

СИСТЕМАТИЗАЦИЯ ИТ-МЕТОДОЛОГИЙ ДЛЯ ПРОВЕДЕНИЯ СРАВНИТЕЛЬНОГО АНАЛИЗА ЯЗЫКОВ И ПАРАДИГМ ПРОГРАММИРОВАНИЯ

Задача ИТ-инженера – конструирование программных систем.

Для начала определимся с терминологией. Парадигмой будем называть общий, глобальный подход к организации конструирования системы, пронизывающий процесс от начала и до конца. В таком рассмотрении одновременно совмещать несколько парадигм нельзя.

Процесс конструирования ПО сам состоит из отдельных (под)процессов. Их я выделяю четыре, остальные оставлю за бортом обсуждения, как не связанные с непосредственно программированием. Итак, процессы конструирования ПО:

0. **Анализ** проблемной области.
1. **Программирование «в малом»** (алгоритмизация).
2. **Программирование «в среднем»** - система типизации, организация вычислительного процесса и т.п.
3. **Программирование «в большом»** - проектирование, **архитектура** программной системы.

Парадигмы языков программирования затрагивают пп. 1, 2, 3, парадигмы проектирования – в пп. 0, 2, 3.

Для каждого процесса существует ряд методологий по его организации. Методологии одного процесса часто могут быть несовместимы между собой – точно так же, как парадигмы процесса разработки в целом. Методологии для разных процессов можно свободно совмещать друг с другом, учитывая, однако, то, что они в разной степени подходят друг другу. Также для разных частей программной системы в рамках одного процесса могут быть применены разные методологии (например, часть системы можно запрограммировать в структурно-модульной методологии, а часть – в объектно-ориентированной).

Причиной путаницы часто является то, что методологии разных процессов носят одно название. Например, объектно-ориентированное программирование, проектирование и анализ – это три отдельных методологии. В рамках какой-либо парадигмы может присутствовать ОО-программирование, но отсутствовать, например, ОО-проектирование (именно так и происходит в чистой Оберон-парадигме, хотя в парадигме Component Pascal/BlackBox ОО-проектирование также может использоваться).

Также, кроме методологий, для каждого процесса выделим средства. Средство – это самостоятельно существующий прием или механизм, который может использоваться в произвольном сочетании с другими средствами, в любых методологиях процесса.

Теперь я попытаюсь расписать известные методологии для процессов. В ряде мест могу ошибаться, особенно касательно функциональных методологий.

0. Анализ проблемной области.

Методологии:

- 1) Структурный анализ (например, семейство IDEF).
- 2) Объектно-ориентированный анализ (например, аналитическая часть Rational Unified Process).

3) Функционально-ориентированный? Мне кажется, что такой должен существовать – анализ с ориентацией на последующую реализацию в стиле ФП.

4) Другие, менее «магистральные»...

1. Программирование «в малом» (алгоритмизация)

Методологии:

- 1) Императивная (основанная на последовательной смене состояния вычислителя).
- 2) Функциональная (основанная на вычислении композиции функций).
- 3) Декларативная (основанная на формулировании правил, запросов и автоматическом выводе).
- 4) Автоматные (основаны на математических автоматных моделях – конечные, машины Гуревича, сети Петри и т.п. Часто подразумевают графическое представление алгоритма).
- 5) Марковские (основаны на обработке цепей символов).
- 6) Другие...

Средства:

- 7) Структурные операторы управления.
- 8) Рекурсивное вычисление функций.
- 9) Методологии формального доказательства и верификации (например, дейкстровская).
- 10) Другие...

2. Программирование «в среднем» (система типизации, организация вычислительного процесса).

Методологии:

- 1) Структурная (развитая система типизации, типы данных гетерогенны – атомарные, массивы, записи, атомарные с прикладными ограничениями и т.п. Единицы выч. процесса – функции-процедуры, обрабатывающие потоки данных).
- 2) Объектно-ориентированная (основные сущности – классы и объекты, типизация стремится к гомогенности - «все есть объект», поддерживается инкапсуляция и расширение объектных типов - «наследование»).
- 3) Функциональная строго-типизированная (строгая типизация с автоматическим выводом типов).
- 4) Функциональная динамически типизированная («операционная» типизация, гомогенность представления кода и данных...).
- 5) Основанные на процессах и сигналах.
- 6) Другие...

Средства:

- 7) Строгая типизация.
- 8) Соккрытие информации, концепция модуля/пакета. Вместе с (1) дает структурно-модульную методологию.
- 9) Инкапсуляция данных и кода.
- 10) Расширение типов данных. Вместе с (1) дает структурно-объектную методологию (Оберон, Ada).

- 11) Безопасность указателей.
- 12) Сборка мусора для динамических объектов.
- 13) Статический полиморфизм.
- 14) Динамический полиморфизм.
- 15) Виртуальные методы.
- 16) Перегрузка операторов.
- 17) Параметризация, макромеханизмы (дают поддержку обобщенного программирования и эмуляцию в одном языке различных стилей программирования).
- 18) Языковые или стандартные средства параллельного программирования.
- 19) Ленивые вычисления, каррированные функции.
- 20) Динамическая типизация.
- 21) Другие...

3. Программирование «в большом» - проектирование, архитектура программной системы.

Методологии:

1) Объектно-ориентированные (класс – не только тип данных, но и элемент архитектуры, инкапсуляции, сокрытия информации. Ключевая проблема для использования в системном программировании – отсутствие оболочки верхнего уровня для групп классов, проблема взаимодействия тесно связанных групп классов с сохранением закрытости для окружения. Слабо выраженная структура программной системы).

(1) Объектная (к проектированию и организации архитектуры применяются те же объектно-ориентированные методы, что и к проектированию абстракций данных и вычислительного процесса. Для отражения специфики проектных решений используются паттерны ОО-проектирования – схемы, позволяющие нагрузить классы и объекты архитектурным содержанием).

(2) Объектные компонентно-ориентированные (являются ответом С-семейства языков на появившуюся первой модульную компонентно-ориентированную методологию – см. ниже. Компонентом является класс, единицей загрузки являются двоичные библиотеки операционной системы, для описания интерфейсов и организации взаимодействия применяются надязыковые механизмы. Между компонентами отсутствует наследование реализации, экспортируются только интерфейсы. Для управления жизненным циклом объектов используется автоматическое управление памятью – обычно подсчет ссылок. Проектировались с прицелом на независимость от языка, поэтому не всегда удобны для использования, внутри одного проекта используются редко. Типичные: COM, CORBA. Особые: модель Java – монопольная со сборщиком мусора; .NET – мультязыковая, прозрачная для использования за счет компиляции всех языков в одно двоичное представление, со сборщиком мусора).

2) Кластерно-модульные (основной элемент архитектуры – модули и кластеры модулей, основные отношения – межмодульные. Являются проекцией

общеинженерных подходов на инженерию ПО. Основная идея – абстракции данных и абстракции архитектуры ортогональны друг другу и должны выражаться через различные конструкции языка и среды программирования).

(1) Модульная (Модуль - «черный ящик», существует в единственном экземпляре, является единицей инкапсуляции, сокрытия информации, исходного кода, компиляции, двоичного кода, загрузки в память... Эталон – Модуль-2).

(2) Модульная с объектно-ориентированными разъемами (расширение модульной архитектуры рядом приемов, основанных на использовании объектных расширяемых интерфейсов. Главный прием – родовая шина сообщений. Эталон - Оберон).

(3) Модульная компонентно-ориентированная (Компонентом и единицей загрузки является модуль языка программирования в двоичном виде. Во избежание проблемы хрупкого базового класса ограничено межмодульное наследование реализации. Для обеспечения гибкости и взаимозаменяемости компонент экспортируются преимущественно абстрактные интерфейсы, конкретные типы скрываются. Внутри модуля нет защиты классов друг от друга, разрешено наследование реализации – по raganoia rule. Для управления жизненным циклом объектов используется автоматическое управление памятью – обычно сборщик мусора. Эталон – Компонентный Паскаль).

3) Функциональные (элементом архитектуры являются функциональные абстракции и отношения между ними).

4) Другие...

Средства:

- 5) Автоматическое управление памятью.
- 6) Динамическая загрузка двоичного кода.
- 7) Поддержка метайнформации и метапрограммирования.
- 8) Средства обработки исключений.

В представленной схеме много условностей, но в целом она дает некоторую систему координат для анализа и сравнения различных языков и парадигм, которые теперь можно разложить на отдельные кирпичики.

Сделаем это для нескольких языков программирования. После номера процесса будем писать список поддерживаемых методологий, затем – список поддерживаемых средств, замечания про особенности поддержки тех или иных методологий, средств.

Модуль-2

1. Императивная. Средства – структурные операторы управления; рекурсивные функции; применимость формальной верификации.
2. Структурно-модульная. Средства – строгая типизация; сокрытие информации и инкапсуляция в модулях; статический полиморфизм процедур; есть реализации с языковыми средствами параллельного программирования (мониторы).
3. Кластерно-модульная классическая.

Оберон

1. Императивная. Средства – структурные операторы управления; рекурсивные функции; применимость формальной верификации.
2. Структурно-объектная. Средства – строгая типизация; сокрытие информации и инкапсуляция в модулях; расширение типов данных; инкапсуляция поведения в процедурных полях объектов; безопасность указателей; сборка мусора; динамический полиморфизм объектных параметров; эмуляция виртуальных таблиц через процедурные переменные каждого объекта.
3. Кластерно-модульная с ОО-разъемами. Средства – автоматическое управление памятью; динамическая загрузка; метапрограммирование.

Компонентный Паскаль

1. Императивная. Средства – структурные операторы управления; рекурсивные функции; применимость формальной верификации.
2. Структурно-объектная; объектно-ориентированная. Средства – строгая типизация; сокрытие информации в модулях; инкапсуляция как в модулях, так и в типах данных (связанные процедуры записей); расширение типов данных; безопасность указателей; сборка мусора; динамический полиморфизм объектных параметров; виртуальные связанные процедуры. Также поддерживается экземплярное ООП в стиле Оберона.
3. Кластерно-модульная с ОО-разъемами; модульная компонентно-ориентированная. Средства - автоматическое управление памятью; динамическая загрузка; метапрограммирование.

Active Oberon – отличается от Компонентного Паскаля встроенной в язык поддержкой параллельного программирования (активные объекты) и несколько более наглядной поддержкой объектно-ориентированной методологии и инкапсуляции в классах на уровне 2 (введен специальный объектный тип данных).

Ада (в стандарте 2005)

1. Императивная. Средства – структурные операторы управления; рекурсивные функции; применимость формальной верификации.
2. Структурно-объектная; объектно-ориентированная. Средства – строгая типизация; сокрытие информации в модулях-пакетах; инкапсуляция как в модулях, так и в типах данных (связанные процедуры теговых типов); расширение типов данных (теговые типы); виртуальные связанные процедуры; безопасность указателей; в некоторых реализациях – ограниченная сборка мусора; статический полиморфизм процедур; динамический полиморфизм объектных параметров; перегрузка операторов; параметризация и обобщенное программирование, языковые средства параллельного программирования (задачи, рандеву, защищенные типы).
3. Кластерно-модульная с ОО-интерфейсами. Средства – обработка исключений. Средства компонентного программирования отсутствуют.

Дельфи

1. Императивная. Средства – структурные операторы управления; рекурсивные функции; ограниченная применимость формальной верификации. Есть неструктурные операторы.
2. Структурно-модульная; объектно-ориентированная. Средства – относительно строгая типизация; сокрытие информации в модулях; инкапсуляция как в модулях, так и в классах (методы); статический полиморфизм процедур; динамический полиморфизм объектных параметров; виртуальные методы. Объекты размещаются только в динамической памяти; указатели небезопасны.
3. Объектно-ориентированная; кластерно-модульная классическая. Расширенные модульные архитектуры невозможны из-за отсутствия расширения записей и только динамического размещения объектов. Средства компонентного программирования отсутствуют.

Java

1. Императивная. Средства – структурные операторы управления; рекурсивные функции; применимость формальной верификации.
2. Объектно-ориентированная. Средства – строгая типизация; сокрытие информации в классах и пакетах; инкапсуляция в классах; динамический полиморфизм объектных параметров и вызовов методов; безопасность указателей; сборка мусора; динамический полиморфизм объектных параметров; виртуальные методы; стандартные средства параллельного программирования (потoki и синхронизированные объекты – мониторы с общей условной переменной). Структурно-модульная методология не поддерживается, т.к. в пакетах отсутствуют процедуры, есть только методы классов.
3. Объектно-ориентированная, объектная компонентно-ориентированная. Средства – автоматическое управление памятью; динамическая загрузка классов; метапрограммирование; обработка исключений. Модульные методологии не поддерживаются, т.к. у «модулей»-пакетов нет состояния, таким образом, построение межмодульных отношений невозможно.

C++

1. Императивная. Средства – структурные операторы управления; рекурсивные функции. Присутствуют неструктурные операторы. Из-за этого и из-за сложности языка возможности формальной верификации очень ограничены.
2. Структурная; объектно-ориентированная; с помощью параметрических механизмов возможна местная ограниченная эмуляция функциональной и декларативной. Средства – сокрытие информации и инкапсуляция в классах; статический полиморфизм процедур и методов; динамический полиморфизм объектных параметров и вызовов методов; перегрузка операторов; параметризация и макромеханизмы. Другие средства не поддерживаются – типизация слаба, указатели опасны, конструкция модуля-пакета отсутствует, сборка мусора невозможна.

3. Объектно-ориентированная. Средства – обработка исключений. Метапрограммирование, автоматическое управление памятью и динамическая загрузка невозможны. Компонентные средства отсутствуют.

Haskell

1. Функциональная. Средства – рекурсивные функции; легкость для формальной верификации.

2. Функциональная со строгой типизацией; объектно-ориентированная. Средства – строгая типизация с автоматическим выводом типов; сокрытие информации в модулях-пакетах; инкапсуляция в модулях-пакетах, типах данных и классах; безопасность – неявность указателей; сборка мусора; всевозможные виды статического и динамического полиморфизма; виртуальные методы; перегрузка операторов; возможна ограниченная эмуляция параметризации и макромеханизмов; монады для эмуляции императивности; ленивые вычисления и каррированные функции.

3. Функциональная. Средства – автоматическое управление памятью, метапрограммирование. Модульные методологии не поддерживаются, т.к. у «модулей»-пакетов нет состояния, таким образом, построение межмодульных отношений невозможно.

LISP

1. Функциональная. Средства – рекурсивные функции. Возможности формальной верификации ограничены из активного использования макромеханизмов.

2. Функциональная с динамической типизацией; с помощью макромеханизмов возможна ограниченная эмуляция многих других методологий. Средства – безопасность – неявность указателей; сборка мусора; всевозможные виды статического и динамического полиморфизма; динамическая типизация.

3. Функциональная. Средства – автоматическое управление памятью, метапрограммирование.

Такая система сравнения теперь ни у кого не должна оставить вопросов, в чем принципиальные отличия и преимущества, например, Компонентного Паскаля перед Дельфи – сразу видно, что эти языки сильно различаются в поддерживаемых методологиях программирования и проектирования. В силу этого Компонентный Паскаль ориентирован более на решение системных и инфраструктурных задач и прикладных задач с «плавающей» структурой, в то время как Дельфи дает преимущества при решении типовых задач массовой индустрии с использованием исключительно объектно-ориентированного проектирования. В то же время разница между Дельфи и Java очень невелика, но при этом Java обладает некоторыми преимуществами.

Ada является очень мощным представителем семейства Паскаля – она обходит C++ по качествам всех сопоставимых средств и имеет множество средств, которых нет в последнем, в частности, обеспечивающих очень высокую надежность приложений. Однако Ada совершенно не приспособлена для компонентно-ориентированного программирования.

Haskell и LISP, являясь языками функционального программирования, тем не менее

сильно различаются между собой. Haskell является богатым языком по средствам уровня (2), LISP является очень динамичным и гибким языком. Однако на уровне проектирования архитектуры оба функциональных языка весьма ограничены.

На этом мы завершим сей опус – мы представили в нем возможную схему для конструктивного сравнения различных языков и парадигм программирования, которую читатель самостоятельно сможет применить к интересующим его языкам.