Модульное программирование

**Мо́дульное программи́рование** — это организация программы как совокупности небольших независимых блоков, называемых модулями, структура и поведение которых подчиняются определенным правилам. Использование модульного программирования позволяет упростить тестирование программы и обнаружение ошибок. Аппаратно-зависимые подзадачи могут быть строго отделены от других подзадач, что улучшает мобильность создаваемых программ.

**Мо́дуль** — функционально законченный фрагмент программы. Во многих языках (но далеко не обязательно) оформляется в виде отдельного файла с исходным кодом или поименованной непрерывной её части. Некоторые языки предусматривают объединение модулей в пакеты.

**Модульность программного кода**

Принцип модульности является средством упрощения задачи проектирования ПС и распределения процесса разработки ПС между группами разработчиков. При разбиении ПС на модули для каждого модуля указывается реализуемая им функциональность, а также связи с другими модулями. Удобство использования модульной архитектуры заключается в возможности обновления (замены) модуля, без необходимости изменения остальной системы.

Роль модулей могут играть структуры данных, библиотеки функций, классы, сервисы и др. программные единицы, реализующие некоторую функциональность и предоставляющие интерфейс к ней.

Программный код часто разбивается на несколько файлов, каждый из которых компилируется отдельно от остальных. Такая модульность программного кода позволяет значительно уменьшить время перекомпиляции при изменениях, вносимых лишь в небольшое количество исходных файлов, и упрощает групповую разработку. Также это возможность замены отдельных компонентов (таких как jar-файлы, so или dll библиотеки) конечного программного продукта, без необходимости переборки всего проекта (например, разработка плагинов к уже готовой программе).

Одним из методов написания модульных программ является объектно-ориентированное программирование. ООП обеспечивает высокую степень модульности благодаря таким свойствам, как инкапсуляция, полиморфизм и позднее связывание.

В информатике **инкапсуляцией** (лат. en capsula) называется упаковка данных и/или функций в единый компонент.

**Модульная система модулей**

Несмотря на то, что модульное программирование никак не связано с деталями конкретного языка (и даже в случае отсутствия явной поддержки со стороны языка может применяться при достаточной дисциплине со стороны программистов), большинство языков выдвигают на верхний уровень свою собственную природу системы модулей,

В 2000 году Ксавье Лерой предложил делать системы модулей модульными, то есть параметризуемыми описанием конкретного ядра языка со своей системой типов. В качестве примера он продемонстрировал обобщённую реализацию языка модулей ML (как наиболее развитой системы модулей из известных на данный момент) и примеры её инстанцирования на традиционный для неё язык ML и на язык Си.

Реализация Лероя сама построена посредством языка модулей ML, а именно в виде функтора, параметризованного данными о ядре языка и описанием его механизма проверки согласования типов. Это значит, что при написании компилятора некоторого языка достаточно описать ядро языка и передать его данному функтору (как библиотечной функции) — в результате получится компилятор расширения известного языка системой модулей ML.

История концепции модулей

История концепции модулей как единиц компиляции восходит к языкам Фортран II и Кобол, то есть, к концу 1950-х годов. В 1976 году появилась публикация, в которой была развита концепция модульности — о языке Mesa (англ.), который был разработан в Xerox PARC. В 1977 году подробно ознакомился с этой концепцией ученый Никлаус Вирт, общаясь с разработчиками в Xerox PARC. Эти идеи были использованы Виртом при создании языка Модула-2, публикация о котором вышла в 1977 году.

Термин «модуль» в программировании начал использоваться в связи с внедрением модульных принципов при создании программ. В 1970-х годах под модулем понимали какую-либо процедуру или функцию, написанную в соответствии с определенными правилами. Например: «модуль должен быть простым, замкнутым (независимым), обозримым (от 50 до 100 строк), реализующим только одну функцию задачи, имеющим одну входную и одну выходную точку».

Первым основные свойства программного модуля более-менее четко сформулировал Д. Парнас (David Parnas) в 1972 году: «Для написания одного модуля должно быть достаточно минимальных знаний о тексте другого». Таким образом, в соответствии с определением, модулем могла быть любая отдельная процедура (функция) как самого нижнего уровня иерархии (уровня реализации), так и самого верхнего уровня, на котором происходят только вызовы других процедур-модулей.

Таким образом, Парнас первым выдвинул концепцию скрытия информации (англ. information hiding) в программировании. Однако существовавшие в языках 70-х годов только такие синтаксические конструкции, как процедура и функция, не могли обеспечить надежного скрытия информации, из-за повсеместного применения глобальных переменных.

Решить эту проблему можно было только разработав новую синтаксическую конструкцию, которая не подвержена влиянию глобальных переменных. Такая конструкция была создана и названа модулем. Изначально предполагалось, что при реализации сложных программных комплексов модуль должен использоваться наравне с процедурами и функциями как конструкция, объединяющая и надежно скрывающая детали реализации определенной подзадачи.

Таким образом, количество модулей в комплексе должно определяться декомпозицией поставленной задачи на независимые подзадачи. В предельном случае модуль может использоваться даже для заключения в него всего лишь одной процедуры, если необходимо, чтобы выполняемое ею локальное действие было гарантировано независимым от влияния других частей программы при любых изменениях.

Впервые специализированная синтаксическая конструкция модуля была предложена Н. Виртом в 1975 г. и включена в его новый язык Modula . Насколько сильно изменяются свойства языка, при введении механизма модулей, свидетельствует следующее замечание Н.Вирта, сделанное им по поводу более позднего языка Модула-2: «Модули — самая важная черта, отличающая язык Модула-2 от его предшественника Паскаля».

Реализация в языках программирования

Языки, формально поддерживающие концепцию модулей: IBM S/360 Assembler, Кобол, RPG, ПЛ/1, Ада, D, F (англ.), Фортран, Haskell, Blitz BASIC, OCaml, Паскаль, ML, Модула-2, Оберон, Компонентный Паскаль, Zonnon, Erlang, Perl, Python и Ruby. В IBM System использовались «модули» от языков RPG, Кобол и CL, когда программировалась в среде ILE.

Модульное программирование может быть осуществлено, даже когда синтаксис языка программирования не поддерживает явное задание имён модулям.

Программные инструменты могут создавать модули исходного кода, представленные как части групп — компонентов библиотек, которые составляются с программой компоновщиком.

Стандартный Паскаль не **предусматривает механизмов раздельной компиляции** частей программы с последующей их сборкой перед выполнением. Вполне понятно стремление разработчиков коммерческих компиляторов Паскаля включать в язык средства, повышающие его модульность.

**Модуль в Паскале** — это автономно компилируемая программная единица, включающая в себя различные компоненты раздела описаний (типы, константы, переменные, процедуры и функции) и, возможно, некоторые исполняемые операторы инициирующей части.

По своей организации и характеру использования в программе модули Паскаля близки к модулям-пакетам (PACKAGE) языка программирования Ада. В них так же, как и в пакетах Ады, явным образом выделяется некоторая «видимая» интерфейсная часть, в которой сконцентрированы описания глобальных типов, констант, переменных, а также приводятся заголовки процедур и функций. Появление объектов в интерфейсной части делает их доступными для других модулей и основной программы. Тела процедур и функций располагаются в исполняемой части модуля, которая может быть скрыта от пользователя.

Модули представляют собой прекрасный инструмент для разработки библиотек прикладных программ и мощное средство модульного программирования. Важная особенность модулей заключается в том, что компилятор размещает их программный код в отдельном сегменте памяти.

**Длина сегмента** не может превышать 64 Кбайт, однако количество одновременно используемых модулей ограничивается лишь доступной памятью, что позволяет создавать большие программы.

Стандарт ISO/IEC 12207-95: основные определения – система, модель жизненного цикла, квалификационные требования. Основные процессы, их содержание, работы и задачи процесса разработки.

Стандарт ISO/IEC 12207-95 определяет стратегию и общий порядок в создании и эксплуатации ПО, он охватывает ЖЦ ПО от концептуализации идей до завершения ЖЦ (жизненного цикла).

Особенности стандарта

Стандарт не предписывает конкретную модель ЖЦ или метод разработки ПО; Он определяет, что стороны-участники использования стандарта ответственны=

за выбор модели ЖЦ для проекта ПО,

за адаптацию процессов и задач стандарта к этой модели,

за выбор и применение методов разработки ПО,

за выполнение действий и задач, подходящих для проекта ПО;

Стандарт ISO/IEC 12207-95 равносильно ориентирован на организацию действий каждой из двух сторон: поставщик (разработчик) и покупатель (пользователь); может быть в равной степени применен, когда обе стороны — из одной организации.

Определения стандарта

Система - это объединение одного или более процессов, аппаратных средств, программного обеспечения, оборудования и людей для обеспечения возможности удовлетворения определенных потребностей или целей.

Модель жизненного цикла — структура, содержащая процессы, действия и задачи, которые осуществляются в ходе разработки, функционирования и сопровождения программного продукта в течение всей жизни системы, от определения требований до завершения ее использования.

Требования квалификации — набор критериев или условий (квалификационные требования), которые должны быть удовлетворены для того, чтобы квалифицировать программный продукт как удовлетворяющий условиям его спецификациям и готовый для использования в целевой окружающей среде.

Стандарт определяет общую структуру жизненного цикла ПО в виде 3-х ступенчатой модели, состоящей из:

· процессов,

· видов деятельности,

· задач

Стандарт не определяет метрики, по которым можно было бы отслеживать ход работ и их результативность. Самыми крупными элементами являются процессы жизненного цикла ПО. Всего выделено 18 процессов, которые объединены в 3 группы:

-основные процессы;

-поддерживающие процессы;

-организационные процессы;

-процесс адаптации.

Основные процессы ЖЦ

1) Процесс приобретения - его задача - определить действия предприятия-покупателя, которое приобретает автоматизированную систему, программный продукт или сервис ПО:

· инициация приобретения;

· подготовка запроса предложений;

· подготовка контракта;

· анализ поставщиков;

· получение ПО.

2) Процесс передачи (поставки) определяет действия предприятия-поставщика, которое снабжает покупателя системой, программным продуктом или сервисом ПО.

3) Процесс разработки - его задача - определить действия предприятия-разработчика, которое создает программный продукт.

Включает следующие работы:

· развертывание процесса разработки;

· анализ системных требований;

· проектирование (программно-аппаратной) системы в целом;

· анализ требований к ПО;

· проектирование архитектуры ПО;

· детальное проектирование;

· кодирование;

· отладочное тестирование;

· интеграцию ПО;

· квалификационное тестирование ПО;

· системную интеграцию;

· квалификационное тестирование системы;

· развертывание (установку или инсталляцию) ПО.

4) Процесс эксплуатации определяет действия предприятия-оператора, которое обеспечивает обслуживание системы в процессе ее функционирования в интересах пользователей. Включает такие работы, как:

· консультирование пользователей;

· получение обратной связи и др.

5) Процесс поддержки ПО определяет действия персонала сопровождения, который обеспечивает:

· инсталляцию и удаление программного изделия на вычислительной системе;

· анализ возникающих проблем;

· внесение изменений;

· экспертизу и передачу измененного ПО;

· перенос ПО с одной платформы на другую;

· изъятие ПО из эксплуатации.

**ISO/IEC 12207**:2008 Systems and software engineering — Software life cycle processes — стандарт ISO, описывающий процессы жизненного цикла программного обеспечения.

**Жизненный цикл программного обеспечения (ПО)** — период времени, который начинается с момента принятия решения о необходимости создания программного продукта и заканчивается в момент его полного изъятия из эксплуатации. Этот цикл — процесс построения и развития ПО.

Стандарт разработан подкомитетом ПК 7 «Системная и программная инженерия» (англ. SC 7 System and Software Engineering) Совместного технического комитета №1 ИСО/МЭК «Информационные технологии» (англ. ISO/IEC JTC 1 Information Technology).

Данный стандарт, используя устоявшуюся терминологию, устанавливает общую структуру процессов жизненного цикла программных средств, на которую можно ориентироваться в программной индустрии. Стандарт определяет процессы, виды деятельности и задачи, которые используются при приобретении программного продукта или услуги, а также при поставке, разработке, применении по назначению, сопровождении и прекращении применения программных продуктов.

**См. также**

**Общая характеристика стандарта ISO/IEC 12207**

**Общие сведения о семействе стандартов 12207**

В основе практически всех современных промышленных технологий создания ПС лежит международный стандарт ISO/IEC 12207 «Системная и программная инженерия. Процессы жизненного цикла программных средств.»

В состав семейства входят:

ISO/IEC 12207:1995 «Information technology–Software life cycle processes» с дополнениями и изменениями ISO/IEC 12207:1995/AMD 1:2002 и ISO/IEC 12207:2002/AMD 2:2004 (принят в новой редакции в 2008 году)

ISO/IEC 12207:2008 «Systems and software engineering–Software life cycle processes»

ISO/IEC TR 15271:1998 Information technology – Guide for ISO/IEC 12207 (Software Life Cycle Processes)

ISO/IEC TR 16326:1999 Software engineering – Guide for the application of ISO/IEC 12207 to project management

Спецификации ISO/IEC 12207:1995, ISO/IEC TR 15271:1998 и ISO/IEC TR 16326:1999 введены в качестве национальных стандартов РФ

Развитие стандарта

Стандарт ISO/IEC 12207 был опубликован 1 августа 1995 года и явился первым международным стандартом, содержавшим представительный набор процессов ЖЦ, действий и задач в отношении ПО, которое рассматривалось как часть большей системы, а также применительно к программным продуктам и услугам. За стандартом ISO/IEC 12207 в ноябре 2002 года последовал стандарт ISO/IEC 15288, посвященный процессам ЖЦ систем. Широта применения ПС привела к тому, что ПО и процессы его разработки не могли рассматриваться в отрыве от систем, но только как составная часть системы и процесса её создания. В Дополнениях к стандарту ISO/IEC 12207 были введены цель процесса и его выходы и определена эталонная модель процесса, отвечающая требованиям стандарта ISO/IEC 15504-2. Международный стандарт ISO/IEC 12207:2008, представляет собой переработанные и исправленные дополнения к стандарту ISO/IEC 12207 и является первым шагом в стратегии SC7 по гармонизации спецификаций, имеющей целью создание полностью интегрированного набора процессов ЖЦ систем и программных средств и руководства по их применению.

**Процессы жизненного цикла**

Группы процессов жизненного цикла включают в себя:

1. процессы соглашения — 2;
2. процессы организационного обеспечения проекта — 5;
3. процессы проекта — 7;
4. технические процессы — 11;
5. процессы реализации программных средств — 7;
6. процессы поддержки программных средств — 8;
7. процессы повторного применения программных средств — 3.
8. Процессы соглашения
9. Поставка
10. Приобретение
11. Процессы организационного обеспечения проекта
12. Процесс менеджмента модели жизненного цикла;
13. Процесс менеджмента инфраструктуры;
14. Процесс менеджмента портфеля проектов;
15. Процесс менеджмента людских ресурсов;
16. Процесс менеджмента качества.
17. Процессы проекта
18. Процессы менеджмента проекта
19. процесс планирования проекта;
20. процесс управления и оценки проекта.
21. Процессы поддержки проекта
22. процесс менеджмента решений;
23. процесс менеджмента рисков;
24. процесс менеджмента конфигурации;
25. процесс менеджмента информации;
26. процесс измерений.
27. Технические процессы
28. Определение требований правообладателей
29. Анализ системных требований
30. Проектирование архитектуры системы
31. Процесс реализации
32. Процесс комплексирования системы
33. Процесс квалификационного тестирования системы
34. Процесс инсталляции программных средств
35. Процесс поддержки приемки программных средств
36. Процесс функционирования программных средств
37. Процесс сопровождения программных средств
38. Процесс изъятия из обращения программных средств
39. Процессы реализации программных средств
40. Процесс анализа требований к программным средствам;
41. Процесс проектирования архитектуры программных средств;
42. Процесс детального проектирования программных средств;
43. Процесс конструирования программных средств;
44. Процесс комплексирования программных средств;
45. Процесс квалификационного тестирования программных средств
46. Процессы поддержки программных средств
47. Процесс менеджмента документации программных средств;
48. Процесс менеджмента конфигурации программных средств;
49. Процесс обеспечения гарантии качества программных средств;
50. Процесс верификации программных средств;
51. Процесс валидации программных средств;
52. Процесс ревизии программных средств;
53. Процесс аудита программных средств;
54. Процесс решения проблем в программных средствах.
55. Процессы повторного применения программных средств
56. Процесс проектирования доменов;
57. Процесс менеджмента повторного применения активов;
58. Процесс менеджмента повторного применения программ.