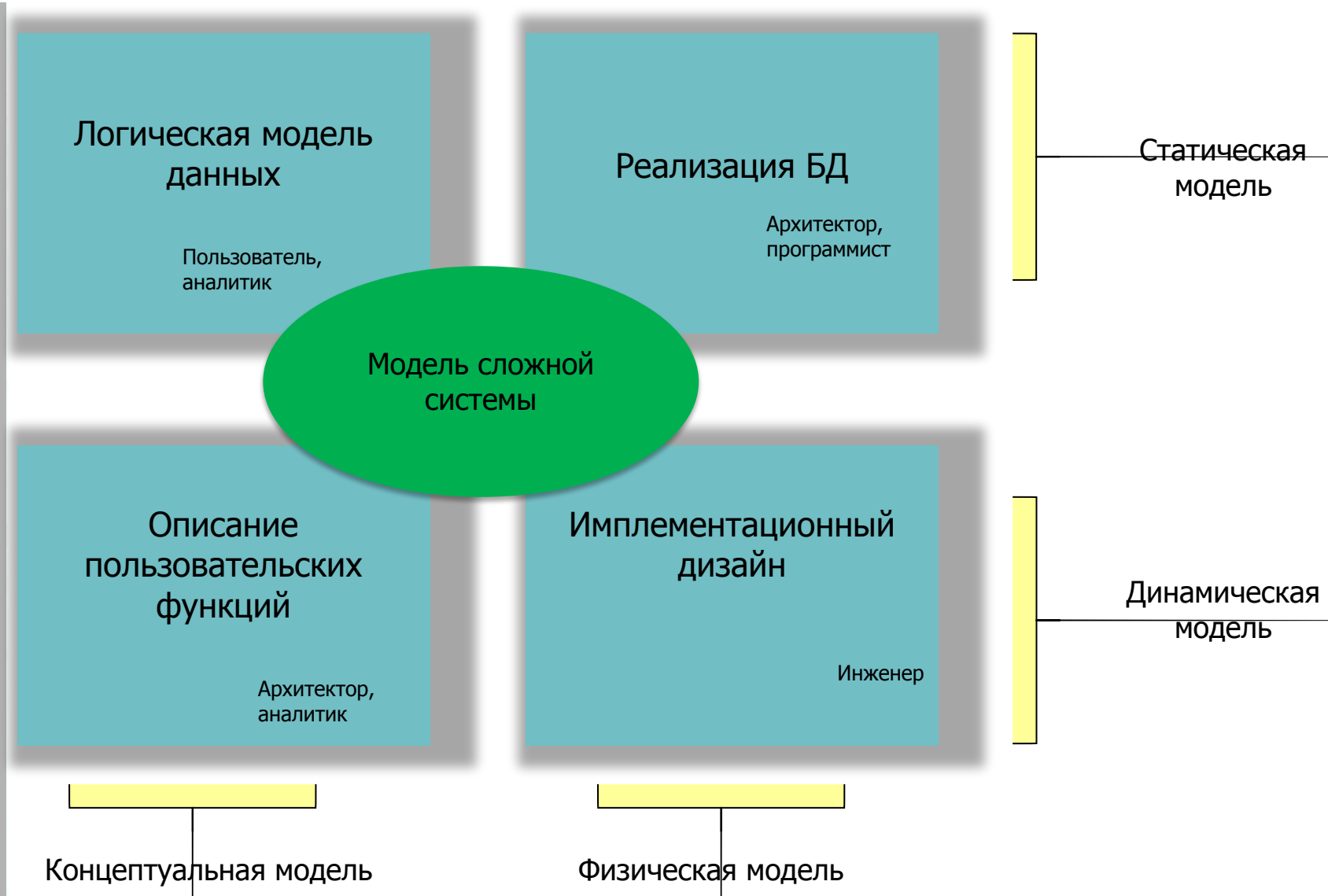
UML. Выводы.

- UML - еще один формальный язык, который необходимо освоить каждому, кто собирается заниматься программной инженерией.
- Само собой разумеется, что знание UML не гарантирует построения разумных и понятных моделей, хотя и является для этого необходимым.
- UML предоставляет огромную свободу при рисовании диаграмм и выборе инструмента рисования. Производители инструментов также воспользовались этой свободой, чтобы по своему разумению "украсить" имеющуюся нотацию.

Общая схема взаимосвязей моделей и представлений сложной системы в процессе ООАП

- С точки зрения методологии ООАП достаточно полная модель сложной системы представляет собой определенное число взаимосвязанных представлений (views), каждое из которых адекватно отражает аспект поведения или структуры системы. При этом наиболее общими представлениями сложной системы принято считать статическое и динамическое, которые в свою очередь могут подразделяться на другие более частные.
- Принцип иерархического построения моделей сложных систем предписывает рассматривать процесс построения моделей на разных уровнях абстрагирования или детализации в рамках фиксированных представлений.
- Уровень представления (layer) — способ организации и рассмотрения модели на одном уровне абстракции, который представляет горизонтальный срез архитектуры модели, в то время как разбиение представляет ее вертикальный срез.
- При этом исходная или первоначальная модель сложной системы имеет наиболее общее представление и относится к концептуальному уровню. Такая модель, получившая название концептуальной, строится на начальном этапе проектирования и может не содержать многих деталей и аспектов моделируемой системы. Последующие модели конкретизируют концептуальную модель, дополняя ее представлениями логического и физического уровня.
- В целом же процесс ООАП можно рассматривать как последовательный переход от разработки наиболее общих моделей и представлений концептуального уровня к более частным и детальным представлениям логического и физического уровня. При этом на каждом этапе ООАП данные модели последовательно дополняются все большим количеством деталей, что позволяет им более адекватно отражать различные аспекты конкретной реализации сложной системы.

Разные уровни моделей UML



Диаграммы UML

В рамках языка UML все представления о модели сложной системы фиксируются в виде специальных графических конструкций, получивших название диаграмм. Диаграмма (diagram) — графическое представление совокупности элементов модели в форме связного графа, вершинам и ребрам (дугам) которого приписывается определенная семантика. Нотация канонических диаграмм - основное средство разработки моделей на языке UML.

В нотации языка UML определены следующие виды канонических диаграмм:

- вариантов использования (use case diagram)
- классов (class diagram)
- кооперации (collaboration diagram)
- последовательности (sequence diagram)
- состояний (statechart diagram)
- деятельности (activity diagram)
- компонентов (component diagram)
- развертывания (deployment diagram)

Перечень этих диаграмм и их названия являются каноническими в том смысле, что представляют собой неотъемлемую часть графической нотации языка UML. Более того, процесс ООАП неразрывно связан с процессом построения этих диаграмм. При этом совокупность построенных таким образом диаграмм является самодостаточной в том смысле, что в них содержится вся информация, которая необходима для реализации проекта сложной системы.

Диаграммы UML

- Каждая из этих диаграмм детализирует и конкретизирует различные представления о модели сложной системы в терминах языка UML.
- При этом диаграмма вариантов использования (функционал, доступный пользователям системы) представляет собой наиболее общую концептуальную модель сложной системы, которая является исходной для построения всех остальных диаграмм.
- Диаграмма классов (логическая модель данных системы, сущностей, из которых будет строиться БД), по своей сути, логическая модель, отражающая статические аспекты структурного построения сложной системы.
- Диаграммы кооперации и последовательностей (описание одного сценария) представляют собой разновидности логической модели, которые отражают динамические аспекты функционирования сложной системы.
- Диаграммы состояний (отображение статусов объектов) и деятельности (блок-схема) предназначены для моделирования поведения системы.
- И, наконец, диаграммы компонентов и развертывания служат для представления физических компонентов сложной системы и поэтому относятся к ее физической модели.

Диаграммы UML

Наиболее часто используются диаграммы, помеченные голубым цветом

Диаграммы UML

- Каждая из диаграмм, использованных в UML, позволяет рассматривать бизнес-процессы под различным углом. К примеру, деловые пользователи при помощи данных диаграмм могут оценить основные положения бизнес-процесса и разобраться в том, кто за что отвечает. Разработчики же применяют диаграммы классов и объектов для получения точного представления о том, как встраивать данные компоненты в свой код.
- Диаграммы классов описывают статическое состояние элементов системы в каждый конкретный момент, показывают структуру объектов, их атрибуты и взаимные связи. Диаграммы деятельности отображают управляющие потоки, идущие от одного действия к другому, а диаграммы вариантов использования иллюстрируют элементы, находящиеся за пределами системы. (К примеру, внутренние операции новой платежной системы отображаются на диаграмме действий, тогда как работа внешних агентов, в частности отдела обработки заказов, представлена на диаграмме вариантов использования.) Диаграммы последовательности отражают интерактивные процессы: вы видите не только объекты и классы, но и сообщения, которыми они обмениваются. Таким образом, с помощью системы можно моделировать ситуации, применяя обычную в таких случаях технологию «что, если». Диаграммы состояния используются для описания динамических объектов.
- Но нужно четко понимать какую цель мы преследуем, когда хотим заняться описанием бизнес-процессов, и подбирать соответствующие средства для этого. Например, если цель сделать автоматизацию БП, в сфере где БП нам известны, то не факт, что имеет смысл делать полномасштабное описание бизнеса, а может только некоторые общие вещи или наоборот -- отдельные детали.

Последовательность разработки моделей

Модель	Назначение
Диаграммы вариантов использования	Описание функционального назначения системы
Диаграммы деятельности	Описание потоков работ в бизнес-процессах и\или внутри UC
Диаграммы последовательности	Описание поведения системы
Диаграммы классов	Описание предметной области
Диаграммы состояний	Описание событий, изменяющих состояния объектов

Диаграмма вариантов использования (UC)

- Диаграммы вариантов использования применяют для моделирования статического вида системы с точки зрения вариантов использования. Этот вид охватывает поведение системы, т.е. видимые извне сервисы, предоставляемые системой в контексте ее окружения

Диаграмма UC

Диаграмма вариантов использования описывает множество сценариев, объединенных вместе некоторой общей целью пользователя

Диаграммы прецедентов служат для представления контекста системы или функциональных требований к системе

вариант использования =
прецедент =
сценарий

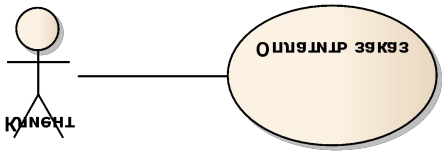
Что внутри UC?

Диаграмма UC. Условные обозначения.

Вариант использования (use case, прецедент) - описание множества последовательностей действий (включая их варианты), которые выполняются системой для того, чтобы актер мог получить определенный результат

Комплексный вариант использования

Актер (actor) - логически связанное множество ролей, которые играют пользователи вариантов использования во время взаимодействия с ними



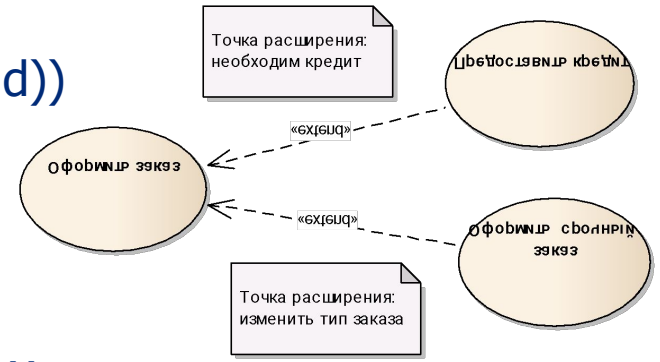
Связь ассоциации (только между ДЛ и UC)

Диаграмма UC. Актер.

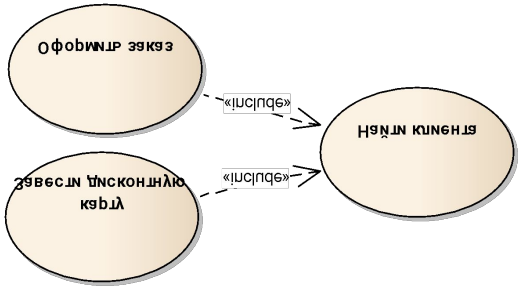
- Актерами могут быть как люди, так и внешние (по отношению к проектируемой системе) системы или аппаратные устройства
- Обычно актер представляет собой роль, которую в данной системе играет человек, аппаратное устройство или другая система
- С точки зрения актера вариант использования делает нечто представляющее для него определенную ценность, например вычисляет результат, создает новый объект или изменяет состояние другого объекта

Диаграмма UC. Условные обозначения.

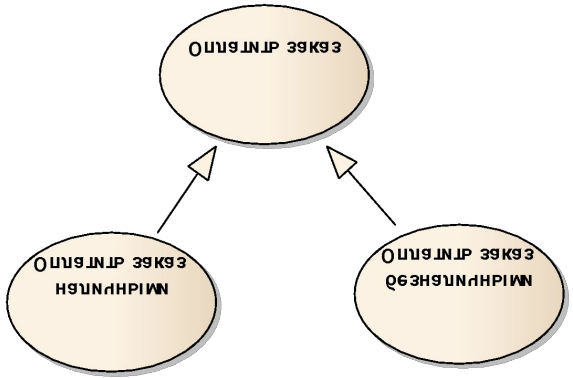
Связь зависимости (вид: расширение (extend))



Связь зависимости (вид: включение (include))



Связь обобщения (наследования)



Отношения

- Между вариантами использования возможны только связи обобщения и зависимости
- Отношение расширения используются для моделирования частей варианта использования, которые пользователь воспринимает как необязательное поведение системы Отношения расширения используются для моделирования отдельных субпотоков, выполняемых лишь при определенных обстоятельствах
- Отношения включения используется для устранения многократного описания одного и того же потока событий. Включаемый вариант использования никогда не существует автономно, а является частью базового варианта использования
- Отношение обобщения (generalization) означает, что вариант использования - потомок наследует поведение и семантику своего базового варианта использования, может замещать или дополнять его поведение, а кроме того, может быть подставлен всюду, где появляется вариант использования - предок