**Содержание**

Аннотация.

Введение.

1. Жизненный цикл ПО

Введение.

Шаги процесса программирования по Райли

Введение.

1.1.1. Постановка задачи.

1.1.2. Проектирование решения.

1.1.3. Кодирование алгоритма.

1.1.4. Сопровождение программы.

1.1.5. Программная документация.

Вывод к п. 1.1

1.2. Определение ЖЦПО по Леману.

Введение.

1.2.1 Определение системы.

1.2.2. Реализация.

1.2.3. Обслуживание.

Вывод к п. 1.2.

1.3. Фазы и работы ЖЦПО по Боэму

1.3.1. Каскадная модель.

1.3.2. Экономическое обоснование каскадной модели.

1.3.3. Усовершенствование каскадной модели.

1.3.4. Определение фаз жизненного цикла.

1.3.5. Основные работы над проектом.

Вывод.

Литература.

**Введение**

Промышленное применение компьютеров и растущий спрос на программы поставили актуальные задачи существенного повышения *производительности разработки ПО*, разработки индустриальных методов планирования и проектирования программ, переноса организационно-технических, технико-экономических и социально-психологических приемов, закономерностей и методов из сферы материального производства в сферу применения компьютеров. *Комплексный подход* к процессам разработки, эксплуатации и сопровождения ПО выдвинул ряд насущных проблем, решение которых исключит «узкие места» в проектировании программ, уменьшит сроки завершения работ, улучшит выбор и адаптацию существующих программ, а может быть и определит судьбу систем со встроенными ЭВМ.

В практике разработок больших программных проектов зачастую отсутствует *единый подход* к оцениванию затрат труда, сроков проведения работ и материальных затрат, что сдерживает повышение производительности разработки ПО, а в конечном счете – эффективное управление жизненным циклом ПО. Поскольку программа любого типа становится изделием (кроме, может быть, учебных, макетных программ), подход к ее изготовлению во многом должен быть аналогичен подходу к производству промышленной продукции, и вопросы проектирования программ становятся чрезвычайно важными. Эта идея лежит в основе книги Б.У. Боэма «Инженерное проектирование программного обеспечения», которую мы использовали при написании данной курсовой работы. В этой книге под проектированием ПО понимается процесс создания проекта программного изделия.

**1 Жизненный цикл ПО**

**ВВЕДЕНИЕ**

ЖЦПО – это непрерывный процесс, который начинается с момента принятия решения о необходимости создания ПО и заканчивается в момент его полного изъятия из эксплуатации.

Существует несколько подходов при определении фаз и работ жизненного цикла программного обеспечения (ЖЦПО), шагов процесса программирования, каскадная и спиральная модели. Но все они содержат общие основополагающие компоненты: постановка задачи, проектирование решения, реализация, обслуживание.

Наиболее известной и полной, пожалуй, является структура ЖЦПО по Боэму, включающая восемь фаз. Она и будет представлена в дальнейшем наиболее подробно.

Одним из возможных вариантов может послужить описание верхнего уровня по Леману, включающее три основные фазы и представляющее описание ЖЦПО в самом общем случае.

И, для разнообразия, – приведем шаги процесса программирования, представленные Д.Райли в книге «Использование языка Модула-2». Это представление, по-моему, является весьма простым и привычным, с него и начнём.

**1.1 Шаги процесса программирования по Райли**

**Введение**

Процесс программирования включает четыре шага (рис. 1):

постановка задачи, т.е. получение адекватного представления о том, какую задачу должна выполнить программа;

проектирование решения уже поставленной задачи (в общем, такое решение является менее формальным, чем окончательная программа);

кодирование программы, т. е. перевод спроектированного решения в программу, которая может быть выполнена на машине;

сопровождение программы, т.е. непрекращающийся процесс устранения в программе неполадок и добавления новых возможностей.

**Постановка задачи**

**Проектирование решения**

**Кодирование алгоритма**

#### Сопровождение программы

#### Документ

**Документ**

**Документ**

Рис. 1.Четыре шага программирования.

Программирование начинается с того момента, когда *пользователь*, т.е. тот, кто нуждается в программе для решения задачи, излагает проблему *системному аналитику.* Пользователь и системный аналитик совместно определяют постановку задачи. Последняя затем передается *алгоритмисту*, который отвечает за проектирование решения. Решение (или алгоритм) представляет последовательность операций, выполнение которых приводит к решению задачи. Поскольку алгоритм часто не приспособлен к выполнению на машине, его следует перевести в машинную программу. Эта операция выполняется кодировщиком. За последующие изменения в программе несет ответственность сопровождающийпрограммист. И системный аналитик, и алгоритмист, и кодировщик, и сопровождающий программист – все они являются программистами.

В случае большого программного проекта число пользователей, системных аналитиков и алгоритмистов может оказаться значительным. Кроме того, может возникнуть необходимость вернуться к предшествующим шагам в силу непредвиденных обстоятельств. Все это служит дополнительным аргументом в пользу тщательного проектирования программного обеспечения: результаты каждого шага должны быть полными, точными и понятными.

**1.1.1 Постановка задачи**

Одним из наиболее важных шагов программирования является постановка задачи. Она выполняет функции контракта между пользователем и программистом (программистами). Как и юридически плохо составленный контракт, плохая постановка задачи бесполезна. При хорошей постановке задачи как пользователь, так и программист ясно и недвусмысленно представляют задачу, которую необходимо выполнить, т.е. в этом случае учитываются интересы как пользователя, так и программиста. Пользователь может планировать использование еще несозданного программного обеспечения, опираясь на знание того, что оно может. Хорошая постановка задачи служит основой для формирования ее решения.

*Постановка задачи* (*спецификация программы*); по существу, означает точное, полное и понятное описание того, что происходит при выполнении конкретной программы. Пользователь обычно смотрит на компьютер, как на черный ящик: для него неважно, как работает компьютер, а важно, что может компьютер из того, что интересует пользователя. При этом основное внимание фокусируется на взаимодействии человека с машиной.

Характеристики Хорошей Постановки Задачи:

*Точность*, т.е. исключение любой неоднозначности. Не должно возникать вопросов относительно того, каким будет вывод программы при каждом конкретном вводе.

*Полнота*, т.е. рассмотрение всех вариантов для заданного ввода, включая ошибочный или непредусмотренный ввод, и определение соответствующего вывода.

*Ясность*, т.е. она должна быть понятной и пользователю и системному аналитику, поскольку постановка задачи – это единственный контракт между ними.

Часто требование точности, полноты и ясности находятся в противоречии. Так, многие юридические документы трудно понять, потому что они написаны на формальном языке, который позволяет предельно точно сформулировать те или иные положения, исключая любые самые незначительные разночтения. Например, некоторые вопросы в экзаменационных билетах иногда сформулированы настолько точно, что студент тратит больше времени на то, чтобы понять вопрос, чем на то чтобы на него ответить. Более того, студент вообще может не уловить основной смысл вопроса из-за большого количества деталей. Наилучшая постановка задачи та, при которой достигается баланс всех трех требований.

Стандартная форма постановки задачи.

Рассмотрим следующую постановку задачи: «Ввести три числа и вывести числа в порядке».

Такая постановка не удовлетворяет приведенным выше требованиям: она не является ни точной, ни полной, ни понятной. Действительно, должны ли числа вводиться по одному на строке или все числа на одной строке? Означает ли выражение «в порядке» упорядочение от большего к меньшему, от меньшего к большему или тот же порядок, в каком они были введены.

Очевидно, что подобная постановка не отвечает на множество вопросов. Если же учесть ответы на все вопросы, то постановка задачи станет многословной и трудной для восприятия. Поэтому Д. Райли предлагает для постановки задачи пользоваться стандартной формой, которая обеспечивает максимальную точность, полноту, ясность и включает:

наименование задачи (схематическое определение);

общее описание (краткое изложение задачи);

ввод;

вывод;

ошибки (явно перечислены необычные варианты ввода, чтобы показать пользователям и программистам те действия, которые предпримет машина в подобных ситуациях);

пример (хороший пример может передать сущность задачи, а также проиллюстрировать различные случаи).

Пример. Постановка задачи в стандартной форме.

НАЗВАНИЕ

Сортировка трех целых чисел.

ОПИСАНИЕ

Ввод и вывод трех целых чисел, отсортированных от меньшего числа к большему.

ВВОД

Вводятся три целых числа по одному числу на строке. При этом целым числом является одна или несколько последовательных десятичных цифр, которым может предшествовать знак плюс «+» или знак минус «–».

ВЫВОД

Выводятся три введенных целых числа, причем все три выводятся на одной строке. Смежные числа разделяются пробелом. Числа выводятся от меньшего к большему, слева направо.

ОШИБКИ

1) Если введено менее трех чисел, программа ждет дополнительного ввода.

2) Строки ввода, кроме первых трех, игнорируются.

3) Если какая-либо из первых трех строк содержит более одного целого числа, то программа завершает работу и выдает сообщение.

ОШИБКА ВВОДА – допускается только одно целое число на строке.

ПРИМЕР:

ввод → – 3

2

+ 17

вывод → – 3 2 + 17

**1.1.2. Проектирование решения**

Этот шаг программирования является наиболее трудным. На данной стадии постановка задачи должна быть превращена в алгоритм. Поэтому алгоритмист должен обладать достаточным опытом программирования и подходить к каждой новой задаче, опираясь на твердо установленную методику проектирования. К сожалению, в настоящее время все большие программы содержат ошибки, что приводит к скверным проектам. Плохие проекты в свою очередь являются следствием сложности задач и использования неадекватных методов проектирования. Чтобы избежать ошибок в программах, алгоритмисты должны использовать тщательно разработанные процедуры конструирования, основанные на правилах логического вывода.

Существуют две основные модели вывода:

Первая модель известна как *дедуктивный вывод.* Эту форму логического мышления обессмертил знаменитый сыщик Шерлок Холмс. Дедуктивная логика применяет общие правила к конкретным случаям. Например, Холмс мог вывести конкретное утверждение «Дворецкий является убийцей» из более общих сведений «Убийца-блондин», а «Дворецкий является единственным блондином, которого можно подозревать».

Вторая модель – это *индуктивный вывод,* который является противоположностью дедуктивному выводу. Индуктивная логика извлекает общие заключения из отдельных случаев. Например, индуктивный вывод может быть использован, чтобы обосновать общее заключение «солнце поднимается на востоке» на основании многих отдельных наблюдений того, что солнце всегда поднималось на востоке.

Эти два процесса вывода умозаключений – *дедукция* (от общего к частному) и *индукция* (от частного к общему) – тесно связаны с двумя наиболее широко распространенными методами проектирования: *«сверху вниз»* и *«снизу вверх».* Подобно дедукции, проектирование «сверху вниз» начинается с большой задачи, которая разбивается на подзадачи. Например, проектировщик нового холодильника-морозильника на основании исходного множества спецификаций агрегата (т.е. постановки задачи) должен дать детальные схемы и спецификации окончательного продукта (т.е. проект).

Если при этом он использует метод проектирования «сверху вниз», то единственная задача проектирования может быть разбита на две меньшие подзадачи: (1) проектирование холодильного агрегата и (2) проектирование морозильника.

Однако можно воспользоваться методом проектирования «снизу вверх» и начать с проектирования холодильного компрессора, а затем охлаждающих труб, агрегата или холодильной камеры. В таком случае этот выбор будет налагать определенные ограничения на весь проект.

Задача проектировщика – создание алгоритма, выполняющего функции связующего звена между постановкой задачи и готовой для исполнения программой. Проверку созданного алгоритма, т. е. насколько последний отражает постановку задачи, осуществляет системный аналитик. В силу этого и системный аналитик, и проектировщик должны уметь читать и понимать алгоритм. Каждый алгоритм записывается на некотором псевдоязыке*.* Алгоритмы, называемые также псевдокодами, не могут быть выполнены ни на каком компьютере.

**1.1.3 Кодирование алгоритма**

Работа кодировщика заключается в переводе алгоритма в программу. Для создания полной, точной и понятной программы необходимы соответствующие методы записи программ. Например, кулинарные рецепты обычно записываются на естественных языках, таких, как английский, французский, русский или японский. Программы же пишутся на языках программирования. В настоящее время ни один из естественных языков нельзя использовать в качестве языка программирования, так как они чересчур сложны, чтобы их могли «понимать» машины. В отличие от естественных, языки программирования созданы специально для такого представления решения задачи, которое может быть выполнено компьютером.

**1.1.4 Сопровождение программы**

Прежде чем завершить работу, кодировщик должен убедиться, что программа соответствует псевдокоду. Затем системный аналитик, алгоритмист и, что самое главное, пользователь должны протестировать и подтвердить, что она работает правильно. После этого можно считать, что программа готова для передачи пользователю в комплекте со всей необходимой документацией.

Однако на этом программирование не заканчивается; далее следует *шаг сопровождения.* Дело в том, что в программе могут быть ошибки, обусловленные либо неадекватной постановкой задачи, либо тем, что проект не удовлетворяет постановке задачи или программа не соответствует проекту. Какова бы ни была причина, пользователь вправе потребовать корректировки программы, поскольку он не представлял, что программа будет работать таким образом. Исправление ошибок является одной из главных задач сопровождения программ. Другой не менее важной задачей сопровождения программ является ее модификация, т. е. добавление в программу новых возможностей или изменение существующих. Пользователь может изменить требования к работе программы, что, в свою очередь, приведет к необходимости ее переписать. Сложность операций по сопровождению программы зависит от типа изменений, которые должны быть сделаны: в худшем случае может потребоваться полная переработка программы от постановки до кодирования. Обычно на сопровождение программы затрачивается большее время, чем на ее создание.

**1.1.5 Программная документация**

Последней составляющей процесса программирования является *документирование.* Оно включает широкий спектр описаний, облегчающих процесс программирования и обогащающих результирующую программу. Постоянное документирование должно составлять неотъемлемую часть каждого шага программирования. Постановка задачи, проектные документы, алгоритмы и программы – все это документы. Внутренняя документация, включенная непосредственно в программу, облегчает чтение кода. Назначение учебного пособия (еще одной формы документации) – научить пользователя применять новую программу; справочное руководство позволяет ознакомиться с описанием команд программного обеспечения.

**Вывод к п.1.1.**

Согласно модели, представленной в данной главе, программирование можно разделить на четыре шага: постановку задачи, проектирование решений, кодирование программы, сопровождение программы. Дополнительно модель включает документирование программы как действия, которые необходимо выполнять в течение всего процесса программирования.

Модель программирования построена специально для решения больших проблем, так как именно они представляют интерес для специалистов в области информатики. Тем не менее, на практике важно использовать тщательно выбранные инженерные методики проектирования программ независимо от размера задач: навыки, приобретенные в процессе решения более мелких задач, могут быть закреплены и успешно реализованы при решении больших задач.

Следует помнить, что хорошее программирование – это не кодирование быстро найденного решения с помощью любой подходящей методики, а тщательно инструментированная инженерная процедура, позволяющая создать полное, точное и легко понимаемое (ясное) программное обеспечение.

**1.2 Определение ЖЦПО по Леману**

**Введение**

В самом общем случае можно считать, что жизненный цикл программной системы состоит из трех фаз:

определения;

реализации;

обслуживания.

**1.2.1 Определение системы**

Процесс создания ПО начинается с практических изысканий, ведущих к системному анализу, задача которого состоит в определении общих требований к системе и программам. Такой анализ должен, прежде всего, установить реальные потребности и цели и по возможности выявить имеющиеся методы, позволяющие достичь поставленной цели. При необходимости анализ может основываться на математических или иных формальных схемах. Считается, что при любом подходе указанный анализ должен иметь определенную структуру и проводиться в соответствии с некоторой теорией. Анализ и внесение уточнений, осуществляемые совместно с аналитиками и потенциальными пользователями, должны привести к выработке окончательной спецификации требований*.*

Процесс составления такой спецификации имеет целью получение правильного технического задания, полного в смысле отражения требований и согласованного с определением реализации. За составлением спецификаций следует фаза *проектирования,* смысл которой состоит в идентификации и структуризации данных, их преобразовании и организации их передачи. Кроме того, на этой фазе необходимо добиться в определенном смысле оптимального распределения функции системы, выбрать алгоритм и процедуры, а также обозначить системные компоненты и связь между ними.

**1.2.2 Реализация**

Завершив *проектирование* можно начинать *реализацию* системы. Однако на практике фаза проектирования и реализации перекрываются. Таким образом, по мере осуществления иерархического процесса разбиения анализ некоторых элементов системы может быть признан достаточно полным для перехода к реализации, в то время как другие элементы требуют дальнейшего уточнения.

В ходе процесса реализации необходимо устанавливать правильность программы. Современные процедуры большей частью основаны на тестировании, хотя в последние годы расширилось использование методов сквозного структурного контроля и аттестации программ.

В любом случае, тестирование посредством исполнения программы обычно осуществляется снизу вверх, в начале на блочном (модульном или процедурном уровне), затем функционально, компонент за компонентом. По мере проверки отдельных компонентов они объединяются в систему в процессе ее компоновки, после чего начинаются системные испытания. В конечном итоге, после того как независимо будет удостоверено качество функционирования системы и оценены ее параметры, она считается готовой к *выпуску.*

**1.2.3 Обслуживание**

Процесс обслуживания начинается сразу после выпуска системы. Ошибки подлежат выявлению и исправлению. Если нормальной работе пользователя препятствует ошибка, то ошибочную программу можно временно исключить из системы или же внести временные или постоянные исправления в некоторые или во все используемые системы. Постоянное исправление или изменение можно затем внести в новый выпуск системы. Для того чтобы учесть все изменения и их комбинации, создаются многочисленные версии системных элементов. Главной задачей становится управление системной конфигурацией. Решающая роль в управлении программирования принадлежит вспомогательным службам, которые автоматически собирают и регулируют все изменения в системе.

**Вывод к п.1.2.**

Метасистема, в рамках которой развивается программа, содержит существенно большее количество контуров обратной связи, чем указано выше. Многие виды деятельности перекрываются, сложным образом переплетаются и систематически повторяются. Поэтому достаточно обоснована модель ЖЦПО представленная Боэмом.

**1.3 Фазы и работы ЖЦПО по Боэму**

**1.3.1 Каскадная модель**

Каскадная модель была введена в 70 – 80 гг. Она удобна для однородных ПП, когда каждое приложение представляло собой единое целое.

Основные характеристики модели:

- Жизненный цикл разбивается на этапы (фазы);

- Переход с этапа на этап – только после полного завершения текущего этапа;

- Этап завершается выпуском полного комплекта документации, достаточной для того, чтобы работа могла быть выполнена другой командой разработчиков.

Главные характерные черты *каскадной модели* следующие:

завершение каждой фазы верификацией и подтверждением, цель которых – устранить возможно большее число проблем, связанных с разработкой изделия;

циклические повторения реализованных фаз с возможно более ранней фазы.

Анализ осуществимости системы

Подтверждение

Планирование и анализ требований к ПО

Подтверждение

Проектирование изделия

Верификация

Детальное проектирование

Верификация

Кодирование

Автономная отладка

Внедрение

Системная отладка

Комплексирование

Верификация изделия

Функционирование и сопровождение

Повторное подтверждение

Рис.2. Каскадная модель ЖЦПО.

В каскадной модели успешное окончание одной из фаз ЖЦПО означает достижение соответствующей цели инженерного программирования (см. п. 2.4.). К этим подцелям необходимо добавить еще две:

*Детальная проектируемость* – получение полных верифицированных спецификаций и структур управления и данных, интерфейсных связей, характеристик, основных алгоритмов и определение условий работы каждого программного компонента.

*Кодируемость* – получение полного, верифицированного набора компонентов программы.

Основные достоинства:

Формирование полного набора проектной документации в конце работы над этапом. Документация отвечает критериям полноты и завершенности;

Возможность планирования сроков и затрат. Для целого ряда ПП эта модель реализуема – это для систем, для которых на этапе анализа можно точно и полно сформировать все требования. Например, сложные вычислительные программы.

Основные недостатки:

- Большие сроки от анализа до завершения;

- Требования к ПО «заморожены» в виде ТЗ до конца разработки.

**1.3.2 Экономическое обоснование каскадной модели**

Не углубляясь в экономический анализ, которому Б.У. Боэм уделяет большое внимание в книге «Инженерное проектирование программного обеспечения», скажем лишь, что экономическое обоснование каскадной модели, ориентированной на последовательное достижение целей, базируется на двух главных предпосылках:

Для получения качественного программного изделия (т.е. такого, которое в полной мере удовлетворяет всем целям требуемого программного изделия) необходимо в любом случае осуществить все подцели на каждом этапе.

Любое другое упорядочение подцелей приводит к созданию менее качественного программного изделия.

**1.3.3 Усовершенствование каскадной модели**

Рассмотрим одно из усовершенствований идеальной каскадной модели – пошаговую разработку.

*Пошаговая разработка* является усовершенствованием метода повторной разработки с созданием прототипа и поуровневой разработкой сверху – вниз. Этот метод предполагает пошаговое увеличение функциональных возможностей ПО в процессе разработки.

Детальное проектирование

Верификация

Кодирование

Автономная отладка

Комплексирование

Верификация изделия

Внедрение

Системная отладка

Эксплуатация и сопровождение

Переподтверждение

Анализ осуществимости модели

Подтверждение

Планирование и анализ требований к ПО

Подтверждение

Проектирование изделия

Верификация

Детальное проектирование

Верификация

Кодирование

Автономная отладка

Комплексирование

**…**

Детальное проектирование

Верификация

Кодирование

**…**

Рис.3. Каскадная модель с использованием пошаговой разработки.

В качестве усовершенствованной каскадной модели пошаговая разработка успешно применялась при создании как очень больших, так и небольших программных изделий.

Главными преимуществами пошаговой разработки перед абсолютно повторной разработкой и поуровневой разработкой сверху – вниз являются следующие:

использование последовательных расширений программы обеспечивает гораздо менее дорогой способ учета в усовершенствованном изделии опыта пользователей, чем при повторной разработке;

расширение функциональных возможностей намного упрощает проверку и полезнее, чем промежуточные изделия при поуровневой разработке.

Значение пошаговой разработки заключается главным образом в изменении распределения затрат труда на проект. Вариант каскадной модели при пошаговой разработке показан на рисунке 3.

**1.3.4 Определение фаз жизненного цикла**

Ниже будут даны формулировки конечных целей каждой фазы для перехода к следующей фазе. Для пошаговой разработки приводимые формулировки относятся к границам фаз каждого шага расширения.

*Начать фазу планирования и анализа требований.* (Завершение концептуального обзора ЖЦПО.)

Получение одобренной и подтвержденной архитектуры системы с включением основных соглашений о распределении функций между аппаратурой и программами. Получение одобренного и подтвержденного общего представления об функционировании ПО с включением основных соглашений о распределении функций между человеком и системой.

Формирование общего плана ЖЦПО с определением основных этапов, ресурсов, обязанностей, сроков и главных работ.

*Завершить фазу планирования и анализа требований. Начать фазу проектирования изделия.* (Завершение обзора требований к ПО).

Формирование детального плана разработки: детальных показателей завершения этапов разработки, планов распределения ресурсов, схем организационной структуры, обязанностей, сроков, работ, методов и изделий.

Формирование детального плана использования: пунктов плана разработки, содержание которых ориентировано на обучение, перенос программ, внедрение, эксплуатацию и поддержание.

Формирование детального плана отладки изделия – план управления конфигурацией технического обеспечения, план контроля качества, общий план верификации и подтверждения.

Одобренная и подтвержденная спецификация требований к ПО: функциональные, технические и интерфейсные спецификации, для которых подтверждены их полнота, непротиворечивость, проверяемость и осуществимость.

Одобренный (формально или неформально) договор на разработку, основанный на приведенных выше пунктах.

*Закончить фазу проектирования изделия. Начать фазу детального проектирования.* (Завершение анализа результатов проектирования изделия.)

Разработка верифицированной спецификации проекта программного изделия:

формирование иерархии программных компонентов, межблочных интерфейсов по данным и управлению;

формирование физической и логической структур данных до уровня отдельных полей;

разработка плана распределения вычислительных ресурсов (времени, памяти, точности);

верификация полноты, непротиворечивости, осуществимости и обоснованности требований.

Установление и разрешение всех противоречий разработки, которые повышают степень риска.

Разработка предварительного этапа комплексирования и отладки, плана руководства для пользователей и приемных испытаний.

*Закончить фазу детального проектирования. Начать фазу кодирования и автономной отладки.* (Завершение сквозного контроля проекта или критического поблочного анализа проекта.)

Верифицированная детальная спецификация каждого блока:

спецификация каждой подпрограммы, имени, назначения, предположений, размеров, последовательности вызовов, ошибочных выходов, входных и выходных данных, алгоритмов и логической схемы;

описание базы данных до уровня отдельных параметров, символов и битов;

верификация полноты, непротиворечивости и соответствия требованиям проектных спецификаций системы и планам распределения ресурсов.

Одобренный план приемных испытаний.

Руководства пользователю, а также завершенный предварительный план комплексирования и отладки.

*Закончить фазу копирования и отладки. Начать фазу комплексирования и отладки.* (Удовлетворение критериев автономной отладки.)

Проверка работы всех блоков не только для номинальных, но также для исключительных и предельных значений.

Проверка всех вариантов ввода и вывода, включая сообщения об ошибках.

Выполнение всех операторов и всех ветвей передачи управления.

Проверка выполнения стандартов программирования.

Завершение поблочного документирования внутренней структуры.

*Закончить фазу комплексирования и испытаний. Начать фазу внедрения.* (Завершение анализа результатов приемных испытаний.)

Проверка удовлетворения тесту приемных испытаний программ:

проверка удовлетворения требованиям к ПО;

демонстрация приемлемости указанных в спецификациях характеристик работы в нештатных условиях.

Приёмка поставляемых программных изделий, отчетов, руководств, баз данных, спецификаций внутренней структуры.

*Закончить фазу внедрения. Начать фазу эксплуатации и сопровождения.* (Завершение анализа приемки системы.)

Проверка удовлетворительности результатов приемных испытаний системы.

Проверка удовлетворительности системных требований.

Проверка производственной готовности ПО, аппаратуры, средств обслуживания и персонала.

Приёмка поставляемых и входящих в систему изделий: аппаратуры, ПО, документации, средств обучения и обслуживания.

Завершение всех специфицированных работ и ввод системы в действие.

*Закончить фазу эксплуатации и сопровождения* (путем снятия с производства).

Выполнение всех пунктов плана снятия с производства: перенос программ, документирование, создание архива, переход к новой системе.

**1.3.5 Основные работы над проектом**

Анализ требований.

Проектирование изделия.

Программирование.

Планирование отладки.

Верификация и подтверждение.

Управление проектом.

Управление конфигурацией и контроль качества.

Документирование.

**Вывод**

Итак, были рассмотрены три подхода к определению жизненного цикла ПО. На мой взгляд, все они имеют право на существование, так как в той или иной степени отражают практику программирования. Тем более, что легко можно обнаружить общие моменты (ставится задача – определяется система – анализируются требования; сопровождение программы – обслуживание – эксплуатация и сопровождение).

Однако, надо заметить, что определение фаз и работ ЖЦПО Боэма наиболее обоснованно, т.к. опирается на более ориентированный подход в инженерном программировании (направленный на получение качественного программного изделия и реализацию эффективного процесса разработки и сопровождения ПО) и обосновывается экономически.

Исходя из данного отчета видно как важно и необходимо знать потребности современного мира при составлении программного продукта (изделия). Важно при составлении программы для автоматизации, какой либо системы, учитывать то, что современный мир постоянно меняется, а значит должна быть способной к изменению и программа.

Важно так же, при составлении программы, учитывать то, что программа должна быть точной; полной по своему содержанию и пригодной для работы как с маленькими, так и с большими проблемами в соответствии со своим предназначением; ясной - для того чтобы пользователь мог спокойно, без затруднений работать с ней. А так же чтобы программу в любой момент можно было бы легко исправить или дополнить в соответствии с изменившимися требованиями в современном мире.

Следует помнить, что хорошее программирование – это не кодирование быстро найденного решения с помощью любой подходящей методики, а тщательно инструментированная инженерная процедура, позволяющая создать полное, точное и легко понимаемое (ясное) программное обеспечение.

**Литература**

1. Б.У. Боэм «Инженерное проектирование программного обеспечения». М.: Радио и связь. 1985.

2. Д.Райли. «Использование языка Модула-2». М.: Мир. 1993.

3. Ю.В. Иванов «Программы и их жизненные циклы» (реферат по дисциплине «Метрология ПО»). 1998.