## Тема 4.1. Определение визуального моделирования

4.1.1. Чертеж и программное обеспечение

4.1.2. Понятие визуального моделирования

4.1.3. Контрольные вопросы по теме «Тема 4.1. Определение визуального моделирования»

4.1.4. Контрольные тесты по теме «Тема 4.1. Определение визуального моделирования»

**4.1.1. Чертежи и программное обеспечение**

В настоящее время производство ПО достигло огромного размаха. В программную индустрию вовлечены большие денежные средств. Рост производства в этой области продолжается высокими темпами.

И одновременно со всем этим разработка ПО остается очень рисковой деятельностью. Низка предсказуемость ресурсов и времени разработки проектов, существует много проблем с соответствием созданного ПО требованиям контекста, где оно будет работать, а также ожиданиям заказчика. Высок процент неудачных проектов по сравнению с другими промышленными областями. Когда разработка ПО входит в более крупный промышленный проект (например, создается новая модель автомобиля, разрабатывается космический корабль), то оказывается, что программная часть разработки является наиболее дорогостоящей и плохо предсказуемой.

В конце 60-х годов прошлого века исследователи в поисках способов упорядочивания и стандартизации процесса создания ПО обратились к другим, уже устоявшимся промышленным областям. Было замечено, что в строительстве, машиностроении, электротехнике и т. д. работы по созданию новых систем разбиваются на два основных этапа – проектирование и реализацию. Проектирование осуществляют архитекторы, конструкторы, инженеры, а изготовляют систему рабочие, строители, монтеры. Результаты проектирования фиксируются с помощью чертежей – схематичных изображений создаваемой системы. Эти чертежи служат хорошим интерфейсом между проектировщиками и теми, кто, собственно, создает, делает саму систему. Чертежи обязывают последних строго следовать принятым решениям, избавляя от необходимости думать над принципиальными вопросами. Главное уже продумано и решено, и все, что нужно – это разобраться в чертежах системы, понять, как и что нужно сделать, и сделать это. Ситуации, когда по ходу разработки приходится менять проектные решения, как правило, случаются крайне редко. Таким образом, чертежи сыграли важную роль в становлении современной промышленности, позволяя эффективно разделить труд между квалифицированными инженерами и обычными рабочими.

Аналогичным образом хотелось бы применять чертежи и в программировании. Однако здесь существуют некоторые особенности, которые не позволили использовать чертежное проектирование «as is».

Инженерные дисциплины, успешно использующие чертежи, занимаются разработкой материальных, видимых объектов. Чем же ПО отличается от этих объектов.

Фундаментальным отличием программного обеспечения от других инженерных объектов является то, что оно в значительно большей степени является объектом психического мира, и в существенно меньшей степени – объектом физического мира.

Фредерик Брукс [[1.1]](http://www.intuit.ru/department/se/vismodtp/popup.lit.html#1) выделил следующие характеристические признаки ПО, отличающие его от других инженерных объектов: невидимость, изменчивость, согласуемость (в основном с людьми – заказчиками и разработчиками, а также между различными категориями задействованных в его создании лиц), а также огромную сложность (другими словами - трудности концептуализации программных систем). Можно сказать, что ПО – это некоторый текст (программа на языке программирования), снабженный большим количеством ментальных интерпретаций – от идеи целевого сервиса, который реализует данное ПО, до концепций его внутреннего устройства. Ну и, конечно, данный текст обладает вычислительной интерпретацией – программа должна исполняться вычислителем.

Очевидно, что объем психических интерпретаций существенно превышает объем физического восприятия ПО. Последнее даже как-то не очень понятно – что же воспринимать здесь органами чувств? Можно видеть тексты программ, можно видеть последовательность сменяющих друг друга окон на мониторе (то есть интерфейс работающего ПО), можно физически воспринимать аппаратуру, которой ПО управляет (например, слышать звонок телефонного аппарата или входить в дверь, открываемую электронным замком). Но все это – косвенные свидетельства.

А вот, например, построенный дом, созданный автомобиль или подводная лодка – это полноценные объекты физического мира. Поэтому эти объекты можно увидеть. А значит, нарисовать, начертить.

Чертежи хорошо «работают» в промышленности, так как позволяют схематично нарисовать то, что можно будет потом увидеть глазами. Ведь сечение здания или механической детали, принципиальная схема электроснабжения квартиры или завода - все это можно увидеть или правдоподобно представить, убрав лишнее, изменив масштаб, упростив изображение несущественных деталей. Имея такой чертеж системы, можно пояснить на нем основные решения, создав тем самым хорошее предписание для тех, кто будет эту систему создавать. Участникам проекта легче понять друг друга, поскольку они вместе видят одни и те же чертежи, которые вызывают в их памяти одни и те же визуальные образы других подобных объектов. А если еще добавить к этому, что до 90% информации современный человек получает именно через зрение, то понятно, почему чертежи так облегчают жизнь при создании искусственных систем. И понятно, почему хочется их использовать в программировании.

Итак, программное обеспечение, находится более в психическом, чем в физическом мире человека и оказывается невидимым. Поэтому его чертежи не привносят в проект той магической ясности, как чертежи (пусть даже очень сложные) строящегося здания, конструируемого самолета, монтируемой электроустановки. Не имея очевидных, зримых образов ПО, мы не можем однозначно сказать, как его изображать. Каждый склонен «видеть» и, соответственно, изображать ПО как-то по-своему или вовсе обходиться без этого.

Не все понятно и с тем, какую часть ПО имеет смысл изображать. Скорее всего, его архитектуру… Но в литературе по программной инженерии на настоящий момент существует более ста различных определений этого. То есть, однозначности нет и в этом вопросе.

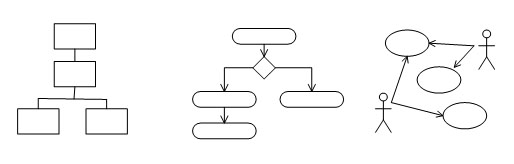
Кроме того, в программировании не удается построить столь же четкое разделение труда, как в других промышленных областях, выделив умных архитекторов и трудолюбивых и послушных разработчиков. Автор «архитектурного решения», как правило, участвует в его реализации, потому что зачастую только он один до конца понимает все хитросплетения своего замысла и варианты его дальнейшего развития. Архитектор должен постоянно участвовать в проекте (возможно, с разной степенью интенсивности), так как разработка ПО итеративна и проектирование не может быть окончательно завершено перед разработкой, а его результаты – зафиксированы чертежами. Архитектор – это лишь опытный разработчик. А инженер от рабочего отличается радикально…

Однако все эти обстоятельства не являются непреодолимым барьером в использовании чертежей при создании ПО. Ситуация не безнадежная, а всего лишь иная.

В силу невидимости ПО центральным аспектом при его визуализации является поиск подходящей метафоры. У нас нет геометрических форм объекта, которые в классическом черчении являются основой всех его схематичных изображений. Поэтому нам нужно чему-то уподобить зрительный образ ПО. Программная система выглядит как … что?

Так возникают метафоры визуализации ПО – способы сопоставлять абстрактные и невидимые человеческому глазу элементы ПО некоторым зрительно воспринимаемым объектам. Важно лишь всем договориться, что ПО мы видим и изображаем так-то и так-то, тем самым задействовав зрительный канал при проектировании и разработке ПО, при передаче знаний о создаваемых и уже созданных и работающих программных системах. В этом смысле неоценимую пользу оказывает развитие стандартных визуальных языков разработки ПО. В настоящее время это, в первую очередь, UML. Люди привыкают к изображениям классов, пакетов, объектов, процессов и пр., к определенному набору диаграмм. Они привыкают мыслить, строя те или иные диаграммы UML. А другие легко читают эти мысли в этих диаграммах. То есть с помощью стандартных визуальных языков мы все вместе договариваемся, как видеть невидимое. Может быть, мы скоро начнем действительно видеть ПО…

Среди различных метафор визуализации ПО выделяются графы, где вершины, изображаемые по-разному, и ребра – стрелки, связи, зависимости и т. д. На [рис. 4.1](http://www.intuit.ru/department/se/vismodtp/1/2.html#image.1.2) приведено несколько типов диаграмм, используемых на практике при проектировании ПО.



**Рис. 4.1-1**  Примеры разных графов, используемых в визуальном моделировании

Очевидно, что на этом рисунке изображены разные графы. На настоящий момент, несмотря на многочисленные попытки, другой общеупотребительной метафоры визуализации ПО не создано.

Самыми распространенными моделями, основанными на графах, являются модель «сущность-связь» и модель «конечных автоматов», объединенная с блок-схемами. В UML и диаграммы классов, и диаграммы компонент, объектов, коммуникаций, развертывания и пр. являются лишь вариациями модели "сущность-связь", а диаграммы конечных автоматов и активностей – вариациями конечных автоматов и блок-схем.

**4.1.2. Понятие визуального моделирования**

Итак, визуальное моделирование (visual modeling) является методом, применяемым в разработке ПО, который:

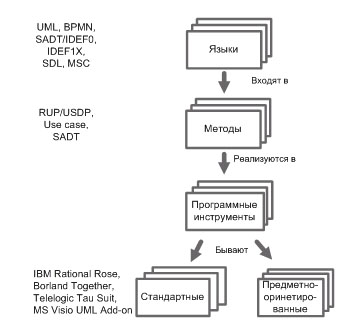
* использует графовые модели для визуализации ПО;
* предлагает моделировать ПО с разных точек зрения;
* может применяться в разработки и эволюции ПО, а также в различных видах деятельности по его созданию.

Использование в рамках визуального моделирования отдельных не графовых моделей нарушает степень общности данного выше определения. Однако на практике графы явно доминируют.

Принципиально, что в одном проекте используются разные визуальные модели ПО, созданные с разных точек зрения. Визуальные модели, как правило, не составляют «сплошных» спецификаций, подобно программам, но часто являются, скорее, фрагментами, формально не связанными друг с другом. Эти модели описывают отдельные аспекты ПО, которые нужно прояснить в определенной ситуации для той или иной категории лиц, участвующих в проекте или как-либо с ним связанных. В целом визуальное моделирование служит для повышения понимаемости решений проекта людьми – разными категориями задействованных в проекте специалистов.

Визуальное моделирование может применяться как при разработке, так и при сопровождении ПО. При разработке – главным образом при проектировании и анализе системы, которые предшествуют непосредственному программированию. При сопровождении – когда новые разработчики изучают доставшееся им ПО. Визуальное моделирование может также использоваться в разных видах деятельности процесса разработки ПО: главным образом при анализе и проектировании, но также и при документировании, тестировании, разработке требований и т. д.

Визуальное моделирование применяется на практике с помощью методов, языков и соответствующих программных инструментов (см. [рис. 1.3](http://www.intuit.ru/department/se/vismodtp/1/2.html#image.1.3)).



**Рис. 4.1-2**  Визуальное моделирование: языки, методы, программные средства

Языки визуального моделирования (или визуальные языки) - это формализованные наборы графических символов и правила построения из них визуальных моделей. Сейчас известны и активно используются на практике такие языки визуального моделирования, как UML и BPMN. Однако существуют и более старые языки: SDL и MSC для моделирования телекоммуникационных систем, SADT/IDEF0 для моделирования бизнес-процессов, IDEF1x для моделирования баз данных и некоторые другие. Кроме того, в исследовательской среде создано множество других визуальных языков

Методы использования визуального моделирования предписывают правила применения визуальных языков для решения тех или иных задач процесса разработки ПО.

Средства, реализующие языки и методы визуального моделирования, бывают двух видов – универсальные и предметно-ориентированные.

Универсальные инструменты являются многофункциональными пакетами, предназначенными для анализа и проектирования ПО «вообще», то есть без какой-либо специализированной ориентации. Как правило, сегодня такие пакеты строятся на базе языка UML и называются CASE-пакетами. Самыми известными CASE-пакетами являются IBM Rational Rose, Borland Together, Telelogic Tau, Microsoft Visio/UML Add-on, VS . Эти средства поддерживают различные виды диаграмм, удобную среду их разработки с такими функциями, как печать и копирование диаграмм, различные способы редактирования графических символов, средства просмотра и поиска в визуальной модели, различные режимы отображения диаграмм и многое другое. Они также обеспечивают генерацию программного кода в разные целевые платформы программирования, версионный контроль визуальных моделей, часто являются кросс-платформенными. .

Термин CASE (Computer Aided Software Engineering) появился в индустрии разработки ПО в начале 1980-х годов. Довольно быстро он стал обозначать графические средства анализа и проектирования ПО, отражая надежды и упования создать на основе визуального моделирования универсальный процесс разработки ПО. Пик развития этих средств приходится на начало 1990-х годов.

С середины 1990-х годов стали появляться CASE-системы для персональных компьютеров. Их эволюция происходила и продолжает происходить уже по иному сценарию. Современные CASE-пакеты не являются комплексными средами разработки, а заняли нишу средств анализа и проектирования, и в основном используются без средств кодогенерации, а лишь как инструменты для построения проектных спецификаций.

Предметно-ориентированные программные инструменты поддержки визуального моделирования предназначены для определенных областей разработки ПО и тоже могут быть коробочными, как, например, пакет WebRatio для моделирования web-приложений. Однако предметно-ориентированные инструменты могут создаваться и отдельными компаниями для своих собственных проектов, особенно в рамках линеек программных продуктов (product lines). Это особенно удобно, поскольку во-первых, такие средства могут хорошо решать задачи именно того процесса, той компании, для которых они создаются. А во-вторых, сейчас на рынке имеются развитые среды для разработки средств визуального моделирования, самые известные из которых - Microsoft Visio, Microsoft DSL Tools и Eclipse/GMF. Эти и другие пакеты делают задачу создания собственного графического редактора посильной для обычных, рядовых компаний-разработчиков.

Идея автоматической генерации программного кода по визуальным моделям понятна и притягательна. Диаграммы являются более близкими к предметной области, чем программный код, понятны инженерам, менеджерам, заказчикам и т.д. Долгое время считалось, что визуальное моделирование является следующим шагом эволюции средств программирования, вслед за алгоритмическими языками высокого уровня ([рис. 4.1-](http://www.intuit.ru/department/se/vismodtp/1/3.html#image.1.4)3).



**Рис. 4.1-3.**  Эволюция средств программирования

Таким образом, программы должны быть максимально доступны для человека – как для самого автора, чтобы упростить процесс его работы, так и для других людей, чтобы сделать работу одного более понятной другим. Эта двойственная природа программ отражена на   
[рис. 4.1-4](http://www.intuit.ru/department/se/vismodtp/1/3.html#image.1.5).

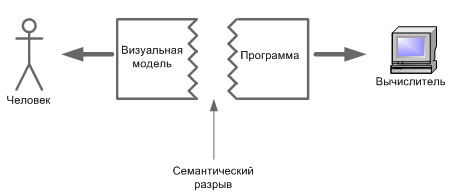


**Рис. 4.1-4.**  Отношения программы с вычислителем и программистом

Ожидалось, что визуальные модели станут следующим этапом, заменив собой алгоритмические языки и предоставив более широкие возможности программистам.

Однако оказалось, что визуальные модели, действительно удобные в работе, «склонны» терять исполняемую семантику. Другими словами, информация, которая в них содержится, оказывается недостаточно полной и детальной, чтобы по ней вычислитель мог бы выполнить свою работу. Ведь никакая «умная» генерация не может добавить то, что отсутствует изначально. Если же визуальные модели усложнять, чтобы они были пригодны для использования вычислителем, то очень часто они теряют наглядность и становятся бесполезными. Кому нужны непонятные, но полные описания программного обеспечения, выполненные с помощью визуальных моделей? Есть тексты на языках программирования, есть документы, есть возможность спросить, в конце концов, разобраться в коде самому…

Таким образом, существует семантический разрыв между визуальными моделями и программами, как показано на [рис. 4.1-5](http://www.intuit.ru/department/se/vismodtp/1/3.html#image.1.6). Этот разрыв препятствует автоматической генерации программного кода по визуальным моделям в общем случае, не позволяя визуальному моделированию стать следующим шагом в развитии средств программирования, вслед за алгоритмическими языками высокого уровня.



**Рис. 4.1-5.**  Семантический разрыв между моделями и программами

Неудача решения вопроса с генерацией «в общем виде» не говорит о том, что это невозможно в частных случаях. Необходимо лишь понизить степень общности ситуации. Это можно сделать, создавая кодогерационные решения для ПО отдельных видов. Генерация кода по визуальным моделям успешно применяется в промышленности в следующих областях:

* в разработке схем реляционных баз данных;
* при создании событийно-ориентированных систем реального времени;
* при формализации бизнес-процессов компаний.

Эти области и будут подробно рассмотрены в этом курсе.

Можно также разрабатывать специальные языки визуального моделирования и программные средства их поддержки для больших проектов или групп проектов, создавая для них эффективные решения.

**4.1.3. Контрольные тесты по теме  
«Тема 4.1. Определение визуального моделирования»**

**4.1.4. Контрольные вопросы по теме   
«Тема 4.1. Определение визуального моделирования»**

1. Расскажите о роли чертежей в промышленных дисциплинах (машиностроении, электротехнике, строительстве и пр.).
2. Что мешает сходным образом использовать чертежи при создании ПО?
3. Что означает выражение "ПО невидимо"?
4. Что такое метафора визуализации?
5. Расскажите о пользе стандартных языков визуального моделирования.
6. Почему графовая метафора является самой распространенной в области визуального моделирования ПО?
7. Что такое визуальное моделирование? Разберите и объясните отдельные части определения.
8. Что такое средства визуального моделирования?
9. Что такое язык визуального моделирования? Приведите примеры таких языков.
10. Что такое метод визуального моделирования? Приведите примеры.
11. Что такое CASE-пакеты? Приведите примеры современных CASE-пакетов.
12. Чем современные CASE-пакеты отличаются от прежних?
13. Каковы выгоды предметно-ориентированного визуального моделирования?
14. Чем стандартные программные средства поддержки визуального моделирования отличаются от предметно-ориентированных?
15. Какие существуют пакеты для разработки предметно-ориентированных средств поддержки визуального моделирования?
16. В чем суть семантического разрыва между визуальными моделями и программным кодом?
17. Как преодолеть этот разрыв?