Программная инженерия

Прототипирование программных систем

Лекция 10

Лекция 8 Управление конфигурациями

Цели:

Читать по Лекции 10	
Книга	**Страницы**
Иан Соммервилл Инженерия программного обеспечения	***Глава 8:*** ***Стр. 169 - 187***

Цели изучения прототипирования:

1. Понять роль прототипирования в процессе разработки ПО
2. Знать различие между эволюционным и экспериментальным прототипированием
3. Знать три метода разработки прототипов:
   1) с использованием ЯП высокого уровня
   2) на основе баз данных
   3) на основе повторного использования программных компонентов
4. Понять, почему прототипирование- наиболее эффективная и удобная технология проектирования разработки пользовательских интерфейсов

Почему прототипирование является

Заказчикам программного обеспечения и конечным пользователям обычно сложно четко сформулировать требования к разрабатываемой программной системе. Трудно предвидеть, как система будет влиять на трудовой процесс, как она будет взаимодействовать с другими системами и какие операции, выполняемые пользователями, необходимо автоматизировать. Тщательный анализ требований помогает уменьшить неопределеиность относительно того, что система должна делать. Однако реально проверить требования, прежде чем их утвердить, практически невозможно. В этой ситуации может помочь прототип системы.

Прототип является начальной версией программной системы, которая используется для демонстрации концепций, заложенных в системе, проверки вариантов требований, а также поиска проблем, которые могут возникнуть как в ходе разработки, так и при эксплуатации системы, и возможных вариантов их решения. Очень важна быстрая разработка прототипа системы, чтобы пользователи могли начать экспериментировать с ним как можно раньше.

Так как основной опасностью при разработке ПО являются ошибки в требованиях, то прототип ПО используют прежде всего в целях
Повышения качества процесса разработки требований:
1)Постановка (впервые формулируемые) требований
(пример использования справочников при заполнении полей формы, файл ПИ-Лек10-dinList.mdb)
2)Проверка сформулированных требований

Наряду с тем что прототипы помогают формировать требования, они имеют и другие достоинства.

1. Различное толкование требований разработчиками ПО и пользователями можно выявить при демонстрации действующего прототипа системы.
2. В процессе создания прототипа разработчики могут выявить неполные или несогласованные требования.
3. Работая, хотя и ограниченно, в виде прототипа, система может продемонстрировать свои слабые и сильные стороны.
4. Прототип может служить основой для написания спецификации высококачественной системы. Разработка прототипа обычно ведет к улучшению спецификации системы.

Действующий прототип может также использоваться для других целей [178].

1. *Обучение пользователя.* Прототип системы можно использовать для обучения персонала перед поставкой окончательного варианта системы.
2. *Тестирование системы.* Прототипы позволяют "прокручивать" тесты. Один и тот же тест запускается на прототипе и на системе. Если получаются одинаковые результаты, это означает, что тест не обнаружил дефектов в системе. Если результаты отличаются, то необходимо исследовать причины различия, что позволяет выявить возможные ошибки в системе.

На основе изучения 39 различных программных проектов, использовавших прототипирование, в работе [133] сделан вывод, что эффективность применения прототипов при разработке ПО состоит в следующем.

1. Улучшаются эксплуатационные качества системы.
2. Система больше соответствует потребностям пользователей.
3. Системная архитектура становится более совершенной.
4. Сопровождение системы упрощается и становится более удобным.
5. Сокращаются расходы на разработку системы.

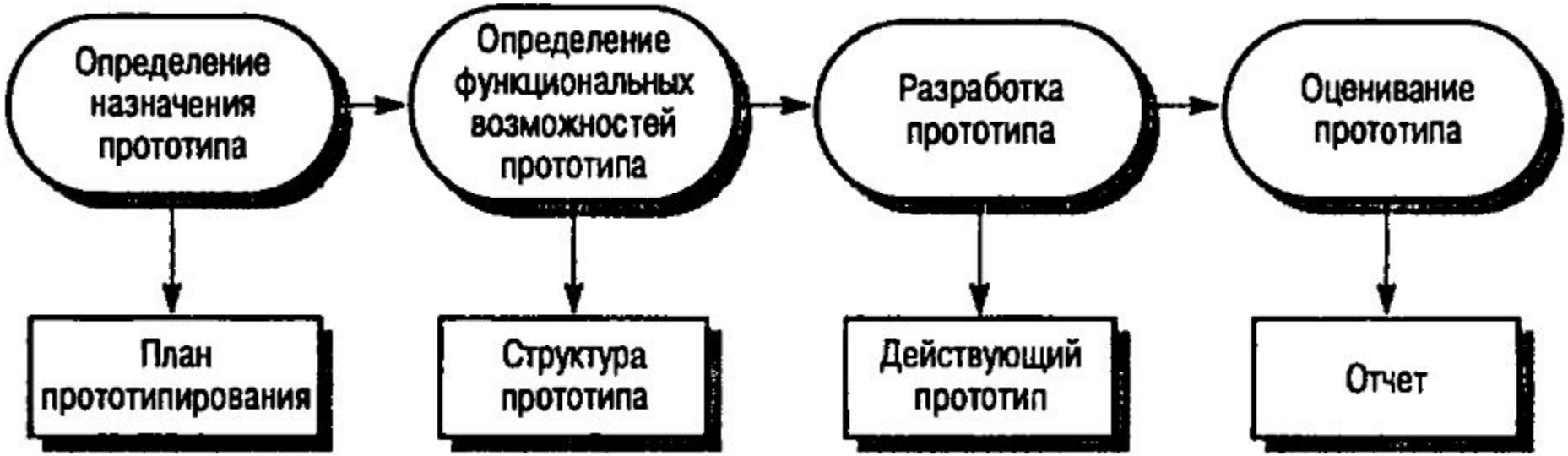


Рис. 8.1. Процесс разработки прототипа

Прототипирование в процессе

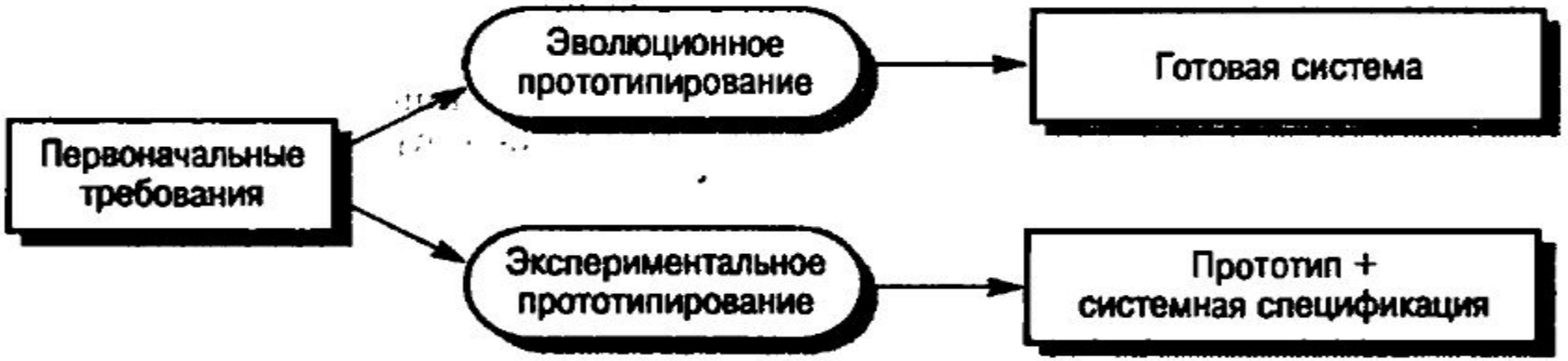
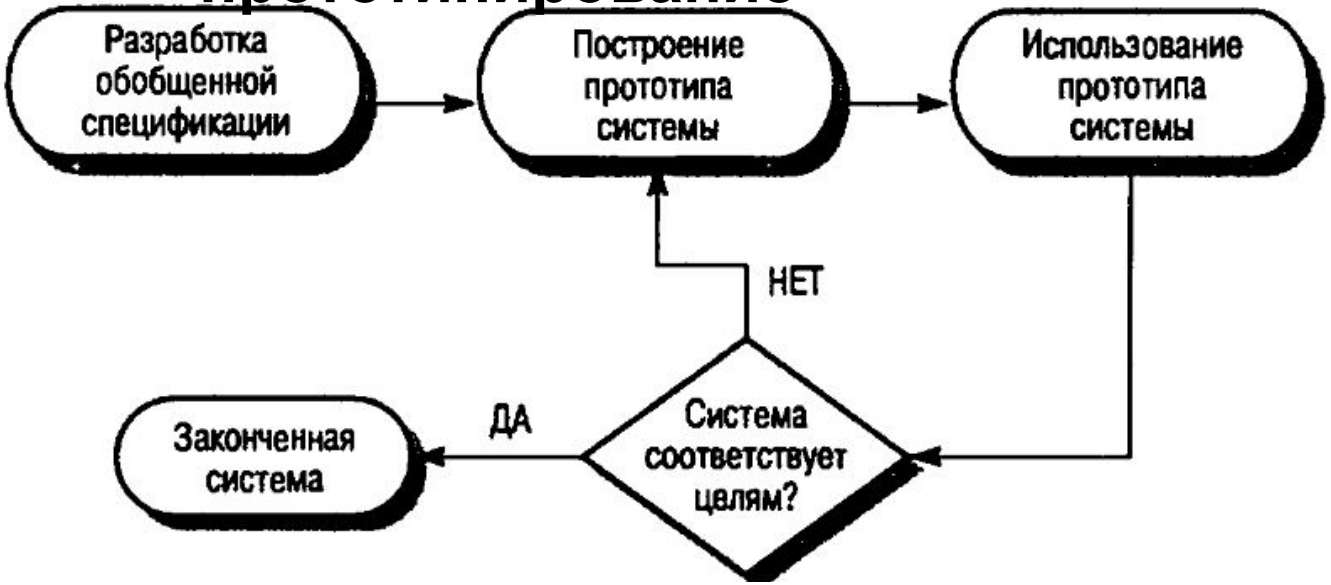


Рис. 8.2. Эволюционное и экспериментальное прототипирование

Эволюционное прототипирование начинается с построения относительно простой системы, которая реализует наиболее важные требования пользователя. По мере выявления новых требований прототип изменяется и дополняется. В конечном счете он становится той системой, которая требуется. В этом процессе не используется детальная системная спецификация, во многих случаях нет даже формального документа с системными требованиями. В настоящее время эволюционное прототипирование является обычной технологией разработки программных систем, которая широко используется при разработке Web-узлов и приложений электронной коммерции.

В противоположность эволюционному подходу метод экспериментального прототипирования предназначен для разработки и уточнения системной спецификации. Прототип создается, оценивается и модифицируется. Данные оценивания прототипа используются для дальнейшей детализации спецификации. Когда системные требования сформированы, прототип больше не нужен.

Эволюционное прототипирование



Этот метод прототипирования имеет два основных преимущества.

1. *Ускорение разработки системы.* Как указывалось во введении, современные темпы изменений в деловой сфере требуют быстрых изменений программного обеспечения. В некоторых случаях быстрая поставка ПО, удобство и простота его использования более важны, чем полный спектр функциональных возможностей системы или долгосрочные возможности ее сопровождения.
2. *Взаимодействие пользователя с системой.* Участие пользователей в процессе разработки означает, что в системе более полно будут учтены пользовательские требования.

Между отдельными методами быстрой разработки ПО существуют различия, но все они имеют некоторые общие свойства.

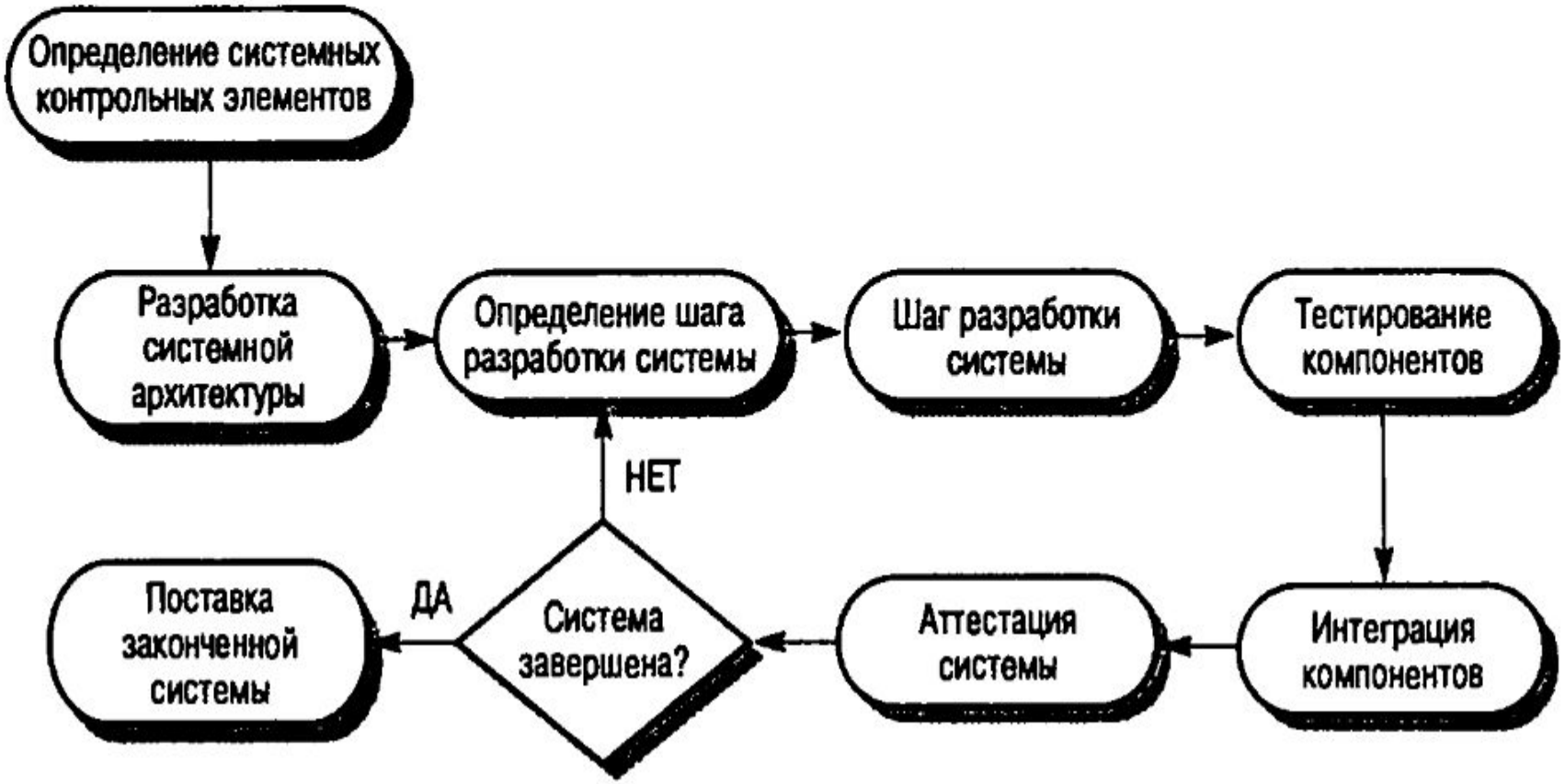
Эволюционное прототипирование и методы, основанные на использовании детальной системной спецификации, отличаются подходами к верификации и аттестации систем. Верификация – процесс проверки системы на соответствие спецификации. Поскольку для прототипа не создается подробной спецификации, его верификация невозможна.

Существует три основные проблемы эволюционного прототипирования, которые необходимо учитывать, особенно при разработке больших систем с длительным сроком жизненного цикла.

1. *Проблемы управления.* Структура управления разработкой программных систем строится в соответствии с утвержденной моделью процесса создания ПО, где для оценивания очередного этапа разработки используются специальные контрольные проектные элементы (см. главу 4). Прототипы эволюционируют настолько быстро, что создавать контрольные элементы становится нерентабельно. Кроме того, быстрая разработка прототипа может потребовать применения новых технологий. В этом случае может возникнуть необходимость привлечения специалистов с более высокой квалификацией.

2. *Проблемы сопровождения системы.* Из-за непрерывных изменений в прототипах изменяется также структура системы. Это означает, что система будет трудна для понимания всем, кроме первоначальных разработчиков. Кроме того, может устареть специальная технология быстрой разработки, которая использовалась при создании прототипов. Поэтому могут возникнуть трудности при поиске людей, которые имеют знания, необходимые для сопровождения системы.
3. *Проблемы заключения контрактов.* Обычно контракт на разработку систем между заказчиком и разработчиками ПО основывается на системной спецификации. При отсутствии таковой трудно составить контракт на разработку системы. Для заказчика может быть невыгоден контракт, по которому приходится просто платить разработчикам за время, потраченное на разработку проекта; также маловероятно, что разработчики согласятся на контракт с фиксированной ценой, поскольку они не могут предвидеть все прототипы, которые потребуется создать в процессе разработки системы.

Пошаговая разработка (рис. 8.4) позволяет избежать некоторых проблем, характерных для эволюционного прототипирования. Общая архитектура системы, определенная на раннем этапе ее разработки, выступает в роли системного каркаса. Компоненты системы разрабатываются пошагово, затем включаются в этот каркас. Если компоненты аттестованы и включены в каркас, ни архитектура, ни компоненты уже не меняются, за исключением случая, когда обнаруживаются ошибки.



Экспериментальное прототипирование

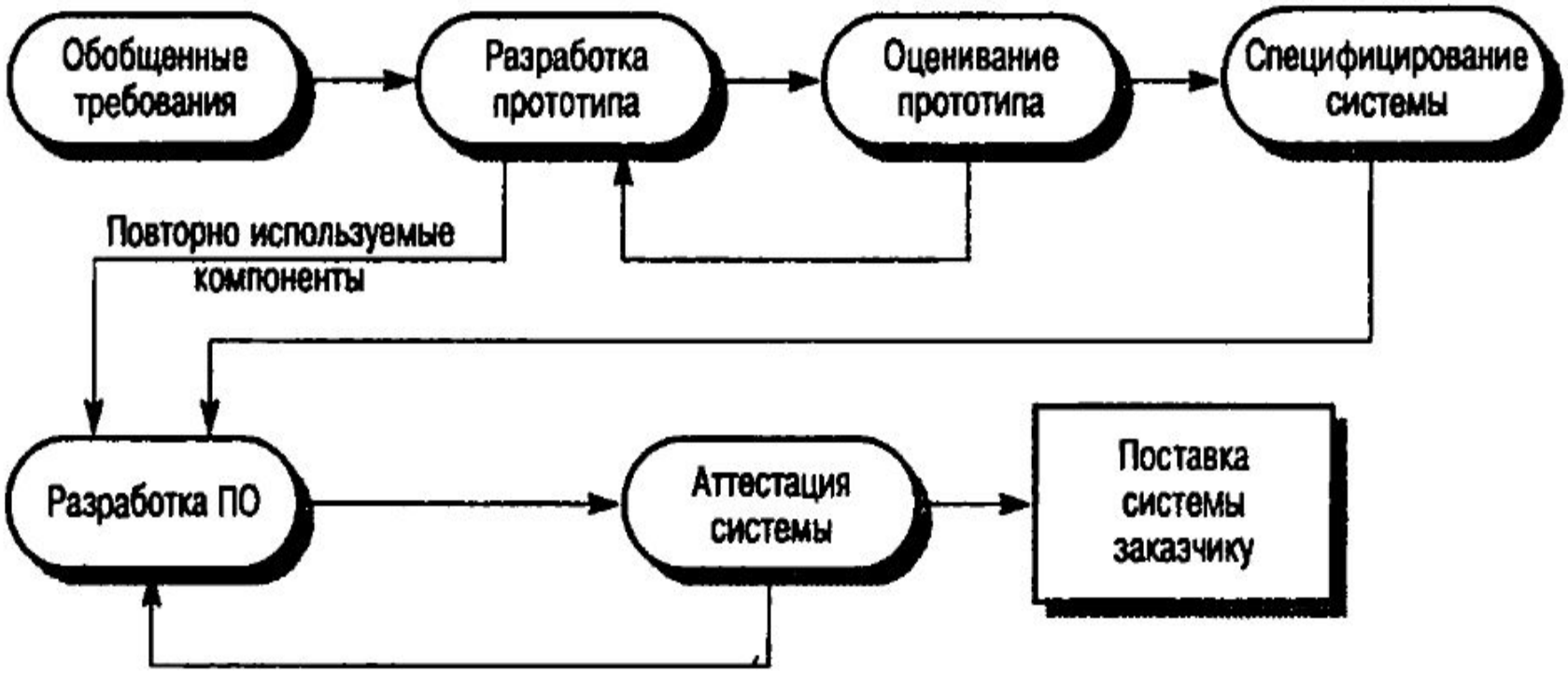


Рис. 8.5. Разработка ПО с использованием экспериментальных прототипов

Этот метод прототипирования обычно используется для разработки аппаратных систем. Прежде чем будет начата дорогостоящая разработка системы, создается макет (прототип), который используется для проверки структуры системы. Электронный макет системы создается с использованием готовых компонентов, что позволяет разработать версию системы до того, как будут вложены денежные средства в разработку специализированных интегральных схем.

Экспериментальный прототип программных систем обычно не используется для проверки архитектуры системы, он помогает разработать системные требования. Прототип часто совершенно не похож на конечную систему. Система разрабатывается по возможности быстро, поэтому для ускорения формирования требований используется упрощенный прототип системы. В экспериментальный прототип закладываются только обязательные системные функции, стандарты качества для прототипа могут быть снижены, критерии эффективности игнорируются. Язык программирования прототипа часто отличается от языка программирования, на котором будет создаваться окончательный вариант системы.

В модели процесса разработки ПО, показанной на рис. 8.5, предполагается, что прототип разрабатывается исходя из обобщенных системных требований, далее над прототипом проводятся эксперименты и он изменяется до тех пор, пока его функциональные возможности не удовлетворят заказчика. После этого на основе прототипа детализируются системные требования, реализуется обычная для организации-разработчика технология разработки ПО и система доводится до окончательной версии. Некоторые компоненты прототипа могут использоваться в системе, поэтому стоимость разработки может быть снижена.

Чтобы быть полезными в процессе разработки требований, экспериментальные прототипы не обязательно должны выполнять роль реальных макетов систем. Бумажные формы, имитирующие пользовательские интерфейсы систем [292], показали свою эффективность при формировании требований пользователя, в уточнении проекта интерфейса и при создании сценариев работы конечного пользователя. Они очень дешевы в разработке и могут быть созданы за несколько дней.

Разработчики иногда подвергаются давлению менеджеров для ускорения работы над прототипом, особенно если намечается задержка в поставке окончательной версии системы. Обычно такое "ускорение" порождает ряд проблем.

1. Невозможно быстро настроить прототип для выполнения таких нефункциональных требований, как производительность, защищенность, устойчивость к сбоям и безотказность, которые игнорировались во время разработки прототипа.
2. Частые изменение во время разработки неизбежно приводят к тому, что прототип плохо документирован. Для разработки прототипа используется только спецификация системной архитектуры. Этого недостаточно для долговременного сопровождения системы.
3. Изменения, сделанные во время разработки прототипа могут нарушить архитектуру системы. Ее обслуживание будет сложным и дорогостоящим.
4. В процессе разработки прототипа ослабляются стандарты качества.

Технологии быстрого прототипирования

Три основных метода быстрого

1. Разработка с применением динамических языков высокого уровня.
2. Использование языков программирования баз данных.
3. Сборка приложений с повторным использованием компонентов.

Таблица 8.1. Языки высокого уровня, используемые при прототипировании

Язык	Тип языка	Тип приложения
Smalltalk	Объектно-ориентированный	Интерактивные системы
Java	Объектно-ориентированный	Интерактивные системы
Prolog	Логический	Системы обработки символьной информации
Lisp	Основанный на списках	Системы обработки символьной информации

Динамические языки высокого уровня для создания прототипа можно использовать совместно, когда различные части прототипа программируются на разных языках. В работе [350] описывается разработка прототипа телефонной сетевой системы, где были использованы четыре различных языка: Prolog для макетирования баз данных, Awk [5] для составления счетов, CSP [163] для спецификации протоколов и PAISLey [351] для имитирования работы системы.

Не существует идеального языка для прототипирования больших систем, поскольку обычно различные части системы разнотипны. Преимущество многоязычного подхода в том, что для создания каждого компонента можно подобрать наиболее подходящий язык и таким образом ускорить разработку прототипа. Недостаток такого подхода в том, что трудно разработать коммуникационные связи для компонентов, написанных на разнородных языках.

Программирование баз данных

Эволюционная разработка в настоящее время является стандартной методикой для создания бизнес-приложений малого и среднего размера. Большинство бизнес-приложений включают в себя систему управления базой данных и обработку данных, находящихся в ней.

Для поддержки разработки таких приложений все коммерческие системы управления базами данных имеют внутренние средства программирования. Программирование баз данных выполняется на основе специализированных языков, которые имеют встроенную базу знаний и средства, необходимые для работы с базами данных. Рабочая среда поддержки языка обеспечивает инструментальные средства для создания пользовательских интерфейсов, числовых вычислений и отчетов. Термин *язык четвертого поколения* применяется как к самому языку программирования баз данных, так и к его рабочей среде.

Обычно рабочая среда языков четвертого поколения включает следующие инструментальные средства (рис. 8.6).

1. В качестве языка программирования баз данных (точнее, языка запросов к базе данных) обычно используется SQL [87].
2. Генератор интерфейсов используется для создания форм ввода и отображения данных.
3. Электронная таблица применяется для анализа данных и выполнения различных действий над числовой информацией.
4. Генератор отчетов предназначен для создания отчетов на основе информации, содержащейся в базе данных.

Рис. 8.6. Компоненты языка четвертого поколения

Большинство бизнес-приложений предполагают структурированные формы для ввода и вывода данных, поэтому языки четвертого поколения обеспечивают мощные средства для определения экранных форм и создания отчетов. Экранные формы часто определяются как ряд взаимосвязанных форм (в одном приложении, которое мы исследовали, было 137 различных форм), поэтому система, генерирующая экраны, должна обеспечивать следующее.

1. *Интерактивное определение форм*, когда разработчик определяет поля ввода и их организацию.
2. *Связывание форм*, когда разработчик задает определенные данные, ввод которых вызывает отображение дальнейших форм.
3. *Проверка входных данных*, когда разработчик при формировании полей форм определяет дозволенный диапазон входных величин.

В настоящее время большинством языков четвертого поколения поддерживается разработка интерфейсов баз данных, основанных на Web-броузерах. Они делают базу данных доступной с помощью Internet. Это снижает стоимость обучения и программного обеспечения и позволяет внешним пользователям иметь доступ к базе данных. Однако ограничения протоколов Internet и медленный просмотр Web-страниц делают этот метод не подходящим для систем, в которых требуется быстрое взаимодействие с пользователем.

Это написано в 2006 году!

Сборка приложений с повторным использованием компонентов

Время, необходимое для разработки системы, можно уменьшить, если многие части такой системы будут использованы неоднократно. Для быстрого построения прототипа необходимо иметь набор компонентов, пригодных для повторного использования, и механизм сборки системы из этих компонентов. Этот подход показан на рис. 8.7.

Рис. 8.7. Сборка повторно используемых компонентов

Для понимания этого метода разработки прототипа полезен составной документ, который представляет собой схему обработки данных прототипом и который можно рассматривать как контейнер для нескольких различных объектов. Эти объекты содержат разные типы данных (такие, как таблица, диаграмма, форма), которые могут обрабатываться различными приложениями.

На рис. 8.8 представлен составной документ для прототипа системы, включающего текстовые элементы, элементы электронной таблицы и звуковые файлы. Текстовые элементы обрабатываются текстовым процессором, таблицы – электронной таблицей, а звуковые файлы – аудиопроигрывателем. Когда пользователь системы обращается к объекту определенного типа, вызывается связанное с ним приложение.

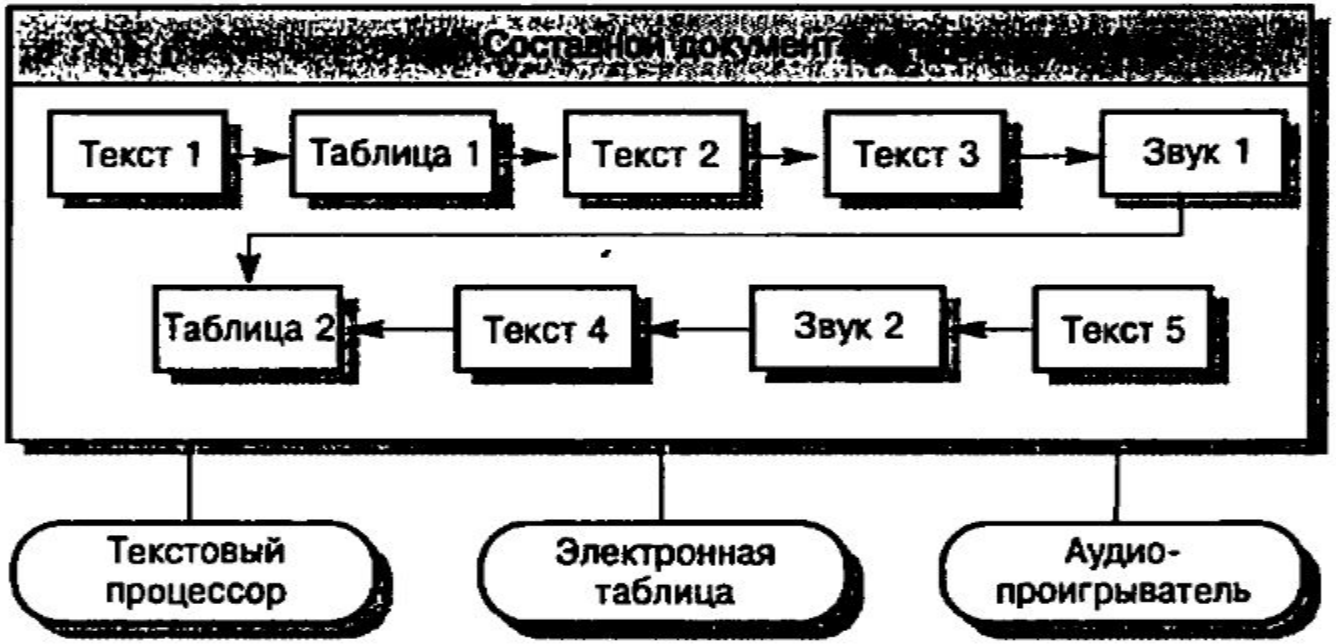


Рис. 8.8. Связывание приложений посредством составного документа

Основное преимущество описываемого подхода к прототипированию состоит в том, что многие функциональные средства прототипа можно реализовать быстро и дешево. Если пользователи, тестирующие прототип, знакомы с приложениями, интегрированными в прототип, им нет необходимости учиться использовать новые средства прототипа. Проблемы при работе с прототипом могут возникнуть только при переключении с одного приложения на другое. Но это в значительной степени зависит от используемой операционной системы. Для организации переключения между приложениями наиболее широко используется механизм связывания и внедрения объектов OLE от Microsoft

Визуальные системы разработки приложений, подобные Visual Basic, поддерживают повторное использование компонентов. Разработчики строят систему в интерактивном режиме, определяя интерфейс в виде набора экранных форм, полей, кнопок и меню. Эти элементы интерфейса именованы, и каждый из них связан со сценарием обработки событий и данных. Эти сценарии могут вызывать повторно используемые программные компоненты.

На рис. 8.9 показан экран приложения, содержащий меню (в верхней части), поля ввода (белые области слева на экране), поля вывода (затенная область слева) и кнопки, представленные скругленными прямоугольниками справа на экране. После расположения этих графических элементов на экране разработчик определяет, какие готовые компоненты будут связаны с ними, либо пишет программы, выполняющие необходимые действия. На рис. 8.9 показаны компоненты, связанные с некоторыми отображаемыми элементами экрана.

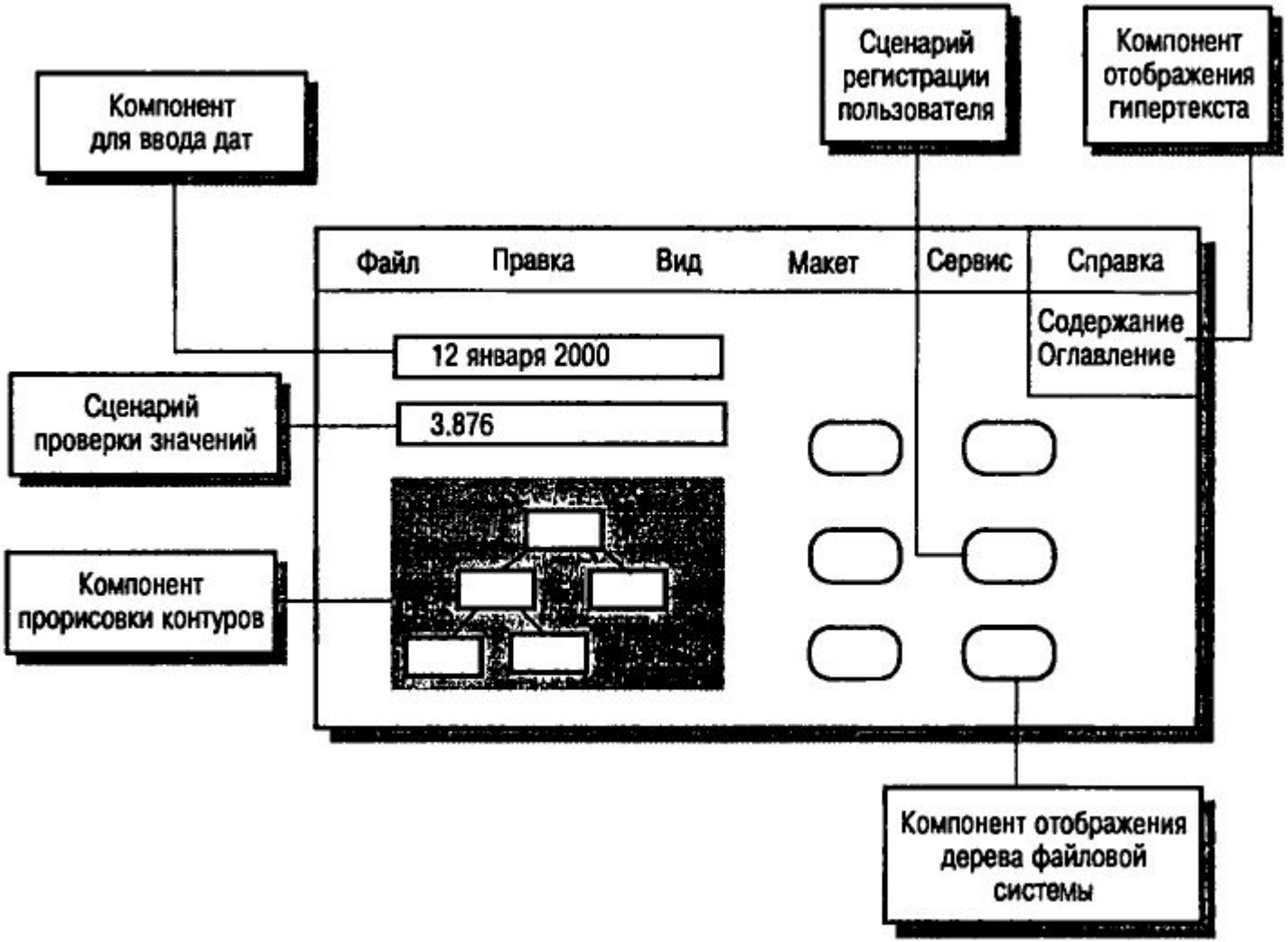


Рис. 8.9. Визуальное программирование с повторным использованием компонентов