**1.3.4. Методы разработки структуры программы.**

Как уже отмечалось выше, в качестве модульной структуры программы принято использовать древовидную структуру, включая деревья со сросшимися ветвями. В узлах такого дерева размещаются программные модули, а направленные дуги (стрелки) показывают статическую подчиненность модулей, т.е. каждая дуга показывает, что в тексте модуля, из которого она исходит, имеется ссылка на модуль, в который она входит.

Другими словами, каждый модуль может обращаться к подчиненным ему модулям, т.е. выражается через эти модули. При этом модульная структура программы, в конечном счете, должна включать и совокупность спецификаций модулей, образующих эту программу.

*Спецификация*программного модуля содержит, во-первых, синтаксическую спецификацию его входов, позволяющую построить на используемом языке программирования синтаксически правильное обращение к нему (к любому его входу), и, во-вторых, функциональную спецификацию модуля (описание семантики функций, выполняемых этим модулем по каждому из его входов). Функциональная спецификация модуля строится так же, как и функциональная спецификация ПС.

В процессе разработки программы ее модульная структура может по-разному формироваться и использоваться для определения порядка программирования и отладки модулей, указанных в этой структуре. Поэтому можно говорить о разных методах разработки структуры программы. Обычно в литературе обсуждаются два метода [7.1, 7.7]: метод восходящей разработки и метод нисходящей разработки.

**Метод *восходящей разработки***

Метод *восходящей разработки*заключается в следующем. Сначала строится модульная структура программы в виде дерева. Затемпоочередно программируются модули программы, начиная с модулей самого нижнего уровня (листья дерева модульной структуры программы), в таком порядке, чтобы для каждого программируемого модуля были уже запрограммированы все модули, к которым он может обращаться.

После того, как все модули программы запрограммированы, производится их поочередное тестирование и отладка в принципе в таком же (восходящем) порядке, в каком велось их программирование. На первый взгляд такой порядок разработки программы кажется вполне естественным: каждый модуль при программировании выражается через уже запрограммированные непосредственно подчиненные модули, а при тестировании использует уже отлаженные модули.

Однако, современная технология не рекомендует такой порядок разработки программы. Во-первых, для программирования какого-либо модуля совсем не требуется текстов используемых им модулей - для этого достаточно, чтобы каждый используемый модуль был лишь специфицирован (в объеме, позволяющем построить правильное обращение к нему), а для тестирования его возможно (и даже, как мы покажем ниже, полезно) используемые модули заменять их имитаторами (заглушками). Во-вторых, каждая программа в какой-то степени подчиняется некоторым внутренним для нее, но глобальным для ее модулей соображениям (принципам реализации, предположениям, структурам данных и т.п.), что определяет ее концептуальную целостность и формируется в процессе ее разработки.

При восходящей разработке эта глобальная информация для модулей нижних уровней еще не ясна в полном объеме, поэтому очень часто приходится их перепрограммировать, когда при программировании других модулей производится существенное уточнение этой глобальной информации (например, изменяется глобальная структура данных). В-третьих, при восходящем тестировании для каждого модуля (кроме головного) приходится создавать ведущую программу (модуль), которая должна подготовить для тестируемого модуля необходимое состояние информационной среды и произвести требуемое обращение к нему. Это приводит к большому объему "отладочного" программирования и в то же время не дает никакой гарантии, что тестирование модулей производилось именно в тех условиях, в которых они будут выполняться в рабочей программе.

**Метод *нисходящей разработки***

Метод *нисходящей разработки* заключается в следующем. Как и в предыдущем методе сначала строится модульная структура программы в виде дерева. Затем поочередно программируются модули программы, начиная с модуля самого верхнего уровня (головного), переходя к программированию какого-либо другого модуля только в том случае, если уже запрограммирован модуль, который к нему обращается.

После того, как все модули программы запрограммированы, производится их поочередное тестирование и отладка в таком же (нисходящем) порядке. При таком порядке разработки программы вся необходимая глобальная информация формируется своевременно, т.е. ликвидируется весьма неприятный источник просчетов при программировании модулей. Существенно облегчается и тестирование модулей, производимое при нисходящем тестировании программы. Первым тестируется головной модуль программы, который представляет всю тестируемую программу и поэтому тестируется при "естественном" состоянии информационной среды, при котором начинает выполняться эта программа.

При этом все модули, к которым может обращаться головной, заменяются на их имитаторы (так называемые заглушки [7.5]). Каждый *имитатор модуля*представляется весьма простым программным фрагментом, сигнализирующим, в основном, о самом факте обращения к имитируемому модулю с необходимой для правильной работы программы обработкой значений его входных параметров (иногда с их распечаткой) и с выдачей, если это необходимо, заранее запасенного подходящего результата.

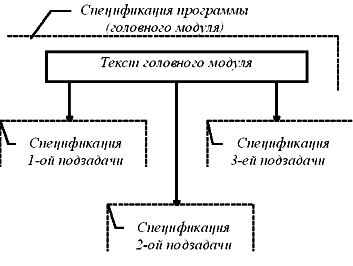
После завершения тестирования и отладки головного и любого последующего модуля производится переход к тестированию одного из модулей, которые в данный момент представлены имитаторами, если таковые имеются. Для этого имитатор выбранного для тестирования модуля заменяется на сам этот модуль и добавляются имитаторы тех модулей, к которым может обращаться выбранный для тестирования модуль. При этом каждый такой модуль будет тестироваться при "естественных" состояниях информационной среды, возникающих к моменту обращения к этому модулю при выполнении тестируемой программы.

Таким образом большой объем "отладочного" программирования заменяется программированием достаточно простых имитаторов используемых в программе модулей. Кроме того, имитаторы удобно использовать для подыгрывания процессу подбора тестов путем задания нужных результатов, выдаваемых имитаторами.

Некоторым недостатком нисходящей разработки, приводящим к определенным затруднениям при ее применении, является необходимость абстрагироваться от базовых возможностей используемого языка программирования, выдумывая абстрактные операции, которые позже нужно будет реализовать с помощью выделенных в программе модулей. Однако способность к таким абстракциям представляется необходимым условием разработки больших программных средств, поэтому ее нужно развивать.

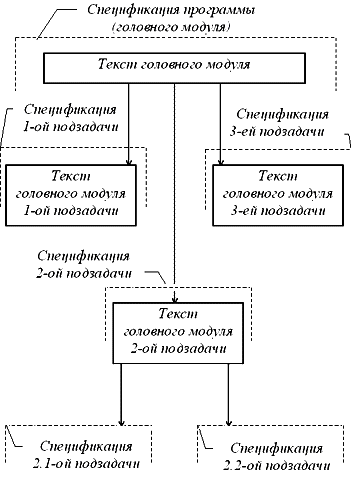
В рассмотренных методах восходящей и нисходящей разработок (которые мы будем называть *классическими*) модульная древовидная структуру программы должна разрабатываться до начала программирования модулей. Однако такой подход вызывает ряд возражений: представляется сомнительным, чтобы до программирования модулей можно было разработать структуру программы достаточно точно и содержательно. На самом деле это делать не обязательно: так при конструктивном и архитектурном подходах к разработке программ модульная структура формируется в процессе программирования модулей.

*Конструктивный подход*к разработке программы представляет собой модификацию нисходящей разработки, при которой модульная древовидная структура программы формируется в процессе программирования модуля. Сначала программируется головной модуль, исходя из спецификации программы в целом, причем спецификация программы является одновременно и спецификацией ее головного модуля, так как последний полностью берет на себя ответственность за выполнение функций программы. В процессе программирования головного модуля, в случае, если эта программа достаточно большая, выделяются подзадачи (внутренние функции), в терминах которых программируется головной модуль. Это означает, что для каждой выделяемой подзадачи (функции) создается спецификация реализующего ее фрагмента программы, который в дальнейшем может быть представлен некоторым поддеревом модулей. Важно заметить, что здесь также ответственность за выполнение выделенной функции берет головной (может быть, и единственный) модуль этого поддерева, так что спецификация выделенной функции является одновременно и спецификацией головного модуля этого поддерева. В головном модуле программы для обращения к выделенной функции строится обращение к головному модулю указанного поддерева в соответствии с созданной его спецификацией. Таким образом, на первом шаге разработки программы (при программировании ее головного модуля) формируется верхняя начальная часть дерева, например, такая, которая показана на рис. 1.3.1.

  
Рис. 1.3.1. Первый шаг формирования модульной структуры программы при конструктивном подходе.

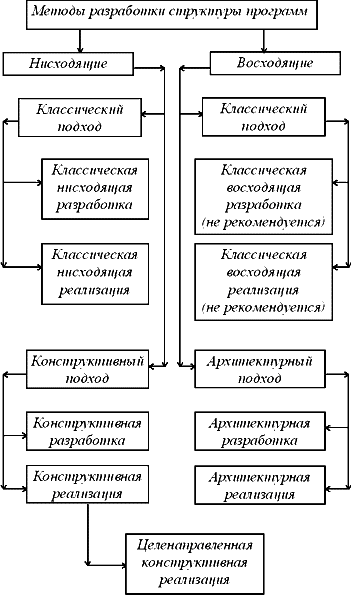
Аналогичные действия производятся при программировании любого другого модуля, который выбирается из текущего состояния дерева программы из числа специфицированных, но пока еще не запрограммированных модулей. В результате этого производится очередное доформирование дерева программы, например, такое, которое показано на рис. 1.3.2.

*Архитектурный подход* к разработке программы представляет собой модификацию восходящей разработки, при которой модульная структура программы формируется в процессе программирования модуля. Но при этом ставится существенно другая цель разработки: повышение уровня используемого языка программирования, а не разработка конкретной программы. Это означает, что для заданной предметной области выделяются типичные функции, каждая из которых может использоваться при решении разных задач в этой области, и специфицируются, а затем и программируются отдельные программные модули, выполняющие эти функции. Так как процесс выделения таких функций связан с накоплением и обобщением опыта решения задач в заданной предметной области, то обычно сначала выделяются и реализуются отдельными модулями более простые функции, а затем постепенно появляются модули, использующие ранее выделенные функции. Такой набор модулей создается в расчете на то, что при разработке той или иной программы заданной предметной области в рамках конструктивного подхода могут оказаться приемлемыми некоторые из этих модулей. Это позволяет существенно сократить трудозатраты на разработку конкретной программы путем подключения к ней заранее заготовленных и проверенных на практике модульных структур нижнего уровня. Так как такие структуры могут многократно использоваться в разных конкретных программах, то архитектурный подход может рассматриваться как путь борьбы с дублированием в программировании. В связи с этим программные модули, создаваемые в рамках архитектурного подхода, обычно параметризуются для того, чтобы усилить применимость таких модулей путем настройки их на параметры.

  
Рис. 1.3.2. Второй шаг формирования модульной структуры программы при конструктивном подходе.

В классическом методе нисходящей разработки рекомендуется сначала все модули разрабатываемой программы запрограммировать, а уж затем начинать нисходящее их тестирование. Однако такой порядок разработки не представляется достаточно обоснованным: тестирование и отладка модулей может привести к изменению спецификации подчиненных модулей и даже к изменению самой модульной структуры программы, так что в этом случае программирование некоторых модулей может оказаться бесполезно проделанной работой. Нам представляется более рациональным другой порядок разработки программы, известный в литературе как метод нисходящей реализации. В этом методе каждый запрограммированный модуль начинают сразу же тестировать до перехода к программированию другого модуля.

Все эти методы имеют еще различные разновидности в зависимости от того, в какой последовательности обходятся узлы (модули) древовидной структуры программы в процессе ее разработки. Это можно делать, например, по слоям (разрабатывая все модули одного уровня, прежде чем переходить к следующему уровню). При нисходящей разработке дерево можно обходить также в лексикографическом порядке (сверху-вниз, слева-направо). Возможны и другие варианты обхода дерева. Так, при конструктивной реализации для обхода дерева программы целесообразно следовать идеям Фуксмана, которые он использовал в предложенном им методе вертикального слоения. Сущность такого обхода заключается в следующем. В рамках конструктивного подхода сначала реализуются только те модули, которые необходимы для самого простейшего варианта программы, которая может нормально выполняться только для весьма ограниченного множества наборов входных данных, но для таких данных эта задача будет решаться до конца. Вместо других модулей, на которые в такой программе имеются ссылки, в эту программу вставляются лишь их имитаторы, обеспечивающие, в основном, контроль за выходом за пределы этого частного случая. Затем к этой программе добавляются реализации некоторых других модулей (в частности, вместо некоторых из имеющихся имитаторов), обеспечивающих нормальное выполнение для некоторых других наборов входных данных. И этот процесс продолжается поэтапно до полной реализации требуемой программы. Таким образом, обход дерева программы производится с целью кратчайшим путем реализовать тот или иной вариант (сначала самый простейший) нормально действующей программы. В связи с этим такая разновидность конструктивной реализации получила название метода *целенаправленной конструктивной реализации*. Достоинством этого метода является то, что уже на достаточно ранней стадии создается работающий вариант разрабатываемой программы. Психологически это играет роль допинга, резко повышающего эффективность разработчика. Поэтому этот метод является весьма привлекательным.

  
Рис. 1.3.3. Классификация методов разработки структуры программ.

Подводя итог сказанному, на рис. 1.3.3. представлена общая схема классификации рассмотренных методов разработки структуры программы.