Лекции по курсу «Технология программирования».

Два подхода к разработке программных средств

Существует два подхода к разработке программных средств:

- для собственных нужд;
- промышленная разработка на заказ с коммерческим использованиеми тиражированием.

При этом можно выделить следующие особенности:

1. *разработка ПС для собственных нужд:*

    1) Знание предметной области разработчиком, как правило, исчерпывающее (т.к. он является специалистом в этой области.);
    2) Объемы программных средств сравнительно небольшие. Разработка ведется одним либо несколькими разработчиками, тесно связанными производственной деятельностью;
    3) Из-за ограниченного количества разработчиков никакой специальной организации коллектива не требуется;
    4) Оформление документации или не осуществляется, или создается документация ограниченного объема в произвольной форме. Создается она для того, чтобы коллеги, не участвовавшие в разработке, могли с полным пониманием использовать эти программные средства;
    5) Программа, как правило, не имеет целей коммерческого использования, и если распространяется, то исключительно среди коллег занятых схожей деятельностью.

2. *разработка ПС на заказ (производственный подход):*

    1) Знание предметной области коллективом разработчиков полностью отсутствуют, либо имеют ограниченный объем, т.к. разработчики не являются специалистами в этой предметной области. При этом требуется изучение предметной области, ее формальное описание, как правило, на основании строгих нормативных документов и согласованием их с заказчиком. На практике, крупные коллективы разработчиков имеют специализацию по проблемным областям (например: в медицине, в бухучете, в банковском деле и т.п., что в свою очередь сильно облегчает изучение и описание предметной области);
    2) Объем ПС может достигать десятков и сотен тысяч операторов исходного кода, что требует привлечения большого количества работников;
    3) Возникает необходимость организации коллектива разработчиков со своей структурой и должностными обязанностями. При этом структура должна быть отражена во внутренней документации организации;
    4) Большой объем ПС, наличие коллектива разработчиков и, как правило, отсутствие у заказчика знаний о структуре системы и особенностях её функционирования, требуют написания соответствующей исчерпывающей документации. При этом документация подразделяется на 2 части: документация разработчиков и эксплуатационная документация;
    5) Разработка ПС, как правило, носит коммерческий характер. При этом подразумевается возможность отторжения разработанных ПС от разработчиков, т.е. возможность эксплуатации без участия разработчиков;

Каждый подход имеет свою сферу применения, которые отражены в описанных выше особенностях. Для небольших ПС бессмысленно использовать многие особенности промышленной разработки (например: организация коллектива, написание исчерпывающей документации и т.д.). Напротив, промышленная разработка больших ПС требует полного и исчерпывающего соблюдения этих правил.

Основные задачи, решаемые при разработке ПС

1. Для разработки современных ПС при использовании высокоэффективных технологий и их разработки необходимо организовать и скоординировать сам процесс разработки.

   При этом необходимо изучить принципы и условия создания ПС.

   Для этого необходимо решение методологическихзадач:

   a. Определить процесс и этапы разработки ПС;
   b. Определить количество и качество разрабатываемого программного продукта;
   c. Разработать методы оценки длительности разработки ПС и трудозатрат;
   d. Оценить факторы, определяющие производительность труда коллектива разработчиков;
   e. Изучить варианты организации коллектива разработчиков и выбрать наиболее подходящий.

2. Для создания унифицированных методов с целью построения сложных ПС и эффективного использования ресурсов необходимо решение структурныхзадач:

   1) Определить структуры построения ПС или комплекса программ (КП) как сложной системы;
   2) Определить структуры переменных и возможных структур БД;
   3) Разработать метод распределения ресурсов ВС для построения конкретного КП;
   4) Разработать структуры КП, надежно функционирующие при наличии ошибок ПС и в аппаратуре.

3. Для реализации сложных КП конкретным коллективом разработчиков необходимо решение технологических задач:

   1) Создание четкой технологической системы разработки (этапы, объемы выполняемых работ, ответственные лица, методы контроля и т.д.);
   2) Охват технологической системой всего жизненного цикла при разработке КП;
   3) Выбор или создание методов и средств автоматизации разработки сложных КП;
   4) Стандартизация процессов и средств разработки КП.

4. Для того, чтобы вести разработку, имея решенными предыдущие задачи, необходимо решить задачу обеспечения разработчиков аппаратными ПС:

   1) Создание структур ВС как для разработчика, так и для конечного пользователя (мощность ВС разработчика, как правило, многократно превышает мощность ВС пользователя);
   2) Выбор технологических средств разработки (средств автоматизации, компиляции, отладчиков).

Основные понятия и определения, используемые при разработке сложных КП

Система – это комплекс средств, элементов и частей, образующих единое целое и предназначенных для достижения определенных целей.

Мы будем рассматривать сложные системные программы(или КП), предназначенные для решения задач обработки информации в “административных” и управленческих задач или же решение задач реального масштаба времени.

Характерные черты сложных систем:

1. Единая цель функционирования, т.е. решение особого набора функциональных задач, выполняющих единую цель;
2. Большое количество взаимосвязанных компонентов (элементов, частей) при функционировании систем (для КП - операторы, программные модули, подсистемы, данные, массивы данных и т.д.);
3. Наличие иерархической структуры, т.е. подчинение одних компонент другим. Подчиненность касается как программного кода, так и данных;
4. Возможность декомпозиции компонент на более мелкие компоненты, которые в свою очередь могут быть сложными системами;
5. Сложное поведение сложных систем с большим количеством обратных связей между компонентами, т.е. поведение системы зависит не только от внешних воздействий, но и от ее внутреннего состояния;
6. Устойчивость к возмущениям, вызванным как внутренними, так и внешними причинами (для сложных КП - изменения характеристик входных потоков, изменения условий функционирования состава и характеристик компонентов.);
7. Надежность функционирования системы в целом, как программных, так и аппаратных ее частей;
8. Наличие целей системы и критериев качества, которые определяют достоинства системы, в частности сложных КП.

Функционирование сложных систем определяется алгоритмом их работы.

Алгоритм–это совокупность формальных правил, действуя согласно которым можно получить необходимое решение соответствующихзадач за конкретное время.

Программа–алгоритм, заданный на некотором языке, доступном ВС, которая может выполнить эту программу.

Особенности КП СРМВ.

Под программами СРМВ (систем реального масштаба времени) будем понимать такие программы, в которых время используется в качестве одной из текущих переменных, и от которой зависит результат. Примерами СРМВ являются системы, предназначенные для управления движущимися объектами, многими промышленными объектами, а также очень большая часть научных задач, в частности управление физическими объектами.

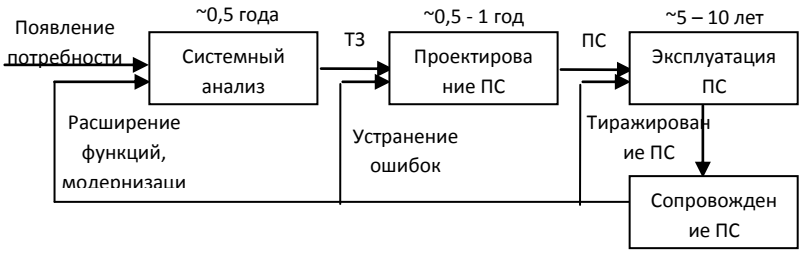
Основные требования к КП СРМВ

1. Выдача информации на объект управления и непосредственный прием информации от объектов управления.
2. Исключительно высокие требования к надежности функционирования систем.
3. Ограничения на временные задержки, иногда весьма строгие.

Жизненный цикл сложных КП

Весь период существования ПО, связанный с подготовкой к его разработке, разработкой, использованием и модификациями, начиная с того момента, когда принимается решение разработать/приобрести/собрать из имеющихся компонентов новую систему или приходит сама идея о необходимости программы определенного рода, до того момента, когда полностью прекращается всякое ее использование, называют ***жизненным циклом ПО***.

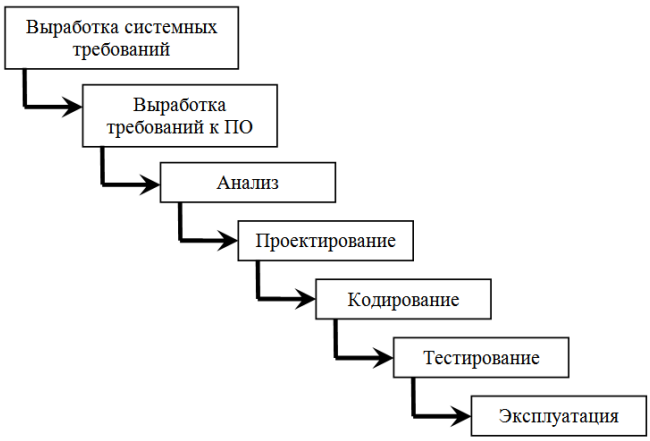
Основные элементы структуры жизненного цикла определяют в виде ***модели жизненного цикла ПО.***



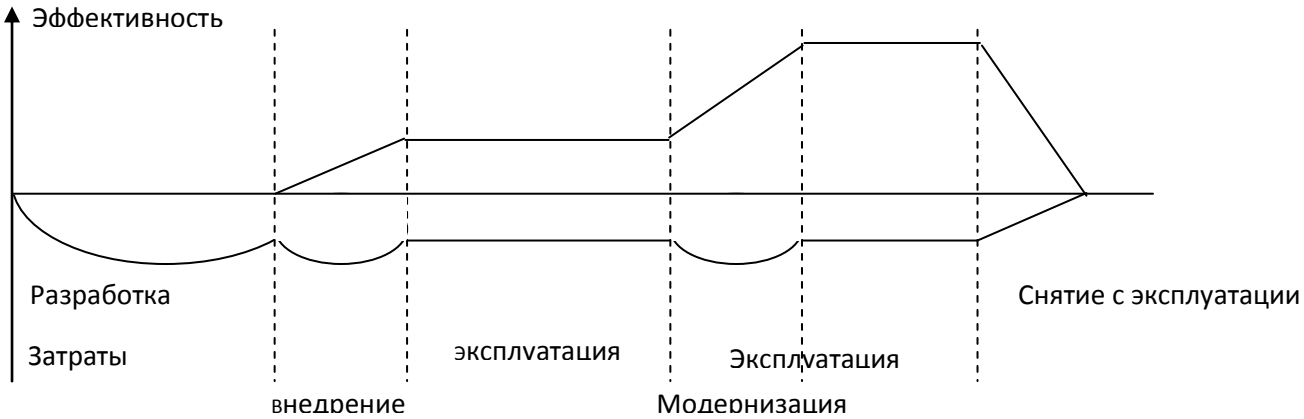
Все временные характеристики на схеме жизненного цикла весьма и весьма условны.

Каскадная модель жизненного цикла

Каскадная (водопадная, waterfall) модель жизненного цикла характерна для тех случаев разработки и эксплуатации, когда состояние предметной области стабильно и достаточно хорошо известно разработчику. Т.е. это характерно для СРМВ, т.е. управляющих систем. Эта модель предполагает последовательное выполнение различных видов деятельности, начиная с выработки требований и заканчивая сопровождением, с четким определением границ между этапами, на которых набор документов, созданный на предыдущей стадии, передается в качестве входных данных для следующей. Можно выделить следующую последовательность разработки:



Последовательность разработки согласно «классической» каскадной модели



На графикепредставлена зависимость также весьма и весьма условная (например, не одна, а несколько модернизаций).

Снятие с эксплуатации в подавляющем большинстве случаев осуществляется параллельно с установкой новой системы.

Работать в соответствии с этой моделью можно, только если удается предвидеть заранее возможные перипетии хода проекта и тщательно собирать и интегрировать информацию на первых этапах, с тем, чтобы впоследствии можно было пользоваться их результатами без оглядки на возможные изменения.

Итеративная модель (Спиральная модель) разработки ПО

Итеративные или инкрементальные модели (известно несколько таких моделей) предполагают разбиение создаваемой системы на набор кусков, которые разрабатываются с помощью нескольких последовательных проходов всех работ или их части.***Итеративные***или ***инкрементальныемодели*** (известно несколько таких моделей) предполагают разбиение создаваемой системы на набор кусков, которые разрабатываются с помощью нескольких последовательных проходов всех работ или их части.

На первой итерации разрабатывается кусок системы, не зависящий от других. При этом большая часть или даже полный цикл работ проходится на нем, затем оцениваются результаты и на следующей итерации либо первый кусок переделывается, либо разрабатывается следующий, который может зависеть от первого, либо как-то совмещается доработка первого куска с добавлением новых функций. В результате на каждой итерации можно анализировать промежуточные результаты работ и реакцию на них всех заинтересованных лиц, включая пользователей, и вносить корректирующие изменения на следующих итерациях. Каждая итерация может содержать полный набор видов деятельности от анализа требований, до ввода в эксплуатацию очередной части ПО.

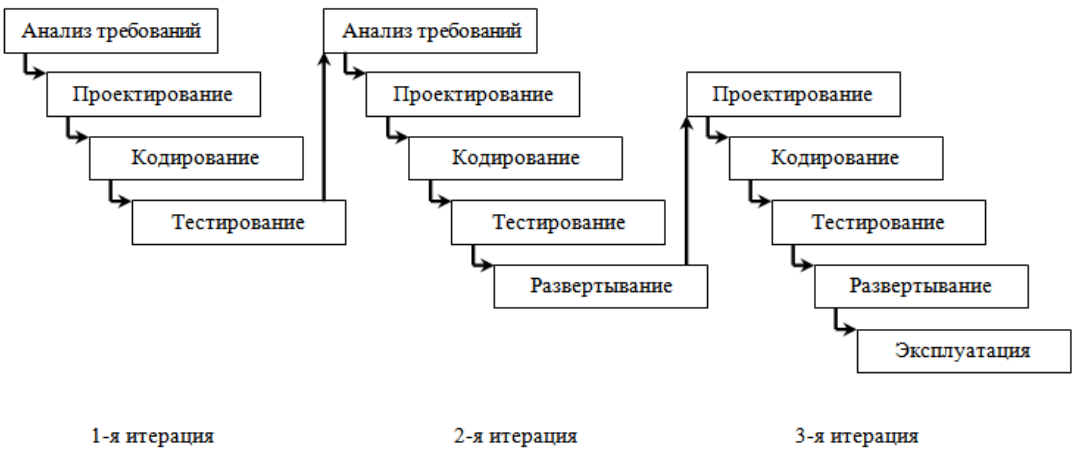


Рисунок. Возможный ход работ по итеративной модели.

Каскадная модель с возможностью возвращения на предшествующий шаг при необходимости пересмотреть его результаты, становится итеративной.

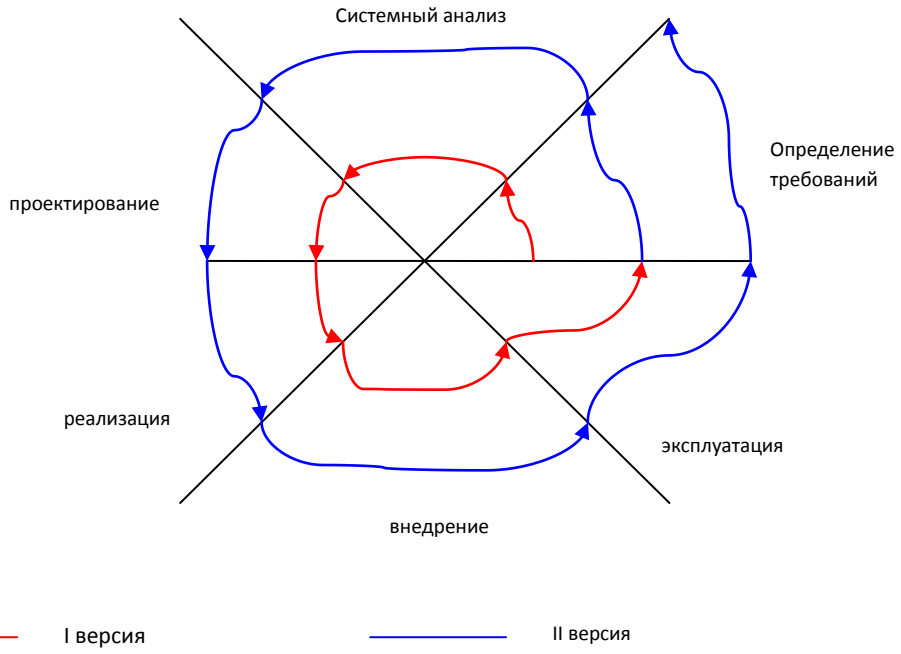
Вместе с гибкостью и возможностью быстро реагировать на изменения, итеративные модели привносят дополнительные сложности в управление проектом и отслеживание его хода. При использовании итеративного подхода значительно сложнее становится адекватно оценить текущее состояние проекта и спланировать долгосрочное развитие событий, а также предсказать сроки и ресурсы, необходимые для обеспечения определенного качества результата.

Спиральная модель жизненного цикла в отличие от каскадной более удачно отражает жизненный цикл автоматизированных информационных систем (АИС), например экономических, административных, т.е. не СРМВ, в которых все было заранее, известно.

При этом разработка ведется итерационно и «непрерывно» в течение всего жизненного цикла КП. Объясняется это следующим: как правило, потребитель (или заказчик) КП не всегда может сформулировать исчерпывающие формальные данные, описывающие предметную область.

При этом важным становится еще на ранних стадиях разработки предоставить заказчику «пилотный» вариант системы, который в дальнейшем на основании анализа заказчиком и разработчиком будет модернизироваться, чтобы как можно лучше удовлетворять требованиям заказчика.

Помимо неопределенности (некоторой) в определении предметной области для АИС характерно достаточно частое изменение законодательных, административных и других требований, которые требуют «практически непрерывной» доработки систем.



Пилотный вариант создается на первом витке спирали, после чего заказчиком и разработчиком проводится анализ необходимых доработок и происходит переход на следующий виток разработки.

Таких витков может быть несколько, и они могут продолжаться даже после внедрения системы в эксплуатацию. Наиболее важным этапом жизненного цикла в спиральной модели является этап эксплуатации, т.к. именно на этом этапе выявляются новые требования и неверно принятые технические решения.

При реальной разработке КП для АИС никогда не бывает приостановки разработки, что связано как с изменением нормативной базы, так и с пожеланиями заказчика.

Недостатком спиральной модели разработки является сложность определения сроков проекта и требуемых ресурсов.

Плюсом спиральной модели разработки является наиболее точное отражение жизненного цикла и процесса разработки АИС.

Примеры итеративных процессов разработки

Существует тяжеловесный и легковесный подход к разработке сложных КП, и как следствие, существуют тяжеловесные и легковесные процессы.

Тяжеловесные процессы – процессы с детально описанными действиями, предполагающими поддержку разработки кода ПО большим количествомвспомогательных действий. Основная цель описания – обеспечить успешное выполнение проекта на основе опыта других компаний и отделение успешной практики разработки и сопровождения ПО от конкретных людей, умеющих их применять. Выполняется иерархическое пошаговое описание действий, предпринимаемых в той или иной ситуации. В ходе проекта создается много промежуточных документов, позволяющих разработчикам последовательно разбивать стоящие перед ними задачи на более простые. Эти же документы служат для проверки правильности решений, принимаемых на каждом шаге, а также отслеживания общего хода работ и уточнения оценок ресурсов, необходимых для получения желаемых результатов. Кроме того, это позволяет обеспечивать коммуникацию между отдельными частями команды и взаимозаменяемость в команде.

Пример: RationalUnifiedProcess (RUP) — методология разработки программного обеспечения, созданная компанией RationalSoftware.

Легковесные процессы (agile процессы) – делают упор на использовании хороших разработчиков, а не хорошо отлаженных процессов разработки. Живые методы избегают фиксации четких схем действий, чтобы обеспечить большую гибкость в каждом конкретном проекте, а также выступают против разработки дополнительных документов, не вносящих непосредственного вклада в получение готовой работающей программы. Основные принципы «живой» разработки ПО зафиксированы в манифесте «живой» разработки, появившемся в 2000 году. Основные принципы:

1. Люди, участвующие в проекте, и их общение более важны, чем процессы и инструменты;
2. Работающая программа более важна, чем исчерпывающая документация;
3. Сотрудничество с заказчиком более важно, чем обсуждение деталей контракта;
4. Отработка изменений более важна, чем следование планам.

«Живые» методы позволяют большую часть усилий разработчиков сосредоточить собственно на задачах разработки и удовлетворения реальных потребностей пользователей. Отсутствие кипы документов и необходимости поддерживать их в связном состоянии позволяет более быстро и качественно реагировать на изменения в требованиях и в окружении, в котором придется работать будущей программе.

Пример: Экстремальное программирование.

Документация на сложные КП

Документация на сложные КП и их компоненты предназначена для детального отображенияи специфики компонент, подсистем и КП в целом.

В процессе разработки, отладки и эксплуатации документация выполняет следующие функции:

1. Формализует методы решения поставленных задач;
2. Формализует используемую информацию каждым компонентом системы и КП в целом;
3. Способствует резкому сокращению количества ошибок из-за неопределенностей постановки задач, пропусков, описок, нечеткого понимания мелких деталей, облегчая тем самым отладку и внедрение (документация разработчика.);
4. Позволяет учитывать вносимые в программы изменения в процессе их отладки и доработки (наиболеенеисполняемый пункт, невыполнение которого, как правило, приводит к тяжелым последствиям.);
5. Обеспечивает возможность контроля на этапахразработкикак со стороны заказчика, так и исполнителя;
6. Обеспечивает возможность замены разработчиком части персонала без полного повторения выполненной работы;
7. Позволяет изучать и квалифицированно эксплуатировать сложные КП с учетом всех возможностей, особенностей и специфики (эксплуатационная документация);
8. Обеспечивает возможность модификации и модернизации системы;
9. Позволяет использовать разработанные ранее компоненты, алгоритмы и прочее в различных частях одной системы или даже в разных системах (при профессиональной разработке в системе, как правило, не бывает больше 20% нового кода).

Документация должна разрабатываться с самого начала проектирования системы и непрерывно корректироваться.

В настоящее время в РФ при разработке сложных КП используется ряд стандартов, как международных, так и отечественных.

К международным стандартам в первую очередь надо отнести ISO (InternationalStandardOrganization). Эти стандарты касаются качества разработки ПС.

Отечественные стандарты условно подразделяются на 3 категории:

1. ГОСТ (Государственный Стандарт). Они должны выполняться на законодательном уровне;
2. ОСТ (Отраслевые Стандарты) – отражают специфику конкретной области (например, специфику разработок в оборонной промышленности). Разработчик несет ответственность за выполнение этого стандарта, а в крупных компаниях этим занимаются целые отделения;
3. СП (Стандарты Предприятий или стандарты группы разработчиков) - крайне желательно при выполнении лабораторных работ отработать СП.

Как ОСТ так и СП могут лишь расширять ГОСТы и не в коем случае им не противоречить.

ГОСТ 19 – единая система программной документации.

ГОСТ 34 – проектирование АСУ: документация, ПО, аппаратная часть.

Определение:ЕСПД (единая система программной документации) - это комплекс государственных стандартов, устанавливающих взаимосвязанные правила разработки, оформление и обращение программ и программной документации.

Правила и положения ЕСПД распространяются на программы и программную документацию для ВМ, комплексов и систем, независимо от их назначения и области применения.

ГОСТ 19.001 – 77

19 – класс стандарта (ЕСПД);

0 – классификационная группа (

0 – общие положения;

1 – основополагающие стандарты,

2 – правила выполнения документации разработки;

3 – правила выполнения документации исполнения;

4 – правила выполнения документации сопровождения;

5 – правила выполнения эксплуатационной документации;

6 – правила обращения программной документации;

7, 8 – резервные группы;

9 – прочие стандарты.)

01 – номер стандарта в группе;

77 – год первого принятия.

Стандарты могут непрерывно корректироваться, но год первого принятия не меняется.

Виды программ и программных документов

ГОСТ 19.107-77

Компонент – программа, рассматриваемая как единое целое и применяемая самостоятельно или в составе комплекса.

Комплекс – программа, состоящая из 2х или более компонент или комплексов, выполняющих взаимосвязанные функции и применяемая самостоятельно или в составе комплекса.

Виды программных документов	**Содержание программных документов**
Спецификация	Состав программы и документация на нее
Ведомость держателей подлинников	Перечень предприятий, на которых хранится подлинники программных документов
Текст программы	Запись программы с необходимыми комментариями
Описание программы	Сведения о логической структуре и функционировании программы

Программа и методики испытаний	Требования, подлежащие проверке при испытании программы, порядок и методы их контроля
Техническое задание	Назначение и область применения программы, технические, технико-экономические и специальные требования, предъявляемые к программе, необходимые стадии и сроки разработки, виды испытаний
Пояснительная записка	Схема алгоритма, общее описание алгоритма и(или) функционирование программы, обоснование принятых технических и технико-экономических решений
Эксплуатационная документация	Сведения для обеспечения функционирования и эксплуатации программы (ведомость эксплутационных документов, формуляр, описание применения, руководство системного программиста, руководство оператора)

Различные виды программных документов разрабатываются на различных стадиях разработки. Для подавляющего большинства документов необходимость их разработки определяется в техническом задании.

Всегда нужны для КП – спецификация, для программы – её код (т.е. текст.)

Допускается объединять отдельные виды документов. В объединенном документе должны содержаться все сведения, требуемые стандартами объединенных документов.

Пояснительная записка

ГОСТ 19.404-79

Отражает понимание заказчиком и разработчиком предметной области, задач, подлежащих решению и выбор методов их решения.

ПЗ должна содержать следующие разделы:

- введение (наименование программы, документы, на основании которых ведется разработка);
- назначение и область применения;
- технические характеристики (выбор алгоритмов, обоснование выбора);
- ожидаемые технико-экономические показатели;
- источники, использованные для разработки.

В зависимости от особенностей документа отдельные разделы допускается объединять, а также вводить новые разделы (подразделы).

По сути дела, пояснительная записка является документом, написанным «нормальным» языком, полностью понятным как исполнителю, так заказчику.

Техническое задание

ГОСТ 19.106-78

Основной документ, определяющий ход разработки. Является абсолютно формальным, при этом он определяет количественные и качественные показатели программного изделия. ТЗ должно быть написано таким образом, чтобы любой квалифицированный специалист смог написать программу, удовлетворяющую всем требованиям этого ТЗ.

Основой ТЗ является конкретика и полная ясность, не допускающая каких-либо разночтений. Это формализованный документ на основании, которого ведется разработка.

В документе приводится формализованное описание всех переменных, структур данных, формализованных алгоритмов и т.д.

ТЗ должно содержать следующие разделы:

- -Введение (см. описание);
- -Основание для разработки (на основании чего ведется разработка);
- -Назначение разработки, т.е. формально, что должна делать программа;
- - Требования к программе и программным изделиям (формально - с переменными, структурами, конкретными действиями, порядком разработки). Является основным разделом ТЗ, как правило, весьма трудоемкий;
- -Требование к программной документации и ее состав;
- -Технико-экономические показатели (качественные и количественные характеристики программы);
- -Стадии и этапы разработки;
- -Порядок контроля и приемки (кто и как принимает и на основании каких документов). Указывается довольно кратко - программа и методика испытаний, руководство оператора, руководство системного программиста;
- -Приложение, если оно требуется;

Как правило, ТЗ в процессе разработки программы претерпевает изменения и доработки. Для внесения изменений или дополнений в ТЗ используются те же самые правила, которые используют для самого ТЗ.

Практика показывает, что нежелательно выпускать дополнение к ТЗ, а лучше перерабатывать его заново.

Стадии разработки

ГОСТ 19.102-77

1. Предварительное проектирование и эскизный проект (системный анализ). Выпускается пояснительная записка.
2. Техническое проектирование (выпускается ТЗ.)

3. Программирование и автономная отладка (не на уровне компилятора).
4. Комплексная отладка.
5. Испытание, сдача заказчику и внедрение.

Стадии разработки, приведенные в ГОСТ, вообще говоря, отличаются от тех стадий, которые мы рассматривали при анализе жизненного цикла программы.

Стадии разработки	**Этапы работ**	**Содержание работ**
1. Техническое задание	Обоснование необходимости разработки программы	Постановка задачи Сбор исходных материалов Выбор и обоснование критериев эффективности и качества разрабатываемой программы. Обоснование необходимости проведения научно-исследовательских работ.
	Научно-исследовательские работы	Определение структуры входных и выходных данных. Предварительный выбор методов решения задач. Обоснование целесообразности применения ранее разработанных программ. Определение требований к техническим средствам. Обоснование принципиальной возможности решения поставленной задачи
	Разработка и утверждение технического задания	Определение требований к программе. Разработка технико-экономического обоснования разработки программы. Определение стадий, этапов и сроков разработки программы и документации на неё. Выбор языков программирования. Определение необходимости проведения научно-исследовательских работ на последующих стадиях. Согласование и утверждение технического задания.

Стадии разработки	Этапы работ	Содержание работ
2. Эскизный проект	Разработка эскизного проекта	Предварительная разработка структуры входных и выходных данных. Уточнение методов решения задачи. Разработка общего описания алгоритма решения задачи. Разработка технико-экономического обоснования.
	Утверждение эскизного проекта	Разработка пояснительной записки. Согласование и утверждение эскизного проекта.
3. Технический проект	Разработка технического проекта	Уточнение структуры входных и выходных данных. Разработка алгоритма решения задачи. Определение формы представления входных и выходных данных. Определение семантики и синтаксиса языка. Разработка структуры программы. Окончательное определение конфигурации технических средств.
	Утверждение технического проекта	Разработка плана мероприятий по разработке и внедрению программ. Разработка пояснительной записки. Согласование и утверждение технического проекта.
4. Рабочий проект	Разработка программы	Программирование и отладка программы.
	Разработка программной документации	Разработка программных документов в соответствии с требованиямиГОСТ 19.101-77.
	Испытания программы	Разработка, согласование и утверждение порядка и методики испытаний. Проведение предварительных государственных, межведомственных, приёмо-сдаточных и других видов

Стадии разработки	Этапы работ	Содержание работ
		испытаний. Корректировка программы и программной документации по результатам испытаний.
5. Внедрение	Подготовка и передача программы.	Подготовка и передача программы и программной документации для сопровождения и (или) изготовления. Оформление и утверждение акта о передаче программы на сопровождение и (или) изготовление. Передача программы в фонд алгоритмов и программ.

Общие требования к программным документам

ГОСТ 19.105-78

Программная документация может быть представлена на различных типах носителей (бумажный, магнитный, оптический и т.д.) На практике часть документов всегда выпускается в печатном виде, а код программы на практике никогда не печатается (в этом нет смысла).

Программный документ состоит из следующих условных частей:

1. Титульной (титульный лист и лист с подтверждениями (с подписью и печатями)). В настоящее время титульный лист и лист подтверждения объединяют;
2. Информационной (аннотация и содержание, если требуется);
3. Аннотация – краткое описание документа, от нескольких строкдо полстраницы, позволяющее определить, нужно ли знакомиться с данным документом или нет;
4. Основной раздел (состав и структура определяются соответствующим стандартом на конкретный документ);
5. Изменения (указывает какие составляющие документа были изменены). При этом может меняться версия документа.

Спецификация

ГОСТ 19.202-78

Информационную часть допускается не включать.

Спецификация в общем случае должна включать следующие разделы:

- Документацию;
- Комплексы;
- Компоненты.

Обозначение документов	Наименование	Примечание
Документы, комплексы, компоненты	Полное наименование и программы	Дополнительные сведения относящиеся к конкретному документу или программе

Пример примечания:

Текст программы представлен на магнитном носителе;

Подлинник находиться на таком-то предприятии.

Вид:

1. Обозначение;
2. Идентификатор проекта;
3. Идентификатор частей проекта;
4. Номер документа или программы.

Для отдельных компонент спецификаций не выпускается.

Например: Ц.51.804.002 Отладка Руководство системного программиста.

В настоящее время, в особенности в АИС (банки, бухгалтерия и т.д.) спецификация, как правило, не выполняется, а наличие документов и программ определяется в ТЗ.

Для систем оборонной промышленности спецификация является обязательной.

Формуляр

ГОСТ 19.501-78

1. Формуляр – юридический документ, определяющий ответственность разработчика (пользователя, заказчика). Документ значительной мере формирует правила работы с программами и программными документами.
2. Информационную часть допускается не включать.
3. В основную часть входят следующие сведения:
   - Общие указания (как пример: перед началом эксплуатации необходимо ознакомиться со следующими документами);
   - Общие сведения (наименование изделия, предприятие – изготовитель, предприятие проводящее эксплуатацию и т.д.);
   - Основные характеристики (за что разработчик несет ответственность и что вправе требовать заказчик и пользователь);

- Комплектность (состоит из подсистем, в которые входят…);
- Периодичность контроля приэксплуатации (характерно для систем оборонного назначения);
- Свидетельство о приеме (с подписями ответственных лиц и печатями);
- Свидетельство об упаковке и маркировки (характерно для систем оборонного назначения);
- Гарантийное обязательство (для программ это вообще чисто ерунда);
- Сведения о рекламациях (кому жаловаться);
- Сведения о хранении (в настоящее время не актуально);
- Сведение о закреплении программного изделия при эксплуатации (кто у заказчика или экплуатационщика отвечает за эксплуатацию (кто или какое подразделение));
- Сведения об изменениях (когда и кем проводились);
- Особые отметки (можно указывать то что заказчик и экплуатационщик считают важным);
- Приложения.

Формуляр составляется обычно для КП оборонного назначения и СРМВ, т.к. заказчик и экплуатационщик с одной стороны, и разработчик с другой стороны, могут быть мало связаны между собой в процессе эксплуатации.

Текст программы

ГОСТ 19.401-78

1. Информационная часть обязательна.

2. Текст программы может представлять собой один из трех видов символьной записи:

    1) символическая запись на исходном языке;
    2) символическая запись на промежуточном языке или коде;
    3) символическая запись в машинных кодах.

3. В настоящее время практически единственным видом записи программы является запись на исходном языке.

Описание программы

ГОСТ 19.402-78

1. Информационная часть обязательна.
2. Описание должно содержать:
    - Общие сведения:

обозначение и наименование программы;
программное обеспечение необходимое для функционирования программы;
языки программирования, используемые при разработке;

- Функциональное назначение:

функции, которые выполняет программа,
функциональные ограничения (если они существуют);

- Описание логических структур, т.е. алгоритм;
- Используемые технического средства;
- Вызов и загрузка

способ вызова программ из соответствующего места хранения (неактуально);
входные точки программы (неактуально);
адреса загрузки, сведения об используемой ОП;

- Входные данные (подробное формальное описание);
- Выходные данные (подробное формальное описание);

- Компиляцияи компоновкаописание процесса, указание всех необходимых библиотек и процедур (кроме стандартных библиотек, используемых системой).

Программа и методика испытаний

ГОСТ 19.301-79

- Разделы документа:
  - Объект испытаний;
  - Цель испытаний;
  - Требования к программе (что должна делать и в каких условиях);
  - Проверки в программе (по всем требованиям);
  - Требования к программной документации (указывается те документы, которые необходимы при испытании);
  - Состав и порядок проведения испытаний;
  - Методы проведения испытаний (указываются конкретные методы, особенно обращается внимание на внештатные ситуации (неправильный ввод, не тот файл)).

Руководство системного программиста

ГОСТ 19.503-79

1. Руководство системного программиста предназначено для предоставления информации по установке программы и ее сопровождению либо как самостоятельного компонента, либо как части комплекса.

2. РСП содержит следующие разделы:

   1) Общие сведения о программе;

   2) Структура программы – если это необходимо (дается менее подробно чем в описании, и предназначена для ознакомления с программой при ее установке и сопровождении);
   3) Настройка программы если она требуется;
   4) Проверка программы если требуется (дается в меньшем объеме, чем в программе и методике испытаний);
   5) Дополнительные возможности, если они имеются;
   6) Сообщение системному программисту (важный раздел).

Руководство оператора

ГОСТ 19.505-79

1. Является эксплуатационным документом. Оно составляется для операторов, которые могут и не быть, и, как правило, не бывают специалистами по вычислительной технике, поэтому РО должен быть написан языком, доступным специалистам конкретной предметной области.

2. РО должно содержать следующие разделы:

   1) Назначение программы;
   2) Условия её выполнения (на языке, доступном оператору, без использования специальных терминов);
   3) Вызов и выполнение программы (подробно, со всеми вариантами);
   4) Сообщение оператору (должны быть исчерпывающим, предусматривающими как штатные так и внештатные ситуации).

Типовая структура программы и программного модуля

1. По сути дела это требования к тексту программы.

2. Текст программы начинается с развернутого комментария, который состоит из:

    1) наименование и назначение программы;
    2) автор программы;
    3) время написания;
    4) особенности программы (если имеются).

    a. описание входных переменных на языке программирования с обязательными комментариями
    b. описание выходных переменных на языке программирования с обязательными комментариями
    c. описание внутренних переменных на языке программирования с обязательными комментариями
    d. тело программы, которое представляет из себя текст программы, определяющий производимые действия.

В теле программы должны быть комментарии, но не для каждого оператора, а лишь для тех, которые определяют логику работы программы(условные переходы, циклы, блоки обработки информации).

Понятие резидентных и кросс–систем автоматизации

1. Разработка программных средств осуществляется на технологических ВС (ТВС), эксплуатация на исполнительных ВС (ИВС).

2. Как правило, при любых вариантах разработки, мощность ТВС превосходит, и в ряде случаев весьма существенно, мощность ИВС.

3. В том случае, когда ИВС совпадает с ТВС по системе команд мы имеем так называемые резидентные системы автоматизации. Т.е. ТВС сразу создает машинный код, который исполним на ИВС.

Преимущества резидентных систем:

1. простота разработки ПС;
2. возможность использования мощных ОС, БД, средств автоматизации, отладки и т.д.

Недостатки:

Часто ИВС оказываются слишком мощными, что может сказываться на стоимости системы в целом.

При разработке ПС на основе кросс-систем автоматизации (КСА), ТВС не совпадает с ИВС по машинному коду, а также, как правило, по набору команд, что характерно главным образом для СРМВ и управляющих систем.

При этом, в большинстве случаев, нецелесообразна разработка мощных служебных программ, таких как ОС и.д., для ИВС, т.к. она является либо слишком дорогой, либо вообще невозможна из-за особенностей системы команд ИВС.

Преимуществакросс-систем:

1. Очень часто не требуется большая мощность ИВС, при этом используются сравнительно недорогие аппаратные решения.
2. Разработка ведется на ТВС большой мощности, обладающих мощными средствами разработки.

Недостатки:

Для проведения разработок приходится создавать специальные технологические средства для ТВС. Это в первую очередь кросс-компилятор, который получая входной текст, как правило, на известных и мощных языках программирования, генерирует машинный код для ИВС, т.е. переводит текст программы с ЯВУ в машинный код ИВС.

Проблемы стандартизации при разработки ПС

1. Под стандартизацией будем понимать как стандартизацию средств разработки, так и стандартизацию методов разработки.

Цели стандартизации:

1. Повышение производительности труда разработчиков;
2. Улучшение качества ПС.

Это достигается за счет единого подхода для всех членов команды разработчиков как по средствам, так и по методам разработки.

Примеры:

Создание и принятие стандартов, часто в мировом масштабе, на языки программирования, на системы управления базами данных (СУБД).

2. Принятие стандартов на документацию, на технологический цикл разработки, на организацию коллектива разработчиков, а также на последовательность проведения работ.

Принципы разработки больших КП

1. Планирование должно базироваться насостоянии системы в целом и на состоянии ее основных частей, на прогнозирование изменения характеристик при взаимодействии с заказчиком, на оценках развития процессов обнаружения и устранения ошибок программы.

Достигается это путем изучения и обобщения опыта разработки различных как малых, так и больших КП.

Каждый долгосрочный план проектирования должен содержать:

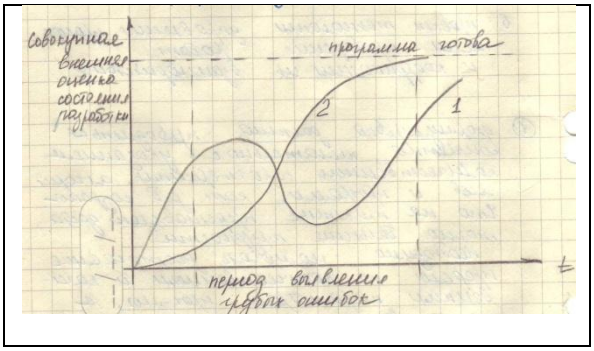
1) Формулирование общих целей разработки КП и частных целей разработки его подсистем;
2) Стратегию проектирования(в частности эксперименты по отладке);
3) Потребности в ресурсах различных видов;
4) Создание структуры коллектива разработчиков;
5) Создание технологии управления процессами проведения работ, и координации их взаимодействия.

2. Формулировка общих целей должна проводиться конкретно, с указанием количественных характеристик КП и их алгоритмов, которые должны быть созданы.На практике этот этап представляется исключительно трудным из-за того что, как правило, до начала разработки, с одной стороны, трудно определить точные объемы работ, а с другой стороны часто отсутствует достаточное понимание предметной области, что в основном связано с трудностями взаимодействия с заказчиком.

3. Необходимо намечать достижимые пределы по количественным и качественным параметрам КП, а также возможные сроки их достижения.

4. Стратегия проектирования должна обеспечивать выполнение поставленных задач с минимальными затратами. При этом она определяет:

    - Принципы и этапы проведения проектирования, опираясь при этом на понятие жизненного цикла.
    - Последовательность и длительность разработки алгоритмов и программ, учитывая то, что практически ни один проект не завершается в предполагаемые заранее сроки.



Случай 1:

1) 5% - функциональная проработка (исследование предметной области);

2) 45% кодирование или написание текста программы (сравнительно большая величина, т.к. приходится исправлять достаточно большое количество грубых ошибок, допущенных при анализе предметной области);

3) 50% отладка.

Случай 2:

1) 35% функциональная проработка (исследование предметной области);

2) 10% кодирование или написание текста программы;

3) 35% отладка.

5. Для больших и очень больших КП необходимо использовать все доступные средства автоматизации, которые, при этом могут приводить к возникновению дополнительных вспомогательных работ, например к разработке программ автоматической генерации тестов.

    <u>В реальных ситуациях второй подход выигрывает очень существенно.</u>

Принципы организации коллективов разработчиков КП

1. Даже большая группа квалифицированных разработчиков, не организованная структурно в коллектив, в котором определены обязанности и ответственности каждого, не выполнит разработку КП на десятки и сотни тысяч операторов.

2. Условная оценка производительности труда:

    1) 1970 – 0.1 оператор/человек в день;

2) 1980 – 1;

3) 1990 – 5;

4) Настоящее время – 10.

3. Это суммарная оценка после завершения проекта с учетом возможных инструментальных разработок, проведения системного анализа, разработки структуры КП.

4. Для разработки больших и очень больших КП требуется привлечение достаточно большого числа разработчиков, различающихся как по специализации, так и по квалификации.

5. Опыт показывает, что один и тот же коллектив должен отвечать за конкретные виды разработок, начиная от постановки задач, непосредственного проектирования и разработки, отладки и внедрения, а также обучение пользователей и тп. Иначе невозможно определить ответственность за те или иные этапы работы, за наличие ошибок и их исправление.

6. Из этого непосредственно следует необходимость организации коллектива разработчиков. Ранее это называлось бригадой программистов.

7. Принято рассматривать 2 основных варианта построение бригады: «хирургическая бригада» и «производственная бригада»:

8. Организация «хирургической бригады» строится по следующему принципу:

   1) Главный хирург. Во главе бригады стоит программист исключительно высокой квалификации (главный хирург), который определяет структуру КП, пишет основной код а также готовит основные принципы, вкладывающиеся в документацию;

   2) Заместитель руководителя – решает те же задачи что и руководитель, действуя под его руководством и готовый в любое время его заменить. Основная задача – обсуждение, критика, советы, возможная связь с другими бригадами. Личной ответственности, при наличии руководителя, ни за какую часть программы не несет, но готов. Имеет возможность и квалификацию, чтобы в любой момент времени заместить главного программиста;

   3) Администратор (технический руководитель) – его задачами является согласование и руководство всеми административными вопросами, как внутри, так и вне бригады, связь с заказчиком;

   4) Редактор – его задачами являются критика документации руководителя а также непосредственная работа над документацией, возможно с какими-либо помощниками;

   5) Инструментальщик – опытный системный программист, который разрабатывает и поддерживает системные и инструментальные ПС (тесты, библиотеки, инструментальные средства разработки и т.п.);

   6) Наладчик – человек, готовящий тесты, имитаторы, генераторы тестов, определяет ход тестирования, регистрирует процесс отладки (возможно с помощником.);

   7) Дополнительно имеется 2-3 технических работника: секретарь, библиотекарь, уборщица;

   8) При этом программу, по сути дела, создает очень ограниченное количество людей вплоть до 2-3 человек;

   9) Использование такого подхода организации коллектива разработчиков характерно для создания стабильных состоявшихся КП, когда не требуется постоянное изучение предметной области.

9. Структуру «производственной бригады» можно представить следующим образом:

1) Руководитель, как правило администратор - бывший опытный программист (хотя и не обязательно), основными задачами которого являютсяразработка ТЗ, коррекция частных ТЗ, определение тестов и порядков отладки.

2) Заместитель руководителя - решает те же задачи, что и руководитель, но при этом несет меньшую личную ответственность. В любое время может заменить руководителя.

3) «Главный инженер» - опытныйдействующий программист и организатор. Задачи – создание структуры всего проекта, организация разработки, возможно решение кадровых вопросов.

4) Группа системного анализа во главе с руководителем, которая проводит изучение предметной области. Состав группы может «динамически» меняться в зависимости от изменения предметной области.

5) Группа программистов функциональных программ (также во главе с руководителем) - тесно связана с группой системного анализа. У руководства группы и ее подразделений стоят опытные программисты возможно каждый со своей специализацией. Важно отметить, что в составе этой группы находится, и на практике это бывает всегда, специалисты с разной квалификацией.

6) Группа программистов – инструментальщиков. Задачи этой группы совпадают с инструментальщиками хирургической бригады. Часто эта группа состоит из 1 человека.

7) Группа отладчиков во главе с руководителем. Основными задачами ее являются знание предметной области, структуры и особенностей КП в целом и его подсистем, выделение основных задач, которые в основном (и к сожалению), и проверяются. Крайне желательно чтобы члены группы отладчиков не писали никаких кодов, хотя на практике это не всегда достижимо.

8) Группа технических писателей, которые создают всю программную документацию. Это люди, хорошо разбирающиеся в работе КП, в целях, решаемых комплексом и его реализации и должны быть хорошо ознакомлены с принятыми стандартами. При этом группа может быть весьма ограничена в количестве.

9) Технический персонал, который включает в себя специалистов по аппаратной части, администраторов, секретарей и т.п.

10) Подобная организация коллектива характерна для разработки больших и очень больших КП, в особенности при периодическом изменении предметной области.

Тестирование и отладка ПО

1. Тестирование – выполнение программы с целью обнаружения факта наличия в программе ошибок.

2. Отладка– определения местоположения ошибок и внесение исправлений в программу.

Принципы тестирования.

1. ошибки в программе есть;
2. тест – это совокупность исходных данных и ожидаемых результатов;
3. тестовые данные должны быть достаточно просты для проверки;
4. тесты готовятся заранее, до выхода на машину;
5. тесты разрабатываются после задания на разработку программы, до написания программного кода (первые тесты);

6. перед началом тестирования должны быть сформулированы цели, которые должны быть достигнуты в результате тестирования;
7. в процессе тестирования необходимо фиксировать выполненные тесты и реально полученные результаты;
8. тесты должны быть одинаково тщательными как для правильных, так и для неправильных входных данных;
9. необходимо проверить два момента: программа делает то, что должна и не делает того, чего не должна;
10. результаты теста необходимо изучать досконально и объяснять полностью;
11. недопустимо ради упрощения тестирования изменять программу;
12. после исправления программы требуется повторное тестирование;
13. ошибки кучкуются.

Для уменьшения количества тестов необходимо классифицировать все возможные варианты выполнения программы - разбить их на классы, эквивалентные с точки зрения проверки правильности программы. Если критерии выбраны и классификация проведена, то достаточно взять по одному тесту из каждого класса. Если все они окажутся неудачными, то тестирование можно заканчивать. Критерии, по которым проводится классификация, называются критериями полноты тестирования (КПТ). После выбора КПТ каждый тест анализируется: удачен ли он (удалось ли с его помощью обнаружить ошибки), достаточен ли набор тестов с точки зрения проверяемого критерия и нужны ли еще какие-либо тесты. Существует два подхода к формированию КПТ:

14. критерии «Черного ящика» - описывают тестирования с точки зрения поставленной задачи, без учета внутреннего устройства программы;
15. критерии «Белого ящика» - описывают тестирование с точки зрения поставленной задачи с учетом внутреннего устройства программы.

Критерии тестирования

Критерии «Черного ящика»:

1. тестирование функций (требуется подобрать такой набор тестов, чтобы был выполнен хотя бы один тест для каждой из функций);
2. тестирование классов входных данных (разделить их на классы таким образом, чтобы все данные из одного класса были равнозначны с точки зрения проверки правильности программы);
3. тестирование классов выходных данных (аналогично предыдущему);

Три этих критерия хорошо согласуются друг с другом.

4. тестирование области допустимых значений;
   если ОДЗ представляет собой простое перечисление, то проверяется правильность обработки каждого из этих значений и невозможность ввести вместо них иные значения;
   если ОДЗ – числовой диапазон, то можно выделить три случая:
   - нормальное условие в середине класса;
   - граничные экстремальные условия;
   - исключительные условия выхода за границы класса.
5. тестирование длинны набора данных (проверяется допустимое количество элементов в наборе);
   - пустой набор;
   - единичный набор;
   - если предусмотрена минимальная допустимая длина, то проверяется слишком короткий набор;
   - набор минимально возможной длины;
   - нормальный набор;
   - набор из нескольких частей;

- набор максимально возможной длины;
- слишком длинный набор;

6. тестирование упорядоченности (сортировка и поиск);
   - данные неупорядочены;
   - упорядочены в прямом порядке;
   - упорядочены в обратном порядке;
   - существуют повторяющиеся значения;
   - экстремальные значения находятся в середине набора;
   - экстремальные значения находятся в начале набора;
   - экстремальные значения находятся в конце набора;
   - существует несколько совпадающих экстремальных значений.

Критерии «Белого ящика»:

1. Критерий покрытия операторов (необходимо подобрать такой набор тестов, чтобы каждый оператор был выполнен хотя бы один раз);
2. критерий покрытия ветвей (подбирается такой набор тестов, чтобы каждая ветвь программы была выполнена хотя бы один раз);
3. критерий покрытия путей (подбирается такой набор тестов, чтобы каждый путь в программе был выполнен хотя бы один раз);

Ветвь – условие if, путь – дополнительный цикл.

4. критерий покрытия условий (подбирается такой набор тестов, чтобы каждое простое условие в сложном получило и значение true, и значение false хотя бы один раз);
5. критерий покрытия решения условий (подбирается такой набор тестов, чтобы каждая ветвь в программе была бы пройдена хотя бы один раз и каждое простое условие получило бы значение true и значение false хотя бы один раз);
6. критерий комбинаторного покрытия условий (подбирается такой набор тестов, чтобы хотя бы один раз выполнялась любая комбинация простых условий).

Существует такое понятие, как минимальное грубое тестирование (МГТ).

МГТ – критерий покрытия решений и условий, усиленный дополнительными требованиями для проверки циклов. Проверка циклов организуется по правилам:

1. для каждого цикла с предусловием должна быть проверена правильность при нулькратном, однократном и многократном повторении тела цикла;
2. для каждого цикла с постусловием должна быть проверена правильность при однократном и многократном повторении тела цикла;
3. проверка цикла со счетчиком зависит от того, фиксированы ли границы изменения счетчика, или вычисляются. Проверка при нулькратном, однократном и многократном повторении тела цикла.

Виды тестирования

1. Модульное тестирование (тестируется отдельный модуль в отрыве от основной системы);
2. Интеграционное тестирование (две и более компоненты тестируются на совместимость);
3. Регрессионное тестирование (тестирование системы в процессе ее разработки и сопровождении на не регресс-изменение системы не ухудшили уже существующей функциональности.Создаются пакеты регрессионных тестов, которые запускаются с определенной периодичностью, например в пакетном режиме, связанном с процедурой постоянно интеграции);

4. Нагрузочное тестирование (тестирование системы на корректную работу с большими объемами данных.Проверка баз данных на обработку большого объема записей);
5. Стрессовое тестирование (тестирование на устойчивость к непредвиденным ситуациям);
6. Приемочное тестирование (тестирование при приемки заказчиком);

Эталоны при проектировании КП

1. Сущность отладки состоит в сравнении результатов работы КП с некоторыми данными, являющимися верными для отлаживаемой программы с очень высокой вероятностью. Такие данные называются эталонами. Отличие результатов работы программы от этих эталонов свидетельствует о некорректности работы программы и необходимости проведения изменений, устраняющих указанные расхождения.

   Источником эталона является ТЗ, поэтому очень важна его тщательная разработка.

   В качестве эталонов при проектировании КП используются:

   - Формализованные правила записи и структуры построения программ (в соответствии требованиям компилятора и других технологических средств разработки, а также стандартов, включая стандарты предприятия);
   - Детерминированные результат, используемые при отладке выполняемой программы;
   - Статистические характеристики функционирования КП или его частей.

Основные способы получения эталонов

1. Расчет вручную или на ВС по известным алгоритмам.
2. Разработка упрощенных и обобщенных математических или алгоритмических моделей алгоритмов для проверяемых программ.
3. Использование результатов функционирования ранее разработанных реальных систем, если это возможно.
4. Разработка правдоподобных гипотез и постановка умозрительных экспериментов (характерно для СРМВ.
5. Эталоны подразделяются на вычислительные и логические.
6. Вычислительные эталоны получить и использовать проще, опираясь на гипотезу гладкости функции, вычисляя лишь отдельные значения при формульном описании.
7. Логические эталоны получить труднее из-за их комбинаторного характера и из-за возникновения при этом «проклятия» размерности.
8. Одно из основных требований к эталону – достоверность. Для увеличения достоверности приходится учитывать точность вычислений, адекватность моделей, используемых в качестве эталонов реальных систем, степень дискретизации процесса и т.п. Однако абсолютная достоверность недостижима, более того, полная проверка программ практически невозможна (для больших КП). Поэтому после завершения отладки можно лишь с некоторой вероятностью считать, что программа функционирует верно, то есть в программах могут принципиально иметься ошибки, и чаще всего они имеются.

Типичные ошибки при проектировании КП

1. С точки зрения потребителя информации ошибкой выходных данных является отклонение величины выдаваемых параметров от их эталонных значений, превышающих заданный допуск, что справедливо главным образом для СРМВ. Для АИС ошибкой является неверная выдача документов или другой информации, что также является отклонением от эталонных значений.

2. Ошибки в КП по разному влияют на результат: от незначительного уменьшения точности до полного прекращения функционирования КП.

3. Анализ реальных разработок позволяет выявить 4 этапа обнаружения ошибок:

1. автономная отладка отдельных программ и замкнутых подсистем, то есть подсистем имеющих небольшое количество внешних связей.
2. проверка связи между программами и замкнутыми подсистемами, или стыковка.
3. комплексная отладка в статике каждой из систем и КП в целом. Для СРМВ не учитываются временные характеристики, для АИС это и так неактуально.
4. Комплексная отладка в динамике подсистем и КП в целом.

4. Практика показывает, что количество ошибок имеет наибольший коэффициент корреляции с количеством передач управления, а не с общим количеством операторов в программе, то есть по сути количество ошибок связано со сложностью программы.

5. Количество ошибок относительно передач управления порядка ~0,12. Автономная отладка выявляет, как правила, ~70% всех ошибок. Проверка связей ~15%. Комплексная отладка в статике ~10%. Комплексная отладка в динамике ~2%.

6. В зависимости от этапов и видов работ ошибки можно разделить на следующие типы:

    1) Технологические ошибки ввода программ в память ВС а также возможные ошибки документации, на основании которой программа вводится в память ВС.
    2) Программные ошибки из-за неверной записи операторов (это не синтаксические ошибки).
    3) Алгоритмические ошибки, связанные с неполным формированием алгоритма решения и некорректной постановки задач.
    4) Системные ошибки, обусловлены отклонениями в принятых в КП алгоритмов от реальных функционирующих алгоритмов и отличие реальных характеристик управляемых объектов от предполагаемых при проектировании.

Технологические ошибки

1. В настоящее время технологические ошибки в реальных системах составляют ничтожную величину и могут быть вызваны лишь сбоями при перенесении исполняемого кода с каких-либо носителей в память ВС. Однако они могут существовать и в ряде систем, в основном СРМВ, где вводятся дополнительный контроль при занесении программ в память. Наиболее простым и в тоже время практически исчерпывающим является использование контрольных сумм.

2. Для АИС технологические ошибки также не являются определяющими, как правило они естественным образом достаточно быстро выявляются и для их исправления достаточно лишь перезагрузить программу.

Программные ошибки

Эти ошибки в значительной степени определяются степенью автоматизации при разработке программ. Разработка программ на ЯВУ позволяет снизить количество ошибок в 10-20 раз по сравнению с программами, разработанными на Assembler. Опыт показывает, что у опытных программистов обычно получается 1-2% неверных операторов (не синтаксических ошибок). 1 программная ошибка приводит, как правило, к изменению нескольких операторов, но самое печальное, что исправление ошибок очень часто приводит к возникновению новых ошибок.

Алгоритмические ошибки

Этот тип ошибок очень трудно поддается обнаружению методами формализованного контроля, как правило, применяемым при отладке, что связано с трудностью получения полных и достоверных эталонов для сложных КП.

Системные ошибки

Определяется, как правило, на последних этапах отладки, что часто приводит к тяжелейшим и даже катастрофическим последствиям. Еще одна трудность при определении подобных ошибок состоит в

сложности получения некоторых режимов реальных объектов (например, сближение самолетов), поэтому при отладке часто используют модели.

Динамика изменения количества сложныхошибок КП при отладке

1. Исследования в динамике изменения ошибок позволяет получить важные данные для оценки уровня отлаженности КП, а также для определения необходимых ресурсов при проектировании КП в заданные сроки.

2. Длительность отладки зависит от допустимого значения не выявленных ошибок, при котором разработку можно считать законченной. Показателем отлаженности может быть вероятность обнаружения ошибок в программе в течение некоторого времени или интенсивность потока искажения результатов в период эксплуатации за счет не выявленных ошибок. Подобный подход представляется весьма удобным и разумным, но на практике получение количественных оценок весьма затруднено.

3. Существует жестокая корреляция между 3 видами проявления ошибок:

    1) Суммарным количеством ошибок в КП имеющихся к определенному времени.
    2) Количеством ошибок в КП выявляемых в единицу времени при постоянных усилиях (затратах).
    3) Количеством искажений результатов на выходе КП из-за не выявленных ошибок.

4. Под постоянными усилиями будем понимать постоянство затрат ресурсов, как человеческих, так и материальных и временных. Человеческие ресурсы определяются количеством людей, занятых в отладке, и их квалификацией. Материальные ресурсы определяются количеством аппаратных и программных средств, используемых при отладке.

5. Если отладка ведется при постоянных усилиях, то можно предположить, что вероятность обнаружения ошибок в первом приближении определяется количеством ошибок в КП. Количество обнаруженных ошибок в единицу времени пропорционально количеству имеющихся ошибок в программе. Считая, что каждая обнаруженная ошибка устраняется, количество ошибок КП уменьшается и, следовательно, интенсивность их обнаружения должна экспоненциально убывать в зависимости от длительности отладки τ:

$$\lambda=\lambda_o*\exp(-\alpha*\tau), \quad (1)$$

где λ_o – интенсивность появления ошибок в начале отладки, λ – интенсивность обнаружения и устранения ошибок в единицу времени.α – некая константа, определяемая постоянными усилиями и соответственными затратами ресурсов.

6. В качестве показателя отлаженности КП можно было бы взять предельную интенсивность выявления ошибок, значение которой, в «теории», можно было бы получить из оценки допустимости потерь от воздействия не выявленных ошибок в процессе эксплуатации. Однако определение таких показателей на практике нереально. Поэтому в реальной жизни время завершения отладки осуществляется по совместному решению заказчика и исполнителя.

7. Выражение (1) абсолютно правильно отражает характер изменения ошибок во времени при сделанных нами предположениях, но не отражает реальное количество ошибок в системе. Это объясняется тем, что различные составляющие КП используются с различной частотой. Примерно с такой же частотой они тестируются, что является вполне разумным.

8. Причиной этого явления является воздействие входных типовых ситуаций, вызывающих выполнение сравнительно ограниченного круга функциональных программ, которые наиболее тщательно тестируются. В других частях программ, которые функционируют (а соответственно и тестируются) реже, может сохраниться большее количество ошибок. Физически мы не можем проверить все сочетания входных переменных. .При этом в режиме эксплуатации получаем

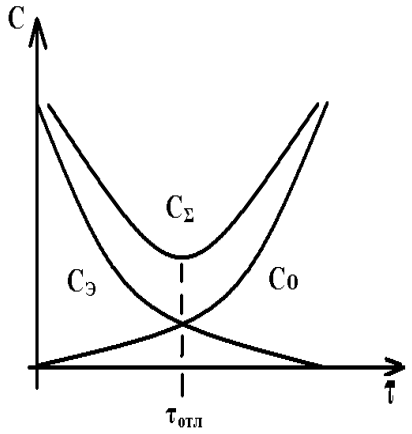
$$\lambda э = k*\lambda о*exp(-\lambda*\tau), \quad (2)$$

т.е. реальных ошибок в kраз больше чем мы можем предположить.

Исследования, проведенные в условиях практической отладки, показывают, что k~100.

Выражения (1) и (2) вообще правильно показывает изменение количества ошибок и интенсивностей выявления ошибок в процессе отладки. Однако их использование на практике практически невозможно.

Определение рациональной длительности отладки



C∑ = C0 + CЭ – развитие отладки.

C0 – затраты на отладку.

CЭ – затраты на эксплуатацию, вызванные воздействиями необнаруженных ранее ошибок.

9. На основе этого графика нельзя делать количественных заключений о длительности отладки. В значительной мере выбор момента окончания отладки определяется опытом руководителя разработки, ограниченностью материальных ресурсов и реальными требованиями заказчика к времени окончания разработки.

Основные принципы отладки

10. Основной целью отладки является установка факта работоспособности КП в соответствии с требованиями, в первую очередь ТЗ, а также других программных документов. В данном случае важна непосредственная связь с эталонами.

11. При различных методах проверки программ должны решаться следующие задачи:

    1) определение наличия ошибок и отклонения от эталонов;
    2) диагностика и локализация ошибок;
    3) информированием оператора, проводящего отладку, о ходе и результатах обработки и соответственно о наличии ошибок.

12. Наиболее сложными задачами являются 1 и 2. Формальных методов их решения не существует, и следовательно, многое зависит от опыта лиц, проводящих отладку, и в особенности от их знания предметной области. Крайне желательно, чтобы отладку, в особенности в ее заключительные этапы, проводили лица не имеющие отношение к разработке конкретной программы.

13. По сути дела процесс отладки состоит в построении последовательности правдоподобных гипотез об источниках возникновения ошибок и их проверке.

Методы отладки

1. Основным методом отладки является последовательное и постепенное усложнение задачи и увеличение объема совместно функционирующих при отладке программ. Постепенность наращивания объема совместно наращиваемых программ существенно снижает эффект маскирования ошибок и позволяет их локализацию.

2. Процесс отладки по уровню сложности, степени связи с объектом управления (СРМВ) и объемом отлаживаемых программ можно условно разделить на следующие этапы:

    1) автономная отладка комплекса замкнутых подсистем сравнительно небольшого объема в статике (то есть без учета временных характеристик).
    2) комплексная отладка КП в целом в статике
    3) комплексная отладка КП в целом в динамике с учетом временных характеристик. Это является наиболее важным и трудным этапом для СРМВ.

С точки зрения метода по типу эталонов, применяемых при выявлении ошибок, используются следующие 3 группы методов:

1) Структурный контроль соответствия программ формализованным правилам построения. Проверка соответствия правилам написания программ в соответствии со структурами, определенными в ТЗ, а также, если это необходимо, в соответствии со стандартами предприятия или группы разработчиков.
2) Отладка по частным деталям реализации тестов.На практике она производится по детерминированным эталонам, которые приходится подбирать “вручную”.
3) Отладка по статистическим характеристикам реализации тестов. Для преодоления «проклятья размерности» осуществляется переход к статистическим методам отладки, которые можно подразделить на 2 части:

   a. статистическая отладка (производится анализ статистических характеристик входных переменных, на их основании генерируется тестовое воздействие, после чего производится статистический анализ результатов работы программы на соответствие некоторому распределению. На практике это вызывает существенные затраты как временные, так и материальные.

   b. проверка динамических характеристик функционирования объекта на основании статистических данных.Этот этап всегда проводится после исчерпывающего завершения первого этапа, чтобы максимально избежать маскирования “тонких” ошибок.

Принципы отладки

1. отладка идей: осуществляется на стадии предварительного эскизного проекта, алгоритм в это время существует лишь в виде «блок-схем». Имеется также формализованное целевое описание. При этом

обеспечивается проверка идеологических основ построения алгоритмов КП, при том достаточно часто с использованием математического моделирования. При этом устраняются принципиальные системы и алгоритмические ошибки.

2. ручная автономная отладка без применения ВС: позволяет выявить наиболее массовые и грубые ошибки, которые сложно выявляются формальными методами, так как эти методы рассчитаны на «хорошую» программу со сравнительно небольшим количеством ошибок. Суть этого метода заключается в проверке логики программы, переменных, структур. Ручная отладка повышает свою эффективность, если отладчик не тот человек, который писал текст программы.
3. автономная отладка программ и замкнутых подсистем с использованием детерминированного тестирования. По сути, это проверка на основе детерминированных эталонов.
4. статистическая автономная отладка, после которой можно переходить к комплексной отладке.
5. Комплексная отладка.

Принципы комплексной отладки и испытаний КП

1. Основная задача комплексной отладки – завершение отладки всего КП и доведение его характеристик до требований ТЗ. При этом требуется провести:

    1) сопряжение по передаче информации и управления для всех составляющих КП.
    2) обеспечение всех характеристики КП в соответствии с ТЗ.
    3) проверка состава и полноты документации.

2. Необходимо, чтобы КП полностью удовлетворял ТЗ не только в «средних», наиболее часто используемых условиях функционирования, но и в граничных, особенно в нештатных ситуациях, в особенности с учетом возможного воздействия невыявленных ошибок.

3. Главный методологический принцип – постепенное подключение в отладку подсистем КП и проведение при этом их непрерывного тестирования. Порядок подключения подсистем не всегда можно формализовать и решение о порядке подключения, как правило, принимается неформально специалистами, прекрасно представляющими предметную область, структуру КП и функциональную задачу.

Комплексная отладка

1. Комплексная отладка (КО) состоит из трёх частей (приближенно):

    1) Статическая КО подсистем и КП в целом, проверка и корректировка сопряжения по информации и управлению подсистем, проверка на основе детерминированных тестов, при этом, по сравнению с автономной отладкой, значительно увеличивается объем.
    2) Динамическая КО всей системы без использования реальных объектов управления, т.е. на моделях. Это касается главным образом СРМВ. Внешние объекты и сигналы взаимодействия с ними представляются моделями и имитаторами, которые реализуются программными или аппаратно-программными методами.
    Последовательность подключения подсистем при отладке определяется их структурными связями в КП, и последовательность подключения выбирается опытными разработчиками и согласуются с заказчиком.
    Для АИС КО также нельзя проводить сразу на объекте с выдачей информации для принятия решений. Но в этом случае специальные модели и имитаторы не создаются, а контроль результатов на основе применения эталонов производится путем анализа либо предшествующей системы, либо «вручную» на основании опытных специалистов данной предметной области.
    Динамическая КО на реальных объектах, если она возможна. Невозможность проверки отдельных режимов на реальных объектах может быть связана с их опасностью, нецелесообразностью и дороговизной создания ситуации.

    3) Последний этап (№3) предназначен для проверки результатов, полученных

a. на 1ом и 2ом этапе КО, так как всегда существует вероятность того, что в модели и имитаторы, используемые на 2ом этапе отладки, вкрались ошибки. Это системные ошибки о которых ранее упоминалось, при неудачных обстоятельствах отладка на объекте может привести к тяжелым последствиям и даже к полному провалу проекта.