МИНИСТЕРСТВО ОБР МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра Автоматизированных Систем Управления

Конспект

по дисциплине Программная инженерия

на книгу Соммервилла, Иан. «Инженерия программного обеспечения»

по части 7 Эволюция программного обеспечения

Выполнил студент группы

АВТ-812 Глинин Евгений

Принял

Астапчук Виктор Андреевич

Новосибирск 2021 г

СОДЕРЖАНИЕ

[1 Наследуемые системы 3](#_Toc87275423)

[1.1. Структуры наследуемых систем 3](#_Toc87275424)

[1.2. Проектирование наследуемых систем 3](#_Toc87275425)

[1.3. Оценивание наследуемых систем 3](#_Toc87275426)

[2. Модернизация программного обеспечения 4](#_Toc87275427)

[2.1. Динамика развития программ 4](#_Toc87275428)

[2.2. Сопровождение программного обеспечения 4](#_Toc87275429)

[2.3. Эволюция системной архитектуры 4](#_Toc87275430)

[3. Реинжиниринг программного обеспечения 5](#_Toc87275431)

[3.1. Преобразование исходного кода программ 5](#_Toc87275432)

[3.2. Анализ систем 5](#_Toc87275433)

[3.3. Совершенствование структуры программ 5](#_Toc87275434)

[3.4. Создание программных модулей 5](#_Toc87275435)

[3.5. Изменение данных 5](#_Toc87275436)

[4. Управление конфигурациями 6](#_Toc87275437)

[4.1. Планирование управления конфигурацией 6](#_Toc87275438)

[4.2. Управление изменениями 6](#_Toc87275439)

[4.3. Управление версиями и выпусками 6](#_Toc87275440)

[4.4. Сборка системы 6](#_Toc87275441)

[4.5. CASE-средства для управления конфигурацией 6](#_Toc87275442)

# 1 Наследуемые системы

Многие менеджеры не желают рисковать, устанавливая неизвестное и непредсказуемое современное

ПО. Замена наследуемой системы —дело рискованное по многим причинам.

1. Редко можно найти такую наследуемую систему, которая имеет полное и точное техническое описание. Старое описание может быть утеряно, а если оно и существует, вряд ли там будут указаны все изменения, сделанные в системе. Поэтому трудно сравнивать технические характеристики и функциональные возможности старой системы с характеристиками ее возможной преемницы.

2. Функционирование наследуемой системы тесно связано с деловой активностью компапин. При замене системы деятельность компании также претерпит изменения, что может привести к непредсказуемым расходам и необратимым последствиям.

3. Некоторые встроенные в систему правила, регулирующие область торговопромышленных отношений компании, могут быть нигде не документированы. Эти правила обеспечивают своеобразные рамки, в которых должна вестись коммерческая деятельность, и нарушение этих рамок окажет не самое лучшее влияние на раэшггне бизнеса. Например, страховая компания могла включить в систему правила оценки риска страховых полисов. Если эти правила не выполняются, компания может быть вовлечена в работу с полисами высокой степени риска, что в дальнейшем вызовет большие затраты на выплату страховых возмещений.

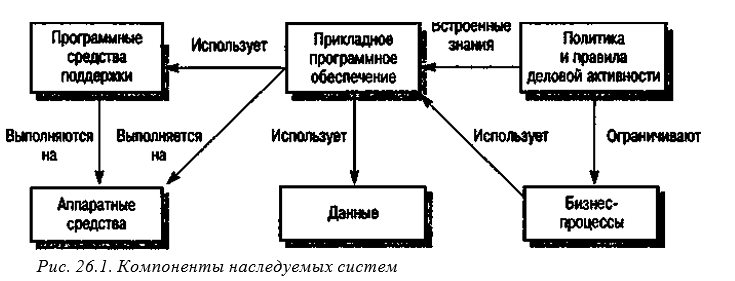
4. Создание новых программных систем связано с риском, так как новизна системы подразумевает появление непредусмотренных проблем. Поставщик ПО может доставить программный продукт не вовремя, либо может измениться его цена.

Использование наследуемых систем избавляет организацию от риска, связанного с их заменой. Однако модернизация старой системы становится дороже с каждым годом эксплуатации. Возрастающая стоимость модернизации системы, находящейся в эксплуатации несколько лет, определяется многими факторами.

# 1.1. Структуры наследуемых систем

Понятие - "наследуемая система" гораздо шире понятия "старые и давно используемые системы ПО", хотя именно программный компонент этих систем нас интересует больше всего. Наследуемая система представляет собой сложную социотехиическую систему, основанную на использовании вычислительной техники, которая включает программное обеспечение, аппаратные средства, используемые данные и бизнес-процессы. Изменения одной из составляющих системы влечет за собой изменение других ее компонентов. Эти системы разрабатывались с учетом организационных стратегий и планов конкретной организации, но не всегда учитывали объективные инженерные критерии.

Логические составляющие наследуемых систем и взаимосвязи между ними показаны на рис. 26.1.



На практике вмешательство в один уровень обязательно повлечет за собой изменения на других уровнях. Это происходит по нескольким причинам.

1. Изменения на каком-либо уровне в большинстве случаев связаны с внедрением новых средств. Чтобы вышестоящий уровень мог использовать эти средства, его нужно также изменить. Например, на уровень программных средств поддержки внедряется новая база данных, которая предоставляет доступ к данным с помощью

Web-броузера. Тогда уровень бизнес-процессов потребуется изменить для того,

чтобы иметь возможность использовать это средство.

2. Изменение программного обеспечения может снизить скорость выполнения системы, для ее повышения нужно установить новые аппаратные средства. Повышение производительности системы провоцирует внесение изменений для использования возможностей, которые ранее были недоступны.

3. Сохранение интерфейсов аппаратных средств со временем часто становится невозможным, особенно в случае кардинальных изменений в аппаратном обеспечении. Такое может случиться с компанией, которая решит перейти от мэйнфреймов к системе клиент/сервер, где, как правило, работают с разными операционными системами. В этом случае необходима серьезная модернизация прикладного программного обеспечения.

Наследуемые системы с централизованными базами данных также имеют недостатки.

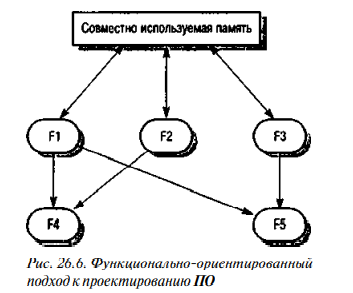
1. Система управления базой данных может быть устаревшей и несовместимой с современными СУБД, используемыми в бизнесе. Из всех современных систем баз данных, применяемых в бизнесе, наиболее эффективными считаются реляционные базы данных. Однако многие наследуемые системы используют иерархические пли с степью базы данных. Такие базы данных создавались скорее для повышения функциональности системы, чем для удобства управления данными. Современная вычислительная техника снимает требование повышения функциональности системы, но переход к реляционным моделям данных может оказаться слишком дорогостоящим.

2. Монитор дистанционной обработки часто создавался для специализированных баз данных, рассчитанных на эксплуатацию на мэйнфреймах. Поэтому такой монитор не подойдет для современных баз данных. Эта часть системы подлежит обязательной замене, вследствие чего возрастают расходы и риск, связанные с изменениями системы.

# 1.2. Проектирование наследуемых систем

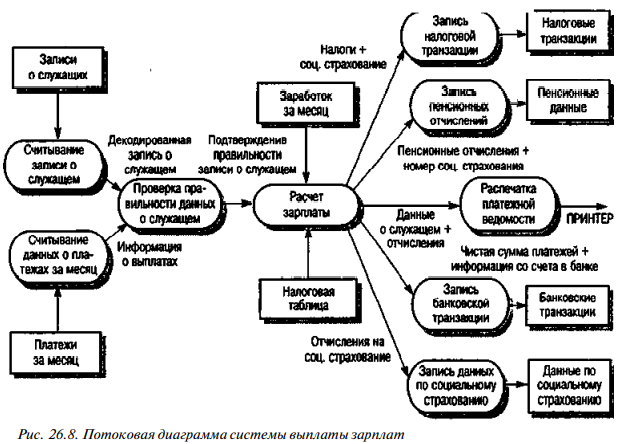
Практически все наследуемые системы были созданы до того, как объектно-ориентированный подход стал широко использоваться при создании ПО. Поэтому, вместо того чтобы представлять собой совокупность взаимосвязанных объектов, программы в таких системах структурированы как множество подпрограмм и функций. Каждая подпрограмма обеспечивает определенную часть функциональности системы и в случае необходимости вызывается другими подпрограммами. В некоторых же языках программирования подпрограммы оперируют собственными данными, имея в то же время доступ к совместно используемым данным.

Стратегия функционально-ориентированного проектирования ПО предусматривает декомпозицию программ на ряд функций и подпрограмм, взаимодействующих с централизованной совместно используемой памятью (рис. 26.6.).



Функционально-ориентированное проектирование скрывает детали алгоритмов в подпрограммах и функциях, однако информация о состоянии системы при этом открыта. В этом могут таиться проблемы, поскольку функция способна изменить состояние системы непредвиденным образом.

При проектировании функционально-ориентированных систем часто используются диаграммы потоков данных, которые описаны в главе 7. Диаграммы потоков данных — это функциональное представление, где прямоугольник с закругленными краями представляет функцию, выполняющую преобразование данных, а стрелка — элемент данных, обрабатываемый функцией. Файлы и другие хранилища данных представлены в виде прямоугольников. Диаграммы отображают сквозной процесс обработки, т.е. показывают все функции системы, которые взаимодействуют с данными, когда они (данные) проходят по разным стадиям обработки и преобразований. Для иллюстрации функционально-ориентированного проектирования с помощью диаграммы потоков данных рассмотрим рис. 26.8, на котором показана структура системы по расчету заработной платы. Это система пакетной обработки данных. Она считывает информацию о служащих компании, затем рассчитываются платежи и отчисления за месяц, после чего начисляется заработная плата. Рассмотрим, как эта система реализует структуру "вход-процесс-выход".



Удачным примером системы обработки транзакций является система, контролирующая сеть банкоматов. На услуги, предоставляемые пользователям, не влияют предварительные операции, поэтому они могут рассматриваться как отдельные транзакции.

# 1.3. Оценивание наследуемых систем

Организации, деятельность которых во многом зависит от наследуемых систем и средства которых на их сопровождение и модернизацию ограниченны, должны хорошо подумать над тем, как получить максимум от вложений в наследуемую систему. Это прежде всего означает корректную оценку наследуемой системы и выбор наиболее подходящей стратегии ее модернизации. Существует четыре стратегических пути решения этой задачи:

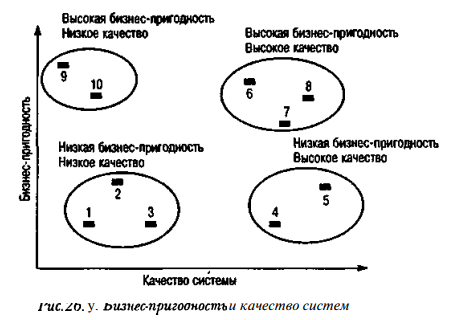
1. Полностью отказаться от системы. Это решение применимо в случае, если система не отвечает своим задачам поддержки бизнес-процессов. Например, со времени установки системы бизнес-процессы изменились настолько, что уже практически не зависят от работы системы. Чаще всего это случается там, где универсальные ЭВМ были заменены персональными компьютерами, а устаревшее программное обеспечение было модернизировано в той мере, в которой это необходимо для продолжения его работы на ПК.

2. Продолжить сопровождение системы. Это решение подходит в ситуациях, когда система более или менее стабильна и все еще полезна в работе, а пользователи не требуют ее значительного изменения.

3. Модернизировать систему для улучшения сопровождения. Этот путь следует выбрать тогда, когда качество работы системы снизилось и результате частых изменений, причем дальнейшие изменения все еще будут необходимы.

4. Заменить старую систему баке повой. Этот вариант применяется в том случае, если в связи с появлением современных аппаратных средств старая система становится непригодной в эксплуатации или если уже имеются подобные системы и разработка новых на их основе не будет слишком дорогостоящей.

При оценивании наследуемую систему нужно рассматривать под разными углами зрения [339]. С коммерческой точки зрения необходимо провести оценку полезности и пригодности системы для бизнеса. Что же касается перспективы дальнейшей работы системы, нужно в первую очередь оценить качество прикладного ПО, а также программных и аппаратных средств поддержки данной системы. Комбинация двух оценок— бизнес пригодность и качество — поможет решить, что же делать с наследуемой системой дальше. Для демонстрации применения такой комплексной оценки рассмотрим организацию, использующую в работе 10 наследуемых систем. Качество и бизнес-пригодность каждой из этих систем были оценены с помощью некоторых количественных показателей. Результаты оценивания представлены на рис. 26.9.



На рис. 26.9 видно четыре группы систем, которые определяются следующими оценками.

1. Низкое качество, низкая бизнес-пригодность. Решение оставить такие системы в действии дорого обойдется, а отдача в бизнесе будет незначительной. Такие системы — прямые кандидаты на списание.

2. Низкое качество, высокая бизнес-пригодность. Такие системы вносят немалый вклад в бизнес-деятельность, поэтому списывать их нельзя. Однако низкий уровень качества означает высокие расходы на сопровождение системы. Такие системы подлежат модернизации или замене (при условии наличия другой подходящей системы).

3. Высокое качество, низкая бизнес-пригодность. Такие системы не очень полезны, но недороги в эксплуатации. Риск вследствие замены таких систем неоправданно, поэтому их можно оставить в работе и в дальнейшем списать.

4. Высокое качество, высокая бизнес-пригодность. Их можно оставить, ведь высокое качество означает отсутствие необходимости модернизации или замены системы. Поэтому с ними продолжаем работать, как и прежде. В идеале для решения того, что делать с системой дальше, должны использоваться подобные объективные оценки. Однако неоднократно отмечалось, что в действительности при принятии таких решений главную роль играют организационные или политические соображения. Например, при слиянии компаний будут использоваться системы более сильного (в политическом смысле) партнера, а все другие системы будут списаны.

# 2. Модернизация программного обеспечения

Невозможно создать систему, которая не потребует изменений в будущем. Как только программное обеспечение вводится в эксплуатацию, возникают новые требования к системе, обусловленные непрерывным развитием бизнес-процессов и все возрастающими общими требованиями к программным системам. Иногда в системе следует изменить некоторые составляющие в целях повышения производительности или улучшения других характеристик, а также для исправления обнаруженных ошибок. Все это требует дальнейшего развития системы после ее ввода в эксплуатацию

Существует несколько стратегических подходов к процессу модернизации ПО.

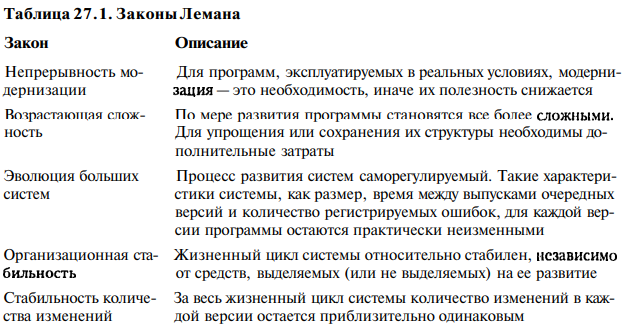
1. Сопровождение программного обеспечения. Это наиболее часто используемый подход, который заключается в изменении отдельных частей ПО в ответ на растущие требования, по с сохранением основной системной структуры.

2. Эволюция системной архитектуры. Этот подход более радикальный, чем сопровождение ПО, так как предполагает существенные изменения в программной системе. Эта стратегия модернизации ПО.

3. Реинжиниринг программного обеспечения. Кардинально отличается от других подходов, так как модернизация предусматривает не внесение каких-то новых компонентов, а наоборот, упрощение системы и удаление из нее всего лишнего. При этом возможны изменения в архитектуре, но без серьезных переделок.

# 2.1. Динамика развития программ

Под динамикой развития программ подразумевается исследование изменений в программной системе. Основной работой в этой области является [213]. Результатом этих исследований стало появление ряда "законов" Лемана (Lehman), относящихся к модернизации систем. Считается, что эти законы неизменны и применимы практически во всех случаях. Они сформулированы после исследования процесса создания и эволюции ряда больших программных систем. Эти законы (в сущности, ие законы, а гипотезы) приведены в табл. 27.1



Из первого закона вытекает необходимость постоянного сопровождения системы. При изменении окружения, в котором работает система, появляются новые требования, и система должна неизбежно изменяться с тем, чтобы им соответствовать.

Второй закон констатирует нарушение структуры системы после каждой модификации. Единственным способом избежать этого, по всей видимости, является только профилактическое обслуживание, которое, однако, требует средств и времени.

Третий закон согласно этому закону, все большие системы имеют собственную динамику изменений, которая устанавливается на начальном этапе разработки системы.

Четвертым закон Лемана утверждает, что крупные проекты по разработке программного обеспечения действуют в режиме "насыщения". Это означает, что изменения ресурсов или персонала оказывает незначительное влияние на долгосрочное развитие системы.

Пятый лаком затрагивает проблему увеличения количества изменений с каждой новой версией программы. Расширение функциональных возможностей системы каждый раз сопровождается новыми ошибками в системе.

# 2.2. Сопровождение программного обеспечения

Сопровождение— это обычный процесс изменения системы после ее поставки заказчику. Эти изменения могут быть как элементарно простыми (исправление ошибок программирования), так и более серьезными, связанными с корректировкой отдельных недоработок либо приведением в соответствие с новыми требованиями. Как упоминалось в вводной части главы, сопровождение не связано со значительным изменением архитектуры системы. При сопровождении тактика простая: изменение существующих компонентов системы либо добавление новых.

Существует три вида сопровождения системы.

1. Сопровождение с целью исправления ошибок. Обычно ошибки в программировании достаточно легко устранимы, однако ошибки проектирования стоят дорого и требуют корректировки или перепрограммирования некоторых компонентов. Самые дорогие исправления связаны с ошибками в системных требованиях, так как здесь может понадобиться перепроектирование системы.

2. Сопровождение с целью адаптации ПО к специфическим условиям эксплуатации. Это может потребоваться при изменении определенных составляющих рабочего окружения системы, например аппаратных средств, операционной системы или программных средств поддержки. Чтобы адаптироваться к этим изменениям, система должна быть подвергнута определенным модификациям.

3. Сопровождение с целью изменения функциональных возможностей системы. В ответ па организационные или деловые изменения в организации могут измениться требования к программным средствам. В таких случаях применяется данный тип сопровождения. Наиболее существенные изменения при этом претерпевает именно программное обеспечение.

На практике однозначно четкое разграничение между различными видами сопровождения провести достаточно сложно. Ошибки в системе могут быть выявлены в том случае, если, например, система использовалась непредсказуемым способом. Поэтому наилучший способ исправления ошибок — расширение функциональных возможностей программы с тем, чтобы сделать работу с ней как можно проще.

# 2.3. Эволюция системной архитектуры

В процессе сопровождения большинство изменений проводится локализовано и не влияет па архитектуру системы. Однако начиная с 1980-х годов экономические показатели компьютерных систем изменилась настолько, что стало более выгодно применять распределенные, а не централизованные, как раньте, системы. Поэтому многие компании поставлены перед необходимостью преобразовать свои централизованные системы, реализованные на мэйнфреймах, в распределенные системы типа клиент/сервер.

Перечислим основные причины перехода от централизованных к распределенным системам.

1. Стоимость аппаратных средств. Закупка и сопровождение распределенных систем клиент/сервер обойдется гораздо дешевле, чем покупка мэйнфрейма эквивалентной мощности.

2. Усовершенствование пользовательских интерфейсов. Многие из наследуемых систем, основанных на мэйнфреймах, имеют текстовые интерфейсы, основанные на формах. Сегодня пользователям привычнее графические интерфейсы и более простое взаимодействие с системами. Такого рода интерфейсы требуют большего количества локальных вычислений и более эффективно работают в системах типа клиент/сервер.

3. Распределенный доступ к системам. Сейчас все больше компаний стараются децентрализовать рабочие места, что требует децентрализации компьютерных систем. При этом необходимо, чтобы компьютерные системы были доступны из разных мест и с разного оборудования. Например, сотрудники могут получить доступ к системам из собственного дома, и такую практику нужно поддерживать.

# 3. Реинжиниринг программного обеспечения

Реинжиниринг — это повторная реализация наследуемой системы в целях повышения удобства ее эксплуатации и сопровождения. В это понятие входят разные процессы, среди которых назовем повторное документирование системы, ее реорганизацию и реструктуризацию, перевод системы на один из более современных языков программирования, модификацию и модернизацию структуры и системных данных. При этом функциональность системы и ее архитектура остаются неизменными.

С технической точки зрения реинжиниринг — это решение "второго сорта" проблемы системной эволюции. Если учесть, что архитектура системы не изменяется, то сделать централизованную систему распределенной представляется делом довольно сложным. Обычно нельзя изменить язык программирования старых систем на объектно-ориентированные языки (например, Java или C++). Эти ограничения вводятся для сохранения архитектуры системы.

По сравнению с более радикальными подходами к совершенствованию систем реинжиниринг имеет два преимущества.

1. Снижение рисков. При повторной разработке ПО существуют большие риски— высока вероятность ошибок в системной спецификации и возникновения проблем во время разработки системы. Реинжиниринг снижает эти риски

2. Снижение затрат. Себестоимость реинжиниринга значительно ниже, чем разработка нового программного обеспечения. Для этой системы был успешно выполнен реинжиниринг стоимостью всего 12 млн. долларов. Приведенные цифры типичны: считается, что реинжиниринг в четыре раза дешевле, чем повторная разработка системы.

# 3.1. Преобразование исходного кода программ

Самый простой способ реинжиниринга программ— это автоматический перевод исходного кода с одного языка программирования на другой, более современный. При этом структура и организация программ остаются неизменными. Программа может переводиться как на обновленную версию исходного языка (например, с языка COBOL-74 на язык COBOL-85), так и на другой "не родственный" язык (например, с языка FORTRAN на С).

Причины перевода на другой язык могут быть следующие.

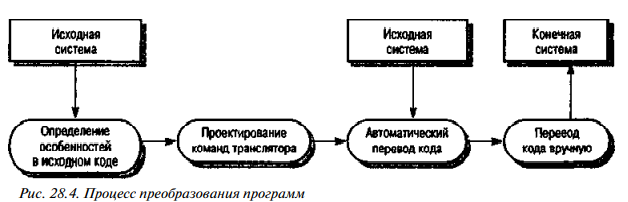
1. Обновление платформы аппаратных средств. В организации может быть принято решение по изменению аппаратной платформы. Новые аппаратные средства могут не поддерживать компиляторы исходного языка программ.

2. Недостаток квалифицированного персонала. Бывает, что для сопровождения программ на исходном языке невозможно найти достаточно квалифицированный персонал, особенно это касается программ, написанных на специфических языках, давно вышедших из употребления.

3. Изменения политики организации. Организация может принять решение о переходе на общий стандартный язык программирования, чтобы снизить затраты на сопровождение программных систем, поскольку сопровождение большого количества версий старых компиляторов невыгодно.

4. Недостаточно средств поддержки старого ПО. Поставщик компиляторов для старого языка программирования может уйти с рынка программных продуктов или прекратить поддержку своего продукта.

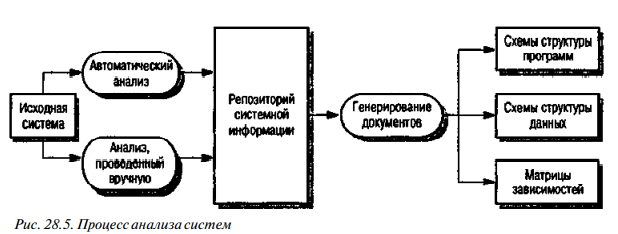
Процесс перевода исходного кода программ показан на рис. 28.4.



# 3.2. Анализ систем

Цель такого анализа — восстановление структуры и спецификации системы. Этот процесс не подразумевает изменения программ. Входными данными процесса анализа обычно служит исходный код системы. Однако зачастую даже он недоступен, тогда процесс анализа начинается с исполняемой программы.

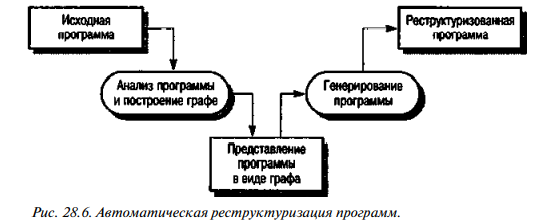
Анализ систем не тождественен реинжинирингу систем. Целью анализа является определение архитектуры и спецификации системы на основе ее исходного кода. Целью реинжиниринга можно назвать создание усовершенствованной и удобной в сопровождении системы. Но, как показано па рис. 28.2, анализ системы может быть составной частью процесса реинжиниринга. Схема процесса анализа системы приведена на рис. 28.5.



# 3.3. Совершенствование структуры программ

Если в процессе эксплуатации наследуемой системы возникла необходимость оптимизировать использование памяти и имеются проблемы с пониманием того, как она работает, это означает, что система плохо структурирована. Управляющая структура наследуемых систем обычно значительно усложнена множеством безусловных переходов и нечеткой логикой программного кода. Регулярное сопровождение системы также не способствует сохранению системной структуры. После частых изменений некоторые фрагменты кода становятся неиспользуемыми, однако это можно обнаружить только после тщательного анализа программы.

В известной работе доказано, что любую программу можно переписать с помощью простых условных операторов if-then-else и циклов while-loop, при этом можно исключить все безусловные операторы перехода. Этапы такого преобразования программ показаны на рис. 28.6.



Автоматизированный способ реструктуризации программ имеет свои проблемы.

1. Потеря комментариев. Если в программе есть встроенные комментарии, они будут утеряны в процессе реструктуризации.

2. Утрата документации. По той же причине обычно нарушается соответствие между новой программой и документацией на исходную программу. Однако в большинстве случаев это не так уж важно, поскольку документация и комментарии уже устарели.

3. Жесткие требования к компьютерной технике. Алгоритмы, встроенные в средства реструктуризации, отличаются высокой сложностью. Процесс реструктуризации больших программ, даже выполненный на современных быстродействующих компьютерах, будет занимать много времени.

# 3.4. Создание программных модулей

Это процесс реорганизации программы в целях объединения ее взаимосвязанных частей в отдельном модуле. После этого легче удалить избыточность в соответствующих компонентах, оптимизировать взаимосвязи и упростить интерфейс всей программы. Например, в программе по обработке сейсмографических данных все операции по графическому представлению данных можно собрать в один модуль. Если система будет распределенной, модули можно инкапсулировать как объекты, доступ к которым будет осуществляться через общий интерфейс.

В программной системе можно выделить различные типы модулей.

1. Абстракции данных. Это абстрактные типы данных, которые создаются путем объединения данных с компонентами их обработки.

2. Аппаратные модули. Тесно связаны с абстракцией данных и объединяют все функции, управляющие отдельными аппаратными устройствами.

3. Функциональные модули. Объединяют все функции, которые выполняют сходные или взаимосвязанные задачи. Например, в один модуль можно объединить все функции, выполняющие ввод данных и их проверку. Этот подход применяется там, где создание абстракций данных невыгодно.

4. Модули поддержки отдельных процессов. В них сгруппированы все функции и данные, отвечающие за поддержку отдельного бизнес-процесса. Например, в библиотечной системе присутствует модуль, объединяющий все функции, отвечающие за выдачу и возврат книг.

Разбиение программы на модули обычно выполняется вручную путем проверки и правки кода. Для этого следует, прежде всего, определить взаимосвязи между компонентами и изучить способ их взаимодействия. Полностью автоматизировать этот процесс нельзя, даже если привлечь средства просмотра и визуализации программ.

# 3.5. Изменение данных

До сих пор все обсуждаемые изменения касались в основном программ и систем. Однако в некоторых, случаях придется столкнуться с проблемой изменения данных. Хранение, структура и формат данных, с которыми работает наследуемая система, должны измениться, чтобы соответствовать изменениям в программном обеспечении. Изменение данных — это процесс анализа и реорганизации структуры данных, а иногда еще и изменение значений системных данных.

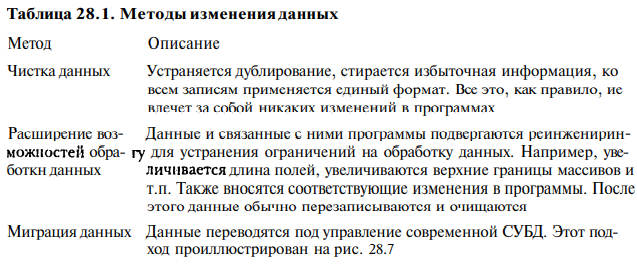
В принципе, если функциональность системы осталась прежней, изменения данных не требуется. Однако существует ряд причин, которые вынуждают изменять данные (так же, как и программы) наследуемой системы.

1. Нарушение данных. С течением времени качество данных снижается. Изменения данных становятся причиной новых ошибок, возможно дублирование значений, изменения во внешнем окружении системы могут не найти адекватного отражения в данных. Эти явления неизбежны, так как время существования данных бывает достаточно большим. Например, персональные данные в банковской системе появляются с созданием нового счета и существуют, по меньшей мере, в течение всей жизни клиента. При изменении обстоятельств у клиента банковские данные должны обновляться, что не всегда происходит корректно. Реинжиниринг системы уменьшает эти трудности, что лишний раз подтверждает его необходимость.

2. Программные ограничения. При разработке систем многие программисты включают в программы ограничения на количество обрабатываемых данных. Но согласно современным требованиям программы должны обрабатывать значительно больше данных, чем было предусмотрено изначально. Именно для устранения подобных ограничений может понадобиться изменение данных. В книге [296] приведен пример системы управления ценными бумагами, которая была способна обрабатывать до 99 транзакций за одну операцию. В компании, где эта система использовалась, осуществлялось управление 2000 транзакций, что вызвало необходимость в создании 23 копий системы. По этой причине впоследствии компания прияла решение о реинжиниринге системы и изменении данных.

3. Эволюция системной архитектуры. При переходе с централизованной системы на распределенную ядром архитектуры должна стать система управления данными с удаленным доступом. Для перемещения данных из отдельных файлов на сервер системы управления базой данных (СУБД) может потребоваться большая работа по изменению этих данных.

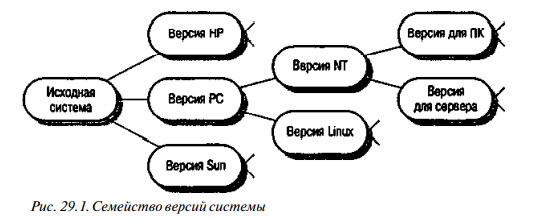
Как и в случае с реинжинирингом программ, изменение данных имеет свои подходы и методы, которые перечислены в табл. 28.1.



# 4. Управление конфигурациями

Управление конфигурацией— это процесс разработки и применения стандартов и правил по управлению эволюцией программных продуктов. Эволюционирующие системы нуждаются в управлении по той простой причине, что в процессе их эволюции создается несколько версий одних и тех же программ. В эти версии обязательно вносятся некоторые изменения, исправляются ошибки предыдущих версий; кроме того, версии могут адаптироваться к новым аппаратным средствам и операционным системам. При этом в разработке и эксплуатации могут одновременно находиться сразу несколько версий. Поэтому нужно четко отслеживать все вносимые в систему изменения.

Существует много причин, объясняющих наличие разных конфигураций одной и той же системы. Различные версии создаются для разных компьютеров или операционных систем, включающих специальные функции, нужные заказчикам, и т.д. (рис. 29.1).



# 4.1. Планирование управления конфигурацией

В плане управления конфигурацией представлены стандарты, процедуры и мероприятия, необходимые для управления. Отправной точкой создания такого плана является набор общих стандартов по управлению конфигурацией, применяемых в организации разработчике ПО, которые адаптируются к каждому отдельному проекту. Обычно план управления конфигурацией имеет несколько разделов.

1. Определение контролируемых объектов, подпадающих под управление конфигурацией, а также формальная схема определения этих объектов.

2. Перечень лиц, ответственных за управление конфигурацией и за поставку контролируемых объектов в команду по управлению конфигурацией.

3. Политика ведения управления конфигурацией, т.е. процедуры управления изменениями и версиями.

4. Описание форм записей о самом процессе управления конфигурацией.

5. Описание средств поддержки процесса управления конфигурацией и способов их использования.

6. Определение базы данных конфигураций, применяемой для хранения всей информации о конфигурациях системы.

Распределение обязанностей по конкретным исполнителям является важной частью плана. Необходимо четко определить ответственных за поставку каждого документа или компонента ПО для команд по управлению качеством и конфигурацией.

# 4.2. Управление версиями и выпусками

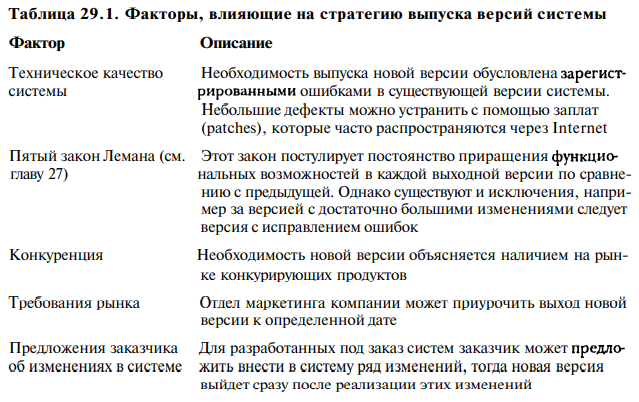
Управление версиями и выпусками ПО необходимо для идентификации и слежения за всеми версиями и выпусками системы. Менеджеры, отвечающие за управление версиями и выпусками ПО, разрабатывают процедуры поиска нужных версий системы и следят за тем, чтобы изменения не осуществлялись произвольно. Они также работают с заказчиками и планируют время выпуска следующих версий системы. Над новыми версиями системы должна работать команда по управлению конфигурацией, а не разработчики, даже если новые версии предназначены только для внутреннего использования. Только в том случае, если информация об изменениях в версиях вносится исключительно командой по управлению конфигурацией, можно гарантировать согласованность версий.

Версией системы называют экземпляр системы, имеющий определенные отличия от других экземпляров этой же системы. Новые версии могут отличаться функциональными возможностями, эффективностью или исправлениями ошибок. Некоторые версии имеют одинаковую функциональность, однако разработаны под различные конфигурации аппаратного или программного обеспечения. Если отличия между версиями незначительны, они называются вариантами одной версии.

Выходная версия (release) системы — это та версия, которая поставляется заказчику. В каждой выходной версии либо обязательно присутствуют новые функциональные возможности, либо она разработана под новую платформу. Количество версий обычно намного превышает количество выходных версий, поскольку версии создаются в основном для внутреннего пользования и не поставляются заказчику.

В настоящее время для поддержки управления версиями разработано много разнообразных CASE-средств (см. раздел 29.5). С помощью этих средств осуществляется управление хранением каждой версии и контроль за допуском к компонентам системы. Компоненты могут извлекаться из системы для внесения в них изменений. После введения в систему измененных компонентов получается новая версия, для которой с помощью системы управления версиями создается новое имя.

Принятие решения о том, когда именно должна выйти следующая выходная версия системы, существенно зависит от технических и общих организационных факторов, которые описаны в табл. 29.1



# 4.3. Сборка системы

Сборкой системы называют процесс компиляции и связывания программных компонентов в единую исполняемую программу. Перед сборкой системы полезно ответить на следующие вопросы.

1. Все ли компоненты, составляющие систему, включены в инструкцию по сборке?

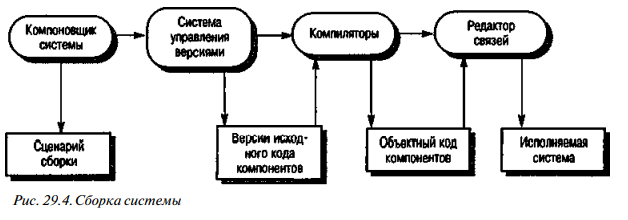
2. Каковы версии компонента, перечисленные в инструкции по сборке?

3. Доступны ли все необходимые файлы данных?

4. Если на файлы данных используются ссылки внутри компонентов, то каковы имена этих файлов в выходной версии?

5. Доступны ли нужные версии компилятора и других необходимых средств? Действующие версии программных средств могут быть несовместимы с более старыми версиями, которые применялись при разработке системы.

В настоящее время существует много средств управления конфигурацией, автоматизирующих процесс сборки системы. Команда управления конфигурацией пишет сценарий, в котором определены зависимости между различными компонентами системы. В нем также указаны средства компилирования и связывания компонентов системы. Средства компоновки интерпретируют сценарий сборки системы и вызывают программы, необходимые для сборки исполняемой системы. Процесс сборки системы представлен на рис. 29.4



# 4.4. CASE-средства для управления конфигурацией

Процесс управления конфигурацией обычно стандартизирован и включает выполнение заранее определенных процедур. Они требуют детализированного контроля за очень большим количеством данных. При сборке системы единственная ошибка в управлении может привести к некорректной работе системы. Поэтому очень важна поддержка процесса управления конфигурацией соответствующими CASE-средствами. Начиная с 70-х годов было разработано большое количество программных средств поддержки разных аспектов процесса управления конфигурацией.

Процесс управления изменениями заключается в заполнении форм запросов на изменения, проведении анализа изменений и передаче этих форм и соответствующих конфигурационных элементов команде управления качеством и команде по управлению конфигурацией. Этот алгоритмический по своей природе процесс позволяет сравнительно легко интегрировать его с системой управления версиями, поскольку, упрощая, можно сказать, что задача управления изменениями заключается в передаче нужных документов нужным людям в нужное время.

Поэтому для поддержки процесса управления изменениями достаточно следующих средств.

1. Редактор форм, позволяющий создавать и заполнять формы запросов на изменения.

2. Система автоматизации документооборота, которая позволяет фиксировать закрепление обработки форм запросов на изменения за членами команды по управлению конфигурацией и определяет порядок этой обработки. Эта система может также автоматизировать процесс передачи заполненных форм "нужным людям в нужное время" и информировать о состоянии процесса внесения изменений. Как правило, эта система использует электронную почту для пересылки сообщений.

3. База данных изменений, которая используется для хранения всех предложенных изменений и может быть связана с системой управления версиями.

Управление версиями предполагает обработку больших массивов информации для регистрации изменений, вносимых в систему, и контроля за ними. Средства управления версиями обязательно включают репозиторий конфигурационных элементов, которые в дальнейшем не изменяются. Если необходимо изменить какой-либо конфигурационный элемент, находящийся в репозиторий, его копия помещается в рабочий каталог. После изменений новая версия элемента также помещается в репозиторий. Системы управления версиями могут отличаться друг от друга, но все они имеют базовый набор средств.

Сборка систем — это очень трудоемкий вычислительный процесс. Например, процесс компиляции большой системы, состоящей из сотен компонентов, может занять несколько часов. Если компиляцию и связывание компонентов такой системы выполнять вручную, то оператор неизбежно сделает какие-либо ошибки. Средства сборки систем автоматизируют этот процесс, что исключает потенциальные ошибки, совершаемые при ручном компилировании, и, возможно, сокращает время сборки системы.