

Технология проектирования программного обеспечения ИС

- *Проектирование и методологии*
- *Методологии, ориентированные на обработку*
- *Методологии, ориентированные на данные*

Проектирование ПО

процесс создания проекта программного обеспечения (ПО), а также дисциплина, изучающая **методы проектирования**.

Проектирование ПО является частным случаем **проектирования** продуктов и процессов.

Проектирование

- Проектирование — процесс определения архитектуры, компонентов, интерфейсов и других характеристик системы или её части (ISO 24765). Результатом проектирования является проект — целостная совокупность моделей, свойств или характеристик, описанных в форме, пригодной для реализации системы

Методы проектирования

Метод — это прием или способ действия с целью достижения желаемого результата. Его выбор зависит не только от вида решаемой задачи, но и индивидуальных черт разработчика (его характера, организации мышления, склонности к риску, способности принимать решения и нести за них ответственность и т. п.), условий его труда и оснащенности средствами оргтехники. Применение метода позволяет найти то или иное решение и, в итоге, *выбрать* окончательное.

Жизненный цикл ПО

Фаза жизненного цикла	Стадия разработки по ГОСТ 19.102-77 ЕСПД	Состояние ПО в конце фазы
1. Системный анализ: а) выработка требований; б) разработка спецификаций	Разработка технического задания	Функциональная архитектура
2. Проектирование: а) проектирование архитектуры; б) детальное проектирование	Эскизное проектирование Техническое проектирование	Системная архитектура
3. Реализация (конструирование): а) кодирование; б) интеграция; в) тестирование (сертификация)	Рабочее проектирование	Коды программ Программное средство
4. Внедрение: а) изготовление; б) установка на ЭВМ пользователя	Внедрение	Программный продукт Программное обеспечение
5. Эксплуатация	Эксплуатация	
6. Сопровождение	Сопровождение	

Длительность ЖЦ сложных программных средств

- 1) программы с малой длительностью эксплуатации – создаются в основном для решения научных и инженерных задач, для получения конкретных результатов вычислений; они содержат от 1 до 10000 команд, разрабатываются одним специалистом или малой группой, не предназначены для тиражирования и передачи; жизненный цикл таких программ не превышает 3-х лет, время эксплуатации практически равно 0;
- 2) программы с большой длительностью эксплуатации – создаются для регулярной обработки информации и управления; такие программы содержат от 1 до 1000000 команд, обладают свойством независимости от разработчика, возможностью модификации в процессе сопровождения и использования различными программистами, допускают тиражирование, сопровождаются документацией как промышленное изделие, являются отчуждаемыми; жизненный цикл таких программ составляет 10 – 20 лет, из них 70 – 90 % занимает время эксплуатации.

Функциональная спецификация включает в себя:

- цель программы;
- граничные условия;
- описание функции (что программа должна делать и что, возможно, она будет делать);
- спецификацию входных и выходных данных;
- верификационные требования (установление тестовых случаев);
- тип и количество документов.

Временные затраты «Системный анализ» – «Проектирование» – «Реализация»

- «Выработка требований» – 10 %;
«Разработка спецификаций» – 10 %;
- «Проектирование» – 15 %;
«Кодирование» – 20 %;
- «Автономное тестирование» – 25 %;
«Комплексное тестирование» – 20 %.

Методы проектирования архитектуры

- На стадии проектирования архитектуры спецификации трансформируются в структуру системы. Методологии, используемые на этой стадии, делятся на две группы:
 - ориентированные на обработку
 - ориентированные на данные.

Методологии, ориентированные на обработку

- модульное программирование,
- проектирование с использованием потока данных,
- технология структурного анализа проекта,
- проектирование с использованием структур данных,
- метод HIPO

Модульное программирование.

Основные концепции

- каждый модуль реализует независимую функцию;
- каждый модуль имеет единственную точку входа/выхода;
- размер модуля по возможности необходимо минимизировать;
- каждый модуль может быть спроектирован и закодирован различными членами бригады программистов и может быть отдельно протестирован;
- вся система построена из модулей.

Проектирование с использованием потока данных

- идентифицируется поток данных, и отображается граф потока данных;
- идентифицируются входные, центральные и выходные преобразования;
- формируется иерархическая структура программы, использующая эти элементы;
- детализируется и оптимизируется структура программы, сформулированная на 3-м шаге.

Технология структурного анализа проекта SADT

- Основана на структурном анализе, предложенном Россом. SA – формальный графический язык, используемый для описания взаимосвязей между компонентами проекта, – иерархический набор диаграмм, причем каждый блок диаграмм раскрывается более детально с помощью другой диаграммы. Структура модели детализируется по мере разработки проекта.

Методология Варнье.

- три основные конструкции Дейкстры (последовательность, выбор, повторение данных) для построения структур данных, а в последствии также и структур программ.

ПОСЛЕДОВАТЕЛЬНОСТЬ	А		А	
	В	Выбор +	Повторение	
	С		В	(n раз)

Методология Варнье.

- При проектировании используются четыре представления проекта:
- диаграммы организации входных и выходных данных;
- диаграммы логического следования – представление логического потока этих данных;
- список инструкций, содержащий команды, используемые в проекте;
- псевдокод для описания конечных результатов проектирования.

Методология Варнье.

Методология Варнье может быть обобщена следующим образом:

- идентифицировать все входные данные системы;
- организовать входные данные в иерархическую систему;
- определить детальный формат каждого элемента входного файла и зафиксировать число их появлений;
- повторить шаги 1 – 3 для выходных данных;
- специфицировать детали программы, идентифицируя типы команд проекта в следующем порядке: чтение, ветвление, вычисление, выходы, вызовы подпрограмм;
- использовать диаграммы Варнье-Орра для показа логической последовательности инструкций;
- пронумеровать элементы логической последовательности и раскрыть их с помощью инструкций, записанных на шаге 5.

Методология Джексона.

Майкл Джексон в 1975 г. предложил представлять структуры данных и программ единым набором основных инструкций.

- Последовательность
- Выбор
- Повторение

Объект А состоит из объектов В, С, D в указанном порядке, слева направо. На нижнем уровне должно быть две или более конструкций.	Объект А состоит либо из объекта В, либо из объекта С, либо из объекта D (один из трех объектов). На нижнем уровне должно быть две или более конструкций.	Объект А состоит из нуля или более объектов С.
--	---	--

Методология Джексона.

Программа рассматривается как механизм, с помощью которого входные данные преобразуются в выходные. Действия при проектировании:

- идентифицировать и изобразить структуру входных данных и структуру выходных данных;
- изобразить структуру программы, соединяя изображения этих структурных элементов;
- определить операции, составляющие программу;
- превратить операции в текст программы.

Методология Джексона.

используется на следующих этапах проектирования:

- спецификация системы – определение форм файлов или баз данных для моделирования информации, описание программ;
- проектирование системы;
- начальный этап тестирования.

Инструментальная поддержка методики Джексона:

- программные средства, ориентированные на автоматизацию разработки документации сопровождения (M.Jackson Systems Ltd, Англия);
- пакет TIP-CREATE (корпорация Technology Information Products, США), автоматизирующий применение методики и нотации Джексона в рамках всего жизненного цикла программ.

Метод HIPO

- Метод иерархических диаграмм, развитый фирмой IBM. HIPO – Hierarchical Input Process Output Diagrams.
- Проект программной системы строится или документируется как иерархическая совокупность HIPO-диаграмм.
- Каждая диаграмма задает один программный модуль.

Заголовок		
Вход	Процесс	Выход
	A1	
Данные	A2	Данные
	B1	
	B2	

Основные характеристики метода HIPO:

- 1) способность представлять связь между данными и процессом обработки;
- 2) возможность декомпонировать систему иерархически, не вовлекая излишние мелкие детали;
- 3) использование только трех элементов: вход, обработка, выход.

Основная процедура проектирования с использованием метода НРО:

- начать с высшего уровня абстракции;
- идентифицировать вход, выход и обработку;
- соединить каждый элемент входа/выхода с соответствующей обработкой;
- документировать каждый элемент системы НРО-диаграммой;
- детализировать диаграмму, используя шаги 1 – 4.

Объектно-ориентированная методология проектирования

- Основана на концепциях упрятывания информации и абстрактных типах данных.
- Решаемая задача осмысливается, чтобы выделить естественные для нее типы данных. Абстрактные типы отражают уровень обобщения решаемой задачи и включают в себя не только определения структур данных, но и определение всех операций над ними.
- Цель – обеспечить возможность работы с абстрактными структурами так же естественно, как с общепринятыми данными.
- Благодаря этому осуществляется стандартизация инструментальных средств программиста и упрощается процесс проектирования сложных программ.

Основные действия методологии ООП

- 1) определить проблему;
- 2) развить неформальную стратегию – последовательность шагов, удовлетворяющих требованиям к системе;
- 3) формализовать стратегию:
 - идентифицировать объекты и их атрибуты;
 - идентифицировать операции над объектами;
 - установить интерфейсы;
 - реализовать операции.

Методология, основанная на проектировании концептуальных баз данных

- Относится к классу методологий, ориентированных на данные, и призвана дать проектировщику методические указания в процессе преобразования спецификаций в концептуальную схему базы данных. Проектирование начинается с определения наиболее общих естественно возникающих классов объектов и событий предметной области. Далее осуществляются итерации – описываются подклассы уже представленных классов и взаимодействия в этих классах.

Детальное проектирование

- Все ранее рассмотренные методы работают на уровне проектирования архитектуры – на модульном уровне. Метод, используемый на кодовом уровне, известен как структурное программирование.

Структурное программирование

- В структурном программировании программист мыслит как конструктор, в распоряжении которого есть ограниченное количество вполне определенных типовых конструкций (структур), причем заданы правила их соединения (сочленение, вложение, разложение на составляющие).
- Структура – это либо оператор языка программирования, либо некоторая абстракция по управлению, используемая программистом.

Классические структуры:

- последовательность,
- выбор,
- итерация.

Структура имеет один вход и один выход и значительно упрощает (с помощью унификации) процесс конструирования программы.

Структурированными считаются программы, которые не имеют циклов с несколькими выходами, переходов внутрь циклов или условных операторов, выходов изнутри циклов или условных операторов. Благодаря строгости стало возможным ввести в программирование формальные методы проектирования, доказательства правильности, автоматического синтеза программ и тестов.

Эффективным способом декомпозиции является абстракция – игнорирование ряда подробностей для сведения задачи к более простой.

- **Абстракция через параметризацию (АП)** позволяет, используя параметры, представить фактически неограниченный набор различных вычислений одной программой, которая есть абстракция всех этих наборов. Аппарат формальных и фактических параметров процедур в языках высокого уровня осуществляет АП.
- **Абстракция через спецификацию (АС)** позволяет абстрагироваться от процесса вычислений, описанных в теле процедуры, до уровня знания того, что данная процедура должна в итоге реализовать. Это достигается путем задания для каждой процедуры спецификации, описывающей эффект ее работы. При этом смысл обращения к процедуре становится ясным через анализ ее спецификации, а не тела процедуры.

Можно определить два различных вида абстракции, каждый из которых использует оба способа абстракции:

- **процедурная абстракция (ПА)** позволяет расширить заданную некоторым языком виртуальную машину новой операцией;
- **абстракция данных (АД)** позволяет добавить к виртуальной машине новые типы данных, состоящие из набора объектов и набора операций, характеризующих поведение этих объектов.

Процедурная абстракция, понятие и преимущества процедурной абстракции

- Процедура есть отражение набора значений входных аргументов в выходной набор результатов с возможной модификацией входных значений. Набор входных или выходных значений
- Процедура объединяет в себе абстракцию через параметризацию и абстракцию через спецификацию, т.к. абстрагирует отдельную операцию. В АП мы абстрагируемся от конкретных используемых данных. АП определяется в терминах формальных параметров. Фактические данные связываются с этими параметрами при использовании абстракции. Значения конкретных данных несущественны, важны лишь их количество и тип. (или оба) могут быть пустыми.
- Параметризация позволяет обобщить модуль, в результате чего уменьшается размер программы и, следовательно, объем модификаций.

АС наделяет программу двумя свойствами:

- **Локальность** – возможность реализации абстракции без необходимости анализа реализации какой-либо другой абстракции. Принцип локальности позволяет создавать программу из абстракций, созданных различными программистами, работающими независимо друг от друга.
- **Модифицируемость** – изменения в реализации абстракции не влияют на программу в целом, если спецификации абстракций остаются прежними.

Спецификация процедурной абстракции

- **Формальные спецификации** имеют точно определенное значение.
- **Неформальные спецификации** легче читать и понимать, но точное их содержание не всегда может быть установлено.

Спецификация процедуры состоит из заголовка и описания функции.

- Заголовок содержит имя процедуры, количество, порядок и типы входных и выходных параметров. Информация в заголовке чисто синтаксическая.
- В семантической части описывается смысл выполняемых процедурой действий на естественном языке, возможно расширенном привычными математическими обозначениями. Семантическая часть спецификации состоит из трех предложений: *requires* (требует), *modifies* (модифицирует), *effects* (выполняет). Предложения *requires* и *modifies* могут быть опущены.

Примеры спецификаций процедурных абстракций:

Concat=proc(a,b: String) returns(ab: String)

effects По возврату *ab* есть новая строка, содержащая символы из *a* (в порядке их расположения в *a*), за которыми следуют символы из *b* (в порядке их расположения в *b*).

RemoveDupls=proc(a: array[real])

modifies *a*

effects Удаляет из массива *a* все повторяющиеся элементы, порядок следования оставшихся элементов может измениться.

Search=proc(a:array[real],x:real) returns(i:int)

requires Массив *a* упорядочен по возрастанию.

effects Если элемент *x* принадлежит массиву *a*, то возвращается значение *i* такое, что $a[i]=x$; в противном случае возвращается значение *i* на единицу большее, чем значение верхней границы массива *a*.

Замечания по реализации процедурных абстракций.

- При реализации ПА могут быть оставлены неопределенными некоторые выполняемые процедурой функции. Такая процедура называется недоопределенной – для некоторых значений входных параметров на выходе вместо единственного правильного результата имеется набор допустимых результатов. Реализация может ограничивать этот набор одним значением, однако он может быть любым из числа допустимых. В спецификации должны быть оговорены все существенные для пользователя подробности, все остальные могут оставаться неопределенными.
- Процедура должна обладать хорошо определенным и легко объяснимым назначением, независимым от контекста ее использования.
- Частичная или общая абстракция? Общие процедуры (не имеют ограничений на входные параметры) являются более безопасными, чем частичные, так как неудовлетворение входных требований в частичной процедуре может привести к неверной работе программы. В свою очередь, частичные процедуры чаще более эффективны в реализации, чем общие. Критерии выбора между частичной и общей абстракцией:
 - 1) эффективность;
 - 2) корректное выполнение с меньшим числом потенциальных ошибок.

Более обобщенные (параметризованные) процедуры

- Процедура *Search* работает с произвольным массивом вещественных чисел. Она была бы более полезной, если бы смогла работать с массивами данных различного типа. Такого обобщения можно достичь, расширяя абстракцию через параметризацию и используя типы данных в качестве параметров. При этом некоторые значения параметров могут не иметь смысла, например, массив может быть отсортирован только в том случае, если элементы, принадлежащие к типу, упорядочены. Ограничения на параметры типа предполагают набор некоторых заданных операций над ними. Спецификация абстракции должна содержать эти ограничения в наборе *requires*.

Search=proc[*t.type*](*a: array*[*t*],*x:t*) ***returns***(*i: int*)

requires *t* имеет операции: *equal*, *less: proctype*(*t,t*) ***returns***(*boolean*);

t упорядочивается через *less*, и массив *a* упорядочен по возрастанию через *less*.

effects Если элемент *x* принадлежит массиву *a*, то возвращается значение *i* такое, что *a*[*i*]=*x*; в противном случае – значение *i* на единицу больше, чем значение верхней границы массива *a*.

Абстракция данных. Понятие абстракции данных

- Процедурные абстракции дают нам возможность добавлять в базовый язык новые операции. Необходимость добавления в базовый язык новых типов данных удовлетворяется абстракцией данных.
- Область применения программы определяет набор новых типов данных. Например, при создании компилятора или интерпретатора полезны стеки и символьные таблицы, в аналитических вычислениях – полиномы, в матричной алгебре – векторы и матрицы. Новые типы данных должны включать в себя абстракцию через параметризацию и абстракцию через спецификацию. Смысл АП – использование параметров, АС – включение операций над типом в сам тип: абстракция данных = данные + операции.

Спецификация абстракции данных

Состоит из заголовка, определяющего имя типа и имена его операций, и двух секций – описания и операций:

- *Dname=***data type is** список операций
- **Описание** – описание абстракции данных
- **Операции** – спецификации всех операций над типом
- *end Dname*
- В секции описания также должно быть указано, изменяемый или неизменяемый это тип (операции изменяемого типа могут модифицировать тип).

Пример. Спецификация очереди целых чисел.

TQueue = **data type is** *Create, Empty, Insert, Remove, Destroy*

Описание

Тип данных *TQueue* используется для хранения целых значений. Элементы включаются в очередь и исключаются из очереди только по принципу «первым пришел – первым обслужен». Тип изменяемый.

Операции

Create = **proc()** *returns*(*TQueue*)

effects Возвращает новую очередь без элементов в ней.

Empty = **proc**(*q: TQueue*) *returns*(*bool*)

effects Возвращает значение *true*, если в очереди *q* нет элементов, и значение *false* в противном случае.

Insert = **proc**(*q: TQueue; NewValue: int*)

modifies *q*

effects Добавляет элемент со значением *NewValue* в конец очереди *q*.

Remove = **proc**(*q: TQueue*) *returns*(*int*)

requires Очередь *q* не пуста.

modifies *q*

effects Возвращает элемент из начала очереди *q* и удаляет этот элемент из очереди.

Destroy = **proc**(*q: TQueue*)

modifies *q*

effects Удаляет из памяти очередь *q*.

end *TQueue*

Обсуждение реализации абстракции данных.

1. **Изменяемость.** Абстракции данных могут быть либо изменяемыми (списки, массивы), либо неизменяемыми (числа, полиномы). Это свойство типа, а не его реализации. Реализация же должна поддерживать это свойство абстракции.
2. **Классы операций:**
 - а) примитивные конструкторы – создают объекты соответствующего типа, не используя никаких объектов в качестве аргументов (операция *Create* очереди);
 - б) конструкторы – используют в качестве аргументов объекты соответствующего им типа и создают другие объекты того же типа, например операция копирования очереди:
Copy = proc(q: TQueue) returns(TQueue)
effects Возвращает копию очереди *q*.
 - в) модификаторы – модифицируют объекты соответствующего им типа (операции *Insert*, *Remove* для очереди); только изменяемые типы могут иметь модификаторы;
 - г) наблюдатели используются для получения информации об объекте; в качестве аргумента они используют объект соответствующего им типа, а возвращают аргумент другого типа, например операции *Empty* или *Size* для очереди.
3. **Полнота типа данных.** Тип данных является полным, если он обеспечивает достаточно операций для эффективной работы с объектами этого типа. В общем случае абстракция данных должна иметь примитивные конструкторы, наблюдатели и либо конструкторы (если она неизменяемая), либо модификаторы (если она изменяемая). Тип данных должен иметь достаточно большой набор операций для его использования.