Российская Академия наук

Центральный дом учёных

15.03.2017

НАУЧНЫЕ ОСНОВЫ ПРОГРАММ И ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ СИСТЕМ

Доклад

доктора физ.- мат. наук, проф. МФТИ,

гнс ИСП РАН

Лаврищевой Е.М.

НАУЧНЫЕ ОСНОВЫ ПРОГРАММ

И ТЕХНОЛОГИЙ:

1. ТЕОРИЯ ПРОГРАММ ДЛЯ ПЕРВЫХ ЭВМ
2. ТЕОРИЯ ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ
3. ТЕОРИЯ ОБЪЕКТНО-КОМПОНЕНТНОГО МОДЕЛИРОВАНИЯ СЛОЖНЫХ СИСТЕМ
4. ТЕОРИЯ ВАРИАБЕЛЬНОСТИ СИСТЕМ
5. ТЕОРИЯ ВЗАИМОДЕЙСТВИЯ СИСТЕМ

Лаврищева Е.М.

1. ТЕОРИЯ ПРОГРАММ

ДЛЯ ПЕРВЫХ ЭВМ

Е.М.Лаврищева. Программная инженерия.Тема1.
Теория программирования. Уч.метод. пособие.
Москва, МФТИ.-2016.- 48 с.

Теория программ

- математическая дисциплина, изучающая математическое представление абстрактных программ, трактуемых как объекты, выраженные на формальном языке и обладающие определенной информационной и логической структурой для последующего исполнения на ЭВМ.

Программа/схема программы состоит из трех частей: синтаксис, интерпретация и семантика.

Схема программы **- это конечный ориентированный граф, показывающий структуру программы с применением сигнатур операций и формальных символов.**

Интерпретация **- это реализация функций и предикатов проверки их значений в рамках задачи конкретной предметной области.**

Семантика **- это сущность программы, задающая результат ее выполнения во множестве вычислимых функций над некоторым множеством базовых операций.**

Теория схем программ

- это новый раздел математической науки, объектом изучения которого являются математические абстракции схем программ на специальных языках с определенной логической и информационной структурой, ориентированной на исполнение на ЭВМ.

Ершов А.П. рассматривал эту теорию как часть математики. Он считал, что специалистов по теоретическому программированию нужно готовить по специальным программам университетов. Так, Н.Н. Непейвода сформировал в НГУ учебный план по предмету «Прикладная логика» для обучения студентов факультета ИТ.

Обучение теоретическому программированию (теории программ) не в университетах не проводится. Научные основы теории программ развивались в СССР и после смерти Ершова (1988).

За рубежом сформировались отдельные направления теории программ: VDM, RSL, OOP, UML, DSL и др. Они используются и сейчас. Теория и методы составили основу нового направления в SE - SEMAT (Software Engineering Theory and Method, 2009). В нем ставится цель - разработать теории и методы изготовления сложных компьютерных систем и поднять уровень научной компетенции академических и университетских специалистов в области ПО и систем.

Теорию схем программ определил вначале А. А. Ляпунов, а потом Ю. И. Янов, А.П.Ершов, В.Е.Котов[1-5] и др.

Схема программ - это модель программы из операторных схем с сигнатурой одноместных операций и допускающие в программе только одну переменную. Для схемы была разработана полная система преобразований. Автомат, воспринимающий схему программы, считался конечным автоматом, эквивалентность которого совпадала с его функциональной интерпретацией.

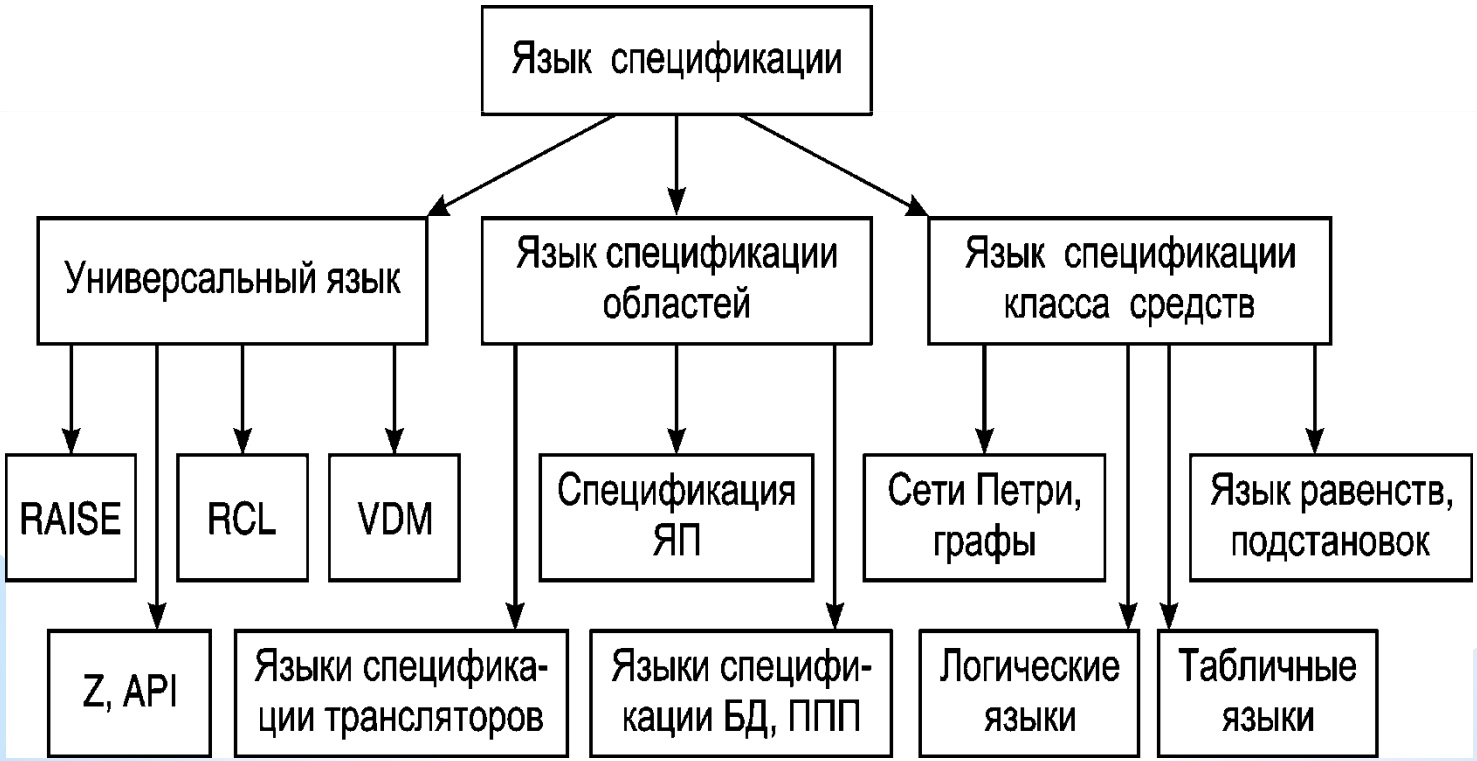
Ляпунов **А.А. определил теорию** перевода схем программ из одной модели в другую (например, в рекурсивную схему) той же сигнатуры. Обратный перевод был невозможен, т.к. требовалась большая память. В [1, 2] описана **теория схем** на ряде конкретных примеров и с применением теории графов.

[1] Ершов А. П., Введение в теоретическое программирование, М., 1977.
[2] Котов В. Е. Теория программирования в графах, Новосибирск, 1978.
[3] Ляпунов А. А., Построение схемы программ, ж. "Проблемы кибернетики», 1958, 1, с. 46-74;
[4] Янов Ю. И., Теория схем программ, "Проблемы кибернетики", 1958, 1, с. 75-127.
[5] Непейвода Н. Н., Логика программирования, 1979, М: 1, с.15-25.
[6] Семантика языков программирования. Сб. статей, пер. с англ., М., 1980.
[7] Скотт Д., "Кибернетический сборник", 1977, в. 14, с.107-121 и др.

Лаврищева Е.М.

Спецификации и доказательство программ

Для формального описания свойств программ разработаны языки спецификаций. *Спецификация программы* – это точное и однозначное ее описание с помощью математических понятий и терминов, а также правил синтаксиса и семантики языка. Некоторые спецификации представляются последовательностью операций и функций, понятных для их интерпретации. Схема языков спецификации приведена ниже.



Спецификации программ[1,2]

Спецификации программ основываются на аппарате математической логики и математических объектов, содержащих правила описания семантики на одном из формальных языков - RSL, VDM, Z, B и др. Элементы, описанные в этих языках, требуется обрабатывать с помощью специально разработанных математических средств, обеспечивающих доказательство свойств программ и правильность семантики. Существует ряд подходов к методам формальной спецификации ПС:

– операционный подход основан на описании свойств модели;
– аксиоматический подход базируется на описании функций и поведении системы;
– алгебраический подход базируется на описании свойств операций и действий над спецификацией;
– гибридный подход объединяет приведенные подходы.

1. Агафонов В.Н. Спецификации программ, понятийные средства и их организация.- Новосибирск: Наука, 1987.-240 с.

2. Непомнящий В.А., Сулимов А.А. Об одном подходе к спецификации и верификации трансляторов.- Программирование.-М.: 1983, Т4.- с.51-58.

Типы спецификации программ:

– **поведенческие** (behavioral) спецификации описывают ограничения на поведение объекта спецификации, включая описание функциональных возможностей, безопасности и выполнения;

– **структурные** (structural) спецификации описывают ограничения на внутренний состав объектов спецификации, а также на использование, состав и отношение зависимости;

– **взаимодействия** (interaction) описывают ограничения на взаимодействие между двумя или большим количеством объектов, а именно на соответствие интерфейсов и протоколов.

– **модельно-ориентированные**, которые задают построение абстрактной модели ИС или ПС, отражающей состояние элементов (state-based). Например, в VDM, Z , RAISE , B и др.;

– **алгебраические спецификации** основаны на свойствах и действиях (action-based). Примеры: OBJ, Larch, Anna, Clear, AteCharts, CSP, CCS.

Доказательство программ[1,2]

Формальное доказательство программ основано на аксиомах, предусловиях, утверждениях и постусловиях, определяющих заключительное условие получения правильного результата.

Предусловия **– это ограничения на совокупность входных параметров, а постусловия – ограничения на выходные параметры. *Пред-* и *постусловия* задаются предикатами, т.е. функциями, результатом которых будет булевое значение (true/false).**

Предусловие истинно тогда, когда входные параметры входят в область допустимых значений данной функции. *Постусловие* задает формальное определение критерия правильности получения результата. Оно истинно тогда, когда совокупность значений удовлетворяет требованиям функций.

Для доказательства программы создается набор утверждений, каждое из которых использует предусловия и последовательность инструкций, приводящих к проверке промежуточного результата относительно помеченной точки программы, для которой сформулировано данное утверждение.

Если утверждение соответствует конечному оператору программы, то с помощью заключительного утверждения и постусловия делается вывод о ее частичной или полной правильности.

1. Андерсон Р. Доказательство правильности программ.- Мир.- 1982.-165с.
2. Hoare C.A. Prof of correctness of data Representation// Acta Information, 1(4).-1972.-p.214-224.

Методы доказательства программ[1,2]

К методам доказательства программ относятся: метод рекурсивных утверждений Флойда, Наура, метод индукции Маккарти, Хоара и др.

Метод Флойда применяется для программ, которые разрабатывались путем последовательной декомпозиции задачи на несколько подзадач и для каждой из них формулируются утверждения с учетом условий ввода и вывода в точках программы, расположенных между входными и выходными утверждениями. Суть доказательства – это истинность выполнения условий и утверждений в заданной программе.

Метод Хоара – основан на аксиоматическом описании семантики исходных программ. Каждая аксиома описывает изменение значений переменных с помощью операторов этого языка. Для каждой метки программы строится правило вывода.

Метод рекурсивных индукций Маккарти состоит в проверке функций, работающих над типами данных, которые изменяют структуры данных и диаграмм перехода при символьном выполнении программ.

Метод Дейкстры включает в себя доказательство правильности программы на основе модели вычислений, анализа пути прохождения программы и правил обработки объемов данных этой программы.

1. R. W. Floyd, "Assigning meanings to programs", Proc. Symp. Appl. Math., 19; in: J.T.Schwartz (ed.), Mathematical Aspects of Computer Science, pp. 19-32, American Mathematical Society, Providence, R.I., 1967
2. Hoar K. О Структурной организации данных //Структурное программирование.– М.: Мир, 1975.– с.92 – 197.

Лаврищева Е.М.

Пример доказательства программ[1]

Дан массив ***array T [1:N]. Требуется расположить его элементы в порядке возрастания их значений в массиве T' [1:N].***

Входное условие задается начальным утверждением:
$A_{нач}$: (*T [1:N]* – массив целых) & (*T' [1:N]* массив целых).
Выходное утверждение $A_{кон}$ - это конъюнкция (2, 3) условий:
(1) (*T*– массив целых) & (*T'* – массив целых)
(2) ($\forall i$, если $i \le N$, то $\exists j$ ($T'(i) \le T' (j)$),
(3) ($\forall i$, если $i < N$, то ($T' (i) \le T' (i+1)$),
Если утверждение (1) - истинно, то истинно и (2). Т.е. если (1) утверждение – A_1 преобразуется к A_2 , то первой теоремой является: $A_1 \rightarrow A_2$.
Если A_3 – следующая точка преобразования, то второй теоремой будет: $A_2 \rightarrow A_3$. И так до конца $A_{нач} \rightarrow A_{кон}$.
Конечная теорема формулируется так:
условие истинно в последней точке, если оно отвечает истинности выходного утверждения: $A_{нач} \rightarrow A_{кон}$.

1. Лаврищева Е.М. Методы программирования.-К.: Наук.думка.- Теория, инженерия, практика.- 2006.- 453с.

Верификация программ

Верификация – это проверка правильности создания модели системы в соответствии с ее спецификацией. Метод верификации помогает сделать заключение о корректности созданной программы после завершения ее проектирования и разработки.

Валидация – это проверка правильности выполнения заданных требований к системе, она включает в себя ряд действий по просмотру, инспекции спецификаций и результатов проектирования после подтверждения того, что имеется корректная реализация начальных требований и выполнены заданные условия и ограничения. Т.е. задачи верификации и валидации - проверка полноты, непротиворечивости и однозначности спецификации требований и правильности выполнения функций системы.

Верификация и валидация - это процессы в стандартах ISO/IEC 3918–99, 12207-1996 и 2007. Они определяют цели, задачи и действия по проверке правильности создаваемого продукта на этапах ЖЦ в соответствии с требованиями. Их основная задача состоит в том, чтобы ***проверить и подтвердить***, что конечный продукт отвечает назначению и удовлетворяет требованиям.

Научная база теории программирования (Лексикон программирования) А.П. Ершова[1,2]

В лексикон включает теорию алгоритмов, теорию задания теорем, основные соотношения и конструкции описаний алгоритмов программ. Они образуют базу теории программ.

Для предметных областей (математика, физика, биология и др.) необходимо определить базу знаний, модели и соотношения, которые обеспечивают достижение требуемой точности и строгости в постановке любых задач для их вычисления на ЭВМ.

Ершов А.П. считал, что перед вычислительным делом стоит задача, аналогичная той, которая стояла перед математикой, естествознанием в XVIII—XIX вв. и которая привела к созданию математического анализа и целого ряда инженерных наук: техническая физика, строительная механика, электротехника и др. «Нам необходимо построить анализ и исчисление программ и на их основе создавать инженерные программные дисциплины для применения на ЭВМ для самых разных областей науки и техники (управления, экономики, производства программ и др.).

1. Ершов А. П., Предварительные соображения о лексиконе программирования, М., Кибернетика и ВТ. Вып.1.-М.: Наука, 1984. с.199-210.
2. Ершов А.П. Научные основы доказательного программирования.– Научное сообщение в президиуме АН СССР, Наука, 1986.–с.1–11.

В.М. Глушков применил (1957-1964) **алгебру и математический анализ** в качестве средств моделирования параметров и свойств поведения математических задач, использующих методы решения дифференциальных и интегральных уравнений[1-3] .

Для решения математических задач сформулированы:

- **физическая модель**, включающая свойства и характеристики объектов;
- **математическая модель** (вместе с физической) задает размерности, параметры и операции работы с данными задачи;
- формальное соответствие математической модели и **численной**;
- математические дисциплины – **алгебра, диф. и интегральное исчисление** – аппарат для описания инженерных задач;
- **язык алгебраического программирования** инженерных задач (**Аналитик**[2]), использующий математический анализ, операции (+, x, …), операторы обработки десятичных, целых, рациональных чисел и дробей, а также механизмы доказательства эквивалентности их преобразования.

Теорема. Если выражения Q1 и Q2 принадлежат некоторой подалгебре Q, в которой задана каноническая форма F, F(Q1) и F(Q2), и они полностью совпадают, то Q1 и Q2 – эквивалентны).

---

1. Капитонова Ю.В., Летичевский А.А. Методы и средства алгебраического программирования // Кибернетика. – 1993.–№ 3. – С. 7–12.
2. Глушков В.М., В.Г.Бондарчук, Т.А.Гринченко и др. АНАЛИТИК-74, 79, 89, 93, 2000. – Кибе6рнетика и системный анализ. –1995. –№5. –с.127–157.
3. Системы компьютерной алгебры. Семейство Аналитик. Теория, реализация, применение.-ИПММС НАНУ, К.: 2010.-762с.

Лаврищева Е.М.

Язык семейства Аналитик[1,2]

Этот язык - основа математического моделирования инженерных задач для машин серии Мир1-Мир3.

Он включает средства для описания инженерных задач:

- формальные обозначения для математических операций и операторов;
- синтаксическое описание объектов и конструкций в языке и их алгебраические преобразования;
- средства распознавания свойств чисел, эквивалентности выражений с целью минимизации числа тождественных преобразований, а также преобразований свойств дистрибутивности, канонических форм и чисел с помощью широкого набора стандартных функций (тригонометрических, логарифмических, экспоненциальных и др.);
- средства отладки, трассировки и выполнения программ;
- библиотека математических и численных задач различных классов.

Язык Аналитик был встроен в «Мир»[1-3] и обеспечивал решение инженерных задач (физических, астрономических, экономических, машиностроительных и др.). Машины «Мир» В.М. Глушков считал «умными», они применялись в СССР на разных предприятиях страны. Развитие - компьютерная алгебра[2]. Идея «умных» машин развивалась в Японии[3] - ЭВМ 5 поколения (1985-1990).

1. Глушков В.М., Бондарчук В.Г., Гринченко Т.А. и др. АНАЛИТИК-74, 79, 89, 93, 2000 // Кибернетика и системный анализ. – 1995. – № 5. – С. 127–157.
2. Системы компьютерной алгебры. Семейство Аналитик. Теория, реализация, применение. ИПММС НАНУ, К.: 2010.-762с.
3. Исидзука М. Представление и использование знаний .-Под ред.- Х.Уэно, М.: Мир, 1989.-220 с.

Лаврищева Е.М.

Адресный язык Е.Л. Ющенко (1957-1967)[1-3]

Язык содержит операции теории множеств и отношений, операторы, которыми записывается одна или несколько математических формул, а также операторы задания порядка операторов - метка и безусловный переход и др. Переменные обозначались буквами, им соответствовали ячейки машины. Содержимое некоторого адреса отмечалось указателем, адресом второго ранга, который использовался для формируемого перехода. Этот язык также применялся для описания программ вычислительного характера и реализации трансляторов для машин УМШН, Урал и Днепр и др.

В (1) определены универсальные алгебры (подалгебры, логики, многоосновные алгебры), САА - системы алгоритмических алгебр (алгебра Поста, тождественные соотношения и преобразования схем адресных алгоритмов), формальные языки и грамматики; синтез автоматов, методы анализа и СМ-формализмы формального описания систем программирования.

1. Ющенко Е.Л. Адресное программирование.- Знание.-1963.-31с.
2. Глушков В.М., Цейтлин Г.Е., Ющенко Е.Л. Алгебра. Языки. Программирование. – К.: Наук. думка, 1978. – 318 с.
3. Цейтлин Г.Е. «Введение в алгоритмику. –Киев, Сфера. –1998. –310 с.

Лаврищева Е.М.

Теория модульных структур программ[1,2]

Модуль – это элементарный программный элемент, имеющий свойства:

– логическая завершенность функции;
– независимость одного модуля от других;
– замена отдельного модуля без нарушения структуры;
– вызов других модулей и возврат данных вызвавшему модулю;
– уникальность именования и др.

Модуль **преобразует множество исходных данных *X* во множество выходных данных *Y* и задается в виде отображения *M* : *X* →*Y*.**

Виды связи **между модулями:**

- связь по управлению (*CP* = $K_1 + K_2$);
- связь по данным.

Граф **модульной структуры *G* = (*X*, D), где *X* – конечное множество вершин, а D – конечное подмножество прямого произведения *X* x *X* x *Z* - множество дуг графа.**

Модульной структурой называется пара *S* = (*T*, *χ*), где *T* – модель модульной структуры; χ – характеристическая функция, определенная на множестве вершин *X* графа модулей *G*.

Значение функции χ определяется так:

***χ*(*x*) = 1, если модуль с вершиной *x* ∈ *X* включен в состав ПС;**

***χ*(*x*) = 0, если модуль с вершиной *x* ∈ *X* не включен в состав ПС и к ней нет обращения из других модулей.**

1. Лаврищева Е.М., Грищенко В.Н. Связь разноязычных модулей в ОС ЕС, М.: 1982.-137с.
2. Лаврищева Е.М., Грищенко В.Н. Сборочное программирование.- К.: 1991.- 231с.

Определение 1. Две модели модульных структур $T_1 = (G_1, Y_1, F_1)$ и $T_2 = (G_2, Y_2, F_2)$ тождественны, если $G_1 = G_2$, $Y_1 = Y_2$, $F_1 = F_2$. Модель T_1 изоморфна модели T_2, если $G_1 = G_2$ между множествами Y_1 и Y_2 существует изоморфизм φ, а для любого $x \in X$ $F_2(x)=\varphi(f_1(x))$.

Определение 2. Две модульные структуры $S_1 = (T_1, \chi_1)$ и $S_2 = (T_2, \chi_2)$ тождественны, если $T_1 = T_2$ и $\chi_1 = \chi_2$ модульные структуры S_1 и S_2 *изоморфны*, при условии T_1 изоморфна T_2 и $\chi_1 = \chi_2$.

Свойства модульных структур[1]:

- граф модульной структуры *G* имеет один или несколько элементов связности, каждая из которых представляет ациклический граф;
- в каждом элементе графа *G* есть единственная вершина, *корневая,* которая характеризуется тем, что входящих в нее дуг нет и соответствующий ей модуль выполняется первым;
- циклы допускаются только для вершины с рекурсивным обращением к самой себе. Обычно такая возможность реализуется компилятором с ЯП и данный тип связи не рассматривается межмодульным интерфейсом;
- пустой граф G_0 соответствует пустой модульной структуре.

Модули повторного использования собираются промышленным способом[1]

---

1. Лаврищева Е.М., Грищенко В.Н. Сборочное программирование. Основы индустрии ПС.- К.: 2009.- 377с.

Теория информационных систем[1]

В.М. Глушков определил базовые принципы проектирования ИС:

1. Принцип системного подхода к проведению анализа систем управления, структуризации, выделения целей и критериев, а также распределения задач управления между ИС и корпоративной сетью.

2. Принцип декомпозиции определяет функциональные признаки и типы подсистем: руководства, кадрового обеспечения, делопроизводства, мониторинга деятельности и т.п.

3. Принцип моделирования обеспечивает анализ элементов декомпозиции системы, типизацию решений по функциям и задачам системы, синтез этих элементов, а также определение экономико-математических моделей поиска проектных решений среди многих вариантов.

4. Принцип добавления новых задач связан с непрерывным процессом улучшения деятельности организации, усовершенствования и введения новых функций (мониторинг ИС, деловая графика и т.п.).

5. Дополнительные принципы, включающие внешние (анализ, учет, контроль) и внутренние (получение, сбор, регистрация, хранение, поиск) задачи принятия управленческих решений.

---

1. В.М. Глушков. Основы безбумажной информатики.-М.: Наука, 1982.

Новые принципы проектирования ИС

1. **Принцип единства** информационного и управленческого процесса с использованием менеджмента планирования и контроля исполнителей;
2. **Принцип интеграции ИС** с различными видами, методами и средствами поддержки процесса проектирования ИС;
3. **Принцип интерактивности и диалога** человека с системой в процессе управления ИС;
4. **Принцип интеллектуализации деятельности** человека при работе с документами для разных сфер применения в соответствии с принятой лексикой и решениями стандартов документооборота;
5. **Принцип адаптивности ИС** при изменении аппаратуры, платформ, сред и ИТ- технологий управления предприятием и подготовкой студентов.
6. **Принципы интероперабельности**, Интернет «вещей», новых стандартов www3c по моделированию и управлению сетевыми сервисами, сообщениями и др.

Модели документов ИС[1]

1. Характеристики объема **– это *регулярная* часть документа из последовательности повторяемых групп полей и *нерегулярная* часть документа, не содержащая повторяемых структур данных.**

Выведены формулы расчетов характеристик объема документов:

- средний объем $V=l_h+n_s k_s l_s^{max}$;

- максимальный объем $V_{max}=l_h+ n_s^{max} k_s l_s^{max}$, где l_h – размер нерегулярной части документа; n_s – количество строк, которые заполняются для данного типа документов; k_s – коэффициент заполнения; l_s^{max} – максимальный размер регулярной части документа.

2. Характеристики времени **включают в себя:**

1) суммарные значения времен обработки разных типов документов в соответствии с их маршрутом;

2) время выполнения отдельных операций над документами в разных P_i и P_j узлах системы;

3) время передачи документов между разными узлами обработки ИС.

---

1. Задорожная Н.Т., Лаврищева Е.М. Проектирование документообороту в управленческих ИС.-Пед.думка.2005. 289с.

Теория дискретных систем

Для решения сложных математических задач предложен *общематематический* процедурный язык - концепторный язык (КЯ), обеспечивающий формальное описание суммирования бесконечных рядов, теоретико-множественные операции с бесконечными множествами, гильбертов оператор и др. КЯ приближен к языку современной математики без потери императивных возможностей. Тексты в КЯ - это концепторы, а не алгоритмы.

КЯ - многосортный логико-математический язык *выражений X,* объектами которого являются типы и выражения. *Тип* – это средство построения выражений и структуризации множества значений (денотаты). *Выражения* состоят из термов и формул. *Термы* – это объекты ПрО, а *формулы* – утверждения об объектах и отношениях между ними. Конструкторы, предикаты и формул разделены на четыре категории: функторы и предикаты, конекторы и субнекторы.

Функтор **– это конструктор, преобразующий термы в термы (арифм. и алгебраические операции над числовыми множествами).**

Предикаты **превращают термы в формулы;**

Конекторы **включают в себя логические связи и кванторы для преобразования одной формулы в другую.**

Субнектор **(дескриптор) – это конструктор формул из термов и выражений, которые содержат формулы над числовыми множествами и вещественными функциями. Описаны конструкторы множеств, кортежей, отношений, семейств, произведений множеств и др.**

Лаврищева Е.М.

Теория дискретных систем [1, 2]

концептор К (< список параметров >)
< список импортных параметров>
< определение констант, типов, предикатов>
< описание глобальных переменных>
< определение процедур>
начало К
< тело концептора>
конец К.

Для формализации семантики КЯ используются теоретико-модельный (денотационный) и аксиоматический подходы. *Денотационный подход* определяет выражение элемента из множества денотатов с помощью функции φ интерпретации символов сигнатуры языка. Каждой константе $c \in C$, функциональному символу $f \in F$ и предикатному символу $p \in P$ сопоставляется объект из множества денотат. Дедуктивная теория выделяет из множества всех формул подмножество аксиом и правил вывода.

Для каждой пары R_1, R_2 и оператора I создается *операторная* формула $\{R_1\}\, I\, \{R_2\}$ с таким утверждением: если R_1 истинно перед выполнением оператора I, то завершение оператора I обеспечивает истинность R_2. Иными словами, формула R_1 это предусловие, а R_2 — постусловие оператора I.

1. В.Н. Коваль. Концепторные языки. Доказательное проектирование.- К.: Наук.думка, 2001.-182 с.
2. Коваль В.Н., Рабинович З.Л. Логико-алгебраический подход к верификации дискретных систем.- IV межд. Конференция Технология ПО-1995.

Лаврищева Е.М.

КЯ применяются для постановок и решения задач **распознавания динамических** обстановок в гидроакустике, радиолокации и т.п. в целях создания прикладного ПО для технических объектов новой техники. Средства автоматического доказательства таких элементов основаны на логико-алгебраической спецификации, позволяющей провести доказательство утверждений о свойствах аппаратно-программных систем.

Дискретная система (*S*) – это черный ящик с конечным набором входов, выходов и состояний. Функционирование системы *S* определяется набором частичных отображений как носителей алгебраической системы с набором операций, которые входят в состав сигнатуры и образуют *частичную алгебру с* алгебраическими спецификациями, отображающими состояние *S* системы.

Логико-алгебраические спецификации КЯ ограничены **хорновскими формулами**, которые упрощают процесс автоматического доказательства теорем. Если их заменить булевыми функциями, то получим характеристические функции их отношений.

Алгебраические системы в КЯ – это многоосновные алгебры, а аксиомы – спецификации: тождественные и квазитождественные. Семантика языков логико-алгебраических спецификаций основана на *переписывании термов,* а создаваемая алгебраическая спецификация получает логическую семантику, используемую при доказательстве теорем.

Лаврищева Е.М.

Языки математического проектирования VDM, Z, RLS, Clear[1-3]

Математическая символика: *X* — натуральные числа с нулем; *N* — натуральные числа без нуля, Int, Bool и др. Объекты - это элементы данных, которыми оперируют функции, множества, деревья, последовательности, отображения, а также операции формирования новых объектов.

***Множество* X-set. Над ним применяются операции e, c, u, n и др. Пример: x e *A* будет корректным только тогда, когда *A* есть подмножество множества, которому принадлежит x.**

***Дистрибутивное объединение* имеет вид: union {(1, 2), (0, 2), (3, 1) } = (0, 1, 2, 3).**

***Списки* — это цепочки элементов одинакового типа из множества *X*. Операция len задает длину списка, а inds — номер элемента списка (inds lst =(i *eX* [*f* < *i* < len]).**

***Операции* - конкатенация списков, дистрибутивная конкатенация и взятие первого элемента списка — hd и остатка (хвоста) списка — tl. Например, hd *(a, b, c, d) = (a),* tl *(a, b, c, d) = (b, c, d).* Операция *dom* строит множество ключей, а *rng* — множество всех его значений.**

***Дерево* — это конструкция *mk*, позволяющая объединять объекты разной природы (последовательности, множества и отображения). В деревьях может использоваться конструктор составных объектов и деструктор именования констант.**

***Отображение* — это конструкция *map*, позволяющая создавать абстрактную таблицу из двух столбцов: ключей и значений, а также операции: исключить строку, слить две таблицы и др.**

1. Biorner D., Jones C. B. The Vienna Development Methods (VDM): The Meta Language. Vol. 61 of Lecture Notes in Computer Science. Heiderberg : Springer Verlag, 1978.
2. Burstall R., Goguen I. The semantic of Clear, a specification language//Lect.Notes.Comp.Sci.-1980, v.86.-40p.
3. Петренко А.К. Венский метод разработки программ.- Программирование, 2001.-№1.-с.3-23.

Лаврищева Е.М.

Теория ООП Г. Буча[1,2]

Теория Г. Буча содержит математические объекты и операции для моделирования моделей систем:

- объект – это конкретные физические предметы или группы предметов реального мира;

- класс объектов – это множество объектов с общими свойствами;

- экземпляризация (выделение экземпляра объекта в классе);

- наследование объектов в другом классе;

- инкапсуляция – скрытие свойств отдельных объектов;

- полиморфизм – множественность объектов и др.

В теорию входят операции: описания объектов, классов; трансформации объектов; интеграции объектов и взаимодействия и др.

Любая предметная область – это совокупность объектов, связанных между собой некоторым множеством отношений и поведения во времени:

<объектная ориентация> = <объекты> + <наследование>.

Развитием ООП является UML. В нем в процессе анализа создается концептуальная ОМ в виде диаграмм прецедентов или диаграмм классов объектов, компонентов, деятельности, состояния, взаимодействи и др. Из них образуется модель системы или каркас в виде сети.

---

1. Буч, Г. Объектно-ориентированный анализ / Г. Буч. — М. : «Лаборатория Бином», 1998.

Программная инженерия ПО[1,2]

Термин Software Engineering – SE появился в 1968г. Он означал систематический подход к разработке, эксплуатации, сопровождению и прекращению использования программных средств (www.swebok.com). В состав SE входят методы, средства и инструменты, которые обеспечивают качественный и производительней труд программистов при создании ПО и ПС. Swebok и Curricula SE – ориентированы на обучение.

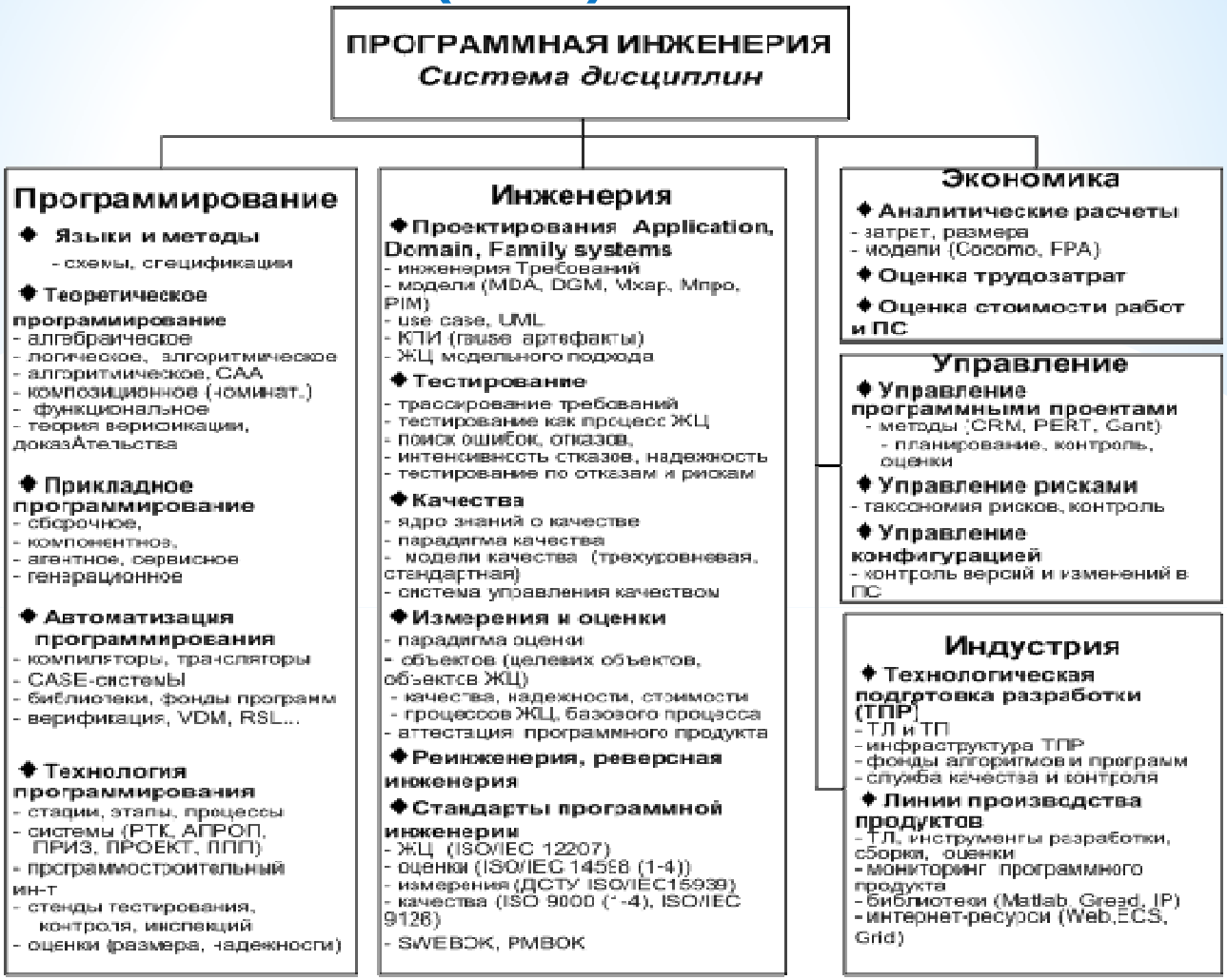
Ядро SWEBOK разработано международным комитетами ASM и IEEE и предназначен для обучения программной инженерии – Curricula 2001 (2004, 2007, 2014). Ядро SWEBOK содержит 10 областей знаний, которые определяют общие знания о подходах к разработке ПО систем.

К ним относятся: разработка требований к ПО, проектирование ПО, конструирование ПО, эксплуатации ПО, сопровождения ПО, верификация ПО, валидация ПО, управление качеством, конфигурацией и др. Регламентацию деятельности по разработке ПО осуществляет стандарт ЖЦ ISO/IEC 12207-1996, 2007.

Процесс управления качеством ЖЦ согласно стандарту ISO/IEC 2844–89 «Оценка качества программных средств» (1987) проводит оценку отдельных свойств и характеристик разрабатываемого ПО с учетом 50 критериев и 6 факторов стандарта измерения качественных показателей программных систем.

Лаврищева Е.М.

Новые дисциплины SE (2008) [1-3]



1. Software engineering as a scientific and engineering discipline E. M. Lavrishcheva, 2008, Volume 44, Number 3, Pages 324-332.

2. Classification of software engineering disciplines. E. M. Lavrischeva, 2008, Volume 44, Number 6, Pages 791-796, Software–Hardware Systems.

3. Ekaterina Lavrischeva, Alexei Ostrovski. New Theoretical Aspects of Software Engineering for Development Applications and E-Learning, Journal of SEA, 2013, 6, 34-40/ (http://www.scirp.org/journal/jsea).

Лаврищева Е.М.

Характеристика дисциплины SE (2008)

– ***научное программирование*** **– *это совокупность* теоретических и формальных основ программирования и автоматизации проектирования разных программных и прикладных систем;**

– ***инженерная дисциплина*** **– это совокупность средств и методов проектирования на основе ЖЦ стандарта отдельных приложений систем, их тестирования и оценки качества выходного кода, интероперабельного к платформам и средам;**

– ***дисциплина управления*** **– это методы управления и организации планирования работ по изготовлению коллективом отдельных элементов систем, анализа рисков срыва планов, верификации и генерации варианта продукта;**

– ***экономическая дисциплина*** **– это совокупность методов экспертного, качественного и количественного оценивания результатов создания с необходимыми расчетами общего времени, объема, трудоемкости и стоимости изготовления готового продукта;**

– ***индустриальная дисциплина*** **– это промышленная технология конвейерного сборочного производства систем из готовых программных ресурсов (КПИ, ГОР) из библиотек и репозиториев.**

SEMAT – Software Engineering Methods and Theory (2009)

SEMAT (I. Jacobson, B. Meyer и P. Soley) поставили цель - изменить программную инженерию, так чтобы разработка ПО квалифицировалась как строгая математическая дисциплина.

Цель состоит в том, чтобы продолжить многолетнюю работу в SE по теории и преодолеть разрыв между теорией академических специалистов и сообществом разработчиков разных видов ПО, т.е. необходимо создать общие методы для всей программисткой общественности.

Работы в SEMAT структурированы по четырем областям: Практика, Образование, Теория и Сообщество. *Практика* развивает практические работы. *Образование* затрагивает все вопросы, связанные с обучением разработчиков, студентов и специалистов. *Теория* занимается созданием общей теории для разработки ПО и систем. *Сообщество* - это лица, которые создает веб-сайты и развивают их для потребностей пользователей. Со временем Практика, Образование и Теория будут интегрироваться. Теория должна будет направлять исследования и создавать научные теории в SE для разработчиков ПО. Образование должно внедрять новые теории при преподавании в ВУЗах.

1. http://www.semat.org

2. http://blog.paluno.uni.de/semat.org/wd-content

Лаврищева Е.М.

2. ТЕОРИЯ ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ СИСТЕМ

1. Лаврищева Е.М.. Программная инженерия. Тема 2. Технология программирования. Уч.метод. пособие. Москва, МФТИ.-2016.- 48 с.
2. Лаврищева Е.М. Методы программирования. Теория, инженерия, практика.- К. Наук.думка, 2006.- 371 с.

Лаврищева Е.М.

Базовые понятия ТП

Модуль – это программный самостоятельный элемент, выполняющий некоторую функцию.

Объект - это абстрактный образ или конкретный физический предмет или группа предметов на множестве характеристик и функций.

Компонент - самостоятельный продукт, который поддерживает объектную парадигму, реализует часть или всю отдельную ПрО и взаимодействует с другими через интерфейсы.

Программная система (ПС) – совокупность объектов или компонентов, реализующих набор функций ПрО в заданной среде.

Семейство программ (Дейкстра) – это совокупность программ, имеющих общие характеристики, заданные на общем множестве понятий и свойств, присущих каждой отдельной программе семейства.

Семейство продуктов (ISO/IEC FDIS 24765:2009 E) - «группа продуктов или услуг, которые имеют общее управляемое множество свойств, которые удовлетворяют потребности определенного сегмента рынка или вида деятельности».

Компоненты повторного использования (КПИ) – это продукты производственного назначения, которые сохранялись в Фондах алгоритмов и программ республиканского и Всесоюзного значения (1976–1992) для применения разными организациями при создании АС, АС, АСНИ, АСУ, АСУ ТП и др. Сегодня это http://e-library.ru, Scopus и др.

Лаврищева Е.М.

Технология программирования (ТП) - это совокупность методов, средств и инструментов для описания свойств и функций разных задач предметной области в виде программ с целью получения решения задачи.

Средства ТП:

1. Программирующая программа (ПП).

ПП (1953) для языка операторных схем Ляпунова А.А. была представлена операциями, содержащими команды управления по графу программы операторов программы, которые зафиксированы в специальных таблицах для реализации программы. ПП-1 (1954) сделана в МГУ Э. З. Любимским, А. П. Ершовым под руководством Ляпунова А.А.

2. Библиотечный метод. В.М.Глушков в статье «Об одном методе автоматизации программирования», ж. Проблемы кибернетики, № 2, 1957) определил это метод для решения системы дифференциальных уравнений для машин (МЭСМ, Урал 1, УМШН, Днепр и др.) с использованием адресного языка (Е. Л. Ющенко, В.С.Королюк).

3. Библиотеки СП численных методов *(* Е. А. Жоголев) и метод интерпретации программ ИС-2 (М. Р. Шура-Бура) на машинах М-20, БЭСМ и др.

4. Трансляторы с языка Алгол (ТА) [1,2] :

ТА1– С. С. Лавров (ЛГУ, 1962);

ТА2 – М. Р. Шура-Бура и Э. З.Любимский (ИПМ, 1963);

ТА-3 (Альфа-система) русской версии язык Алгол-60 (СО АН СССР, 1964);

ТА-4 – Е. Л. Ющенко, Е. М. Лаврищева – для УВК «Днепр-2» (ИК АН УССР).

Трансляторы ТА1-ТА3 и ОС для новых ЭВМ (М-20, СТРЕЛА, БЭСМ) были реализованы в машинном коде, а транслятор ТА4 – на Адресном языке. Все трансляторы были сданы Госкомиссии СССР проекта ОМО для ОС и Библиотек программ для ЭВМ: М-20, СТРЕЛА, БЭСМ, Днепр, Урал и др.

Транслятор с АЛГОЛ 68 [3] (Г.С. Цейтин, А.Н. Терехов, ЛГУ, 1968–1991) в ЛГУ, Минске, Новосибирске (Бета). Был сделан для первых ЭВМ – СМ1, СМ2 и ЕС ЭВМ. В этом языке программы переводились в промежуточный язык и в код конкретной машины.

1. Шура-Бура М.Р., Любимский Э.З. Транслятор с языка Алгол-60 // ЖВМ и МФ. –1964. – Т. 4. – № 1.
2. Лаврищева Е.М., Борисенко Л.Г., Гришкевич К.И. и др. Транслятор с языка Д-АЛГАМС для УВК «Днепр-2». – К.: ИК АН УССР, 1970. – 186 с.
3. Терехов А.Н. Проблемы идентификации и структура компилятора с языка АЛГОЛ 68. – http://www.math.cpbu.ru/usr/ant/all_articles/003

5. Операционные системы [1]

На БЭСМ-6 была сделана ОС, которая выполняла общие функции управления памятью, задачами и данными. В ОС «Электроника ССБИС» (1985–1991) в ядро вошли средства взаимодействия с IBM и Эльбрус, набор протоколов обслуживания программ, СП с ЯП (Фортран, С, Паскаль и др.) и средства поддержки абстрактных типов данных, кластерного сборочного конвейера для загрузки модулей в ОС и в библиотеку программ трансляторов, а также средства отладки программ на ЯП.

6. Р-технология [2,3]

Для конструирования произвольных программ с помощью визуальных Р-графов и схемной их реализации разработан Графический стиль программирования (И.В.Вельбицкого) разных задач для применения во многих организациях ВПК СССР. Анализ программ в Р-языке проводился в комплексах ТКП.

1. Иванников В.П., Королев Л.Н., Любимский Э.З., Томилин А.Н.. Разработки московской школы операционных систем. International Symposium “Computers In Europe. Past, Present and Future. Proceedings, October 5-9, 1998.- Ukraine, Kiev.- 480 p.

2. Вельбицкий И.В. Технология программирования. – Киев: Техника, 1984. – 278 с.

3. Вельбицкий И.В., Ходаковский В.Н., Шолмов Л.И. Технологический комплекс автоматизации программ на машинах ЕС ЭВМ и БЭСМ-6. – М.: Статистика, 1980. – 263 с.

Лаврищева Е.М.

7. Системы синтеза, композиции и сборки программ [1-4]

***Система* ПРИЗ** (Э. Х. Тыугу) выполняла синтез программ в языке *Утопист* и ЯП на основе семантической модели предметной области и программ. Метод синтеза программ в PL/1, Fortran, Assembler реализован путем подстановки семантики их реализации в некотором из этих ЯП.

Композиция программ (Редько В.Н.) – это операции объединения функций и данных типа: «данные–функция–имя» и «функции-композиция-дескрипция» на некотором множестве именованных данных, дескрипций и денотатов (значений). Операции композиции – это подкласс стандартных композиций и композиционных функций.

Сборка программ (Лаврищева Е.М.). Выпуск сложных программ по типу сборочного конвейера Форда В. М. Глушков предложил собирать из модулей в разных ЯП на основе *паспортов,* содержащих описание входных и выходных данных в разных ЯП (Алгол, Кобол, Фортран, ПЛ-1, Модула и др.), которые задавались в модулях-посредниках интерфейсного типа (позднее stub и skeleton в IDL (Interface Definition language) системы CORBA. Сборка выполнялась специальным сборщиком (потом брокером) в системе АПРОП, переданной в 52 организации СССР.

1. Кахро М. И., Калья А. П., Тыугу 9. Х. Инструментальная система про-граммирования ЕС ЭВМ (ПРИЗ). М.: Финансы и статистика, 1981.
2. Редько В.Н. Основания программологии // Кибернетика и системный анализ. – 2000.– № 1.– С. 35–57.
3. Лаврищева Е.М., Грищенко В.Н. Сборочное программирование.- К.: Наук. думка.- 1991.-231с.
4. Липаев В.В., Позин Б.А., Штрик А.А. Технология сборочного программирования.-Мю: Радио и связь, 1992.-276 с.

Лаврищева Е.М.

8. Виды интерфейса в программировании

Межмодульный интерфейс – это модуль-посредник между двумя связываемыми модулями, выполняет функции передачи, приема и преобразования нерелевантных данных. Язык определения интерфейсов (ЯОИ) в системе «АПРОП» сыграл свою роль при сборке разноязыковых модулей в системе «АПРОП» (1982). Определен стандарт языка IDL (Interface Definition Language) в CORBA (1994).

Межъязыковой интерфейс – совокупность средств и методов спецификации и трансформации структур данных ЯП для взаимно однозначного обмена данными между разноязычными модулями через интерфейс. Библиотека интерфейса (64 функции) для ЯП (Algol-60, PL/1, Cobol, Fortran, Assembler) ОС ЕС была передана в 52 организации СССР для применения при работе программ на разных языках/

Технологический интерфейс – совокупность методов и средств для осуществления процессов и операций на технологических и продуктовых линиях (Product Line/ProductFamily) при реализации программного продукта из готовых ресурсов и решений (assets).

Компоненты повторного использования (КПИ) – это продукты производственного назначения, которые сохранялись в Фондах алгоритмов и программ республиканского и Всесоюзного значения (1976–1992) для применения разными организациями при создании АС, АС, АСНИ, АСУ, АСУ ТП и др.

8. Метод сборки модулей и интерфейсов (1980) [1-3] :

- один из методов программирования для повторного использования готовых программных средств, КПИ;

- основан на стандартном описании КПИ и интерфейсов в отличие от других методов (синтеза, композиции).

Объект сборки обладает свойствами (наследования, полиморфизма и инкапсуляции), включает данные и операции для установления связей между разными объектами. Объекты сборки могут обладать общими свойствами и методами, образовывать классы и быть экземплярами класса.

Процесс сборки объектов может проводиться вручную (программистом) или автоматизированным способом. Автоматизированный сборщик осуществляет объединение отдельных элементов системы с помощью стандартных правил взаимосвязи разнородных объектов с помощью примитивов и функций разного рода библиотек в (IBM, VS.Net и др.).

1. Лаврищева Е.М., Грищенко В.Н. Сборочное программирование.К.: Наук.думка.- 1991.-231с.
2. Грищенко В.Н., Лаврищева Е.М. О создании межъязыкового интерфейса для ОС ЕС // УсиМ. – 1978. – № 1. – С. 34–41.
3. Коваль Г.И., Коротун Т.М., Лаврищева Е.М. Технологический интерфейс для разработки систем // Труды Международ. конференции «Интерфейс-СЭВ», 1987. – С. 97–110.
4. Лаврищева Е.М.Технологические основы модульного программирования.-Кибернетика, 1980.-№2.-44-49.

Лаврищева Е.М.

Научные основы доказательного программирования [1]

Создание научных основ доказательного программирования (1986), которые должны стать опорой специальной подготовки программистов и новой технологической дисциплиной программирования. В эти основы внесли вклад ученые из разных стран: Ф. Л. Бауэр, Р. Берсталл, Э. Дейкстра, Дж. Маккарти, В. Турский, Р. Флойд, А. Хоар и советские ученые - Я. М. Бардзинь, В. М. Глушков, С. С. Лавров, А. А. Летичевский, А. А. Ляпунов, В.Н.Редько, Э.Х.Тыугу и др.

А.П. Ершов выделил три вида доказательного программирования: синтезирующее, сборочное и конкретизирующее.

***Синтезирующее программирование* — это процесс получения программы, исходя из условия задачи и метода ее решения. Условие задачи записывается в виде совокупности логических формул, в которые входят символы известных и неизвестных величин, операций и функций. Спецификация программы определяет формулировку теоремы, утверждающую существование решения задачи.**

Синтез программы состоит в поиске доказательства теоремы существования в некотором конструктивном логическом исчислении (В.Турский, Э.Х.Тыугу).

---

1. Доклад А.П. Ершова в Президиуме АН СССР 1986г. на звание академика АН СССР

Сборочное программирование **решает задачу многократного и быстрого процесса создания программ из заранее изготовленных «деталей», роль которых играют модули, обладающие определенной структурной и функциональной целостностью и приспособленные к контролируемому взаимодействию с другими модулями.**

Это может быть синтез программы, который извлекает из условия задачи схему сборки модулей и по формальным правилам процесс построения программы делается автоматическим.

Теоретическую основу сборочного программирования составляет, так называемая модель , граф предметной области, в вершинах которого находятся модули, а на дугах задаются связи (данные для интерфейса).

Конкретизирующее программирование **– это конкретизация из универсальной программы некоторых программ данного класса, для которого требуется теория определения и интерпретации данных в структуре предметной области. Полнота этой теории базируется на определении метода решения любой задачи из заданного класса.**

Лаврищева Е.М.

Перспективы технологии программирования по Ершову[1]

Три поколения интегральной промышленной ТП.

Первое поколение (организац. Программирование, 1975–1985):

- язык программирования (ЯП) не формализован. Переход от прототипа к программной версии не формализован;
- ЯП – ФОРТРАН, КОБОЛ, ПЛ/1, Ассемблер;
- база знаний отсутствует…
- развитие продукта – версионное.

Второе направление (сборочное программирование, 1985 – 1995):

- переход от протопипа к промышленной версии регламентирован;
- язык спецификации регламентирован;
- язык разработки – формализованный язык высокого уровня со средствами модульности;
- ЯП объединен с языком разработки, допуская комплексирование с ассемблерными модулями.

---

1. А.П.Ершов: ВК ТП 1986, Ж. Программирование» 1986, №3

Третье направление (*доказательное программирование* [1]) **1995–2005 гг.**

Язык разработки формализован и содержит систему формальных преобразований, необходимых для доказательства программ…

Язык программирования объединен с языком разработки…

База данных проекта машинизирована.

Развитие продукта – эволюционное – *адаптивное*…

Заключение. **«Было бы полезно выработать норматив по технологии *второго поколения*, который, не затрагивая конкретного методологического или языкового наполнения, унифицировал бы: общую этапность разработки ПП; нормативы производительности и надежности; документационную структуру и вычислительную среду; *межмодульный интерфейс* поддержки *сборочного программирования…*».**

---

1. А.П.Ершов: ВК ТП 1986, Ж. Программирование» 1986, №3

Теория типов данных[1-3]

Теория структурной организации данных базируется на аксиоматическом подходе к определению типов данных.

Основу этой теории составляют типы, операции над ними и форма представления их на машине. Тип - это математическое понятие, которое задает множество значений элементов. Базовый тип (целое, действительное и др.) – тип, значение которого определяется аппаратурой, компиляторами с ЯП и присваивается переменной, а операции над значениями типа задаются аксиомами. Каждое значение типа "строится" из операций (например, операция «+» для переменных, матрицы), и это значение - конечное.

К операциям над типом относятся операции преобразования значений одного типа в значение другого типа, а также выбор элементов из сложного типа данных

---

1. Hoar К. О Структурной организации данных //Структурное программирование.– М.: Мир, 1975.– с.92 – 197.
2. Турский В. Методология программирования.–Пер.с анг. – Мир, Москва.–1981.–265с.
3. Агафонов В.Н. Типы и абстракция данных в языках программирования //Данные в языках программирования.—М.:Мир, 1982.— С. 267—327.
4. Замулин А.В. Типы данных в языках программирования и базах данных.— М.: Наука, 1987.— 152 с.

Лаврищева Е.М.

Фундаментальные и общие типы данных [1,2]

Тип — совокупность элементов, выделенных на всём множестве предметной области (R. Hindley, 1960).

Полиморфный тип — представление набора типов как единственного типа (R. Miller, 1960).

Математический тип определяется двумя способами:

1. Множеством всех значений, принадлежащих типу.
2. Предикатом, определяющим принадлежность объекта к данному типу.

Тип данных (ТД) появился в описании программ на ЯП (Algol, Пролог, PL/1, Fortran, Pascal и др.). Аксиоматика ТД разработана С.Дейкстрой, Н. Виртом, В. Турским, П. Науром, А.Замулиным, Н. Агафоновым и др. в 1970.

Любые данные, которыми оперируют программы в ЯП, относятся к следующим стандартным типам данных.

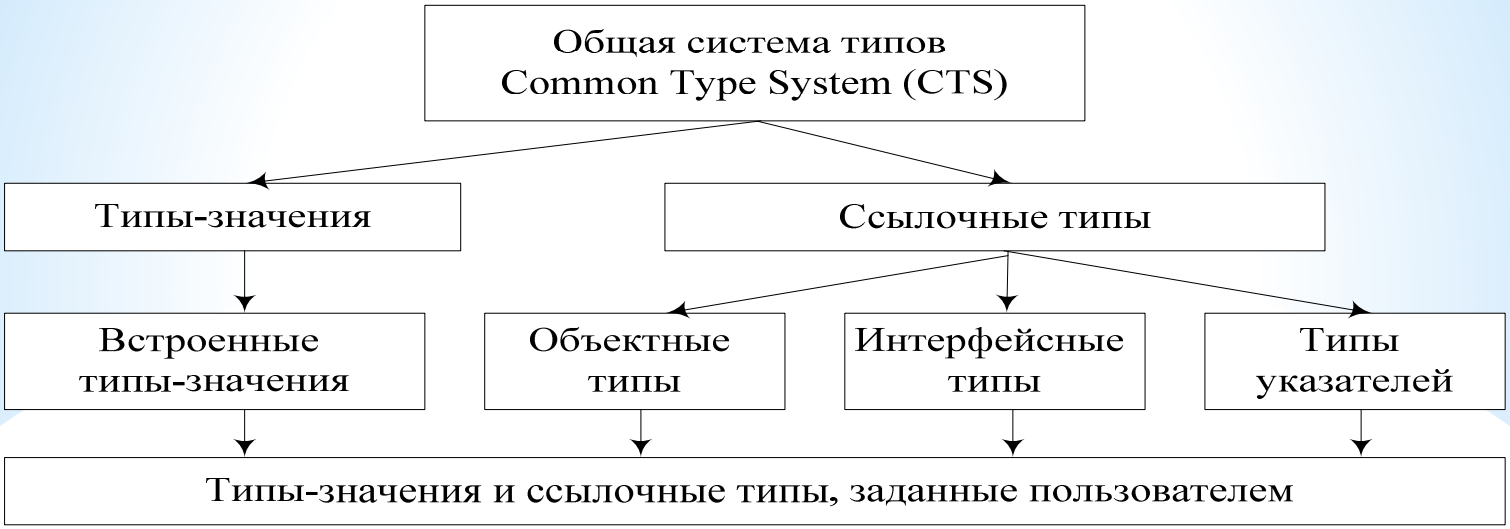
1. FDT – Fundamental Data Type. **Фундаментальные типы данных.**
2. GDT – General Data Types. **Общие типы данных** (ISO/IEC 11404 GDT).
3. Big Data – **Большие данные.**

1. Стеняшин А.Ю., Лаврищева Е.М. О формальном описании типов и структур данных в разнородных программах.– Проблемы программирования, №2, 2011.– с.50–61.

2. Лаврищева Е.М., Рыжов А.Г. Применение теории общих типов данных стандарта ISO/IEC 11404 GDT к Big Data.- XXX1 Межд. Научно-практ. конференция» Актуальные проблемы в современной науке и пути их решения» Евразийский союз ученых.- М.: Т31, 2018.-с. 99-108.

Лаврищева Е.М.

Система типов в .NET

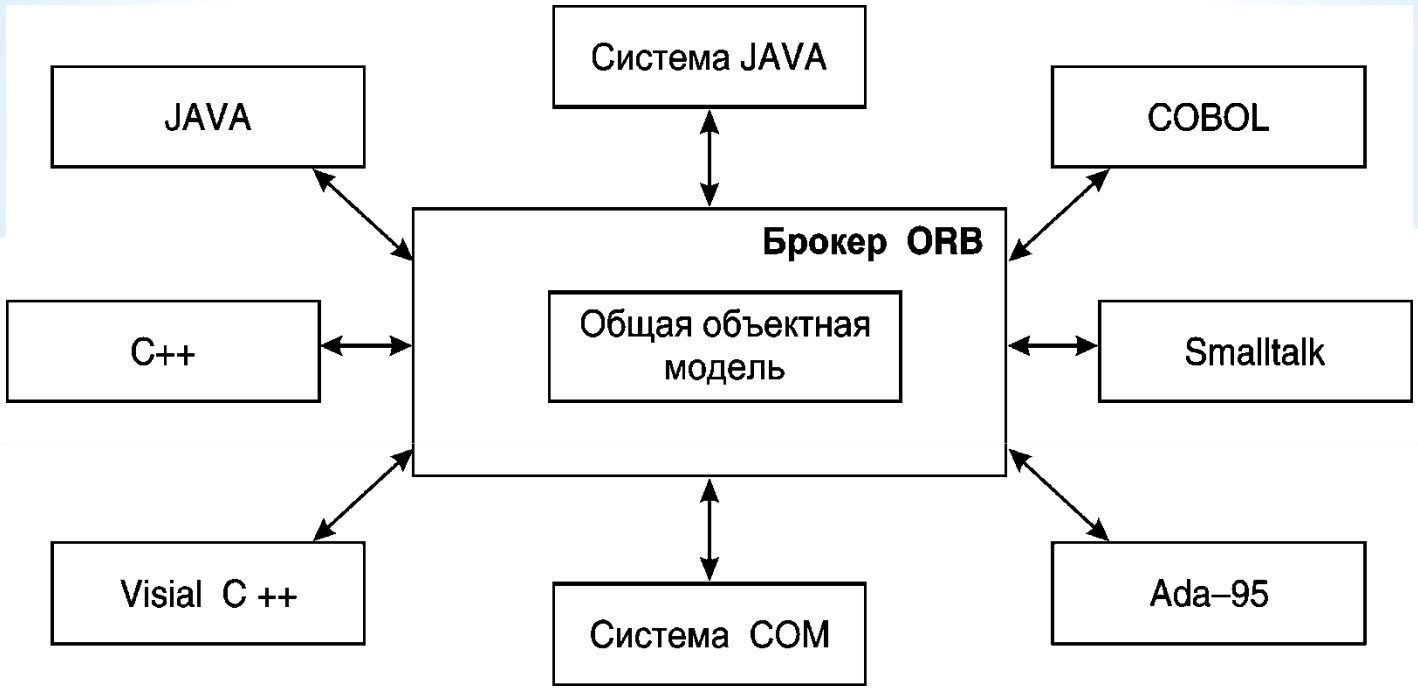


В этой системе две группы типов: *типы-значения* (value type) *и типы-ссылки* (reference type). *Типы-значения* – это статические типы в CTS, их значения могут занимать память от 8 до 128 байтов. Они копируются при присвоении им значений.

Типы-ссылки **используют указатели на объекты, которые они типизируют, а также механизмы хранения и освобождения памяти. Объекты этого типа – динамические, память под них выделяется из «кучи» и освобождается после уничтожения, «уборки мусора». Ссылочные типы включают: объектные типы (object type), интерфейсные типы (interface type), типы-указатели (pointer type).**

Система типов в системе CORBA[1-3]

CORBA реализует связь модулей в разных ЯП с помощью брокера ORB объектной модели и посредников Stub-клиента и Skeleton-сервера.



1. Лаврищева Е,М. , Грищенко В.Н. Связь разноязыковых модулей в ОС ЕС. М.: Фин. и стат. 1982, 136 с.

2. Эммерих В. Конструирование распределенных объектов в архитектурах OMG, CORBA, Microsoft COM, Java RMI, Мир,- 2002.-510 с.

3. Лаврищева Е,М., Грищенко В.Н.. Сборочное программирование. - К.: Наук.Думка.-1991.-256с.

Лаврищева Е.М.

3. ТЕОРИЯ ОБЪЕКТНО-КОМПОНЕНТНОГО МОДЕЛИРОВАНИЯ СИСТЕМ

Лаврищева Е.М. Теория объектно-компонентного моделирования программных систем.- www.ispras.ru/preprints/prep_29_2016.pdf

Основные понятия теории ООП[1]

Класс - совокупность объектов с общими свойствами.

Метод (или функция) - операция над экземплярами объектов класса.

Преемственность (Наследование) - сохранение атрибутов и операций родительского объекта для объектов класса.

Инкапсуляция - доступность к методам объекта и скрытие внутренней информации об особенностях реализации.

Полиморфизм - возможность оперировать с многими объектами без ограничений.

Создание из объектов систем с помощью ОМ архитектуры системы упрощает и процесс сборки объектов (КПИ, reuses, assets, artifacts и др.), а также дает возможность добавлять или удалять объекты на основе характеристической модели (МХ).

---

1. Буч Г. Объектно-ориентированный анализ.-Москва, Бином.- 1998.-500 с.

Принципы объектного моделирования

Принцип всеобщности означает, что при моделировании в ООП все сущности - суть объекты.

Следствие 1. Предметная область, которая моделируется, сама является объектом.

Следствие 2. Предметная область, которая моделируется, может быть отдельным объектом в составе другой предметной области (иерархия предметных областей).

Принцип существенности объектных различий. Каждый объект является уникальным элементом.

Следствие 3. Каждый объект имеет по крайней мере одно свойство или характеристику, которая задает его уникальную идентификацию во множестве объектов.

Принцип объектной упорядоченности. На произвольном шаге объектного анализа все объекты упорядочены в соответствии с отношениями между объектами.

Следствие 4. Каждый объект имеет одно отношение с другим объектом, которое обеспечивает его упорядоченность в рамках этой пары объектов.

Принцип целостности объектной модели. На произвольном шаге объектного моделирования совокупность объектов и отношений между ними однозначно определяют объектную модель предметной области для определенного уровня ее абстракции.

Следствие 5. Объектную модель можно представить в виде ориентированного связного графа, вершинами которого являются объекты, а дуги соответствуют отношениям между объектами.

Лаврищева Е.М.

Г.Буч - весь материальный мир состоит из объектов.

Любая предметная область – это совокупность объектов, связанных между собой некоторым множеством отношений и поведением в течение некоторого времени. То есть,

<объектная ориентация> = <объекты> + <наследование>.

Каждое понятие ПрО вместе с его свойствами и особенностями поведения является отдельным объектом.

В качестве объекта выступают как абстрактные образы, так и конкретные физические предметы или группы предметов с указанными общими характеристиками и функциями.

При определении объекта используется треугольник Фреге, согласно которому объект есть *денотат*. Символ (знак) используется для идентификации объекта, а денотат соответствует уровню знаний о сущности объектов моделируемого мира, которые отражаются этим объектом.

Денотат идентифицируется с помощью выбранного алфавита. Одному объекту могут соответствовать несколько концептов. Каждому треугольнику Фреге может соответствовать определенный объект с именем и содержанием.

Объектно-компонентный метод (ОКМ) [1]

ОКМ - метод проектирования ОМ системы из объектов, функции которых реализуются программными компонентами и интерфейсами. Метод включает:

- **математические операции (*объединения* ∪, *пересечения* ∩, *вычитания* ⊕, *симметричного вычитания* \ и др.) для формирования сложных объектов и отображения ОМ к компонентной модели (КМ);**
- **объектную и компонентную алгебры для изменения КПИ (reuses, assets, servises, artifacts);**
- **операции сборки (конфигурирования) КПИ и преобразования данных к разным форматам современных сред (VS.Net, IBM, Corba, Eclipse и др.) с помощью примитивов GDT ISO/IEC 11404–2007 к FDT ЯП (и обратно);**
- **линии (ProductLine, ProductFamily) для создания отдельных систем из семейства ПС за счет готовых КПИ, операций сборки и системы трансформации данных GDT↔FDT.**

---

1. Лаврищева Е.М. Теория объектно-компонентного моделирования программных систем.- www.ispras.ru/preprints/prep_29_2016.pdf.

Объект задается треугольником Фреге[1]

Name$_i$ **- имя объекта,**

Den$_i$ **- денотат – сущность реальной действительности,**

Con$_i$ **- концепт – семантика (смысл) денотата.**

1. Фреге Г. Логика и логическая семантика.- М.: Аспектпресс, 2000.

Лаврищева Е.М.

Объект по теории Фреге

$O_i = O'_i(Name_i, Den_i, Con_i)$,

входит во множество $O=(O_1, O_2,\ldots, O_n)$ объектов ПрО. Каждый объект задается треугольником Фреге, содержащим

$Name_i$ – знак (имя);

Den_i – денотат;

Con_i – концепт.

Концепты объектов определяются на множестве предикатов $P=(P_1, P_2, \ldots P_r)$:

$$Con_i = (P_{i1}, P_{i2}, \ldots, P_{is}).$$

Денотат может представляться в виде совокупности однородных или неоднородных предметов.

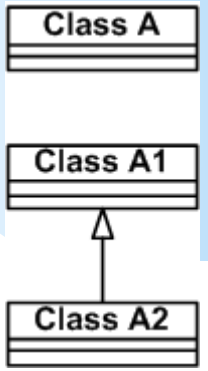
Эти совокупности могут изменяться *декомпозиционным* или *композиционным* путем.

***Декомпозиционное изменение денотата - это*:**

формирование новых однородных объектов:

$$decds(O_i): O_i \rightarrow \{O_{i1}, \ldots, O_{ik}\},$$

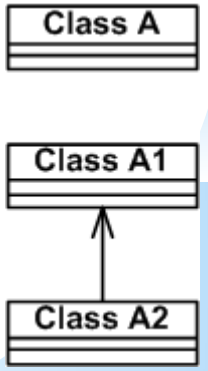
где $O_i = O_i(Name_i, Den_i, Con_i)$, $\forall\, j\ Con_{ij} = Con_i$; $Den_i = Den_{i1} \cup \ldots \cup Den_{ik}$.;



формирование новых неоднородных объектов:

$$decdn\ (O_i): O_i \rightarrow \{O_{i1}, \ldots, O_{ik}\},$$

где $O_i = O_i(Name_i, Den_i, Con_i)$, $\forall j\ Con_{ij} = \varnothing$; $Den_i = Den_{i1} \cup \ldots \cup Den_{ik}$.



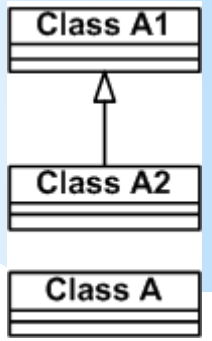
Лаврищева Е.М.

Композиционное изменение денотатов:

Композиция однородных объектов формирует новый объект:

$$comds(O_{i1}, \ldots, O_{ik}): \{O_{i1}, \ldots, O_{ik}\}, \rightarrow O_i,$$

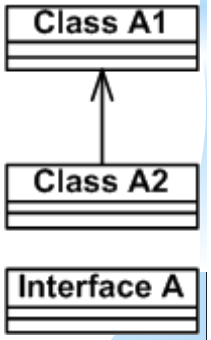
где $O_i = O_i(Name_i, Den_i, Con_i)$, $\forall\, j\ Con_i = Con_{ij}$; $Den_{i1} \cup \ldots \cup Den_{ik} = Den_i$.



Композиция неоднородных объектов формирует новый объект:

$$comdn(O_{i1}, \ldots O_{ik}): \{O_{i1}, \ldots, O_{ik}\}, \rightarrow O_i,$$

где $O_i = O_i(Name_i, Den_i, Con_i)$, $\forall\, j\ Con_i = \varnothing$; $Den_{i1} \cup \ldots \cup Den_{ik} = Den_i$.



Лаврищева Е.М.

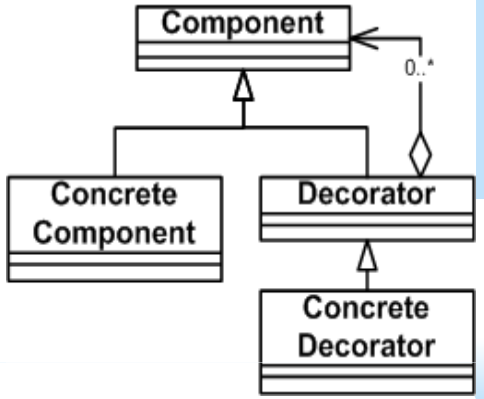
Расширение и сужение концепта

Расширение концепта существующего объекта

Если предикат $P_t \in P$, $P_t \notin Con_i$ и $P_t(O_i)$ принимает значение истины, то

$$conexp(O_i, P_t): O_i \rightarrow O'_i ,$$

где $O_i` = O_i`(Name_i, Den_i, Con_i`)$, $Con_{ij} \cup \{ P_t \} = Con_i$.

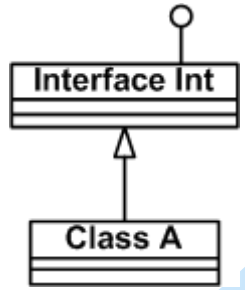


Сужение концепта существующего объекта

Если предикат $P_t \in Con_i$, *то*

$$connar(O_i, P_t): O_i \rightarrow O'_i , ,$$

где $O_i` = O_i`(Name_i, Den_i, Con_i`)$, $Con_{ij}' = Con_{ij} \setminus P_t$.



Лаврищева Е.М.

Формирование классов объектов и интерфейсов

Класс - это множество объектов, имеющих общие переменные, структуру и поведение. Объект – экземпляр класса. Его поведение определяется операциями создания, уничтожения и сериализации.

Совокупность внешних переменных и методов класса определяет **интерфейс**, с помощью которого экземпляры классов взаимодействуют между собой.

Инкапсуляция реализуется с помощью интерфейсов, которые состоят из методов объектов, атрибутов и параметров передачи (обмена) данных через внешние переменные экземпляра класса.

Для каждой внешней переменной существуют методы выборки значения переменной (**get-метод**) и присвоение ему нового значения (**set-метод**).

Интерфейс экземпляра объекта состоит из совокупности методов и их вызовов. Объект может иметь несколько интерфейсов, а также специальный интерфейс, методы которого работают с экземплярами (Home-интерфейс в модели EJB языка Java).

В состав этих методов входят методы поиска экземпляров объектов, создания, уничтожения и т.п. Поиск экземпляра объекта происходит по его **уникальному имени** или **значению** переменной.

Лаврищева Е.М.

Интерфейс объектов класса

Объект класса имеет:

- специальный интерфейс, методы которого работают с экземплярами объекта и непосредственно не определяют функциональных свойств экземпляров;
- один или несколько функциональных интерфейсов, которые реализуются в экземплярах объектов и определяют их функциональные свойства.

На абстрактном уровне интерфейс может рассматриваться как частичный вид абстрактного класса. Поэтому эта операция семантически может быть сведена к операции приведения классов. Если выполняется операция приведения экземпляра класса в соответствии с определенным интерфейсом, то обязательным условием является реализация этим экземпляром всех методов соответствующего интерфейса.

Утверждение 1: Любой экземпляр объекта обязательно является экземпляром определенного класса и имеет все методы, которые определены в этом классе. Экземпляр класса может быть приведен в соответствии с одним из суперклассов и интерфейсом, который реализуется в этом классе.

ОКМ моделирование ПС[1,2]

ОКМ обеспечивает представление домена в виде множества объектов со свойствами и характеристиками, которые необходимы и достаточны для их определения и идентификации, а также описания их поведения в рамках системы понятий и абстракций.

Объекты в ОКМ определяются на четырех уровнях проектирования с привлечением логико-математических формализмов описания ОМ и с уточнением функций объектов и их свойств и характеристик.

К формальным уровням представления ОМ из объектов относятся:

1. Обобщающий уровень
2. Структурный уровень
3. Характеристический уровень
4. Поведенческий уровень

1. Katerina Lavrischeva, Andrey Stenyashin, Andrii Kolesnyk. Object-Component Development of Application and Systems. Theory and Practice /Journal of Software Engineering and Applications, 2014, 7 , August 2014, http://www/.scirp.org/journal/jsea.

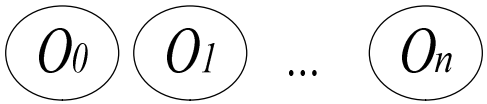
2. Лаврищева Е.М. Теория объектно-компонентного моделирования систем.-www.ispras.ru/prprint_29_2016.-48с.

Лаврищева Е.М.

Уровни логико-математического моделирования

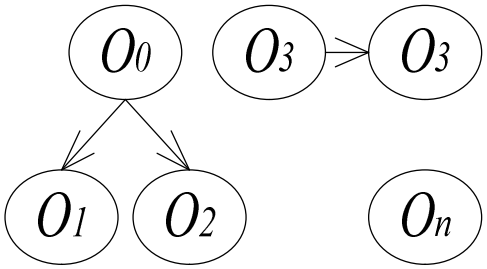
I. Generalized Level

Defines a set of objects



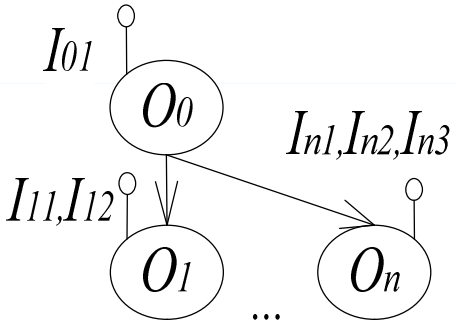
II. Structural Level

Defines hierarchy of objects



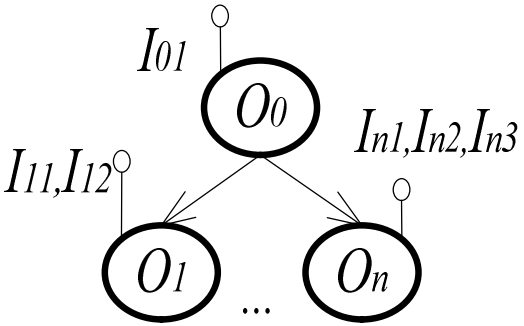
III. Characteristic Level

Defines a set of predicates for objects



IV. Behavioral Level

Defines a objects behavior



Лаврищева Е.М.

Теоретико-множественные операции ОКМ:

объединения $\cup$ ***(A, B)***, где *A* и *B* – объекты со статусом множеств. Результат применения операции - новый объект-множество, которое получено объединением множеств *A* и *B;*

пересечения $\cap$***(A, B)***, где *A* и *B* – объекты-множества. Результат применения операций: новый объект-множество, который является пересечением множеств *A* и *B*;

вычитания $D = A \oplus B$ при $B \subset A$. Результат применения операции: новый объект-множество со всеми элементами *A*, которые не входят в *B*;

симметричного вычитания $S = A \setminus B$, где $A \cap B$, и не $A \subset B$, не $B \subset A$. Результат применения операции – это новый объект-множество с множеством элементов, которые принадлежат *A* или *B*.

Лаврищева Е.М.

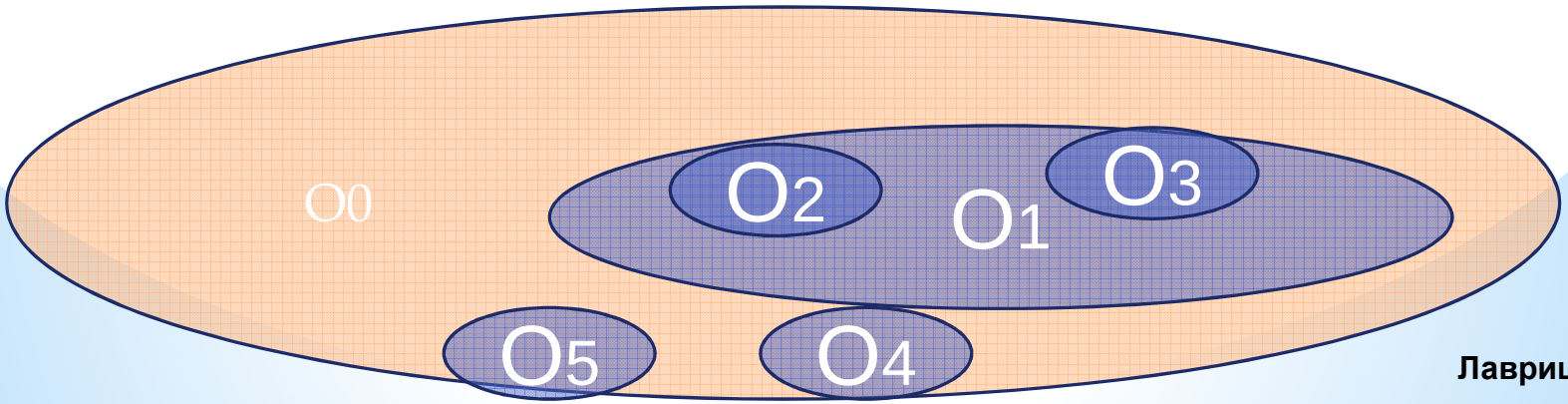
Обобщающий уровень

Объект – это класс или множество (класс рассматривается как соответствующее понятие аксиоматической теории множеств Геделя-Бернайса):

$$O = (O_0, O_1, \ldots O_n)$$

***O* – совокупность объектов предметной области,**
***O_0* – собственно предметная область.**

$$\forall\, i\ \exists\, j[(i>0)\&(\, j\geq 0)\&\ (i \neq j)\&\ (O_i \in\ O_j)].$$



Структурный уровень

Объекты обобщающего уровня представлены множеством элементов

$$O = (O_0, O_1, \ldots O_n).$$

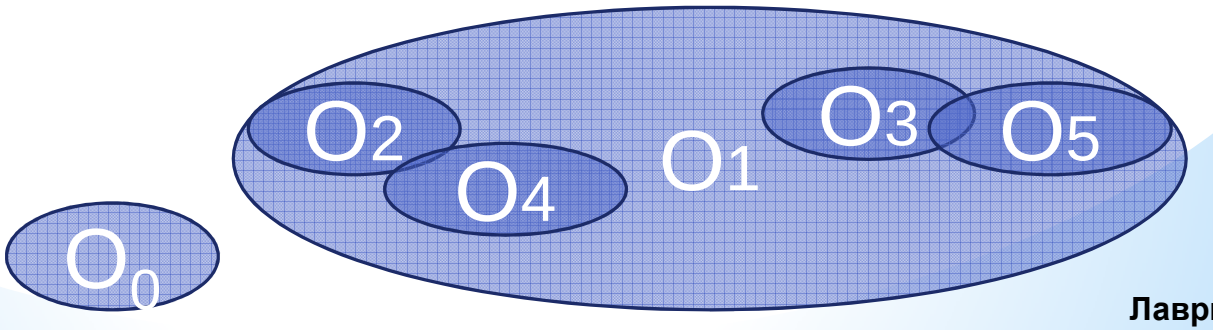
На структурном уровне, исключим элемент O_0 из множество O, получим новое множество

$$O' = (O_1, \ldots, O_n).$$

$$\forall\, i\ \exists\, j[(i>0)\&(j\geq 0)\&(i \neq j)\&(O_i \in O'_j)].$$

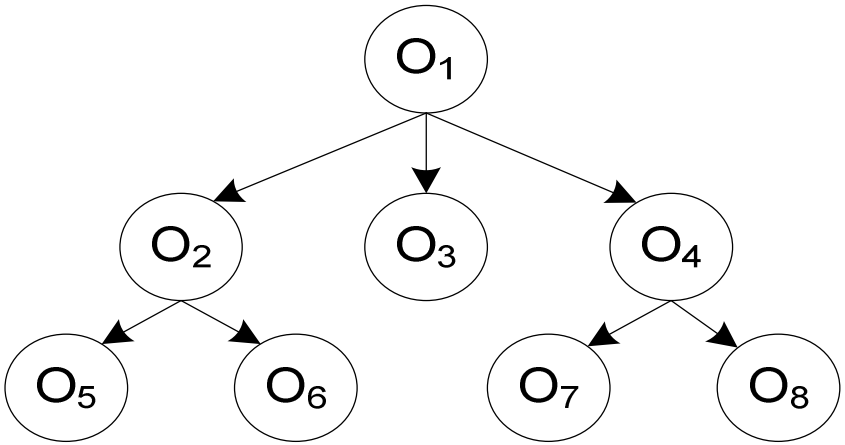
На множестве объектов O' определена алгебраическая система $\Sigma = (O', \Omega)$,

где Ω - совокупность теоретико-множественных операций ($\cup$, $\cap$, I, $\Diamond$, $\oplus$, – и др.).



Лаврищева Е.М.

Объектный граф *G*=(*O*) структурного уровня



отражает:

– множество вершин графа G =(O) и задает взаимно однозначное отображение множества объектов, определенных на обобщенном и структурном уровне описания ПрО;

– для каждой вершины существует хотя бы одна связь (структурная) с другой вершиной графа (стрелки);

– существует лишь одна вершина O_1 графа G, которая имеет статус множества объектов, отображающего ПрО в целом.

Лаврищева Е.М.

Характеристический уровень

Для каждого объекта формируется концепт с помощью множества унарных предикатов:
$P = (P_1, P_2, \ldots, P_r)$,
которые связаны со свойствами объектов домена.

Концепт $Con_i = \{P_{ik}\}$, $P_{ik} = P_k(O_i)$ = true.
Алгебраическая система $\Sigma = (O', \Omega)$ для характеристического уровня включает операции $\Omega = (O, P)$ определения характеристик объектов.

```
MyExemplar is IEnumerable
MyExemplar is IComparable
MyExemplar is ICloneable
MyExemplar is ICollection
MyExemplar is IAsyncResult
MyExemplar is IDisposable
```

На характеристическом уровне

выделяются характеристические свойства функций ПрО.
Аксиома. Каждый объект ПрО имеет хотя бы одну характеристику, которая задает семантику и уникальную идентификацию во множестве объектов ПрО.
Характеристики образуют множество свойств, каждое свойство принадлежат только одному объекту и задает статус объекта в структуре графа.
Определяются внешние и внутренние характеристики объектов.
***Внешние характеристики* служат для описания проблемной ориентации объектов.**
***Внутренние характеристики* предназначены для определения внутренних свойств объектов и удовлетворяет условиям:**
– объект-множество имеет внешнюю и внутреннюю характеристику;
– объект-элемент имеет только внутреннюю характеристику.
Формируется граф *G* ={O, *I*, *R*}. В нем *O* - множество объектов, *I* – множество интерфейсов, *R* - множество отношений (relations).

```
MyExemplar is IEnumerable
MyExemplar is IComparable
MyExemplar is ICloneable
```

```
MyExemplar is ICollection
MyExemplar is IAsyncResult
MyExemplar is IDisposable
```

Поведенческий уровень

Совокупность атрибутов объектов и их значений задает последовательность состояний объектов с помощью диаграмм переходов состояний.

Взаимосвязи между объектами задаются бинарными предикатами свойств объектов *O*, которые детализируют взаимосвязи между состояниями объектов.

Параметр времени *t* (*Timer*) вносится в объектную модель, который служит для рассылки специальных сообщений в текущий момент времени.

Каждый объект имеет метод, который анализирует полученное значение и выполняет переход к другому состоянию или оставляет его без изменений.

Лаврищева Е.М.

Граф G модели ОМ имеет вид:

$OM = \langle G_{t1}, G_{t2}, G_{t3}, G_{t4} \rangle$,

где G_{t1} – граф объектов ПрО на обобщающем уровне (t=1);

G_{t2} – граф характеристического уровня (t=2);

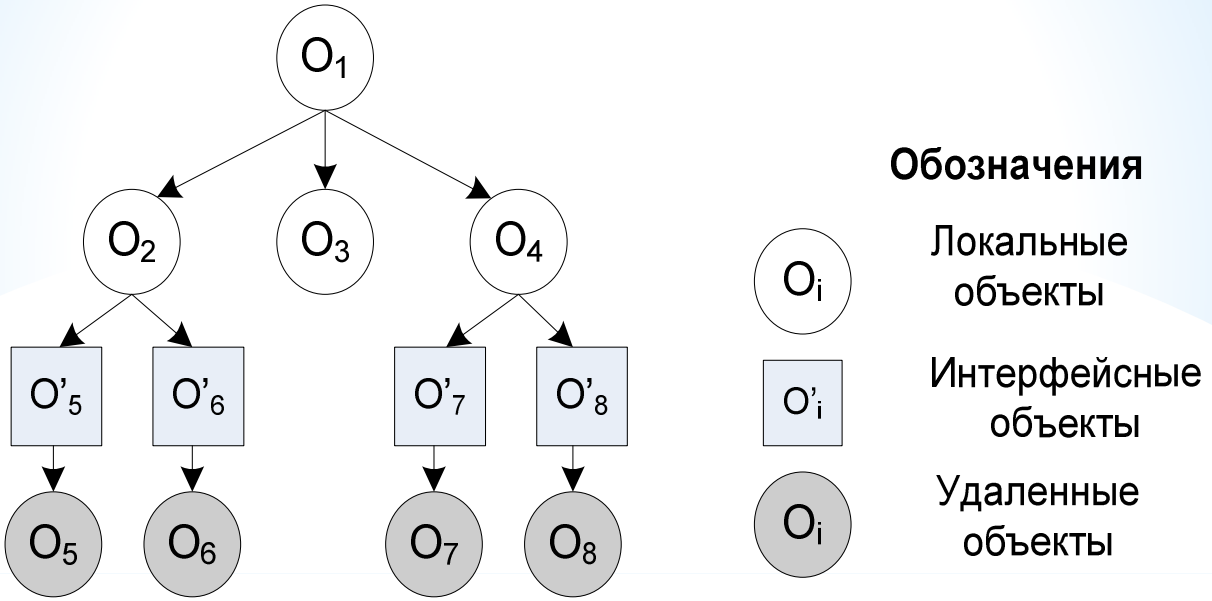
G_{t3} – граф структурного уровня (t=3);

G_{t4} – граф интерфейсов для взаимосвязи объектов на поведенческом уровне (t=4).

Объектам функций G_{t1} и их характеристикам соответствуют методы и данные (уровня 2, 3), необходимые для реализации отдельных систем из семейства СПС и обеспечения их взаимодействия.

Лаврищева Е.М.

Граф объектной модели с интерфейсами



Объекты графа связаны интерфейсными объектами (например, O'_5, O'_7), у которого множество входных интерфейсов совпадает с множеством выходных интерфейсов объекта-приемника, а множество исходных интерфейсов совпадает с множеством исходных интерфейсов объекта-передатчика.

Аксиома. Сборка объектов является корректной, если объект-передатчик полностью обеспечивает интерфейс, который необходим объекту-приемнику для выполнения функций объекта.

Объекты могут унаследовать интерфейсы других объектов, тогда последние предоставляют сервис для всего множества исходных объектов.

Лаврищева Е.М.

Задание программ по графу G

Вершины графа *G* задают:

- O_1, O_2, O_3, O_4 , O_5 , O_6 , O_7 , O_8 - функциональные объекты;

- O'_5, O'_6, O'_7, O'_8, - интерфейсные объекты.

Они размещаются в репозитории.

Элементы графа O_1 - O_8 описываются в ЯП (Fortran, Basic, C++ и др.), а интерфейсные объекты O'_5 - O'_8 - в языке IDL (Interface Definition Language).

По графу G можно задать программы P_1 - P_5 с использованием операции объединения :

$P_1 = O_2 \cup O_5$, *link* P_1 =*In* $O'_5 (O_2 \cup O_5)$;

$P_2 = O_2 \cup O_6$, *link* P_2 =*In* $O'_6 (O_2 \cup O_8)$;

P_3 ;

$P_4 = O_4 \cup O_7$, *link* P_4 =*In* $O'_7 (O_4 \cup O_7)$;

$P_5 = O_4 \cup O_8$, *link* P_4 =*In* $O'_8 (O_4 \cup O_8)$;

$P_0 = (P_1 \cup P_2 \cup P_3 \cup P_4 \cup P_5)$.

Лаврищева Е.М.

Алгебра анализа объектов

$$\Sigma = (O', \Psi, I, P),$$

где $O'=(O_1, O_2, \ldots, O_n)$ – множество объектов,

$\Psi = \{decds, decdn, comds, comdn, conexp, connar\}$ – операции замены денотат, расширения и сужения их концептов;

$I= (I_1, I_2, \ldots, I_k)$ – множество связей и отношений;

$P=(P_1, P_2, \ldots, P_r)$ – множество предикатов для определения свойств концептов объектов (например, $Con_i = (P_{i1}) = true/false$).

Теорема. Множество операций Ψ алгебры Σ есть система операций с функциями детализации, экземпляризации и агрегации.

Лаврищева Е.М.

Правила анализа объектов

все изменения в ОМ соответствуют процессам детализации описания предметной области в рамках представления объектов как треугольников Фреге;

- новые объекты на определенном уровне объектного анализа определяются как денотаты существующих объектов;
- все изменения в ОМ соответствуют условиям существования и определения формальных уровней абстракции представления объектов;
- функции объектного анализа определяются преобразованиями и изменениями объектной модели и ее отдельных элементов;
- на каждом уровне объектного анализа обеспечивается условие целостности объектной модели.

Операции над классами объектов

G - объектный граф может отражать связи и отношения между классами и экземплярами.

Каждый класс представлен в виде:

$Oclass_i = \{ClassName_i, Method_i, Field_i\}$,

где $ClassName_i$ - имя класса; $Method_i = \{Method_{ii}\}$ - множество методов; $Field_i = \{Fieldn_i\}$ - множество переменных, определяющих состояние экземпляров класса.

Если $Pfield_i \subset Field_i$ - множество внешних переменных (public) и каждому $Pfieldn_i \in Pfield_i$ поставим в соответствие методы get <$Pfieldn_i$> и set <$Pfieldn_i$> для присвоения и выборки значений соответствующей переменной. В других классах вместо непосредственного обращения к таким переменным будут использоваться указанные методы.

Тогда множество методов имеет вид:

$Imethod_i = Method_i \cup \{get<Pfieldn_i>\} \cup \{set<Pfieldn_{ji}>\}$, которому сопоставляется интерфейс $Ifunc_i$, состоящий из прототипов методов, входящих в $Imethod_i$.

4. ТЕОРИЯ ВАРИАБЕЛЬНОСТИ ПРОГРАММНЫХ СИСТЕМ

1. К.Pohl. Software Product Engineering Foundation .- Principles and Technigues /Springer Verlag, 2005

2. Лаврищева Е.М. Software engineering компьютерных систем. Парадигмы, Технологии, CASE- средства.- К.:2014.

Вариабельность[1,2] **– это свойство (системы/ продукта) к расширению, изменению, приспособлению или конфигурированию для использования в определенном контексте и обеспечения последующей его эволюции.**

Вариантная характеристика **содержит множество видов одной характеристики и образует их коллекцию. Каждый член коллекции соответствует требованию к артефактам ПС, СПС и присоединяется к СПС в точках вариантности.**

Точка вариантности **– это место в ПС, по которой осуществляется выбор варианта по точкам в системе. Вариантная характеристика транслируется в коллекцию вариантов с количеством точек вариантности в ПС. Объекты, помеченные вариантными точками, встраиваются в контексте ПС для обеспечения версии (или варианта) продукта.**

Внешняя изменяемость **– это изменение *области артефактов,* которая понятна заказчику при выборе вариантов, которые ему требуются.**

Внутренняя изменяемость **– изменение артефактов, которые скрыты от заказчиков и находятся в пределах обязанностей stakeholders, представляющих поставщика.**

1. Лаврищева Е.М. Software Engineering компьютерных систем. Теория, Парадигмы, CASE – средства.-К.:2014.-287с.

2. Лаврищева Е.М., Петренко А.К. Моделирование семейств программных систем.-Труды ИСП РАН. Том.28, вып.6..-с.49-65.

Лаврищева Е.М.

Модель Вариабельности ПС

Модель ПС – $M_{пс} = (CL, M_f, M_s, M_i, M_d)$, где
CL – отдельные элементы в $L = L_1, L_2, ..., L_N$;
$M_f = (O_1, O_2, ..., O_r)$ - множество функциональных элементов;
$M_s = (Ms_{in}, Ms_{out}, Ms_{inout})$ – множество интерфейсов IDL;
M_d - множество данных и метаданных ПС.

Модель вариабельности ПС – $MF_{var} = (SV, AV)$,
SV – подмодель вариабельности артефактов в структуре ПС;
AV – подмодель вариабельности готовых продуктов ПС.
Подмодель $SV = ((G_t, TR_t), Con, Dep)$, где $G_t = (F_t, LF_t)$ – граф артефактов типа t (требования, компоненты, артефакты, тесты);
TR_t – двусторонние связи артефактов типа t;
Con и Dep – предикаты, которые задают ограничения и зависимости.
Управление вариабельностью ПС (в артефактах и структуре) является основой разработки ПС из готовых ресурсов.

Лаврищева Е.М.

Модель вариабельности семейства СПС

Модель семейства систем SPS имеет вид:

$M_{SPS} = \{M_{PrO}, \{OM, KPV, PRG, RPC\}, M_{FM}, M_{var}, MK\}$, где

M_{PrO} – модель ПрО;

OM – объектная модель;

M_{FM} – модель характеристик (Feature Model);

KPV – множество КПИ;

PRG – предикат принадлежности к *KPV*;

RPC – сборочный предикат операции сборки КПИ в ПС;

M_{var} – модель вариабельности ПС;

MK – модель конфигурационной сборки.

Модель вариабельности СПС – это кортеж:

$SV_{СПС} = \langle\langle CF; \langle DR,TC\rangle; \langle CM,FR,TS,TA\rangle; \langle ER,TF\rangle\rangle; Con; Dep\rangle$,где

$CF = \langle SF, LF\rangle$ – характеристики ПС в вариантах LF;

$DR = \langle SF \cup SR, LF \cup LR\rangle$ – детализированные характеристики $f \in SF$ и $r \in SR$ с учетом требований к ПС и ограничений LR;

$TC = \langle (r,f), r \in SR, f \in SF)\rangle$ – связи между требованиями к ПС и их свойствами;

CM – множество формально описанных элементов и TS – тестов;

TA – интерфейсы элементов ПС, ER и TF – ER-модель БД.

Лаврищева Е.М.

Компонентная модель[1]

Компонентная модель (CM) **имеет вид:**

CM **=** $\langle RC, In, ImC, Fim \rangle$**, где**

RC **– базовые компоненты КПИ множества С, полученные после трансформации объектов модели ОМ;**

In **– множество интерфейсов для КПИ в точках вариантности;**

ImC **– множество реализованных базовых компонентов в среде;**

m (·) **– функции преобразования входных и выходных параметров интерфейсов во множестве данных ПС, СПС.**

Модель $M_{кпи}$ **=** (T, I, F, R, S)**, где**

T **– тип компонента,**

I **– интерфейс компонента;**

F **– функциональность,**

R **– реализация,**

S **– функциональный сервис для взаимосвязи с другими.**

1. Лаврищева Е.М. Компонентная теория и коллекция технологий для разработки индустриальных приложений из готовых ресурсов, Труды Четвертой научно-практической конференции «Актуальные проблемы системной и программной инженерии», АПСПИ-2015, 20-21мая 2015, с. 101-119.

Лаврищева Е.М.

Модели компонентного программирования:

Модель компонента. **Определяет типовые решения, касающиеся функциональной сущности компонента, его структуры, свойств и характеристик и имеет вид.**

$$MComp = (CName, CIn, CFact, CIm, CSer)$$, **где**

CName **– уникальное имя компонента;**

$CIn = \{CIni\}$ **– множество интерфейсов компонента;**

CFact **– управление экземплярами компонента;**

$CIm = \{CImj\}$ **– множество реализаций компонента;**

$CSer = \{CSer\}$ **– множество системных сервисов.**

Множество интерфейсов $CIn = CInI \cup CIno$ **состоит из входных** ***CInI*** **и выходных** ***CIno*** **интерфейсов.**

Модель интерфейса **компонента имеет вид:**

CIn **= (*InName , InFun , CIn, InSpec*), где**

InName **– имя интерфейса;**

InFun **– функциональность (метод) интерфейса;**

CIn **– интерфейс управления экземплярами компонента;**

InSpec **– спецификация интерфейса (описания типов, констант, сигнатур методов и т. д.).**

Интерфейс задает операции управления экземплярами:

CIn **= {*Locate, Create, Remove*}, где**

Locate **- поиск и определение экземпляра компонента;**

Create **- создание экземпляра компонента;**

Remove **- удаление экземпляра компонента.**

Компонентная среда – это множество серверов приложений, где разворачиваются компоненты, контейнеры, экземпляры которых обеспечивают реализацию функциональности компонента.

Модель компонентной среды имеет вид:

CE = (NameSpace, InRep, ImRep, CSer, CSerIm), где

NameSpace – множество имен компонентов среды;

$InRep = \{InRep_i\}$ – репозиторий интерфейсов;

$ImRep = \{ImRep_j\}$ – репозиторий реализаций;

$CSer = \{CSer_k\}$ – системные сервисы;

CSerIm = {CSerImr} – реализации сервисов.

Лаврищева Е.М.

Каркас компонентной среды

– это совокупность имен компонентов, интерфейсов и реализаций, заданная как пустые множества $FW = (\varnothing, \varnothing, \varnothing, CSer, CSerIm)$.

Если каркасы $FW_1 = (\varnothing, \varnothing, \varnothing, CSer_1, CSerImp_1)$ и $FW_2 = (\varnothing, \varnothing, \varnothing, CSer_2, CSerIm_2)$, то FW_1 совместим с FW_2, если существует отображение $SMap: CSer_1 \rightarrow CSer_2$ такое, что $SMap(CSer_1) \subseteq CSer_2$.

В $CSer_1$,$CSer_2$ входит сервис именования и между этими сервисами должна существовать такая связь, при которой сервис первой среды – это именование объектов среды и наоборот.

Любая компонентная среда использует те же сервисы, а их связи определяют их совместимость. Взаимодействие компонентов, расположенных в разных серверах, поддерживают сервисы этих серверов. Если совместимость между серверными сервисами существует, то такие компоненты могут входить в состав общей среды компонента.

Операции над компонентами:

Операция добавления компонента из множества С к компонентой среде обозначается $\oplus$. Добавление компонента $C \oplus CE1 = CE2$. Операция выполняется согласно правилам,

$CE2.NameSpace = \{C.CName\} \cup CE_1.NameSpace$,

$CE2.InRep = \{C.(CIni, CName)\} \cup CE_1.InRep$,

$CE2.ImRep = \{C.(Cj, CName)\} \cup CE_1.ImRep$.

Операция удаления компонента из компонентной среды обозначается знаком $\Diamond$ и подчиняется следующим правилам:

$CE1 \Diamond C = CE2$,

$\exists CName_k: CName_k \in CE1.NameSpace\ (CNamek = C.CName)$

$\Rightarrow CE2.NameSpace = CE1.NameSpace \setminus \{C.CName\}\ CE2.IntRep =$

$CE1.IntRep$.

$\forall i\ IntRepi.CName = C.CName)\ IntRepi\}\ CE2.ImpRep = CE1.ImRep \setminus$

$\forall j\ \&ImtRepj.CName = C.CName)\ ImRepj\}$.

Лаврищева Е.М.

Операция замены компонента задается знаком "–" и имеет вид:
$CE.NameSpace(C1) - C2 = (CE \lozenge C\ 1) \oplus C\ 2$.

Операция объединения ($\cup$) компонентных сред выполняется согласно следующим правилам:

$$CE1 \cup CE2 = CE3\ ,$$

$$CE3.NameSpace = CE1.NameSpace \cup CE2.NameSpace,$$

$$CE3.InRep = CE1.InRep \cup CE2.InRep,$$

$$CE3.ImRep = CE1,\ ImRep \cup CE2.ImRep.$$

Теорема. ***Операция объединения сред ассоциативна:***

$$(CE1 \cup CE2) \cup CE3 = CE1 \cup (CE2 \cup CE3).$$

Доказательство

$$\forall C \in (CE1 \cup CE2) \cup CE3 \Rightarrow C \in CE1\ v\ C \in CE2\ v\ C \in CE3;$$

$$\forall C \in CE1 \cup (CE2 \cup CE3) \Rightarrow C \in CE1\ v\ C \in CE2\ v\ C \in CE3.$$

Операция объединения сред коммутативна

$$CE1 \cup CE2 = CE2 \cup CE1.$$

Лаврищева Е.М.

Операции управления созданием ПС из компонентов в CASE-среде:

Link ($C_1 \cup C_2 \Rightarrow C_3$) – объединение компонентов;

Conf ($C_1 \cup C_2 \cup C_3$) – конфигурация компонентов;

Reing (C_n) – реинженерии и др.;

Prove PS (CPI_i) – доказательство правильности ПС из КПИ;

Creat (C_j) – образовать класс компонентов и др.

Компонентная модель CM имеет вид:

$CM = \{CLm\{Lm_1,\ldots, Lm_n\}, P\{P_i, \ldots, P_m\}, CL_n\{In_1,\ldots, In_k\}, Cd_i,$

где CL_m – компоненты из множества реализаций в языках L;

$P\{P_i, \ldots, P_m\}$ – множество предикатов, определяющих процессы сборки или конфигурации КПИ, реализаций компонентов CLm и интерфейсов In;

CL_n – множество интерфейсов компонентов;

CD_i – множество данных.

Лаврищева Е.М.

Компонентная алгебра – это

$$\sum = \{\varphi 1, \varphi 2, \varphi 3\},$$

где $\varphi 1 = \{CSet, CESet, \Omega 1\}$ – внешняя алгебра,

$\varphi 2 = \{CSet, CESet, \Omega 2\}$ – внутренняя алгебра,

$\varphi 3 = \{Set, CESet, \Omega 3\}$ – алгебра эволюции компонентов.

Внешняя компонентная алгебра включает операции над компонентами среды:

$$\varphi 1 = \{ CSet, CESet, \Omega 1 \},$$

где *CSet* – множество компонентов;

CESet – множество компонентных сред;

$\Omega 1$– множество операций:

$CSet2 = Cset \oplus CESet1$ - инсталляция (развертывание);

$CESet3 = CESet1 \cup CESet2$ - объединение сред ;

$CESet2 = CESet1 \setminus CSet$ удаления компонента из среды.

Лаврищева Е.М.

Внутренняя компонентная алгебра **имеет вид:**

$$\varphi 2 = \{CSet, CESet, \Omega 2\},$$

где *Cset = {OldC, NewC}* **– множества старых** *OldC* **и новых компонентов** *NewC,* **разработанных или преобразованных из старого к** *NewC;*

OldC = (*OldCName,OldIn, CFact, OldIm, CSer*) **– интерфейсы реализации старых компонентов в серверной среде;**

NewC = (*NewCName, NewIn, CFact, NewIm,CSer*) **– интерфейсы реализации новых компонентов в серверной среде;**

$\Omega 2$ = {*addIm, addIn, replIn, replIm*},

где *addim* **– операции добавления реализации;**

addIn **– добавления интерфейса;**

replIm **– операция замещения реализации компонента,**

replIm **– замещения интерфейса.**

Алгебра эволюционного изменения **имеет вид:**

$$\varphi 3 = \{CSet, CESet, \Omega 3\},$$

где $\Omega 3 = \{O_{refac}, O_{Reing}, O_{Rever}\}$**, в котором**

O_{refac} **– рефакторинг,**

O_{Reing} **– реинженерия,**

O_{Rever} **– реверсная инженерия компонентов.**

Семантика этих операций состоит во внесении изменений (замены, добавления, переименования) компонентов и/или их интерфейсов и добавления новых компонентов.

Модель рефакторинга **компонентов:**

$$M_{refac} = \{O_{refac}, \{CSet = \{NewCn\}\},$$

где *CSet*, O_{refac} **– элемент алгебры эволюции;**

O_{refac} = {*AddOImp, AddNIm, ReplIm, AddIn*} **– операции рефакторинга:**

AddOIm **– добавить реализацию компонента;**

AddNIm **– добавить новую реализация;**

AddIn **– добавить интерфейс во множество** *CSet*;

ReplIm **– заменить реализацию компонентов в репозитории.**

Лаврищева Е.М.

Модель реинженерии компонентов:

$M_{Reing} = \{O_{Reing}, \{CSet = \{NewC_n\}\}$, где

CSet, O_{Reing} – элемент алгебры эволюции функций;

O_{Reing}= {rewrite, restruc, adop, conver } – операции реинженерии

Rewrite – перепись компонента;

Restruc – реструктуризация структуры компонента;

Adop – адаптация компонента к среде выполнения;

Conver – конвертирование компонента в другой язык.

Модель реверсной инженерии компонентов:

$M_{Rever} = \{O_{Rever}, , \{CSet = \{NewCn\}\}$, где

CSet, O_{Rever} – элемент компонентной алгебры эволюции;

O_{Rever} = {Restruc, Design}– операции реверсной инженерии,

Design – проектирование компонента или системы;

Restruc – трансформация структуры системы.

Лаврищева Е.М.

Обработка компонентов

Члены множества С компонентов помещаются в репозитории на разных языках, с их именами и интерфейсами. Они могут изменяться и заменяться новыми компонентами с целью получения разных вариантов продуктов ПС.

Аксиома. Два компонента С1 и *С2* являются тождественными (равными), если тождественны их соответствующие составные.

Как следствие, замена С1 на С2 не влияет на компонентную модель, к которой принадлежит компонент С1.

Средства описания КПИ:

- язык спецификации интерфейсов КПИ (IDL, API, OSWL, XML)

- сборка компонентов и КПВ в приложения, домены, семейства программ и систем;

- представление КПИ из разных предметных областей в репозиторий.

Лаврищева Е.М.

Базовые компоненты сборки

1. Модель сборки КПИ в ПС, СПС (описание интерфейсных типов данных в IDL, вызовов RPC, RMI).

- функции изменения ФТД к форматам КПВ другой среды, тестирование экспорта и импорта интерфейсных данных и др.

- алгебра преобразования типов данных (библиотека функций изоморфного отображения нерелевантных типов данных ЯП и отличий в форматах frameworks).

- межъязыковый и межмодульный интерфейс взаимосвязи ЯП и программ в классе ЯП (средства описания типов данных в ЯП, операций вызовов и др.).

- система генерации модулей-посредников в GDM, CORBA.

2. Модель сборки Common Language Runtime в .NET (описание компонентов, манифеста сборки и имен), разворачивание приложений, содержащих манифест.

- описание метаданных в манифесте (типов, свойств, методов, аргументов, атрибутов, базовых классов и т.п.) и необходимых ресурсов.

- описание компонентов в VBasic, C++, C# для сборки (версия, варианты, типы и т.д.).

- идентификация типов объектов ЯП в пространстве имен подобно типов данных.

Microsoft Intermediate Language (MSIL, IL) - компиляция семейства языков в IL или управляемый код для сборки.

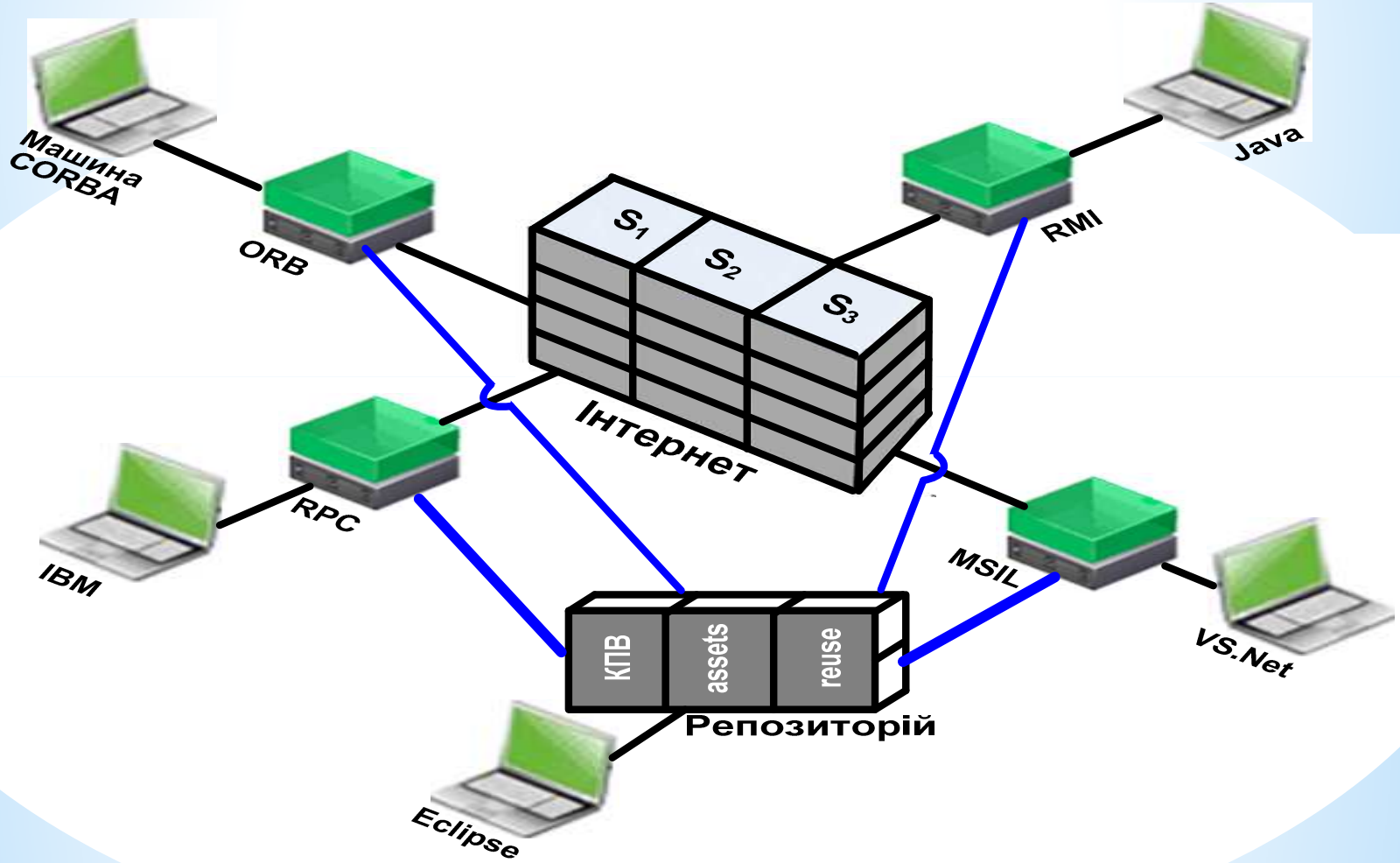
Лаврищева Е.М.

5. ТЕОРИЯ ВЗАИМОДЕЙСТВИЯ СИСТЕМ

1. Бей.И. Взаимодействие разноязыковых программ. Диалектика.-М.,СПБ-Киев.-Мир.-2005.
2. Лаврищева Е.М. Взаимодействие программ, систем и операционных сред.- Проблемы программирования.- К.-2011, №3.-13-24.

Взаимодействие систем и сред [1]

основано на интерфейсе и механизмах обработки данных, которые передаются по сети с общих и глобальных хранилищ типа Grid и Cloud.



1. Lavrischeva K.: Formal Fundamentals of Component Interoperability in Programming. In: Cybernetics and Systems Analysis, vol. 46, no. 4, pp. 639–652, Heidelberg (2010)

2. Лаврищева К.М. Взаимодействие программ, систем и операционных сред// Проблемы программирования. – 2011 – №3 – С. 11–23.

Лаврищева Е.М.

Взаимодействие – это взаимосвязь двух и больше объектов или систем

Модель взаимодействия имеет вид:

***Мвз* = {*Мпр, Мсис, Мсред*}, где**

***Мпр* = {*Com, Int, Pro*} – модель программы,**

***Мсис* = {*FPC, Int, Pro*} – модель системы,**

***Мсред* = {*Envir, Int, Pro*} – модель среды,**

***Int, Pro* - совокупность интерфейсов и протоколов, которые передают данные между программами разных сред сети.**

***Мвз* по отношению к стандартной модели открытых систем OSI является моделью верхнего уровня, которая реализуется сервисом Eclipse в Интернет. Разработаны модели [1]:**

Модель Visual Studio ↔ Eclipse

Модель CORBA ↔ Visual Studio ↔ Eclipse

Модель JAVA ↔ Visual Studio ↔ Eclipse

---

1. Островский А.В. Подход к обеспечению взаимодействия сред Java MS.Net // Проблемы программирования. – 2011. – № 2. – С. 37–44.

2. Лаврищева Е.М. Программная инженерия. Технология программирования.- Уч. Пособие .- МФТИ, 2016.-52 с.

Модель Visual Studio ↔ Eclipse [1,2]

Взаимодействие систем стандартизовано в модели OSI (Open System Interconnection, 1994) открытых систем. Средства взаимодействия определены на семи уровнях: прикладном, представительном, сеансовом, транспортном, сетевом, канальном и физическом.

Механизмы взаимодействия реализованы в современных операционных средах (VS.Net, CORBA, Eclipse, JAVA и др.). Приложения, изготовленные в этих средах, не могут переноситься в другую среду.

Основу взаимодействия составляет интерфейс и методы доступа к данным сетевой среды. Программы, изготовленные в одной из сред, могут быть перенесены из одной среды в другую через предложенные механизмы взаимодействия в п1.

1. Радецький И.. А. Один подход к обеспечению взаимодействия сред MS.Net и Eclipse // Проблемы программирования, № 2, 2011.– с. 45–52.
2. Лаврищева Е.М., Карпов Л.В., Томилин А.Н. Системная поддержка бизнес задач в глобальной информационной сети. -Труды конференции «Научный сервис в сети Интернет -2015», 21-26 сентября 2015, Новоросийск, с. 193-218.

Лаврищева Е.М.

Модель CORBA ↔ Visual Studio ↔ Eclipse

Взаимодействие систем реализовано в модели OSI (Open System Interconnection) открытых систем. Средства взаимодействия определены на семи уровнях: прикладной, представительный, сеансовый, транспортный, сетевой, канальный и физический.

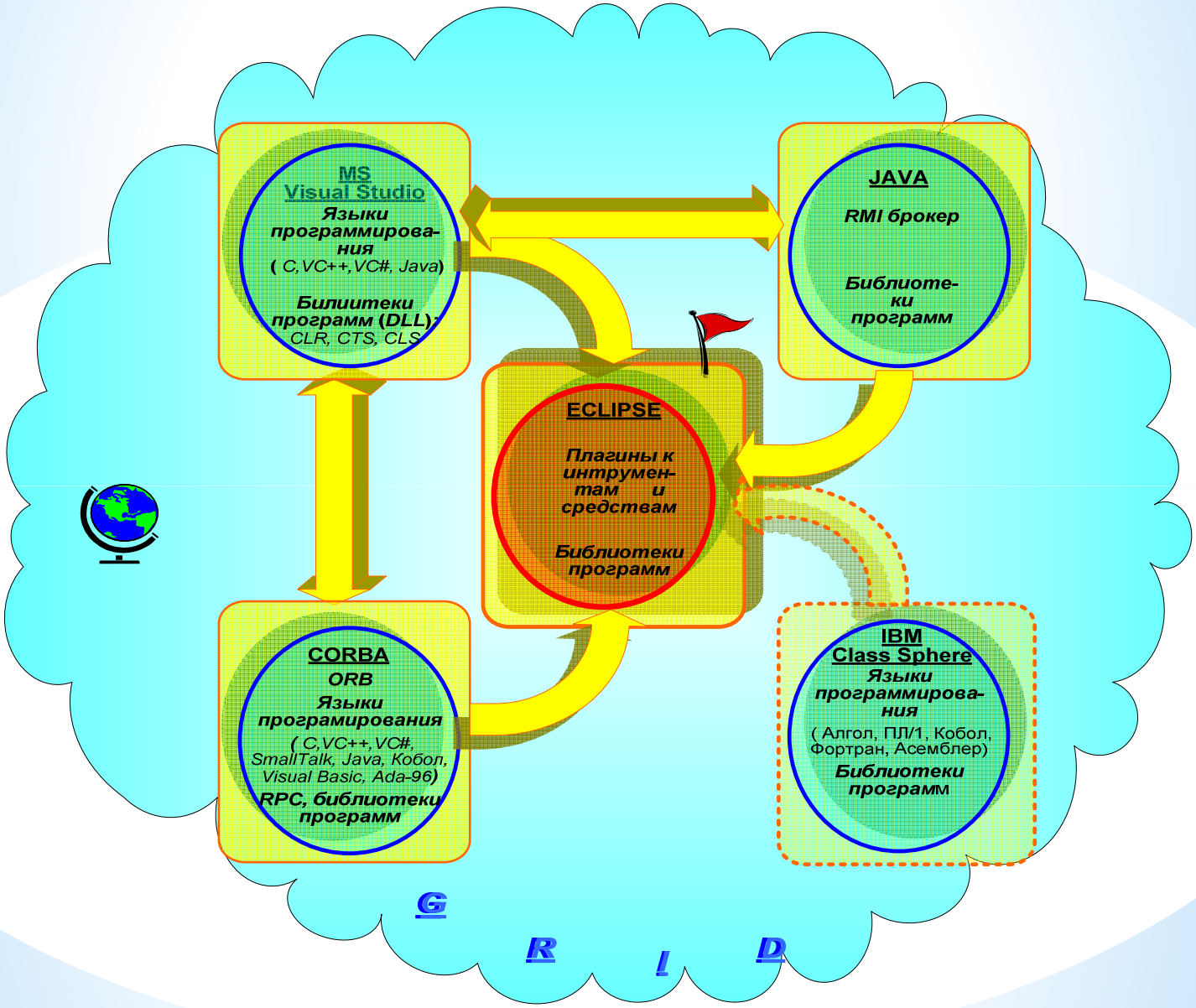
Механизмы взаимодействия реализованы в современных операционных средах (VS.Net, CORBA, Eclipse, JAVA и др.). Приложения, изготовленные в этих средах, не могут переноситься в другую среду.

Visual Studio.Net↔Eclipse – это среда для разработки отдельных программ в языке C# и спецификации интерфейса для переноса готового продукта в репозиторий Eclipse.

Основу их взаимодействия составляет интерфейс и методы доступа к данным сетевой среде. Программы, изготовленные в одной из сред, могут быть перенесены из среды Visual Studio.Net в среду Eclipse через механизмы взаимодействия реализованные в этих средах.

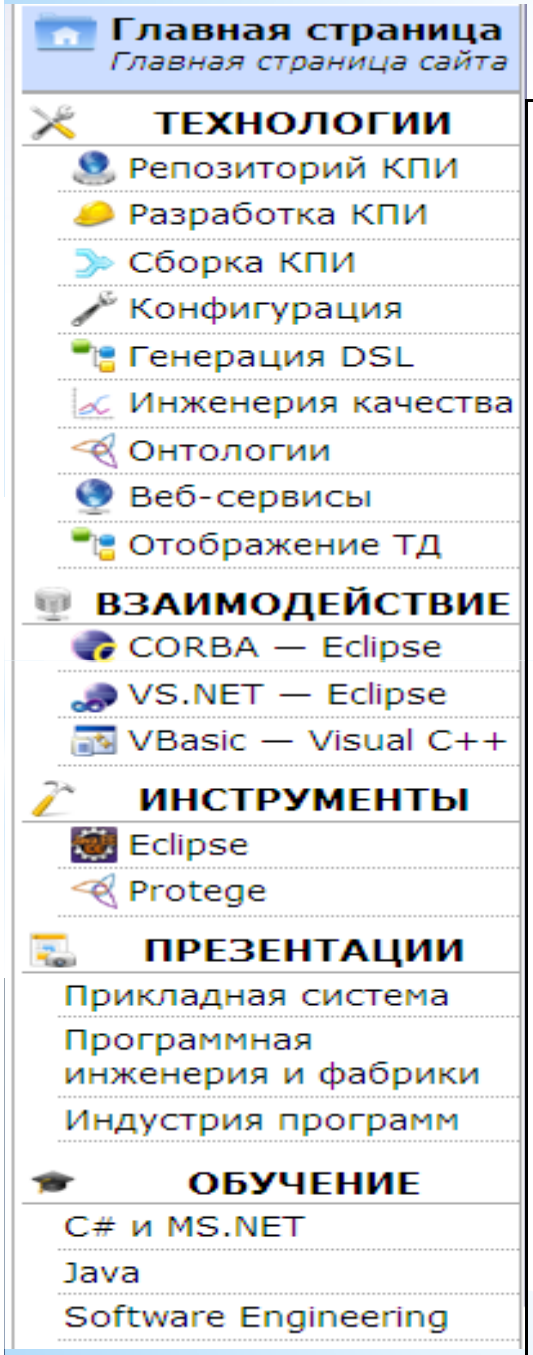
Лаврищева Е.М.

Общая модель взаимодействия современных сред



Е.М.Лаврищева

Содержание разделов веб-сайта ИТК (http://7dragons.ru)



ТЕХНОЛОГИИ ИТК:

Технология обслуживания репозиторию КПИ,

Технология разработки КПИ,

Технология сборки КПИ,

Технология конфигурирования КПИ,

Технология генерации описания КПИ в языке DSL,

Технология оценка затрат и качества,

Технология онтологии вычислит. геометрии, ЖЦ ISO/IEC12207

Технология веб-сервисов,

Технология генерация типов данных ISO/IEC 11404.

ВЗАИМОДЕЙСТВИЕ программ, систем и сред:

Модель Corba–Eclipse–Java,

Модель VS.Net C#–Eclipse,

Модель Basic–C++.

ИНСТРУМЕНТЫ ИТК:

Система Eclipse,

Система Protege

ПРЕЗЕНТАЦИИ В ИТК:

Система ведения зарубежных командировок для НАНУ,

Слайды про ИТК, фабрики программ

Методологии построения ТЛ

ОБУЧЕНИЕ:

технологии разработки программ на C# VS.Net, языке Java,

электронный учебник по предмету «Программная инженерия»

веб-сайт КНУ http://programsfactory.univ.kiev.ua.

Dear Dr. Lavrischeva Ekaterina! **4.11.2016**

I am writing with reference to your research paper titled, **“Object-Component Development of Application and Systems Theory and Practice”** and have acknowledged it worthy of commendation. I found your research work to be exceedingly impressive and impactful. This research can indeed prove to be significant for fellow researchers and scientists working in the same domain.

Taking note of your research interests which matches with our journal domain, I would like to welcome you to associate with us. To follow this, our Editorial Board has agreed to recognise you under **"Quarterly Franklin Membership"** (Membership ID#513442|UK) of London Journals Press. Also, we encourage you to have your upcoming research articles/papers published in, **London Journal of Research in Computer Science and Technology (LJCST)**. As a part of this honorary membership, APC/Membership fee is fully waived off for you.

London Journals Press (U.K.) is an internationally acclaimed publication organisation and an accreditation authority in research standards. We follow COPE and Research Councils UK standards and are in association with researchers in all the leading discipline like Computer Science, Engineering, Management, Medical Research, Science, Social Science, etc.

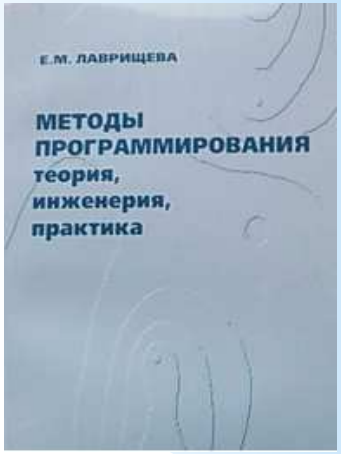
Yours Fainthfully
Dr.Wael

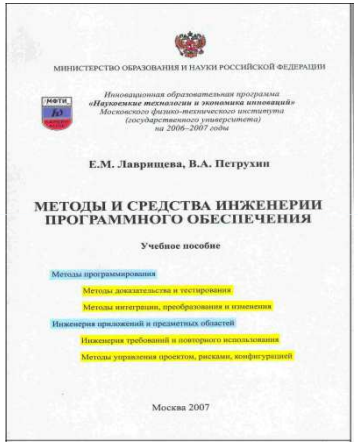
London Journals Press

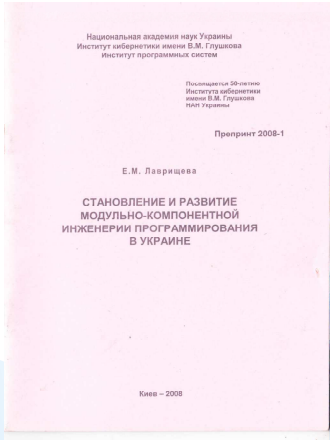
Е.М.Лаврищева

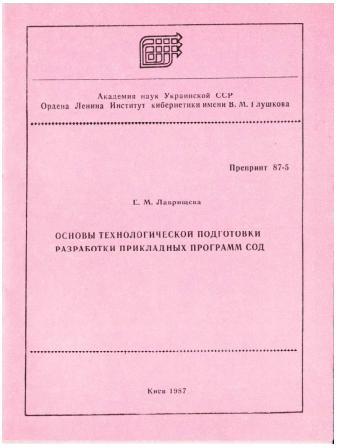
Теории и методы программ и технологий и инженерии систем опубликованы в книгах, учебниках и препринтах и статьях

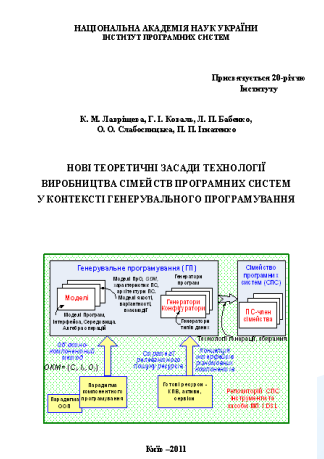
Е.М.Лаврищева











* Е.М.Лаврищева

БЛАГОДАРЮ ЗА ВНИМАНИЕ!

Е.М.Лаврищева