Математическое моделирование

Лекция 1

Вычислительный эксперимент

Вычислительный эксперимент — метод изучения устройств или физических процессов с помощью математического моделирования. Он предполагает, что вслед за построением математической модели проводится ее численное исследование, позволяющее изучить поведение исследуемого объекта в различных условиях или в различных модификациях.

Численное исследование модели дает возможность определять характеристики процессов, оптимизировать конструкции или режимы функционирования проектируемых устройств. В ходе вычислительного эксперимента исследователь открывает новые свойства, которые опасно получить в ходе натурных исследований.

Сфера применения.

Вычислительный эксперимент занимает промежуточное положение между натурным экспериментом и аналитическим исследованием.

Натурный (физический) эксперимент при надлежащей постановке можетдать исчерпывающие и надежные результаты. Однако во многих случаях предпочтение отдается вычислительному эксперименту.

В **вычислительном эксперименте** в роли опытной установки выступает не конкретное физическое устройство, аматематическая модель . Ее построение и последующие модификации, как правило, требуют существенно меньших затрат, чем подобные манипуляции над реальным объектом.

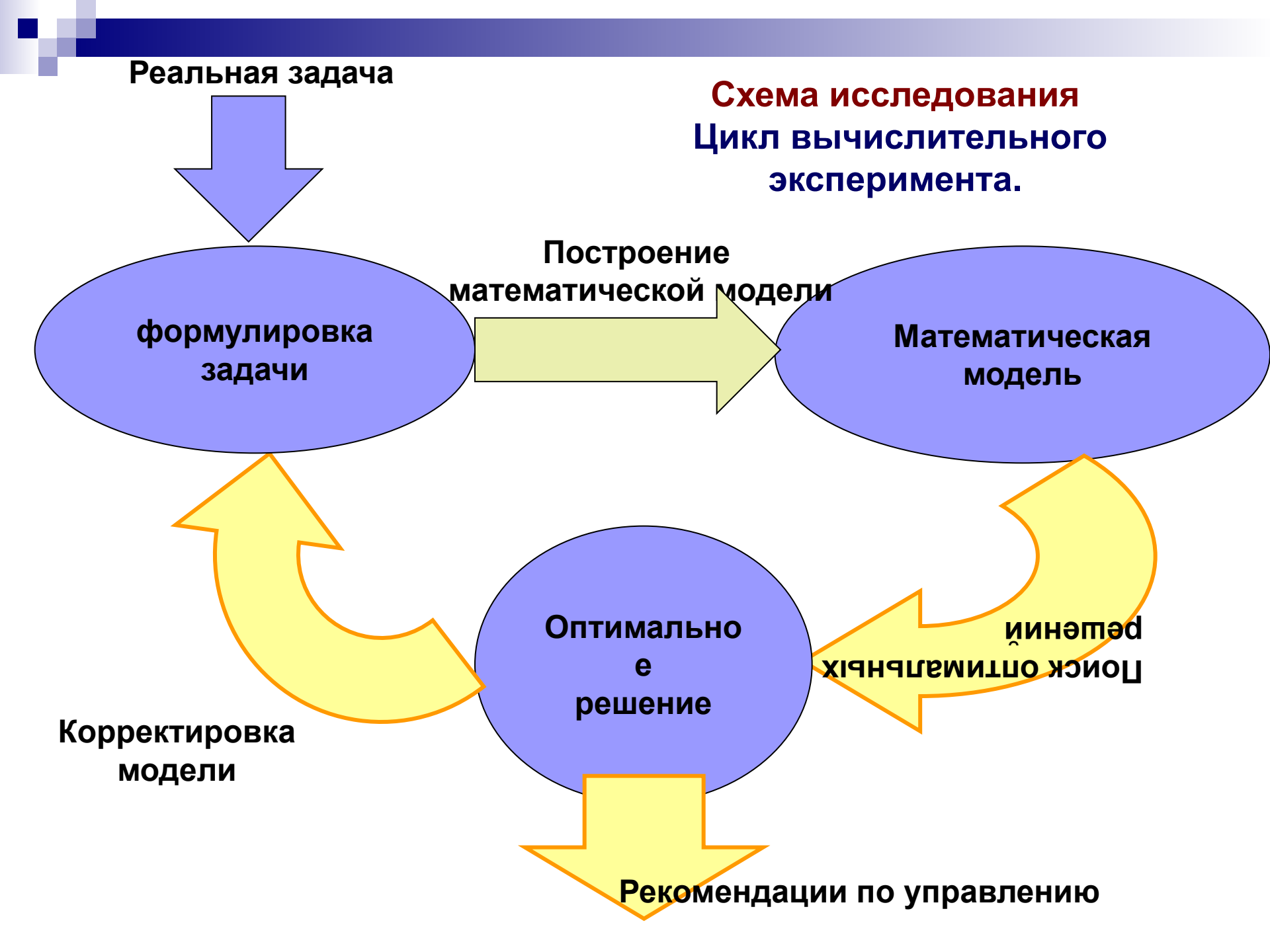
Кроме того, в опытной установке нередко просто невозможно бывает воссоздать некоторые критические режимы или экстремальные условия. Поэтому математическое моделирование может оказаться практически единственно возможным способом исследования процесса .

При **аналитическом подходе** так же, как и в вычислительном эксперименте, строится математическая модель. Но исследуется эта модель исключительно посредством аналитических выкладок, без привлечения каких-либо численных методов. Если аналитических выкладок оказывается достаточно, то данный подход приводит к строгому точному решению.

Однако на практике, как это ни парадоксально, аналитическому подходу обычно отводится роль инструмента для (сравнительно быстрого) получения грубых оценок. Объясняется это тем, что аналитическими выкладками удается ограничиться только для несложных, сильно упрощенных моделей реальных процессов. Получаемое тут строгое аналитическое решение оказывается весьма далеким от совершенства. Напротив, численные методы, применяемые в вычислительном эксперименте, дают возможность изучать более сложные модели, достаточно полно и точно отражающие исследуемые процессы.

Отмеченные достоинства вычислительного эксперимента вывели его в число основных методов исследования таких крупных физических и инженерно-технических проблем, как задачи ядерной энергетики, освоения космического пространства и др.

Программные комплексы, обслуживающие вычислительный эксперимент, объемны и сложны, в их создание вовлечено большое количество программистов. Поэтому особую актуальность приобретает изучение готовых программных продуктов , которые позволяют исследователю получать результаты не в даваясь в сложности программирования .


Реальная задача
Схема исследования
Цикл вычислительного эксперимента.
Построение математической модели
формулировка задачи
Математическая модель
Оптимальное решение
Поиск оптимальных решений
Корректировка модели
Рекомендации по управлению

Этап 1. Построение математической модели (составление уравнений, описывающих исследуемое явление).

Этап 2. Выбор численных методов расчета (построение дискретной модели, аппроксимирующей исходную математическую задачу, построение разностной схемы, разработка вычислительного алгоритма и т. д.).

Этап 3. Создание программы, реализующей вычислительный алгоритм.

Этап 4. Проведение расчетов и обработка полученной информации.

Этап 5. Анализ результатов расчетов, сравнение (если это возможно) с натурным экспериментом, идентификация параметров модели.

Обычно на последнем (5-м) этапе исследователь приходит к заключению о том, что необходимо внести определенные изменения в решения, принятые на этапах 1, 2 или 3.

Так, может выясниться, что построенная модель недостаточно хорошо отражает особенности исследуемого явления. В этом случае модель корректируется, вносятся соответствующие поправки в численные методы и реализующие их программы и выполняется новый расчет. Тем самым цикл вычислительного эксперимента воспроизводится в полном объеме.

При анализе результатов могут быть выявлены недостатки используемых численных методов. Изменение методов влечет за собой изменение соответствующих программ и т. д. Иначе говоря, цикл повторяется в несколько сокращенном виде (этапы 2–5).

Наконец, может оказаться неудачным некоторое программное решение. Пересмотр таких решений приводит к повторению этапов 3–5.

Особенности программирования. Разумеется, циклы, подобные циклу вычислительного эксперимента, возникают практически в любом сложном программном продукте . Самая первая версия программы обычно чем-нибудь не удовлетворяет разработчика или заказчика, и приходится уточнять постановку задачи, улучшать отдельные алгоритмы и т. д. Однако в большинстве случаев достаточно бывает выполнить несколько циклов, требующих сравнительно небольших усилий, и программа обретает желаемый вид.

Совершенно иначе обстоит дело в вычислительном эксперименте. Здесь программа мыслится как экспериментальная установка, от первых опытов с которой вряд ли следует ожидать сколько-нибудь интересных результатов. Данные первых опытов послужат всего-навсего отправной точкой длительного процесса изменений и усовершенствований установки, который только и способен привести к достаточно обоснованным заключениям об исследуемом объекте.

Таким образом, появление первоначальной версии программы лишь в самых общих чертах намечает направление исследований. Основная же работа по программированию связана с многократными модификациями программы, отражающими эволюцию математической модели и методов ее расчета. Число циклов вычислительного экспериментанередко достигает десятков тысяч. Поэтому рациональная организация таких модификаций — ключ к эффективному программированию данного класса задач.

Продолжая параллель с натурным экспериментом, можно заметить, что там обычно не спешат выбрасывать на свалку отдельные узлы, изъятые или замененные в экспериментальной установке при проведении очередного опыта: они еще не раз могут пригодиться впоследствии. Аналогично и решения (и соответствующие им фрагменты программ), пересматриваемые на очередном цикле вычислительного эксперимента, как правило, не отбрасываются и могут использоваться затем для других расчетов. Например, метод, оказавшийся непригодным для одной модели, вполне может подойти для расчета следующей модели и т. д.

Итак, главное направление деятельности программиста, занятого вычислительным экспериментом, — не создание новых, а развитие существующих программ. Это развитие осуществляется не за счет замены имеющихся модулей их более совершенными версиями, а за счет расширения: включения в программный фонд все новых и новых модулей, отражающих различные решения, принимаемые в ходе эксперимента.

Накапливаемые модули могут затем комбинироваться в самых разнообразных сочетаниях, позволяя тем самым провести достаточно систематическое и глубокое исследование. Потребность в подобных манипуляциях над модулями регулярно возникает в связи с тем, что исследователь постоянно конструирует новые варианты модели, сочетающие в себе те или иные выполнявшиеся когда-либо изменения или уточнения. Таким образом, интересующая нас многовариантность программ вычислительного эксперимента является закономерным следствием изначальной многовариантности модели.

Организовать эффективное функционирование и развитие столь обширного, сложного и специфичного программного хозяйства очень нелегко. Тем не менее жизнь показала, что все возникающие здесь трудности вполне преодолимы — методом вычислительного эксперимента были успешно решены многие важные практические задачи. История программирования задач вычислительного эксперимента насчитывает свыше трех десятилетий, и за это время накоплен весьма значительный опыт, позволяющий говорить о существовании определенной технологии работы с многовариантными программами. Эта технология оказалась достаточно надежной и эффективной; именно добротностью применявшейся технологии объясняется жизнестойкость известных программных реализаций вычислительного эксперимента.

В задачах вычислительного эксперимента в полной мере проявляются практически все специфические особенности многовариантных программ. В то же время вычислительный эксперимент является наиболее крупным потребителем технологии многовариантности. Поэтому выражения «программирование задач вычислительного эксперимента» и «создание многовариантных программ» иногда будут использоваться как синонимы.

Поэтапная разработка программ (Программирование «вширь»).

Расчленение процесса создания программы на ряд относительно самостоятельных этапов наделяет этот процесс многими полезными качествами.

Разбиение программы на модули облегчает ее восприятие, поскольку каждый из модулей, как правило, существенно проще, чем программа в целом. Подобный эффект достигается и при разбиении процесса разработки на этапы: процесс становится обозримым, поскольку на каждом из этапов в рассмотрение вовлекается лишь некоторая часть из всего множества конструкций программы.

При поэтапной разработке можно на ранних стадиях «обкатать» наиболее тонкие алгоритмические решения и застраховаться тем самым от крупных переделок программы на заключительной стадии.

Количество и состав этапов, выполненных к определенному сроку, позволяют объективно оценить степень готовности разрабатываемой программы, что дает возможность своевременно скорректировать усилия как разработчиков, так и заказчика.

Иногда на ранних этапах удается создать небольшой макет будущего программного продукта, реализующий основные функциональные возможности. В процессе пробной эксплуатации такого макета заказчик уточняет спецификации программы, а затем на последующих этапах неэффективные части макета заменяются эффективными.

Теперь, убедившись в полезности расчленения процесса разработки на этапы, перейдем к рассмотрению известных стратегий такого расчленения.

В теории и практике системного анализа наиболее популярны стратегии **«сверху вниз»** и **«снизу вверх».** К этим стратегиям ведет следующее вполне логичное построение. Прежде всего постулируется, что включает в себя взаимодополняющие процессы анализа и синтеза.

Анализ трансформируется в движение «сверху вниз» — расчленение решаемой проблемы на относительно независимые аспекты и, соответственно, расчленение разрабатываемой программы на модули. Синтез позволяет соединять между собой «снизу вверх» вновь создаваемые части нижнего уровня, а также накопленный ранее программный материал.

Поэтапную разработку сравнительно легко удается скрестить как с тем, так и с другим направлением. Наиболее хорошо известен результат скрещивания поэтапной разработки с направлением «сверху вниз», который получил название пошаговое уточнение (step-wise refinement).

При пошаговом уточнении очередной этап разработки оставляет после себя некий полуфабрикат создаваемой программы, где верхняя, заголовочная структура уже в основном сформирована, а «на нижних этажах» встречаются еще нереализованные части. На месте каждой из нереализованных частей располагается пара: заглушка и спецификация.

Заглушка служит для отладки, имитируя в весьма ограниченном объеме поведение отсутствующего пока куска программы. На следующем этапе разработчик выбирает одну или несколько нереализованных частей и уточняет их, т. е. подставляет на их место тексты алгоритмов (реализующих спецификации), которые, в свою очередь, могут содержать заглушки и спецификации. Так постепенно, этап за этапом, происходит реализация (уточнение) недостающих частей. На выходе последнего этапа уточнения появляется полноценный текст программы.

Похоже выглядит и поэтапная стратегия «снизу вверх». Для отладки опять потребуются заглушки. Здесь они имитируют поведение не подчиненных частей, а среды, в которой будет функционировать отлаживаемый нижележащий фрагмент в окончательной версии создаваемой программы.

Обе стратегии можно применять на различных стадиях жизненного цикла программы. Можно сначала с их помощью выполнить проектирование, полностью определив состав и интерфейсы входящих в программу модулей, и лишь затем приступить к реализации, которая, в свою очередь, пойдет по направлению «сверху вниз» и/или «снизу вверх». Можно впасть и в другую крайность, торопясь программировать и отлаживать еще сырые модули незаконченного проекта.

Такие полярные варианты взаимодействия процессов проектирования и реализации оказываются применимыми только для программ небольшого или среднего размера. Создание же крупной программы обычно связано с поиском разумного компромисса между этими вариантами.

Стратегия «вширь». При программировании «вширь» главным конфигурационным ориентиром провозглашается набор однородных модулей. Желательно, чтобы суммарный объем выделенных однородных модулей оказался достаточно большим, они должны оттянуть на себя основную массу работ по созданию программы. Практика показывает, что главное — это поверить в перспективность однородных наборов; после этого, как правило, однородные модули вычленяются относительно легко.

Приведем характерные примеры больших однородных наборов.В текстовом редакторе — процедуры, выполняемые при нажатии функциональных клавиш. В оптимизационных задачах — реализации методов, используемых для приближения к экстремуму заданной функции. В задачах математической физики — алгоритмы обработки различных типов узлов сетки. При моделировании некоторой установки или явления — модули, учитывающие различные физические эффекты.

В одной программе может быть выявлено несколько однородных наборов.

После того как однородные модули выявлены, приступают к программированию, которое разбивается на этапы. На первом этапе создается лишь минимальное число представителей каждого из выделенных однородных наборов. Если поиск однородности проводился достаточно энергично, то обычно оказывается, что объем работ первого этапа реализации сравнительно невелик.

Несмотря на это программа, получаемая на первом этапе, технологически уже вполне самостоятельна: для ее отладки не требуется никаких заглушек. В приведенных выше примерах однородных наборов будет получен, скажем, операционная система, обслуживающая лишь один класс периферийных устройств, редактор, реагирующий на единственную клавишу Конец сеанса, и т. д.

Последующие этапы разработки заключаются в непосредственном программировании все новых и новых однородных модулей. Если место размещения однородного набора оформить в виде наборного гнезда то подключение новых модулей будет происходить безболезненно не требуя редактирования текстов существующих программ. Подобно рассмотренному в предыдущей главе сменному вариантному модулю, вновь создаваемый модуль для наборного гнезда программируется в благоприятной атмосфере, его интерфейс с средой не только заранее разработан, но и в значительной степени отлажен на однородных предшественниках этого нового модуля.

Итак, предложив новый конфигурационный ориентир, нам удалось построить перспективную стратегию поэтапной разработки программы. Пока нельзя, к сожалению, сказать, что программирование «вширь» завоевало прочные позиции в системе программистских ценностей. Из крупных отечественных работ, пропагандирующих близкие конструкции, можно назвать только «вертикальное слоение».

Недостаточное распространение программирования «вширь» отчасти объясняется объективными причинами. Эта стратегия может принести весомую пользу и в неприспособленной операционной среде. Однако для того чтобы почувствовать все ее преимущества, требуются специализированные средства системной поддержки. Но, как уже не раз отмечалось, в традиционных операционных средах подобные средства обычно отсутствуют.

Программные средства для математического моделирования

Лекция 2

Программные средства для моделирования можно разделить на две группы.

К первой отнесем пакеты, предназначенные для решения сложных промышленных и научно-исследовательских задач большими производственными или научными коллективами.

В таких проектах ведущую роль играет организация работ:

- хорошо налаженное взаимодействие между отдельными группами;
- быстрый доступ к многочисленным экспериментальным данным и библиотекам программ;
- тщательное документирование и тестирование;
- многовариантные расчеты.

При этом обычно используются хорошо изученные модели, которые лишь модифицируются и приспосабливаются для решения конкретных задач.

В некотором смысле это относится и к большим научным проектам, когда успех во многом предопределен предварительными исследованиями, но для получения окончательных результатов требуется хорошо скоординированная совместная работа. Пакеты первой группы условно назовем «промышленными».

Однако такие проекты невозможны без предварительных исследований, выполняемых отдельными учеными или проектировщиками.

Стартовой точкой в них является гипотеза, а основной задачей — ее проверка.

Исходным материалом служат плохо формализованные модели, то есть модели, чьи свойства еще не волне осознаны.

В самом начале исследований обычно ничего другого предложить невозможно, как двигаться вперед на ощупь, практически без плана, формируя его по мере накопления материала.

Главное — это пробовать и видеть отклик. Это означает, что необходимо уметь организовывать и поддерживать непрерывную обратную связь между исследователем и исследуемой моделью.

Аналогичная задача возникает и при обучении, когда необходима обратная связь между обучающей программой и учеником, или когда учитель прямо на занятии с помощью модели объясняет суть явления.

Промышленные пакеты слишком сложны и громоздки для проведения исследований на ранних стадиях и тем более обучения, для этого нужны специальные программные средства.

Именно они и образуют вторую группу пакетов. Назовем пакеты второй группы универсальными, подчеркивая этим, что они уступают по количеству уникальных возможностей промышленным, зато более просты для освоения и доступны отдельному исследователю при решении относительно несложных задач из практически любой прикладной области.

Под несложными мы понимаем не простые задачи, а задачи посильные одному разработчику, не являющемуся специалистом в области программирования и вычислений.

В универсальных пакетах нужны разнообразные численные библиотеки, способные справиться с широким спектром проблем, а не методы, ориентированные на узкий класс задач.

Для них нужны графические библиотеки, обеспечивающие показ изучаемого явления с разных сторон, а не одним, принятым в конкретной области, способом и, конечно же, поддержка интерактивного вмешательства в ход компьютерного эксперимента.

Изучаемые с помощью универсальных пакетов модели можно условно разделить на модели для:

- естественнонаучных областей;
- технических объектов.

В первом случае мы обычно имеем дело с моделью, сведенной к одной, итоговой системе уравнений, или другими словами с однокомпонентной моделью, а во втором — со структурированной, многокомпонентной моделью, итоговая система для которого должна строиться автоматически, используя описания отдельных компонентов.

И среди однокомпонентных, и среди многокомпонентных, наибольший интерес представляют модели, чье поведение меняется во времени в зависимости от наступающих событий. Их часто называют гибридными системами. В отечественной литературе также используются синонимы — непрерывно–дискретные, системы с переменной структурой, реактивные, событийно–управляемые. Еще недавно единственным способом изучения гибридных систем было исследование их отдельных фаз или режимов и «склеивание» общего поведения вручную, подобно тому, как мы склеиваем панораму из отдельных фотографий. Теперь появилась возможность моделировать глобальное поведение таких объектов.

Под гибридными системами мы понимаем динамические системы, имеющие различное поведение в различных областях фазового пространства.

Их фазовая траектория, в зависимости от происходящих событий, оказывается то в одной области, то в другой.

Таким образом, к гибридным можно отнести классические динамические системы, чье фазовое пространство разбивается на ячейки с различным поведением, системы с разрывными правыми частями, и системы, у которых меняется размерность в различных областях фазового пространства.

Достижение фазовой траекторией границы областей будем называть событием, приводящим к смене поведения.

Каждой области можно поставить в соответствие вершину некоторого графа, а его направленные дуги трактовать как возможные пути смены текущего локального поведения.

Границы областей обычно задают с помощью предикатов, которые приписываются соответствующим дугам графа.

Таким образом, гибридная система может быть представлена в виде графа, вершинам которого поставлены в соответствие классические динамические системы, и одна из вершин помечена как начальная, а дугам — условия смены поведения и мгновенные действия, выполняемые при смене поведения. Такая формализация называется гибридным автоматом.

Глобальное поведение гибридной системы определяется всеми возможными путями, которые можно построить из начальной вершины.

По мере движения модельного времени, фазовая траектория пересекает границы областей, и меняется вид решаемых уравнений.

Можно также представить себе ситуацию, когда какая–либо из областей будет покидаться системой немедленно, как только система туда попадет. В этом случае гибридная система будет демонстрировать как длительные, «непрерывные» поведения, так и мгновенные, «дискретные».

Необходимость обеспечения обратной связи между исследователем и моделью опять же приводит нас к событийно–управляемым системам и дополнительно заставляет проводить и визуализировать вычислительный эксперимент в реальном времени.

Назовем такой способ познания действительности — *активным компьютерным экспериментом*, в отличие от традиционного пассивного вычислительного эксперимента, план которого может быть составлен заранее.

Отличительной чертой современных пакетов для математического моделирования, является объектно-ориентированный подход, позволяющий обеспечить еще одно очень важное и характерное для научных исследований и обучения требование — возможность легко пополнять и модифицировать разрабатываемую библиотеку моделей, представляющую обычно последовательность все более сложных моделей, свойства которых приходится постоянно сравнивать.

Модели, используемые на ранних стадиях научных исследований и проектирования, и практически все модели, используемые в образовании, названные несложными, можно в свою очередь разделить на группы.

Каждая из таких групп обычно является основной для определенной категории пользователей, у которых уже сложились свои требования к возможностям пакетов, и свое виденье способа общения с ними.

Классификация моделей возникла потому, что для моделей различных типов существуют различия в технологии моделирования.

Поэтому мы выделяем:

- изолированные однокомпонентные системы;
- открытые однокомпонентные системы;
- изолированные многокомпонентные системы;
- открытые многокомпонентные системы;

Изначально вычислительные пакеты были ориентированны на эксперименты с изолированными моделями, изучать которые можно с помощью пассивного вычислительного эксперимента, то есть, по сути, речь идет об иной форме столь памятного еще многим пакетного режима.

И именно эту черту унаследовали знаменитые Matlab, Maple, Mathematica и другие математические пакеты.

Существующие в них диалоги активно используются в основном при создании модели, но не при ее изучении.

Сегодня, с появлением потребности в активном вычислительном эксперименте, ситуация принципиально изменилась.

Диалог стал активно использоваться так же и при проведении эксперимента.

Таким образом сейчас, основным типом изучаемых моделей становятся открытые системы.

Аналогичная ситуация наблюдается и со вторым «измерением» нашей классификации.

В тех же прекрасно работающих с однокомпонентными моделями математических пакетах практически невозможно создать многокомпонентную модель.

Пакет Simulink, позволивший быстро создавать модели из типовых блоков, был обречен на успех, особенно среди инженеров.

Однако реализованный в нем подход устаревает, так как восходит к эре аналоговых машин.

В последнее время все больше внимания начинает уделяться многокомпонентным моделям различной физической природы и их автоматическому синтезу с использованием других подходов (например, проект Modelica), и хотя в отдельных областях есть несомненные достижения, задачи здесь еще только ставятся.

Изолированная однокомпонентная гибридная система

В изолированных однокомпонентных гибридных системах смена поведения зависит исключительно от внутренних событий.

Например, в уравнении математического маятника коэффициенты могут меняться периодически, или при достижении маятником заданного положения.

Модели этого типа достаточно сложно реализуются в математических пакетах.

Интерфейс пользователя для изолированных однокомпонентных гибридных систем требует еще одного окна — например, окна Редактора карт состояния (Simulink, StateFlow), или их расширения — гибридных карт состояний (Model Vision, Anylogic).

Существует много различных способов описать смену фаз или режимов — это и сети Петри, и графы связей, но наиболее перспективным является использование гибридных автоматов.

Для описания дискретных действий в гибридных моделях требуется расширенный набор типов переменных, а также некоторый набор алгоритмических операторов (как минимум, оператор присваивания, условный оператор и оператор цикла).

Открытая однокомпонентная гибридная система

Это наиболее востребованная практикой модель, которая позволяет проводить активный компьютерный эксперимент и может быть использована в качестве библиотечной компоненты.

Такая модель имеет доступные извне переменные, посредством которых она может взаимодействовать с окружающим миром: другими компонентами и с экспериментатором.

Например, используя модель маятника в виде изолированной системы, можно на ее основе построить нужную нам открытую систему, объявив, например, длину маятника не коэффициентом, а входом, и связав его с движком в окне двумерной анимации.

Переход от изолированной модели к открытой может быть нетривиальным: например, маятник, у которого длина стержня непрерывно меняется, описывается совсем другими уравнениями.

Окна переменных («живые таблицы») и анимации (по крайней мере, 2D) становятся интерактивными: пользователь может с помощью ползунков, круговых регуляторов, кнопок и тумблеров активно воздействовать на модель во время эксперимента. Очевидно, что активный эксперимент требует синхронной визуализации.

Структурная многокомпонентная система с ориентированными блоками

Эта модель строится из ориентированных блоков, т.е. если все ее внешние переменные можно классифицировать либо как входы, значения которых не могут изменяться внутри компоненты, либо как выходы, значения которых, напротив, могут изменяться только внутри компоненты.

Модели этого класса наиболее приспособлены для изучения и проектирования технических объектов, собираемых из типовых блоков. Такой способ проектирования принят во многих пакетах — Simulink, Dymola, Model Vision Studium, Anylogic.

Отличия технологий проектирования можно свести к двум:

Первое связано с возможностью пользователя самостоятельно конструировать и добавлять новые элементарные блоки, используя входной язык пакета (Model Vision Studium, Anylogic, Dymola);

Либо для этого необходимо обращаться к «низкоуровневым» процедурным языкам (Simulink), т.е. с возможностью использовать механизмы наследования и полиморфизма при проектировании на уровне входного языка (Modelica, Model Vision).

К новым элементам интерфейса таких моделей следует отнести окно Структуры.

Автоматическое построение совокупной системы уравнений в этом случае не вызывает никаких проблем.

При визуализации многокомпонентных систем возникает интересная и еще не до конца решенная проблема об автоматическом построении анимационного образа всей многокомпонентной системы, составленного из анимационных образов отдельных компонентов.

Примером простейшей из таких систем может служить маятник в падающем лифте.

Многокомпонентная гибридная система переменной структуры

Модели этого наиболее сложного типа не реализованы в полном объеме ни в одном из известных пакетов. Речь идет о событийно–управляемых многокомпонентных иерархических моделях переменной структуры, построенных из блоков с переменными вход, выход, состояние и контакт.

Традиционные математические пакеты, такие как MathCAD, Maple, Mathematica наилучшим образом приспособлены для проведения хорошо спланированного пассивного вычислительного эксперимента для естественнонаучных дисциплин, когда производится серия длительных расчетов с заранее выбранной формой обработки и визуализации результатов для большого числа значений параметров моделей, или различных модификаций моделей.

В таких многовариантных расчетах, накладные расходы, связанные с написанием специальной программы на языке пакета, управляющей экспериментом, с лихвой окупаются той легкостью, с которой возможно повторить все вычисления заново, при внесении изменений в исходную модель.

В учебном же процессе, преимущества математических пакетов наиболее полно проявляются при решении простых, хорошо продуманных преподавателем задач, иллюстрирующих одно конкретное свойство изучаемого объекта. Программирование таких задач сводиться к написанию относительно небольших по объему программ, состоящих в основном из макро–операторов, что позволяет получить нужный результат почти без видимых усилий, и сосредоточить свое внимание только на изучаемом явлении.

Особое место среди пакетов, предназначенных для изучения естественнонаучных дисциплин, занимают специализированные учебные пакеты, такие как Modellus, xyZET.

Они ориентированы на школьников, школьных преподавателей, и студентов университетов, где компьютерное моделирование изучается как один из возможных способов познания действительности.

Эти пакеты просты для изучения, содержат достаточно выразительную графику, и прекрасно приспособлены для изучения простых неструктурированных моделей, заданных системами дифференциальных уравнений невысоко порядка.

Среди существующих пакетов для моделирования многокомпонентных объектов наиболее распространенными и известными являются коммерческие пакеты, требующие больших усилий для их освоения, и предназначенные в основном для проектирования больших промышленных систем.

Стоимость этих пакетов, особенно ориентированных на конкретную прикладную область, достаточно высока, что обычно заставляет пользователя, будь то отдельный ученый, или университет, выбирать какой–либо один пакет на долгие годы.

К числу универсальных пакетов, которые используются и для обучения, можно отнести компоненты пакета Matlab, а именно Simulink и StateFlow, и построенные по их образу и подобию пакеты VisSim, пакет МВТУ.

Пакет Simulink пользуется заслуженной популярностью, а его язык блок–схем во многих публикациях стал средством описания изучаемых объектов. Но именно он и подвергается наибольшей критике со стороны авторов языка Modelica, выступающих за «физический» способ построения моделей из неориентированных блоков.

В пакете Simulink практически невозможно реализовать гибридное поведение. Не решаясь на коренные переделки, для реализации гибридного автомата, авторы пакета создают компоненту StateFlow, позволяющую к существующей модели добавить карту состояния Харела, управляющую сменой поведения модели. Такой способ несколько ранее был реализован в пакете Model Vision 2.1, но от него решено было отказаться, так как модели становились чрезвычайно сложными для восприятия. Это приводило к многочисленным ошибкам при проектировании, и возникали сложности численного моделирования в окрестности точки смены поведения из–за возможных разрывах правой части дифференциальных уравнений.

Среди пакетов для моделирования специально выделим пакет Dymola, поддерживающий продолжающий развиваться универсальный язык моделирования Modelica.

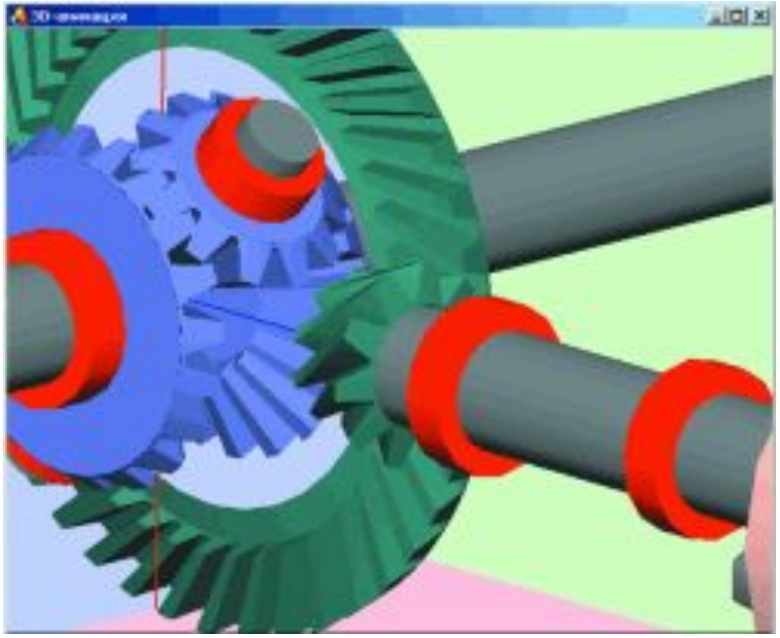
Несмотря на то, что Modelica поддерживает как ориентированные, так и неориентированные блоки, основным достоинством авторам пакета видится возможность соединять неориентированные блоки с помощью контактов.

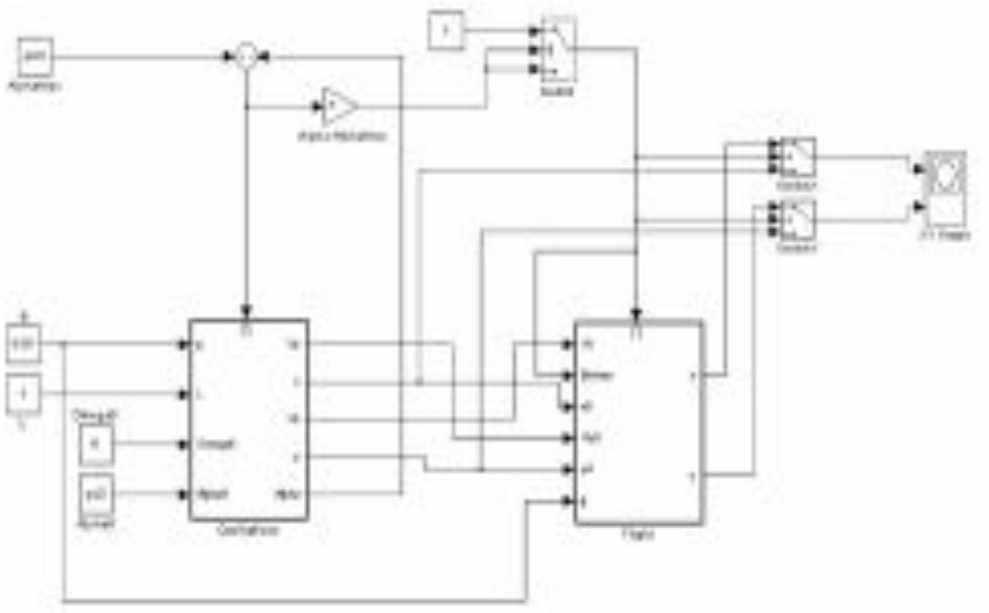
И здесь, несомненно, достигнуто очень много. Однако и здесь смена поведения остается камнем преткновения. Если проследить за развитием языка в ежегодно публикуемых документах, то можно увидеть, как сокращаются возможности сменить поведение блока.

И это не случайно, так как возникают трудности математического характера при построении совокупной системы уравнений при соединении неориентированных блоков, меняющих свое поведение.

Пакеты Model Vision Studium, AnyLogic наиболее приспособлены для проведения активных компьютерных экспериментов и используют современные объектно-ориентированные входные языки.

Пакеты Model Vision Studium, AnyLogic используют гибридные автоматы как элемент входного языка, однако не могут работать с неориентированными блоками. Хотя пакет Model Vision Studium снабжен редактором трехмерной анимации, компактен и прост в освоении, зато пакет AnyLogic позволяет строить апплеты, и более приспособлен для описания моделей со сложным дискретным поведением, так как позволяет его описывать непосредственно на языке Java.





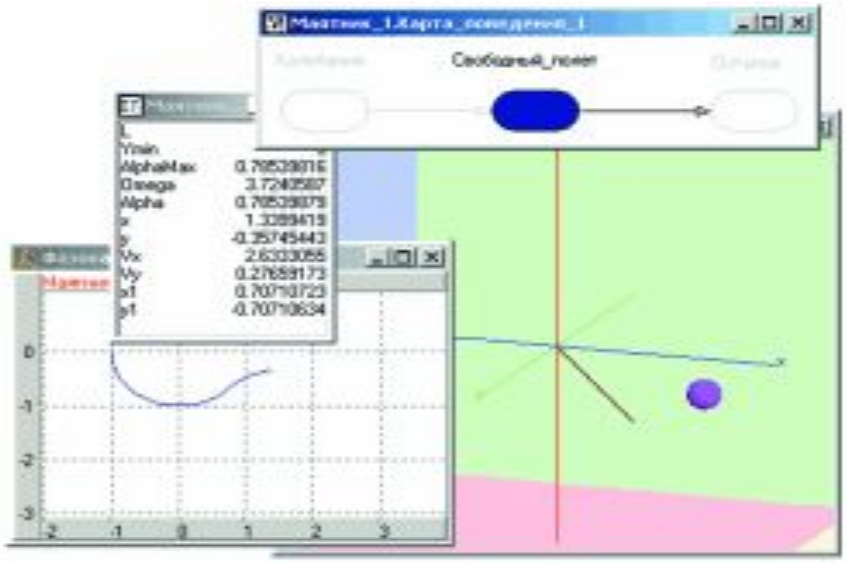
Здесь будет уместно перечислить достоинства гибридного автомата при использовании его пакетах моделирования:

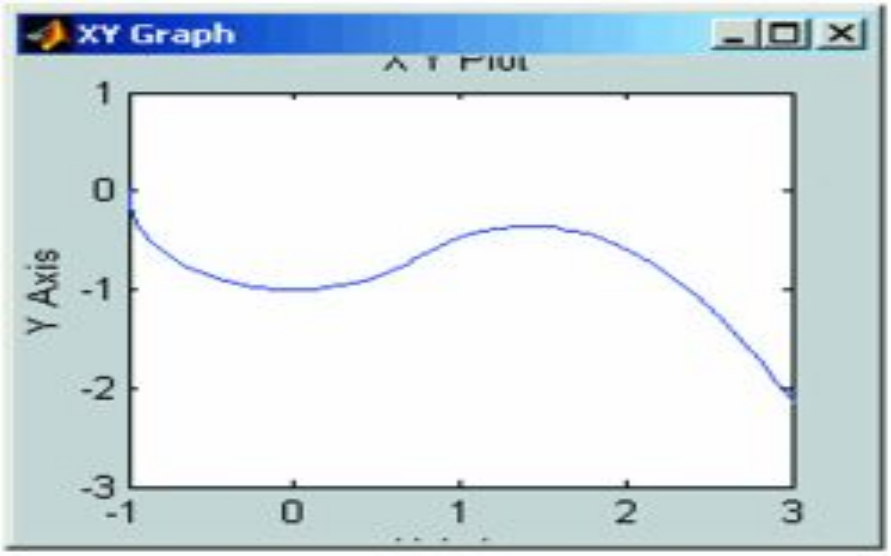
- компактно и наглядно описывает все возможные варианты смены поведения, подобно тому, как конечный автомат столь же наглядно описывает все допустимые цепочки принимаемого им языка;
- в явном виде требует указывать особые события, приводящие к смене поведения, и облегчает работу численных методов в окрестности точки смены поведения;
- позволяет ввести «алгебру» локальных поведений, и тем самым упрощает применение объектно-ориентированного подхода;
- позволяет легко описывать сложные эксперименты с моделью, и тем самым упрощается проектирование испытательных стендов.
- позволяет в наглядной форме следить за появлением событий, приводящих к смене поведения, и самими переключениями при отладке

Не останавливаясь на описании других пакетов, можно утверждать, что все они настолько различны, и в данном случае это недостаток, что нельзя остановиться на каком–либо одном, что чрезвычайно важно для отдельного пользователя и в учебном процессе.

Все перечисленные выше пакеты обладают еще одним, чрезвычайно важным недостатком. В силу того, что они являются коммерческими, они вбирают в себя в момент создания последние научные достижения, профильтрованные через индивидуальные предпочтения разработчиков, и надолго замораживают их.

Действительно, создание конкурентоспособного нового продукта требует огромного стартового финансирования и усилий больших творческих коллективов специалистов различного профиля, и его коренная перестройка экономически невыгодна.





Количество универсальных существующих и разрабатываемых пакетов для моделирования все увеличивается, и это дает право утверждать, что мы имеем дело с востребованной областью вычислительного эксперимента, и позволяет сделать некоторые обобщения:

- в этой новой области мы имеем сейчас ситуацию аналогичную той, которая предшествовала созданию первых специализированных коллекций, т.е. накопленного опыта достаточно для того, чтобы сформулировать единые требования к пакетам для проведения активного вычислительного эксперимента.
- все яснее вырисовывается структура и требования к отдельным компонентам, такие как: Редакторы Уравнений, Гибридного автомата, Структуры, Панелей Управления, Двумерной и Трехмерной Анимации, Движитель Гибридного времени, а также Численная библиотека.
- разработать их все одновременно на высоком уровне трудно даже большому коллективу специалистов, что и находит отражение в сегодняшних пакетах. Это нетрудно увидеть, сравнив, например, численные библиотеки пакетов, как математических, так и универсальных.

- несмотря на существование прекрасно организованных сайтов, где можно найти тщательно документированные и протестированные программы, отбор методов в пакетах производится порой случайным образом.
- при разработке нового пакета для создания требуемой библиотеки достаточно проделать высококвалифицированную, но большей частью организационную работу. Взяв единую библиотеку за основу, можно ее дополнять при создании конкретного пакета. Наличие такой общей библиотеки только усилит доверие пользователя к новому программному средству.
- аналогичная ситуация наблюдается и с Редакторами Уравнений, или Структуры — это относительно независимые компоненты, и они могут работать во многих пакетах одновременно.
- более того, если зафиксировать интерфейс таких блоков и способы обмена данными, то можно просто заменять их целиком. Если бы удалось согласовать интерфейс этих блоков, и в качестве первоначальных вариантов взять уже готовые компоненты, к работе над созданием новых универсальных пакетов могли бы подключится многие специалисты. И как минимальный результат, через несколько лет можно было бы создать прототип работающей системы.

Объектно-ориентированное моделирование

Объектно-ориентированный подход в последнее время стал так прочно ассоциироваться с программированием (даже аббревиатуру ООП обычно расшифровывают как Объектно–Ориентированное *Программирование*), что многие забывают о его прямой связи с моделированием — ведь первоначально он был использован в языке моделирования SIMULA–67.

Однако в дальнейшем объектно-ориентированный подход развивался почти исключительно программистами.

Своеобразным итогом тридцатилетнего развития объектно-ориентированного программирования можно считать появление «унифицированного языка моделирования» UML, предназначенного для создания объектно-ориентированных спецификаций программных систем на ранних этапах разработки.

И только в последние годы объектно-ориентированный подход стал востребованным в своей «родной области», моделировании.

Объектно-ориентированное моделирование (ООМ) сегодня весьма модный термин, но на практике этот подход поддерживают не так уж много пакетов.

Объектом принято называть некоторую сущность, которая инкапсулирует в себе данные и методы как единое целое и взаимодействует с внешним окружением через определенный интерфейс.

С понятием объекта тесно связано отношение двойственности — «класс–экземпляр». Каждый объект всегда является экземпляром какого–то класса.

Естественным кандидатом на роль объекта в ООМ является, конечно же, компонента.

Компонента является совокупностью переменных и поведения, она взаимодействует с внешним миром только через внешние переменные.

Компонента всегда — явно или неявно — является экземпляром некоторого класса.

Например, когда вы в пакете SIMULINK, который формально не поддерживает ООМ, размещаете на функциональной схеме новый блок, вы неявно порождаете новый экземпляр выбранного вами предопределенного класса из библиотеки.

В современном ООМ компонента несет в себе информацию о поведении, структуре и динамическом образе моделируемого объекта. Это и есть три основных элемента определения класса.

Модель может содержать много экземпляров одного и того же класса.

Без понятия класса практически невозможно моделировать многокомпонентные системы, имеющие регулярную или переменную структуру.

В ООП объекты делят на:

- Пассивные;
- Активные.

Пассивные объекты (большая часть объектов в программах) только «откликаются» на вызовы методов и сообщений извне, но сами ничего не делают, т.е. не могут изменять значения своих данных по собственной инициативе.

Активные объекты (например, экземпляры класса Thread в языке Delphi) имеют свою собственную «нить управления» и функционируют независимо от других объектов и параллельно с ними.

Объекты ООМ, конечно же, активные, причем их активность связана не с циклической последовательностью дискретных действий, а с непрерывным воспроизведением поведения и непрерывной реакцией на внешние воздействия.

Более сложными механизмами ООМ являются наследование и полиморфизм.

Наследование позволяет строить новые классы, модифицируя старые путем введения новых элементов и переопределения уже существующих.

Это очень похоже на объектное программирование, но элементами компоненты модели теперь являются переменные, системы уравнений и карты состояний.

Например, чтобы создать модель прыгающего мячика в воздухе, можно взять модель прыгающего мячика в вакууме и переопределить поведение в состоянии «Полет», изменив уравнения с учетом сопротивление воздуха.

Полиморфизм позволяет использовать вместо компоненты одного класса компоненту другого похожего класса.

Обычный полиморфизм по наследованию предполагает, что вместо компоненты базового класс всегда может использоваться компонента любого производного класса (например, для локационной станции и аэробус, и истребитель выступают как летательный аппарат).

Иногда оказывается полезным полиморфизм по интерфейсу, когда один компонент может быть заменен другим, имеющим точно такой же набор внешних переменных (например, сопротивление, конденсатор и индуктивность — это все двухполюсники и с точки зрения собирания электрической схемы они одинаковы).

Структурное программирование

Структурное программирование — это метод программирования, опирающийся на структурную организацию программы.

Основной принцип структурного программирования - обеспечить максимальное соответствие структуры текста программы логике решаемой проблемы.

Способы реализации основного принципа структурного программирования:

1) **Структурируемости и читаемости текста программы –** отступы, обозначения, группировка частей текста.

2) **Отлаживаемость программы** – пригодность конструкций языка к отладке. В значительной степени связана с особенностями типизации данных. Наличие жесткого контроля типов с диагностикой ошибок при трансляции программы. Другая важная часть – наличие интегрированной среды разработки.

3) **Инкапсуляция данных** — наличие структурных типов данных (массивы, записи, строки, множества, файловые типы, потоки и т.п.) с возможностью оперирования переменной структурного типа как единым целым.

4) **Инкапсуляция программного кода -** блоки разных уровней, модули, пакеты и т.п. с особыми правилами прозрачности и интерфейсами между ними.

5) **Инкапсуляция программного кода и данных во время выполнения программы**, со специальными ограничениями доступа, правилами прозрачности, интерфейсами, диагностикой ошибок во время выполнения программы. Примеры: обработка исключительных ситуаций во время выполнения, обработки событий, подпроцессы.

Основные задачи структурного программирования

Структурное программирование предназначено для ***решения трех основных задач***:

- **Повышение эффективности разработки программ**: а) увеличение скорости написания программ; б) увеличение качества их сопровождения (внесения усовершенствований, исправления ошибок); в) уменьшение стоимости разработки; г) обеспечение возможности групповой работы с проектом; д) обеспечение возможности проектирования ПО (software engineering).
- **Повышение надежности работы программ**. Никому не нужна дешевая, мгновенно написанная и занимающая мало ресурсов программа, которая работает очень быстро, но с ошибками. Из программистского фольклора: “Беремся написать вам ПО : 1. Быcро 2.дешево 3.надежно. – Два из трех.
- **Повышение эффективности работы программ**: увеличение скорости их выполнения, расширение функциональных и сервисных возможностей программ, а также уменьшение их ресурсоемкости (размеров программного кода, объема используемой оперативной и дисковой памяти).

Одним из важнейших критериев решения перечисленных задач является ***коэффициент повторного использования программного кода***, т.е. многократного использования одного и того же куска программы. Преимущества процедур, функций и библиотек подпрограмм, и в особенности объектного программирования, в значительной степени связаны с увеличением коэффициента повторного использования кода.

Основные принципы объектно-ориентированного программирования (ООП)

ООП — это метод программирования, развивающий принципы структурного программирования и основанный на следующих абстракциях данных:

- ***Инкапсуляция***: объединение данных с процедурами и функциями в единый блок программного кода (данные и методы работы с ними рассматриваются как поля объекта).
- ***Наследование*** (наличие экземпляров класса; потомки, прародители, иерархия).
- ***Полиморфизм*** (единое имя для некого действия, которое по-разному осуществляется для объектов иерархии).

Компонентное программирование:

Реализация объектов в виде независимо распространяемых исполняемых модулей. Известны два типа компонентов:

- Компоненты конкретного языка программирования (Visual BASIC, Object PASCAL, Java). Предназначены преимущественно для поддержки средств визуального проектирования
- Компоненты, поддерживаемые на уровне ОС (компоненты Activex, .NET).

Спасибо