МИНИСТЕРСТВО НАУКИ И ОБРАЗОВАНИЯ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО
ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
УРАЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ЛЕСОТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ

Кафедра информационных технологий и моделирования

О.А. Карасева

Программная инженерия

Курс лекций

направления 230700.62- Прикладная информатика

ЕКАТЕРИНБУРГ

2012

Введение

ПРОГРАММНАЯ ИНЖЕНЕРИЯ КАК СОВОКУПНОСТЬ ИНЖЕНЕРНЫХ МЕТОДОВ И СРЕДСТВ СОЗДАНИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Программная инженерия. Понятие модели архитектуры ПО. Особенности современных крупных проектов ЭИС.

Проектирование экономических информационных систем – логически сложная, трудоемкая и длительная работа, требующая высокой квалификации участвующих в ней специалистов. Однако до настоящего времени проектирование ИЭС нередко выполняется на интуитивном уровне неформализованными методами, включающими в себя элементы искусства, практический опыт, экспертные оценки и дорогостоящие экспериментальные проверки качества функционирования ИЭС. Кроме того, в процессе создания и функционирования ИЭС информационные потребности пользователей постоянно изменяются и уточняются, что еще более усложняет разработку и сопровождение таких систем.

Основная доля трудозатрат при создании ИЭС приходится на прикладное программирование и базы данных. Производство ПО – это крупнейшая отрасль мировой экономики, в которой занято около трех млн. специалистов.

Потребность контролировать процесс разработки ПО, прогнозировать и гарантировать стоимость разработки, сроки и качество результатов привела в конце 70-х годов к необходимости перехода от кустарных к индустриальным способам создания ПО и появлению совокупности инженерных методов и средств создания ПО, объединенных общим названием «***программная инженерия***». Впервые этот термин был использован как тема конференции, проводившейся под эгидой НАТО в 1968 г. Спустя 7 лет, в 1975г. в Вашингтоне была проведена первая международная конференция, посвященная программной инженерии.

В процессе становления и развития программной инженерии можно выделить два этапа: 70-е и 80-е годы - систематизация и стандартизация процессов создания ПО (на основе структурного подхода) и 90-е годы - начало перехода к сборочному, индустриальному способу создания ПО (на основе объектно-ориентированного подхода).

В основе программной инженерии лежит одна фундаментальная идея: ***проектирование ПО является формальным процессом, который можно изучать и совершенствовать.*** Освоение и правильное применение методов и средств создания ПО позволяет повысить качество ЭИС, обеспечить управляемость процесса проектирования ЭИС и увеличить срок ее жизни.

Тенденции развития современных информационных технологий определяют постоянное возрастание сложности ПО ЭИС. Современные крупные проекты ЭИС характеризуют, как правило, следующие особенности:

- Сложность описания (достаточно большое количество функций, процессов, элементов данных и сложные взаимосвязи между ними), требующая тщательного

моделирования и анализа данных и процессов.

- Наличие совокупности тесно связанных подсистем, имеющих локальные задачи и цели функционирования.
- Отсутствие полных аналогов, ограничивающее возможность использования каких-либо типовых проектных решений и прикладных систем.
- Необходимость интеграции существующих и вновь разрабатываемых приложений.
- Функционирование в неоднородной среде на нескольких аппаратных платформах.
- Разобщенность и разнородность отдельных групп разработчиков по уровню квалификации и сложившимся традициям использования тех или иных инструментальных средств.
- Значительная временная протяженность проекта, обусловленная с одной стороны, ограниченными возможностями коллектива разработчиков и различной степенью готовности отдельных ее подразделений к внедрению ЭИС.

Для успешной реализации проекта объект проектирования (ПО ЭИС) должен быть прежде всего адекватно описан, т.е. должны быть построены полные и непротиворечивые ***модели архитектуры ПО***, включающие совокупность структурных элементов системы и связей между ними, поведение элементов системы в процессе их взаимодействия, а также иерархию подсистем, объединяющих структурные элементы.

Под *моделью* понимается полное описание системы ПО с определенной точки зрения. Модели представляют собой средства для визуализации, описания, проектирования и документирования архитектуры системы.

Моделирование является центральным звеном всей деятельности по созданию качественного ПО. Модели строятся для того, чтобы понять и осмыслить структуру и поведение будущей системы, облегчить управление процессом ее создания и уменьшить возможный риск, а также документировать принимаемые проектные решения.

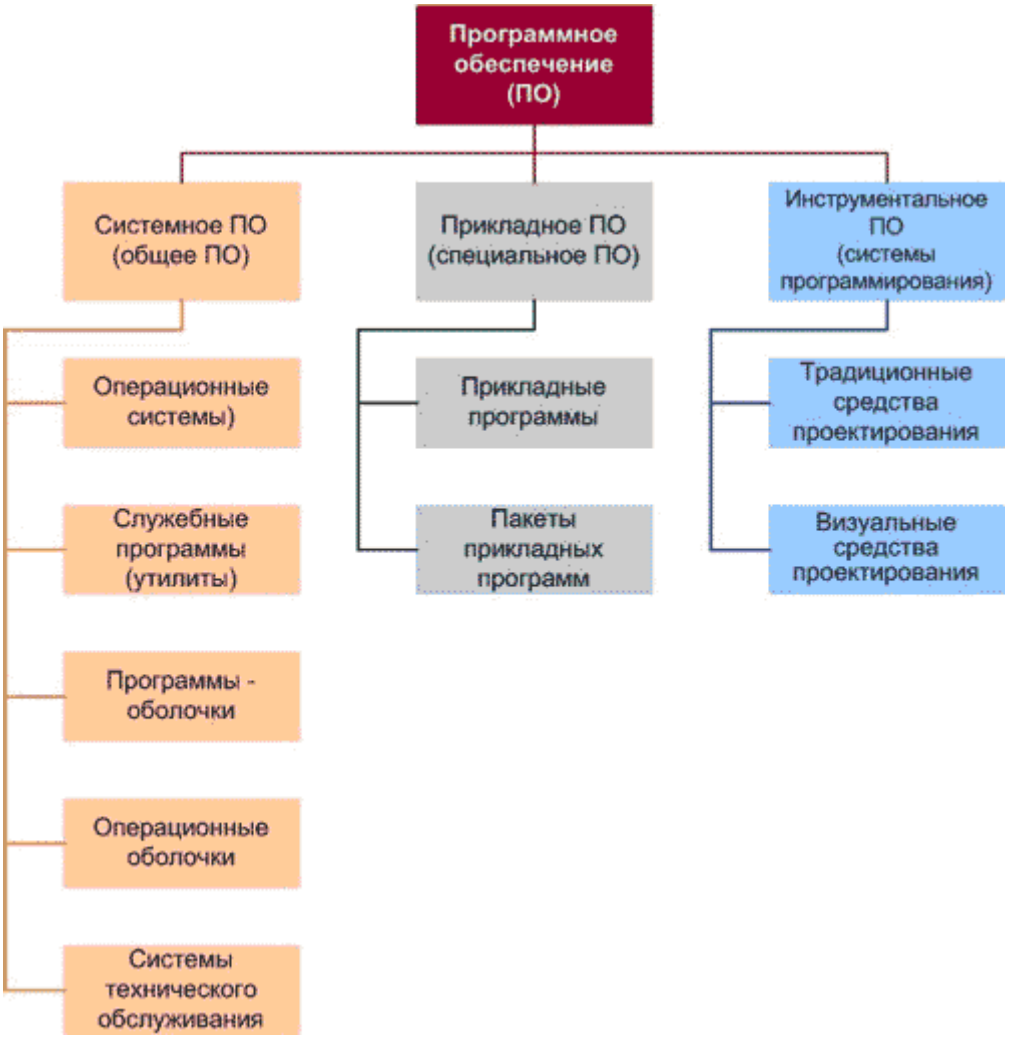
Разработка модели архитектуры системы ПО промышленного характера на стадии, предшествующей ее реализации или обновлению, также необходима, как и наличие проекта для строительства большого здания.

Тема 1. Программное обеспечение

Структура программного обеспечения ПК

Совокупность программ, предназначенная для решения задач на ПК, называется программным обеспечением. Состав программного обеспечения ПК называют программной конфигурацией.

Программное обеспечение, можно условно разделить на три категории:
 системное ПО (программы общего пользования), выполняющие различные вспомогательные функции, например создание копий используемой информации, выдачу справочной информации о компьютере, проверку работоспособности устройств компьютера и т.д.
 прикладное ПО, обеспечивающее выполнение необходимых работ на ПК: редактирование текстовых документов, создание рисунков или картинок, обработка информационных массивов и т.д.
 инструментальное ПО (системы программирования), обеспечивающее разработку новых программ для компьютера на языке программирования.



Системное ПО

Это программы общего пользования не связаны с конкретным применением ПК и выполняют традиционные функции: планирование и управление задачами, управления вводом-выводом и т.д.

Другими словами, системные программы выполняют различные

вспомогательные функции, например, создание копий используемой информации, выдачу справочной информации о компьютере, проверку работоспособности устройств компьютера и т.п.

К системному ПО относятся:

- операционные системы (эта программа загружается в ОЗУ при включении компьютера)
- программы – оболочки (обеспечивают более удобный и наглядный способ общения с компьютером, чем с помощью командной строки DOS, например, Norton Commander)
- операционные оболочки – интерфейсные системы, которые используются для создания графических интерфейсов, мультипрограммирования и.т.
- Драйверы (программы, предназначенные для управления портами периферийных устройств, обычно загружаются в оперативную память при запуске компьютера)
- утилиты (вспомогательные или служебные программы, которые представляют пользователю ряд дополнительных услуг)

К утилитам относятся:

- диспетчеры файлов или файловые менеджеры
- средства динамического сжатия данных (позволяют увеличить количество информации на диске за счет ее динамического сжатия)
- средства просмотра и воспроизведения
- средства диагностики; средства контроля позволяют проверить конфигурацию компьютера и проверить работоспособность устройств компьютера, прежде всего жестких дисков
- средства коммуникаций (коммуникационные программы) предназначены для организации обмена информацией между компьютерами
- средства обеспечения компьютерной безопасности (резервное копирование, антивирусное ПО).

Необходимо отметить, что часть утилит входит в состав операционной системы, а другая часть функционирует автономно. Большая часть общего (системного) ПО входит в состав ОС. Часть общего ПО входит в состав самого компьютера (часть программ ОС и контролирующих тестов записана в ПЗУ или ППЗУ, установленных на системной плате). Часть общего ПО относится к автономными программам и поставляется отдельно.

Прикладное ПО

Прикладные программы могут использоваться автономно или в составе программных комплексов или пакетов. Прикладное ПО – программы, непосредственно обеспечивающие выполнение необходимых работ на ПК: редактирование текстовых документов, создание рисунков или

картинок, создание электронных таблиц и т.д.

Пакеты прикладных программ – это система программ, которые по сфере применения делятся на проблемно – ориентированные, пакеты общего назначения и интегрированные пакеты. Современные интегрированные пакеты содержат до пяти функциональных компонентов: тестовый и табличный процессор, СУБД, графический редактор, телекоммуникационные средства.

К прикладному ПО, например, относятся:

- Комплект офисных приложений MS OFFICE
- Бухгалтерские системы
- Финансовые аналитические системы
- Интегрированные пакеты делопроизводства
- CAD – системы (системы автоматизированного проектирования)
- Редакторы HTML или Web – редакторы
- Браузеры – средства просмотра Web - страниц
- Графические редакторы
- Экспертные системы

И так далее.

Инструментальное ПО

Инструментальное ПО или системы программирования - это системы для автоматизации разработки новых программ на языке программирования.

В самом общем случае для создания программы на выбранном языке программирования (языке системного программирования) нужно иметь следующие компоненты:
1. Текстовый редактор для создания файла с исходным текстом программы.
2. Компилятор или интерпретатор. Исходный текст с помощью программы-компилятора переводится в промежуточный объектный код. Исходный текст большой программы состоит из нескольких ***модулей*** (файлов с исходными текстами). Каждый модуль компилируется в отдельный файл с объектным кодом, которые затем надо объединить в одно целое.
3. Редактор связей или сборщик, который выполняет связывание объектных модулей и формирует на выходе работоспособное приложение – исполнимый код.
Исполнимый код – это законченная программа, которую можно запустить на любом компьютере, где установлена операционная система, для которой эта программа создавалась. Как правило, итоговый файл имеет расширение .EXE или .COM.
4. В последнее время получили распространение визуальный методы программирования (с помощью языков описания сценариев),

ориентированные на создание Windows-приложений. Этот процесс автоматизирован в средах быстрого проектирования. При этом используются готовые визуальные компоненты, которые настраиваются с помощью специальных редакторов.

Наиболее популярные редакторы (системы программирования программ с использованием визуальных средств) визуального проектирования:

- Borland Delphi - предназначен для решения практически любых задачи прикладного программирования
- Borland C++ Builder – это отличное средство для разработки DOS и Windows приложений
- Microsoft Visual Basic – это популярный инструмент для создания Windows-программ
- Microsoft Visual C++ - это средство позволяет разрабатывать любые приложения, выполняющиеся в среде ОС типа Microsoft Windows

Тема 2-3. ЖИЗНЕННЫЙ ЦИКЛ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Понятие ЖЦ ПО. Международный стандарт ISO/IEC 12207: 1995. Основные и вспомогательные процессы ЖЦ ПО. Организация процессов ЖЦ. Связь между процессами.

ПОНЯТИЕ ЖЦ

Жизненный цикл (ЖЦ) программного обеспечения (ПО) определяется как период времени, который начинается с момента принятия решения о необходимости создания ПО и заканчивается в момент его полного изъятия из эксплуатации.

Основным нормативным документом, регламентирующим состав процессов ЖЦ ПО, является международный стандарт ISO/IEC 12207: 1995 “Information Technology - Software Life Cycle Processes” (ISO - International Organization for Standardization - Международная организация по стандартизации, IEC - International Electrotechnical Commission - Международная комиссия по электротехнике. Он определяет структуру ЖЦ, содержащую процессы, действия и задачи, которые должны быть выполнены во время создания ПО.

В данном стандарте ***ПО (или программный продукт)*** определяется как набор компьютерных программ, процедур и, возможно, связанной с ними документации и данных.

Процесс определяется как совокупность взаимосвязанных действий, преобразующих некоторые входные данные в выходные. Каждый процесс характеризуется определенными задачами и методами их решения, исходными данными, полученными от других процессов, и результатами.

Каждый процесс разделен на набор действий, каждое действие – на набор

задач. Каждый процесс, действие или задача инициируется и выполняется другим процессом по мере необходимости, причем не существует заранее определенных последовательностей выполнения (естественно, при сохранении связей по входным данным).

В России существуют стандарты:

ГОСТ 34601 - 90. «Информационная технология. Комплекс стандартов на автоматизированные системы. Автоматизированные системы. Стадии создания».

ГОСТ 34601 - 89. «Информационная технология. Комплекс стандартов на автоматизированные системы. Техническое задание на создание автоматизированной системы».

ГОСТ 34601 - 92. «Информационная технология. Виды испытаний автоматизированных систем».

Однако процессы создания программного обеспечения для современных распределенных ЭИС, функционирующих в неоднородной среде, в этих стандартах отражены недостаточно, а отдельные их положения явно устарели. Поэтому в отечественных разработках целесообразно использовать современные международные стандарты.

В соответствии с ISO/IEC 12207: 1995 все процессы ЖЦ ПО разделены на три группы:

Основные процессы:

- приобретение;
- поставка;
- разработка;
- эксплуатация;
- сопровождение.

Вспомогательные процессы:

- документирование;
- управление конфигурацией;
- обеспечение качества;
- верификация;
- аттестация;
- совместная оценка;
- аудит;
- разрешение проблем.

Организационные процессы:

- управление;
- усовершенствование;
- создание инфраструктуры;
- обучение.

ОСНОВНЫЕ ПРОЦЕССЫ

Процесс приобретения состоит из действий и задач заказчика:

Действие - инициирование приобретения - включает задачи:

- определение заказчиком своих потребностей в приобретении;
- анализ требований к системе;
- принятие решения относительно приобретения;
- проверку наличия необходимой документации, гарантий, сертификатов, лицензий и поддержки в случае приобретения ПО;
- подготовку и утверждение плана приобретения, включающего требования к системе, тип договора, ответственность сторон.

Действие – подготовка заявочных предложений. Заявочные предложения должны содержать:

- требования к системе;
- перечень программных продуктов;
- условия и соглашения;
- технические ограничения (например, среда функционирования системы).

Заявочные предложения направляются выбранному поставщику. Поставщик – это организация, которая заключает договор с заказчиком на поставку системы, ПО или программной услуги на условиях, оговоренных в договоре.

Действие - подготовка и корректировка договора - включает задачи:

- определение заказчиком процедуры выбора поставщика, включающей критерии оценки предложений возможных поставщиков;
- выбор конкретного поставщика на основе анализа предложений.;
- подготовку и заключение договора с поставщиком;
- внесение изменений (при необходимости) в договор в процессе его выполнения.

Действие - надзор за деятельностью поставщика - осуществляется в соответствии с действиями, предусмотренными в процессах совместной оценки и аудита**.**

В процессе ***приемки*** подготавливаются и выполняются необходимые тесты. ***Завершение работ*** по договору осуществляется в случае удовлетворения всех условий приемки.

Процесс поставки охватывает действия и задачи, выполняемые поставщиком, который снабжает заказчика программным продуктом или услугой. Данный процесс включает действия:

Инициирование поставки заключается в рассмотрении поставщиком заявочных предложений и принятии решения согласиться с выставленными требованиями и условиями или предложить свои.

Планирование включает задачи:

- принятие решения поставщиком относительно выполнения работ своими силами или с привлечением субподрядчика;
- разработку поставщиком плана управления проектом, содержащего организационную структуру проекта, разграничение ответственности, технические требования к среде разработки и ресурсам, управление субподрядчиком.

Процесс разработки предусматривает действия и задачи, выполняемые

разработчиком, и включает следующие действия:

Подготовительная работа начинается с выбора модели ЖЦ ПО, соответствующей масштабу, значимости и сложности проекта. Действия и задачи процесса должны соответствовать выбранной модели. Разработчик должен выбрать, адаптировать к условиям проекта и использовать согласованные с заказчиком стандарты, методы и средства разработки, а также составить план выполнения работ.

Анализ требований к системе подразумевает определение ее функциональных возможностей, пользовательских требований, требований к надежности и безопасности, требований к внешним интерфейсам и т.д. Требования с системе оцениваются исходя из критериев реализуемости и возможности проверки при тестировании.

Проектирование архитектуры системы на высоком уровне заключается в определении компонентов ее оборудования, ПО и операций, выполняемых эксплуатирующим систему персоналом. Архитектура системы должна соответствовать требованиям, предъявляемым к системе, а также принятым проектным стандартам и методам.

Анализ требований к ПО предполагает определение следующих характеристик для каждого компонента ПО:

- функциональных возможностей, включая характеристики производительности и среды функционирования компонента;
- внешних интерфейсов;
- спецификаций надежности и безопасности;
- эргономических требований;
- требований к используемым данным;
- требований к установке и приемке;
- требований к пользовательской документации;
- требований к эксплуатации и сопровождению.

Требования к ПО оцениваются исходя из критериев соответствия требованиям к системе, реализуемости и возможности проверки при тестировании.

Проектирование архитектуры ПО включает задачи (для каждого компонента ПО):

- трансформацию требований к ПО в архитектуру, определяющую на высоком уровне структуру ПО и состав ее компонентов;
- разработку и документирование программных интерфейсов ПО и баз данных;
- разработку предварительной версии пользовательской документации;
- разработку и документирование предварительных требований к тестам и планам интеграции ПО.

Архитектура компонентов ПО должна соответствовать требованиям, предъявляемым к ним, а также принятым проектным стандартам и методам.

Детальное проектирование ПО включает следующие задачи:

- описание компонентов и интерфейсов между ними на более низком уровне, достаточном для их последующего самостоятельного кодирования и

тестирования;

- разработку и документирование детального проекта базы данных;
- обновление (при необходимости) пользовательской документации;
- разработку и документирование требований к тестам и плана тестирования компонентов ПО;
- обновление плана интеграции ПО.

Кодирование и тестирование ПО охватывает задачи:

- разработку и документирование каждого компонента ПО и базы данных а также совокупности тестовых процедур и данных для их тестирования;
- тестирование каждого компонента ПО и базы данных на соответствие предъявляемых к ним требованиям. Результаты тестирования компонентов должны быть документированы;
- обновление (при необходимости) пользовательской документации;
- обновление плана интеграции ПО.

Интеграция ПО предусматривает сборку разработанных компонентов ПО в соответствии с планом интеграции и тестирование агрегированных компонентов. Для каждого из агрегированных компонентов разрабатываются наборы тестов и тестовые процедуры, предназначенные для проверки каждого из квалификационных требований при последующем квалификационном тестировании. ***Квалификационное тестирование*** -это набор критериев и условий, которые необходимо выполнить, чтобы квалифицировать программный продукт как соответствующий своим спецификациям и готовый к использованию в условиях эксплуатации.

Квалификационное тестирование ПО проводится разработчиком в присутствии заказчика (по возможности) для демонстрации того, что ПО удовлетворяет своим спецификациям и готово к использованию в условиях эксплуатации. Квалификационное тестирование выполняется для каждого компонента ПО по всем разделам требований при широком варьировании тестов. При этом также проверяются полнота технической и пользовательской документации и ее адекватность самим компонентам ПО.

Интеграция системы заключается в сборке всех ее компонентов, включая ПО и оборудование. После интеграции система, в свою очередь, подвергается квалификационному тестированию на соответствие совокупности требований к ней. При этом также производится оформление и проверка полного комплекта документации на систему.

Установка ПО осуществляется разработчиком в соответствии с планом в той среде и на том оборудовании, которые предусмотрены договором. В процессе установки проверяется работоспособность ПО и баз данных. Если устанавливаемое программное обеспечение заменяет существующую систему, разработчик должен обеспечить их параллельное функционирование в соответствии с договором.

Приемка ПО предусматривает оценку результатов квалификационного тестирования ПО и системы и документирование результатов оценки, которые проводятся заказчиком с помощью разработчика. Разработчик выполняет

окончательную передачу ПО заказчику в соответствии с договором, обеспечивая при этом необходимое обучение и поддержку.

Процесс эксплуатации охватывает действия и задачи оператора – организации, эксплуатирующей систему и включает действия:

Подготовительная работа включает проведение оператором следующих задач:

- планирование действий и работ, выполняемых в процессе эксплуатации, и установку эксплуатационных стандартов;
- определение процедур локализации и разрешения проблем, возникающих в процессе эксплуатации.

Эксплуатационное тестирование осуществляется для каждой очередной редакции программного продукта, после чего она передается в эксплуатацию.

Эксплуатация системы выполняется в предназначенной для этого среде в соответствии с пользовательской документацией.

Поддержка пользователей заключается в оказании помощи и консультаций при обнаружении ошибок в процессе эксплуатации ПО.

Процесс сопровождения предусматривает действия и задачи, выполняемые службой сопровождения. В соответствии со стандартом IEEE-90 под ***сопровождением*** понимается внесение изменений в ПО в целях исправления ошибок, повышения производительности или адаптации к изменившимся условиям работы или требованиям.

Изменения, вносимые в существующее программное обеспечение, не должны нарушать его целостность. Процесс сопровождения включает перенос ПО в другую среду (миграцию) и заканчивается снятием ПО с эксплуатации.

Процесс сопровождения охватывает следующие действия:

Подготовительная работа службы сопровождения включает в себя следующие задачи:

- планирование действий и работ, выполняемых в процессе сопровождения;
- определение процедур локализации и разрешения проблем, возникающих в процессе сопровождения.

Анализ проблем и запросов на модификацию ПО, выполняемый службой сопровождения, включает следующие задачи:

- анализ сообщения о возникшей проблеме или запроса на модификацию ПО относительно его влияния на организацию, существующую системы и интерфейсы с другими системами. При этом определяются следующие характеристики возможной модификации: тип (корректирующая, улучшающая, профилактическая или адаптирующая к новой среде); масштаб (размеры модификации, стоимость и время ее реализации); критичность (воздействие на производительность, надежность или безопасность);
- оценка целесообразности проведения модификации и возможных вариантов ее проведения);
- утверждение выбранного варианта модификации.

Модификация ПО предусматривает определение компонентов ПО, их версий

и документации, подлежащих модификации, и внесение необходимых изменений в соответствии с правилами процесса разработки. Подготовленные изменения тестируются и проверяются по критериям, определенным в документации. При подтверждении корректности изменений в программах производится корректировка документации.

Проверка и приемка заключается в проверке целостности модифицированной системы и утверждении внесенных изменений.

При переносе ПО в другую среду используются имеющиеся или разрабатываются новые средства переноса, затем выполняется конвертирование программ и данных в новую среду. С целью облегчить переход предусматривается параллельная эксплуатация ПО в старой и новой среде в течение некоторого периода, когда проводится необходимое обучение пользователей в новой среде.

Снятие ПО с эксплуатации осуществляется по решению заказчика при участии эксплуатирующей организации, службы сопровождения и пользователей. При этом программные продукты и соответствующая документация подлежат архивированию в соответствии с договором.

ВСПОМОГАТЕЛЬНЫЕ ПРОЦЕССЫ ЖЦ ПО

Процесс документирования предусматривает формализованное описание информации, созданной в течение ЖЦ ПО. Данный процесс состоит из набора действий, с помощью которых планируют, проектируют, разрабатывают, выпускают, редактируют, распространяют и сопровождают документы, необходимые для всех заинтересованных лиц, таких, как руководство, технические специалисты и пользователи системы.

Процесс документирования включает действия:

- подготовительную работу;
- проектирование и разработку;
- выпуск документации;
- сопровождение.

Процесс управления конфигурацией предполагает применение административных и технических процедур на всем протяжении ЖЦ ПО для определения состояния компонентов ПО в системе, управления модификациями ПО, описания и подготовки отчетов о состоянии компонентов ПО и запросов на модификацию, обеспечения полноты, совместимости и корректности ПО, управления хранением и поставкой ПО. Согласно стандарте IEEE - 90 под ***конфигурацией*** ПО понимается совокупность ее функциональных и физических характеристик, установленных в технической документации и реализованных в ПО.

Управление конфигурацией позволяет организовать, систематически учитывать и контролировать внесение изменений в ПО на всех стадиях ЖЦ ПО. Общие принципы и рекомендации по управлению конфигурацией ПО отражены в проекте стандарта ISO/IEC 12207-2: 1995 “Information Technology - Software Life Cycle Processes. Part2. Configuration Management for Software”.

Процесс управления конфигурацией включает действия:

- ***подготовительную работу*** (планирование управления конфигурацией);
- ***идентификацию конфигурации*** (устанавливает правила, с помощью которых можно однозначно идентифицировать и различать компоненты ПО и их версии). Кроме того, каждому компоненту и его версиям соответствует однозначно обозначаемый комплект документации. В результате создается база для однозначного выбора и манипулирования версиями компонентов ПО, использующая ограниченную и упорядоченную систему символов, идентифицирующих различные версии ПО.
- ***контроль конфигурации*** (предназначен для систематической оценки предполагаемых модификаций ПО и координированной их реализации с учетом эффективности каждой модификации и затрат на ее выполнение). Он обеспечивает адекватность реально изменяющихся компонентов и их комплектной документации;
- ***учет состояния конфигурации*** (представляет собой регистрацию состояния компонентов ПО, подготовку отчетов обо всех реализованных и отвергнутых модификациях версий компонентов ПО). Совокупность отчетов обеспечивает однозначное отражение текущего состояния системы и ее компонентов, а также ведение истории модификаций;
- ***оценку конфигурации*** (заключается в оценке функциональной полноты компонентов ПО, а также соответствия их физического состояния текущему техническому описанию);
- ***управление выпуском и поставку*** (охватывают изготовление эталонных копий программ и документации, их хранение и поставку пользователям в соответствии с порядком, принятым в организации).

Процесс обеспечения качества обеспечивает соответствующие гарантии того, что ПО и процессы его ЖЦ соответствуют заданным требованиям и утвержденным планам. Под ***качеством ПО*** понимается совокупность свойств, которые характеризуют способность ПО удовлетворять заданным требованиям.

Для получения достоверных оценок создаваемого ПО процесс обеспечения его качества должен происходить независимо от субъектов, непосредственно связанных с разработкой ПО. При этом могут использоваться результаты других вспомогательных процессов, таких, как верификация, аттестация, совместная оценка, аудит и разрешение проблем.

Процесс обеспечения качества включает действия:

- ***подготовительная работу*** (заключается в координации с другими вспомогательными процессами и планировании самого процесса обеспечения качества с учетом используемых стандартов, методов, процедур и средств);
- ***обеспечение качества продукта*** подразумевает гарантирование полного соответствия программных продуктов и их документации требованиям заказчика, предусмотренным в договоре;
- ***обеспечение качества процесса*** предполагает гарантирование соответствия процессов ЖЦ ПО, методов разработки, среды разработки

и квалификации персонала условиям договора, установленным стандартам и процедурам;

- ***обеспечение прочих показателей качества системы*** осуществляется в соответствии с условиями договора и стандартом ISO 9001.

Процесс верификации состоит в определении того, что программные продукты, являющиеся результатами некоторого действия, полностью удовлетворяют требованиям или условиям, обусловленным предшествующими действиями (верификация в узком смысле означает формальное доказательство правильности ПО).

Верификация может проводится с различными степенями независимости. Степень независимости может варьироваться от выполнения верификации самим исполнителем или другим специалистом данной организации до ее выполнения специалистом другой организации с различными вариациями. Если процесс верификации осуществляется организацией, не зависящей от поставщика, разработчика, оператора или службы сопровождения, то он называется ***процессом независимой верификации***.

Процесс верификации включает следующие действия:

- подготовительную работу;
- верификацию;

В процесс верификации проверяются следующие условия:

– непротиворечивость требований к системе и степень учета потребностей пользователей;

– возможности поставщика выполнять заданные требования;

– соответствие выбранных процессов ЖЦ ПО условиям договора;

– адекватность стандартов, процедур и среды разработки процесса ЖЦ ПО;

– соответствие проектных спецификаций ПО заданным требованиям;

– корректность описания в проектных спецификациях входных и выходных данных, последовательности событий, интерфейсов, логики;

– соответствие кода проектным спецификациям и требованиям;

– тестируемость и корректность кода, его соответствие принятым стандартам кодирования;

– корректность интеграции компонентов ПО в систему;

– адекватность, полнота и непротиворечивость документации.

Процесс аттестации предусматривает определение полноты соответствия заданных требований и созданной системы или программного продукта их конечному функциональному назначению. Под ***аттестацией*** обычно понимается подтверждение и оценка достоверности проеденного тестирования. Аттестация должно гарантировать полное соответствие ПО спецификациям, требованиям и документации, а также возможность его безопасного и надежного применения пользователем. Аттестацию рекомендуется выполнять путем тестирования во всех возможных ситуациях и использовать при этом независимых специалистов. Аттестация может проводиться на начальных стадиях ЖЦ ПО или как часть работы

по приемке ПО.

Аттестация, так же как и верификация, может осуществляться с различными степенями независимости. Если процесс аттестации выполняется организацией, не зависящей от поставщика, разработчика, оператора или службы сопровождения, то он называется ***процессом независимой аттестации*.**

Процесс совместной оценки предназначен для оценки состояния работ по проекту и ПО. Он сосредоточен в основном на контроле планирования и управления ресурсами, персоналом, аппаратурой и инструментальными средствами проекта.

Оценка применяется как на уровне управления проектом, так и на уровне технической реализации проекта и проводится в течение всего срока договора. Данный процесс может выполняться двумя любыми сторонами, участвующими в договоре, при этом одна сторона проверяет другую.

Процесс совместной оценки включает действия:

- подготовительную работу;
- оценку управления проектом;
- техническую оценку.

Процесс аудита представляет собой определение соответствия требованиям, планам и условиям договора. Аудит может выполняться двумя любыми сторонами, участвующими в договоре, когда одна сторона проверяет другую.

Аудит – это ревизия (проверка), проводимая компетентным органом (лицом) в целях обеспечения независимой оценки степени соответствия ПО или процессов установленным требованиям. Аудит служит для установления соответствия реальных работ и отчетов требованиям, планам и контракту. Аудиторы не должны иметь прямой зависимости от разработчиков ПО. Они определяют состояние работ, использование ресурсов, соответствие документации требованиям и стандартам, корректность тестирования.

Процесс разрешения проблем предусматривает анализ и решение проблем (включая обнаруженные несоответствия) независимо от их происхождения или источника, которые обнаружены в ходе разработки, эксплуатации, сопровождения или других процессов. Каждая обнаруженная проблема должна быть идентифицирована, описана, проанализирована и разрешена.

ОРГАНИЗАЦИОННЫЕ ПРОЦЕССЫ ЖЦ ПО

Процесс управления состоит из действий и задач, которые могут выполняться любой стороной, управляющей своими ресурсами. Данная сторона (менеджер) отвечает за управление выпуском продукта, управление проектом и управление задачами соответствующих процессов, таких, как приобретение, поставка, разработка, эксплуатация, сопровождение и т.д.

Процесс управления включает следующие действия:

- ***инициирование о определение области управления.*** Менеджер должен убедиться, что необходимые для управления ресурсы (персонал,

оборудование и технология) имеются в его распоряжении в достаточном количестве;

- ***планирование*** подразумевает выполнение, как минимум, следующих задач:
  - составление графиков выполнения работ;
  - оценку затрат;
  - выделение требуемых ресурсов;
  - распределение ответственности;
  - оценку рисков, связанных с конкретными задачами;
  - создание инфраструктуры управления.

Процесс создания инфраструктуры охватывает выбор и поддержку (сопровождение технологии), стандартов и инструментальных средств, выбор и установку аппаратных и программных средств, используемых для разработки, эксплуатации или сопровождения ПО. Инфраструктура должна модифицироваться и сопровождаться в соответствии с изменениями требований к соответствующим процессам. Инфраструктура, в свою очередь, является одним из объектов управления конфигурацией.

Процесс усовершенствования предусматривает оценку, измерение, контроль и усовершенствование процессов ЖЦ ПО.

Процесс обучения охватывает первоначальное обучение и последующее постоянное повышение квалификации персонала.

СВЯЗЬ МЕЖДУ ПРОЦЕССАМИ ЖЦ ПО

Процессы ЖЦ ПО, регламентированные стандартом ISO/IEC 12207, могут использоваться различными организациями в конкретных проектах самым различным образом. Тем не менее, стандарт предлагает некоторый базовый набор взаимосвязей между процессами с различных точек зрения (рис.1). Такими аспектами являются:

- договорный аспект;
- аспект управления;
- аспект эксплуатации;
- инженерный аспект;
- аспект поддержки.

В ***договорном аспекте*** заказчик и поставщик вступают в договорные отношения и реализуют соответственно процессы приобретения и поставки. В ***аспекте управления*** заказчик, поставщик, разработчик, оператор, служба сопровождения и другие участвующие в ЖЦ ПО стороны управляют выполнением своих процессов. В ***аспекте эксплуатации*** оператор, эксплуатирующий систему, предоставляет необходимые услуги пользователям. В ***инженерном аспекте*** разработчик или служба сопровождения решают соответствующие технические задачи, разрабатывая или модифицируя программные продукты. В ***аспекте поддержки*** службы, реализующие вспомогательные процессы, предоставляют необходимые услуги всем остальным участникам работ.

Взаимосвязи между процессами, описанные в стандарте, носят статистический характер. Более важные динамические связи между процессами и реализующими их сторонами устанавливаются в реальных проектах.

Тема 4. МОДЕЛИ И СТАДИИ ЖЦ ПО

Понятие модели ЖЦ ПО (каскадная, спиральная). Стадии: формирование требований к ПО; проектирование; реализация; тестирование; ввод в действие; эксплуатация и сопровождение; снятие с эксплуатации. Подход RAD. Модели качества процессов проектирования.

Под ***моделью*** ЖЦ ПО понимается структура, определяющая последовательность выполнения и взаимосвязи процессов, действий, задач на протяжении ЖЦ. Модель ЖЦ зависит от специфики, масштаба и сложности проекта и специфики условий, в которых система создается и функционирует.

Международный стандарт ISO/IEC 12207: 1995 не предлагает конкретную модель ЖЦ и методы разработки ПО. Его положения являются общими для любых моделей ЖЦ, методов и технологий разработки ПО. Стандарт описывает структуру процессов ЖЦ ПО, но не конкретизирует в деталях, как реализовать и выполнить действия и задачи, включенные в эти процессы.

Модель ЖЦ любого конкретного ПО ЭИС определяет характер процесса его создания, который представляет собой совокупность упорядоченных во времени, взаимосвязанных и объединенных в стадии работ, выполнение которых необходимо и достаточно для создания ПО, соответствующего заданным требованиям.

Под стадией создания ПО понимается часть процесса создания ПО, ограниченная некоторыми временными рамками, и заканчивающаяся выпуском какого-то конкретного продукта (моделей ПО, программных компонентов, документации), определяемого заданными для этой стадии требованиями. Стадии создания ПО выделяются по соображениям рационального планирования и организации работ, заканчивающихся заданными результатами. В состав ЖЦ ПО обычно включают следующие стации:

1. Формирование требований к ПО.
2. Проектирование.
3. Реализация.
4. Тестирование.
5. Ввод в действие.
6. Эксплуатация и сопровождение.
7. Снятие с эксплуатации.

Формирование требований к ПО является одной из важнейших, поскольку определяет успех всего проекта. Данная стадия включает этапы:

Планирование работ, предваряющее работы над проектом. Основными задачами являются:

- определение целей разработки;
- предварительная экономическая оценка проекта;
- построение плана – графика выполнения работ;

- создание и обучение совместной рабочей группы.

Проведение обследования деятельности автоматизируемого объекта (организации), в рамках которого осуществляются:

- предварительное выявление требований к будущей системе;
- определение структуры организации;
- определение перечня целевых функций организации;
- анализ распределения функций по подразделениям и сотрудникам;
- выявление функциональных взаимодействий между подразделениями, информационных потоков внутри подразделений и между ними, внешних по отношению к организации объектов и внешних информационных взаимодействий;
- анализ существующих средств автоматизации деятельности организации.

Построение моделей деятельности организаций, предусматривающее обработку материалов обследования и построение двух видов моделей:

- ***модели "AS-IS" («как есть»),*** отражающей существующее на момент обследования положение дел в организации и позволяющее понять, каким образом функционирует данная организация, а также выявить узкие места и сформулировать предложения по улучшению ситуации;
- ***модели "TO-BE" («как должно быть»)***, отражающей представление о новых технологиях работы организации.

Каждая из моделей включает в себя полную функциональную и информационную модель деятельности организации, а также, в случае необходимости, модель, описывающую динамику поведения организации.

Переход от модели ***"AS-IS"*** к модели ***"TO-BE"*** может выполняться двумя способами:

- совершенствованием существующих технологий на основе оценки их эффективности;
- радикальным изменением технологий и перепроектированием бизнес-процессов (реинжиниринг бизнес-процессов).

Построенные модели имеют самостоятельное практическое значение. Например, модель ***"AS-IS"*** позволяет выявить узкие места в существующих технологиях и предлагать рекомендации по решению проблем независимо от того, предполагается на данном этапе дальнейшая разработка ЭИС или нет. Кроме того, модель облегчает обучение сотрудников конкретным направлениям деятельности организации за счет использования наглядных диаграмм (известно, что «одна картинка стоит тысячи слов»).

Стадия проектирования включает этапы:

Разработка системного проекта. На этом этапе дается ответ на вопрос: «Что должна делать будущая система?», а именно: определяются архитектура системы, ее функции, внешние условия функционирования, интерфейсы и распределение функций между пользователями и системой, требования к программным и информационным компонентам, состав исполнителей и сроки разработки. Основу системного проекта составляют модели проектируемой ЭИС, которые строятся на

основе модели ***"ТО-ВЕ"*** . Документальным результатом является техническое задание.

Разработка технического проекта. На этом этапе на основе системного проекта осуществляется собственно проектирование системы, включающее проектирование архитектуры системы и детальное проектирование. Таким образом, дается ответ на вопрос: «Как построить систему, чтобы она удовлетворяла предъявленным к ней требованиям?». Модели проектируемой ЭИС при этом уточняются и детализируются до необходимого уровня.

К настоящему времени наибольшее распространение получили следующие две модели ЖЦ ПО: каскадная (1970-1985) и спиральная (1986-1990).

Принципиальной особенностью каскадного подхода (рис.1) является: ***переход на следующую стадию осуществляется только после того, как будет полностью завершена работа на текущей стадии, и возвратов на пройденные стадии не предусматривается***. Каждая стадия заканчивается получением некоторых результатов, которые служат в качестве исходных данных для следующей стадии.

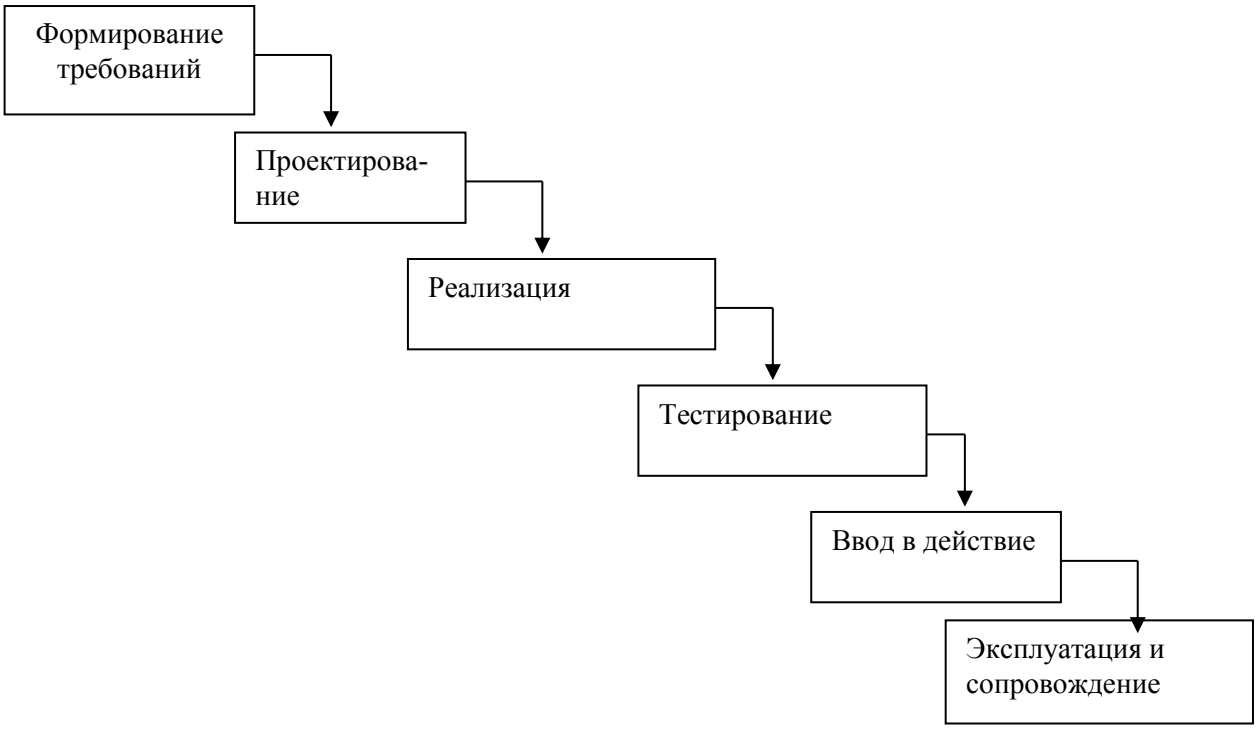


Рис.1. Каскадная модель

Каждая стадия завершается выпуском комплекта документации, достаточной для того, чтобы разработка могла быть продолжена другой командой разработчиков. Критерием качества разработки при таком подходе является точность выполнения спецификаций технического задания.

Преимущества применения каскадного способа:

- на каждой стадии формируется законченный набор проектной документации, отвечающий требованиям полноты и согласованности;

- выполняемые в логической последовательности стадии работ позволяют планировать сроки завершения всех работ и соответствующие затраты.

Каскадный подход хорошо зарекомендовал себя при построении ЭИС, для которых в самом начале разработки можно достаточно точно и полно сформулировать все требования, с тем, чтобы предоставить разработчикам свободу реализовать их технически как можно лучше.

В то же время этот подход обладает рядом недостатков, вызванных, прежде всего тем, что реальный процесс создания программного обеспечения никогда полностью не укладывается в такую жесткую схему. Процесс создания ПО носит, как правило, итерационный характер: результаты очередной стадии часто вызывают изменения в проектных решениях, выработанных на предыдущих стадиях. Таким образом, постоянно возникает потребность в возврате к предыдущим стадиям и уточнении или пересмотре ранее принятых решений. В результате реальный процесс создания ПО принимает иной вид (рис.2).

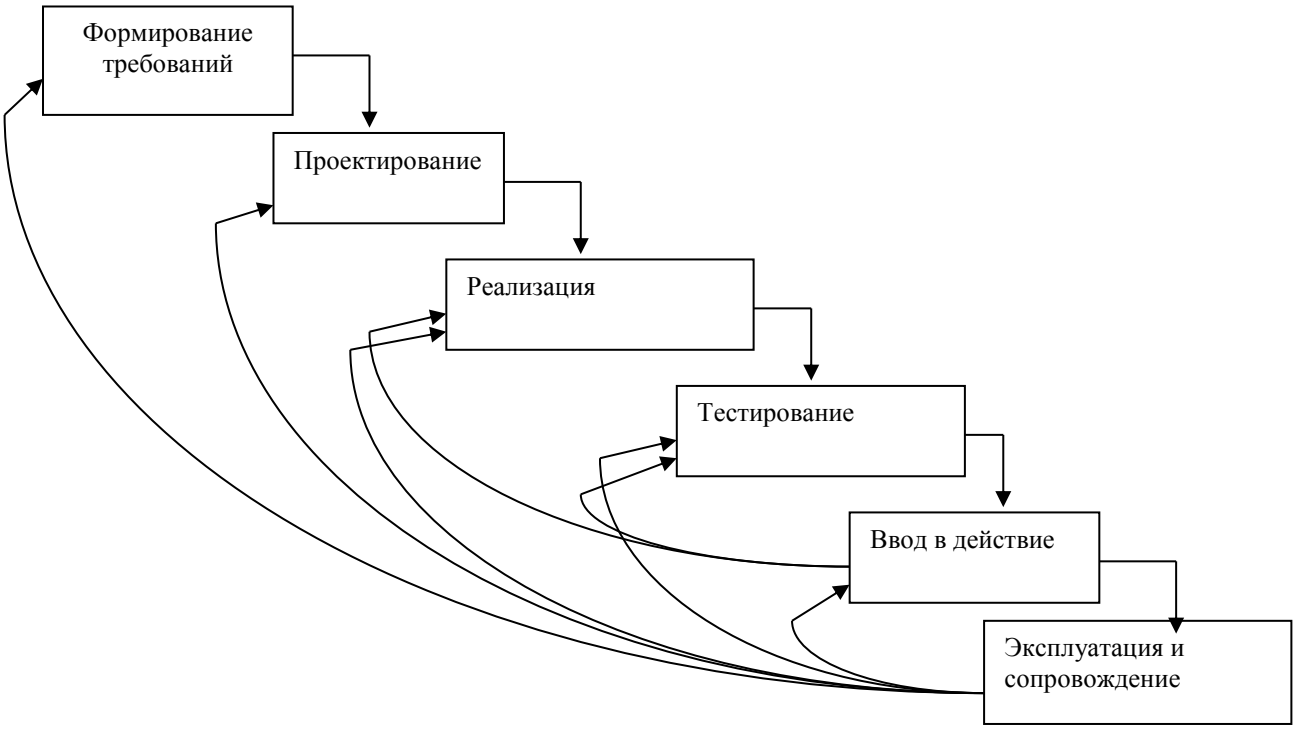


Рис. 2. Итерационный процесс создания программного обеспечения

Изображенную схему можно отнести к отдельной модели – модели с промежуточным контролем, в которой межстадийные корректировки обеспечивают большую надежность по сравнению с каскадной моделью, хотя увеличивают весь период разработки.

Основным недостатком каскадной модели является существенное запаздывание с получением результатов и, как следствие, высокий риск создания

системы, не удовлетворяющей и изменившимся потребностям пользователей. Это объяснятся двумя причинами:

- пользователи не в состоянии сразу изложить все свои требования и не могут предвидеть, как они изменятся в ходе разработки;
- за время разработки могут произойти изменения во внешней среде, которые повлияют на требования к системе.

В рамках каскадного подхода требования к ЭИС фиксируются в виде технического задания на все время ее создания, а согласование получаемых результатов с пользователями производится только в точках, планируемых после завершения каждой стадии (при этом возможна корректировка результатов по замечаниям пользователей, если они не затрагивают требования, изложенные в техническом задании). Таким образом, пользователи могут внести существенные замечания только после того, как работа над системой будет полностью завершена. Пользователи могут получить систему, не удовлетворяющую их потребностям. В результате приходится начинать новый проект, который может постигнуть та же участь.

Для преодоления перечисленных проблем в середине 80-х годов была предложена спиральная модель ЖЦ (рис.3). Ее принципиальной особенностью является следующее: ***прикладное ПО создается не сразу, как в случае каскадного подхода, а по частям с использованием метода прототипирования***. Под ***прототипом*** понимается действующий программный компонент, реализующий отдельные функции и внешние интерфейсы разрабатываемого ПО. Создание прототипов осуществляется в несколько итераций, или витков спирали. Каждая итерация соответствует созданию фрагмента или версии ПО, на ней уточняются цели и характеристики проекта, оценивается качество полученных результатов и планируются работы следующей итерации. На каждой итерации производится тщательная оценка риска превышения сроков и стоимости проекта, чтобы определить необходимость выполнения еще одной итерации, степень полноты и точности понимания требований к системе, а также целесообразность прекращения проекта. Спиральная модель избавляет пользователей и разработчиков от необходимости точного и полного формулирования требований к системе на начальной стадии, поскольку они уточняются на каждой итерации. Таким образом, углубляются и последовательно конкретизируются детали проекта, и в результате выбирается обоснованный вариант, который доводится до реализации.

Спиральная модель - классический пример применения эволюционной стратегии конструирования. Спиральная модель (автор Барри Боэм, 1988) базируется на лучших свойствах классического жизненного цикла и макетирования, к которым добавляется новый элемент — анализ риска, отсутствующий ранее.

Как показано на рис. 3, модель определяет четыре действия, представляемые четырьмя квадрантами спирали:

1. Планирование — определение целей, вариантов и ограничений.
2. Анализ риска — анализ вариантов и распознавание/выбор риска.
3. Конструирование — разработка продукта следующего уровня.

4. Оценивание — оценка заказчиком текущих результатов конструирования.

Интегрирующий аспект спиральной модели очевиден при учете радиального измерения спирали. С каждой итерацией по спирали (продвижением от центра к периферии) строятся все более полные версии ПО.

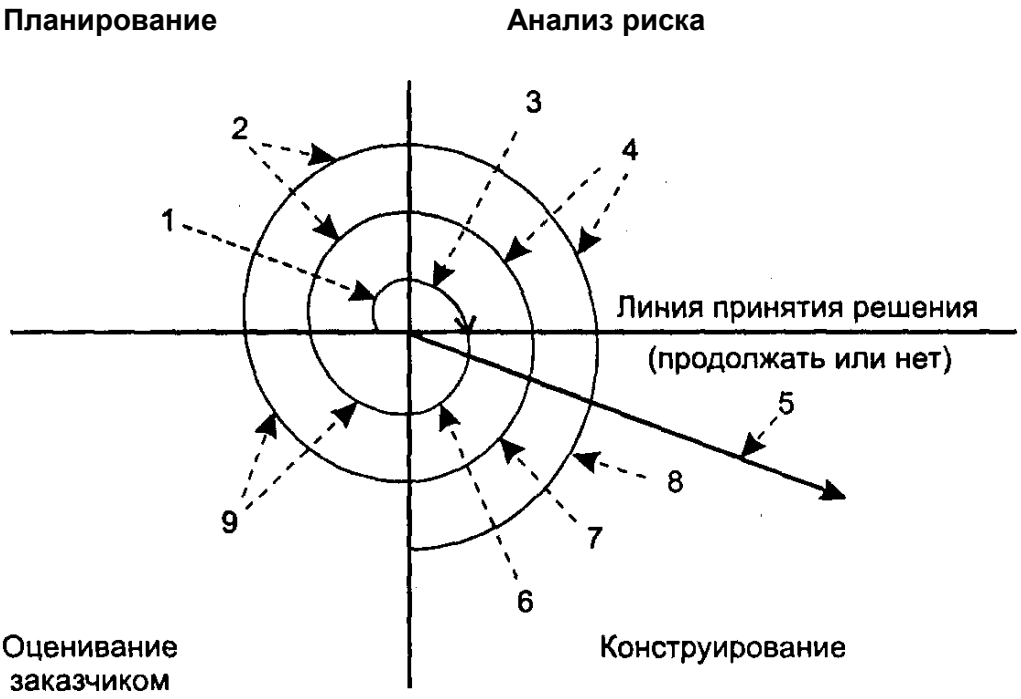


Рис. 3. Спиральная модель: 1 - начальный сбор требований и планирование проекта; 2 - та же работа, но на основе рекомендаций заказчика; 3 - анализ риска на основе начальных требований; 4 - анализ риска на основе реакции заказчика; 5 - переход к комплексной системе; б - начальный макет системы; 7 - следующий уровень макета; 8 - сконструированная система; 9 - оценивание заказчиком.

В первом витке спирали определяются начальные цели, варианты и ограничения, распознается и анализируется риск. Если анализ риска показывает неопределенность требований, на помощь разработчику и заказчику приходит макетирование (используемое в квадранте конструирования). Для дальнейшего определения проблемных и уточненных требований может быть использовано моделирование. Заказчик оценивает инженерную (конструкторскую) работу и вносит предложения по модификации (квадрант оценки заказчиком). Следующая фаза планирования и анализа риска базируется на предложениях заказчика. В каждом цикле по спирали результаты анализа риска формируются виде «продолжать, не продолжать». Если риск слишком велик, проект может быть остановлен.

В большинстве случаев движение по спирали продолжается, с каждым шагом продвигая разработчиков к более общей модели системы. В каждом цикле по спирали требуется конструирование (нижний правый квадрант), которое может быть реализовано классическим жизненным циклом или макетированием. Заметим, что количество действий по разработке (происходящих в правом нижнем квадранте) возрастает по мере продвижения от центра спирали.

При итеративном способе недостающую часть работы можно выполнять на следующей итерации. Главная же задача - как можно быстрее показать пользователям системы работоспособный продукт, тем самым активизируя процесс уточнения и дополнения требований.

Спиральная модель не исключает каскадного подхода на завершающих стадиях проекта в тех случаях, когда требования к системе оказываются полностью определенными.

Основная проблема спирального цикла - определение момента перехода на следующую стадию. Для ее решения необходимо ввести временные ограничения на каждую из стадий ЖЦ. Переход осуществляется в соответствии с планом, даже если не вся запланированная работа закончена. План составляется на основе статистических данных, полученных на предыдущих проектах, и личного опыта разработчиков.

Достоинства спиральной модели:

- наиболее реально (в виде эволюции) отображает разработку программного обеспечения;
- позволяет явно учитывать риск на каждом витке эволюции разработки;
- включает шаг системного подхода в итерационную структуру разработки;
- использует моделирование для уменьшения риска и совершенствования программного изделия.

Недостатки спиральной модели:

- новизна (отсутствует достаточная статистика эффективности модели);
- повышенные требования к заказчику;
- трудности контроля и управления временем разработки.

ПОДХОД RAD

Одним из возможных подходов к разработке прикладного ПО в рамках спиральной модели ЖЦ является получивший широкое распространение способ так называемой ***быстрой разработки приложений, или RAD*** (Rapid Application Development). RAD предусматривает наличие трех составляющих:

- небольших групп разработчиков (3-7 чел.), выполняющих работы по проектированию отдельных подсистем ПО. Это обусловлено требованием максимальной управляемости коллектива;
- короткого, но тщательного проработанного производственного графика

(до 3 месяцев);

- повторяющегося цикла, при котором разработчики по мере того, как приложение начинает приобретать форму, запрашивают и реализуют в продукте требования, полученные в результате взаимодействия с заказчиком.

Команда разработчиков должна представлять собой группу профессионалов, имеющих опыт в проектировании, программировании и тестировании ПО, способных хорошо взаимодействовать с конечным пользователем и трансформировать их предложения в рабочие прототипы.

ЖЦ ПО в соответствии с подходом RAD включает 4 стадии:

1. ***Анализ и планирование требований*** предусматривает действия:
   - определение функций, которые должна выполнять система;
   - выделение наиболее приоритетных функций, требующих проработки в первую очередь;
   - описание информационных потребностей. Формулирование требований к системе осуществляется в основном силами пользователей под руководством специалистов-разработчиков.

   Кроме того, на данной стадии реализуются следующие задачи:
   - ограничивается масштаб времени;
   - устанавливаются временные рамки для каждой из последующих стадий. Определяется сама возможность реализации проекта в заданных рамках финансирования, на имеющихся аппаратных средствах.

Результатом стадии должны быть список расставленных по приоритету функций будущего ПО ЭИС и предварительные модели ПО.

2. ***Проектирование.*** На этой стадии часть пользователей принимает участие в техническом проектировании системы под руководством специалистов-разработчиков. Для быстрого получения работающих прототипов приложений используются соответствующие инструментальные средства (CASE – средства). Пользователи, непосредственно взаимодействуя с разработчиками, уточняют и дополняют требования к системе, которые не были выявлены на предыдущей стадии:
   - более детально рассматриваются процессы системы;
   - при необходимости для каждого элементарного процесса создается частичный прототип: экранная форма, диалог, отчет, устраняющий неясности и неоднозначности;
   - устанавливаются требования разграничения доступа к данным;
   - определяется состав необходимой документации.

После детального определения состава процессов оценивается количество так называемых функциональных точек разрабатываемой системы и принимается решение о разделении ЭИС на подсистемы, поддающиеся реализации одной командой разработчиков за приемлемое время (до 3 месяцев). Под ***функциональной точкой*** понимается любой из следующих элементов разрабатываемой системы:

- входной элемент приложения (входной документ или экранная

форма);

- выходной элемент приложения (отчет, документ, экранная форма)
- запрос (пара «вопрос/ответ»);
- логический файл (совокупность записей данных, используемых внутри приложения);
- интерфейс приложения (совокупность записей данных, передаваемых другому приложению или получаемых от него).

Далее проект распределяется между различными командами разработчиков. (Опыт показывает, что для повышения эффективности работ необходимо разбить проект на отдельные слабо связанные по данным и функциям подсистемы. Реализация подсистем должна выполняться отдельными группами специалистов. При этом необходимо обеспечить координацию ведения общего проекта и исключить дублирование результатов работ каждой проектной группы, которое может возникнуть в силу наличия общих данных и функций).

Результатом данной стадии должно быть:

- общая информационная модель системы;
- функциональные модели системы в целом и подсистем, реализуемых отдельными командами разработчиков;
- точно определенные интерфейсы между автономно разрабатываемыми подсистемами;
- построенные прототипы экранных форм, отчетов, диалогов.

На стадии ***реализации*** выполняется непосредственно сама быстрая разработка приложения.

Разработчики производят итеративное построение реальной системы на основе полученных на предыдущей стадии моделей, а также требований нефункционального характера (требования к надежности, производительности и т.д.).

Пользователи оценивают получаемые результаты и вносят коррективы, если в процессе разработки система перестает удовлетворять определенным ранее требованиям. Тестирование системы осуществляется в процессе разработки.

После окончания работ каждой отдельной команды разработчиков производится постепенная интеграция данной части системы с остальными, формируется полный программный код, выполняется тестирование совместной работы данной части приложения, а затем тестирование системы в целом. Реализация системы завершается выполнением работ:

- Осуществляется анализ использования данных и определяется необходимость их распределения.
- Производится физическое проектирование базы данных
- Формируются требования к аппаратным ресурсам.
- Устанавливаются способы увеличения производительности.
- Завершается разработка документации проекта.

Результатом стадии является готовая система, удовлетворяющая всем согласованным требованиям.

На стадии внедрения производится обучение пользователей,

организационные изменения и параллельно с внедрением новой системы продолжается эксплуатация существующей системы (до полного внедрения новой). Так как стадия реализации достаточно продолжительна, планирование и подготовка к внедрению должны начинаться заранее, как правило, на стадии проектирования системы. Приведенная схема разработки ЭИС не является абсолютной. Возможны различные варианты, зависящие, например, от начальных условий, в которых ведется разработка:

- разрабатывается совершенно новая система;
- уже было проведено обследование организации и существует модель ее деятельности.

В организации уже существует некоторая ЭИС, которая может быть использована в качестве начального прототипа или должна быть интегрирована с разрабатываемой системой.

Следует отметить, что подход RAD, как и любой другой подход, не может претендовать на универсальность. Он хорош в первую очередь для относительно небольших проектов, разрабатываемых для конкретного заказчика. Если же разрабатывается крупномасштабная система (например, масштаба отрасли), которая не является законченным продуктом, а представляет собой комплекс программных компонентов, адаптируемых к программно-аппаратным платформам, системам управления базами данных (СУБД), средствам телекоммуникаций, то на первый план выступают такие показатели проекта как управляемость и качество, которые могут войти в противоречие с простотой и скоростью разработки. Для таких проектов необходимы высокий уровень планирования и жесткая дисциплина проектирования, строгое следование заранее разработанным протоколам и интерфейсам, что снижает скорость разработки.

Подход RAD не применим для построения сложных расчетных программ, операционных систем.

Не годится этот подход и для приложений, в которых отсутствует ярко выраженная интерфейсная часть, наглядно определяющая логику работы системы и приложений, от которых зависит безопасность людей (например программа управления самолетом или атомной станцией), так как итеративный подход предполагает, что первые несколько версий наверняка не будут полностью работоспособны, что в данном случае исключается.

Итак, перечислим основные принципы подхода RAD.

- Разработка приложений итерациями.
- Необязательность полного завершения работ на каждой стадии ЖЦ ПО.
- Обязательность вовлечения пользователей в процесс разработки ЭИС.
- Целесообразность применения CASE – средств, обеспечивающих целостность проекта и генерацию кода приложений.
- Целесообразность применения средств управления конфигурацией, облегчающих внесение изменений в проект и сопровождение готовой системы.
- Использование прототипирования, позволяющее полнее выяснить и

удовлетворить потребности пользователей.

- Тестирование и развитие проекта, осуществляемые одновременно с разработкой.
- Ведение разработки немногочисленной хорошо управляемой командой профессионалов.
- Грамотное руководство разработкой системы, четкое планирование и контроль выполнения работ.

ТЕМА 5.МОДЕЛИ КАЧЕСТВА ПРОЦЕССОВ КОНСТРУИРОВАНИЯ

В современных условиях, условиях жесткой конкуренции, очень важно гарантировать высокое качество вашего процесса конструирования ПО. Такую гарантию дает сертификат качества процесса, подтверждающий его соответствие принятым международным стандартам. Каждый такой стандарт фиксирует свою модель обеспечения качества. Наиболее авторитетны модели стандартов ISO 9001:2000, ISO / IEC 15504 и модель зрелости процесса конструирования ПО (Capability Maturity Model - CMM) Института программной инженерии при американском университете Карнеги-Меллон.

Модель стандарта ISO 9001:2000 ориентирована на процессы разработки из любых областей человеческой деятельности. Стандарт ISO/IEC 15504 специализируется на процессах программной разработки и отличается более высоким уровнем детализации. Достаточно сказать, что объем этого стандарта превышает 500 страниц. Значительная часть идей ISO/IEC 15504 взята из модели CMM.

Базовым понятием модели CMM считается зрелость компании. Незрелой называют компанию, где процесс конструирования ПО и принимаемые решения зависят только от таланта конкретных разработчиков. Как следствие, здесь высока вероятность превышения бюджета или срыва сроков окончания проекта.

Напротив, в зрелой компании работают ясные процедуры управления проектами и построения программных продуктов. По мере необходимости эти процедуры уточняются и развиваются. Оценки длительности и затрат разработки точны, основываются на накопленном опыте. Кроме того, в компании имеются и действуют корпоративные стандарты на процессы взаимодействия с заказчиком, процессы анализа, проектирования, программирования, тестирования и внедрения программных продуктов. Все это создает среду, обеспечивающую качественную разработку программного обеспечения.

Таким образом, модель CMM фиксирует критерии для оценки зрелости компании и предлагает рецепты для улучшения существующих в ней процессов. Иными словами, в ней не только сформулированы условия, необходимые для достижения минимальной организованности процесса, но и даются рекомендации по дальнейшему совершенствованию процессов.

Очень важно отметить, что модель СММ ориентирована на построение системы постоянного улучшения процессов. В ней зафиксированы пять уровней зрелости (рис.5) и предусмотрен плавный, поэтапный подход к совершенствованию процессов - можно поэтапно получать подтверждения об улучшении процессов после каждого уровня зрелости.

Рис. 4. Пять уровней зрелости модели СММ

Начальный уровень (уровень 1) означает, что процесс в компании не формализован. Он не может строго планироваться и отслеживаться, его успех носит случайный характер. Результат работы целиком и полностью зависит от личных качеств отдельных сотрудников. При увольнении таких сотрудников проект останавливается.

Для перехода на повторяемый уровень (уровень 2) необходимо внедрить формальные процедуры для выполнения основных элементов процесса конструирования. Результаты выполнения процесса соответствуют заданным требованиям и стандартам. Основное отличие от уровня 1 состоит в том, что выполнение процесса планируется и контролируется. Применяемые средства планирования и управления дают возможность повторения ранее достигнутых успехов.

Следующий, определенный уровень (уровень 3) требует, чтобы все элементы процесса были определены, стандартизованы и документированы. Основное отличие от уровня 2 заключается в том, что элементы процесса уровня 3 планируются и управляются на основе единого стандарта компании. Качество разрабатываемого ПО уже не зависит от способностей отдельных личностей.

С переходом на управляемый уровень (уровень 4) в компании принимаются количественные показатели качества как программных продуктов, так и процесса. Это обеспечивает более точное планирование проекта и контроль качества его результатов. Основное отличие от уровня 3 состоит в более объективной, количественной оценке продукта и процесса.

Высший, оптимизирующий уровень (уровень 5) подразумевает, что главной задачей компании становится постоянное улучшение и повышение эффективности существующих процессов, ввод новых технологий. Основное отличие от уровня 4 заключается в том, что технология создания и сопровождения программных продуктов планомерно и последовательно совершенствуется.

Каждый уровень CMM характеризуется областью ключевых процессов (ОКП), причем считается, что каждый последующий уровень включает в себя все характеристики предыдущих уровней. Иначе говоря, для 3-го уровня зрелости рассматриваются ОКП 3-го уровня, ОКП 2-го уровня и ОКП 1-го уровня. Область ключевых процессов образуют процессы, которые при совместном выполнении приводят к достижению определенного набора целей. Например, ОКП 5-го уровня образуют процессы:

- предотвращения дефектов;
- управления изменениями технологии;
- управления изменениями процесса.

Если все цели ОКП достигнуты, компании присваивается сертификат данного уровня зрелости. Если хотя бы одна цель не достигнута, то компания не может соответствовать данному уровню CMM.

Тема 6. ПРОЕКТИРОВАНИЕ ПО

Определение метода и технологии. Требования к технологии. Стандарт проектирования. Стандарт оформления проектной документации. Стандарт интерфейса конечного пользователя с системой.

ОПРЕДЕЛЕНИЕ МЕТОДА И ТЕХНОЛОГИИ

Метод проектирования ПО представляет собой организованную совокупность процессов создания ряда моделей, которые описывают различные аспекты разрабатываемой системы с использованием четко определенной нотации. На более формальном уровне метод определяется как совокупность составляющих:

- ***Концепций и теоретических основ***. В качестве таких основ могут выступать структурный или объектно-ориентированный подход.
- ***Нотаций,*** используемых для построения моделей статической структуры и динамики поведения проектируемой системы. В качестве таких нотаций обычно используются графические диаграммы, поскольку они наиболее наглядны и просты в восприятии (диаграммы потоков данных, и диаграммы «сущность – связь» для структурного подхода, диаграммы вариантов

использования, диаграммы классов и др. – для объектно-ориентированного подхода.

- ***Процедур,*** определяющих практическое применение метода (последовательность и правила построения моделей, критерии, используемые для оценки результатов).

Методы реализуются через конкретные технологии и поддерживающие их методики, стандарты и инструментальные средства, которые обеспечивают выполнение процессов ЖЦ ПО.

Технология проектирования определяется как совокупность технологических операций проектирования в их последовательности и взаимосвязи, приводящая к разработке проекта ПО.

ТРЕБОВАНИЯ К ТЕХНОЛОГИИ

Современная технология проектирования должна обеспечивать:

1. Соответствие стандарту ISO/IEC 12207: 1995 (поддержка всех процессов ЖЦ ПО).
2. Гарантированное достижение целей разработки ЭИС в рамках установленного бюджета, с заданным качеством и в установленное время.
3. Возможность декомпозиции проекта на составные части, разрабатываемые группами исполнителей ограниченной численности (3-7 чел.), с последующей интеграцией составных частей.
4. Минимальное время получения работоспособного ПО ЭИС. Речь идет не о сроках готовности всей ЭИС, а о сроках реализации отдельных подсистем. Практика показывает, что даже при наличии полностью завершенного проекта внедрение ЭИС идет последовательно по отдельным подсистемам.
5. Независимость получаемых проектных решений от средств реализации ЭИС (СУБД, ОС, языков и систем программирования).
6. Поддержка комплексом согласованных CASE – средств, обеспечивающих автоматизацию процессов, выполняемых на всех стадиях ЖЦ ПО.

Реальное применение любой технологии проектирования ПО ЭИС в конкретной организации и конкретном проекте невозможно без выработки ряда стандартов (правил, соглашений), которые должны соблюдаться всеми участниками проекта. К таким стандартам относятся:

1. Стандарт проектирования.
2. Стандарт оформления проектной документации.
3. Стандарт интерфейса конечного пользователя с системой.

Стандарт проектирования должен устанавливать:

- Набор необходимых моделей (диаграмм) на каждой стадии проектирования и степень их детализации.
- Правила фиксации проектных решений на диаграммах, в том числе правила именования объектов (включая соглашения по терминологии), набор атрибутов для всех объектов и правила их заполнения на каждой стадии, правила оформления диаграмм (включая требования к форме и размерам объектов) и т.д.
- Требования к конфигурации рабочих мест разработчиков, включая

настройки ОС, настройки CASE – средств и т.д.

- Механизм обеспечения совместной работы над проектом, в том числе правила интеграции подсистем проекта, правила поддержания проекта в одинаковом для всех разработчиков состоянии (регламент обмена проектной информацией, механизм фиксации общих объектов и т.д.), правила анализа проектных решений на непротиворечивость и т.д.

Стандарт оформления проектной документации. Он должен устанавливать:

- комплектность, состав и структуру документации на каждой стадии проектирования (в соответствии со стандартом ГОСТ Р ИСО 9127 – 94 «Системы обработки информации. Документация пользователя и информация на упаковке потребительских программных пакетов»);
- требования к оформлению документации (включая требования к содержанию разделов, подразделов, пунктов, таблиц и т.д.);
- правила подготовки, рассмотрения, согласования и утверждения документации с указанием предельных сроков на каждой стадии;
- требования к настройке издательской системы, используемой в качестве встроенного средства подготовки документации;
- требования к настройке CASE – средств для обеспечения подготовки документации в соответствии с установленными правилами.

Стандарт интерфейса конечного пользователя с системой. Он должен регламентировать:

- Правила оформления экранов (шрифты и цветовая палитра), состав и расположение окон и элементов управления.
- Правила использования клавиатуры и мыши.
- Правила оформления текстов помощи.
- Перечень стандартных сообщений.
- Правила обработки реакций пользователя.

Стандарт пользовательского интерфейса для диалоговых информационных технологий фирмы IBM с некоторыми пояснениями приведен в приложении 1 данной работы.

Детальное конструирование ПО

СУЩНОСТЬ СТРУКТУРНОГО ПОДХОДА. МЕТОДЫ ДОКУМЕНТИРОВАНИЯ ПО

Сущность структурного подхода. Принципы, на которых базируется структурный подход. Метод SADT. Метод DFD.

Проблема сложности является главной проблемой, которую приходится решать при создании больших систем любой природы, в том числе и ЭИС. Ни один разработчик не в состоянии выйти за пределы человеческих возможностей и понять все систему в целом. Единственно эффективный подход к решению этой проблемы заключается в построении сложной системы из небольшого количества крупных частей, каждая из которых, в свою очередь, строится из частей меньшего

размера и т.д., до тех пор, пока самые небольшие части можно будет строить из имеющегося материала. Этот подход известен под самыми разными названиями, среди них такие, как «разделяй и властвуй», иерархическая декомпозиция и др. по отношению к проектированию сложной программной системы это означает, что ее необходимо разделять (декомпозировать) на небольшие подсистемы, каждую из которых можно разрабатывать независимо от других. Это позволяет при разработке подсистемы любого уровня держать в уме информацию только о ней, а не обо всех остальных частях системы. Правильная декомпозиция является главным способом преодоления сложности разработки больших систем. Понятие «правильная» по отношению к декомпозиции означает следующее:

1. Количество связей между отдельными подсистемами должно быть минимальным.
2. Связность отдельных частей внутри каждой подсистемы должна быть максимальной.

Структура системы должна быть таковой, чтобы все взаимодействия между ее подсистемами укладывались в ограниченные, стандартные рамки:

1. Каждая подсистема должна инкапсулировать свою содержимое (скрывать его от других подсистем).
2. Каждая подсистема должна иметь четко определенный интерфейс с другими подсистемами.

На сегодняшний день в программной инженерии существуют два основных подхода к разработке ПО ЭИС, принципиальное различие которых обусловлено разными способами декомпозиции систем. Первый подход называется ***функционально-модульным или структурным***. В его основу положен принцип функциональной декомпозиции, при которой структура системы описывается в терминах иерархии ее функций и передачи информации между отдельными функциональными элементами. Второй, объектно-ориентированный подход использует объектную декомпозицию. При этом структура системы описывается в терминах объектов и связей между ними, а поведение системы описывается в терминах обмена сообщениями между объектами.

Итак, сущность структурного подхода к разработке ПО ЭИС заключается в ее декомпозиции (разбиении) на автоматизируемые функции: система разбивается на функциональные подсистемы, которые, в свою очередь, делятся на подфункции, те - на задачи и так далее до конкретных процедур. При этом система сохраняет целостное представление, в котором все составляющие компоненты взаимоувязаны. При разработке системы «снизу-вверх», от отдельных задач ко всей системе, целостность теряется, возникают проблемы при описании информационного взаимодействия отдельных компонентов.

Все наиболее распространенные методы структурного подхода базируются на ряде общих принципов:

1. Принцип «разделяй и властвуй»;
2. Принцип иерархического упорядочения - принцип организации составных частей системы в иерархические древовидные структуры с добавлением новых деталей на каждом уровне.

Выделение двух базовых принципов не означает, что остальные принципы

являются второстепенными, т.к. игнорирование любого из них может привести к непредсказуемым последствиям (в том числе и к провалу всего проекта»). Основными из этих принципов являются:

1. Принцип абстрагирования - выделение существенных аспектов системы и отвлечение от несущественных.
2. Принцип непротиворечивости обоснованность и согласованность элементов системы.
3. Принцип структурирования данных - данные должны быть структурированы и иерархически организованы.

В структурном подходе в основном две группы средств, описывающих функциональную структуру системы и отношения между данными. Каждой группе средств соответствуют определенные виды моделей (диаграмм), наиболее распространенными среди них являются:

DFD (Data Flow Diagrams) - диаграммы потоков данных;

SADT (Structured Analysis and Design Technique - метод структурного анализа и проектирования) - модели и соответствующие функциональные диаграммы;

ERD (Entity - Relationship Diagrams) - диаграммы «сущность-связь».

Практически во всех методах структурного подхода (структурного анализа) на стадии формирования требований к ПО используются две группы средств моделирования:

1. Диаграммы, иллюстрирующие функции, которые система должна выполнять, и связи между этими функциями - DFD или SADT (IDEF0).
2. Диаграммы, моделирующие данные и их отношения (ERD).

Конкретный вид перечисленных диаграмм и интерпретация их конструкций зависят от стадии ЖЦ ПО.

На стадии формирования требований к ПО SADT - модели и DFD используются для построения модели “AS-IS” и модели “TO-BE”, отражая таким образом существующую и предлагаемую структуру бизнес - процессов организации и взаимодействие между ними (использование SADT - моделей , как правило, ограничивается только данной стадией, поскольку они изначально не предназначались для проектирования ПО). С помощью ERD выполняется описание используемых в организации данных на концептуальном уровне, не зависимо от средств реализации базы данных (СУБД).

На стадии проектирования DFD используются для описания структуры проектируемой системы.

Перечисленные модели в совокупности дают полное описание ПО ЭИС независимо от того, является ли система существующей или вновь разрабатываемой.

ТЕМА 7. ТЕСТИРОВАНИЕ ПО

ТЕСТИРОВАНИЕ - СПОСОБ ОБЕСПЕЧЕНИЯ КАЧЕСТВА

Качество программного продукта характеризуется набором свойств,определяющих, насколько продукт "хорош" с точки зрения

заинтересованных сторон, таких как заказчик продукта, спонсор,конечный пользователь, разработчики и тестировщики продукта, инженеры поддержки, сотрудники отделов маркетинга, обучения и продаж. Каждый из участников может иметь различное представление о продукте и о том,насколько он хорош или плох, то есть о том, насколько высоко качество продукта. Таким образом, постановка задачи обеспечения *качества продукта* выливается в задачу определения заинтересованных лиц, их критериев качества и затем нахождения оптимального решения,удовлетворяющего этим критериям. *Тестирование* является одним из наиболее устоявшихся способов обеспечения качества разработки программного обеспечения и входит в набор эффективных средств современной системы обеспечения *качества программного продукта*.

Правильное определение тестирования таково: ***Тестирование — процесс выполнения программы с намерением найти ошибки.***
Невозможно гарантировать отсутствие ошибок в нетривиальной программе; в
лучшем случае можно попытаться показать наличие ошибок. Если программа правильно ведет себя для солидного набора тестов, нет основании утверждать,
что в ней нет ошибок; со всей определенностью можно лишь утверждать, что не
известно, когда эта программа не работает. Конечно, если есть причины считать
данный набор тестов способным с большой вероятностью обнаружить все возможные
ошибки, то можно говорить о некотором уровне уверенности в правильности программы, устанавливаемом этими тестами.
Психологические эксперименты показывают, что большинство людей, поставив цель
(например, показать, что ошибок нет), ориентируется в своей деятельности на
достижение этой цели. Тестовик подсознательно не позволит себе действовать
против цели, т. е. подготовить тест, который выявил бы одну из оставшихся в программе ошибок. Поскольку мы все признаем, что совершенство в
проектировании и кодировании любой программы недостижимо и поэтому каждая
программа содержит некоторое количество ошибок, самым плодотворным
применением тестирования будет найти некоторые из них. Если мы хотим добиться
этого и избежать психологического барьера, мешающего нам действовать против
поставленной цели, наша цель должна состоять в том, чтобы найти как можно

больше ошибок. Сформулируем основополагающий вывод:
Если ваша цель — показать отсутствие ошибок, вы. их найдете не слишком много. Если же ваша цель — показать наличие ошибок, вы найдете значительную их часть. Надежность невозможно внести в программу в результате тестирования, она определяется правильностью этапов проектирования. Наилучшее решение проблемы надежности — с самого начала не допускать ошибок в программе. Однако вероятность того, что удастся безупречно спроектировать большую программу, бесконечно мала. Роль тестирования состоит как раз в том, чтобы определить местонахождение немногочисленных ошибок, оставшихся в хорошо спроектированной программе. Попытки с помощью тестирования достичь надежности плохо спроектированной программы совершенно бесплодны.
Тестирование оказывается довольно необычным процессом (вот почему оно и считается трудным), так как этот процесс разрушительный. Ведь цель проверяющего (тестовика) — заставить программу сбиться. Он доволен, если это ему удается; если же программа на его тесте не сбивается, он не удовлетворен. Еще одна причина, по которой трудно говорить о тестировании — это тот факт, что о нем известно очень немногое.

ОСНОВНЫЕ ОПРЕДЕЛЕНИЯ

Хотя в тестировании можно выделить несколько различных процессов, такие термины, как тестирование, отладка, доказательство, контроль и испытание, часто используются как синонимы и, к сожалению, для разных людей имеют разный смысл. Хотя стандартных, общепринятых определений этих терминов нет, попытка сформулировать их была предпринята на симпозиуме по тестированию программ.

Нашу классификацию различных форм тестирования мы начнем с того, что дадим эти определения, слегка их дополнив и расширив их список.

Тестирование (testing), как мы уже выяснили,-процесс выполнения программы (или части программы) с намерением (или целью) найти ошибки.

Доказательство (proof) — попытка найти ошибки в программе безотносительно к внешней для программы среде. Большинство методов доказательства предполагает формулировку утверждений о поведении программы и затем вывод и доказательство математических теорем о правильности программы.
Доказательства могут рассматриваться как форма тестирования, хотя они и не предполагают прямого выполнения программы. Многие исследователи считают доказательство альтернативой тестированию — взгляд во многом ошибочный; более подробно это обсуждается в гл. 17.

Контроль (verification) — попытка найти ошибки, выполняя программу в тестовой, или моделируемой, среде.

Испытание (validation) — попытка найти ошибки, выполняя программу в заданной реальной среде.

Аттестация (certification) — авторитетное подтверждение правильности программы, аналогичное аттестации электротехнического оборудования Underwriters Laboratories. При тестировании с целью аттестации выполняется сравнение с некоторым заранее определенным стандартом.

Отладка (debugging) не является разновидностью тестирования. Хотя слова «отладка» и «тестирование» часто используются как синонимы, под ними подразумеваются разные виды деятельности. Тестирование — деятельность, направленная на обнаружение ошибок; отладка направлена на установление точной природы известной ошибки, а затем — на исправление этой ошибки. Эти два вида деятельности связаны — результаты тестирования являются исходными данными для отладки.

Тестирование модуля, или автономное тестирование (module testing, unit testing) — контроль отдельного программного модуля, обычно в изолированной среде (т. е. изолированно от всех остальных модулей). Тестирование модуля иногда включает также математическое доказательство.

Тестирование сопряжении (integration testing) — контроль сопряжении между частями системы (модулями, компонентами, подсистемами).

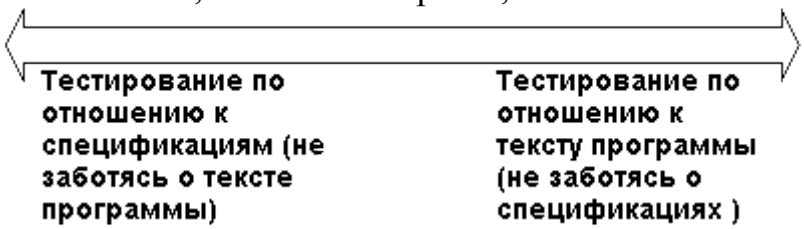
Тестирование внешних функций (external function testing) — контроль внешнего поведения системы, определенного внешними спецификациями.

Комплексное тестирование (system testing) — контроль и/или испытание системы по отношению к исходным целям. Комплексное тестирование является процессом контроля, если оно выполняется в моделируемой среде, и процессом испытания, если выполняется в среде реальной, жизненной.

Тестирование приемлемости (acceptance testing) — проверка соответствия программы требованиям пользователя.

Тестирование настройки (installation testing) — проверка соответствия каждого конкретного варианта установки системы с целью выявить любые ошибки, возникшие в процессе настройки системы.

Отношения между этими типами тестов и проектной документацией, на которой основывается тест, показаны на рис.3,



Спектр подходов к проектированию тестов

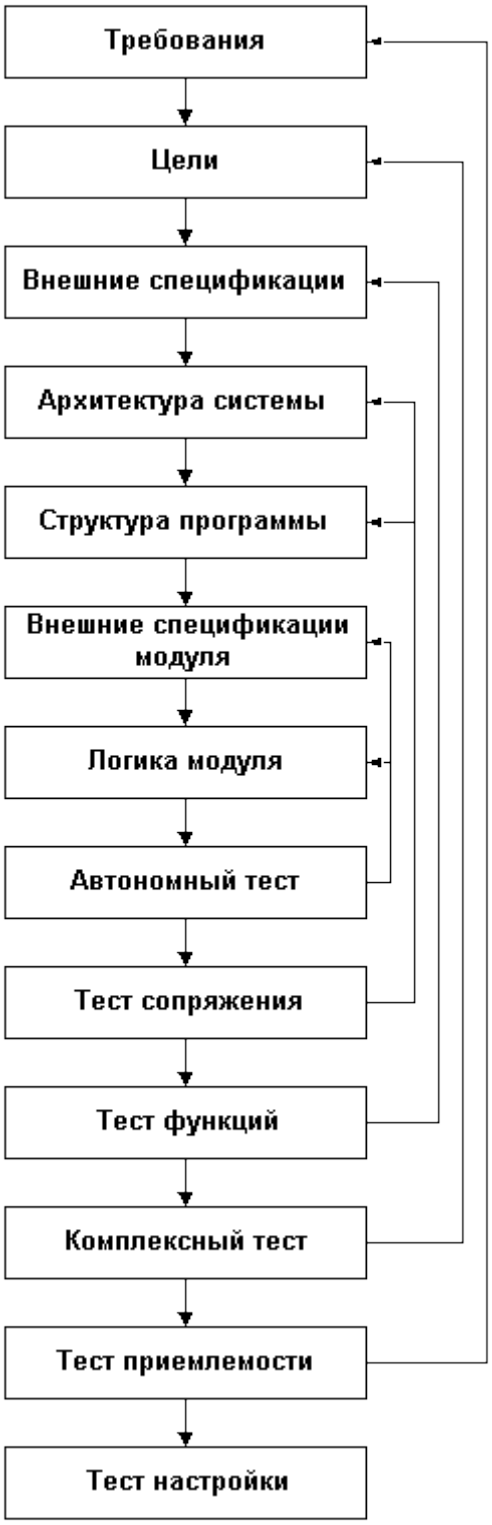


Рис. 3. Процессы тестирования и их связь с процессами проектирования.

ФИЛОСОФИЯ ТЕСТИРОВАНИЯ

Тестирование программного обеспечения охватывает целый ряд видов деятельности, весьма аналогичный последовательности процессов разработки программного обеспечения. Сюда входят постановка задачи для теста, проектирование, написание тестов, тестирование тестов и, наконец,

выполнение тестов и изучение результатов тестирования. Решающую роль играет проектирование теста. Возможен целый спектр подходов к выработке философии, или стратегии проектирования тестов, изображенный на рис.2. Чтобы ориентироваться в стратегиях проектирования тестов, стоит рассмотреть два крайних подхода, находящихся на границах спектра. Следует отметить также, что многие из тех, кто работает в этой области, часто бросаются в одну или другую крайность.

Сторонник (или сторонница) подхода, соответствующего левой границе спектра, проектирует свои тесты, исследуя внешние спецификации или спецификации сопряжения программы или модуля, которые он тестирует. Программу он рассматривает как черный ящик. Позиция его такова: «Меня не интересует, как выглядит эта программа и выполнил ли я все команды или все пути. Я буду удовлетворен, если программа будет вести себя так, как указано в спецификациях». Его идеал — проверить все возможные комбинации и значения на входе.

Приверженец подхода, соответствующего другому концу спектра, проектирует свои тесты, изучая логику программы. Он начинает с того, что стремится подготовить достаточное число тестов для того, чтобы каждая команда была выполнена по крайней мере один раз. Если он немного более искушен, то проектирует тесты так, чтобы каждая команда условного перехода выполнялась в каждом направлении хотя бы раз. Его идеал — проверить каждый путь, каждую ветвь алгоритма. При этом его совсем (или почти совсем) не интересуют спецификации.

Ни одна из этих крайностей не является хорошей стратегией. Читатель, однако, уже, вероятно, заметил, что первая из них, а именно та, в соответствии с которой программа рассматривается как черный ящик, предпочтительней. К сожалению, она страдает тем недостатком, что совершенно неосуществима.

Рассмотрим попытку тестирования тривиальной программы, получающей на входе три числа и вычисляющей их среднее арифметическое. Тестирование этой программы для всех значений входных данных невозможно. Даже для машины с относительно низкой точностью вычислений количество тестов исчислялось бы миллиардами. Даже имей мы вычислительную мощность, достаточную для выполнения всех тестов в разумное время, мы потратили бы на несколько порядков больше времени для того, чтобы эти тесты подготовить, а затем проверить. Такие программы, как системы реального времени, операционные системы и программы управления данными, которые сохраняют «память» о предыдущих входных данных, еще хуже. Нам потребовалось бы тестировать программу не только для каждого входного значения, но и для каждой последовательности, каждой комбинации входных данных. Поэтому исчерпывающее тестирование для всех входных данных любой разумной программы неосуществимо. Эти рассуждения приводят ко второму фундаментальному принципу тестирования: тестирование — ***проблема в значительной степени экономическая .***

Поскольку исчерпывающее тестирование невозможно, мы должны ограничиться чем-то меньшим. Каждый тест должен давать максимальную отдачу по сравнению с нашими затратами. Эта отдача измеряется вероятностью тою, что тест выявит не обнаруженную прежде ошибку. Затраты измеряются временем и стоимостью подготовки, выполнения и проверки результатов теста. Считая, что затраты ограничены бюджетом и графиком, можно утверждать, что искусство тестирования, по существу, представляет собой искусство отбора тестов с максимальной отдачей. Более того, каждый тест должен быть представителем некоторого класса входных значений, так чтобы его правильное выполнение создавало у нас некоторую убежденность в том, что для определенного класса входных данных программа будет выполняться правильно. Это обычно требует некоторого знания алгоритма и структуры программы, и мы, таким образом, смещаемся к правому концу спектра.

ИНТЕГРАЦИЯ МОДУЛЕЙ

Вторым по важности аспектом тестирования (после проектирования тестов) является последовательность слияния всех модулей в систему или программу. Эта сторона вопроса обычно не получает достаточного внимания и часто рассматривается слишком поздно. Выбор этой последовательности, однако, является одним из самых жизненно важных решении, принимаемых на этапе тестирования, поскольку он определяет форму, в которой записываются тесты, типы необходимых инструментов тестирования, последовательность программирования модулей, а также тщательность и экономичность всего этапа тестирования. По этой причине такое решение должно приниматься на уровне проекта в целом и на достаточно ранней его стадии. Имеется большой выбор возможных подходов, которые могут быть использованы для слияния модулей в более крупные единицы. В большинстве своем они могут рассматриваться как варианты шести основных подходов, описанных в следующих шести разделах. Сразу же за ними идет раздел, где предложенные подходы сравниваются по их влиянию на надежность программного обеспечения.

ВОСХОДЯЩЕЕ ТЕСТИРОВАНИЕ

При восходящем подходе программа собирается и тестируется снизу вверх. Только модули самого нижнего уровня («терминальные» модули; модули, не вызывающие других модулей) тестируются изолированно, автономно. После того как тестирование этих модулей завершено, вызов их должен быть так же надежен, как вызов встроенной функции языка или оператор присваивания. Затем тестируются модули, непосредственно вызывающие уже проверенные. Эти модули более высокого уровня тестируются не автономно, а вместе с уже проверенными модулями более

низкого уровня. Процесс повторяется до тех пор, пока не будет достигнута вершина. Здесь завершаются и тестирование модулей, и тестирование сопряжении программы.

При восходящем тестировании для каждого модуля необходим *драйвер:* нужно подавать тесты в соответствии с сопряжением тестируемого модуля. Одно из возможных решении — написать для каждого модуля небольшую ведущую программу.

Тестовые данные представляются как «встроенные» непосредственно в эту программу переменные и структуры данных, и она многократно вызывает тестируемый модуль,

с каждым вызовом передавая ему новые тестовые данные. Имеется и лучшее решение: воспользоваться программой тестирования модулей — это инструмент тестирования, позволяющий описывать тесты на специальном языке и избавляющий от необходимости писать драйверы.

НИСХОДЯЩЕЕ ТЕСТИРОВАНИЕ

Нисходящее тестирование (называемое также нисходящей разработкой не является полной противоположностью восходящему, но в первом приближении может рассматриваться как таковое, При нисходящем подходе программа собирается и тестируется сверху вниз. Изолировано тестируется только головной модуль.

После того как тестирование этого модуля завершено, с ним соединяются (например, редактором связей) один за другим модули, непосредственно вызываемые им, и тестируется полученная комбинация. Процесс повторяется до тех пор, пока не будут собраны и проверены все модули.

При этом подходе немедленно возникает два вопроса: что делать, когда тестируемый модуль вызывает модуль более низкого уровня (которого в данный момент еще не существует), и как подаются тестовые данные. Ответ на первый вопрос состоит в том, что для имитации функций недостающих модулей программируются *модули-заглушки»,* которые моделируют функции отсутствующих модулей. Фраза «просто напишите заглушку» часто встречается в описании этого подхода, но она способна ввести в заблуждение, поскольку задача написания заглушки» может оказаться трудной. Ведь заглушка редко сводится просто к оператору RETURN, поскольку вызывающий модуль обычно ожидает от нее выходных параметров. В таких случаях в заглушку встраивают фиксированные выходные данные, которые она всегда и возвращает. Это иногда оказывается неприемлемым, так как вызывающий модуль может рассчитывать, что результат вызова зависит от входных данных. Поэтому в некоторых случаях заглушка должна быть довольно изощренной, приближаясь по сложности к модулю, который она пытается моделировать.

Интересен и второй вопрос: в какой форме готовятся тестовые данные и как они передаются программе? Если бы головной модуль содержал все

нужные операции ввода и вывода, ответ был бы прост: тесты пишутся в виде обычных для пользователей внешних данных и передаются программе через выделенные ей устройства ввода. Так, однако, случается редко. В хорошо спроектированной программе физические операции ввода-вывода выполняются на нижних уровнях структуры, поскольку физический ввод-вывод — абстракция довольно низкого уровня. Поэтому для того, чтобы решить проблему экономически эффективно, модули добавляются не в строго нисходящей последовательности (все модули одного горизонтального уровня, затем модули следующего уровня), а таким образом, чтобы *обеспечить функционирование операций физического ввода-вывода как можно быстрее.* Когда эта цель достигнута, нисходящее тестирование получает значительное преимущество: все дальнейшие тесты готовятся в той же форме, которая рассчитана на пользователя.

Остается еще один вопрос: в какой форме пишутся тесты до тех пор, пока не будет достигнута эта цель? Ответ: они включаются в некоторые из заглушек.

Нисходящий метод имеет как достоинства, так и недостатки по сравнению с восходящим. Самое значительное достоинство — в том, что этот метод совмещает тестирование модуля, тестирование сопряжении и частично тестирование внешних функций. С этим же связано другое его достоинство — когда модули ввода-вывода уже подключены, тесты можно готовить в удобном виде. Нисходящий подход выгоден также в том случае, когда есть сомнения относительно *осуществимости* программы в целом или если в проекте программы могут оказаться серьезные дефекты.

Преимуществом нисходящего подхода очень часто считают отсутствие

необходимости в драйверах; вместо драйверов вам просто следует написать «заглушки». Как читатель сейчас уже, вероятно, понимает, это преимущество спорно.

Нисходящий метод тестирования имеет, к сожалению, некоторые недостатки.

Основным из них является тот, что модуль редко тестируется досконально сразу после его подключения. Дело в том, что основательное тестирование некоторых модулей может потребовать крайне изощренных заглушек. Программист часто решает не тратить массу времени на их программирование, а вместо этого пишет простые заглушки и проверяет лишь часть условий в модуле. Он, конечно, собирается вернуться и закончить тестирование рассматриваемого модуля позже, когда уберет заглушки. Такой план тестирования определенно не лучшее решение, поскольку об отложенных условиях часто забывают.

Второй тонкий недостаток нисходящего подхода состоит в том, что он может породить веру в возможность начать программирование и тестирование верхнего уровня программы до того, как вся программа будет полностью спроектирована.

Эта идея на первый взгляд кажется экономичной, но обычно дело обстоит совсем наоборот. Большинство опытных проектировщиков признает,

что проектирование программы — процесс итеративный. Редко первый проект оказывается совершенным.

Нормальный стиль проектирования структуры программы предполагает по окончании проектирования нижних уровней вернуться назад и подправить верхний уровень, внеся в него некоторые усовершенствования или исправляя ошибки, либо иногда даже выбросить проект и начать все сначала, потому что разработчик внезапно увидел лучший подход. Если же головная часть программы уже запрограммирована и оттестирована, то возникает серьезное сопротивление любым улучшениям ее структуры. В конечном итоге за счет таких улучшений обычно можно сэкономить больше, чем те несколько дней или недель, которые рассчитывает выиграть проектировщик, приступая к программированию слишком рано.

МОДИФИЦИРОВАННЫЙ НИСХОДЯЩИЙ МЕТОД

Нисходящий подход имеет еще один существенный недостаток, касающийся полноты тестирования. Предположим, что есть большая программа и где-то ближе к нижнему ее уровню находится модуль, предназначенный для вычисления корней квадратного уравнения. Для заданных входных переменных *A, B* и C он решает уравнение.

$$AX^2 + BX + C = 0$$

При проектировании и программировании модуля с такой функцией всегда следует понимать, что квадратное уравнение может иметь как действительные, так и комплексные корни. Для полной реализации этой функции необходимо, чтобы результаты могли быть действительными или комплексными числами (или, если дополнительные затраты на нахождение комплексных корней не оправданы, модуль должен по крайней мере возвращать код ошибки в случае, когда входные коэффициенты задают уравнение с комплексными корнями). Предположим, что конкретный контекст, в котором используется модуль, исключает комплексные корни (т. е. вызывающие модули никогда не задают входных параметров, которые привели бы к комплексным корням). При строго нисходящем методе иногда бывает невозможно тестировать модуль для случая комплексных корней (или тестировать ошибочные условия). Можно попытаться оправдывать это тем, что, поскольку такое уравнение никогда не будет дано модулю, никого не должно заботить, работает ли он и в этих случаях. Да, это безразлично сейчас, но окажется важным в будущем, когда кто-то попытается использовать модуль в новой программе или модифицировать старую программу так, что станут возможными и комплексные корни.

Эта проблема проявляется в разнообразных формах. Применяя нисходящее тестирование в точном соответствии с предыдущим разделом, часто невозможно тестировать определенные логические условия, например ошибочные ситуации или защитные проверки. Нисходящий метод, кроме

того, делает сложной или вообще невозможной проверку исключительных ситуаций в некотором модуле, если программа работает с ним лишь в ограниченном контексте (это означает, что модуль никогда не получит достаточно полный набор входных значений). Даже если тестирование такой ситуации в принципе осуществимо, часто бывает трудно определить, какие именно нужны тесты, если они вводятся в точке программы, удаленной от места проверки соответствующего условия.

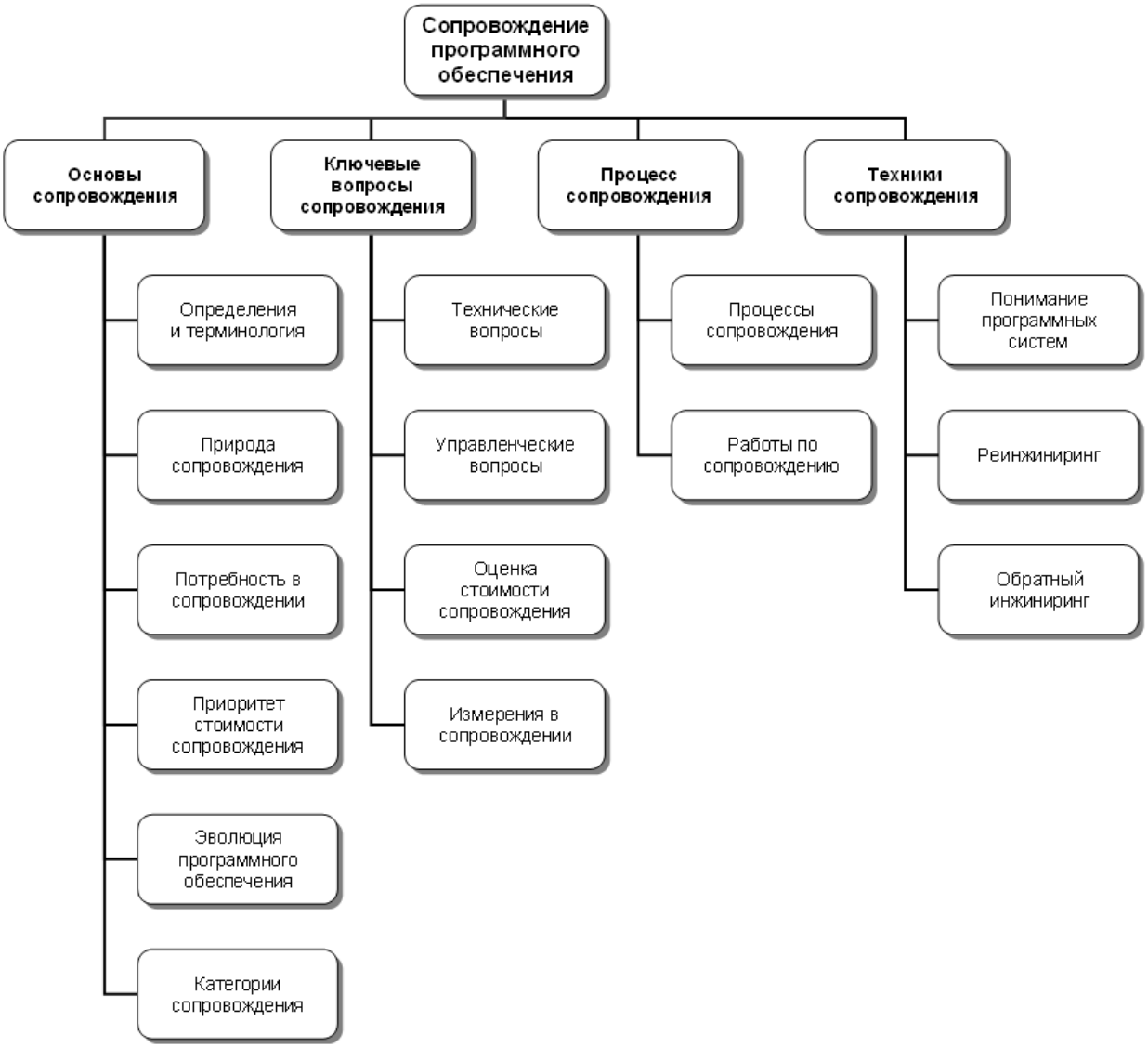
Метод, называемый модифицированным нисходящим подходом, решает эти проблемы: требуется, чтобы каждый модуль прошел автономное тестирование перед подключением к программе. Хотя это действительно решает все перечисленные проблемы, здесь требуются и драйверы, и заглушки для каждого модуля.

ТЕМА 8. СОПРОВОЖДЕНИЕ ПО

Автор данного перевода SWEBOK с замечаниями и комментариями - Сергей Орлик.
Оригинальный перевод доступен в блоге Сергея Орлика http://sorlik.blogspot.com/

Сопровождение программного обеспечения в SWEBOK определяется как вся совокупность деятельности, необходимой для обеспечения эффективной (с точки зрения затрат) поддержки программных систем. Эти работы выполняются как перед вводом системы в эксплуатацию, так и после этого. Предварительные работы включают планирование деятельности по сопровождению системы, а также организацию перехода к ее полнофункциональному использованию. Если новая система должна заменить старую систему, предназначенную для решения тех же задач, просто на новом уровне эффективности, стоимости использования, новых функциональных возможностей, в этом случае важно обеспечить плавный переход со старой системы на новую, максимально естественный для пользователей. С этим связано не только планирование, например, переноса информации, хранимой в соответствующих базах данных, но и обучение пользователей, подготовка, настройка и проверка “боевой” конфигурации, определение последовательности операций, организация и обучение службы поддержки (help-desk) и т.п.

Область знаний “Сопровождение программного обеспечения” связана с другими аспектами программной инженерии. По сути, описание этой области знаний непосредственно пересекается со всеми другими дисциплинами.



Определения и терминология (Definitions and Terminology)

Сопровождение программного обеспечения определяется стандартом IEEE Standard for Software Maintenance (IEEE 1219) как модификация программного продукта после передачи в эксплуатацию для устранения сбоев, улучшения показателей производительности и/или других характеристик (атрибутов) продукта, или адаптации продукта для использования в модифицированном окружении. Интересно, что данный стандарт также касается вопросов подготовки к сопровождению до передачи системы в эксплуатацию, однако, структурно это сделано на уровне соответствующего информационного приложения, включенного в стандарт.

В свою очередь, стандарт жизненного цикла 12207 (IEEE, ISO/IEC, ГОСТ Р ИСО/МЭК) позиционирует сопровождение как один из главных процессов жизненного цикла. Этот стандарт описывает сопровождение как процесс модификации программного продукта в части его кода и

документации для решения возникающих проблем <при эксплуатации> или реализации потребностей в улучшениях <тех или иных характеристик продукта>. Задача состоит в модификации продукта при условии сохранения его целостности. Международный стандарт ISO/IEC 14764 (Standard for Software Engineering - Software Maintenance) определяет сопровождение программного обеспечения в тех же терминах, что и стандарт 12207, придавая особое значение работам по подготовке к деятельности по сопровождению до передачи системы в реальную эксплуатацию, например, вопросам планирования регламентов и операций по сопровождению.

Природа сопровождения (Nature of Maintenance)

Сопровождение поддерживает функционирование программного продукта на протяжении всего операционного жизненного цикла, то есть периода его эксплуатации. В процессе сопровождения фиксируются и отслеживаются запросы на модификацию (также называемые “запросами на изменения” – change requests, в частности, в контексте конфигурационного управления), оценивается влияние предлагаемых изменений, модифицируется код и другие активы (артефакты) продукта, проводится необходимое тестирование и, наконец, выпускается обновленная версия продукта. Кроме того, проводится обучение пользователей и обеспечивается их ежедневная поддержка при работе с текущей версией продукта. В SWEBOK отмечается, что сопровождение, с точки зрения операций отслеживания и контроля, обладает большим содержанием, чем разработка (в общем понимании). С точки зрения автора, объем и активность операций по контролю разработки в большой степени зависит от сложившихся практик, внутрикорпоративных регламентов и требований, а также применяемых методологий и концепции управления (в частности – проектного менеджмента). Так или иначе, отслеживание и контроль – ключевые элементы деятельности по сопровождению программного обеспечения (как и других ИТ-активов предприятия).

Стандарт 12207 определяет понятие “maintainer” - в соответствующем ГОСТ он именуется как “персонал сопровождения”, подразумевая организацию, выполняющая работы по сопровождению. SWEBOK использует данный термин, также, и в отношении лиц (individuals), проводящих определенные работы по сопровождению, в отличие, например, от разработчиков, занимающихся реализацией системы в программном коде.

Стандарт жизненного цикла 12207 также идентифицирует основные работы по сопровождению: реализация процесса <сопровождения>, анализ проблем и модификаций (изменений), реализаций модификаций, обзор (оценка)/принятие <решений> по сопровождению, миграция (с одной версии программного продукта на другую, с одного продукта на другой) и вывод

системы из эксплуатации. Эти работы описываются далее в теме 3.2 “Работы по сопровождению” (Maintenance Activities).

Специалисты по сопровождению (персонал сопровождения) могут получать знания о программном продукте непосредственно от разработчиков. Взаимодействие с разработчиками и раннее привлечение этих специалистов помогает уменьшить усилия, необходимые для адекватного сопровождения программной системы. По мнению автора, передача знаний персоналу сопровождения, его обучение, должно начинаться не позднее начала опытной эксплуатации продукта. В противном случае, усилия на одновременную поддержку прикладной системы и обучение соответствующих специалистов не только превысит реально допустимые нормы загрузки персонала (как группы или службы сопровождения и техподдержки, так и разработчиков системы), но снизит эффективность поддержки пользователей на критически важном этапе первоначального использования новой системы. По опыту автора и результатам обсуждения этого вопроса с сотрудниками внутрикорпоративных ИТ-департаментов, обычно, в зависимости от сложности системы, пик нагрузки на службу сопровождения приходится в течении первых 2 - 6 недель, с момента передачи системы в реальную эксплуатацию, тем более, при отказе от использования старой системы или ее предыдущей версии. SWEBOK отмечает, что, в некоторых случаях, инженеры (создававшие систему) не могут быть привлечены к обучению и поддержке персонала сопровождения. Особенно часто это касается тиражируемых или “коробочных” систем. Это создает дополнительные трудности для специалистов, обеспечивающих сопровождение. В то же время, инженеры, занимающиеся технической поддержкой (несколько боле узкий круг в команде сопровождения, включающей менеджеров, администраторов и других специалистов), должны (в зависимости от типа продукта) иметь доступ к активам проекта (например, описанию его внутренней архитектуры), включая код, документацию и т.п. Именно таким образом начинает формироваться информационная инфраструктура службы технической поддержки и сопровождения у производителей программных продуктов.

Практика показывает, что инженеры по технической поддержке производителя программного обеспечения (не только “коробочного”, но и создаваемого и настраиваемого интеграторами, обладающими собственными программными решениями) должны не просто иметь доступ ко всем ключевым активам проекта (код, документация, спецификации требований, внутренние модели и т.п.), но в их обязанности входит создание “патчей” (patch – “заплата”), исправлений ошибок и, в особых случаях, такие изменения, до выпуска новой версии продукта, создаются с привлечением непосредственно разработчиков продукта (групп и подразделений R&D – Research and Development). При этом, разработчики продукта информируются о найденных ошибках и, в случае нахождения

соответствующих решений специалистами технической поддержки, такие решения передаются разработчикам с тем, чтобы те либо включили такие изменения в новую версию программного продукта (безусловно, в случае успешного прохождения всех необходимых тестов), либо нашли более адекватное решение в контексте новой функциональности либо тех изменений, которые включены в новую версию продукта. В обязанности инженеров службы сопровождения, в общем случае, входит: проверка пользовательского сценария, приводящего к сбою; идентификация причин сбоя, т.е локализация ошибки/причин ее появления; предоставление соответствующих исправлений или, при невозможности создания таковых на данном этапе либо в заданные сроки – предоставление обходного пути решения проблемы для достижения требуемых бизнес-задач (такие обходные пути, обычно, называют “workaround”); журналирование всех работ и операций; помещение описания проблемы и ее решения в базу знаний службы сопровождения; передача всей информации разработчикам; своевременное информирование пользователя о статусе запроса и некоторые другие работы, содержание которых может варьироваться, в зависимости от регламентов и корпоративных стандартов в конкретной организации, либо параметров контракта на сопровождение и техническую поддержку, если таковой есть.

Потребность в сопровождении (Need for Maintenance)

Сопровождение необходимо для обеспечения того, чтобы программный продукт на протяжении всего периода эксплуатации удовлетворяет требованиям пользователей. Деятельность по сопровождению применима для программного обеспечения, созданного с использованием любой модели жизненного цикла и методологии разработки. На первый взгляд, это утверждение SWEBOK может показаться тривиальным. Однако, обратитесь к своему опыту разработки и использования различного программного обеспечения. Наверняка, Вы найдете случаи из собственной практики или практики коллег, когда столь очевидное утверждение хорошо бы донести до разработчика того или иного программного продукта. Изменения программной системы могут быть обусловлены как действиями по корректировке ее поведения или несвязанные с необходимостью корректировки (подразумевая уже не исправление ошибок, а, например, повышение производительности или расширение функциональности).

В общем случае, работы по сопровождению должны проводиться для решения следующих задач:

- устранение сбоев;
- улучшение дизайна;
- реализация расширений (функциональных возможностей);

- создание интерфейсов взаимодействия с другими (внешними) системами;
- адаптация (например, портирование) для возможности работы на другой аппаратной платформе (или обновленной платформе), применения новых системных возможностей, функционирования в среде обновленной телекоммуникационной инфраструктуры и т.п.;
- миграции унаследованного (legacy) программного обеспечения;
- вывода программного обеспечения из эксплуатации.

Деятельность персонала сопровождения включает четыре ключевых аспекта: поддержка контроля (управляемости) программного обеспечения в течение всего цикла эксплуатации:

- поддержка модификаций программных систем;
- совершенствование существующих функций;
- предотвращение падения производительности программной системы до неприемлемого уровня.

Автор убежден, что говоря о предотвращении деградации производительности, мы должны понимать, что это, при всем желании совершенствования системы, может делаться и за счет обновления мощности аппаратной части и/или соответствующей телекоммуникационной инфраструктуры, если это более обосновано, чем модификация самой программной системы. На самом деле это вопрос того, что окажется дешевле (и менее рискованно), т.е. связано с затратами/ стоимостью соответствующих работ, оборудования и поддержки обновленного системного окружения (что, к сожалению, часто также не учитывается даже при более-менее сложившейся практике сопровождения).

Приоритет стоимости сопровождения (Majority of Maintenance Costs)

Работы по сопровождению потребляют если не большую (как отмечает SWEBOK), то значительную часть финансовых ресурсов жизненного цикла программного обеспечения. Общее понимание сопровождения подразумевает лишь устранение сбоев. Однако, исследования и опросы на протяжении многих лет показывают, что более 80% усилий по сопровождению связаны не столько устранением сбоев, сколько с другими работами, не связанными с исправлением дефектов. Многие менеджеры по сопровождению объединяют в отчетности вопросы расширения функциональности и исправления ошибок в поддерживаемых программных системах. Такое смешение качественно различных работ приводит к неправильному представлению о реальной, на самом деле, не столь высокой стоимости сопровождения в части устранения дефектов. Понимание различных категорий работ в рамках деятельности по сопровождению помогает понять структуру реальных затрат. Кроме того, понимание

факторов, влияющих на возможности сопровождения системы, помогают не только сохранять необходимый уровень затрат, но и снижать их.

Существуют как технические, так и другие (например, организационные, являющиеся, по мнению автора, наиболее сильно влияющими на объем затрат) факторы, оказывающие влияние на стоимость сопровождения, в целом:

- тип приложения;
- новизна программного обеспечения;
- наличие и квалификация персонала по сопровождению;
- длительность использования программной системы;
- характеристики и специфика аппаратной части (а также телекоммуникационной инфраструктуры);
- качество дизайна (например, модульность или масштабируемость), кода, документации и соответствующих работ по тестированию системы.

Эволюция программного обеспечения (Evolution of Software)

В 1969 году Леман (см. рекомендуемую литературу к данной секции SWEBOK) впервые связал деятельность по сопровождению и вопросы эволюции программного обеспечения. Результаты более чем 20-ти летних исследований во главе с Леманом привели к формулированию ряда важных положений, ключевое из которых утверждает, что деятельность по сопровождению, по-сути, представляет собой эволюционную разработку программных систем. Принятию тех или иных решений в процессе сопровождения, помогает понимание того, что происходит с программной системой в процессе ее эксплуатации. Существующее (особенно, корпоративное) программное обеспечение никогда не бывает полностью завершенным и продолжает эволюционировать в течение всего срока эксплуатации. В процессе эволюционирования, программная система становится все более сложной до тех пор, пока не предпринимаются специальные усилия (в том числе, в рамках специального проекта по модификации) по уменьшению его сложности.

В то же самое время, если можно выделить тенденции развития программной системы и ее поведение достаточно стабильно, его эволюционирование можно измерить. Последние годы делаются попытки разработать соответствующие модели оценки усилий по сопровождению. В результате уже создаются определенные средства (численные и инструментальные) управления работами по сопровождению, ряд из которых приводится в ссылках к данной секции SWEBOK.

Категории сопровождения (Categories of Maintenance)

Многие источники, в частности, стандарт IEEE 1216, определяют три категории работ по сопровождению: корректировка, адаптация и совершенствование. Такая классификация была обновлена в стандарте ISO/IEC 14764 Standard for Software Engineering - Software Maintenance введением четвертой составляющей. Таким образом, сегодня говорят о четырех категориях сопровождения:

- Корректирующее сопровождение (corrective maintenance): “реактивная” модификация программного продукта, выполняемая уже после передачи в эксплуатацию для устранения сбоев;
- Адаптирующее сопровождение (adaptive maintenance): модификация программного продукта на этапе эксплуатации для обеспечения продолжения его использования с заданной эффективностью (с точки зрения удовлетворения потребностей пользователей) в изменившемся или находящемся в процессе изменения окружении; в первую очередь, подразумевается изменение бизнес-окружения, порождающее новые требования к системе;
- Совершенствующее сопровождение (perfective maintenance): модификация программного продукта на этапе эксплуатации для повышения характеристик производительности и удобства сопровождения;
- Профилактическое сопровождение (preventive maintenance): модификация программного продукта на этапе эксплуатации для идентификации и предотвращения скрытых дефектов до того, когда они приведут к реальным сбоям.

ISO/IEC 14764 (*Standard for Software Engineering - Software Maintenance*) классифицирует адаптивное и совершенствующее сопровождение как работы по расширению (функциональности) продукта. Этот стандарт также объединяет корректирующую и профилактическую деятельность в общую категорию работ по корректировке системы. Профилактическое сопровождение (новейшая категория работ по сопровождению) наиболее часто проводится для программных систем, связанных с вопросами безопасности (людей).

Таблица . Категории сопровождения программного обеспечения.

	Корректирующие работы	Работы по расширение
“Проактивный” подход	Профилактическое сопровождение	Совершенствующее сопровождение
“Реактивный” подход	Корректирующее сопровождение	Адаптирующее сопровождение

КАЧЕСТВО ПРОГРАММНЫХ СРЕДСТВ

В современном мире разработка ПО превратилась в одну аз самых дорогостоящих индустрии и любые узкие, места в технологическом процессе его создания могут привести к нежелательным результатам. Удлинение сроков разработки ПО чревато удорожанием конечного продукта, а не выявленные в ходе тестирования ошибки приводят как минимум к снижению его производительности. Примитивные ошибки, невнятные сообщения и неряшливый интерфейс раздражают пользователей, которые в итоге выбирают более качественный продукт конкурента, а фирма рискует потерять не только клиентов, но и свою долю рынка. Итак, качество ПО приобретает первостепенное значение. Но как оценить это самое качество и в чем его измерить? Можно ли создать "добротный" программный продукт, пользуясь убогими инструментальными средствами? Ответам на поставленные вопросы, а также описанию инструментария, позволяющего оценивать качество ПО, и посвящен следующий раздел курса.

ОСНОВНЫЕ ПОНЯТИЯ КАЧЕСТВА ПРОГРАММНЫХ СРЕДСТВ

Системы качества**.** Качество функционирования. Качество в использовании. Основные факторы качества.

Имеется множество определений фундаментальной категории - качество, которые, по существу, сводятся к совокупности технических, технологических и эксплуатационных характеристик продукции или процессов, посредством которых они способны отвечать требованиям потребителя и удовлетворять его при применении. В соответствии со стандартами ***обеспечение качества*** - это "совокупность планируемых и систематически проводимых мероприятий, необходимых для уверенности в том, что продукция или процессы удовлетворяют определенным требованиям потребителей к качеству. Для реализации этого положения предназначены ***системы качества***, каждая из которых включает: совокупность организационной структуры, ответственности, процедур, процессов и ресурсов, обеспечивающую осуществление руководства качеством продукции или процессов.

Изучением и реализацией методов и средств количественного оценивания качества продукции занимается научная дисциплина – квалиметрия. Эффективное управление качеством возможно лишь при наличии достаточно точных и объективных методов измерения или оценивания качества продукции или процессов. Создание и развитие квалиметрии подготовило обоснованное применение: численных, количественных методов в решении задач при оценке качества технологических процессов и готовой продукции, методов выбора

предпочтений при анализе альтернативных групп продуктов, методов расчета интегрального качества, определении достоверности выборок при статистических оценках качества и ряд других задач управления качеством. В основе квалиметрии лежат ***три базовых положения***:

- практическая необходимость методов количественной оценки характеристик качества продукции для решения задач их планирования и контроля на различных уровнях управления созданием и применением;
- подход к качеству как к единому динамическому сочетанию ряда отдельных свойств, каждое из которых в силу своего характера и взаимосвязей с другими свойствами (с учетом их весомости и приоритета) оказывает влияние на формирование иерархической структуры обобщенного качества продукции;
- наличие принципиальной возможности измерения в количественной форме, как отдельных свойств, так и их сочетаний, в том числе интегрального качества.

Теоретическая квалиметрия абстрагируется от конкретных объектов и изучает общие закономерности и математические модели, связанные с оцениванием качества. Ее содержанием являются общие методологические проблемы количественной оценки качества, а также методы, направленные на преодоление общих трудностей, характерных для многих конкретных методик, предназначенных для количественной оценки качества конкретных объектов разного назначения.

Качество объекта зависит от того, для какой ***цели***, для какого ***потребителя*** и для каких ***условий*** делается его оценка. Один и тот же объект может иметь несколько различных оценок качества, произведенных для различных целей и разных условий определения. При квалиметрических измерениях и оценках, качество рассматривается как иерархическая совокупность свойств, расположенных на различных уровнях. Каждое из свойств на одном уровне зависит от ряда других свойств, лежащих на более низких уровнях. Число уровней свойств по мере углубления знаний о конкретной продукции может возрастать. Изучение взаимосвязи между свойствами, входящими в состав обобщенного качества должно теоретически обосновать правомочность его разложения для целей соединения оценок отдельных свойств в комплексные оценки.

Практической задачей квалиметрии является разработка и развитие всех комплексных и дифференциальных методов оценки качества. Дифференциальные и экономические оценки являются основой, для комплексной оценки и определения интегральных показателей качества продукции, основанных на обобщении и сопоставлении ее отдельных полезных свойств и затрат ресурсов. Для получения комплексной оценки используется экспертное определение весомости каждого свойства и в первую очередь должно учитываться влияние этого свойства на эффективность использования данного вида продукции.

Значительную роль в квалиметрии играют экспертные методы. При экспертных методах, оценки, даваемые отдельными экспертами -

субъективны, зависят от целого ряда их индивидуальных особенностей: профессии и квалификации эксперта, его знания условий применения продукции, содержательности и количества информации, которой он пользуется. Математическая обработка совокупностей субъективных оценок позволяет получать более объективную оценку качества. Величина погрешности и надежность такой оценки, в значительной степени, зависят от точности оценок отдельных экспертов, их числа, методов обобщения и обработки результатов.

Большое место в квалиметрии занимают статистические методы исследования. Многие показатели качества продукции определяются при помощи статистических методов по опытным данным или по материалам эксплуатационной статистики. Такие обобщенные квалиметрические оценки качества часто получаются путем измерения и сравнения физических, экономических, эстетических и других характеристик с лучшими образцами, которые формально такими эталонами не являются.

Разнообразие областей применения компьютеров становится все шире и их корректная работа часто является определяющей для качественного управления объектами, успеха предприятий или безопасности человека. Поэтому тщательное ***специфицирование и оценивание характеристик качества программного продукта - ключевой фактор обеспечения их адекватного применения***. Это может быть достигнуто на основе выделения и определения подходящих характеристик с учетом целей использования и функциональных задач ПС. Важно, чтобы ПС оценивалось по каждой применимой характеристике качества с использованием стандартизированной или формализованной метрики.

Применительно к программным средствам ***система обеспечения качества*** - это совокупность методов и средств организации управляющих и исполнительных подразделений предприятия, участвующих в проектировании, разработке и сопровождении комплексов программ с целью придания им свойств, обеспечивающих удовлетворение потребностей заказчиков и потребителей при минимальном или допустимом расходовании ресурсов. Для сложных ПС с высокими требованиями к качеству проектирование, развитие и применение таких систем должно сопровождать весь жизненный цикл основной продукции - комплексы программ. Различия фактических и требуемых показателей качества объектов или процессов квалифицируются как дефекты или ошибки и являются первичными стимулами для принятия и реализации решений по изменению определяемых значений качества. Для этого необходимы экономические и моральные причины, а также воля руководителей, организация исполнителей, методы и технология для управления качеством и корректировки программ.

Потребителя-заказчика, прежде всего, интересуют функции и качество готового конечного продукта - программного средства, и обычно не очень беспокоит, как они достигнуты. Требуемое качество при разработке проектов ПС, как и любой продукции, можно обеспечить двумя методами:

- путем использования только заключительного контроля и испытаний готовых объектов и исключения из поставки или направлением на доработку продуктов, не соответствующих требуемому качеству;
- посредством применения регламентированных технологий и систем обеспечения качества процессов проектирования и разработки, предотвращающих дефекты и гарантирующих высокое качество продукции во время ее создания и модификации.

Первый метод может приводить к значительным экономическим потерям за счет затрат на создание части не пригодного к использованию брака, что может быть очень дорого для сложных систем. Достижение необходимого качества за счет только выходного контроля, при отсутствии адекватной технологии и системы обеспечения качества в процессе разработки, может приводить к длительному итерационному процессу массовых доработок и повторных испытаний продукции.

Второй метод обеспечивает высокое качество выполнения всего процесса проектирования и разработки, и тем самым минимум экономических потерь от брака, что более рентабельно при создании сложных систем. При этом сокращается, но не исключается выходной контроль качества продукции, Для создания современных прикладных высококачественных информационных систем необходимы оба метода, с акцентом на применение регламентированных технологий. Таким образом, ***обеспечение и удостоверение качества сложных ПС должно базироваться на проверках и испытаниях:***

- технологий обеспечения жизненного цикла программных средств, поддержанных регламентированными системами качества;
- готового программного продукта с полным комплектом адекватной эксплуатационной документации.

Глубокая взаимосвязь качества разработанных программ с качеством технологии их создания и с затратами на разработку становится особенно существенной при необходимости получения конечного продукта с предельно высокими значениями показателей качества. Установлено, что затраты на разработку резко возрастают, когда показатель качества приближается к пределу, достижимому при данной технологии и уровне автоматизации процесса разработки. Это привело к ***существенному изменению в последние годы объектов, методологии и культуры в области создания и совершенствования ПС***. Непрерывный рост требований к качеству ПС стимулировали создание и активное применение международных стандартов и регламентированных технологий, автоматизирующих основные процессы их жизненного цикла, начиная с инициирования проекта.

Основой для формирования требований к ПС является анализ свойств, характеризующих качество его функционирования с учетом технологических и ресурсных возможностей разработчика. При этом под ***качеством функционирования*** понимается совокупность свойств, обусловливающих пригодность ПС обеспечивать надежное и своевременное представление

требуемой информации потребителю для ее дальнейшего использования по назначению. Адекватный набор показателей качества программ зависит от функционального назначения и свойств каждого ПС. В соответствии с принципиальными особенностями ПС при проектировании должны выбираться номенклатура и значения показателей качества, необходимых для его эффективного применения пользователями, которые впоследствии отражаются в технической документации и в спецификации требований на конечный продукт.

Каждый критерий качества может использоваться, если определена его метрика и может быть указан способ ее оценивания и сопоставления с требующимся эталонным значением. Для конкретных ПС доминирующие критерии качества выделяются и определяются при проектировании его функциональным назначением и требованиями технического задания. Программы для ЭВМ как объекты проектирования, разработки, испытаний и оценки качества характеризуются следующими обобщенными показателями:

- проблемно-ориентированной областью применения, техническим и социальным назначением программного комплекса;
- конкретным типом решаемых функциональных задач с достаточно определенной областью применения соответствующими пользователями;
- объемом и сложностью совокупности программ и базы данных, решающей единую целевую задачу данного типа;
- необходимыми составом и требуемыми значениями характеристик качества функционирования программ и величиной допустимого риска (ущерба) из-за недостаточного их качества;
- степенью связи решаемых задач с реальным масштабом времени или допустимой длительностью ожидания результатов решения задачи;
- прогнозируемыми значениями длительности эксплуатации и перспективой создания, множества версий комплекса программ;
- предполагаемым тиражом производства и применения комплекса программ;
- степенью необходимой документированности программ.

Качество в использовании - это основное качество системы, содержащей ПС, которое воспринимается пользователями. Оно измеряется скорее в терминах результата функционирования и применения программ, чем внутренних свойств самого ПС. Цель такого оценивания - определение, имеет ли продукт требуемый эффект в специфическом контексте использования. Качество ПС в среде пользователей может отличаться от качества в среде разработчиков, поскольку некоторые функции могут быть невидимы пользователю или не использоваться им. Пользователь оценивает только те атрибуты ПС, которые видимы и полезны ему в процессе реального применения. Поэтому к дефектам комплексов программ следует относить не только прямые потери при их применении пользователями, но и избыточные свойства, которые не нужны пользователям и потребовали дополнительных

затрат при разработке. Иногда атрибуты ПС, специфицированные пользователем на этапе анализа требований, впоследствии не удовлетворяют его надежды при применении продукта вследствие изменения взглядов и понятий, а также трудности специфицирования неявных потребностей в начале проектирования.

Качество изменяется в течение жизненного цикла ПС, то есть его требуемое и реальное значение в начале ЖЦ почти всегда отличается от фактически достигнутого при завершении проекта и качества поставляемой пользователям версии продукта. На практике важно оценивать качество программ не только в завершенном виде, но и в процессе их проектирования, разработки и сопровождения. Кроме того, оценки показателей качества могут быть субъективными и отражать различные точки зрения и потребности разных специалистов. Чтобы эффективно управлять качеством на каждом этапе ЖЦ, необходимо уметь определять и примирять эти различные представления требуемого качества и его изменения. Характеристики этого процесса в значительной степени определяются совокупными затратами, необходимыми для достижения заданного качества конечного продукта - версии программного средства.

Требуемые характеристики качества ПС с различных позиций отражают их свойства и особенности, и в свою очередь зависят от ряда факторов и ограничений. При системном анализе и проектировании программных средств необходимо определять и учитывать связи, влияние и взаимодействие следующих ***основных факторов***, которые отражаются на их качестве:

- назначение, содержание и описание функциональных характеристик, субхарактеристик и атрибутов, определяющих специфические особенности целей, задач, свойств и сферы применения конкретного программного средства – его функциональную пригодность;
- конструктивные характеристики качества, способствующие улучшению и совершенствованию назначения, функций и возможностей применения ПС;
- метрики, меры и шкалы, выбранных и пригодных для измерения и оценивания конкретных характеристик и атрибутов качества ПС с учетом определенной достоверности;
- уровни возможной детализации при описании и оценивании определенных характеристик и атрибутов качества ПС;
- цели и особенности потребителей результатов оценивания характеристик качества ПС;
- внешние и внутренние, негативные факторы, влияющие на достигаемое качество создания и применения ПС;
- доступные ресурсы, ограничивающие возможные величины реальных характеристик качества ПС;
- конкурентоспособность, выраженная отношением эффективности применения к стоимости приобретения и эксплуатации ПС.

Влияние перечисленных факторов на качество ПС зависит, прежде всего, от его назначения и требований к функциям. Как отмечалось выше, множество характеристик качества программных средств можно разделить на две принципиально различающихся группы:

- функциональные характеристики (функциональность) - определяющие назначение, свойства и задачи, решаемые комплексом программ для основных пользователей, отличающиеся очень широким спектром и разнообразием, состав и специфику которых трудно унифицировать и можно категоризировать только по большому количеству классов и свойств ПС;
- конструктивные характеристики качества, номенклатура которых может быть унифицирована, адаптирована и использована для описания остальных, внутренних и внешних, стандартизируемых характеристик качества, поддерживающих и улучшающих реализацию основных, функциональных требований к качеству объектов и процессов ЖЦ программных средств.

Определение и сравнение ***функционального качества программ*** целесообразно рассматривать в пределах ограниченных классов ПС, выполняющих подобные функции. Такие классы функций могут выделяться в пределах крупных проблемно-ориентированных сфер применения (административные, банковские, медицинские, авиационные и т.п.), и для решения более мелких, специальных, функциональных задач в этих областях. Каждая из таких задач может быть описана рядом специфических свойств, характеристик и атрибутов, полная номенклатура которых содержит многие тысячи названий, мер и шкал, которые трудно или невозможно унифицировать. Функциональные характеристики и их параметры могут подвергаться значительным модификациям в течение всего ЖЦ ПС и являются обычно наиболее динамичными компонентами из всех характеристик качества.

Функциональная пригодность (см. стандарт ISO-9126) непосредственно определяет основное назначение и функции ПС для пользователей. В контракте и техническом задании для каждого проекта, она должна быть выделена и формализована для однозначного понимания и оценивания всеми партнерами на каждом этапе ЖЦ и при значительных модификациях задач ПС. В силу своей специфичности, при последующем изложении функциональная пригодность обозначается как основная цель и главная характеристика для всего множества типов ПС.

Вторая группа характеристик - ***конструктивных***, играет подчиненную роль и должна, в первую очередь, поддерживать и обеспечивать высокое качество реализации функций ПС и его применения по основномуназначению. Номенклатура этих характеристик относительно не велика, и стандартами рекомендуется в составе: корректности, способности к взаимодействию, защищенности, надежности, ресурсной эффективности, практичности, сопровождаемости и мобильности. Их выбор и значения определяются требованиями к функциональной пригодности ПС. Исходная номенклатура этой группы характеристик, субхарактеристик и их атрибутов

практически инвариантна к функциям ПС и стандартизирована, во взаимосвязи со стандартами на жизненный цикл комплексов программ при регламентировании их этапов и процессов. Для каждого конкретного проекта ПС из них может быть выделена представительная группа, наиболее важных и оказывающих наибольшее влияние на решение определенных функциональных задач.

РЕСУРСЫ ДЛЯ ЖИЗНЕННОГО ЦИКЛА СЛОЖНЫХ ПРОГРАММНЫХ СРЕДСТВ

Доступные ресурсы. Сценарии, используемые при экономическом анализе проектов ПС.

Общее понятие - ***доступные ресурсы*** жизненного цикла ПС включает реальные финансовые, временные, кадровые и аппаратурные ограничения, в условиях которых происходит создание и совершенствование комплексов программ. В зависимости от характеристик объекта разработки на ее выполнение выделяются ресурсы различных видов и объема. Эти факторы проявляются как дополнительные характеристики программных продуктов и их рентабельности, которые следует учитывать и оптимизировать, начиная с системного анализа ЖЦ ПС. В результате доступные ресурсы становятся косвенными критериями или факторами, влияющими на выбор методов разработки, на достигаемые качество и эффективность применения ПС. Многие проекты информационных систем терпели и терпят неудачу из-за отсутствия у разработчиков и заказчиков при подготовке контракта четкого представления о реальных финансовых, трудовых, временных иных ресурсах, необходимых для их реализации. Поэтому одной из основных задач при системном проектировании ПС является ***экономический анализ и определение необходимых ресурсов для создания и всего ЖЦ ПС*** в соответствии с требованиями контракта и технического задания.

Наиболее общим видом ресурсов, используемых в жизненном цикле ПС, являются ***допустимые финансово-экономические затраты*** или эквивалентные им величины трудоемкости соответствующих работ. При разработке, тестировании и анализе качества этот показатель может применяться или как вид ресурсных ограничений, или как оптимизируемый критерий, определяющий целесообразную функциональную пригодность ПС. При этом необходимо также учитывать затраты на разработку, закупку и эксплуатацию системы качества, на технологию и комплекс автоматизации проектирования программ и баз данных, которые могут составлять существенную часть совокупной стоимости и трудоемкости разработки и всего ЖЦ ПС.

Время или допустимая длительность разработки определенных версий ПС является "невосполнимым ограниченным ресурсом реальных проектов. Этот ресурс все больше определяет достижимое качество

комплексов программ в процессе их разработки и сопровождения. Высокие требования заказчиков к сжатым срокам реализации проектов, естественно, ограничивают разработчиков и испытателей в продолжительности и объеме возможного системного анализа и проектирования, разработки и, особенно, тестирования программ. Увеличение числа, привлекаемых для этого специалистов, при опытной эксплуатации или тестировании, только в некоторых пределах позволяет ускорять разработку и увеличивать совокупное число тестов при проверках, для повышения качества программ.

Кадры специалистов можно оценивать численностью, а также тематической и технологической квалификацией, которые всегда ограничены. В создании крупномасштабных ПС участвуют системные аналитики и руководители различных рангов, программисты и вспомогательный обслуживающий персонал в некотором, желательно, рациональном сочетании. Определяющими являются совокупная численность и структура коллектива, а также его подготовленность к коллективной разработке конкретного типа ПС и к применению им системы обеспечения качества функционирования.

Доступные разработчикам ПС вычислительные ресурсы объектных и технологических ЭВМ являются одним из важнейших факторов, определяющим достижимое качество сложных ПС. В процессе проектирования целесообразно выделять определенные ресурсы ЭВМ на оперативное обеспечение качества, повышение защищенности и надежности функционирования. Допустимая величина и рациональное распределение ресурсов ЭВМ на отдельные методы улучшения определенных конструктивных характеристик качества ПС, оказывают существенное влияние на достигаемые их значения.

Обобщенными ресурсами проекта ПС являются доступные стоимость или совокупные трудовые, временные и материальные затраты, необходимые для приобретения, создания, модификации и эксплуатации компонентов и всего комплекса программ. Эти характеристики непосредственно влияют практически на все показатели качества и определяют рентабельность покупки или создания заново конкретного программного продукта. Это означает, что качество является относительным понятием, которое зависит от ресурсов и субъектов, осуществляющих его оценку с позиции эффективности использования, а также от состояния рынка соответствующей продукции, ее производителей и технологий. Ориентация на потребителей подразумевает анализ их нужд и определение возможностей рынка удовлетворить эти потребности. При этом следует учитывать рыночную конкуренцию двух видов: между поставщиками готовых к применению программных средств с фиксированным качеством и между разработчиками, способными обеспечить жизненный цикл ПС или его существенную часть, с характеристиками качества, требующимися конкретному заказчику. В последнем случае на требуемое качество могут оказывать влияние не только заказчик и непосредственные пользователи, но

и различные посредники, организационные и торговые структуры, а также исполнители проекта.

В начале проектирования ПС всегда возникает задача оценивания ресурсов, которые необходимы и доступны для создания и обеспечения всего ЖЦ ПС, а также возможной экономической эффективности последующего применения комплекса программ по назначению при условии реализации требуемых характеристик качества. Экономическая эффективность и затраты имеют самостоятельное значение и методологию при анализе ЖЦ ПС. При планировании проектов программных средств, часто инициатором разработки является разработчик-поставщик, который самостоятельно принимает все решения о проектировании за счет собственных ресурсов и предполагает возместить затраты при реализации ПС на рынке. В других случаях имеется определенный заказчик потребитель, который способен задать основные цели, характеристики качества и обеспечить ресурсы для реализации проекта. Таким образом, ***при экономическом анализе проектов ПС возможны два сценария***:

- создание и весь жизненный цикл комплекса программ и/или базы данных ориентируется разработчиком на массовое тиражирование и распространение на рынке, для заранее не известных покупателей-пользователей в различных сферах применения, при этом отсутствует приоритетный внешний потребитель-заказчик, который определяет и диктует основные требования, а также финансирует проект;
- разработка проекта ПС и/или БД предполагается поставщиком разработчиком для конкретного потребителя-заказчика, который его финансирует, с определенным, необходимым ему тиражом и известной, ограниченной областью применения результатов разработки.

Первый сценарий базируется на маркетинговых исследованиях рынка программных продуктов и на стремлении поставщика занять на рынке достаточно выгодное место, обеспечивающее ему необходимую прибыль. Важнейшим ***фактором конкурентоспособности ПС является соотношение между ценностью*** имеющегося или предполагаемого продукта с позиции его использования потребителем, и стоимостью его при создании или приобретении в условиях реального рынка. Для этого ему нужно определить наличие на рынке гаммы близких по назначению ПС, оценить их экономическую эффективность, стоимость и применяемость, а также возможную конкурентоспособность предполагаемого программного продукта для потенциальных пользователей и их возможное число. Кроме того, следует оценить рентабельность затрат на обеспечение всего ЖЦ нового ПС и выявить функциональные и конструктивные характеристики качества, которые способны привлечь достаточно покупателей и оправдать затраты на предстоящую разработку.

Для удовлетворения потребностей пользователей, необходимы их затраты на приобретение готового или на заказ разработки и обеспечения жизненного цикла, соответствующего программного продукта. При этом особое значение имеет системное проектирование и технико-экономическое

обоснование всего жизненного цикла ПС. Поэтому значительное внимание необходимо уделять разработке концепции, технического задания и спецификаций, когда должен быть выбран первичный набор характеристик качества и их значений, который в последующем следует конкретизировать, развивать и реализовать в течение ЖЦ ПС.

Потенциальные покупатели-пользователи перед приобретением ПС обычно оценивают конкурентоспособность новой продукции на рынке ***по величине отношения***:

- возможной экономической ***эффективности*** (ценности) применения ПС и способности удовлетворить пользователями свои потребности с необходимым качеством при его использовании;
- к ***стоимости*** (цене), которую требуется заплатить пользователям при приобретении и эксплуатации данного комплекса программ или базы данных.

При выборе предполагаемого продукта и поставщика, покупатель стремится максимизировать это отношение, как за счет поиска ПС с наилучшими функциями, эффективностью и высокими характеристиками качества, так и за счет минимальной или рациональной стоимости покупаемого продукта. В этом сценарии при организации проектирования вся ответственность за цели, характеристики качества и рентабельность проекта ложится на его руководителей, и особую роль должны играть специалисты по маркетинговому анализу на рынке аналогичных продуктов. Они должны оценивать риск успешного продвижения создаваемого продукта на рынок, сроки и график выполнения этапов жизненного цикла, потребность и достаточность ресурсов для реализации проекта, а также перспективы длительного развития, модификаций и распространения версий ПС. Отбраковка вариантов реализации ПС ведется по показателю эффективность/стоимость для пользователей, с учетом прогнозов конкурентоспособности и возможностей распространения на рынке.

Такие проекты чаще относительно не велики по объему и срокам реализации первой версии, однако могут предполагать длительный ЖЦ и множество модификаций для адаптации к нуждам и среде различных пользователей. При этом должны обязательно учитываться затраты ресурсов на непосредственную разработку и обеспечение всего ЖЦ ПС, и возможная рентабельность проекта с учетом прогноза его жизненного цикла и распространения на рынке. Для этого при системном проектировании разработчикам необходимо прогнозировать затраты на создание и весь ЖЦ ПС так, как это делается при втором сценарии.

Второй сценарий предполагает наличие определенного заказчика-потребителя проекта ПС, который определяет основные технические и экономические требования и характеристики. Он выбирает конкурентоспособного поставщика-разработчика, которого оценивает на возможность реализовать проект с необходимым качеством с учетом ограничения сроков, бюджета и других ресурсов. Этому помогает опыт и экономические характеристики ранее выполненных поставщиками проектов,

но некоторые проекты могут не иметь явных прецедентов, и тогда приходится использовать имеющуюся статистику в этой области. При этом предполагается, что результаты разработки не обязательно подлежат широкому тиражированию, могут не поступать на открытый рынок вследствие чего, маркетинговые исследования для таких проектов не являются доминирующими и обычно предварительно могут не проводиться.

Однако для заказчика и разработчика перед заключением контракта необходим достаточно достоверный системный анализ, прогнозированием технико-экономическое обоснование (ТЭО) требуемых ресурсов по трудоемкости, стоимости, срокам и другим характеристикам. Противоположность экономических интересов поставщика и потребителя при оценивании стоимости и необходимых ресурсов проекта, требует поиска компромисса, при котором разработчик не продешевит, а заказчик не переплатит за конкретные выполненные работы и весь проект. Поэтому оба партнера заинтересованы в достоверном экономическом прогнозировании затрат ресурсов на проект ПС.

Жизненный цикл ПС можно разделить на ***две части***, существенно различающиеся экономическими особенностями процессов, характеристиками и влияющими на них факторами. ***В первой части Ж***Ц производятся системный анализ, проектирование, разработка, тестирование и испытания базовой версии ПС. Номенклатура работ, их трудоемкость, длительность и другие экономические характеристики на этих этапах ЖЦ существенно зависят от характеристик объекта, технологии и инструментальной среды разработки. Особенно важно учитывать возможное возрастание суммарных затрат при завышении требований к качеству программного продукта. Как и для других видов промышленной продукции, улучшение качества комплексов программ обычно достигается не пропорциональным, а более значительным возрастанием требуемых для этого ресурсов. Сокращение этой потребности в ресурсах часто возможно только за счет принципиального изменения и совершенствования технологии проектирования и разработки.

Многие молодые программисты - "художники" лихо утверждают, что они могут "писать" программы со скоростью тысяч строк в неделю. Однако такие программы способны решать относительно небольшие, автономные задачи с неизвестным и, как правило, низким качеством, не поддерживаются полноценной документацией и не соответствуют требованиям стандартов на программный продукт. Исследования последних десятилетий показали, что при разработке крупномасштабных комплексов программ ***реальная средняя производительность*** разработчиков почти на два порядка ниже таких оценок и измеряется только несколькими десятками строк в неделю. При этом средняя стоимость разработки одной строки полностью новой программы высокого качества, составляет свыше 10$ и достигает 100$. При таком анализе, естественно, должен учитываться весь цикл разработки, начиная от подготовки технического задания, до завершения успешных квалификационных испытаний, а также весь коллектив участников проекта,

включая руководителей, системных аналитиков и вспомогательный персонал. Непосредственным "написанием" текстов программ в коллективе занимается только треть или четверть разработчиков и почти столько же их автономным и квалификационным тестированием.

Ориентирами для оценивания необходимых ресурсов трудоемкости, длительности и числа специалистов по крупным этапам работ, при создании сложных ПС высокого качества, могут служить данные, опубликованные в [5,6]. В табл. 1 представлен вариант, относительного распределения затрат ресурсов по этапам работ для сложных встроенных ПС реального времени, который можно использовать как базу для приближенных оценок. Соотношение затрат по этапам в процентах достаточно инвариантно для различных по объему и сложности проектов и для ориентировочных оценок может быть пересчитано в реальные значения с учетом размера ПС, доступных ресурсов и средней производительности труда в фирме.

Максимум трудоемкости и числа специалистов приходится на этапы программирования и тестирования компонентов, когда привлекается основная масса программистов-кодировщиков для разработки компонентов ПС. Характерно, что продолжительность всех выделенных этапов, более или менее, одинаковая и имеет тенденцию уменьшаться на средних этапах проекта. Полную длительность цикла разработки наиболее трудно оценивать, и при планировании, почти всегда, она значительно занижается, причем ошибка часто составляет 30-50%. При активном использовании и совершенствовании технологий системного анализа и проектирования, происходит перераспределение всех видов затрат в сторону увеличения трудоемкости начальных этапов разработки. Это дает значительное снижение использования совокупных ресурсов для всего проекта. Менее изучены распределения необходимых ресурсов по этапам работ, с учетом реализации требуемых конкретных характеристик качества ПС. Опубликованные данные и зависимости для различных классов ПС, позволяют приближенно прогнозировать совокупные затраты и другие основные технико-экономические показатели (ТЭП), планы и графики работ при системном анализе вновь создаваемых проектов.

Относительная трудоемкость, длительность и число специалистов при разработке сложных программных средств

Табл. 2.

Этапы жизненного цикла	Относительная трудоемкость этапа работ (%)	Относительная длительность этапа работ (%)	Относительное число специалистов на этапе (%)
1. Анализ требований к программам и планирование	8	22	25
2. Архитектурное	16	22	40

проектирование программного средства			
3. Детальное проектирование программного средства	26	18	60
4. Кодирование и тестирование программных компонентов	28	18	100
5. Интеграция, квалификационное тестирование и испытания программного средства	22	20	70

Вторая часть ЖЦ, отражающая эксплуатацию, сопровождение, модификацию и перенос ПС на иные платформы, в меньшей степени связана с функциональными характеристиками объекта и среды разработки. Номенклатура работ на этих этапах более или менее определенная, но их трудоемкость и длительность могут сильно варьироваться, в зависимости от массовости и других внешних факторов распространения и применения версий ПС. Успех ПС у пользователей и на рынке, а также будущий процесс развития версий трудно предсказать, и он не связан напрямую с экономическими параметрами процессов разработки ПС. Определяющими становятся потребительские характеристики ПС, а их экономические особенности с позиции разработчиков и вложенные затраты на очередную версию отходят на второй план (см. первый сценарий).

Вследствие этого в широких пределах могут изменяться трудоемкость и число специалистов, необходимое для поддержки этих этапов ЖЦ. Это затрудняет статистическое обобщение ТЭП различных проектов и прогнозирование на их основе аналогичных характеристик новой разработки. Поэтому планы на этих этапах имеют характер общих взаимосвязей содержания работ, которые требуют распределения во времени, индивидуально для каждого проекта. В результате планирование трудоемкости и длительности этапов приходится производить итерационно на базе накопления опыта и анализа развития конкретных версий ПС, а также их успеха на рынке.

СТАНДАРТЫ, РЕГЛАМЕНТИРУЮЩИЕ КАЧЕСТВО ПРОГРАММНЫХ

СРЕДСТВ

Стандарт ISO 9126:1991 - Оценка программного продукта. Основные факторы, определяющие качество сложных программных средств. Внутренние метрики. Внешние метрики. Метрики качества в использовании.

Стандарт ISO 9126:1991 - Оценка программного продукта. Характеристики качества и руководство по их применению - является основой формального ***регламентирования характеристик качества*** ПС. Развитие этого международного стандарта проводится в направлении уточнения, детализации и расширения, описаний характеристик качества комплексов программ. Для замены редакции 1991 года завершается разработка и формализован проект стандарта, состоящего из четырех частей **ISO 9126:1-4**. Стандарт ISO 9126:1991 предполагается заменить, на две взаимосвязанные серии стандартов: ISO 9126:1-4 (проект) - Качество программных средств - и утвержденный стандарт ISO 14598:1-6:1998-2000 - Оценивание программного продукта. Проект нового стандарта ISO 9126 состоит из следующих частей под общим заголовком - ***Информационная технология - Качество программных средств***:

Часть 1: Модель качества.

Часть 2: Внешние метрики качества.

Часть 3: Внутренние метрики качества.

Часть 4: Метрики качества в использовании.

Часть первая стандарта ISO 9126-1 - (пересмотренная и расширенная редакция ISO 9126:1991), сохранила практически ту же номенклатуру нормативных характеристик качества программных средств. В ней приводится схема взаимосвязи частей стандарта ISO 9126 и частей стандарта ISO 14598, а также область применения, нормативные ссылки, термины и определения. Модель характеристик качества ПС состоит из шести групп базовых показателей, каждая из которых детализирована несколькими нормативными субхарактеристиками:

Функциональная пригодность детализируется:

- пригодностью для применения;
- корректностью (правильностью, точностью);
- способностью к взаимодействию;
- защищенностью.

Надежность характеризуется:

- уровнем завершенности (отсутствия ошибок);
- устойчивостью к дефектам;
- восстанавливаемостью;
-доступностью-готовностью.

Эффективность рекомендуется отражать:

- временной эффективностью;
- используемостью ресурсов.

Применимость (практичность) предлагается описывать:

-понятностью;
- простотой использования;
- изучаемостью;
- привлекательностью.

Сопровождаемость представляется:
- удобством для анализа;
- изменяемостью;
- стабильностью;
- тестируемостью.

Переносимость (мобильность) предлагается отражать:
- адаптируемостью;
- простотой установки - инсталляции;
- сосуществованием - соответствием;
- замещаемостью.

Дополнительно каждая характеристика сопровождается субхарактеристикой ***согласованность***, которая должна отражать отсутствие противоречий с иными стандартами и нормативными документами, а также с другими показателями в данном стандарте. Характеристики и субхарактеристики в этой части стандарта определены очень кратко (2-3 строки), без комментариев и подробных рекомендаций по их применению к конкретным системам и проектам. Материалы имеют концептуальный характер и не содержат рекомендаций по выбору и упорядочению приоритетов необходимого минимума критериев в зависимости от особенностей объекта среды разработки и применения. Кроме того, отсутствуют методики измерения характеристик и сопоставления с требованиями спецификаций, а также рекомендации, на каких этапах ЖЦ ПС их целесообразно применять.

Описания показателей качества ориентированы на высококвалифицированных системных аналитиков и заказчиков ПС, которым предоставляет возможность выбирать необходимую номенклатуру и способ оценивая характеристик в соответствии с назначением, областью применения и конкретными особенностями создаваемых объектов.

ОСНОВНЫЕ ФАКТОРЫ, ОПРЕДЕЛЯЮЩИЕ КАЧЕСТВО СЛОЖНЫХ ПРОГРАММНЫХ СРЕДСТВ

Общее представление о качестве ПС международным стандартом ISO 9126 рекомендуется отражать тремя взаимодействующими и взаимозависимыми метриками характеристик качества, отражающими:

- внутреннее качество, проявляющееся в процессе разработки и других промежуточных этапов жизненного цикла ПС;
- внешнее качество, заданное требованиями заказчика в спецификациях и отражающееся характеристиками конечного продукта;

- качество при использовании в процессе нормальной эксплуатации и результативностью достижения потребностей пользователей с учетом затрат ресурсов.

Эти типы метрик применимы при определении целей проекта и требований к качеству ПС, включая промежуточные компоненты и продукты. Подходящие внутренние атрибуты качества ПС являются предпосылкой достижения в жизненном цикле требуемого внешнего поведения, а приемлемое внешнее поведение - предпосылка достижения качества в использовании.

Внутренние метрики в соответствии со стандартами могут применяться в ходе проектирования и программирования к неисполняемым компонентам ПС таким, как спецификация или исходный программный текст. При разработке ПС промежуточные компоненты следует оценивать с использованием внутренних метрик, которые отражают некоторые функциональные и конструктивные свойства программ. Основная цель применения внутренних метрик – обеспечение требуемого внешнего качества. Рекомендуется использовать внутренние метрики, которые имеют наиболее сильные связи с приоритетными внешними метриками, чтобы они могли помогать при прогнозировании их достижимых значений.

Внутренние метрики дают возможность разработчикам, испытателям и заказчикам, начиная с системного проектирования, прогнозировать качество жизненного цикла программ и заниматься вопросами технологического обеспечения качества до того, как ПС становится готовым к использованию продуктом. Измерения внутренних метрик используют свойства, категории, числа или характеристики элементов из состава ПС, которые, например, имеются в процедурах исходного программного текста, в графе потока управления, в потоке данных и в описаниях изменения состояний памяти. Качество документации также может оцениваться с использованием внутренних метрик.

Внешние метрики используют меры ПС, выведенные из поведения системы, частью которых они являются, путем испытаний, эксплуатации и наблюдения исполняемых программ или функционирования информационной системы. Перед приобретением или использованием ПС его следует оценить с использованием метрик, основанных на реализации деловых и профессиональных целей, связанных с применением программного продукта в определенной организационной и технической среде. Внешние метрики обеспечивают заказчикам, пользователям и разработчикам возможность прослеживать и анализировать качество ПС в ходе испытаний или опытной эксплуатации. Подходящие внешние метрики специфицируются для получения числовых значений или категорий и свойств внутренних характеристик качества, чтобы их можно было использовать для проверки того, что промежуточные продукты в процессе разработки удовлетворяют внутренним спецификациям качества.

Метрики качества в использовании отражают, в какой степени продукт удовлетворяет потребности конкретных пользователей в достижения

заданных целей. Эта метрика не отражена в числе шести базовых характеристик ПС, регламентируемых стандартом ISO 9126-1 вследствие ее общности, однако рекомендуется для интегральной оценки результатов функционирования и применения комплексов программ в стандарте ISO 9126-4. Качество в использовании - это объединенный эффект функциональных и конструктивных характеристик качества ПС для пользователей. Связь качества в использовании с другими характеристиками ПС зависит от ***задач и функций их потребителей***:

- для заказчика требуется полное соответствие характеристик программного продукта условиям контракта, технического задания и спецификаций требований;
- для конечного оперативного пользователя ПС по основному назначению, качество в использовании обусловливают, в основном, характеристики функциональных возможностей, надежности, практичности и эффективности;
- для персонала сопровождения ПС качество в использовании определяется преимущественно сопровождаемостью;
- для персонала, выполняющего перенос ПС на иные платформы, а также инсталляцию и адаптацию к среде применения, качество в использовании определяется, прежде всего, мобильностью.

Практически невозможно измерить все внутренние или внешние субхарактеристики и их атрибуты для всех компонентов крупномасштабных ПС.

Аналогично, обычно не практикуется формализовать требования и оценивать качество в использовании для всех возможных сценариев задач пользователей. Поэтому их необходимо ранжировать и выделять приоритетные процессы и объекты для оценивания характеристик с различной достоверностью.

Для выбора характеристик качества ПС и достоверного сравнения их с требованиями, а также для сопоставления их значений между различными программными продуктами необходимы оценки, измерения и использование определенных мер и шкал. Стандартами рекомендуется, чтобы было предусмотрено измерение каждой характеристики качества ПС (субхарактеристики или ее атрибута) с точностью и определенностью, достаточной для сравнений с требованиями технических заданий и спецификаций, и чтобы измерения были объективны и воспроизводимы. Следует предусматривать нормы допустимых ошибок измерения, вызванных инструментами и/или ошибками человека-эксперта. Чтобы измерения были объективными, должна быть документирована и согласована процедура для присвоения числового значения, свойства или категории каждому атрибуту программного продукта. Процедуры измерений должны давать в результате одинаковые меры с приемлемой устойчивостью, получаемые различными субъектами при выполнении одних и тех же измерений характеристик ПС в различных случаях.

Характеристики, субхарактеристики и атрибуты качества ПС с позиции возможности и точности их измерения можно разделить на ***три уровня детализации показателей***, особенности которых следует уточнять при их выборе:

- *категорийные-описательные*, отражающие набор свойств и общие характеристики объекта - его функции, категории ответственности, защищенности и важности, которые могут быть представлены номинальной шкалой категорий-свойств;
- *количественные*, представляемые множеством упорядоченных числовых точек, отражающих непрерывные или дискретные закономерности и описываемые интервальной или относительной шкалой, которые можно объективно измерить и численно сопоставить с требованиями;
- *качественные* - содержащие несколько упорядоченных или отдельных свойств - категорий, которые характеризуются порядковой или точечной шкалой набора категорий (есть - нет, хорошо - плохо), устанавливаются, выбираются и оцениваются в значительной степени субъективно и экспертно.

К первому уровню относятся показатели качества, которые характеризуются наибольшим разнообразием значений - свойств программ и наборов данных и охватывают весь спектр классов, назначений и функций современных ПС. Эти свойства можно сравнивать только в пределах однотипных ПС и трудно упорядочивать по принципу предпочтительности. Среди стандартизированных показателей качества к этой группе, прежде всего, относится ***Функциональная пригодность***, являющаяся самой важной и доминирующей характеристикой любых ПС. Номенклатура и значения всех остальных показателей качества непосредственно определяются требуемыми функциями программного средства и, в той или иной степени, влияют на выполнение этих функций. Поэтому выбор функциональной пригодности ПС, подробное и корректное описание ее свойств являются основными исходными данными для установления при системном проектировании требуемых значений всех остальных стандартизированных показателей качества.

Ко второму уровню показателей качества относятся достаточно достоверно и объективно измеряемые численные характеристики ПС. Значения этих характеристик обычно в наибольшей степени влияют на функциональную пригодность и метрики в использовании ПС. Поэтому выбор и обоснование их требуемых значений должно проводиться наиболее аккуратно и достоверно уже при проектировании ПС. Их субхарактеристики могут быть описаны упорядоченными шкалами объективно измеряемых значений, требуемые численные величины которых могут быть установлены и выбраны заказчиками или пользователями ПС. Такими характеристиками являются надежность и эффективность комплексов программ. Надежность может отражаться временем наработки на отказ, средним временем восстановления, а также коэффициентом готовности – вероятностью застать ПС в работоспособном состоянии при нормальной эксплуатации. Эти

величины могут выбираться и фиксироваться в техническом задании или спецификации требований, и сопровождаться методикой объективных, численных измерений при квалификационных испытаниях для сопоставления с требованиями.

Атрибуты временной эффективности тесно связаны между собой и также значительно влияют на функциональную пригодность ПС. Длительность решения основных задач, пропускная способность по числу их решений за некоторый интервал времени, длительность ожидания результатов (отклика), и некоторые другие характеристики динамики функционирования ПС, могут быть выбраны и установлены количественно в спецификациях требований заказчиком. Эта субхарактеристика не всегда может быть выбрана и достаточно точно зафиксирована в требованиях на начальных этапах разработки, но она может количественно измеряться и последовательно уточняться в жизненном цикле ПС.

Третий уровень стандартизированных показателей качества ПС трудно полностью описать измеряемыми количественными значениями и их некоторые субхарактеристики и атрибуты имеют описательный, качественный вид. В зависимости от функционального назначения ПС по согласованию с заказчиком можно определять экспертно степень необходимости (приоритет) этих свойств и бальные значения уровня реализации их атрибутов в жизненном цикле конкретного ПС. Например, не всегда может требоваться мобильность программ на иные операционные и аппаратные платформы. В других случаях мобильность можно оценивать категориями: отличная, хорошая, удовлетворительная или неудовлетворительная. Такие оценки могут проводиться экспертно на основе анализа возможной трудоемкости и длительности, реализации процессов переноса комплекса программ на новую платформу.

Практичность тесно связана с функциональной пригодностью. Обобщенно этот показатель можно отразить трудоемкостью и длительностью, которые необходимы для изучения и полного освоения функций и технологии применения соответствующего ПС. Каждая из субхарактеристик практичности имеет ряд качественных атрибутов, которые могут выбираться и оцениваться экспертно с учетом функционального назначения ПС, а также надежности и ресурсной эффективности комплекса программ. Некоторые из этих атрибутов можно квалифицировать количественно.

Сопровождаемость может иметь ограниченный характер полной замены программ на вновь разработанные версии и тем самым сливаться с процессами разработки или осуществляться как непрерывная поддержка множества пользователей консультациями, адаптациями и корректировками программ. В зависимости от этого различаются функции и трудоемкость процессов сопровождения, которая может использоваться как обобщенная качественная характеристика при выборе требований к этому показателю качества. Соответственно при проектировании качественно могут быть

установлены субхарактеристики сопровождаемости и описаны требуемые их свойства.

При любом виде деятельности людям свойственно непредумышленно ошибаться, результаты чего проявляются в процессе создания или применения изделий или систем. В общем случае под ошибкой подразумевается дефект, погрешность или неумышленное искажение объекта или процесса. При этом предполагается, что известно его правильное, эталонное состояние, по отношению, к которому может быть определено наличие отклонения - ***дефекта или ошибки***. Для систематической, координированной борьбы с ними необходимы исследования факторов, влияющих на качество ПС со стороны различных, существующих и потенциально возможных дефектов в конкретных программах. Это позволит целенаправленно разрабатывать комплексы методов и средств обеспечения качества сложных ПС различного назначения при реально достижимом снижении уровня дефектов проектирования и разработки.

Различия между ожидаемыми и полученными результатами функционирования программ могут быть следствием ошибок не только в созданных программах и данных, но и ***системных ошибок в первичных требованиях спецификаций***, явившихся исходной базой при создании ПС. Тем самым проявляется объективная реальность, заключающаяся в невозможности абсолютной корректности исходных спецификаций сложных ПС после проектирования. На практике в процессе разработки ПС исходные требования уточняются и детализируются по согласованию между заказчиком и разработчиком. Базой таких уточнений являются неформализованные представления и знания специалистов, а также результаты промежуточных этапов жизненного цикла. Однако установить ошибочность исходных данных и спецификаций еще труднее, чем обнаружить ошибки в созданных программах, так как принципиально отсутствуют формализованные данные, которые можно использовать как эталонные, и их заменяют неформализованные представления заказчиков и разработчиков.

Дефекты функционирования программных средств, не имеющие злоумышленных источников или последствий физических разрушений аппаратных компонентов, проявляются внешне как случайные, имеют разную природу и последствия. В частности, они могут приводить к нарушениям функциональной работоспособности, и к отказам при использовании ПС. В жизненном цикле, на ПС воздействуют различные негативные, дестабилизирующие факторы, которые можно разделить на внутренние, присущие самим объектам уязвимости, и внешние, обусловленные средой, в которой эти объекты функционируют.

Введение строгих количественных метрик в программирование должно было способствовать решению ряда практических задач:

- предсказывать вероятное число ошибок в системе с самого начала проектирования; - на основе анализа фазы проектирования системы предсказывать уровень сложности последующего сопровождения;

- на основе анализа исходного кода программ прогнозировать уровень сложности процессов тестирования и процент остающихся ошибок; - по оценкам сложности фазы проектирования системы определять конечный размер кода;
- определять корреляцию отдельных характеристик программного кода с качеством готовой системы;
- контролировать стадии развития проекта;
- анализировать явные и скрытые дефекты;
- на основе экспериментального сравнения выявлять лучшие методы и технологии.

По мере роста актуальности программных метрик на рынке стали появляться различные "измерительные" программы. Одни из них исследовали характеристики проектов и ПО комплексно, другие ориентировались на вполне конкретные цели: анализ исходного кода, размеров и структуры отдельных модулей.

ХАРАКТЕРИСТИКИ КАЧЕСТВА БАЗ ДАННЫХ

Функциональные и конструктивные характеристики качества.

Современные базы данных (БД) являются одними из массовых специфических объектов в сфере информатизации, для которых в ряде областей необходимо особенно высокое качество и его квалифицированное системное проектирование. Естественно возникают вопросы, что означает качество таких объектов, какие требования следует предъявлять к их качеству, какими характеристиками нужно описывать качество, как их задавать и оценивать. Для этого полезны, как прототипы, методы и стандарты, разработанные для анализа качества сложных программных средств.

Базу данных можно рассматривать как два компонента: ***систему программ управления данными и совокупность данных, упорядоченных по некоторым правилам***. Поэтому при анализе качества базу данных целесообразно делить на два компонента:

- программные средства системы управления базой данных (СУБД), независимые от сферы их применения, структуры и смыслового содержания накапливаемых и обрабатываемых данных;
- информацию базы данных (ИБД), доступную для накопления, упорядочивания, обработки и использования в конкретной проблемно-ориентированной сфере применения.

При комплексном анализе качества баз данных, не всегда удается четко разделить требования и значения характеристик качества для каждого из этих объектов. При этом одна и та же система управления базой данных (СУБД) может обрабатывать различные по структуре, составу и содержанию данные, а одни и те же данные могут управляться программными средствами различных СУБД. Хотя эти компоненты тесно взаимодействуют при

реализации конкретной прикладной БД, первоначально при проектировании они создаются или выбираются практически независимо и могут рассматриваться в их жизненном цикле (ЖЦ) как два объекта, которые различаются:

- номенклатурой и содержанием показателей качества, определяющих их назначение, функции и потребительские свойства;
- технологией и средствами автоматизации разработки и обеспечения всего ЖЦ каждого объекта;
- категориями специалистов, обеспечивающих: создание, эксплуатацию или применение компонентов БД;
- комплектами эксплуатационной и технологической документации, поддерживающими жизненный цикл объектов.

Первым компонентом для системного анализа и требований к качеству является комплекс программ СУБД. Практически весь набор характеристик и атрибутов качества ПС, изложенный в стандарте ISO- 9126, в той или иной степени, может использоваться при формировании требований к качеству СУБД. Особенности состоят в адаптации и изменении акцентов при выборе и упорядочении этих показателей. Во всех случаях важнейшими характеристиками качества СУБД являются требования функциональной пригодности для процессов формирования и изменения информационного наполнения БД администраторами, а также доступа к данным и представления результатов пользователям БД. Качество интерфейса специалистов с БД, обеспечиваемого средствами СУБД, определяется, в значительной степени, субъективно, однако имеется ряд характеристик, которые можно оценивать достаточно корректно.

Различия требований к характеристикам качества привели к созданию весьма широкого спектра локальных, специализированных и распределенных СУБД. Значения ряда показателей качества ПС, составляющих СУБД, существенно зависят от характеристик и организации информации в БД. Специализированные СУБД характеризуются относительно узкой сферой применения и более четким выделением группы требований к приоритетным показателям качества. В универсальных СУБД спектр характеристик качества шире, что позволяет соответственно расширять сферу применения конкретного типа СУБД. Однако и для них существуют области приоритетного, наиболее эффективного использования.

За основу принята номенклатура и содержание стандартизированных характеристик сложных комплексов программ, которые адаптируются применительно к понятиям и особенностям компонентов баз данных. В зависимости от конкретной проблемно-ориентированной области применения СУБД, приоритет при системном анализе требований качеству может отдаваться различным, конструктивным характеристикам: либо надежности и защищенности применения (финансовая сфера), либо удобству использования малоквалифицированными пользователями (социальная сфера), либо эффективности использования ресурсов (сфера материально-технического снабжения). Однако, практически во всех случаях сохраняется

некоторая роль ряда других конструктивных показателей качества. Для каждого из них необходимо анализировать и определять его приоритет для конкретной сферы применения, меры и шкалы необходимых и допустимых характеристик качества.

Вторым компонентом БД является собственно накапливаемая и обрабатываемая информация. В системах баз данных доминирующее значение приобретают сами данные, их хранение и обработка. Ниже сделан акцент на системный анализ требований и составляющих характеристик качества этого объекта - на информацию баз данных с предположением, что средства СУБД способны их обеспечить. Для оценивания качества информации БД может сохраняться общий, методический подход к выделению адекватной номенклатуры стандартизированных в ISO 9126 базовых характеристик и субхарактеристик качества ПС. Однако их содержание для применения к качеству ИБД при проектировании требуется уточнить и пояснить. Выделяемые показатели качества должны иметь практический интерес для пользователей БД и быть упорядочены в соответствии с приоритетами практического применения. Кроме того, каждый выделяемый показатель качества ИБД должен быть пригоден для достаточно достоверного оценивания или измерения, а также для сравнения с требуемым значением
при испытаниях.

При проектировании каждой БД в контракте, техническом задании и в спецификации должны селектироваться и формализоваться представительный набор функциональных требований к качеству ИБД, адекватный ее назначению и области применения, а также требованиям заказчика и потенциальных пользователей. Так же как для ПС, характеристики качества ИБД можно разделить на ***функциональные и конструктивные***. Их номенклатура, содержание и субхарактеристики ниже базируются на описаниях, рекомендуемых стандартом ISO 9126. Они представляются достаточно универсальными и применимыми ***для систематизации характеристик качества информации баз данных***. Тем самым может быть заложена основа для стандартизированного формирования требований к качеству баз данных. Однако номенклатура показателей качества не всегда может ограничиваться только характеристиками информации в БД, а должна включать ряд уточнений, отражающих комплексную эффективность и функциональную пригодность совместного применения СУБД и ИБД пользователями в реальных условиях.

Функциональная пригодности ИБД при системном проектировании может представлять сложную проблему для определения соответствия требований реальным значениям необходимых атрибутов качества особенно, для больших распределенных БД при циркулировании разнообразной и сложной информации об анализируемых объектах. Мерой качества функциональной пригодности может быть степень покрытия целей, назначения и функций БД доступной пользователям информацией. Так же как для ПС, для баз данных в составе функциональной пригодности

целесообразно использовать группу субхарактеристик, определяющих функциональные и структурные требования к базам данных, основное содержание которых подобно приведенным для ПС выше. Дополнительно функциональная пригодность многих ИБД может отражаться:

- полнотой накопленных описаний объектов – относительным числом объектов или документов, имеющихся в БД, к общему числу объектов по данной тематике или по отношению к числу объектов в аналогичных БД того же назначения;
- идентичностью данных - относительным числом описаний объектов, не содержащих дефекты и ошибки, к общему числу документов об объектах в ИБД;
- актуальностью данных - относительным числом устаревших данных об объектах в ИБД к общему числу накопленных и обрабатываемых данных.

Разнообразие назначения и функций ИБД ограничивает возможность стандартизации требований к ним только общими правилами их организации и структурирования, которые достаточно подробно изложены выше и далее не рассматриваются.

К ***конструктивным характеристикам качества информации БД*** в целом можно отнести, с некоторой корректировкой и уточнением понятий, субхарактеристик и атрибутов, практически все стандартизированные показатели качества ПС, которые представлены в ISO 9126. Требования к информации баз данных так же должны содержать особенности обеспечения ее надежности, эффективности использования ресурсов ЭВМ, практичности, применимости, сопровождаемости, мобильности. Содержание и атрибуты этих конструктивных характеристик в данном случае несколько отличаются от применяемых для программ, однако, их сущность, как базовых понятий и характеристик качества объектов, целесообразно использовать при проектировании для систематизации и регламентированного формирования требований к этим компонентам информационных систем. Меры и шкалы для оценивания конструктивных характеристик, в значительной степени, могут применяться те же, что при анализе качества программных средств.

Корректность или достоверность данных - это степень соответствия информации об объектах в БД реальным объектам вне ЭВМ в данный момент времени, определяющаяся изменениями самих объектов, некорректностями записей о их состоянии или некорректностями расчетов их характеристик. При системном проектировании выбор и установление требований к корректности данных в БД, можно оценивать ***по степени покрытия накопленными, актуальными и достоверными данными*** состояния и изменения внешних объектов, которые они отражают. Кроме того, к корректности БД можно отнести некоторые объемно-временные характеристики сохраняемых и обрабатываемых данных:

- объем базы данных - относительное число записей описаний объектов или документов в базе данных, доступных для хранения и обработки, по сравнению с полным числом реальных объектов во внешней среде;

- оперативность - степень соответствия динамики изменения описаний данных в процессе сбора и обработки, состояниям реальных объектов или величина допустимого запаздывания между появлением или изменением характеристик реального объекта, относительно его отражения в базе данных;
- глубина ретроспективы - максимальный интервал времени от даты выпуска и/или записи в базу данных самого раннего документа до настоящего времени;
- динамичность - относительное число изменяемых описаний объектов к общему числу записей в БД за некоторый интервал времени, определяемый периодичностью издания версий БД.

Защищенность информации БД реализуется, в основном, программными средствами СУБД, однако в сочетании с поддерживающими их средствами организации и защиты данных. Цели, назначение и функции защиты тесно связаны с особенностями функциональной пригодности каждой ИБД. При проектировании свойства защищать информацию баз данных от негативных воздействий описываются обычно составом и номенклатурой методов и средств, используемых для защиты от внешних и внутренних угроз. Косвенным показателем ее качества может служить относительная доля вычислительных ресурсов, используемых непосредственно средствами защиты информации БД.

Основное внимание в практике обеспечения безопасности применения БД сосредоточено на защите от злоумышленных разрушений, искажений и хищений информации баз данных. Основой такой защиты является аудит санкционирования доступа, а также контроль организации и эффективности ограничений доступа. В реальных БД возможны и не всегда учитываются катастрофические последствия и аномалии информации БД, отражающиеся на безопасности применения, при которых их источниками являются случайные, непредсказуемые, дестабилизирующие факторы или дефекты, и отсутствуют непосредственно заинтересованные лица в подобных нарушениях. Качество защиты ИБД можно характеризовать величиной предотвращенного ущерба, возможного при проявлении дестабилизирующих факторов и реализации конкретных угроз безопасности, а также средним временем между возможными проявлениями угроз, преодолевающих защиту данных.

Надежность информации баз данных может основываться на применении при системном проектировании понятий и методов теории надежности, которая позволяет получить ряд четких, измеряемых интегральных показателей их качества. Надежная ИБД, прежде всего, должна обеспечивать достаточно низкую вероятность потери работоспособности - отказа, в процессе ее функционирования в реальном времени. Быстрое реагирование на потерю или искажение данных и восстановление их достоверности и работоспособности за время меньшее, чем порог между сбоем и отказом, обеспечивают высокую надежность БД. Если в этих ситуациях происходит достаточно быстрое восстановление, такое что не

фиксируется отказ, то такие события не влияют на основные показатели надежности – наработку на отказ и коэффициент готовности ИБД. Непредсказуемость вида, места и времени проявления дефектов ИБД в процессе эксплуатации приводит необходимости создания специальных, дополнительных систем оперативной защиты от непредумышленных, случайных искажений данных. Надежность должна повышаться за счет средств обеспечения помехоустойчивости, оперативного контроля и восстановления ИБД.

Стандартом ISO 9126 рекомендуется анализировать и учитывать надежность комплексов программ четырьмя субхарактеристиками, которые могут быть применены также для формирования требований к характеристикам качества информации БД. Завершенность - свойство ИБД, состоящее в способности не попадать в состояния отказов вследствие потерь, искажений, ошибок и дефектов в данных. Они могут быть обусловлены не полным тестовым покрытием при испытаниях компонентов и ИБД в целом, а также недостаточной завершенностью их тестирования и защищенностью от искажений.

Устойчивость к дефектам и ошибкам - свойство ИБД автоматически поддерживать заданный уровень качества данных в случаях проявления дефектов и ошибок или нарушения установленного интерфейса по данным с внешней средой. Для этого в ИБД рекомендуется вводить оперативное обнаружение дефектов и ошибок информации, их идентификацию и автоматическое восстановление (рестарт) нормального функционирования ИБД. Относительная доля вычислительных ресурсов, используемых непосредственно для быстрой ликвидации последствий отказов и оперативного восстановления данных (рестарт) определяет значение устойчивости и снижается на повышении надежности ИБД.

Восстанавливаемость - свойство ИБД в случае отказа возобновлять требуемый уровень качества информации, а также корректировать поврежденные данные. Для этого необходимы вычислительные ресурсы и время на выявление неработоспособного состояния, диагностику причин отказа, а также на реализацию процессов восстановления. Основными показателями процесса восстановления данных являются его длительность и вероятностные характеристики ИБД в процессе ручного или автоматического их перезапуска - рестарта.

Доступность или готовность - свойство ИБД быть в состоянии полностью выполнять требуемую функцию в данный момент времени при заданных условиях использования информации базы данных. Обобщением характеристик отказов и восстановления производится в критерии коэффициент готовности ИБД. Этот показатель отражает вероятность иметь восстанавливаемые данные в работоспособном состоянии в произвольный момент времени. Нижние границы шкал атрибутов надежности могут быть отражены значениями, при которых использование конкретной ИБД становится неудобным, опасным или нерентабельным.

Эффективность использования ресурсов ЭВМ при системном анализе реального функционирования БД отражается временными характеристиками взаимодействия конечных пользователей и администраторов ИБД в процессе эксплуатации базы данных по прямому назначению.

Временная эффективность БД определяется длительностью выполнения заданных функций и ожидания результатов от ИБД в средних и/или наихудших случаях, с учетом приоритетов задач. Она зависит от объема, структуры и скорости обработки данных, влияющих непосредственно на интервал времени завершения конкретного вычислительного процесса, и от пропускной способности - производительности, т.е. от числа заданий, которое можно реализовать на данной ЭВМ в заданном интервале времени.

Используемость ресурсов или ресурсная экономичность в стандартах отражается занятостью ресурсов центрального процессора, оперативной, внешней и виртуальной памяти, каналов ввода-вывода, терминалов и каналов сетей связи. Эта величина определяется структурой, функциями и объемом ИБД, а также архитектурными особенностями и доступными ресурсами ЭВМ. В зависимости от конкретных задач и особенностей ИБД и ЭВМ при системном проектировании и выборе атрибутов качества ИБД может доминировать либо абсолютная величина занятости ресурсов различных видов, либо относительная величина использования ресурсов каждого вида при нормальном функционировании ИБД.

Практичность-применимость - зачастую значительно определяет функциональную пригодность и полезность применения ИБД для квалифицированных пользователей. В число пользователей могут быть включены администраторы, конечные и косвенные пользователи, которые находятся под влиянием или зависят от качества информации БД. В эту группу показателей качества входят субхарактеристики и атрибуты с различных сторон отражающие функциональную понятность, удобство освоения, системную эффективность и простоту использования данных. Некоторые субхарактеристики можно оценивать экономическими показателями - затратами труда и времени специалистов на реализацию некоторых функций взаимодействия с данными. Практичность зависит не только от собственных характеристик ИБД, но также от организации и адекватности документирования процессов их эксплуатации.

Понятность зависит от качества документации и субъективных впечатлений потенциальных пользователей от функций и характеристик ИБД. В системном проекте ее можно представить качественно четкостью функциональной концепции, широтой демонстрационных возможностей, полнотой, комплектностью и наглядностью представления в эксплуатационной документации возможных функций и особенностей реализации данных в БД. Она должна обеспечиваться корректностью и полнотой описания исходной и результирующей информации, а также всех деталей применения ИБД для пользователей.

Простота использования ИБД - возможность удобно и комфортно ее

эксплуатировать и управлять данными. Для этого должны быть обеспечены: достаточный объем параметров управления, реализуемых по умолчанию, информативность сообщений пользователям, наглядность унифицированность управления экраном, а также доступность изменений функций ИБД в соответствии с квалификацией пользователей и минимум операций, необходимых для реализации определенного задания и анализа результатов. Некоторые атрибуты этой субхарактеристики доступны при установлении количественных требований путем указания трудоемкости длительности соответствующих процессов подготовки и обучения квалифицированных пользователей к эффективной эксплуатации ИБД.

Изучаемость может определяться требованиями ограниченной трудоемкости и длительности подготовки пользователя к полноценной эксплуатации информации БД. Изучаемость ИБД зависит от внутренних свойств и сложности структуры информации БД, а также от субъективных характеристик квалификации конкретных пользователей. Она может также характеризоваться объемом эксплуатационной документации и/или объемом и качеством электронных учебников.

Сопровождаемость информации БД в системном проекте может отражаться удобством и эффективностью исправления, усовершенствования или адаптации структуры и содержания описаний данных в зависимостисти от изменений во внешней среде применения, а также в требованиях и функциональных спецификациях заказчика. Обобщенно качество сопровождаемости ИБД можно представить потребностью трудовых и временных ресурсов для ее обеспечения и для реализации. Возможные затраты экономических, трудовых и временных ресурсов на развитие и совершенствование качества ИБД зависят не только от внутренних свойств данных, но также и от запросов и потребностей пользователей на новые функции и от готовности заказчика и разработчика удовлетворить эти потребности. По объему предполагаемых изменений, а также вновь вводимых в очередную версию данных с учетом сложности и новизны их разработки могут быть сформулированы требования на их реализацию.

Совокупность субхарактеристик сопровождаемости ПС, представленная в стандарте ISO 9126, вполне применима для описания требований к этому показателю качества информации БД, в основном, теми же организационно-технологическими субхарактеристиками. *Анализируемость* ИБД зависит от стройности архитектуры, унифицированности интерфейса, полноты и корректности технологической и эксплуатационной документации на БД. *Изменяемость* состоит в приспособленности структуры и содержания данных к реализации специфицированных изменений и к управлению конфигурацией данных. Изменяемость зависит не только от внутренних свойств ИБД, но также от организации и инструментальной оснащенности процессов сопровождения и конфигурационного управления, на которые ориентирована в проекте архитектура, внешние и внутренние интерфейсы данных.

Тестируемость зависит от величины области влияния изменений, которые

необходимо тестировать при модификациях структуры и содержания данных в ИБД, от сложности тестов для проверки их характеристик. Ее атрибуты зависят от четкости формализации в системном проекте правил структурного построения компонентов и всего комплекса ИБД, от унификации межмодульных и внешних интерфейсов, от полноты и корректности технологической документации. Субхарактеристики ***изменяемость и тестируемость*** данных доступны количественному определению по величине трудоемкости и длительности реализации этих функций при типовых операциях с данными при применении различных методов и средств автоматизации.

Мобильность данных БД, так же как для программ, можно характеризовать в системном проекте в основном длительностью и трудоемкостью их инсталляции, адаптации и замещаемости при переносе ИБД на иные аппаратные и операционные платформы. Информация о процессах, происходящих во внешней среде, может иметь большие объем и трудоемкость первичного накопления и актуализации, что определяет необходимость ее тщательного хранения и регламентированного изменения. Формирование и заполнение информацией баз данных достаточно сложный и трудоемкий процесс, технико-экономические показатели которого сильно зависит от структуры, состава, объема, связности и других характеристик исходной информации. Особенности и трудоемкость переноса ИБД зависят, прежде всего, от характеристик совместимости архитектуры и содержания информации переносимой БД между рассматриваемыми платформами:

- ***форматная совместимость*** характеризуется степенью соответствия данных в ИБД анализируемых платформ требованиям стандартов на форматы представления данных для документальных, фактографических, словарных и иных БД;
- ***лингвистическая совместимость*** определяется степенью использования в рассматриваемых ИБД единых лингвистических средств (классификаторов, рубрикаторов, словарей), формализованных соответствующими стандартами этих платформ;
- физическая совместимость заключается в степени соответствия кодировки информации БД одинаковым стандартам на машиночитаемые носители информации.

Так как перенос БД часто обусловлен необходимостью увеличения ресурсов ЭВМ, доступных для решения новых перспективных задач, их системный проект становится естественным расширением функций ИБД относительно исходной версии проекта. Для оценки качества и определения требований к мобильности ИБД, так же как для ПС, следует решать задачу сравнения достигаемого эффекта и затрат для методов переноса или повторной разработки компонентов и наполнения базы данных в конкретных условиях с учетом всех перечисленных факторов и затрат. Эти задачи значительно упрощаются при одновременном сокращении затрат при применении идеологии и концепции открытых компьютерных систем,

поддержанных комплексом международных стандартов, а также современных версий ОС и СУБД, как стандартов де-факто.

МОДЕЛИ ОЦЕНКИ ХАРАКТЕРИСТИК КАЧЕСТВА И НАДЕЖНОСТИ ПО

Размерно-ориентированные метрики. Функционально- ориентированные метрики. Пример применения метрик. Достоинства и недостатки размерно – ориентированных и функционально-ориентированных метрик.

РАЗМЕРНО-ОРИЕНТИРОВАННЫЕ МЕТРИКИ

Размерно-ориентированные метрики прямо измеряют программный продукт и процесс его разработки. Основываются размерно-ориентированные метрики на LOC – оценках (Lines Of Code). LOC - оценка – это количество строк в программном продукте.

Исходные данные для расчета этих метрик сводятся в таблицу (табл.3).

Исходные данные для расчета LOC- метрик

Табл. 3.

Проект	Затраты, чел.-мес.	Стоимость тыс. $	KLOC, тыс. LOC	Страниц	Ошибки	Количество человек
A01	24	168	12,1	365	29	3
B02	62	440	27,2	1224	86	5
C03	43	314	20,2	1050	64	6

Таблица содержит данные о проектах за последние несколько лет. Например, запись о проекте A01 показывает: 12100 строк программы были разработаны за 24 чел.-мес. И стоили $168 000. Кроме того, по проекту было разработано 365 страниц документации, а в течение первого года эксплуатации было зарегистрировано 29 ошибок. Разрабатывали проект три человека.

На основе таблицы вычисляются размерно-ориентированные метрики производительности и качества проекта:

Производительность = Длина / Затраты (тыс.LOC/чел.-мес.);

Качество = Ошибки / Длина (Единиц/тыс. LOC);

Удельная стоимость = Стоимость /Длина (тыс.$/LOC);

Документированность = Страниц_Документа / Длина (Страниц/тыс.LOC)

Достоинства размерно-ориентированных метрик:

- широко распространены;

- просты и легко вычисляются.

Недостатки размерно-ориентированных метрик:

- зависимы от языка программирования;
- требуют исходных данных, которые трудно получить на начальной стадии проекта;
- не приспособлены к непроцедурным языкам программирования.

ФУНКЦИОНАЛЬНО-ОРИЕНТИРОВАННЫЕ МЕТРИКИ

Функционально-ориентированные метрики косвенно измеряют программный продукт и процесс его разработки. Вместо подсчета LOC – оценки при этом рассматривается не размер, а функциональность или полезность продукта.

Используется 5 информационных характеристик.

1. ***Количество внешний вводов***. Подсчитываются все вводы пользователя, по которым поступают разные прикладные данные. Вводы должны быть отделены от запросов, которые подсчитываются отдельно.

2. ***Количество внешних выводов***. Подсчитываются все выводы, по которым к пользователю поступают результаты, вычисленные программным приложением. В этом контексте выводы означают отчеты, экраны, распечатки, сообщения об ошибках. Индивидуальные единицы данных отчета отдельно не подсчитываются.

3. ***Количество внешних запросов***. Под запросами понимают диалоговый ввод, который приводит к немедленному программному ответу в форме диалогового вывода. При этом диалоговый ввод в приложении не сохраняется, а диалоговый вывод не требует вычислений. Подсчитываются все запросы – каждый учитывается отдельно.

4. ***Количество внутренних логических файлов***. Подсчитываются все логические файлы (т.е. логические группы данных, которые могут быть частью базы данных или отдельным файлом).

5. ***Количество внешних интерфейсных файлов***. Подсчитываются все логические файлы из других приложений, на которые ссылается данное приложение.

Выводы, вводы, запросы относятся к категории **транзакция**. Транзакция – это элементарный процесс, различаемый пользователем и перемещающий данные между внешней средой и программным приложением. В своей работе транзакции используют внутренние и внешние файлы. Приняты следующие определения.

Внешний ввод – элементарный процесс, перемещающий данные из внешней среды в приложение. Данные могут поступать с экрана ввода или из другого приложения. Данные могут использоваться для обновления внутренних логических файлов. Данные могут содержать как управляющую, так и деловую информацию. Управляющие данные не должны модифицировать внутренний логический файл.

Внешний вывод **-** элементарный процесс, перемещающий данные, вычисленные в приложении, во внешнюю среду. Кроме того, в этом процессе могут обновляться внутренние логические файлы. Данные создают отчеты или выходные файлы, посылаемые другим приложениям. Отчеты и файлы создаются на основе внутренних логических файлов и внешних интерфейсных файлов. Дополнительно этот процесс может использовать вводимые данные, их образуют критерии поиска и параметры, не поддерживаемые внутренними логическими файлами. Вводимые данные поступают извне, но носят временный характер и не сохраняются во внутреннем логическом файле.

Внешний запрос **–** элементарный процесс, работающий как с вводимыми, так и с выводимыми данными. Его результат – данные, возвращаемые из внутренних логических файлов и внешних интерфейсных файлов. Входная часть процесса не модифицирует внутренние логические файлы, а выходная не несет данных, вычисляемых приложением (в этом и состоит отличие запроса от вывода).

Внутренний логический файл **–** распознаваемая пользователем группа логически связанных данных, которая размещена внутри приложения и обслуживается через внешние вводы.

Внешний интерфейсный файл **–** распознаваемая пользователем группа логически связанных данных, которая размещена внутри другого приложения и поддерживается им. Внешний файл данного приложения является внутренним логическим файлом в другом приложении.

Каждой из выявленных характеристик ставится в соответствие сложность. Для этого характеристике назначается низкий, средний или высокий ранг, а затем формируется числовая оценка ранга.

Для транзакций ранжирование основано на количестве ссылок и количестве типов элементов данных. Для файлов ранжирование основано на количестве типов-элементов записей и типов элементов данных, входящих в файл.

Тип элемента-записи – подгруппа элементов данных, распознаваемая пользователем в пределах файла.

Тип элемента данных – уникальное не рекурсивное (неповторяемое) поле, распознаваемое пользователем. Примеры элементов данных для различных характеристик приведены в табл.4, 5 и содержат правила учета элементов из графического интерфейса пользователя (ГИП).

Примеры элементов данных

Табл.4.

Информационная характеристика	Элементы данных
Внешние вводы	Поля ввода данных, сообщения об ошибках, вычисляемые значения, кнопки

Внешние выводы	Поля данных в отчетах, вычисляемые значения, сообщения об ошибках, заголовки столбцов, которые читаются из внутреннего файла
Внешние запросы	Вводимые элементы: поле, используемое для списка, щелчок мыши. Выводимые элементы: отображаемые на экране поля.

Правила учета элементов данных из ГИП

Табл.5.

Элемент данных	Правило учета
Группа радиокнопок	Так как в группе пользователь выбирает только одну радиокнопку, все радиокнопки группы считаются одним элементом данных
Группа флажков (переключатели)	Так как в группе пользователь может выбрать несколько флажков, каждый флажок считают элементом данных
Командные кнопки	Командная кнопка может определять действия добавления, изменения, запроса. Кнопка ОК может вызвать транзакции (различных типов). Кнопка Next может быть входным элементом запроса или вызвать другую транзакцию. Каждая кнопка считается отдельным элементом данных.
Списки	Список может быть внешним запросом, но результат запроса может быть элементом данных внешнего вида.

Например, ГИП для обслуживания клиентов может иметь поля Имя, Адрес, Город, Страна, Почтовый_Индекс, Телефон, Email. Таким образом, имеется 7 полей или семь элементов данных. Восьмым элементом данных может быть командная кнопка (Добавить, Изменить, Удалить). В этом случае каждый из внешних вводов Добавить, Изменить, Удалить будет состоять из 8 элементов данных (7 полей плюс командная кнопка).

Обычно этому экрану ГИП соответствует несколько транзакций. Типичный экран включает несколько внешний запросов, сопровождающих внешний ввод.

Обсудим порядок учета сообщений. В приложении с ГИП генерируются три типа сообщений: сообщение об ошибке, сообщение подтверждения и сообщение уведомления. Сообщения об ошибке (например, «Требуется пароль») и сообщение подтверждения (например, «Вы действительно хотите удалить запись о клиенте?») указывают, что произошла ошибка или что процесс может быть завершен. Эти сообщения не образуют самостоятельного процесса, они являются частью другого процесса, то есть считаются элементом данных соответствующей транзакции.

С другой стороны, уведомление является независимым элементарным процессом. Например, при попытке получить от банкомата сумму денег,

превышающую их количество на счете, генерируется сообщение «Не хватает средств для завершения транзакции». Оно является результатом чтения информации из файла счета и формирования заключения. Сообщение уведомления рассматривается как внешний вывод.

Данные для определения ранга и оценки сложности транзакций и файлов приведены в табл.6-10 (числовая оценка указана в круглых скобках). Использовать их очень просто. Например, внешнему вводу, который ссылается на два файла и имеет 7 элементов данных по табл.6 назначается средний ранг и оценка сложности 4.

Ранг и оценка сложности внешних вводов

Табл.6.

Ссылки на файлы	Элементы данных		
	1-4	5-15	>15
0-1	Низкий (3)	Низкий (3)	Средний (4)
2	Низкий (3)	Средний (4)	Высокий (6)
>2	Средний (4)	Высокий (6)	Высокий (6)

Ранг и оценка сложности внешних выводов

Табл.7.

Ссылки на файлы	Элементы данных		
	1-4	5-19	>19
0-1	Низкий (4)	Низкий (4)	Средний (5)
2-3	Низкий (4)	Средний (5)	Высокий (7)
>3	Средний (5)	Высокий (7)	Высокий (7)

Ранг и оценка сложности внешних запросов

Табл.8.

Ссылки на файлы	Элементы данных		
	1-4	5-19	>19
0-1	Низкий (3)	Низкий (3)	Средний (4)
2-3	Низкий (3)	Средний (4)	Высокий (6)
>3	Средний (4)	Высокий (6)	Высокий (6)

Ранг и оценка сложности внутренних логических файлов

Табл.9.

Ссылки на файлы	Элементы данных		
	1-19	20-50	>50
0-1	Низкий (7)	Низкий (7)	Средний (10)
2-5	Низкий (7)	Средний (10)	Высокий (15)
>5	Средний (10)	Высокий (15)	Высокий (15)

Ранг и оценка сложности внешних интерфейсных файлов

Табл.10.

Ссылки на файлы	Элементы данных		
	1-19	20-50	>50
0-1	Низкий (5)	Низкий (5)	Средний (7)
2-5	Низкий (5)	Средний (7)	Высокий (10)
>5	Средний (7)	Высокий (10)	Высокий (10)

Отметим, что если во внешнем запросе ссылка на файл используется как на этапе ввода, так и на этапе вывода, она учитывается только один раз. Такое же правило распространяется на элемент данных (однократный учет).

После сбора всей необходимой информации приступают к расчетам метрики – количества функциональных указателей FP (Function Points). Автором этой метрики является А. Альбрехт (1979).

Исходные данные для расчета сводятся в табл. 11. В таблицу заносится количественное значение характеристики каждого вида (по всем уровням сложности). Места подстановки значений отмечены прямоугольником (этот символ играет роль метки - заполнителя). Количественные значения характеристик умножаются на числовые оценки сложности. Полученные в каждой строке значения суммируются, давая полное значение для данной характеристики. Эти полные значения суммируются по вертикали, формируя общее количество.

Исходные данные для расчета FP – метрик

Табл.11.

Имя характеристики	Ранг, сложность, количество			
	Низкий	Средний	Высокий	Итого
Внешние вводы	□*3=_	□*4=_	□*6=_	=□
Внешние выводы	□*4=_	□*5=_	□*7=_	=□
Внешние запросы	□*3=_	□*4=_	□*6=_	=□
Внутренние логические файлы	□*7=_	□*10=_	□*15=_	=□
Внешние интерфейсные файлы	□*5=_	□*7=_	□*10=_	=□
			Общее количество	=□

Количество функциональных указателей вычисляется по формуле:

$$FP = \text{Общее количество} * (0{,}65 + 0{,}01 * \Sigma F_i), \quad (1)$$

Где F_i – коэффициент регулировки сложности (I=1..14).

Каждый коэффициент может принимать следующие значения: 0- нет влияния, 1- случайное, 2- небольшое, 3- среднее, 4 – важное, 5 – основное. Значения выбираются эмпирически в результате ответа на 14 вопросов, которые характеризуют системные параметры приложения (табл.12).

После вычисления FP на его основе формируются метрики производительности, качества и другие оценки.

Производительность = ФункцУказатель / Затраты (FP/чел.-мес.);

Качество = Ошибки / ФункцУказатель (Единиц/FP);

Удельная Стоимость = Стоимость / ФункцУказатель (Тыс.$/FP);

Документированность=СтраницДокумента/ФункцУказатель (Страниц/FP)

Определение системных параметров приложения

Табл.12.

№	Системный параметр	Описание
1	Передачи данных	Сколько средств данных требуется для пердачи или обмена информацией с приложением или системой?
2	Распределенная обработка данных	Как обрабатываются распределенные данные и функции обработки?
3	Производительность	Нуждается ли пользователь в фиксации времени ответа или производительности?
4	Распространенность используемой конфигурации	Насколько распространена текущая аппаратная платформа, на которой будет выполняться приложение?
5	Скорость транзакций	Как часто выполняются транзакции? (каждый день, каждую неделю, каждый месяц)?
6	Оперативный ввод данных	Какой процент информации нужно вводить в режиме онлайн?
7	Эффективность работы конечного пользователя	Приложение проектировалось для обеспечения эффективной работы конечного пользователя?
8	Оперативное обновление	Как много внутренних файлов обновляется в онлайновой транзакции?
9	Сложность	Выполняет ли приложение интенсивную

	обработки	логическую или математическую обработку?
10	Повторная используемость	Приложение разрабатывалось для удовлетворения требований одного или многих пользователей?
11	Легкость инсталляции	Насколько трудны преобразования и инсталляция приложения?
12	Легкость эксплуатации	Насколько эффективны и/или автоматизированы процедуры запуска, резервирования и восстановления?
13	Разнообразные условия размещения	Была ли спроектирована, разработана и поддержана возможность инсталляции приложения в разных местах для различных организаций?
14	Простота изменений	Была ли спроектирована, разработана и поддержана в приложении простота изменения?

Область применения функциональных указателей – коммерческие информационные системы. Для продуктов с высокой алгоритмической сложностью используются ***метрики свойств*** (Features Points). Они применимы к системному и инженерному ПО, ПО реального времени и встроенному ПО.

Для вычисления указателя свойств добавляется одна характеристика – *количество алгоритмов*. Алгоритм здесь определяется как ограниченная программа вычислений, которая включается в общую компьютерную программу. Примеры алгоритмов: обработка прерываний, инвертирование матрицы, расшифровка битовой строки. Для формирования указателя свойств составляется табл. 13.

Исходные данные для расчета указателя свойств

Табл.13.

№	Характеристика	Количество	Сложность	Итого
1	Вводы	□	*4	=□
2	Выводы	□	*5	=□
3	Запросы	□	*4	=□
4	Логические файлы	□	*7	=□
5	Интерфейсные файлы	□	*7	=□
6	Количество алгоритмов	□	*3	=□
			Общее количество	=□

После заполнения таблицы по формуле (1) вычисляется значение указателя свойств. Для сложных систем реального времени это значение на 25-30% больше значения, вычисляемого по таблице для количества функциональных указателей.

Достоинства функционально-ориентированных метрик:

- не зависят от языка программирования;
- Легко вычисляются на любой стадии проекта.

Недостаток функционально-ориентированных метрик: результаты основаны на субъективных данных, используются не прямые, а косвенные измерения.

FP – оценки легко пересчитать в LOC – оценки. Как показано в табл.14, результаты пересчета зависят от языка программирования, используемого для реализации ПО.

Пересчет FP – оценок в LOC – оценки

Табл.14.

Язык программирования	Количество операторов на 1 FP
Ассемблер	320
C	128
Паскаль	90
C++	64
Java	53
Visual Basic	32
Visual C++	34
Delphi Pascal	29
HTML 3	15
LISP	64
Prolog	64

ТЕМА 13. ДОКУМЕНТИРОВАНИЕ ПРОГРАММНЫХ СРЕДСТВ

13.1. Документация, создаваемая и используемая в процессе разработки программных средств

При разработке ПС создается и используется большой объем разнообразной документации. Она необходима как средство передачи информации между разработчиками ПС, как средство управления разработкой ПС и как средство передачи пользователям информации, необходимой для применения и сопровождения ПС. На создание этой документации приходится большая доля стоимости ПС.

Эту документацию можно разбить на две группы:

- Документы управления разработкой ПС.
- Документы, входящие в состав ПС.

Документы управления разработкой ПС (*software process documentation*) управляют и протоколируют процессы разработки и сопровождения ПС, обеспечивая связи внутри коллектива разработчиков ПС и между коллективом разработчиков и *менеджерами ПС* (*software managers*) - лицами, управляющими разработкой ПС. Эти документы могут быть следующих типов :

- *Планы, оценки, расписания.* Эти документы создаются менеджерами для прогнозирования и управления процессами разработки и сопровождения ПС.
- *Отчеты об использовании ресурсов в процессе разработки.* Создаются менеджерами.
- *Стандарты.* Эти документы предписывают разработчикам, каким принципам, правилам, соглашениям они должны следовать в процессе разработки ПС. Эти стандарты могут быть как международными или национальными, так и специально созданными для организации, в которой ведется разработка ПС.
- *Рабочие документы.* Это основные технические документы, обеспечивающие связь между разработчиками. Они содержат фиксацию идей и проблем, возникающих в процессе разработки, описание используемых стратегий и подходов, а также рабочие (временные) версии документов, которые должны войти в ПС.
- *Заметки и переписка.* Эти документы фиксируют различные детали взаимодействия между менеджерами и разработчиками.

Документы, входящие в состав ПС (*software product documentation*), описывают программы ПС как с точки зрения их применения пользователями, так и с точки зрения их разработчиков и сопроводителей (в соответствии с назначением ПС). Здесь следует отметить, что эти документы будут использоваться не только на стадии эксплуатации ПС (в ее фазах применения и сопровождения), но и на стадии разработки для управления процессом разработки (вместе с рабочими документами) - во всяком случае, они должны быть проверены (протестированы) на соответствие программам ПС. Эти документы образуют два комплекта с разным назначением:

- Пользовательская документация ПС (П-документация).
- Документация по сопровождению ПС (С-документация).

13.2. Пользовательская документация программных средств

Пользовательская документация ПС (*user documentation*) объясняет пользователям, как они должны действовать, чтобы применить разрабатываемое ПС. Она необходима, если ПС предполагает какое-либо взаимодействие с пользователями. К такой документации относятся документы, которыми должен руководствоваться пользователь при *инсталляции* ПС (при установке ПС с соответствующей настройкой на среду применения ПС), при применении ПС для решения своих задач и при управлении ПС (например, когда разрабатываемое ПС будет взаимодействовать с другими системами). Эти документы частично затрагивают вопросы сопровождения ПС, но не касаются вопросов, связанных с модификацией программ.

В связи с этим следует различать две категории пользователей ПС: ординарных пользователей ПС и администраторов ПС. *Ординарный пользователь ПС* (e*nd-user*) использует ПС для решения своих задач (в своей предметной области). Это может быть инженер, проектирующий техническое устройство, или кассир, продающий железнодорожные билеты с помощью ПС. Он может и не знать многих деталей работы компьютера или принципов программирования. *Администратор ПС* (*system administrator*) управляет использованием ПС ординарными пользователями и осуществляет сопровождение ПС, не связанное с модификацией программ. Например, он может регулировать права доступа к ПС между ординарными пользователями, поддерживать связь с поставщиками ПС или выполнять определенные действия, чтобы поддерживать ПС в рабочем состоянии, если оно включено как часть в другую систему.

Состав пользовательской документации зависит от аудиторий пользователей, на которые ориентировано разрабатываемое ПС, и от режима использования документов. Под *аудиторией* здесь понимается контингент пользователей ПС, у которого есть необходимость в определенной пользовательской документации ПС . Удачный пользовательский документ существенно зависит от точного определения аудитории, для которой он предназначен. Пользовательская документация должна содержать информацию, необходимую для каждой аудитории. Под *режимом использования* документа понимается способ, определяющий, каким образом используется этот документ. Обычно пользователю достаточно больших программных систем требуются либо документы для изучения ПС (использование в виде *инструкции*), либо для уточнения некоторой информации (использование в виде *справочника*).

В соответствии с работами можно считать типичным следующий состав пользовательской документации для достаточно больших ПС:

- *Общее функциональное описание ПС*. Дает краткую характеристику функциональных возможностей ПС. Предназначено для пользователей, которые должны решить, насколько необходимо им данное ПС.
- *Руководство по инсталяции ПС*. Предназначено для администраторов ПС. Оно должно детально предписывать, как устанавливать системы в конкретной среде, в частности, должно содержать описание компьютерно-считываемого носителя, на котором поставляется ПС, файлы, представляющие ПС, и требования к минимальной конфигурации аппаратуры.
- *Инструкция по применению ПС.* Предназначена для ординарных пользователей. Содержит необходимую информацию по применению ПС, организованную в форме удобной для ее изучения.
- *Справочник по применению ПС*. Предназначен для ординарных пользователей. Содержит необходимую информацию по применению ПС, организованную в форме удобной для избирательного поиска отдельных деталей.
- *Руководство по управлению ПС*. Предназначено для администраторов ПС. Оно должно описывать сообщения, генерируемые, когда ПС взаимодействует с другими системами, и как должен реагировать администратор на эти сообщения. Кроме того, если ПС использует системную аппаратуру, этот документ может объяснять, как сопровождать эту аппаратуру.

Как уже говорилось ранее (см. лекцию 4), разработка пользовательской документации начинается сразу после создания внешнего описания. Качество этой документации может существенно определять успех ПС. Она должна быть достаточно проста и удобна для пользователя (в противном случае это ПС, вообще, не стоило создавать). Поэтому, хотя черновые варианты (наброски) пользовательских документов создаются основными разработчиками ПС, к созданию их окончательных вариантов часто привлекаются профессиональные технические писатели. Кроме того, для обеспечения качества пользовательской документации разработан ряд стандартов, в которых предписывается порядок разработки этой документации, формулируются требования к каждому виду пользовательских документов и определяются их структура и содержание.

Документация по сопровождению программных средств

Документация по сопровождению ПС (*system documentation*) описывает ПС с точки зрения ее разработки. Эта документация необходима, если ПС предполагает изучение того, как оно устроена (сконструирована), и модернизацию его программ. Как уже отмечалось, сопровождение - это продолжающаяся разработка. Поэтому в случае необходимости модернизации ПС к этой работе привлекается специальная команда разработчиков-сопроводителей. Этой команде придется иметь дело с такой же документацией, которая определяла деятельность команды

первоначальных (основных) разработчиков ПС, - с той лишь разницей, что эта документация для команды разработчиков-сопроводителей будет, как правило, чужой (она создавалась другой командой). Чтобы понять строение и процесс разработки модернизируемого ПС, команда разработчиков-сопроводителей должна изучить эту документацию, а затем внести в нее необходимые изменения, повторяя в значительной степени технологические процессы, с помощью которых создавалось первоначальное ПС.

Документация по сопровождению ПС можно разбить на две группы:

1. документация, определяющая строение программ и структур данных ПС и технологию их разработки;
2. документацию, помогающую вносить изменения в ПС.

Документация первой группы содержит итоговые документы каждого технологического этапа разработки ПС. Она включает следующие документы:

- Внешнее описание ПС (Requirements document).
- Описание архитектуры ПС (description of the system architecture), включая внешнюю спецификацию каждой ее программы (подсистемы).
- Для каждой программы ПС - описание ее модульной структуры, включая внешнюю спецификацию каждого включенного в нее модуля.
- Для каждого модуля - его спецификация и описание его строения (design description).
- Тексты модулей на выбранном языке программирования (program source code listings).
- Документы установления достоверности ПС (validation documents), описывающие, как устанавливалась достоверность каждой программы ПС и как информация об установлении достоверности связывалась с требованиями к ПС.

Документы установления достоверности ПС включают, прежде всего, документацию по тестированию (схема тестирования и описание комплекта тестов), но могут включать и результаты других видов проверки ПС, например, доказательства свойств программ. Для обеспечения приемлемого качества этой документации полезно следовать общепринятым рекомендациям и стандартам.

Документация второй группы содержит

- *Руководство по сопровождению ПС* (system maintenance guide), которое описывает особенности реализации ПС (в частности, трудности, которые пришлось преодолевать) и как учтены возможности развития ПС в его строении (конструкции). В нем также

фиксируются, какие части ПС являются аппаратно- и программно-зависимыми.

Общая проблема сопровождения ПС - обеспечить, чтобы все его представления шли в ногу (оставались согласованными), когда ПС изменяется. Чтобы этому помочь, связи и зависимости между документами и их частями должны быть отражены в руководстве по сопровождению, и зафиксированы в базе данных управления конфигурацией.

Тема 14. Технико-экономическое обоснование проектов программных средств

Понятие технико-экономического обоснования программного средства. Экономика жизненного цикла ПС.

Приступая к разработке крупных проектов, руководители, прежде всего, пытаются понять целесообразно ли их создание и оценить какова будет возможная эффективность применения готового продукта, оправдаются ли затраты на его разработку и использование. Поэтому такие проекты традиционно начинаются с анализа и разработки *технико-экономического обоснования* (ТЭО) предстоящего жизненного цикла (ЖЦ) проекта и эксплуатации предполагаемого продукта.

Заказчику проекта необходимо оценить реальную потребность в его создании и возможную конкурентоспособность, а потенциальному разработчику-поставщику создаваемого продукта, провести оценку реализуемости проекта в условиях и ресурсах, предлагаемых заказчиком. Должен быть подготовлен согласованный между заказчиком и разработчиком *первичный документ,* в котором определены цели и задачи проекта, предполагаемые характеристики продукта и необходимые ресурсы для его реализации. Эти данные должны быть предварительно сбалансированы обеспечивать реализацию целей проекта при выделенных ресурсах с минимальным допустимым риском.

Однако масштабы целей и функций сложных проектов имеют устойчивую тенденцию изменяться и увеличиваться по мере развития, а первоначально выделяемые ресурсы не удовлетворять их реализацию. Технико-экономическое обоснование проектов на начальном этапе их

развития должно содержать *оценки рисков* реализации поставленных целей, обеспечивать возможность планирования и выполнения жизненного цикла продукта или указывать на недопустимо высокий риск его реализации и целесообразность прекращения разработки.

Массовое создание сложных программных средств промышленными методами и большими коллективами специалистов вызвало необходимость их четкой организации, планирования работ по затратам, этапам и срокам реализации. Совокупные затраты в мире на такие разработки составляют миллиарды, а для отдельных проектов - миллионы долларов в год, поэтому требуется тщательный анализ эффективности создания и использования ПС. Для решения этих задач формируется новая область знания и научная дисциплина - *экономика жизненного цикла программных средств* как часть экономики промышленности и вычислительной техники в общей экономике государств и предприятий.

Развитие этой области экономики связано с большими трудностями, типичными для новых разделов науки и техники, появляющихся на стыке различных областей знания. В данном случае особенности состоят в том, что руководители и разработчики комплексов программ, как правило, не знают даже основ экономики разработки и производства сложной продукции, а экономисты не представляют сущность объектов разработки - программных средств, а также особенностей их создания, технологического процесса и применения.

Объективно положение осложнено трудностью измерения характеристик таких объектов. Широкий спектр количественных и качественных показателей, которые с различных сторон характеризуют содержание этих объектов, и невысокая достоверность оценки их значений, определяют значительную дисперсию при попытке описать и измерить свойства создаваемых или используемых ПС.

Особенности развития методов и процессов технико-экономического обоснования проектов ПС обусловлены, в частности, сложностью, и, в ряде случаев, неопределенностью характеристик предполагаемого продукта, технологических этапов и процессов разработки, производства и применения программ для ЭВМ. При разработке комплексов программ сложно переплетаются содержание, этапы и распределение работ, возможен ряд возвратов на более ранние технологические этапы в процессе создания компонентов ПС, они имеют не совсем определенные границы начала и завершения. Специалисты в коллективе могут на некотором интервале времени решать несколько производственных задач и заменять друг друга.

Положение усугубляется трудностью поэтапного определения качества

компонентов и его прогнозирования в процессе разработки, что непосредственно отражается на технико-экономических показателях (ТЭП) проекта в целом. Следствием этого являются *большие ошибки при планировании* сроков, трудоемкости и стоимости создания ПС. Эта стихийность при создании крупных комплексов программ в большинстве случаев приводит к значительному запаздыванию разработок и превышению предполагавшихся затрат.

Практика последних лет показывает, что вследствие *пренебрежения тщательным технико-экономическим обоснованием*, до 15% проектов сложных программных комплексов не доходит до завершения, а почти половина проектов не укладывается в выделенные бюджет и сроки и не обеспечивает требуемые характеристики качества. Типичны ситуации, когда отставание сроков внедрения промышленных систем управления и обработки информации полностью зависит от неготовности программ для них.

Для небольших относительно простых проектов ПС, во многих случаях достаточно достоверными могут быть интуитивные оценки требуемых ресурсов, выполняемые опытными руководителями, реализовавшими несколько аналогичных проектов. При начале проектировании крупных ПС, требующих заведомо больших экономических, трудовых и временных затрат, необходима хотя бы приближенная, формализованная их оценка по определенной методике. *Интуитивные оценки руководителями* и исполнителями - размеров, сложности и трудоемкости конкретных программных проектов, как правило, *отличаются существенными недостатками:*

- человек, в основном оптимистичен, и каждому хочется, чтобы проект ПС было меньше по размеру и более простым, что ведет к первоначальным недооценкам его сложности и к конфликтным ситуациям при разработке;

- человек обычно не полностью использует предыдущий опыт о сложности функций аналогичных ПС и, особенно, о большом размере вспомогательных компонентов комплексов программ, которые также должны быть разработаны;

- отдельные специалисты, как правило, не знакомы со всем объемом проекта и пожеланиями пользователей, что приводит к недооценке второстепенных функций и компонентов ПС, к отсутствию реалистичного применению накопленных знаний при оценивании размера и сложности проекта.

Эти обстоятельства приводят к большим ошибкам оценивания ТЭП на начальных этапах разработки, которые можно значительно сократить *при относительно небольших усилиях, применяя, в частности*, формализованные методики экспертной оценки основных технико-экономических характеристик проектов комплекса программ. Тем самым проекты сложных ПС с самого начала могли бы выполняться с учетом

более достоверной оценки необходимых ресурсов. Следствием недостатков или отсутствия технико-экономическим обоснованием проектов новых ПС являются *острые конфликты между заказчиками и разработчиками.*

Часто разработчики ПС не в состоянии привести заказчику или руководителю проекта достаточно обоснованные доказательства не реальности выполнения выдвигаемых требований и предложенных ограниченных бюджета и сроков. Это приводит к оптимистической переоценке выгод новой программной разработки, к недооценке роли других конкурирующих предложений при заключении контрактов на разработку, и вследствие этого - к неизбежным перерасходам средств и к снижению качества ПС.

Руководители конкретных проектов обычно не в состоянии достаточно обоснованно определять, сколько времени и затрат труда потребуется на каждый этап и работу программной части проекта системы. Вследствие этого, они не могут оценить, насколько успешно выполняется имеющийся план разработки ПС. Это, как правило, означает, что программная часть проекта системы с самого начала выходит из-под контроля и возможна катастрофа с реализацией и завершением проекта всей системы в требуемый срок с заданным качеством.

Большую часть этих негативных последствий можно избежать, используя существующие, достаточно точные *методы оценивания и прогнозирования затрат, а также управления проектами ПС,* для их успешного завершения. Эти последствия объясняются многими причинами, из которых наиболее важными, являются следующие:

- исходные тексты программных компонентов различны и по отдельности не определяют сложность и размер конечного продукта;

- разработка сложных ПС требует творчества и сотрудничества разных специалистов, индивидуальное и групповое поведение которых, как правило, трудно предсказать;

- в области экономики жизненного цикла ПС накоплен относительно небольшой опыт количественных оценок, и его трудно увеличивать, проводя и не обобщая разрозненные эксперименты.

За последние несколько лет ряд исследований и работ *по сбору и обобщению экономических данных о ЖЦ ПС* заложили основы для методов и моделей оценивания затрат, которые обладают удовлетворительной точностью. Современная экономическая модель оценки разработки ПС считается хорошей, если с ее помощью можно оценить затраты на программную разработку с точностью 20 % в 70% случаев, при условии использования

модели, в классе проектов, на которые она ориентирована. Имеющиеся модели не всегда столь точны, как хотелось бы, но могут весьма существенно помочь в технико-экономическом анализе и обосновании решений при создании сложных ПС.

Необходимы дальнейшие, активные исследования на разных уровнях детализации, начиная от экономики и планирования создания программных средств в масштабах страны или предприятия и кончая экономикой выполнения частных операций отдельными специалистами при разработке или производстве конкретных ПС. Одна из важнейших задач состоит в том, чтобы увязать четкими экономическими категориями взаимодействие разных специалистов и предприятий в типовой производственной цепочке: заказчик - разработчик - изготовитель - пользователь. Для этого объект потребления - программное средство и все процессы взаимодействия в цепочке должны быть связаны системой экономических и технических характеристик, в той или иной степени, использующих основные экономические показатели - *реальные затраты ресурсов: финансов, труда и времени специалистов на конечный продукт.*

Для решения этой задачи необходимо детально исследовать требуемые ресурсы современных процессов создания и использования программ различных классов и назначения - встроенных, коммерческих, административных, учебных, уникальных и др. Только на базе серьезных статистических исследований технико-экономических показателей жизненного цикла и практического маркетинга ПС *возможны обобщения и создание теоретических и практических основ экономики ПС.* Перечисленные выше проблемы и задачи требуют для своего решения выполнения крупных, комплексных научно-исследовательских работ, многие из которых еще не поставлены и далеки от разрешения.

Цели и задачи технико-экономического анализа и обоснования комплекса программ.

Технико-экономический анализ разработки комплексов программ состоит в выборе и прогнозировании наиболее адекватных экономических и функциональных критериев для обобщенного описания эффективности, стоимости создания и использования комплексов программ в зависимости от их назначения, области применения и других факторов. Применение программных средств как *продукции* существенно повысило *актуальность технико-экономического обоснования и прогнозирования* их характеристик и процессов создания. Для получения обобщенных, конструктивных результатов ниже основной целью считается разработка сложных программных средств различных классов независимо от конкретных областей, в которых применяются системы, используемые для управления и обработки информации.

Предполагается, что основной целью создания ПС является повышение эффективности производства промышленных изделий или управления объектами и системами, в которых применяются крупные комплексы программ. Такими системами могут быть средства автоматизированного управления прокатными станами, самолетами или электростанциями, информационно-справочные системы административного управления, системы автоматизации проектирования и обучения и т.п. В ряде случаев программы невозможно или очень трудно характеризовать непосредственной экономической эффективностью. Примером могут служить ПС в системах управления воздушным движением или космическими аппаратами, а также в системах военного назначения или автоматизации научного эксперимента. В таких случаях при анализе программ невозможно определять изменение прямой эффективности систем в зависимости от затрат и целесообразно из анализа исключать характеристики полной экономической эффективности и сопутствующие ей функциональные критерии качества. Тогда исследование эффективности ПС можно проводить, *минимизируя затраты на разработку* в предположении, что полностью обеспечены заданные функциональные характеристики.

Обеспечение жизненного цикла любых изделий не может быть бесплатным, оно требует определенных затрат ресурсов, которые обычно тем больше, чем выше требуемое их качество. Многие проекты информационных систем терпели неудачу из-за отсутствия у разработчиков и заказчиков при подготовке контракта четкого представления о реальных трудовых, временных и иных ресурсах, необходимых для их реализации. Существенными факторами, влияющими на трудоемкость, длительность реализации и качество ПС и БД, оказывают ограничения ресурсов, доступных для обеспечения их жизненного цикла, а также возможная экономическая эффективность применения. Общее понятие - доступные ресурсы разработки включает реальные финансовые, кадровые и аппаратурные ограничения, в условиях которых предстоит создание и развитие комплекса программ. Эти факторы влияют на рентабельность процессов разработки, которые следует учитывать и оптимизировать при создании и применении ПС. Поэтому одной из основных задач при обеспечении ЖЦ ПС является *технико-экономический анализ и обоснование необходимых ресурсов для создания проекта ПС* в соответствии с требованиями контракта. В ряде случаев этому помогает опыт и экономические характеристики ранее выполненных проектов фирмы, но некоторые проекты могут не иметь прецедентов, и тогда приходится использовать обобщенный опыт и имеющуюся статистику в этой области.

При планировании ЖЦ программных средств, часто имеется определенный заказчик-потребитель, который способен задать основные цели, характеристики и обеспечить ресурсы для реализации проекта. Однако иногда инициатором разработки ПС является разработчик-поставщик, который самостоятельно принимает все решения о проектировании за счет собственных ресурсов и предполагает возместить затраты путем реализации программного продукта на рынке. Таким образом, *при технико-экономическом анализе и обосновании проектов ПС возможны два сценария:*

- создание и весь жизненный цикл комплекса программ и/или базы данных ориентируется на массовое тиражирование и распространение их на рынке, среди заранее не известных пользователей в различных сферах и внешней среде применения; при этом отсутствует конкретный внешний потребитель-заказчик, который определяет и диктует основные требования к ПС и финансирует проект;

- разработка проекта ПС и/или БД предполагается для конкретного потребителя-заказчика с определенным, относительно небольшим тиражом и с известной областью и внешней средой применения.

Эти сценарии *принципиально различаются методами* технико-экономического анализа и обоснования их характеристик.

Первый сценарий базируется на маркетинговых исследованиях рынка программных продуктов и на стремлении поставщика занять на рынке достаточно выгодное место. Для этого ему необходимо определить наличие на рынке всей гаммы близких по назначению и функциям ПС, оценить их эффективность, стоимость и применяемость, а также *возможную конкурентоспособность* предполагаемого к разработке программного продукта для потенциальных пользователей и их возможное число. Следует оценить рентабельность затрат на создание и обеспечение всего ЖЦ нового ПС, выявить факторы, функциональные, экономические и конструктивные показатели качества, которые способны привлечь достаточное число покупателей и оправдать затраты на предстоящую разработку. Для этого разработчикам необходимо произвести оценки возможной конкурентоспособности предполагаемой продукции на рынке *по величине соотношения*:

- возможной *эффективности* (ценности, достоинств) последующего применения ПС и способности удовлетворить потребности пользователей при его использовании;

- *к стоимости* (цене, затратам), которую готов заплатить пользователь при приобретении и эксплуатации данного комплекса программ или базы

данных.

Второй сценарий предполагает наличие определенного заказчика-потребителя конкретного проекта ПС и/или БД, который определяет основные технические и экономические требования. Он выбирает конкурентоспособного поставщика-разработчика, которого оценивает на возможность реализовать проект с необходимым качеством с учетом ограничения требуемых бюджета, сроков и других ресурсов. При этом результаты разработки не обязательно подлежат широкому тиражированию, могут не поступать на рынок, маркетинговые исследования для таких проектов являются второстепенными и предварительно могут не проводиться. Однако для заказчика и разработчика при заключении контракта необходимо достаточно достоверное прогнозирование требований к программному продукту и технико-экономическое обоснование требуемых ресурсов по трудоемкости, стоимости, срокам и другим показателям. Заказчик заинтересован в получении ПС высокого качества при минимальных затратах, а разработчик желает получить максимальную оплату за созданный продукт и достаточные ресурсы на его реализацию. Противоположность интересов поставщика и потребителя при оценке стоимости и других ресурсов проекта, требует поиска компромисса, при котором разработчик ПС не продешевит, а заказчик не переплатит за конкретные выполненные работы и весь проект. Поэтому оба партнера заинтересованы в достоверном технико-экономическом прогнозировании и обосновании проекта ПС. Ниже основное внимание сосредоточено на, *технико-экономическом анализе и обосновании процесса разработки и всего жизненного цикла ПС,* путях минимизации затрат и на повышении эффективности автоматизации применяемых технологий. При таком анализе должны учитываться следующие цели.

Первая цель состоит в *прогнозировании реальных затрат* на разработку определенного проекта компонентов и ПС в целом с учетом их сложности и требуемого качества. Для этого должна быть изучена существующая практика разработки программ, и/или обобщены ТЭП современных проектов ПС. Такие обобщения должны выявить трудоемкость (стоимость) и производительность труда при разработке реальных программ определенных классов и назначения, а также основные факторы, влияющие на эти показатели при создании конкретных ПС. Кроме того, необходимо определить длительность всего процесса разработки программ и его отдельных этапов. Для этого должны быть разработаны и внедрены методики сбора первичных технико-экономических данных и их обработки, по завершенным или находящимся в процессе разработки проектам ПС. В результате могут быть получены современные значения основных ТЭП создания программ разных классов.

Вторам цель - создание методов и методик прогнозирования затрат и длительности разработки комплексов программ. Методики должны учитывать полученные значения ТЭП. основные характеристики создаваемых ПС, а также технологию, оснащенность и организацию их разработки. Получаемые прогнозы позволят эффективно планировать разработки, управлять созданием программ и осуществлять проекты ПС в соответствии с заданными требованиями, сроками и затратами на основе анализа аналогов - прототипов.

Третьей целью анализа является *обоснование и создание методов и средств снижения совокупных затрат и сроков* разработки сложных ПС. При этом возникают задачи:

- эффективного распределения общих трудовых ресурсов, используемых при разработке программ;

- развития и повышения экономической эффективности технологий, применяемых для создания ПС различных классов;

- рационального повышения уровня комплексной автоматизации технологий разработки ПС;

- выбора методов и инструментальных средств, в наибольшей степени способствующих снижению длительности создания и совокупных затрат на ПС, а также повышению их качества.

Четвертой целью технико-экономического исследования процессов разработки программ является *создание методических и нормативных документов,* как основы промышленной разработки аналогичных программных продуктов. Наличие нормативов может коренным образом изменить характер разработки ПС и приблизить его к отрасли современного регламентированного промышленного проектирования. В результате появится возможность управления затратами на разработку, количеством и качеством создаваемых ПС и их компонентов на различных этапах.

В качестве основного критерия эффективности новой техники и ПС, в частности, широко применяется критерий *экономии совокупных затрат общественного труда*, которая получается в результате внедрения этой техники. Однако во многих случаях созданная техника способствует повышению качества изделий или является принципиально новой продукцией, что затрудняет ее оценку по критерию непосредственной экономической эффективности. Поэтому широко применяется критерий *минимальных приведенных затрат,* которые требуются при создании и эксплуатации анализируемых изделий. Приведенные затраты включают затраты на проектирование, изготовление и эксплуатацию изделий по

всему жизненному циклу или пересчитанные к годовому интервалу времени. Этот критерий может поддерживаться (детализироваться) рядом *локальных критериев:* повышением производительности труда, экономией материальных и производственных ресурсов при выполнении частных работ, улучшением использования оборудования и т. п. Критерий минимальных приведенных затрат применим, если различные технические решения сопоставимы по функциям, достигаемым целям и качеству продукции. Однако этот критерий непосредственно не учитывает возможные различия назначения, функциональных и технических характеристик создаваемых и эксплуатируемых систем. Обычно предполагается, что для каждого изделия зафиксирован эффект от его создания и использования и необходимо выявить все основные факторы, способствующие *минимизации совокупных затрат на всем жизненном цикле.*

На практике классы систем при анализе обычно имеют ряд близких по значимости целей применения, и соответствующих им характеристик качества. В результате эффективность технологических решений приходится оценивать одновременно по нескольким показателям. Для этого стремятся сформулировать обобщенную скалярную функцию эффекта и затрат или строится нормированный вектор показателей качества. Для многокритериальной, векторной оптимизации решений разработан ряд методов, использование которых зависит от конкретных особенностей анализируемых изделий. Кроме того, широко применяется последовательный анализ по отдельным показателям качества с учетом их приоритета.

Во многих случаях эффективность сложной новой техники и ПС в процессе проектирования приходится прогнозировать *в условиях неопределенности* целей, различных факторов и характеристик. Обычно недостаточно известны перспективы внедрения и эксплуатации объектов разработки - новых программных продуктов. Трудно формализуемыми и оцениваемыми являются размеры (масштабы) и структура систем, взаимодействие основных подсистем, цели, функции и критерии оценки эффективности их функционирования. Значительные неопределенности содержатся также в технико-экономических характеристиках технологий, а также инструментальных средств автоматизации проектирования и изготовления ПС. В результате экономический анализ и прогнозы могут иметь значительный разброс оцениваемых показателей.

Программно-целевое планирование и промышленная разработка ПС как продукции целесообразны только для определенных классов комплексов программ. С этой позиции программы для вычислительных машин можно разделить на три класса, которые впоследствии рассмотрены подробнее:

- к *первому классу* относятся программы автоматического или автоматизированного управления, непосредственно входящие, встроенные

в системы управления, функционирующие в реальном масштабе времени;
- *второй класс* представляется сложными ПС: информационно-справочных систем обработки информации организационного и административного направления, систем автоматизации проектирования, которые функционируют вне жесткого реального масштаба времени;
- к *третьему классу* относятся программы, разрабатываемые для решения частных инженерных и научно-исследовательских задач, которые характеризуются относительно малым использованием ресурсов вычислительных систем и кратковременной эксплуатацией.
С позиции технико-экономического анализа жизненный цикл ПС можно разделить на две части, существенно различающиеся особенностями процессов, технико-экономическими характеристиками и влияющими на них факторами.

В первой части ЖЦ производятся системный анализ, проектирование, разработка, тестирование и испытания первой базовой версии ПС. Номенклатура работ, их трудоемкость, длительность и другие характеристики на этих этапах ЖЦ существенно зависят от создаваемого объекта, требуемых показателей качества, внешней и технологической среды разработки. Изучение подобных зависимостей для различных ПС позволяет прогнозировать состав и основные технико-экономические показатели, планы и графики работ для вновь создаваемых ПС. *Вторая часть ЖЦ,* отражающая эксплуатацию и сопровождение ПС, относительно слабо связана с характеристиками объекта и среды разработки. Программы первого и второго классов характеризуются *длительной непрерывной эксплуатацией*, продолжительность которой обычно значительно превышает длительность разработки первой версии. После того как программы созданы и испытаны, в ряде случаев они становятся недоступными для разработчиков и эксплуатируются неизменными до внедрения очередной версии при модернизации системы. Жизненный цикл таких ПС может составлять десяток лет, в течение которых необходимо обеспечить их *сопровождение.* В процессе сопровождения программы могут подвергаться эпизодическим корректировкам, которые должны регистрироваться, накапливаться и передаваться пользователям экземпляров системы. Необходимо обеспечить адекватность документации каждой версии эксплуатируемого ПС в любой момент времени.

Номенклатура работ на этих этапах более или менее стабильна, а их трудоемкость и длительность могут сильно варьироваться и зависят от массовости и других факторов распространения и применения ПС. Успех ПС у пользователей и на рынке, а также процесс развития версий трудно предсказать, и он не связан непосредственно с техническими параметрами комплексов программ. Определяющими становятся потребительские характеристики и качество ПС, а их технико-экономические особенности

с позиции разработчиков отходят на второй план (см. выше, первый сценарий). Вследствие этого в широких пределах изменяются трудоемкость и необходимое число специалистов, поддерживающих эти этапы. Это затрудняет обобщение ТЭП различных проектов и прогнозирование на их основе аналогичных характеристик новой разработки. Поэтому планы работ на этих этапах имеют характер общих взаимосвязей работ, которые требуют ручного распределения во времени индивидуально для каждого проекта. В результате планирование трудоемкости, длительности и числа специалистов для этих этапов приходится производить итерационно на *базе накопления опыта и анализа развития конкретных типов и версий ПС.*

Характеристики и технико-экономические показатели программного средства.

Труднее всего обосновать технико-экономические показатели разработки комплекса программ в начале проекта, когда еще не сформировались достаточно четкие представления о функциях и свойствах ПС, подлежащего разработке. На базе этих оценок желательно сделать общий вывод, стоит ли заниматься данным проектом в дальнейшем и на каких условиях следует заключить контракт на его выполнение. Когда разработка программного проекта близится к завершению, с целью уточненного оценивания ТЭП следует учитывать некоторые дополнительные аспекты и спецификации. Однако общую смету, время работы над проектом и объем необходимых трудозатрат необходимо оценивать как можно раньше. При этом целесообразно поэтапно рассматривать ряд факторов, влияющих на технико-экономические показатели разработки ПС, представленные в таблице №1, которые в данном разделе используются как основа для последовательности их изложения.

При оценке стоимости проекта и количества времени, требуемого для его выполнения, предстоит выполнить ***два основных этапа.*** Первый этап состоит в ***оценивании размера*** - ***масштаба проекта,*** на втором этапе представление об этих размерах наряду с информацией о других факторах среды разработки используется для ***оценивания трудозатрат и, соответственно, стоимости и продолжительности работ.***

Оценка технико-экономических показателей программных средств.

Основными ресурсами у разработчиков при создании сложных комплексов программ являются: допустимые ***трудозатраты*** на разработку ПС с требуемым качеством; ***время*** - длительность полного цикла создания

программного продукта; необходимое и доступное ***число специалистов*** соответствующей квалификации. Потребность в этих ресурсах в наибольшей степени зависит от размера - масштаба и сложности разрабатываемого ПС. Когда впервые рассматривается масштаб нового проекта ПС, интуитивные и экспертные оценки его трудоемкости могут отличаться от конечного значения примерно в полтора раза в ту или другую сторону. Рисунок 1 иллюстрирует достоверность, с которой могут быть получены оценки трудоемкости или стоимости ПС, представленные в виде функции этапа ЖЦ (горизонтальная ось), или уровень знаний о предполагаемой работе над ПС. Такая достоверность оценок обусловлена уровнем неопределенности на данном этапе знаний о конечном содержании и возможном размере программного продукта. Хотя на рис.1 представлена симметричная картина, общая тенденция состоит в том, что на начальных этапах оценки затрат чаще всего занижаются.

Эта неопределенность уменьшается по мере детализации и углубления содержания и функций проекта, как только фиксируются конкретные принципы функционирования и концепция ПС. На этом этапе оценка достоверности размера уменьшается приблизительно до 40%.

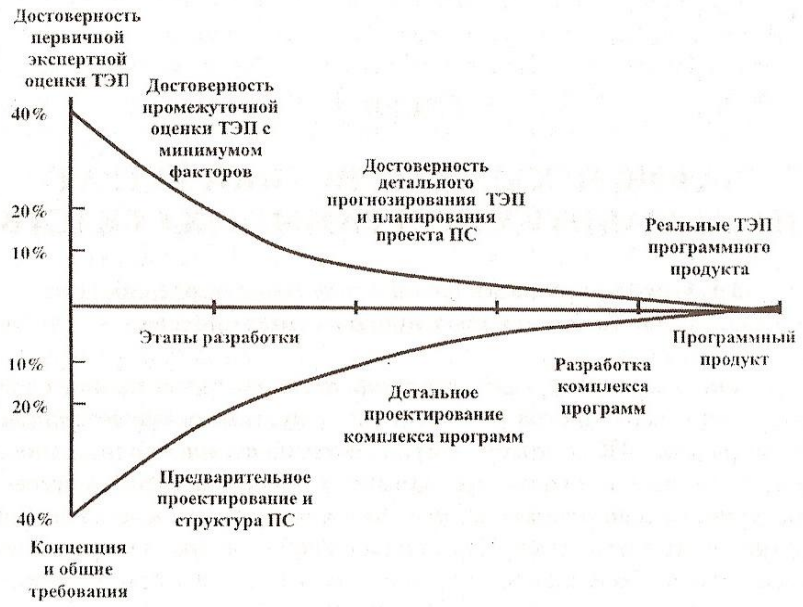


Рис.1

Это вполне объяснимо, поскольку еще не уточнены структура и многие детали проекта. Эти вопросы могут быть разрешены во время разработки структуры и спецификаций требований к ПС и тогда можно оценить размер ПС с точностью до 15 -20%. После завершения разработки и подтверждения проектных спецификаций при детальном проектировании комплекса про-

грамм может быть определена структура внутренних данных и функции программных компонентов. На этом этапе оценка размера и трудоемкости проекта может составить около 10%. Неопределенности оценок могут быть обусловлены: особенностями конкретных алгоритмов, управления их работой, обработки ошибок, инициализации и завершения сеансов работы и т.д. Эти уточнения размеров ПС и компонентов могут быть решены к концу детального проектирования, однако при этом сохраняется неопределенность оценки размера комплекса программ и его трудоемкости порядка 5 - 10%, связанная с тем, насколько хорошо программисты понимают спецификации, в соответствии с которыми они должны кодировать программу. Основной вывод, вытекающий из рис.1, состоит в необходимости быть последовательным при определении исходных данных при оценке ТЭП для различных компонентов программного продукта и этапов проектирования.

В общем случае желательно достигать ***сбалансированного*** набора целей оценивания ТЭП комплекса программ, который бы давал примерно одинаковую величину уровня неопределенности для всех исходных данных и компонентов ПС. По мере выполнения проекта, оценки ТЭП необходимо пересматривать и изменять, когда это становится выгодным или необходимым. Можно начинать с определения оценок трудоемкости в соответствии с точностью определения размера ПС по данным спецификаций требований с учетом минимума факторов, но при дальнейшем анализе уточнять оценки с точностью деталей функционирования и с учетом влияния совокупности важнейших факторов и характеристик проекта.

Рис.2

При оценках ТЭП целесообразно учитывать:

- цели оценивания ТЭП должны быть согласованы с потребностями в информации, способствующей принятию решений на соответствующем этапе проекта ПС;

- достоверность оценок ТЭП должны быть сбалансированы для различных компонентов системы и величина уровня неопределенности для каждого компонента должна быть примерно одинаковой, если в процессе принятия решения все компоненты имеют одинаковый вес;

- следует возвращаться к предшествующим целям оценивания и изменять их, когда это необходимо для ответственных бюджетных решений, принимаемых на ранних этапах и влияющих на следующие этапы.

Измерение размера, оценка и составление графика разработки сложным образом переплетаются в процессе планирования проекта. На самом деле довольно сложно создать реальный график (учитывающий обязанности исполнителей, их зависимости, перекрытие действий, а также дату поставки

продукта) без информации об объеме трудозатрат, требуемых для выполнения каждой задачи (например, определения нагрузки сотрудников на месяц). Достаточно трудно оценить объем трудозатрат, необходимых для выполнения задачи, без достоверной информации относительно *ее размера.* Таким образом, измерение размера (сложности) предшествует оценке ТЭП, а эта оценка, в свою очередь, предшествует составлению графика работ. Каждое из этих важных действий проекта может быть выполнено с помощью различных методик. Недостаточно достоверные оценки влекут проблемы взаимодействия разработчика с заказчиком и увеличивают степень риска проекта.

В индустрии ПС часто обращаются к использованию метрики *LOC,* единицы измерения, хорошо знакомой практикам в области разработки ПС. Они находят ее комфортной и простой в применении. Какая бы технология не использовалась, оценка размера будущего продукта является весьма важной, поскольку она является частью одной наиболее важной задачи проекта: *установка реальных ожиданий.* Нереальные ожидания, основанные на неточных оценках, представляют собой одну из *частых причин провала проектов.* Зачастую причина кроется не в недостаточной производительности команды проекта, а в неточном предвидении уровня этой производительности и размера проекта. Точное оценивание ТЭП обеспечивает улучшенный контроль над проектом и является жизненно важным в деле проектного менеджмента. Для обеспечения точного оценивания в первую очередь следует иметь представление относительно размера продуцируемого ПС. Эта величина должна определяться на ранних стадиях цикла разработки и выражаться в единицах, которые являются достаточно простыми.

Исходные данные реальных завершенных разработок для оценивания ТЭП, собираются, накапливаются и обрабатываются, с начала 70-х годов в разных отечественных организациях и за рубежом. Они позволили получать и прогнозировать основные обобщенные технико-экономические показатели процессов разработки ПС. При этом компоненты операционных систем, драйверы, средства контроля и тестирования, а также повторно используемые компоненты обычно не учитывались при оценке размера вновь созданных комплексов программ и трудоемкости их полной разработки. Поэтому технико-экономические показатели проектов этого периода можно отнести к полностью оригинальным разработкам комплексов программ. При этом обычно рассматривался полный технологический процесс разработки ПС от начала подготовки технического задания (ТЗ) до завершения испытаний базовой версии ПС. Учитывались все категории специалистов, участвующих в создании программ и обеспечивающих разработку, а также все виды работ, связанные с созданием программного продукта на

выделенном интервале времени. Теоретические работы и системный анализ до подготовки ТЗ в значениях ТЭП не учитывались.

Наиболее полно и подробно *основные закономерности и влияние факторов на технико-экономические показатели* процессов разработки сложных ПС в 70 - 80 годы, представлены в материалах Б.У. Боэма под названием «Инженерное проектирование программного обеспечения» для более десяти моделей прогнозирования. В 1981 году на основе исследования процессов разработки 63 проектов ПС опубликована модель прогнозирования ТЭП под названием КОМОСТ. В последующем, эта модель развита, детализирована и опубликована как COCOMO, а в 2000 году под названием COCOMO II. В этой модели на основе анализа более 160 реальных проектов разработки комплексов программ различной сложности уточнены рейтинги влияния выделенных факторов на основные ТЭП. Эти данные ниже используются и рекомендуются как базовые для прогнозирования затрат при создании ПС.

Кроме того, в 1988 году опубликованы результаты анализа ТЭП комплексного проекта ПРОМЕТЕЙ на основе обобщения результатов разработки свыше 250 отечественных проектов сложных ПС, отдельные фрагменты которых также использованы ниже. В общем случае для оценки технико-экономических характеристик новых проектов *необходимы исходные данные:*

- обобщенные характеристики использованных ресурсов и технико-экономические показатели завершенных разработок - прототипов ПС, а также оценки влияния на их характеристики различных факторов объекта и среды разработки;

- реализованные и обобщенные перечни выполненных работ и реальные графики проведенных ранее разработок различных классов ПС;

- цели и содержание частных работ в процессе создания сложных комплексов программ и требования к их выполнению для обеспечения необходимого качества ПС в целом;

- структура и содержание документов, являвшихся результатом выполнения частных работ.

Наиболее важен учет и анализ факторов на начальных этапах разработки, когда прогнозируются первичные совокупные затраты C_p на создание ПС (табл.1). На этих этапах неопределенность оценки параметров и факторов наибольшая (рис.1), тем не менее применение приводимых характеристик

позволяет избегать наиболее крупных ошибок при оценке затрат, которые делаются экспертами без детального анализа влияния факторов. Подобный анализ является базой для рационального распределения ресурсов и для управления их использованием по мере развития проекта. Учет и прогнозирование возможного изменения затрат в зависимости от основных параметров способствует упорядочению процесса разработки ПС и концентрации усилий на тех факторах, которые могут дать максимальный эффект уменьшения затрат в конкретных условиях создания комплекса программ. После проведения структурного проектирования и распределения ресурсов ПС возможна ошибка в оценке затрат на разработку приблизительно в полтора-два раза меньше, а перед программированием она может уменьшаться до 10 - 15%. Такие достоверности можно получить, конечно, только при подробном анализе и оценке влияния важнейших факторов.

Этапы жизненного цикла программ существенно различаются между собой степенью определенности и прогнозируемости их характеристик. Соответственно изменяются возможности подготовки и достоверность планов проведения работ. В процессе разработки сложных программных средств уточняются и детализируются их спецификации требований, функции, структура и характеристики компонентов. Поэтому на начальных этапах, особенно принципиально новых проектов, трудно спланировать точно все последующие этапы. В результате *планирование проводится итерационно* в соответствии с последовательным повышением достоверности и точности сведений об объекте и среде разработки.

Разработка сложных ПС, особенно на начальных и завершающих этапах, характеризуется высокой долей творческого труда. Поэтому трудоемкость и длительность отдельных операций и частных работ существенно зависят от индивидуальных особенностей их исполнителей и характеристик конкретного проекта. Вследствие этого *отсутствует достоверная кооперационная статистика* разработки программ, и пока *невозможно создать типовые нормативы* на большинство частных операций при создании ПС. Отсюда принципиальной особенностью планирования разработки комплексов программ является активное участие руководителей и заказчиков проекта в составлении планов на базе исследованных характеристик прототипов завершенных разработок ПС и их личного опыта. Такие планы должны иметь разумные ограничения в детализации работ на уровне, обеспечивающем необходимую управляемость всего процесса проектирования. В процессах и системах автоматизации планирования и управления разработкой ПС *целесообразно учитывать следующие их особенности:*

- последовательную, иерархическую детализацию и поэтапное уточнение планов и прогнозов в соответствии с повышением полноты и достоверности исходных данных об объекте и среде разработки, получаемых в процессе создания ПС;

- использование прототипов технико-экономических показателей, перечней и графиков частных работ как основных исходных данных для прогнозирования и планирования разработки новых ПС;

- целесообразность и возможность выбора первичного прототипа перечня работ, достаточно адекватного исходным данным проектируемого ПС, и возможность его уточнения проектировщиком;

- регистрацию, обобщение и хранение реализованных рабочих планов и значений ТЭП для их использования в качестве прототипов при планировании последующих аналогичных разработок.

Таким образом, в процессе планирования после использования прототипов для прогнозирования и планирования очередной разработки происходит ее реализация, данные которой могут быть использованы в качестве прототипов для последующих проектов. Тем самым может быть создана последовательно уточняющаяся база данных, позволяющая повышать достоверность прогнозирования и планирования разработок ПС определенного класса. Внутри этого цикла может происходить разработка конкретного ПС, для которой характерно также последовательное уточнение и детализация прогнозов и планов.

В качестве базового варианта целесообразно принять статистические данные ТЭП, перечень работ и документов жизненного цикла создания наиболее сложного встроенного комплекса программ реального времени (табл.1). На основе этих исходных данных могут быть оценены ТЭП для полного цикла разработки ПС конкретного вида в более простых случаях путем исключения из базового варианта работ и документов, в которых отсутствует необходимость. По оставшимся работам могут быть оценены ТЭП для вариантов и выбран из них предпочтительный. Для технико-экономического анализа процесса создания программ важнейшей составляющей являются совокупные трудовые затраты - *трудоемкость C* на непосредственную разработку ПС.

Трудоемкость разработки программных средств наиболее сильно зависит от размера — масштаба программ, выраженного числом операторов на ассемблере или строк на языке программирования высокого уровня. В п. 4

обсуждался вопрос о способах и единицах измерения размера разрабатываемых программ и обоснована целесообразность проводить эти оценки в числе операторов (команд) на языке ассемблера. Реальное изменение создаваемых в настоящее время сложных ПС от 10^4 до 10^7 строк (LOC) определяет диапазон трудоемкости разработки таких программ от человеко-года до десятков тысяч человеко-лет. Широкий диапазон изменения размера создаваемых программ в наибольшей степени определяет значения суммарной трудоемкости их разработки. Подтверждена по большому числу проектов высокая статистическая корреляция между размером комплексов программ в операторах на ассемблере и трудоемкостью их разработки. С другой стороны, отсутствуют какие либо данные о значительном преимуществе других мер размера и сложности при прогнозировании затрат на разработку сложных и сверхсложных ПС. Объем программ в числе операторов на ассемблере характеризуется простотой и экономичностью определения, а также удобством для расчетов и прогнозирования.

Опыт подсказывает, что по мере увеличения размера комплекса программ возрастают не только абсолютные, но и относительные затраты на разработку каждой строки текста в программе. Вследствие этого затраты труда при разработке одной строки текста в программах разного объема могут различаться в несколько раз. Согласно материалам М.Х. Холстеда («Начала науки о программах»), трудоемкость разработки программного модуля пропорциональна квадрату размера программы. Модульно-иерархическое построение крупных ПС позволяет замедлить квадратичный рост сложности и соответствующей ей трудоемкости разработки при увеличении размера программ. Суммарную трудоемкость разработки сложного ПС можно представить *двумя сомножителями.* Первый сомножитель является *доминирующим,* он прямо пропорционален размеру программ *П* и отражает практически линейное возрастание трудоемкости создания любых программ при увеличении их размера.

Однако при увеличении размера программ в ряде случаев наблюдается более быстрый рост их сложности разработки, чем описываемый простой линейной зависимостью. Логично предположить, что по мере увеличения размера ПС возрастает относительная трудоемкость разработки каждой строки в программе. Это обстоятельство можно учесть поправочным *вторым сомножителем,* отражающим изменение трудоемкости на разработку каждой строки в программе при увеличении ее размера.

В ряде исследований размер базы данных либо совсем не учитывается, либо включается в объем ПС. Этому способствует также развитие теории сложности в направлении преимущественного анализа функциональной

сложности программ при почти полном игнорировании сложности данных и их влияния на технико-экономические показатели. В статистической теории сложности программ показано, что для программных модулей и относительно небольших групп программ велика корреляция между числом имен переменных и числом операторов в программе. Однако для сложных и сверхсложных ПС эта корреляция меньше, что определяет необходимость *разделения ПС на два типа:* на осуществляющие преимущественно логическую обработку относительно небольшого потока данных и на информационно-поисковые системы при наличии больших баз данных и относительно простой их логической обработке. Для характеристики этих типов программ может использоваться отношение числа имен переменных к числу строк или операторов текста программ. Для первого типа ПС это отношение не превышает десяти процентов, а для второго оно может значительно превышать единицу и достигать десятков и сотен. Затраты при разработке ПС второго типа оказываются зависящими от размера базы данных, который влияет на сложность взаимодействия программ с данными.

Хотя размеры базы данных для сложных ПС по числу типов имен переменных изменяются в очень широких пределах, приблизительно от 10^3 до 10^8 , их непосредственное влияние *на трудоемкость разработки строки* в программе даже при числе переменных, в десятки раз превышающем размер программы, оценивается на уровне 10%. При этом степень этого влияния трудно формализовать, так как большую роль играет структура базы данных и ее функциональное назначение. Поэтому далее этот фактор отдельно не учитывается и только для очень больших и сложных структур баз данных рекомендуется увеличивать трудоемкость на десяток процентов.

Накопленный опыт создания ПС позволил выделить группы факторов, влияющих на выбор технологии и на затраты *С* (рис.2). *Абсолютная величина С,* так же как и длительность разработки, зависят от ряда факторов, которые могут изменять их в различных направлениях. Наибольшее влияние на них оказывает размер ПС, который из всех параметров изменяется в самом широком диапазоне. Поэтому при первичной оценке непосредственных затрат и длительности полного цикла разработки сложных ПС размер программ используется в качестве *базового доминирующего параметра.* Остальные факторы можно учитывать поправочными коэффициентами при уточнении интегральных показателей.

Для совокупностей ПС первого и второго классов, исследовалась зависимость трудоемкости разработки программ *С* от их объемов - *П.* Для аппроксимации зависимости трудоемкости от размера ПС наиболее часто использована *степенная функция* вида:

$$C = A x П^E \qquad (1)$$

При разработке ПС большого размера в значительной степени, должна возрастать сложность разработки по сравнению с ПС малого объема, так как в больших программах существенно усложняются взаимосвязи компонентов по информации и управлению, а также становятся более трудоемкими процессы планирования и управления проектом в ходе разработки. Выдвинутая гипотеза, о возрастании трудоемкости разработки с ростом размера ПС быстрее, чем по линейному закону, справедлива, если показатель степени в полученном уравнении регрессии $E > 1$. По методу наименьших квадратов в ряде работ определены коэффициенты A и E в уравнениях степенной регрессии, показывающие характер зависимости трудоемкости от размера ПС. В таблице 3.1. представлены значения коэффициентов регрессии для моделей КОМОСТ, СОСОМО и ПРОМЕТЕЙ, для основных классов проектов программных средств. Выражение (1) с использованием этих коэффициентов и значений $П$ размера ПС в тысячах строк ассемблера рекомендуется для прогнозирования трудоемкости полной разработки в человеко-месяцах.

Таблица 2

Коэффициенты моделей для оценки трудоемкости разработки программных средств

Коэффициент А	Коэффициент Е	Модель и тип программных средств
2,4	1,05	Базовая - КОМОСТ
3,6	1,20	Детализированная модель СОСОМО:
3,0	1,12	- встроенный;
2,4	1,05	- полунезависимый;
		- независимый.
2,94	1,15	СССОМО 11.2000 Крупный проект

		100 KSLOC
		ПРОМЕТЕЙ
10,0	1,21	
6,1	1,17	Системы реального времени; Информационно-поисковые системы.

При разработке крупномасштабных ПС делаются большие затраты на создание технологии, средств автоматизации и унификации разработки, чем при разработке малых ПС. Небольшие ПС часто разрабатываются неопытными коллективами, которые к тому же пренебрегают автоматизацией технологии и применением современных методов структурного проектирования комплексов программ. Так как малые ПС во многих случаях относятся исторически к первому временному периоду — 70 - 90-е годы, когда уровень автоматизации технологии был низок, то и трудоемкость их разработки была достаточно высокой. Эти обстоятельства приводят к тому, что возрастает трудоемкость создания относительно небольших. ПС, а рост суммарных затрат на разработку крупных ПС замедляется, что отражается на величине показателя степени *E,* значения которого в некоторых анализируемых выборках иногда получены меньше единицы.

Если бы представилась возможность получить ТЭП по однородной выборке ПС разного объема, разработанных по единой технологии на более или менее одном интервале времени, то, конечно, трудоемкость возрастала бы при увеличении *П* с коэффициентом $E > 1$. На практике часто пользуются упрощенной *линейной зависимостью трудозатрат* от размера ПС *(*$E = 1$**)**. Такое упрощение при недостаточном объеме статистических данных и отсутствии сведений по заранее обусловленным (управляемым) значениям факторов разработки ПС иногда можно считать допустимым.

На рис. 3 по уравнениям регрессии (1) построены в логарифмическом масштабе зависимости трудозатрат от размера для ПС двух классов. Первый (встроенные - СРВ) и второй (ИПС) классы ПС, отчетливо различаются по трудоемкости разработки. Более высокой точности оценки трудоемкости разработки только по одной переменной - размеру ПС, по-видимому, невозможно получить, так как процесс разработки зависит от большого числа факторов, которые следует учитывать при оценке трудоемкости. Наибольшие трудозатраты обычно необходимы для разработки крупномасштабных комплексов программ реального времени, так как данный класс программ используется в наиболее ответственных автоматизированных системах.

Затраты на разработку *С* и объем программ *П* могут быть связаны через показатель интегральной *средней производительности труда* разработчиков *Р*.

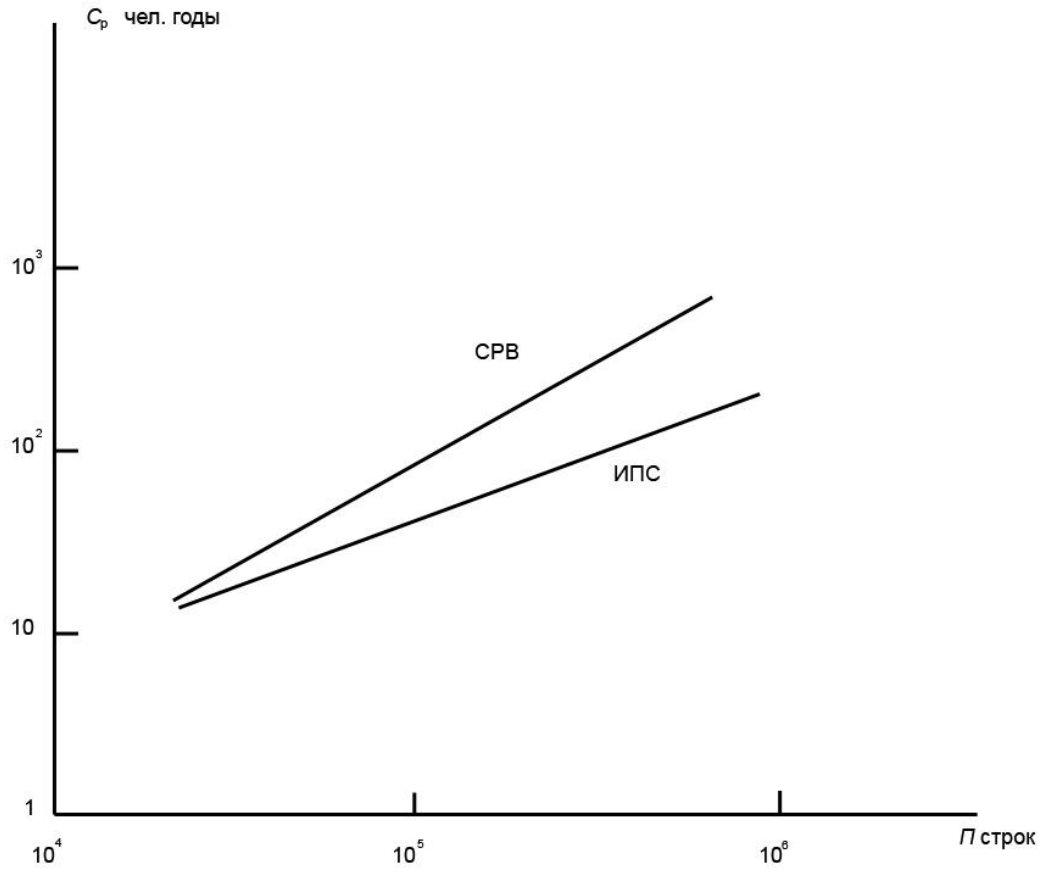


Рис.3

Для учета влияния на *С* различных факторов удобно пользоваться коэффициентами (рейтингами) изменения трудоемкости (КИТ) - *М(i, j)*, учитывающими зависимость *j*-го фактора от *i*-й составляющей совокупных затрат. В них входят факторы процесса непосредственной разработки, факторы программной и аппаратурной оснащенности, а также квалификация специалистов. Непосредственно затраты на разработку можно представить как частное от размера ПС и производительности труда *Р* = 1 / *А,* корректируемой произведением коэффициентов изменения трудоемкости (КИТ - *М (i, j)*):

$$\frac{П^{E}}{P} \quad C = x \prod M(i,j) = A \; x \; ПЕ \; x \prod M(i,j) \qquad (2)$$

$i,j \quad i,j$

Длительность разработки программных средств является важнейшим технико-экономическим показателем, поскольку часто она определяет общие сроки разработки систем, а значит, быстроту реализации идей в различных областях автоматизации. В таблице 3 за начало разработки ПС принят момент начала создания технического задания (ТЗ), а за окончание — завершение испытаний программного продукта в целом или момент предъявления его на испытания.

Таблица 3
Коэффициенты моделей для оценки трудоемкости разработки программных средств

Коэффициент G	Коэффициент H	Модель и тип программных средств
2,5	0,38	Базовая - КОМОСТ
		Детализированная модель СОСОМО:
2,5	0,32	- встроенный;
2,5	0,35	- полунезависимый;
2,5	0,38	- независимый.
		СССОМО 11.2000
3,67	0,328	Крупный проект 100 KSLOC
		ПРОМЕТЕЙ
3,51	0,31	Системы реального времени;
3,78	0,28	Информационно-поисковые системы.

Диапазону размеров современных ПС в три-четыре порядка (до 10 млн.

строк) соответствуют приблизительно такие же диапазоны изменения трудоемкостей и стоимостей их разработок. Однако, очевидна принципиальная нерентабельность разработки даже очень сложных ПС более 5 лет. С другой стороны, программы даже в несколько тысяч строк по полному технологическому циклу с испытаниями как продукции редко создаются за время, меньшее, чем полгода-год. Таким образом, вариация длительностей разработок ПС много меньше, чем вариация их трудоемкостей, и не превышает десятикратный диапазон. Длительности разработок *Т* ограничены сверху и снизу, и одним из основных факторов, определяющих эти границы, является объем программ – *П.*

Относительный *«консерватизм»* значений длительностей по сравнению с трудоемкостью определяется объективной необходимостью создавать ПС в рациональные сроки.

Любые ПС должны поступать на эксплуатацию до того, как в них пропадает необходимость. Их цели, концептуальная основа и алгоритмы не должны устареть за время разработки. Отсюда появляется *верхний предел* допустимых длительностей разработки. Этот верхний предел не может иметь единственное значение для любых классов и объемов ПС. Однако недопустима его вариация в том же диапазоне, что и размер. Поэтому на практике по мере возрастания размеров ПС увеличиваются коллективы специалистов-разработчиков, что обеспечивает основной прирост необходимой трудоемкости. Чем крупнее создаваемое ПС, тем большие усилия обычно прилагаются для автоматизации и совершенствования технологии разработки. Это также способствует замедлению роста длительностей разработки, однако по мере увеличения сложности программ, длительность их разработки все же заметно возрастает.

Стремление ограничивать длительность реальных разработок ПС приводит к объективному формированию верхнего предела, за которым распространяется зона *«нерациональных»* длительностей, зависящих от размера и трудоемкости ПС. Даже для довольно сложных ПС, имеющих размер *свыше* 500 тыс. строк, вряд ли допустима длительность разработки более 3-5 лет. Большие длительности, иногда имеющиеся на практике, обусловлены в основном низкой квалификацией разработчиков и заказчиков, недостаточной автоматизацией технологии, малым коллективом специалистов и рядом других, преимущественно организационных и технологических причин. Подобные ситуации чаще встречаются при относительно небольших разработках (10 - 50 тыс. строк), когда у руководителей и коллектива мал опыт их проведения, следствием чего является избыточный оптимизм в начале разработки, а также пренебрежение технологией и организацией работ.

Границу снизу определяют естественный технологический процесс коллективной разработки и необходимость выполнения ряда взаимоувязанных работ на последовательных этапах, которые обеспечивают получение ПС требуемого качества. Подготовка текстов

программ, их тестирование, комплексирование, документирование и испытания могут проводиться только последовательно, и каждый этап требует определенного времени. Попытки форсировать отдельные этапы работ, как правило, приводят к увеличению продолжительности других этапов. Подключение дополнительных специалистов увеличивает затраты на разработку и только в некоторых пределах дает сокращение сроков. В некоторых случаях увеличение числа специалистов *может давать обратный эффект* - длительность разработки увеличивается вместе с увеличением затрат.

Под воздействием перечисленных факторов формируется объективный минимум длительностей - граница снизу области *«невозможного»*, зависящая в первую очередь от объема, разрабатываемых ПС. Нижняя граница длительностей еще более *«консервативна»,* чем верхняя. Изменение этой границы возможно в небольших пределах только за счет совершенствования технологии, повышения программной и аппаратурной оснащенности разработки, а также роста коллективной квалификации разработчиков и заказчиков.

Практическая граница *«нерациональных»* длительностей имеет значения, приблизительно вдвое большие, чем значения границы *«невозможных»* длительностей, при том же объеме ПС. Это означает, что даже большие усилия по автоматизации и организации разработки программ приводят к сокращению длительностей только в 2 - 3 раза, в то время как трудоемкость уменьшается значительно больше. По результатам реальных разработок может быть оценена средняя или наиболее вероятная длительность разработок ПС определенного класса при заданных условиях. Конкретное распределение длительностей зависит от исходных данных, имеющихся в базе данных технико-экономических показателей завершенных разработок, и от метода их усреднения.

Для конкретного планирования длительностей создания ПС определенных классов необходимо для каждого предприятия исследовать и обобщать технико-экономические показатели реальных разработок, однородных по технологии и другим условиям. Такие обобщения при конкретных условиях разработок позволяют получить опорные абсолютные значения длительностей для некоторых размеров ПС. Эти абсолютные значения могут быть использованы для расчета коэффициентов регрессии с целью прогнозирования длительностей разработок на базе выявленных закономерностей и реальных опорных значений для конкретных условий разработки.

Обобщенные данные длительности разработки *T* по классам программ в ряде работ *аппроксимировались уравнениями регрессии* по методу наименьших квадратов в зависимости от размера ПС и от трудоемкости их

разработки (таблица 2):

$$T = G \times C^{H}. \qquad (3)$$

Зависимости *Т* от размера программ *П* значительно различаются для классов ПС. Это определяется различием сложности классов программ, применяемых языков программирования и единиц измерения объема ПС, следствием чего является различие значений размера созданных программ при одной и той же длительности и трудоемкости разработки. Чтобы исключить ошибки, связанные с неопределенностью измерения размера программ, исследована *зависимость длительности разработки от ее трудоемкости*. Учитывалась только трудоемкость непосредственной разработки программ *С* без затрат на средства автоматизации разработки. Обработка тех же, что выше, наборов данных позволила получить коэффициенты уравнения регрессии представленные в таблице 2. Средние значения длительности разработки классов ПС практически не различаются в зависимости от трудоемкости разработки программ.

Оценка требуемого среднего числа специалистов для конкретного проекта ПС предварительно может быть рассчитана путем деления оценки величины трудоемкости разработки (2) на длительность разработки (3). Однако рациональное число специалистов, участвующих в проекте ПС распределяется не равномерно по этапам работ. Поэтому целесообразно определять число и квалификацию необходимых специалистов с учетом этапов разработки комплексов программ.

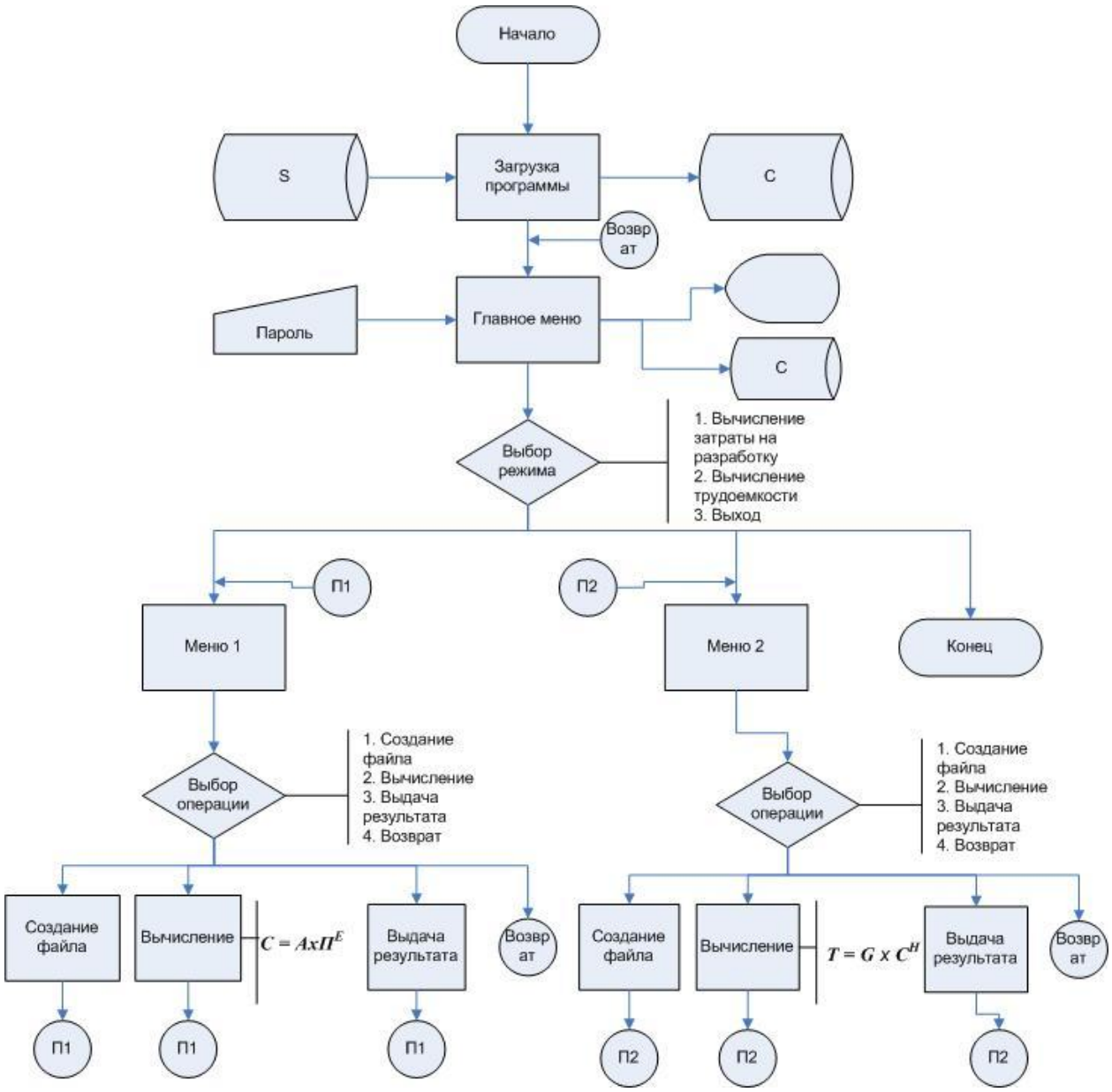
Средняя производительность труда коллектива специалистов при разработке сложного полностью нового комплекса программ *Р* в выражении (2) может служить *ориентиром* для сравнения эффективности труда при создании различных проектов ПС. Эта характеристика, конечно, различается для различных классов, размеров и других параметров комплексов программ, однако диапазон этих различий не столь велик как изменения размера, требований к качеству и других параметров. Так при диапазоне изменения размеров программ СРВ на четыре порядка средняя производительность труда изменяется только в два раза, что в ряде случаев существенно облегчает упрощенные оценки и прогнозирование ТЭП.

При разработке программных модулей и компонентов отдельными специалистами или небольшими группами производительность труда при написании одних и тех же текстов автономных программ может различаться в десяток раз в зависимости от их таланта и трудоспособности и достигать тысяч строк за человеко-месяц. Однако достаточно полное тестирование, документирование, комплексирование и оформление в крупные комплексы программного продукта, приводят к

снижению интегральной производительности до величин в несколько сотен строк текста за человеко-месяц. Для крупных проектов класса СРВ 80-е годы приводятся величины 100 - 150 строк на человеко-месяц, в отечественных проектах в те же годы эта величина приближалась к 80 - 100. Совершенствование технологии, квалификации специалистов и инструментальных средств автоматизации разработки позволили в последние годы повысить среднюю производительность труда при создании полностью новых оригинальных программных продуктов СРВ в несколько раз по экспертным оценкам до величин 300 - 500 строк на человеко-месяц.

При разработке ПС необходимо учитывать, что экономические, временные, вычислительные и другие ресурсы *па разработку и весь ЖЦ программ* всегда ограниченны и используемые затраты для улучшения каждой характеристики должны учитывать эти ограничения. Для рационального распределения этих ресурсов необходимо знать *как отражается изменение затрат на улучшении каждой характеристики качества ПС.* Эта взаимосвязь затрат ресурсов и значений каждой характеристики зависит от назначения, а также от ряда свойств и других особенностей комплекса программ, что усложняет учет влияния таких связей. Тем не менее, выявлены основные тенденции такого взаимодействия, которые могут служить *ориентирами* при выборе и установлении требований к определенным характеристикам качества в конкретных проектах ПС.

Схема работы программы по вычислению ТЭП:



БИБЛИОГРАФИЯ

1. А.М. Вендров. Проектирование программного обеспечения экономических информационных систем. Учебник. М.: «Финансы и статистика». 2000. - 339 с.

2. А.М. Вендров. Практикум по проектированию программного обеспечения экомических информационных систем. М.: «Финансы и статистика». 2002. -190 с.

3. Практические аспекты информатизации. Стандартизация, сертификация и лицензирование. Справочная книга руководителя. Под

редакцией Л.Д. Реймана. М.: 2000. -259с.

4. В.В. Липаев. Качество программных средств. Методические рекомендации. М.: «Янус-К». 2002. – 298с.

5. Боэм Б.У. Инженерное проектирование программного обеспечения: Пер. с англ./Под ред. А.А. Красилова. М.:Радио и связь, 1985.

6. В.В. Липаев, А.И. Потапов. Оценка затрат на разработку программных средств. М.: Финансы и статистика. 1988.

7. С.А. Орлов. Технологии разработки программного обеспечения. Учебник для вузов. М., Санкт-Петербург: «Питер». 2002.

8. Г. Коллинз, Дж. Блей. Структурные методы разработки систем: от стратегического планирования до тестирования. М.: «Статистика», 1980. 260с.:ил.

9. ГОСТ Р ИСО 9127 – 94 «Системы обработки информации. Документация пользователя и информация на упаковке потребительских программных пакетов».

10. ГОСТ 34601 – 90. «Информационная технология. Комплекс стандартов на автоматизированные системы. Автоматизированные системы. Стадии создания».

11. ГОСТ 34601 – 89. «Информационная технология. Комплекс стандартов на автоматизированные системы. Техническое задание на создание автоматизированной системы».

12. ГОСТ 34601 – 92. «Информационная технология. Виды испытаний автоматизированных систем».

13. Информационные системы в экономике. Под ред. Проф. В.В. Дика. Учебник для вузов, М., «Финансы и статистика». 1996. – 270 с.

ПРИЛОЖЕНИЕ

О СТАНДАРТЕ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА ДЛЯ

ДИАЛОГОВЫХ ИТ

Стандарт фирмы IBM. Проектирование пользовательского интерфейса на персональном компьютере.- Вильнюс, 1992

Стандартизация и согласованность интерфейса экономят время пользователя и разработчика.

Пользовательский интерфейс включает в себя три понятия: общение приложения с пользователем, общение пользователя с приложением, язык общения. Язык общение определяется разработчиком программного приложения. Свойствами интерфейса являются наглядность и конкретность. Наиболее распространенный ранее командный интерфейс имел ряд недостатков (многочисленность команд, отсутствие стандарта для приложения), что ограничивало круг его применения. Для преодоления этих недостатков были предприняты попытки его упростить (например, NC). Однако настоящим решением проблемы стало создание графической оболочки для ОС. В настоящее время практически все распространенные ОС используют для своей работы графический интерфейс. Примером здесь может служить интерфейс, разработанный в исследовательском центре Пало Альто фирмы Xerox для компьютеров Macintosh фирмы Apple. Немного позже была разработана графическая оболочка под названием Microsoft Windows, реализующая технологию WIMP и удовлетворяющая стандарту CUA. Новшеством были применение мыши, выбор команд из меню, предоставление программам отдельных окон, использование для обозначения программ образов в виде пиктограмм.

Удобство интерфейса и богатство возможностей делают Windows оптимальной системой для повседневной работы. Приложения, написанные под Windows, используют тот же интерфейс, поэтому его единообразие сводит к минимуму процесс обучения работе с любым приложением. Выход на рынок Windows-95 еще более упростил работу пользователя, так как интерфейс стал еще более простым, документированным, включающим встроенные коммуникационные возможности.

Одной из важнейших функций интерфейса является формирование у пользователя одинаковой реакции на одинаковые действия приложений, их согласованность. Согласование должно быть выполнено в трех аспектах: ***физическом***, который относится к техническим средствам (пока отсутствует), ***синтаксическом***, который определяет последовательность и порядок появления элементов на экране (язык общения) и последовательность запросов (язык действий), ***семантическом***, который обусловлен значениями элементов, составляющих интерфейс. Согласованность интерфейса экономит время пользователя и разработчика. Для пользователя уменьшается время изучения, а затем использования системы, сокращается число ошибок, появляется чувство комфортности и уверенности. Разработчику согласованный интерфейс позволяет выделить общие блоки интерфейса, стандартизировать отдельные элементы и

правила взаимодействия с ними, сократить время проектирования новой системы.

Разработка пользовательского интерфейса состоит из проектирования панелей и диалога. Панель приложения разделена на три части: место действий, тело панели, область функционирования клавиш.

Преимущество использования ***меню действий*** (и выпадающего меню) заключается в том, что эти действия наглядны и могут быть запрошены пользователем установкой курсора, функциональной клавишей вводом команды либо каким-то другим простым способом. На цветном экране меню действий имеет обычно другой цвет по отношению к цвету панели. На монохромном экране используется сплошная линия для его отделения. Меню действий содержит объекты, состоящие из одного или нескольких слов. Два последних из них резервируются для действий ‘‘выход’’ и ‘‘справка’’. Размещаются объекты слева направо по мере убывания частоты их использования. Возможны системы с многоуровневой системой выпадающих меню, но оптимальное число уровней – три, т.к. иначе могут появиться трудности в понимании многоуровневых меню.

Тело панели содержит элементы тела панели: разделители областей, идентификатор и заголовок панели, инструкцию, заголовки столбца, группы, поля; указатель протяжки; область сообщений и команд; поля ввода и выбора (см.прил.2.1.).

Область функциональных клавиш – необязательная часть, показывающая соответствие клавиш и действий, которые выполняются при их нажатии. В области функциональных клавиш отображаются только те действия, которые доступны на текущей панели.

Для указания текущей позиции на панели используется курсор выбора. Для более быстрого взаимодействия можно предусмотреть функциональные клавиши, номер объекта выбора, команду или мнемонику.

Разбивка панели на области основана на принципе ‘‘объект-действие’’. Этот принцип разрешает пользователю сначала выбрать объект, затем произвести действия с этим объектом, что минимизирует число режимов, упрощает и ускоряет обучение работе с приложениями и создает для пользователя комфорт. Если панель располагается в отдельной ограниченной части экрана, то она называется окном, которое может быть первичным или вторичным. В первичном окне начинается диалог, и если в приложении не нужно создавать другие окна, окном считается весь экран. Первичное окно может содержать столько панелей, сколько нужно для ведения диалога. Вторичные же окна вызываются из первичных. В них пользователь ведет диалог параллельно с первичным окном. Часто вторичные окна используются для подсказки. Первичные и вторичные окна имеют заголовок в верхней части окна. Пользователь может переключаться из первичного окна во вторичное и наоборот. Существует также понятие ‘‘всплывающие окна’’, которые позволяют улучшить диалог пользователя с приложением, ведущийся из первичного или вторичного окна.

Когда пользователь и ЭВМ обмениваются сообщениями, диалог движется по одному из путей приложения, т.е. пользователь перемещается по приложению, выполняя конкретные действия. При этом действие необязательно требует от компьютера обработки информации. Оно может обеспечить переход от одной

панели к другой, от одного приложения к другому. Если пользователь перешел к другой панели и его действия привели к потере информации, рекомендуется требовать подтверждение того, следует ли ее сохранять. При этом пользователю предоставляется шанс сохранить информацию, отменить последний запрос, вернуться на один шаг назад.

Путь, по которому движется диалог, называется навигацией. Он может быть изображен в виде графа, где узлы - действия, дуги - переходы. Диалог состоит из двух частей: запросов на обработку информации и навигации по приложению. Часть запросов на обработку и навигацию является унифицированной. Унифицированные действия диалога – это действия, имеющий одинаковый смысл во всех приложениях. Некоторые унифицированные действия могут быть запрошены из выпадающего меню посредством действия “команда” функциональной клавишей. К унифицированным действиям диалога относятся: "отказ", “команда”, “ввод”, “выход”, “подсказка”, “регенерация”, “извлечение”, “идентификаторы”, “клавиши”, “справка” (см. прил.3).

Существующий стандарт закрепляет названия унифицированных действий на английском языке. При переводе на русский названия могут не совпадать в разных приложениях.

СТАНДАРТ ФИРМЫ IBM. ЭЛЕМЕНТЫ ЭКРАНА

Минимальные единицы панели называются элементами тела панели. К ним относятся: разделители областей; идентификатор панели, заголовок панели, инструкция, заголовок столбца и группы, заголовок поля; указатели протяжки; область сообщений; область команд; поле ввода; поле выбора.

Разделители делят тело панели на области. В качестве разделителей могут использоваться цветовые границы, линии, простые строки или столбцы, заголовки столбцов.

Идентификатор панели – защищенная алфавитно - цифровая информация (имя), предназначенная для идентификации панели. По умолчанию идентификатор выключен (не высвечивается). Действия с идентификатором осуществляются с помощью функциональных клавиш.

Заголовок панели сообщает пользователю о том, какая информация содержится в теле панели. Панель должна иметь заголовок, если это не оговорено другими правилами. Сообщения в всплывающем окне могут не иметь заголовка. Если другие области тела панели должны протягиваться, то заголовок образует самостоятельную область и не протягивается. Он может содержать переменную информацию, но не может содержать поле выбора или поле ввода.

Инструкция сообщает пользователю, что нужно сделать и как продолжить работу.

Заголовок столбца идентифицирует поле ввода или выбора, если все объекты столбца принадлежат к одному типу. Если информация столбца протягивается, то заголовок образует отдельную область и не протягивается. В горизонтальной протяжке заголовок столбца протягивается вместе с информацией столбца.

Заголовок группы указывается, если имеется несколько столбцов с полем

выбора или ввода.

Заголовок поля обозначает поле выбора, поле ввода поле переменной информации.

Указатель протяжки используется в тех случаях, когда выполняется скроллинг, обозначается стрелками, специальной линейкой или текстовыми указателями.

Сообщения обеспечивают пользователя информацией, которую он не просил явно, но которая, по мнению разработчика, ему необходима. Сообщения делятся на информационные, предупреждающие и критические. Информационные сообщения описывают состояние системы. Ответы пользователя не требуются. Они используются, чтобы сообщить пользователю, что обработка продолжается, завершилась, изменилось содержание панели, а также в многозадачных системах, когда одновременно выполняется несколько задач. Предупреждающие сообщения обращают внимание пользователя на состояние системы, которое требует его вмешательства. Пользователь в ответ может выполнить какое либо действие, либо пренебречь этим сообщением. Критические сообщения указывают условие, при котором продолжение работы невозможно без вмешательства пользователя (произошла ошибка, исключительное состояние системы и т.п.). При этом измененная информация не сохраняется и пользователь в явном виде должен указать, нужно ли ее сохранять. Сообщения размещаются во всплывающих окнах, в нижней части тела панели над областью функциональных клавиш и областью команд, если они есть. Критические сообщения должны выдаваться только во всплывающих окнах. После действия пользователя сообщение автоматически удаляется. При выдаче предупреждающих и критических сообщений может предусматриваться подача звукового сигнала.

Область команд можно разместить во вторичном, всплывающем окне или в нижней части панели над областью функциональных клавиш. Она должна содержать заголовок и поле ввода.

Область команд и меню действий не противоречат и не исключают друг друга. Функции, доступные из меню действий и из области команд, должны называться одинаково. Для упрощения ввода команд можно использовать уже знакомое нам меню действий. Это сокращает время выбора команды. При этом действие содержится в выпадающем меню, а параметры во всплывающем окне.

Поле выбора – это обобщенное определение набора взаимосвязанных объектов (слов, пиктограмм и их сочетаний). Когда пользователь выбрал объект, приложение визуально отмечает это при помощи цвета, подсветки или символа, размещаемого перед выбранным объектом. Цвет и подсветка называются выделением, а символ - указателем выбора. Различают поля ***однозначного, многозначного и расширенного выбора.***

В поле ***однозначного*** выбора пользователь должен выбрать только один объект. Если на панели несколько полей выбора, то пользователь явно указывает поле выбора.

В поле ***многозначного*** выбора пользователь может выбрать один, несколько объектов или ничего. Каждый объект выбирается явно. Для выбора нескольких объектов нажимается “/” или пробел. Когда пользователь выбирает доступный

объект поля выбора, он отображается как “выбран”, даже если текущая панель удаляется. Когда пользователь выбирает недоступный объект, появляется всплывающее окно с сообщением, почему объект недоступен. Объект выбора считается доступным, если пользователь может его выбрать, и недоступным, если текущее состояние приложения не позволяет выбрать этот объект ввиду невыполнения каких либо условий. Недоступные объекты обычно выделяются уменьшение яркости. Наряду с недоступными некоторые поля могут быть неуполномоченными, или несанкционированными. Для доступа к ним требуется обладать специальным правом.

В поле ***расширенного*** выбора пользователь выбирает объект, и к нему во всплывающем или вторичном окне дается пояснение (расширение). Если в первоначальном состоянии имеется один объект, то это поле рассматривается как поле однозначного выбора, а если есть несколько объектов, то многозначного.

Объекты поля выбора могут представляться тремя способами: по столбцам, выровненным влево; в одной строке; в несколько столбцов, которые разделены пробелами. Каждое поле может быть идентифицировано заголовком поля, столбца, группы, протягиваемого поля выбора. Объект поля выбора можно представить одним или несколькими словами, пиктограммами или их сочетанием. Поля однозначного выбора могут нумероваться, если их не более 9. При использовании мнемоники каждому объекту присваивают уникальную букву. Мнемоника активна только в том поле выбора, на которое указывает курсор. Протягиваемые поля выбора используются только для списка объектов, размещенных в одном столбце.

Курсор выбора может быть в виде контура, линейки, подчеркивания, изменения цвета.

Поле ввода - это место на панели, в которое пользователь вводит информацию. Обычно поле ввода имеет заголовок поля и, если необходимо, заголовок столбца. Когда курсор установлен в требуемое поле ввода, он называется текстовым. Поле ввода может быть протягиваемым.

При первоначальном отображении панели каждый элемент должен иметь свои яркость и цвет. По мере углубления диалога для показа текущего состояния объекта, с которым работает пользователь, цвета и эффекты выделения могут изменяться.

Рекомендуемая палитра:

Панель в первичных и вторичных окнах, за исключение панели “Справка”,- белая. Панель в окне справка – синяя. Панель во всплывающих окнах нечетного уровня – голубая, а четного уровня – белая. Ошибки выделяются красным. Предупреждения об ошибках – желтые. Критические сообщения – красные.

СТАНДАРТ ФИРМЫ IBM. УНИФИЦИРОВАННЫЕ ДЕЙСТВИЯ ДИАЛОГА

Действие “отказ” должно включаться во все выпадающие меню (при этом отменяется панель, в которой помещается курсор), во все всплывающие окна, за исключение информационных сообщений. Рекомендуется включать “отказ” во все

панели, составляющие некоторую единицу выполняемой работы.

Действие “ввод” включается, если панель содержит поле ввода или более одного поля выбора (многозначный выбор).

“Выход” используется, если пользователь может завершить выполнение текущей функции в текущей панели. “Выход” должно быть в меню действий первичного окна и области функциональных клавиш. При выборе данного действия управление возвращается на предыдущий уровень иерархии. Этот пункт необходим в каждом выпадающем меню. Для предсказания появления всплывающего окна используется многоточие. Оно подтверждает выход и при необходимости напоминает пользователю, что не сохраненных данных.

Выход из приложения осуществляется по какой-либо клавише. При выходе на наивысший уровень назначается клавиша для появления выпадающего меню, содержащего действия “продолжить” или “окончательный выход”, для подтверждения выхода.

Унифицированное действие “справка” должно содержать следующие действия в выпадающем меню в порядке расположения:

Как получить справку. Для этого используют всплывающее меню с информационной панелью, где сообщается, как получить справку.

Общая справка. Обеспечивает общую справку о панели, из которой она затребована.

Описание клавиш. Должен быть представлен список используемых функциональных клавиш с их функциями.

Указатель. Содержит перечень имеющихся в приложении справок в алфавитном порядке. Тот же список отображается при выборе клавиши “указатель” в панели “справка”.

Учебная справка. Предусматривается в режиме приложения и должна быть последней в выпадающем меню “Справка”.

“Справка” должна быть включена во все панели и меню действий. Если меню отсутствует, то справка появляется в области функциональных клавиш.

“Подсказка” сообщает пользователю, как завершить работу с полем ввода. Для получения подсказок пользователь устанавливает курсор выбора в то поле ввода, список допустимых значений которого должен быть высвечен. По действию “подсказка” появляется всплывающее меню с панели типа меню. Меню может содержать поля однозначного и многозначного выбора. После выбора одного или нескольких объектов всплывающее окно исчезает, а выбранные объекты копируются в поле ввода, как если бы пользователь выбрал эти значения на клавиатуре. Если пользователь выбрал несколько объектов поля многозначного выбора, то порядок их следования определяется приложением. Пользователь должен иметь возможность отказаться от выбора объекта в всплывающем окне подсказки. Отказ не влияет на поле ввода. Если пользователь запрашивает подсказку, не установив курсор выбора в поле ввода, никакого действия не происходит. Если курсор выбора установлен в поле ввода и пользователь просит подсказку, а приложение не предусматривает ее, то выдается звуковой сигнал и во всплывающем окне или в области сообщений этой панели появляется сообщение, что приложение не поддерживает эту подсказку.

“Подсказка” включается, когда панель содержит поле ввода, и зависит от позиции курсора. Во всплывающем окне высвечивается одно или несколько значений, которые могут быть выбраны пользователем для заполнения поля ввода.

“Регенерация” зависит от типа панели, из которой запрашивается это действие. В панели ввода восстанавливается исходное состояние панели без учета того, что было набрано пользователем с момента последнего заполнения или обработки этой панели. В панели, отражающей текущее состояние объектов, повторно выводится содержимое панели с учетом всех изменений объектов, которые появились с момента последнего отображения этой панели. Если панель позволяет ввести информацию, то “регенерация” выполняется, как в панели ввода. Действие “регенерация” рекомендуется включать в панели, содержащие поля выбора и ввода.

Действие ***“Клавиши”*** изменяет представление области функциональных клавиш в нижней части выпадающего меню. По умолчанию появляется длинная форма представления, по запросу – краткая. При повторном запросе краткая форма исчезает и появляется длинная и т.д.

Посредством действия ***“Идентификатор”*** пользователь запрашивает включение или выключение идентификатора панели.