Содержание:

ВВЕДЕНИЕ

Актуальность исследования. В последние годы в программировании четко прослеживалось становление и развитие объектноориентированного проектирования (ООП) программных систем (ПС). Определилась не только технологическая база разработки разных видов ПС, но и инструментальные поддержки (Rational Rose, Rational Software, RUP, Demral , OOram и др.).

Совершенствование ООП достигло такого уровня, что он стал базисом генерирующего (порождающего) программирования, в котором определены пути его развития и устранения ряда недостатков, связанных не только с особенностью проектирования уникальных ООП ПС, но и с отсутствием механизмов использования готовых компонентов, свойств изменчивости, взаимодействия, синхронизации и др.[3] В данном программировании объединены и другие концепции и методы программирования с целью обеспечения инженерии предметной области (ПрО), ориентированной на проектирование семейств ПС из объектов, компонентов, аспектов, сервисов, повторного использования компонентов (ПИК), систем, характеристик и т.п. Все это расширило рамки всех концепций и положило начало объединенной целостной и стройной технологии генерации как отдельных ПС, так и их семейств. Порождающее программирование заложило базис будущего программирования, основанный на современных методах программирования, новых формализмах и объединяющих моделях, посредством которых новое поколение программистов будут создавать более долговечные и качественные программные изделия семейств ПС с использованием конвейерной автоматизации.

Целью данной работы является анализ методов современного программирования, для достижения поставленной цели, были выделены следующие задачи:

- рассмотреть теоретические аспекты систем и языков программирования;

- изучить современные методы программирования.

Объект исследования - методы программирования.

Предмет исследования - современные методы программирования.

Структура работы состоит из введения, основной части, заключения и списка литературы.

Теоретической и методологической базой данной работы послужили труды российских и зарубежных авторов в области информатики, материалы периодических изданий и сети Интернет.

ГЛАВА 1 ТЕОРЕТИЧЕСКИЕ АСПЕКТЫ СИСТЕМ И ЯЗЫКОВ ПРОГРАММИРОВАНИЯ

1.1 Определение и назначение языков и систем программирования

В процессе своей работы компьютер производит выполнение инструкций различных программ, т.е. набора определённых команд следующих в определённом порядке. Машинная команда, доступная для восприятия и выполнения компьютером, записана в виде двоичного кода, т.е. состоит из нулей и единиц, указывает, какое именно действие должен выполнить центральный процессор. Следовательно, для того чтобы задать компьютеру последовательность действий, необходимых для выполнения, требуется задать последовательность двоичных кодов соответствующих команд. Программы в машинных кодах состоят из сотен и тысяч команд и их написание очень сложный процесс. Программисту требуется знать и помнить комбинацию нулей и единиц двоичного кода каждой программы, а также двоичные коды адресов данных, используемых при её выполнении. Значительно проще для программиста создавать программу на языке близком к естественному человеческому языку, а затем переводить полученный программный код в машинные команды. Перевод текста программы, записанной на языке близком к естественному человеческому языку, производится с помощью специальных программ-трансляторов. При этом перевод программного текста в исполнимый код, записанный на машинном языке, производится по *определенным* правилам путем сопоставления и дальнейшей замены команды языка программирования на соответствующую ей команду или серию команд на машинном языке.

Таким образом, любой язык программирования – это формальная знаковая система, предназначенная для описания алгоритмов в форме, которая удобна для

исполнителя (компьютера).

Язык программирования определяет набор правил, используемых при составлении компьютерной программы. Каждый язык программирования имеет алфавит, словарный запас, свою грамматику и синтаксис, а также семантику.

Алфавит – фиксированный для данного языка набор основных символов, допускаемых для составления текста программы на этом языке.

Синтаксис – система правил, определяющих допустимые конструкции языка программирования из букв алфавита. Синтаксис языка программирования устанавливает жесткие правила, которым должна удовлетворять запись кода.

Семантика – система правил однозначного толкования отдельных языковых конструкций, позволяющих воспроизвести процесс обработки данных. Другими словами можно сказать, что семантика - совокупность правил, определяющих смысл синтаксически корректных конструкций языка, его содержание. Семантику языка программирования закладывают в его компилятор. Таким образом, синтаксически корректная программа, написанная на языке программирования, после преобразования ее в последовательность машинных команд обеспечит выполнение ЭВМ требуемых операций.

Язык программирования позволяет программисту точно определить то, на какие события будет реагировать компьютер, как будут храниться и передаваться данные, а также какие именно действия следует выполнять над этими данными при различных обстоятельствах. Алгоритмические конструкции в языке программирования записываются с помощью соответствующих *операторов*. Информация, подаваемая на вход программе, называется *данными*.

Языки программирования являются ядром систем программирования.

Система программирования – это комплекс инструментальных программных средств, предназначенный для работы с программами на одном из языков программирования. Системы программирования представляют разработчикам сервисные возможности, необходимые для разработки прикладных и системных программ.

В общем состав систем программирования представляют:

- трансляторы с языков высокого уровня;

- специализированные средства, предназначенные для редактирования, компоновки и загрузки программ;
- макроассемблеры (машинно-ориентированные языки);
- средства отладки программ.

В состав современных систем программирования входят:

- Текстовый редактор (*меню Edit*), осуществляет функции записи и редактирования исходного текста программы;
- Загрузчик программ (*меню File – Open …*), позволяет выбирать из директории нужный текстовый файл программы;
- «Запускатель» программ (*меню Run*);
- Компилятор (*меню Compile*), предназначен для компиляции или интерпретации исходного текста программы в машинный код с диагностикой синтаксических и логических ошибок;
- Отладчик (*меню Debu*g), выполняет сервисные функции по отладке и тестированию программы;
- Диспетчер файлов (*меню File*), предоставляет возможность выполнения операций с файлами: сохранение, поиск, уничтожение и т.д.

Помимо перечисленных традиционных сервисов систем программирования многие системы программирования включают в себя:

- *Search* – поиск и замена фрагментов в тексте;
- *Tools* – набор инструментальных программных средств;
- *Options* – установка опций интегрированной среды;
- *Window* – работа с окнами.

Современные системы программирования (среды программирования) являются по своей функциональности интегрированными средами разработки IDE - integrated development environment. Самая первая среда разработки была создана для работы с языком программирования Basic.

Различают два класса систем программирования:

1. системы программирования общего назначения;
2. языково-ориентированные системы (среды) программирования.

Системы программирования

Системы программирования общего назначения

Языково-ориентированные системы программирования

Интерпретирующие и компилирующие системы программирования

Синтаксически-управляемые системы программирования

Рис. 1 Классификация систем программирования

Системы программирования *общего назначения* содержат набор программных инструментов, поддерживающих разработку программ на разных языках программирования.

Языково-ориентированная система программирования предназначена для поддержки разработки программы на каком-либо одном языке программирования, и знания об этом языке существенно использовались при построении такой системы. Вследствие этого в такой среде могут быть доступны достаточно мощные возможности, учитывающие специфику данного языка. Такие системы программирования разделяются на два подкласса:

1. Интерпретирующие и компилирующие системы;
2. Синтаксически-управляемые системы программирования.

Интерпретирующие и компилирующие системы программирования имеют встроенный текстовый редактор и обеспечивают трансляцию (интерпретацию или компиляцию) программы, записанной в редакторе, на данном языке программирования.

Синтаксически-управляемая система программирования уже на этапе написания текста программы использует знание синтаксиса языка программирования, на который она ориентирована. В такой системе вместо обычного текстового редактора используется синтаксически-управляемый редактор, позволяющий пользователю использовать различные шаблоны синтаксических конструкций. Результатом применения такого редактора разрабатываемая программа по окончании написания всегда является синтаксически правильной.

1.2 Классификация и история развития языков программирования

Первые языки программирования были очень примитивными и мало чем отличались от формализованных упорядоченных последовательностей единиц и

нулей, понятных компьютеру. Такие языки программирования называют «***машинными***». Использование таких языков было крайне неудобно с точки зрения программиста, так как он должен был знать числовые коды всех машинных команд, должен был сам распределять память под команды программы и данные.

Для того чтобы облегчить общение человека с ЭВМ были созданы ***машинно-ориентированные*** языки программирования типа Ассемблер. Переменные величины стали изображаться символическими именами. Числовые коды операций заменились на мнемонические (символьные) обозначения, которые легче запомнить. Язык программирования приблизился к человеческому языку, и отдалился от языка машинных команд.

Следующим этапом развития языков программирования стали ***языки программирования высокого уровня***. Эти языки являются универсальными (на них можно создавать любые прикладные программы) и алгоритмически полными, имеют более широкий спектр типов данных и операций, поддерживают технологии (парадигмы) программирования. *Язык программирования высокого уровня* - это язык программирования, понятия и структура которого удобны для восприятия человеком и не зависят от конкретного компьютера, на котором будет выполняться программа.

Принципиальными отличиями языков программирования высокого от языков низкого уровня являются:

- использование переменных;
- возможность записи сложных выражений;
- расширяемость типов данных за счет конструирования новых типов из базовых;
- расширяемость набора операций за счет подключения библиотек подпрограмм;
- слабая зависимость от типа ЭВМ.

Одновременно с развитием универсальных языков программирования высокого уровня стали развиваться:

- 1. проблемно-ориентированные языки программирования, которые решали экономические задачи (COBOL), задачи реального времени (Modula-2, Ada), символьной обработки (Snobol), моделирования (GPSS, Simula, SmallTalk), численно-аналитические задачи (Analitic) и другие.

2. языки сверхвысокого уровня (непроцедурные), при использовании которых программист задает отношения между объектами в программе, например систему линейных уравнений, и определяет, что нужно найти, но не задает, как получить результат.

На языке высокого уровня программист задает процедуру (алгоритм) получения результата по известным исходным данным, поэтому они называются процедурными языками программирования. Языки сверхвысокого уровня называются непроцедурными т.к. сама процедура поиска решения встроена в язык (в его интерпретатор). Такие языки используются, например, для решения задач искусственного интеллекта (Lisp, Prolog). К непроцедурным языкам относят и языки запросов систем управления базами данных (QBE, SQL).

Таким образом, в ходе развития языков программирования появилось очень большое количество языков программирования. Существуют различные классификации языков программирования.

Наиболее распространенной классификацией языков программирования является разделение языков программирования на языки программирования низкого и высокого уровня.

Языки программирования

Низкого уровня

(машинно-зависимые)

Высокого уровня *(машинно-независимые)*

Машинные языки

Языки символического кодирования

Императивные *(как делать?)*

Декларативные

(что делать?)

Процедурные

на основе подпрограмм

Объектно-ориентированные

Традиционные

основаны на классах

Прототипные

используется объект-прототип

Функциональные

Логические

Рис. 2 Традиционная классификация языков программирования

Другой широко распространенной классификацией языков программирования предполагается разделение языков программирования на процедурные, непроцедурные и объектно-ориентированные.

Языки программирования

Процедурные

Непроцедурные

Объектно-ориентированные

Структурные

Операционные

Функциональные

Логические

Рис. 3 Классификация языков программирования (второй вариант)

В процедурных языках программа явно описывает действия, которые необходимо выполнить, а результат задается только способом получения его при помощи некоторой процедуры, которая представляет собой определенную последовательность действий. В структурных языках одним оператором записываются целые алгоритмические структуры: ветвления, циклы и т.д. В операционных языках для этого используются несколько операций.

Непроцедурное (декларативное) программирование появилось в начале 70-х годов 20 века и предполагает реализацию программы без описания последовательности действий. К непроцедурному программированию относятся функциональные и логические языки.

В функциональных языках программа описывает вычисление некоторой функции. Обычно эта функция задается как композиция других, более простых, те в свою очередь делятся на еще более простые задачи и т.д.

В логических языках программа вообще не описывает действий. Она задает данные и соотношения между ними. Машина перебирает известные и заданные в программе данные и находит ответ на вопрос. Порядок перебора не описывается в программе, а неявно задается самим языком.

На сегодняшний день существует много различных языков программирования. Для решения большинства задач можно использовать любой из них. Опытные программисты знают, какой язык лучше использовать для решения каждой конкретной задачи, так как каждый из языков имеет свои возможности, ориентацию на определённые типы задач, свой способ описания понятий и объектов.

Языки программирования

Степень ориентации на возможности ЭВМ

Степень

детализации алгоритма получения результата

Степень

ориентации на определенный класс задач

Возможность дополнения новыми типами данных и операциями

Машинно-зависимые

Машинно-независимые

Низкого уровня

Высокого уровня

Сверхвысокого уровня

Проблемно- ориентированные

Универсальные

Расширяемые

Нерасширяемые

Возможность управления реальными объектами и процессами

Способ получения результата

Тип решаемых задач

Систем реального времени

Систем условного времени

Процедурные

Непроцедурные

Системного программирования

Прикладного программирования

Рис. 4 Сводная схема классификации языков программирования

Реляционные

Функциональные

Логические

По типу встроенной процедуры поиска решения

1.3 Поколения языков программирования

Языки программирования по своим функциональным возможностям и времени создания принято разделять на несколько поколений (Generation Language, GL). На сегодняшний день выделяют пять поколений языков программирования. Каждое последующее поколение языков по своей функциональной мощности качественно

отличается от предыдущего:

Первое поколение языков программирования (1GL): машинно-ориентированные языки программирования с ручным управлением памяти.

Второе поколение языков программирования (2GL): автокоды – языки программирования с мнемоническим представлением команд.

Третье поколение языков программирования (3GL): языки программирования общего назначения, предназначенные для разработки прикладных программ любого типа.

Четвертое поколение языков программирования (4GL): усовершенствованные языки программирования 3GL, разработанные для создания специальных прикладных программ, для управления базами данных.

Пятое поколение языков программирования (5GL): декларативные языки программирования для построения программ с использованием методов искусственного интеллекта, объектно-ориентированные и визуальные принципы программирования.

В первое поколение (1GL) входят языки, созданные в 40 - 50-х годах. Программы на языках поколения 1GL писались в машинных кодах, т. е. каждая компьютерная команда вместе с ее операндами вводилась в ЭВМ в двоичном виде, что требовало огромных усилий по набору цифровых текстов и приводило к множеству трудноуловимых ошибок. К первому поколению языков программирования также относят первый ассемблер (50-е гг.), который позволял задавать названия команд в символическом виде и указывать числа не только в двоичном, но и в десятичном или шестнадцатеричном формате, что существенно облегчало работу программистов. Языки первого поколения продолжают использоваться - программы в машинных кодах для новых микропроцессоров, для которых еще не разработаны компиляторы, поддерживающие требуемый набор команд.

Языки программирования 2GL - конец 50-х - начало 60-х годов - создан символический ассемблер, позволявший писать программы без привязки к конкретным адресам памяти. Было введено понятие переменной. Скорость разработки и эффективность функционирования программ резко возросли. Ассемблеры активно применяются в настоящее время, как правило, для создания программ, максимально использующих возможности аппаратуры - различных драйверов, модулей состыковки с нестандартным оборудованием и т.д.

Вторая половина 60-х гг. – появление языков программирования 3GL. Языки программирования 3GL - универсальные языки программирования высокого уровня, позволяющие решать задачи из любых прикладных областей. Основные качества языков программирования 3GL - качества новых языков - как относительная простота, независимость от конкретного компьютера и возможность использования мощных синтаксических конструкций.

Языки программирования 4GL не имеют единой практической или алгоритмической направленности. В рамках четвертого поколения языков программирования выделяют четыре основных направления эволюционирования языков программирования:

1. Непроцедурное программирование – языки программирования искусственного интеллекта;
2. Объектно-ориентированное программирование (внедрение в программирование принципов наследования, полиморфизма и инкапсуляции);
3. Языки обработки баз данных – языки запросов;
4. Языки параллельного программирования для создания программного обеспечения для вычислительных средств параллельной архитектуры (многомашинные, мультипроцессорные среды и др.).

Языки программирования пятого поколения - интенсивно развивающиеся языки искусственного интеллекта, экспертных систем, баз знаний и т.н. «естественные языки», не требующие освоения какого-либо специального синтаксиса.

ГЛАВА 2 СОВРЕМЕННЫЕ МЕТОДЫ ПРОГРАММИРОВАНИЯ

2.1 Модульное, сборочное программирование

Возможности. Одной из ключевых проблем раннего программирования (70-90е годы прошлого столетия) является формирование понятия модуля и использование его в качестве строительных блоков новых ПС. Программирование с помощью модулей прошло длинный путь своего развития от библиотек стандартных подпрограмм общего назначения, библиотек, банков модулей до современных библиотек классов, методов вычисления матриц и т. п.

Модуль - это логически законченная часть программы, выполняющая определенную функцию. Он обладает такими свойствами: завершенность, независимость, самостоятельность, раздельная трансляция, повторное использование и др. Модуль раньше других программных объектов накапливался в библиотеках или Банках программ и модулей как готовых «деталей», из которых собирается ПС и адаптируется к новым условиям среды обработки. Программирование с помощью модулей привело к появлению сборочного программирования (СБ) как метода проектирования ПС снизу вверх из более простых элементов - модулей. Идею сборки модулей по принципу конвейера сформулировал академик В.М.Глушков в 1976 г. С этого момента в Институте кибернетики АН УССР проводились разработки методов автоматизации построения сложных программ из модулей. В том числе система автоматизации программ - АПРОП (1976-1990 гг.), основанная на понятиях: модуль с паспортными характеристиками, интерфейс для стыковки модулей и автоматизированная сборка по принципу фабрики программ.

Базовой концепцией СБ из готовых модулей, записанных в языках программирования - ЯП (Фортран, ПЛ/1, Алгол, Ассемблер), является интерфейс, обеспечивающий связь модулей в этих языках. Идея интерфейса значительно позже получила развитие, когда мировое сообщество пришло к необходимости объединения разных видов программ для разных систем и машин. Были созданы языки описания интерфейсов API (Application Program Interface), IDL (Interface Definition Language) и др. Они и сейчас выступают в качестве главной доминанты взаимодействия компонентов, объектов, аспектов в современных распределенных средах.

Интерфейс двух модулей в СБ определяется как модуль-посредник, в котором описываются операторы вызова модуля, его параметры и их типы. Другой тип интерфейса - это интерфейс пары ЯП, сущность которого заключается в преобразовании несовпадающих структур и типов данных в этих ЯП, которое возникает при вызове процедур в этих ЯП. Это преобразование осуществляется статически с помощью заранее разработанных функций и макросов релевантного преобразования библиотеки интерфейса, которая для указанных ЯП включала более 65 таких элементов.

Созданная концепция интерфейса модулей для класса ЯП положена в основу метода сборки (интеграции) модулей в ПС [5, 23-25]. Метод сборки основывается на математических формализмах определения связей (по данным и по управлению) между разнородными модулями и функциях преобразования данных в модуле-

посреднике для каждой пары разноязыковых модулей, в которых содержатся операторы CALL к вызывающему модулю.

Задача обеспечения интерфейса решается путем построении взаимно-однозначного соответствия между множеством фактических параметров V= {v^1, v^2 ,..., $y^к$} вызывающего модуля и множеством формальных параметров F = { f^1, f^2 , ..., D^1 } вызываемого модуля и их отображения в релевантные типы с помощью алгебраических систем преобразования данных в классе ЯП.

Алгебраическая система $G_a{}^4 = <X_a{}^4 , 0_a{}^4 >$, где $X_a{}^4$ - множество значений типа данных, а $0_a{}^4$ - множество операций над этим типом, ставит в соответствие типу $T_a{}^4$ языка l_a операции преобразования и изоморфное отображение алгебраических систем $G_a{}^4$ в Gp^d. Построенный класс алгебраических систем S = {$G_a{}^b$, $G_a{}^c$, $G_a{}^i$, $G_a{}^r$, $G_a{}^a$, $G_a{}^z$ } для типов данных ЯП (b-boolean, c-character, i-integer, r -real, a -array, z-record и др.) учитывает все виды преобразований. Доказан изоморфизм алгебраических систем класса S, кроме значений $X_a{}^4$ и $Xp^ч$ для типов $T_a{}^4$ и $Tp^ч$ из-за отсутствия соответствия.

Инструменты. Система АПРОП [23 - 24] воплотила метод сборки разноязыковых модулей. В дальнейшем в качестве готовых элементов ПС стали использовать любые порции формализованных знаний (объекты, компоненты, программы, каркасы и др.), полученные при реализации отдельных программ и систем. Возникли новые проблемы их использования, например перенос на другую платформу или другую среду функционирования, т.е. проблема интероперабельности. Стандартным механизмом решения этой проблемы на данный момент является обеспечение связи Java- и C/C++- компонентов с помощью интерфейса JNI (Java Native Interface). Обращение Java-классов к функциям и библиотекам на других ЯП включает: анализ Java-классов для поиска прототипов обращений к функциям на C/C++; генерацию заглавных файлов при компиляции в C/C++; обращение из Java-классов к COM-компонентам [25]. Для платформы .Net интероперабельность реализована средствами языка CLR (Common Language Runtime), в который транслируются объекты в ЯП (C#, Visual Basic, C++, Jscript), используется библиотека стандартных классов независимо от ЯП и стандартные средства генерации в представление .Net-компонента. Особенности ОС и архитектур компьютеров учитывает также среда CORBA. Стандарт ДСТУ [26] обеспечивает независимо от ЯП спецификацию типов данных разноязыковых компонентов средствами специального языка, механизмов их агрегации и преобразования. Язык этого стандарта - альтернатива IDL, API, RPC.

2.2 Структурное программирование

Возможности. Основу структурного метода программирования составляет декомпозиция создаваемой системы на отдельные функции и задачи, которые в свою очередь разбиваться на более мелкие. Процесс декомпозиции продолжается вплоть до определения конкретных процедур. Для функций разрабатываются программные компоненты и устанавливаются связи между ними.

Метод структурного программирования базируется на двух общих принципах:

- решение сложных проблем путем их разбиения на множество меньших независимых задач, легких для понимания и решения;
- организация составных частей проблемы в иерархические древовидные структуры с добавлением новых деталей на каждом уровне.

Приведенные принципы дополнены такими механизмами:

- абстракция - выделение существенных аспектов системы и отвлечение от несущественных;
- формализация, т.е. строгое формальное решение проблемы;
- непротиворечивость - обоснование и согласование элементов системы;
- структуризация данных согласно иерархической организации.

Сформировался ряд моделей этого метода программирования, главными из которых являются:

- SADT (Structured Analysis and Design Technique) - структурный анализ и техника проектирования;
- SSADM (Structured Systems Analysis and Design Method) - метод структурного анализа и проектирования и др.

На стадии проектирования эти модели дополняются структурными схемами и диаграммами.

SADT представляет собой совокупность методов, правил и процедур, предназначенных для построения функциональной модели ПрО. Функциональная модель в SADT отображает структуру функций объекта автоматизации, производимые действия и связи между ними. Концепции метода: графическое представление блочного моделирования функций в виде блоков с интерфейсами (вход/выход) в виде дуг, задающих взаимодействие блоков в соответствии с

правилами:

- строгость и точность количества блоков на каждом уровне декомпозиции (от 3 до 6 блоков) и связь диаграмм через номера этих блоков;
- уникальность меток и наименований;
- разделение входов и управлений (определение роли данных).
- исключение влияния организационной структуры на функциональную модель.

Результатом применения метода SADT является модель, которая состоит из диаграмм, фрагментов текстов и глоссария со ссылками на ее элементы.

SSADM базируется на таких структурах: последовательность, выбор и итерация (цикл). Моделируемый объект задается сгруппированной последовательностью компонентов, следующих друг за другом, операторами выбора компонентов из группы и цикличеким выполнением отдельных компонентов. Каждый компонент изображается на диаграмме четырехугольником с прямыми или закругленными углами и дугами с входными и выходными данными.

Базовая диаграмма является иерархической и включает: список всех компонентов описываемого объекта; идентифицированные группы выбранных и повторяемых, а также последовательных компонентов. Модель процесса включает структурные диаграммы, которые в SSADM создаются в процессе:

- определения функций;
- моделирования взаимосвязей событий и сущностей;
- логического проектирования данных;
- проектирования диалога;
- логического проектирования БД;
- физического проектирования.

Наиболее распространенным средством моделирования данных являются диаграммы "сущность-связь" (ERD), с помощью которых определяются объекты (сущности) ПрО, их свойства (атрибуты) и отношения (связи) [24]. Сущность (Entity) - реальный либо воображаемый объект, существенный для ПрО. Каждая сущность и ее экземпляр имеют уникальные имена. Сущность обладает свойствами:

- имеет один или несколько атрибутов, которые либо принадлежат сущности, либо наследуются через связь (Relationship);
- каждая сущность может обладать любым количеством связей с другими сущностями модели.

Связь - это ассоциация между двумя сущностями ПрО. Каждый экземпляр родительской сущности ассоциирован с произвольным количеством экземпляров второй сущности (потомками), а каждый экземпляр сущности-потомка - с одним экземпляром сущности-родителя. Таким образом, экземпляр сущности-потомка может существовать только при наличии сущности «родитель».

Инструменты. Основным инструментом является комплекс инструментальных, методических и организационных средств системы SSADM. Она принята государственными органами Англии в качестве основного системного средства и используется многими организациями как внутри страны, так и за ее пределами. На основе идеологии структурного программирования создан ряд CASE-средств (SilverRun, Oracle Disigner, Erwin для прямой и обратной связи с Rational Rose и др.).

2.3 Объектно-ориентированное проектирование (ООП)

Возможности. ООП представляет собой стратегию проектирования ПС из объектов. Объект - это нечто, способное пребывать в различных состояниях и имеющее множество операций. Состояние определяется как набор атрибутов объекта. Операции, связанные с объектом, предоставляют сервисы другим объектам для выполнения определенных вычислений. Объекты объединяются в классы, каждый из которых имеет описания всех атрибутов и операций.

ПС состоит из взаимодействующих объектов, имеющих локальное состояние и способных выполнять набор операций, определяемый состоянием объекта. Объекты инкапсулируют информацию об их состоянии и ограничивают к ним доступ.

Процессы ООП - это проектирование классов объектов и взаимоотношений между ними. Проект ПС реализуется в виде исполняемой программы, в которой все необходимые объекты создаются динамически с помощью классов. ООП включает три этапа проектирования ПС:

- Анализ ОО. Создание ОО модели предметной области, в которой объекты отражают реальные ее сущности и операции, выполняемые ими;
- Проектирование ОО. Уточнение ОО модели с учетом требований для реализации конкретных задач системы;
- Программирование ОО. Реализация ОО модели средствами C++, Java и др.

Данные этапы могут происходить друг за другом, и на каждом этапе может применяться одна и та же система нотаций. Переход к следующему этапу приводит к усовершенствованию результатов предыдущего этапа путем более детального описания определенных ранее классов объектов и определения новых.

Изменение реализации какого-нибудь объекта или добавление ему новых функций не влияет на другие объекты системы. Четкое соответствие между реальными объектами (например, аппаратными средствами) и управляющими объектами ПС облегчает понимание и реализацию проекта в целом.

Новый проект ПС можно разрабатывать на базе ранее созданных объектов, что снижает стоимость и уменьшает риск разработки нового проекта.

ООП использует возможности структурного подхода, например декомпозицию в диаграммах использования и состояний, а также диаграммы потоков данных, широко используемые при логическом и физическом проектировании БД. На стадии проектирования требований в ООП прослеживается связь между диаграммами «сущность - связь» и классов. Далее в процессе проектирования преимущественное значение имеет моделирование ПрО в терминах взаимодействующих объектов.

Инструменты. Одной из инструментальной поддержки метода ООП является RUP - неофициальный стандарт разработки одиночных ПС из элементов Use Case, отвечающих требованиям конечных пользователей. Варианты использования - главные элементы этого процесса включают группы сценариев, описывающих применение ПС и интеграцию на этапах разработки с измерением времени выполнения этапов и измерения компонентов процесса на качество. При измерении времени выполнения этапов используются контрольные точки ПС. Компоненты процессов RUP описываются в категориях действий, рабочих потоков и исполнителей, которые измеряются и оцениваются. Основным недостатком ООП является отсутствие возможности задавать такие аспекты, как синхронизация выполнения объектов, удаленное взаимодействие в распределенной среде, защита данных, устойчивость и т.д. Этот недостаток относится и к компонентным технологиям ActiveX и JavaBeans и получит развитие в генерирующем программировании.

2.4 Метод моделирования UML

Метод UML (United Modeling Language - унифицированный язык моделирования) широко используется программными организациями как метод моделирования ПС

исходя из таких базовых понятий:

- онтологии домена для определения состава классов объектов домена, их атрибутов и взаимоотношений, а также услуг, которые могут выполнять объекты классов;
- модели поведения, предназначенной для определения возможных состояний объектов, инцидентов, инициирующих переходы из одного состояния к другому, а также сообщений, которыми обмениваются объекты;
- модели процессов, которая определяет действия, выполняемые объектами.

Язык UML поддерживает задание статических и динамических моделей, в том числе концептуальной модели и модели последовательностей, узлы которой определяют последовательность взаимодействий между объектами. В концептуальной модели отражаются требования к системе в виде совокупности нотаций - диаграмм, которые визуализируют основные элементы структуры проектируемой системы. Эффективное представление взаимодействий между разными участниками разработки сопровождается заданием комментариев: традиционного текста или стереотипа.

Стереотип - это средство метаклассификации элемента, который представлен в диаграмме с описанием назначения (например, <<треугольник>>,<<точка>>). Он может расширяться и адаптироваться к конкретным областям применения.

ПС описывается с помощью рассматриваемых ниже диаграмм.

Диаграмма классов отображает онтологию домена, состав классов объектов, их взаимоотношения, список атрибутов класса и его операций. Атрибутами могут быть:

- рцЬИе (общий) - операция класса, вызываемая из любой части программы любым объектом ПС;
- protected (защищенный) - операция, вызываемая объектом того класса, в котором она определена или наследована;
- private (частный) означает операцию, вызванную только объектом того класса, в котором она определена.

Под операцией понимается сервис, который экземпляр класса может выполнять, если к нему будет произведен соответствующий вызов.

Классы могут находиться в следующих отношениях.

Ассоциация - зависимость между объектами разных классов, каждый из которых является равноправным ее членом и обозначает количество экземпляров объектов каждого класса, участвующих в связи (0 - ни одного, 1 - один, n - много).

Зависимость между классами, при которой класс использует определенную операцию другого класса.

Экземпляризация - зависимость между параметризированным абстрактным классом-шаблоном (template) и реальным классом, который инициирует параметры шаблона (например, контейнерные классы языка C++).

Диаграммы моделирования поведения системы. Под поведением системы понимается множество объектов, обменивающихся сообщениями с помощью следующих диаграмм:

- последовательность, упорядочивающая взаимодействие объектов при обмене сообщениями;
- сотрудничество, определяющее роль объектов во время их взаимодействия;
- активность, показывающая потоки управления при взаимодействии объектов;
- состояние, указывающее на динамику изменения объектов.

Диаграмма последовательности применяется для задания взаимодействия объектов. Любому из объектов сценария ставится в соответствие линия жизни, которая отображает ход событий от его создания до разрушения. Взаимодействие объектов контролируется событиями, которые происходят в сценарии и стимулируют посылки сообщений другому объекту.

Диаграммы сотрудничества представляют совокупность объектов, поведение которых имеет значение для достижения целей системы, и взаимоотношения ролей, важных в сотрудничестве. Диаграмма позволяет моделировать статическое взаимодействие объектов без учета фактора времени. При параметризации она представляет абстрактную схему сотрудничества - паттерн, в котором определяется множество конкретных схем.

Диаграмма деятельности задает поведение системы в виде работ, которые может выполнять система или актер, которые зависят от принятия решений в зависимости от сложившихся условий. Эта диаграмма предусматривает параллельное выполнение нескольких видов деятельностей и их синхронизацию.

Диаграмма состояний отражает условия переходов и действия при входе в состояние, его активность при выходе из этого состояния.

Диаграмма реализации состоит из диаграммы компонента и диаграммы размещения. Диаграмма компонента отображает структуру системы как композицию компонентов и связей между ними в виде графа, узлами которого являются компоненты, а дуги - отношения зависимости. Диаграмма размещения позволяет задать состав физических ресурсов системы (узлов системы) и отношений между ними.

Пакет - это совокупность элементов (объектов, классов, подсистем и т.п.), представленная группой подсистем разного уровня детализации. Пакет определяет пространство, занимаемое элементами, т. е. его составляющими и ссылками на него. Объединение элементов в пакеты происходит тогда, когда они используются совместно или касаются такого аспекта, как интерфейс с пользователем, ввод/вывод и т.п. Пакет может быть элементом конфигурации, структуры ПС из отдельных модулей или из заведомо определенного состава их вариантов.

Инструменты. Rational Rose - средство визуального моделирования ОО ПС. Rose является CASE- средством, графические возможности которого основаны на использовании UML и позволяют проектировать разные виды диаграмм с заданием всех свойств, отношений и взаимодействий. Средства системы обеспечивают проектирование и моделирование общей модели процессов (абстрактной) предприятия до конкретной (физической) модели классов создаваемой ПС. Допускается прямое и обратное проектирование, т.е. доработка старой системы. Результатом моделирования является визуальная логическая модель системы, которая дополняется моделями конкретных классов на ЯП.

В рамках Rational Rose поставляются Rose DataModeler для проектирования системы и баз данных, Rational Rose Professional для прямого и обратного проектирования шаблонов системы на ЯП, Rose Enterprise для проектирования предприятия и др.

2.5 Компонентное программирование

Возможности. По оценкам экспертов, 75 % работ по программированию в мире дублируются. Поэтому переход к повторному использованию компонентов (ПИК) наиболее производителен. Этот процесс происходил эволюционно: от подпрограмм, модулей и объектов до компонентов путем совершенствования этих элементов и методов, их спецификации и композиции. Компонентное программирование обобщает ООП. Объекты рассматриваются на логическом

уровне проектирования программной системы, а компоненты - как непосредственная физическая реализация объектов. Один компонент может быть реализацией нескольких объектов или даже некоторой части объектной системы, полученной на уровне проектирования [7]. Компоненты в отличие от объектов могут изменяться и пополняться новыми функциями и интерфейсами. Они конструируются в виде некоторой программной абстракции, состоящей из трех частей: информационной, внешней и внутренней.

Информационная часть содержит описание: назначение, дата изготовления, условие применения (ОС, платформа и т.п.), возможность ПИК, среды окружения и др.

Внешняя часть определяет взаимодействие компонента с внешней средой и с платформой, на которой он будет выполняться и включает следующие характеристики:

- интероперабельность - способность взаимодействовать с компонентами других сред;
- переносимость - способность компонента выполняться на разных платформах компьютеров;
- интеграционное^ - объединение компонентов на основе интерфейсов в более сложные ПС;
- нефункциональность - обеспечение безопасности, надежности работы компонентов и т.п.

Внутренняя часть - это программный фрагмент кода, кластер, системная или абстрактная структура,

представленные в виде проекта компонента, спецификации, выходного кода и др. Данная часть компонента включает:

- интерфейс (interface),
- реализацию (implementation),
- схемы развертки (deployments).

Интерфейс отображает способ обращения к другим компонентам с помощью средств языков IDL или APL. В нем указываются типы данных и операции передачи параметров для взаимодействия компонентов друг с другом. Каждый компонент может реализовывать определенную совокупность интерфейсов.

Реализация - это код, который будет выполняться на компьютере с использованием заданных интерфейсов. Компонент может иметь несколько реализаций в зависимости от ОС, модели данных и соответствующей системы управления БД.

Развертка - это создание физического файла, готового для выполнения, содержит все необходимые данные для настройки и запуска компонента.

Компонент описывается в ЯП, не зависит от ОС и реальной платформы, где он будет функционировать. Компоненты наследуются в виде классов и используются в модели или композиции, а также в каркасе интегрированной среды. Управление компонентами осуществляется на трех уровнях: архитектурном, компонентном и интерфейсном, между которыми есть взаимная связь.

Компонент может быть представлен разными структурами.

Контейнер - структура, в которой заданы функции, порядок их выполнения, вызываемые события, сервис, интерфейс, необходимый для обращения к провайдеру сервиса.

Паттерн - абстракция, которая содержит описание взаимодействий совокупности объектов, ролей участников и их ответственности. Может быть и ПИК.

Каркас - представляет собой высокоуровневую абстракцию проекта ПС, в которой отделены функции компонентов от задач управления ними. В бизнес-функциях компонентов каркас задает надежное управление, объединяет множество взаимодействующих между собою объектов в некоторую интегрированную среду для решения заданной конечной цели и может быть “белым или черным ящиком”.

Каркас типа “белый ящик” включает абстрактные классы для представления цели объекта и его интерфейса. При реализации эти классы наследуются в конкретные классы с указанием соответствующих методов реализации, что соответствует ООП. Каркас типа «черный ящик» в видимой части имеет точки входа и выхода.

Компонентная модель включает многочисленные проектные решения по композиции компонентов, разные типы паттернов, связи между ними, способы взаимодействия и операции развертки ПС в среде функционирования.

Композиция компонентов из каркасов включает следующие типы:

- компонент-компонент обеспечивает непосредственное их взаимодействие через интерфейс на уровне приложения;

- композиция каркас-компонент способствует взаимодействию каркаса с компонентами, управлению ресурсами компонентов и их интерфейсами на системном уровне;
- компонент-каркас обеспечивает взаимодействие компонента с каркасом по типу «черного ящика», в видимой части которого находится спецификация для развертывания и выполнения сервисной функции на сервисном уровне;
- каркас-каркас создает возможность взаимодействовать с каркасом, каждый из которых может разворачиваться в гетерогенной среде и взаимодействовать через интерфейсы на сетевом уровне.

ПИК - это готовые компоненты, которые используются в других ПС [15], они упрощают и сокращают сроки разработки новых ПС благодаря:

- отображению в них базовых функций и понятий ПС;
- скрытию представления данных, операций обновления и получения доступа к этии данным;
- обработке возникающих исключительных ситуаций и др.

Как и любые элементы промышленного производства, ПИК должны отвечать определенным требованиям, иметь характерные свойства, структуру, интерфейс и др. Для практического применения ПИК проводится их классификация и каталогизация, а также после выбора настройка на новые условия среды функционирования. Классификация ориентирована на унификацию представления информации о компонентах для поиска и отбора в среде хранения. Каталогизация направлена на физическое размещение ПИК в разных хранилищах (репозитариях, банках компонентов и др.) для извлечения их из них с целью использования.

Инструменты. Примером инструментария по разработке ПС из ПИК является система RSEB [17]. В ней реализован метод, основанный на нотации UML, объектно-ориентированном методе и использовании ПИК многократного применения. Построение разных ПС проводится с использованием элементов Use Case. Компонент в RSEB — это тип, класс или любое другое программное средство (например, Use Case анализа, проектирования или реализации), разработка которого осуществляется с учетом его многократного применения. Компонентная система в RSEB рассматривается как ПС, содержащая ряд ПИК и повторно используемых нефункциональных характеристик. В качестве примера компонентной системы можно привести повторно используемые каркасы графических пользовательских интерфейсов и математические библиотеки [18]. Практически компонентная система позволяет производить другие прикладные системы, если в ней выделены родовые подсистемы с многократно используемыми

решениями. Инженерия предметной области в системе RSEB состоит из двух процессов конструирования:

1. семейства ПС путем разработки и сопровождения общей многоуровневой архитектуры систем;
2. компонентных систем из семейства. Процесс разработки различных частей ПС направлен на конструирование и реализацию надежных, расширяемых и гибких компонентов повторного использования

Система OOram также обеспечивает поддержку инженерии разработки ПС с помощью ПИК на основе следующих процессов [19]: создание модели, например ролевой, ее синтез и разработку спецификации объектов; процесс разработки системы в соответствии с ЖЦ, начиная с формулирования требований и заканчивая введением в действие ПС; процесс производства ПС из набора компонентов многократного применения в соответствии с этапами: анализ рынка, разработка и упаковка продукта, маркетинг ПИК и др.

2.6 Аспектно-ориентированное программирование (АОП)

Возможности. АОП базируется на методах ООП разбиения задач ПрО на ряд функциональных компонентов, а также на аспектах (синхронизации, взаимодействия, защиты и др.), которые встраиваются в отдельные компоненты как некоторые их реализации и влияют на композицию компонентов [10]. В качестве аспекта может быть некоторая идея, которая интересует нескольких заинтересованных лиц и представляется в виде варианта использования, функции, как обязательный элемент компонента или программы. Некоторые аспекты могут действовать на протяжении ЖЦ процесса разработки, они кодируются, тестируются и упрощают результат разработки. Как правило, создание продукта связано с выполнением разных ролей в процессе разработки, которые могут выполнять агенты с такими функциями: определение архитектуры, управление проектом, повышение качества и продуктивности разработки ПС.

Так как современные языки программирования не позволяют инкапсулировать аспекты в проектные решения системы, то в некоторые инструментальные компонентные системы введен механизм фильтрации входных сообщений, с помощью которых выполняется изменение параметров и имен текстов аспектов в конкретном компоненте. «Нечистый» код компонента (код с пересекаемыми его

аспектами) требует разработки новых подходов к композиции компонентов, ориентированных на ПрО и на выполнение ее функций [15].

Общие средства композиции объектов или компонентов (вызов процедур, RPC, RMI, IDL и др.) в АОП являются недостаточными, так как аспекты требуют декларативного сцепления между частичными описаниями, а также связывания отдельных обрывков из различных объектов. Одним из механизмов композиции является фильтр композиции, суть которого состоит в обновлении заданных аспектов синхронизации или взаимодействия без изменения функциональных возможностей компонента с помощью входных и выходных параметров сообщений, которые проходят фильтрацию и изменения, связанные с переопределением имен или функций самих объектов. Фильтры делегируют внутренним компонентам параметры, переадресовывая установленные ссылки, проверяют и размещают в буфере сообщения, локализуют ограничения на синхронизацию и готовят компонент для выполнения.

Поскольку в ОО-программах может содержаться много мелких методов, которые самостоятельно не выполняют расчетов и обращаются к другим методам, расположенным в областях внешнего уровня, Деметер сформулировал закон [12], согласно которому не разрешаются длинные последовательности методов, связанные с передачами параметров через внутренние объекты. В результате создается код алгоритма, который содержит имена классов, не задействованных в выполнении расчетных операций. При необходимости внесения изменений в структуру классов, создается новый дополнительный класс, который расширяет ранее созданный код и не вносит качественных изменений в расчетные программы.

С точки зрения моделирования аспекты можно рассматривать как каркасы декомпозиции системы, в которых отдельные аспекты синхронизации, взаимодействия и др. пересекают ряд многократно используемых ПИК. Разным аспектам проектируемой системы могут отвечать и разные парадигмы программирования: объектно-ориентированные, структурные и др. Они по отношению к проектируемой ПрО образуют мультипара- дигмную концепцию аспектов, такую, как синхронизация, взаимодействие, обработка ошибок и др., и требуют значительных доработок процессов их реализации. Кроме того, можно устанавливать связи с другими предметными областями для описания аспектов приложения в терминах родственных областей. Появились языки АОП, которые позволяют описывать пересекающиеся аспекты в разных ПрО. В процессе компиляции переплетения объединяются, оптимизируются и генерируются [12] в динамике выполнения.

Существенной чертой любых аспектов является модель, которая пересекает структуру другой модели, для которой первая является аспектом. Так как аспект связан с моделью, то ее можно перестроить так, чтобы аспект стал, например, модулем и выполнял функцию посредника, беря на себя все образцы взаимодействия. Однако решение таким образом проблемы пересечения может привести к усложнению и понижению эффективности выполнения созданного модуля или компонента. Переплетение может проявиться на последующих этапах процесса разработки, когда реализуются аспекты, и они делают запутанным выходной код. Одним из путей оптимальной реализации аспектов является минимизация сцепления между аспектами и компонентами, которая реализуется ссылками на языковые конструкции, варианты использования, сопоставление с образцом. Реализация аспектов в различных блоках кода позволяет устанавливать перекрестные ссылки между ними, тем самым декларируется связь с соответствующей моделью. В этих блоках появляются точки соединения сообщений, которые обеспечивают связь между транзакциями, обработкой ошибок и т. п.

Связь между характеристиками и аспектами может быть выявлена в ходе анализа ПрО. Создается динамическое связывание через косвенный код или таблицы виртуальных таблиц для повторного связывания или статическое («жесткое») связывание в период компиляции.

Для целей АОП хорошо подходит модель модульных расширений, создаваемая в рамках метамодельного программирования. Она предлагает оперативное распространение новых механизмов композиции в отдельные части ПС или их семейств с учетом предметно-ориентированных возможностей языков (например, SQL) и каркасов, которые поддерживают разного рода аспекты [14].

Инструменты. Для эффективной реализации аспектов разработана 1Р-библиотека расширений. В ней размещены некоторые функции компиляторов, методов, средства оптимизации, редактирования, отображения. и др. Например, библиотека матриц, с помощью которой вычисляются выражения с массивами, обеспечивается скорость выполнения, предоставления памяти и т.п. Использование таких библиотек в расширенных средах программирования называют родовым программированием, а решение проблем экономии, перестройки компиляторов под каждое новое языковое расширение, использование шаблонов и результатов предыдущей обработкой относят к области ментального программирования [12].

2.7 Порождающее (генерирующее) программирование

Возможности. Порождающее программирование (деиега1е programming) - это парадигма разработки ПС, основанная на моделировании групп или отдельных элементов ПС, таких, что при описании списка конкретных требований до системы из этих элементов, аспектов, элементарных ПИК и каркасов конфигурации автоматически генерируется промежуточный или конечный продукт[3].

Данное программирование представляет объединенную целостную и стройную концепцию создания инженерии ПрО путем проектирования семейств ПС на основе объектов, компонентов, аспектов, сервисов, ПИК, систем, характеристик и т.п. По существу это программирование является дальнейшим развитием ООП в направлении использования ПИК, каркасов и разных аспектов в создаваемых ПС. Главным элементом программирования является не уникальный программный продукт, созданный из ПИК для конкретных применений, а семейство ПС или конкретные его экземпляры. Элементы семейства не создаются с нуля, а генерируются на основе общей генерирующей модели (generative domain model), т.е. модели семейства, включающей средства определения ее членов, компоненты реализации и ПИК, из которых собирается любой член этого семейства, и базу конфигураций, отображающую спецификации членов семейства.

В созданном программном члене семейства отражается максимум знаний о его производстве, а именно конфигурации, инструментарии измерения и оценки, методах тестирования и планирования, отладки, визуального представления и пр. Эти аспекты отображают специфику ПрО, многократно используемых ПИК, представленных в активных библиотеках [13].

Активные библиотеки содержат не только базовый код реализации понятий ПрО, но и целевой код по обеспечению компиляции, оптимизации, отладки, визуализации и др. Этот код представляется разным инструментальным системам (компиляторами, анализаторами кода, отладчиками и т.п.) в виде описания функций. Фактически компоненты активных библиотек выполняют роль интеллектуальных агентов, в процессе взаимодействия которых создаются новые специализированные агенты, ориентированные на предоставление пользователю возможности решать конкретные задачи ПрО. Для связи агентов при выполнении задач генерации, преобразования и взаимодействия разных объектов создается инфраструктура, т.е. расширяемая среда программирования. Используя эту среду,

можно конструировать ПС из компонентов библиотек, а также из специальных метапрограмм среды, которые осуществляют редактирование, отладку, визуализацию, взаимодействие и др. Кроме того, есть возможность пополнять ее новыми сгенерированными компонентами в рамках отдельных ПС семейства, которые относятся к компонентам многоразового применения. Вместе с тем ЯП компонентов дополняются новыми аспектами, которые расширяют одновременно и ПрО новыми возможностыми.

Целью порождающего программирования является разработка правильных компонентов для целого семейства и после этого автоматически предоставлять их другим семействам. Реализации этой цели соответствует два сформировавшихся направления использования ПИК:

1. прикладная инженерия - процесс производства конкретных ПС из ПИК, созданных ранее самостоятельных ПС, или отдельных элементов процесса инженерии некоторой ПрО;
2. инженерия ПрО - построение семейства ПС путем сбора, классификации и фиксации ПИК, опыта конструирования систем или готовых частей систем для конкретной ПрО. При этом создаются системные инструментальные системы поддержки поиска, адаптации ПИК и внедрения их в новую ПС семейства.

Разбиение инженерии ПрО на конструирование семейства приложений и конструирование компонентных систем обусловлено четким разделением задач по разработке общей архитектуры и многократно используемых решений для отдельных подсистем.

Основными этапами инженерии ПрО являются: анализ ПрО и выявление характеристик; определение области действий ПрО; определение общих и изменяемых характеристик в характеристической модели. Эта модель устанавливает зависимость между различными членами семейства и в пределах самого члена семейства, а также создает базис для производства конкретных программных членов семейства с механизмами изменчивости независимо от средств их реализации. На основе характеристической модели и компонентов реализации генерируется доменная модель для семейства с использованием данной модели, знания о конфигурациях и спецификации высокого уровня компонентов автоматически генерируются в некоторый член семейства. При этом используются такие стандартные системные средства, как JavaBeans с графическим интерфейсом, визуальными компонентами или C++ с компонентами стандартной библиотеки шаблонов.

Функциональные компоненты представляются в виде объектов, процедур, модулей, которым необходимы такие свойства, как безопасность, синхронизацияи др. Эти свойства, как правило, выражаются небольшими фрагментами кода в нескольких функциональных компонентах. Они могут пересекать ряд компонентов и представлять собой отдельные аспекты в терминологии АОП. Смесь компонентов и аспектов образует ряд небольших классов и методов, что в конце концов усложняет создаваемую ПС.

Инженерия ПрО ориентирована на разработку решений, связанных с группами ПС, обеспечивает поддержку мультисистемного проектирования и моделирование изменчивости. Она состоит в следующем: разработка моделей групп систем семейства; моделирование понятий данной ПрО и выявление альтернативных методов реализации этапов; разработка групп систем для последующего их повторного использования; системное моделирование характеристик компонентов в соответствии с характеристической моделью семейства ПрО; реализация процесса сборки каждого члена семейства на основе базы знаний о конфигурации.

Для выполнения инженерии ПрО используются следующие процессы:

корректировка процессов для разработки решений на основе ПИК;

моделирование изменчивости и зависимостей, которое начинается с разработки словаря описания различных понятий, фиксации их в характеристической модели и в справочной информации сведений об изменчивости моделей (объектных, Use Case, взаимодействия и др.). Фиксация зависимостей между характеристиками избавляет пользователей от некоторых конфигурационных манипуляций, которые выполняются, как правило, вручную;

разработка инфраструктуры ПИК - описание, хранение, поиск, оценивание и объединение готовых

ПИК.

При определении членов семейства ПрО используются новые понятия - пространство задач, а в технологии реализации компонентов на основе каркаса конфигураций - пространство решений.

Пространство задач. Областью разработки является семейство систем, в которых используются компоненты многократного применения. Процесс разработки с повторным использованием организуется таким образом, чтобы в нем применять не только ПИК, но и инструменты, созданные в ходе разработки ПрО. В рамках

инженерии ПрО разрабатывается характеристическая модель, которая обобщает характеристики системы и изменяемые параметры разных частей семейства, а также решения, связанные с группами ПС.

Инженерия ПрО включает разработку моделей групп систем, моделирование понятий ПрО, разработку характеристических моделей и групп систем для последующего их повторного использования.

В данном программировании нашли отражение идеи международного OMG-комитета, касающиеся видов ПрО горизонтального и вертикального типов. К горизонтальным ПрО отнесены общие системные средства: графические пользовательские интерфейсы, СУБД, системные программы, библиотеки расчета матриц, контейнеры, каркасы и т.п., а к вертикальным видам ПрО - прикладные системы (медицинские, биологические, научные и т.д.), методы инженерии ПрО и горизонтальные методы обслуживания архитектуры многократного применения, интерфейсов, библиотек и др.

Пространство решений включает компоненты, каркасы и образцы проектирования. При этом каркас оснащается изменяемыми параметрами модели, что может привести к излишней его фрагментации и появлению «множества мелких методов и классов». Каркас обеспечивает динамическое связывание аспектов и компонентов в процессе реализации изменчивости между разными приложениями. Образцы проектирования обеспечивают создание многократно используемых решений в различных ПС. Для задания таких аспектов, как синхронизация, удаленное взаимодействие, защита данных и т.д., применяются компонентные технологии ActiveX и JavaBeans, а также новые механизмы композиции, метапрограммирования и др.

Инструменты. Примером поддержки инженерии ПрО и реализации горизонтальных методов является система DEMRAL [52, 54], предназначенная для разработки библиотек: численного анализа, контейнеров, распознавания речи, графовых вычислений и т. д. Основными видами абстракций этих библиотек ПрО являются абстрактные типы данных (abstract data types - ADT) и алгоритмы. DEMRAL позволяет моделировать характеристики ПрО в виде высокоуровневой характеристической модели и предметно-ориентированных языков конфигурирования.

Система конструирования RSEB [47] базируется на вертикальных методах, ПИК и ориентирована на использование элементов Use Case при проектировании крупных ПС. Эффект достигается, когда вертикальные методы инженерии ПрО «вызывают»

различные горизонтальные методы, относящиеся к разным прикладным подсистемам. При работе над отдельной частью семейства системы могут быть задействованы такие основные аспекты: взаимодействие, структуры, потоки данных и др. Главную роль, как правило, выполняет один из методов, например графический пользовательский интерфейс в бизнес-приложениях или метод взаимодействия компонентов в распределенной открытой среде (например, в CORBA).

ЗАКЛЮЧЕНИЕ

Со времени создания первых программируемых машин человечество придумало более двух с половиной тысяч языков программирования. Каждый год их число пополняется новыми. Некоторыми языками умеет пользоваться только небольшое число их собственных разработчиков, другие становятся известны миллионам людей. Профессиональные программисты иногда применяют в своей работе более десятка разнообразных языков программирования.

Язык программирования может быть представлен в виде набора спецификаций, определяющих его синтаксис и семантику. Для многих широко распространённых языков программирования созданы международные стандарты. Специальные организации проводят регулярное обновление и публикацию спецификаций и формальных определений соответствующего языка.

В рамках таких комитетов продолжается разработка и модернизация языков программирования и решаются вопросы о расширении или поддержке уже существующих и новых языковых конструкций.

В представленной работе проведено исследование основных положений, связанных с языками и системами программирования, рассмотрены основные классификации языков программирования и их эволюция.

На основе изученных данных и при использовании литературных источников и материалов глобальной сети Интернет в работе определены основные характеристики и методологий работы современных языков программирования, рассмотрены классы систем программирования, определены тенденций развития современных языков и систем программирования.

СПИСОК ЛИТЕРАТУРЫ

1. Буч Г. Объектно-ориентированный анализ.- М.:”Бином”, 2016. - 560 с.
2. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного проектирования. Паттерны проектирования. - СПб: Питер, 2015. - 368 с.
3. Чернецки К., Айзенекер У. Порождающее программирование. Методы, инструменты, применение. - Изд. дом «Питер».- М., С-Пет.- 2015. - 730 с.
4. Римский Г.В. Структура и функционирование системы автоматизации модульного программирования // Программирование. - 2016. - №2. - С.31-38.
5. Лаврищева Е.М., ГрищенкоВ.Н. Сборочное программирование. - Киев. - Наукова Думка, 2015 - 213с.
6. Yourdan E. Modern Srtucrured Analysis. - New York: Prentice Hall, 2016. - 297 p.
7. Грищенко В.Н., Лаврищева Е.М. Методы и средства компонентного программирования // Кибернетика и системный анализ. - 2013.№1. - С.39-55.
8. Грищенко В.Н., Лаврищева Е.М. Компонентно-ориентированное программирование. Состояние, направления и перспективы развития // Проб. программирования. - 2012. - № 1-2. - С. 80-90.
9. WCOP ’99 - Proc. of the Fourth International Workshop on Component-Oriented Programming // Ed. Boush J., Szyperski C. , Week W. -ISSN 1581. - С.113.
10. Lopes C., Kiczales G., Murphy G., Lee A. (organizers). Proc. of the ECOOP on Aspect-Oriented Programming, Kyoto, Japan, April 20, 2016.
11. Elrad T., Filman R.E. Aspect-oriented programming // Com. of the ACM. - 2015. - Vol.44, № 10. - P.33-38.
12. Яковсон Айвар. Мечты о будующем программирования. Открытые системы. - М.: 2025. - №12. - С.59-63.
13. Летичевский А.А, Маринченко В.Г. Объекты в системе алгебраического программирования // Кибернетика и системный анализ. - 2017. №2. - С.160-180.
14. Редько В.Н. Экспликативное программирование: ретроспективы и перспективы // Проб. программирования. - 2016. - №2. - С. 22- 41.
15. НикитченкоН.С. Композиционно-номинативный подход к уточнению понятия программы // Там же. - 2014. - №1. - С.16-31.
16. Letichevsky A.A., Gilbert D.R. A model for interaction of agents and environments // Recent trends in algebra's development technique language, 2015. - P.311 - 329.
17. Letichevsky A.A., Gilbert D.R. A General Theory of Action Language // Кибернетика и системный анализ. - 2016. - № 1. - С.16-36.
18. Летичевский А.А., Капитонова Ю.В. Доказательство теорем в математической информационной среде // Там же. - 2016. - №.4. - С. 3 - 12.
19. Лаврищева Е.М., Грищенко В.Н. Связь разноязыковых модулей в ОС ЕС. - М: Финансы и статистика. -2012. - 127 с.

20. Лаврищева Е.М. Сборочное программирование. Некоторые итоги и перспективы // Проб. программирования. -2016. - №1-2. - С.20 - 31.
21. Марка Д.А., Мак Груэн К. Методология структурного анализа и проектирования. - М.: МетаТехнология, 2017. - 346 с.
22. Skidmore.S, Mills.G, FarmerR. SSADM: Models and Methods. - Prentice-Hall, Englewood Cliffs, 2014. - 693 p.
23. DeMarko D.A., McGovan R.L. SADT: Structured Analysis and Design Technique. - New York: Mcgray Hill, 2016. - 378 p.
24. Barker R. CASE-method. Entity Relatiionship Modeling.Copyrigth Oracle Corporation UK Limited New York: - 2015. - 312 p.
25. Шлеер С., Меллор С. Объектно-ориентированный анализ: моделирование мира в состояниях. - Киев: Диалектика, 2013. - 238с.
26. Martin J., Odell J.J. Object-oriented analysis and design. - Prentice Hall. - 2014. - 367 p.
27. Rumbaugh Jet. al. Object Oriented Modeling and Design.- Englewood Cliffs: Prentice Hall, 2016.- NJ. - 271 p.
28. Firesmith D. Object-oriented Methods, Standarts and Procedures. - Englewood Cliffs: Printice Hall, 2014. - 367 p.

Целью данной работы является анализ методов современного программирования, для достижения поставленной цели, были выделены следующие задачи:

- рассмотреть теоретические аспекты систем и языков программирования;

- изучить современные методы программирования.

Объект исследования - методы программирования.

Предмет исследования - современные методы программирования.

Структура работы состоит из введения, основной части, заключения и списка литературы.

Теоретической и методологической базой данной работы послужили труды российских и зарубежных авторов

Целью данной работы является анализ методов современного программирования, для достижения поставленной цели, были выделены следующие задачи:

- рассмотреть теоретические аспекты систем и языков программирования;

- изучить современные методы программирования.

Объект исследования - методы программирования.

Предмет исследования - современные методы программирования.

Структура работы состоит из введения, основной части, заключения и списка литературы.

Теоретической и методологической базой данной работы послужили труды российских и зарубежных авторов

Целью данной работы является анализ методов современного программирования, для достижения поставленной цели, были выделены следующие задачи:

- рассмотреть теоретические аспекты систем и языков программирования;

- изучить современные методы программирования.

Объект исследования - методы программирования.

Предмет исследования - современные методы программирования.

Структура работы состоит из введения, основной части, заключения и списка литературы.

Теоретической и методологической базой данной работы послужили труды российских и зарубежных авторов

Целью данной работы является анализ методов современного программирования, для достижения поставленной цели, были выделены следующие задачи:

- рассмотреть теоретические аспекты систем и языков программирования;

- изучить современные методы программирования.

Объект исследования - методы программирования.

Предмет исследования - современные методы программирования.

Структура работы состоит из введения, основной части, заключения и списка литературы.

Теоретической и методологической базой данной работы послужили труды российских и зарубежных авторов