Автономная некоммерческая образовательная организация
высшего образования
«Воронежский экономико-правовой институт»
(АНОО ВО «ВЭПИ»)

АНОО ВО "ВОРОНЕЖСКИЙ ЭКОНОМИКО-ПРАВОВОЙ ИНСТИТУТ", АНОО ВО "ВЭПИ", ВЭПИ

Подписано цифровой подписью: АНОО ВО "ВОРОНЕЖСКИЙ ЭКОНОМИКО-ПРАВОВОЙ ИНСТИТУТ", АНОО ВО "ВЭПИ", ВЭПИ
Дата: 26.09.2023 10:16:18 +03'00'

УТВЕРЖДАЮ
Проректор
по учебно-методической работе
_____ А.Ю. Жильников
«__» _____ 20__ г.

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ ПО ДИСЦИПЛИНЕ (МОДУЛЮ)

Б1.О.14 Программная инженерия
(наименование дисциплины (модуля))

09.03.03 Прикладная информатика
(код и наименование направления подготовки)

Направленность (профиль) Прикладная информатика в экономике
(наименование направленности (профиля))

Квалификация выпускника Бакалавр
(наименование квалификации)

Форма обучения Очная, заочная
(очная, заочная)

Рекомендованы к использованию Филиалами АНОО ВО «ВЭПИ»

Воронеж 2018

Методические рекомендации по выполнению лабораторных работ по дисциплине (модулю) рассмотрены и одобрены на заседании кафедры прикладной информатики.

Протокол от «13» декабря 2018 г. № 5

Заведующий кафедрой Г.А. Курина

Разработчики:

Доцент А. И. Кустов

ЛАБОРАТОРНЫЕ РАБОТЫ

Лабораторная работа № 1
«Введение»

Цель работы: знать основные понятия программной инженерии

1. Краткие теоретические сведения

Проблема. На первых этапах становления программной инженерии (даже когда она так еще не называлась) было отмечено, что высокая стоимость программ связана с разработкой одинаковых (или похожих) фрагментов кода в различных программах. Вызвано это было тем, что в различных программах как части этих программ решались одинаковые (или похожие) задачи: решение нелинейных уравнений, расчет заработной платы. Использование при создании новых программ ранее написанных фрагментов сулило существенное снижение сроков и стоимости разработки.

Модульное программирование. Главный принцип модульного программирования состоял в выделении таких фрагментов и оформлении их в виде модулей. Каждый модуль снабжался описанием, в котором устанавливались правила его использования – интерфейс модуля. Интерфейс задавал связи модуля с основной программой – связи по данным и связи по управлению. При этом возможность повторного использования модулей определялась количеством и сложностью этих связей, или насколько эти связи удалось согласовывать с организацией данных и управления основной программы. Наиболее простыми в этом отношении оказались модули решения математических задач: решения уравнений, систем уравнений, задач оптимизации. К настоящему времени накоплены и успешно используются большие библиотеки таких модулей.

Для многих других типов модулей возможность их повторного использования оказалась проблематичной в виду сложности их связей с основной программой. Например, модуль расчета зарплаты, написанный для одной фирмы, может не подойти для другой, т.к. зарплата в этих фирмах рассчитывается не во всем одинаково. Повторное использование модулей со сложными интерфейсами является достаточно актуальной и по сей день. Для ее решения разрабатываются специальные формы (структуры) представления модулей и организации их интерфейсов.

Рост сложности программ (структурное программирование)

Проблема. Следующий этап возрастания стоимости ПО был связан с переходом от разработки относительно простых программ к разработке сложных программных комплексов. К числу таких сложных программ относятся: системы управления космическими объектами, управления оборонным комплексом, автоматизации крупного финансового учреждения и т.д. Сложность таких комплексов оценивалась следующими показателями:

1. Большой объем кода (миллионы строк).
2. Большое количество связей между элементами кода.

3. Большое количество разработчиков (сотни человек).

4. Большое количество пользователей (сотни и тысячи).

5. Длительное время использования.

Для таких сложных программ оказалось, что основная часть их стоимости приходится не на создание программ, а на их внедрение и эксплуатацию. По аналогии с промышленной технологией стали говорить о жизненном цикле программного продукта, как о последовательности определенных этапов: этапа проектирования, разработки, тестирования, внедрения и сопровождения.

Структурное программирование. Этап сопровождения программного комплекса включал действия по исправлению ошибок в работе программы и внесению изменений в соответствии с изменившимися требованиями пользователей. Основная причина высокой стоимости (а порой и невозможности выполнения) этапа сопровождения состояла в том, что программы были плохо спроектированы – документация была не понятна и не соответствовала программному коду, а сам программный код был очень сложен и запутан. Нужна технология, которая обеспечит «правильное» проектирование и кодирование. Основные принципы технологии структурного проектирования и кодирования:

1. Нисходящее функциональное проектирование, при котором в системе выделяются основные функциональные подсистемы, которые потом разбиваются на подсистемы и т.д. (принцип «разделяй и властвую»).

2. Применение специальных языков проектирования и средств автоматизации использования этих языков.

3. Дисциплина проектирования и разработки: планирование и документирование проекта, поддержка соответствие кода проектной документации.

4. Структурное кодирование без goto.

2. Порядок выполнения работы и содержание отчета

Порядок выполнения работы:

1. Основные понятия программной инженерии.

2. Программа, программное обеспечение, задачи и приложения. технологические и функциональные задачи.

3. Процесс создания программ: постановка задачи, алгоритмизация, программирование.

4. Понятие программного продукта.

5. Характеристика программного продукта и его специфика.

6. Показатели качества программного продукта: мобильность, надежность, эффективность, легкость применения, модифицируемость и коммуникативность.

Содержание отчета:

1) цель работы;

2) задание на лабораторную работу для своего варианта;

3) алгоритм решаемого задания с необходимыми пояснениями;

4) выводы по работе.

3. Контрольные вопросы

1. Основные понятия программной инженерии.
2. Показатели качества программного продукта.

Лабораторная работа № 2
«Модели и профили жизненного цикла программных средств»

Цель работы: знать модели и профили жизненного цикла программных средств.

1. Краткие теоретические сведения

При создании и сопровождении сложных, распределенных, тиражируемых ПС требуется гибкое формирование и применение гармонизированных совокупностей базовых стандартов и нормативных документов разного уровня, выделение в них требований и рекомендаций, необходимых для эффективной реализации конкретных функций систем. Для унификации и регламентирования реализации этих функций совокупности базовых стандартов должны адаптироваться и конкретизироваться в программной инженерии применительно к определенным классам проектов, их функций, процессов и компонентов. В связи с этим выделилось и сформировалось понятие «профиля стандартов», как основного инструмента функциональной стандартизации.

Профиль стандартов — это совокупность нескольких (или подмножество одного) базовых стандартов (и других нормативных документов) с четко определенными и гармонизированными подмножествами обязательных и факультативных возможностей, предназначенная для реализации заданной функции или группы функций. Функциональная характеристика (заданный набор функций) объекта стандартизации является исходной для формирования и применения профиля этого объекта или процесса. В профиле выделяются и устанавливаются допустимые факультативные возможности и значения параметров каждого базового стандарта и/или нормативного документа, входящего в профиль. Профиль не может противоречить использованным в нем базовым стандартам и нормативным документам.

Он должен использовать факультативные возможности и значения параметров в пределах допустимых, выбранные из альтернативных вариантов.

На базе одной и той же совокупности базовых стандартов могут формироваться и утверждаться различные профили для разных проектов и сфер применения. Эти ограничения базовых документов профиля и их гармонизация, проведенная разработчиками профиля, должны обеспечивать качество, совместимость и корректное взаимодействие компонентов системы, соответствующих профилю, в заданной области его применения.

Основными целями применения профилей стандартов при создании и применении ПС являются:

1) снижение трудоемкости, длительности, стоимости и улучшение других технико-экономических показателей проектов систем и комплексов программ;

2) повышение качества разрабатываемых или применяемых покупных компонентов и ПС в целом при их разработке, приобретении, эксплуатации и сопровождении;

3) обеспечение расширяемости ПС по набору прикладных функций и масштабируемости в зависимости от размерности решаемых задач;

4) поддержка функциональной интеграции в системах задач, ранее решавшихся раздельно;

5) обеспечение переносимости программ и данных между разными аппаратно-программными платформами.

Состояние и развитие стандартизации в области программной инженерии характеризуется следующими особенностями, которые необходимо учитывать при формировании и применении профилей:

несколько сотен разработанных международных и национальных стандартов не полностью и неравномерно покрывают потребности в стандартизации объектов и процессов создания и применения сложных систем, программных средств и их компонентов;

большая длительность разработки, согласования и утверждения международных и национальных стандартов (3—5 лет) приводит к их консерватизму, а также к хроническому отставанию требований и рекомендаций этих документов от современного состояния техники и от текущих потребностей практики и технологии создания сложных систем;

стандарты современных ПС должны: учитывать необходимость их построения как открытых систем; обеспечивать расширяемость при наращивании или изменении выполняемых функций; переносимость программных средств и данных систем между разными аппаратно-программными платформами;

2. Порядок выполнения работы и содержание отчета

Порядок выполнения работы:

1. Понятие жизненного цикла программы и его этапы.
2. Анализ требований к программе.
3. Определение спецификации программы.
4. Проектирование. Кодирование. Тестирование. Эксплуатация. Сопровождение.
5. Характеристика этапов жизненного цикла программы.
6. Особенности создания программного продукта.
7. Этапы жизненного цикла программного продукта и его специфика.
8. Особенности разработки программного продукта.

Содержание отчета:

1) цель работы;
2) задание на лабораторную работу для своего варианта;
3) алгоритм решаемого задания с необходимыми пояснениями;
4) выводы по работе.

3. Контрольные вопросы

1. Понятие жизненного цикла программы и его этапы, анализ требований к программе.
2. Этапы жизненного цикла программного продукта и его специфика.

Лабораторная работа № 3
«Модели и процессы управлении проектами программных средств»

Цель работы: знать модели и процессы управлении проектами программных средств.

1. Краткие теоретические сведения

Назначение методологии CMM/CMMI — системы и модели оценки зрелости — состоит в предоставлении необходимых общих рекомендаций и инструкций предприятиям, производящим ПС, по выбору стратегии совершенствования качества процессов и продуктов, путем анализа степени их производственной зрелости и оценивания факторов, в наибольшей степени влияющих на качество ЖЦ ПС, а также посредством выделения процессов, требующих модернизации. Для достижения устойчивых результатов программной инженерии в процессе развития технологии и организации управления жизненным циклом ПС в стандарте ISO 15504:1-9 рекомендуется использовать эволюционный путь, который позволяет совершенствовать и постепенно повышать качество процессов и продуктов, вскрывать преимущества и недостатки предприятия. Виды деятельности для высоких уровней зрелости в соответствии с CMM в стандарте делятся на базовые и общие. Базовые виды деятельности являются обязательными и сгруппированы в пять категорий: контрактная; инженерная; управленческая; вспомогательная; организационная. Уровни зрелости характеризуются степенью формализации, адекватностью измерения и документирования процессов и продуктов ЖЦ ПС, широтой применения стандартов и инструментальных средств автоматизации работ, наличием и полнотой реализации функций системой обеспечения качества технологических процессов и их результатов.

Описание процессов ЖЦ ПС в CMM сфокусировано на поэтапном определении реально достигаемых результатов и на оценивании качества их выполнения. Качество процессов зависит от технологической среды, в которой они выполняются. Зрелость процессов — это степень их управляемости, возможность поэтапной количественной оценки качества, контролируемости и эффективности результатов. Модель зрелости предприятия представляет собой методический нормативный материал, определяющий правила создания и функционирования системы управления жизненным циклом ПС, методы и стандарты систематического повышения культуры и качества производства. Рост зрелости обеспечивает потенциальную возможность возрастания эффективности и согласованности использования процессов создания, сопровождения и оценивания качества компонентов и ПС в целом. Реальное использование регламентированных процессов предполагает поэтапный контроль и документирование их характеристик качества. На предприятиях, достигших высокого уровня зрелости, формализованные процессы ЖЦ ПС должны принимать статус стандарта, фиксироваться в организационных структурах и определять

производственную тактику и стратегию корпоративной культуры производства и системы обеспечения качества ПС.

Уровень 1 — Начальный. Массовые разработки проектов ПС характеризуются относительно небольшими размерами программ в несколько тысяч строк, создаваемых несколькими специалистами. Они применяют простейшие неформализованные технологии с использованием типовых инструментальных компонентов операционных систем. Основные процессы ЖЦ ПС на этом уровне не регламентированы, выполняются не совсем упорядоченно и зависят от некоординированных индивидуальных усилий и свойств отдельных специалистов. Успех проекта, как правило, зависит от энергичности, таланта и опыта нескольких руководителей и исполнителей. Процессы на первом уровне характеризуются своей непредсказуемостью по трудоемкости и срокам в связи с тем, что их состав, назначение и последовательность выполнения могут меняться случайным образом в зависимости от текущей ситуации.

Уровень 2 — Управляемый — базовое управление. Для сложных проектов ПС объемом в десятки и сотни тысяч строк, в которых участвуют десятки специалистов разной квалификации, необходимы организация, регламентирование технологии и унификация процессов деятельности каждого из них. Процессы на этом уровне заранее планируются, их выполнение контролируется, чем достигается предсказуемость результатов и времени выполнения этапов, компонентов и проекта в целом. Основной особенностью второго уровня является наличие формализованных и документированных процессов управления проектами, которые пригодны для модернизации, а их результаты поддаются количественной оценке. На этом уровне акценты управления сосредоточиваются на предварительном упорядочении и регламентировании процессов создания, сопровождения и оценивания качества программного средства, однако для крупных проектов ПС с гарантированным качеством риск провала может оставаться еще достаточно большим.

Уровень 3 — Определенный — стандартизация процессов. При высоких требованиях заказчика и пользователей к конкретным характеристикам качества сложного ПС и к выполнению ограничений по использованию ресурсов необходимо дальнейшее совершенствование и повышение уровня зрелости процессов ЖЦ ПС. Процессы ЖЦ ПС на этом уровне должны быть стандартизированы и представлять собой целостную технологическую систему, обязательную для всех подразделений. На основе единой технологии поддержки и обеспечения качества ЖЦ ПС для каждого проекта могут разрабатываться дополнительные процессы последовательного оценивания качества продуктов с учетом их особенностей. Описание каждого процесса должно включать условия его выполнения, входные данные, рекомендации стандартов и процедуры выполнения, механизмы проверки качества результатов, выходные данные, условия и документы завершения процессов. В описания процессов включаются сведения об

инструментальных средствах, необходимых для их выполнения, роль, ответственность и квалификация специалистов.

Уровень 4 — Предсказуемый — количественное управление. Для реализации проектов крупных, особенно сложных ПС, в жестко ограниченные сроки и с высоким гарантированным качеством, необходимы активные меры для предотвращения и выявления дефектов и ошибок на всех этапах ЖЦ ПС. Управление должно обеспечивать выполнение процессов в соответствии с текущими требованиями к характеристикам качества компонентов и ПС в целом. На этом уровне должна применяться система детального поэтапного оценивания характеристик качества как технологических процессов ЖЦ, так и самого создаваемого программного продукта и его компонентов. Должны разрабатываться и применяться методики количественной оценки реализации процессов и их качества. Одновременно с повышением сложности и требований к качеству ПС следует совершенствовать управление проектами за счет сокращения текущих корректировок и исправлений дефектов при выполнении процессов. Результаты процессов становятся предсказуемыми по срокам и качеству в связи с тем, что они измеряются в ходе их выполнения и реализуются в рамках заданных ресурсных ограничений.

Уровень 5 — Оптимизационный — непрерывное совершенствование и улучшение. Дальнейшее последовательное совершенствование и модернизация технологических процессов ЖЦ ПС для повышения качества их выполнения и расширение глубины контроля за их реализацией. Одна из основных целей этого уровня — сокращение проявлений и потерь от случайных дефектов и ошибок путем выявления сильных и слабых сторон используемых процессов. При этом приоритетным является анализ рисков, дефектов и отклонений от заданных требований заказчика. Эти данные также используются для снижения себестоимости ЖЦ особо сложных ПС в результате внедрения новых технологий и инструментария, а также для планирования и осуществления модернизации всех видов процессов. Технологические нововведения, которые могут принести наибольшую выгоду, должны стандартизироваться и адаптироваться в комплексную технологию обеспечения и оценивания системы качества предприятия и его продукции.

2. Порядок выполнения работы и содержание отчета

Порядок выполнения работы:

1. Важность учета и контроля проекта.
2. Зачем нужны проверки: пассивные и активные данные.
3. Планирование учета проекта.
4. Поэтапный учет результатов.
5. Метод допустимых границ.
6. Анализ товарных запасов.
7. Учет методом S-образной кривой.
8. Метод прибавочной стоимости.

9. Отчеты о результатах проверок и организация рабочих совещаний.
10. Выработка корректирующих воздействий.

Содержание отчета:

1) цель работы;
2) задание на лабораторную работу для своего варианта;
3) алгоритм решаемого задания с необходимыми пояснениями;
4) выводы по работе.

3. Контрольные вопросы

1. Прикладные программы с высокой степенью автоматизации.
2. Адаптируемость пакетов программ.

Лабораторная работа № 4
«Управление требованиями к программному обеспечению»

Цель работы: знать управление требованиями к программному обеспечению.

1. Краткие теоретические сведения

Наличие отличных требований всего лишь часть решения — они также должны хорошо управляться и эффективно доводиться до участников проекта. Управление версиями отдельных требований и их совокупностей — один из базовых аспектов управления требованиями.

Процесс управления требованиями.

Под управлением требованиями подразумевают все действия, по обеспечению целостности, точности и своевременности обновления соглашения о требованиях в ходе проекта.

В организации необходимо определить действия, которые, как ожидается, проектная команда будет выполнять для управления требованиями. Задокументируйте эти действия и организуйте тренинг для специалистов, чтобы сотрудники организации могли выполнять их последовательно и эффективно. Обратите внимание на следующее:

• на инструменты, приемы и соглашения для различения версий отдельных требований и их наборов;

• на составление наборов требований и их одобрение, а также на определение базовых версий требований;

• на способы, с помощью которых новые требования и изменения существующих требований предлагаются, обрабатываются, обсуждаются и доводятся до всех заинтересованных лиц;

• на оценку влияния предложенного изменения;

• на атрибуты требований и процедуры отслеживания состояния требований, в том числе состояния требований, которые будут использоваться, и кто их будет менять;

• кто отвечает за обновление информации связей требований и когда это должно делаться;

• как отлеживаются и разрешаются проблемы с требованиями;

• как на планах и обязательствах проекта отразятся изменения требований;

• как эффективно использовать инструменты управления требованиями.

Вы можете включить всю эту информацию в одно описание процесса управления требованиями. Или же разработать отдельные процедуры для управления версиями, управления изменениями, анализа влияния и отслеживания состояния. Эти процедуры следует довести до сведения каждого сотрудника организации, поскольку они представляют общие функции, которые должны выполняться каждым в проектной команде. Описания процесса должны определять роли в команде, которые отвечают за каждое из действий по управлению требованиями. Бизнес-аналитик проекта

как правило, несет основную ответственность за управление требованиями. Он выбирает механизм хранения требований, определяет атрибуты требований, координирует обновление данных о состоянии и отслеживает обновление данных, а также следит за изменениями в операциях, если это необходимо. Описание процесса также должно определять ответственного за изменения в процессе управления требованиями, порядок работы с исключениями и путь передачи вопросов вверх по цепочке ответственности в случае возникновения проблем.

2. Порядок выполнения работы и содержание отчета

Порядок выполнения работы:

1. Дисциплина требования.
2. Место дисциплины в разработки программного обеспечения.
3. Планирование процесса управления требованиями.
4. Анализ потребностей заинтересованных сторон.
5. Сбор и установление требований.
6. Организация и документирование требований.
7. Корректировка требований и управление ими.
8. Управление изменениями и внесение изменений в требования.

Содержание отчета:

1) цель работы;
2) задание на лабораторную работу для своего варианта;
3) алгоритм решаемого задания с необходимыми пояснениями;
4) выводы по работе.

3. Контрольные вопросы

1. Типовые приемы конструирования пакетов программ сложной структуры.
2. Организация проектирования программного обеспечения (ПО).

Лабораторная работа № 5
«Проектирование программного обеспечения»

Цель работы: знать, как происходит проектирование программного обеспечения.

1. Краткие теоретические сведения

Сегодня процесс создания сложных программных приложений невозможно представить без разделения на этапы жизненного цикла. Под жизненным циклом программы будем понимать совокупность этапов:

1. Анализ предметной области и создание ТЗ (взаимодействия с заказчиком).
2. Проектирование структуры программы.
3. Кодирование (набор программного кода согласно проектной документации).
4. Тестирование и отладка.
5. Внедрение программы.
6. Сопровождение программы.
7. Утилизация

Остановимся детально на процессе проектирования. В ходе проектирования архитектором или опытным программистом создается проектная документация, включающая текстовые описания, диаграммы, модели будущей программы. В этом нелегком деле нам поможет язык UML.

UML — является графическим языком для визуализации, описания параметров, конструирования и документирования различных систем (программ в частности). Диаграммы создаются с помощью специальных CASE средств, например Rational Rose (http://www-01.ibm.com/software/rational/) и Enterprise Architect (http://www.sparxsystems.com.au/). На основе технологии UML строится единая информационная модель. Приведенные выше CASE средства способны генерировать код на различных объектно-ориентированных языках, а так же обладают очень полезной функцией реверсивного инжиниринга. (Реверсивный инжиниринг позволяет создать графическую модель из имеющегося программного кода и комментариев к нему.)

Рассмотрим типы диаграмм для визуализации модели (это must have, хотя типов гораздо больше):

1. Диаграмма вариантов использования (use case diagram).
2. Диаграмма классов (class diagram).
3. Диаграмма состояний (statechart diagram).
4. Диаграмма последовательности (sequence diagram).
5. Диаграмма кооперации (collaboration diagram).
6. Диаграмма компонентов (component diagram).
7. Диаграмма вариантов использования (use case diagram).

Проектируемая система представляется в виде множества сущностей или актеров, взаимодействующих с системой с помощью, так называемых

прецедентов. При этом актером (actor) или действующим лицом называется любая сущность, взаимодействующая с системой извне. Другими словами, каждый вариант использования определяет некоторый набор действий, совершаемый системой при диалоге с актером. При этом ничего не говорится о том, каким образом будет реализовано взаимодействие актеров с системой.

Диаграмма классов (class diagram).

Диаграмма классов служит для представления статической структуры модели системы в терминологии классов объектно-ориентированного программирования. Диаграмма классов может отражать, в частности, различные взаимосвязи между отдельными сущностями предметной области, такими как объекты и подсистемы, а также описывает их внутреннюю структуру (поля, методы…) и типы отношений (наследование, реализация интерфейсов …). На данной диаграмме не указывается информация о временных аспектах функционирования системы. С этой точки зрения диаграмма классов является дальнейшим развитием концептуальной модели проектируемой системы. На этом этапе принципиально знание ООП подхода и паттернов проектирования.

Диаграмма состояний (statechart diagram).

Главное предназначение этой диаграммы — описать возможные последовательности состояний и переходов, которые в совокупности характеризуют поведение элемента модели в течение его жизненного цикла. Диаграмма состояний представляет динамическое поведение сущностей, на основе спецификации их реакции на восприятие некоторых конкретных событий.

Диаграмма последовательности (sequence diagram).

Для моделирования взаимодействия объектов в языке UML используются соответствующие диаграммы взаимодействия. Взаимодействия объектов можно рассматривать во времени, и тогда для представления временных особенностей передачи и приема сообщений между объектами используется диаграмма последовательности. Взаимодействующие объекты обмениваются между собой некоторой информацией. При этом информация принимает форму законченных сообщений. Другими словами, хотя сообщение и имеет информационное содержание, оно приобретает дополнительное свойство оказывать направленное влияние на своего получателя.

Диаграмма кооперации (collaboration diagram).

На диаграмме кооперации в виде прямоугольников изображаются участвующие во взаимодействии объекты, содержащие имя объекта, его класс и, возможно, значения атрибутов. Как и на диаграмме классов, указываются ассоциации между объектами в виде различных соединительных линий. При этом можно явно указать имена ассоциации и ролей, которые играют объекты в данной ассоциации.

В отличие от диаграммы последовательности, на диаграмме кооперации изображаются только отношения между объектами, играющими определенные роли во взаимодействии.

Диаграмма компонентов (component diagram).

Диаграмма компонентов, в отличие от ранее рассмотренных диаграмм, описывает особенности физического представления системы. Диаграмма компонентов позволяет определить архитектуру разрабатываемой системы, установив зависимости между программными компонентами, в роли которых может выступать исходный, бинарный и исполняемый код. Во многих средах разработки модуль или компонент соответствует файлу. Пунктирные стрелки, соединяющие модули, показывают отношения взаимозависимости, аналогичные тем, которые имеют место при компиляции исходных текстов программ. Основными графическими элементами диаграммы компонентов являются компоненты, интерфейсы и зависимости между ними.

Диаграмма развертывания (deployment diagram).

Диаграмма развертывания предназначена для визуализации элементов и компонентов программы, существующих лишь на этапе ее исполнения (runtime). При этом представляются только компоненты-экземпляры программы, являющиеся исполнимыми файлами или динамическими библиотеками. Те компоненты, которые не используются на этапе исполнения, на диаграмме развертывания не показываются.

Диаграмма развертывания содержит графические изображения процессоров, устройств, процессов и связей между ними. В отличие от диаграмм логического представления, диаграмма развертывания является единой для системы в целом, поскольку должна всецело отражать особенности ее реализации. Эта диаграмма, по сути, завершает процесс ООАП для конкретной программной системы и ее разработка, как правило, является последним этапом спецификации модели.

На этом закончим обзорный экскурс по диаграммам в частности и проектированию в общем. Стоит отметить, что процесс проектирования уже давно стал стандартом разработки ПО, но часто приходится сталкиваться с великолепно написанной программой, которая из за отсутствия нормальной документации обрастает ненужным побочным функционалом, костылями, становится громоздкой и теряет былое качество.

2. Порядок выполнения работы и содержание отчета

Порядок выполнения работы:

1. Методы проектирования программных продуктов и признаки их классификации.
2. Неавтоматизированное и автоматизированное проектирование алгоритмов и программ.
3. Структурное проектирование программных продуктов и его методы.
4. Принцип системного проектирования.
5. Нисходящее проектирование. Модульное проектирование. Структурное проектирование.
6. Функционально-ориентированные методы и методы структурирования данных.

7. Информационное моделирование предметной области и его составляющие.
8. Технологии информационного моделирования.
9. Инфологические и даталогические модели.
10. Логический и физический уровень представления даталогической модели.
11. Сущность объектно-ориентированного подхода к проектированию программных продуктов.
12. Объектно-ориентированный анализ предметной области и объектно-ориентированное проектирование.
13. Объектно-ориентированная технология и ее преимущества.

Содержание отчета:

1) цель работы;
2) задание на лабораторную работу для своего варианта;
3) алгоритм решаемого задания с необходимыми пояснениями;
4) выводы по работе.

3. Контрольные вопросы

1. Этапы процесса проектирования.
2. Основные процессы жизненного цикла ПС.
3. Вспомогательные процессы жизненного цикла ПС.
4. Организационные процессы жизненного цикла ПС.

Лабораторная работа № 6
«Конструирование (детальное проектирование) программного обеспечения»

Цель работы: знать конструирование программного обеспечения

1. Краткие теоретические сведения

Для каждой единицы программного обеспечения (или единицы конфигурации программного обеспечения, если идентифицировано). Эта деятельность состоит из нескольких задач:

1. Разработчик должен разработать подробный проект для каждого компонента единицы программного обеспечения. Компоненты программного обеспечения должны быть усовершенствованы в более нужных уровнях, вмещающих элементы, блоки, узлы, части программного обеспечения, которые могут быть запрограммированы, откомпилированы и тестированы.

2. Разработчик должен разработать и регламентировать детальный проект для внешних интерфейсов к единицам программного обеспечения, между компонентами программного обеспечения и между частями программного обеспечения. Детальный проект интерфейсов должен позволять программировать без потребности в дальнейшей информации.

3. Разработчик должен разработать и документировать детальный проект для базы данных.

4. Разработчик должен модернизировать документацию пользователя по мере необходимости.

5. Разработчик должен определить и документировать требования к испытаниям и режимы тестирования частей программного обеспечения. Требования к тестированию должны включать вес элемента в пределах требований.

6. Разработчик должен модифицировать требования к тестированию и план интеграции программного обеспечения.

7. Разработчик должен оценить детальный проект и требования к тестированию согласно критериям, описанным ниже. Результаты оценок должны быть документированы. Разработчик должен разработать и документировать следующее:

а) трассируемость к требованиям единицы программного обеспечения;

б) внешняя согласованность с архитектурой;

в) внутренняя согласованность между компонентами программного обеспечения и единицами программного обеспечения;

г) соответствие методов проектирования и используемых стандартов;

д) возможность тестирования;

е) возможность функционирования и сопровождения.

Для каждой единицы программного обеспечения (или единицы конфигурации программного обеспечения, если унифицировано) эти действия состоят из следующих задач:

а) каждый элемент программного обеспечения и базу данных;

б) испытательные процедуры и дату тестирования каждого элемента программного обеспечения и базы данных.

2. Порядок выполнения работы и содержание отчета

Порядок выполнения работы:

1. Языки программирования и их классификация.

2. Выбор и обоснование языка программирования.

3. Языки программирования для решения экономических, инженерных, научных задач, языки системного программирования, комбинирование языков программирования в рамках одной задачи.

Содержание отчета:

1) цель работы;

2) задание на лабораторную работу для своего варианта;

3) алгоритм решаемого задания с необходимыми пояснениями;

4) выводы по работе.

3. Контрольные вопросы

1. Способы формального представления знаний.

2. Основы устройства и использование экспертных систем в разработке адаптируемого программного обеспечения.

Лабораторная работа № 7
«Тестирование программного обеспечения»

Цель работы: уметь тестировать программное обеспечение.

1. Краткие теоретические сведения

Существующие на сегодняшний день методы тестирования ПО не позволяют однозначно и полностью выявить все дефекты и установить корректность функционирования анализируемой программы, поэтому все существующие методы тестирования действуют в рамках формального процесса проверки исследуемого или разрабатываемого ПО.

Такой процесс формальной проверки или верификации может доказать, что дефекты отсутствуют с точки зрения используемого метода (то есть нет никакой возможности точно установить или гарантировать отсутствие дефектов в программном продукте с учётом человеческого фактора, присутствующего на всех этапах жизненного цикла ПО).

Существует множество подходов к решению задачи тестирования и верификации ПО, но эффективное тестирование сложных программных продуктов — это процесс в высшей степени творческий, не сводящийся к следованию строгим и чётким процедурам или созданию таковых.

С точки зрения ISO 9126, качество (программных средств) можно определить как совокупную характеристику исследуемого ПО с учётом следующих составляющих:

· надёжность;
· сопровождаемость;
· практичность;
· эффективность;
· мобильность;
· функциональность.

Более полный список атрибутов и критериев можно найти в стандарте ISO 9126 Международной организации по стандартизации. Состав и содержание документации, сопутствующей процессу тестирования, определяется стандартом IEEE 829-1998 Standard for Software Test Documentation.

История развития тестирования программного обеспечения.

Тестирование программного обеспечения.

Существует несколько признаков, по которым принято производить классификацию видов тестирования. Обычно выделяют следующие:

По объекту тестирования:

· функциональное тестирование (functional testing);
· нагрузочное тестирование;
· тестирование производительности (perfomance/stress testing);
· тестирование стабильности (stability/load testing);
· тестирование удобства использования (usability testing);
· тестирование интерфейса пользователя (UI testing);

· тестирование безопасности (security testing);
· тестирование локализации (localization testing);
· тестирование совместимости (compatibility testing).

По знанию системы:
· тестирование чёрного ящика (black box);
· тестирование белого ящика (white box);
· тестирование серого ящика (gray box).

По степени автоматизированности:
· ручное тестирование (manual testing);
· автоматизированное тестирование (automated testing);
· полуавтоматизированное тестирование (semiautomated testing).

По степени изолированности компонентов:
· компонентное (модульное) тестирование (component/unit testing);
· интеграционное тестирование (integration testing);
· системное тестирование (system/end-to-end testing).

По времени проведения тестирования:
· альфа тестирование (alpha testing);
· тестирование при приёмке (smoke testing);
· тестирование новых функциональностей (new feature testing);
· регрессионное тестирование (regression testing);
· тестирование при сдаче (acceptance testing);
· бета тестирование (beta testing).

По признаку позитивности сценариев:
· позитивное тестирование (positive testing);
· негативное тестирование (negative testing).

По степени подготовленности к тестированию:
· тестирование по документации (formal testing);
· Эд Хок (интуитивное) тестирование (ad hoc testing).

Уровни тестирования

Модульное тестирование (юнит-тестирование) — тестируется минимально возможный для тестирования компонент, например, отдельный класс или функция. Часто модульное тестирование осуществляется разработчиками ПО.

Интеграционное тестирование — тестируются интерфейсы между компонентами, подсистемами. При наличии резерва времени на данной стадии тестирование ведётся итерационно, с постепенным подключением последующих подсистем.

Системное тестирование — тестируется интегрированная система на её соответствие требованиям.

2. Порядок выполнения работы и содержание отчета

Порядок выполнения работы:

1. Сущность и необходимость тестирования программного обеспечения.
2. Различие между тестированием и отладкой программ.

3. Основные принципы организации тестирования.
4. Стадии тестирования.
5. Виды тестовых проверок.
6. Объекты тестирования и категории тестов.
7. Виды тестирования. Методы структурного тестирования программного обеспечения.
8. Принцип «белого ящика».
9. Пошаговое и монолитное тестирование модулей.
10. Нисходящее и восходящее тестирование программного обеспечения.
11. Методы функционального тестирования.
12. Принцип «черного ящика».
13. Метод эквивалентного разбиения.
14. Метод анализа граничных условий.
15. Метод функциональных диаграмм.
16. Комбинированные методы тестирования.
17. Средства тестирования.
18. Ручное и автоматизированное тестирование.
19. Инструментальные средства тестирования.

Содержание отчета:

1) цель работы;
2) задание на лабораторную работу для своего варианта;
3) алгоритм решаемого задания с необходимыми пояснениями;
4) выводы по работе.

3. Контрольные вопросы

1. Сущность и необходимость тестирования программного обеспечения.
2. Применение методов и инструментальных средств тестирования.

Лабораторная работа № 8
«Сопровождение программного обеспечения»

Цель работы: знать, как происходит сопровождение программного обеспечения.

1. Краткие теоретические сведения

Процесс сопровождения предусматривает действия и задачи, выполняемые сопровождающей организацией (службой сопровождения). Данный процесс активизируется при изменениях (модификациях) программного продукта и соответствующей документации, вызванных возникшими проблемами или потребностями в модернизации либо адаптации ПО.

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

Процесс сопровождения охватывает следующие действия:

1) подготовительную работу;
2) анализ проблем и запросов на модификацию ПО;
3) модификацию ПО;
4) проверку и приемку;
5) перенос ПО в другую среду;
6) снятие ПО с эксплуатации.

Сопровождение программного обеспечения — процесс улучшения, оптимизации и устранения дефектов программного обеспечения (ПО) после передачи в эксплуатацию.

Сопровождение ПО — это одна из фаз жизненного цикла программного обеспечения, следующая за фазой передачи ПО в эксплуатацию.

В ходе сопровождения в программу вносятся изменения, с тем, чтобы исправить обнаруженные в процессе использования дефекты и недоработки, а также для добавления новой функциональности, с целью повысить удобство использования и применимость ПО.

Самый лучший вид сопровождения — это, конечно, отсутствие всякого сопровождения. Однако при существующем уровне развития программного обеспечения сдача с первого предъявления является маловероятным событием. Обычно требуются, по крайней мере, корректировка программного обеспечения, а возможно, даже и его адаптация, и совершенствование. Сдача с первого предъявления предполагает применение большинства из описанных выше методов повышения надежности и немного удачи.

Существует три вида сопровождения:

1. Сопровождение с целью исправления ошибок. Самые дорогие исправления, связанные с ошибками в системных требованиях, то есть здесь может наблюдаться перепроектирование системы.

2. Сопровождение с целью адаптации ПО специфическим условиям эксплуатации, это может потребоваться при изменении определенных составляющих рабочего окружения системы. Что бы адаптироваться к этим изменениям система должна быть подвержена определенным модификациям.

3. Сопровождение с целью изменения функциональных возможностей системы в ответ на изменения в организации могут измениться требования к программным средствам.

2. Порядок выполнения работы и содержание отчета

Порядок выполнения работы:

1. Сопровождение программных продуктов.
2. Внесение изменений.
3. Обеспечение надежности при эксплуатации.
4. Необходимая документация и предпродажная подготовка программных средств.

Содержание отчета:

1) цель работы;
2) задание на лабораторную работу для своего варианта;
3) алгоритм решаемого задания с необходимыми пояснениями;
4) выводы по работе.

3. Контрольные вопросы

1. Сопровождение программных продуктов.
2. Предпродажная подготовка программных средств.

Лабораторная работа № 9
«Конфигурационное управление»

Цель работы: знать конфигурационное управление.

1. Краткие теоретические сведения

Конфигурация — это знание о том, какова работающая (актуальная) система, выделенная из множества ее возможных вариантов. Конфигурация находится под контролем, если конфигурация определения системы соответствует конфигурации воплощения системы. Если какие-то части этих конфигураций не соответствуют друг другу, то говорят о конфигурационных коллизиях.

Управление конфигурацией (configuration management) — техническая дисциплина системной инженерии, обеспечивающая поддержание надлежащей (задуманной, одобренной) конфигурации системы во время всего её жизненного цикла. Если говорить попроще, то управление конфигурацией — это практика, обеспечивающая на протяжении всего жизненного цикла совместимость версий (отсутствие коллизий) и полноту частей системы (отсутствие коллизий!).

Управление конфигурацией — практика системноинженерного менеджмента — она занимается поддержанием целостности системы на протяжении всего ЖЦ. В рамках этой практики выпускаются различные виды спецификаций закупаемого/изготавливаемого оборудования/изделий — BOM (bill of materials, список комплектующих).

Отнесение управления конфигурацией к системной инженерии означает, что конфигурация обязательно относится ко всей системе.

Управляющий конфигурацией — системный инженер, наряду с инженером по требованиям, системным архитектором, интегратором.

Основные понятия.

Управление конфигурацией включает в себя следующие понятия:

базис (configuration baseline) — исходная (утвержденная) конфигурация. Базис определяется на следующих этапах:

1. Базис определяется на этапе Тип спецификации. Характеристики. Описываемый элемент. Функциональный (Functional).

2. Выбор концепции A. Функциональные спецификации. Система. Физический (Allocated). Техническое проектирование B.

3. Проектная документация. Элемент конфигурации. Продукции (Product). Техническое проектирование C, D, E. Производственно-технологическая документация. Элемент конфигурации версия/ревизия (version/revision).

4. Элемент конфигурации (configuration item, CI) — элемент системы, который является основой для описания и формального управления проектированием системы, базовая часть системы, которая проектируется, конструируется и создается силами одной организации. Характеристики и интерфейсы CI с другими составными частями должны быть определены и

контролироваться, чтобы гарантировать надлежащее функционирование CI в составе системы в целом. Различают:

а) аппаратные элементы конфигурации (hardware CI — HWCI);

б) элементы конфигурации программного обеспечения компьютера (computer software CI — CSCI);

в) управление интерфейсами;

г) управление изменениями.

2. Порядок выполнения работы и содержание отчета

Порядок выполнения работы:

1. Внутренняя организация программных продуктов.
2. Типовая структура программных продуктов.
3. Головной, управляющий модуль, рабочие и сервисные модули.
4. Структура пакета прикладных программ.
5. Библиотеки стандартных программ и подпрограмм.
6. Возможность использования встроенных функций.

Содержание отчета:

1) цель работы;
2) задание на лабораторную работу для своего варианта;
3) алгоритм решаемого задания с необходимыми пояснениями;
4) выводы по работе.

3. Контрольные вопросы

1. Цели структуризации программных продуктов.
2. Встроенные функции.

Лабораторная работа № 10
«Управление программной инженерией»

Цель работы: управлять программной инженерией.

1. Краткие теоретические сведения

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

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

Затраты в жизненном цикле ПС определяются не только этапами разработки, но и этапами эксплуатации и сопровождения. Затраты на этих этапах могут значительно превышать затраты при разработке и характеризуются своими особыми закономерностями. Однако эффективность процесса разработки ПС невозможно определять без учета эффективности последующей эксплуатации, а, для, долго модифицируемых программ, без оценки эффективности их сопровождения. Ряд факторов влияет на затраты при разработке сложных ПС не только непосредственно, но и через возможное изменение затрат в дальнейшем при сопровождении или эксплуатации. Каждый из этапов: разработка, сопровождение и эксплуатация может быть достаточно длительным. В пределах этапов различные группы

затрат могут быть неодновременными и разделяться интервалами времени, исчисляемыми годами. Однако, разновременность затрат трудно учитывать в общем виде, и при существующих методиках, имеется некоторая условность при оценке влиянии времени на совокупные затраты проекта.

Цель планирования жизненного цикла программного средства состоит в выборе и определении способов создания и совершенствования ПС, которые способны удовлетворить требованиям технического задания, спецификаций и контракта, а также обеспечить уровень качества, соответствующий заданным требованиям. В современных стандартах подчеркивается, что эффективное планирование определяющий фактор высокого качества всего ЖЦ программного средства, удовлетворяющего требованиям заказчика. В стандартах ISO 12207 и ISO 16326 рекомендуется определить администратора, который должен подготовить планы для выполнения процессов ЖЦ ПС. Планы, связанные с выполнением процесса, должны содержать описания соответствующих работ и задач, обозначения создаваемых компонентов программного средства и охватывать следующие задачи:

1) установление графиков своевременного решения частных задач и всего ПС; оценки необходимых трудозатрат на задачи и проект в целом; определение ресурсов, необходимых для выполнения задач и проекта; распределение задач по исполнителям;

2) определение обязанностей исполнителей; определение критических ситуаций, связанных с задачами или процессами ЖЦ ПС; установление используемых в процессах ЖЦ ПС критериев управления качеством; - определение затрат, связанных с реализацией каждого процесса; обеспечение условий и определение инфраструктуры выполнения процессов ЖЦ ПС.

Должны быть определены обязанности специалистов по подготовке и утверждению (согласованию) планов. Следует установить модель жизненного цикла программного средства, задачи, распределение задач, их блокировку и соответствующие ресурсы. В программном проекте должен быть определен один основной график работ, а все вспомогательные графики должны быть связаны и согласованы с основным графиком. С помощью структуры классификации работ можно эффективно проверять ход процессов и обеспечивать контроль этих процессов и продуктов. План должен быть применен так, чтобы обеспечить управление программным проектом на всех уровнях его детализации с использованием соответствующих технологий в зависимости от объема, сложности, критичности и риска проекта. Оценки проекта, используемые при планировании, должны охватывать: стоимость реализации соответствующих процессов; инфраструктуру обеспечения реализации процессов; потребности в ресурсах, включая соответствующее управление и контроль; оценку и контроль качества реализации процессов; управление риском результатов процессов; обеспечение среды программной инженерии проекта ПС; задания, выполняемые в каждом процессе и (или) работе.

Общее представление о качестве ПС международным стандартом ISO 9126:1-4:2002 рекомендуется описывать тремя взаимодействующими и взаимозависимыми метриками характеристик качества, отражающими:

1) внутреннее качество, проявляющееся в процессе разработки и других промежуточных этапов жизненного цикла ПС;

2) внешнее качество, заданное требованиями заказчика в спецификациях и отражающееся характеристиками конечного продукта;

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

Внутренние метрики в соответствии со стандартами могут применяться в ходе проектирования и программирования к компонентам ПС таким, как спецификация или исходный программный текст. Основная цель применения внутренних метрик обеспечивать, чтобы разработчиками было получено требуемое внешнее качество. Рекомендуется использовать внутренние метрики, которые имеют наиболее сильные связи с приоритетными внешними метриками, чтобы они могли помогать при прогнозировании их достижимых значений. Внутренние метрики дают возможность разработчикам, испытателям и заказчикам, начиная с системного проектирования, прогнозировать качество жизненного цикла программ и заниматься вопросами технологического обеспечения качества до того, как ПС становится готовым к использованию продуктом. Измерения внутренних метрик используют свойства, категории, числа или характеристики элементов ПС, которые, например, имеются в процедурах исходного программного текста, в графе потока управления, в потоке данных и в описаниях изменения состояний памяти.

2. Порядок выполнения работы и содержание отчета

Порядок выполнения работы:

1. Построение моделей программных систем с использованием структурного и объектно-ориентированного подходов.
2. Диаграммы потоков данных и диаграммы «сущность-связь».
3. Построение концептуальной модели предметной области.
4. Диаграммы моделирования языка UML.
5. Работа в среде CASE-средства.

Содержание отчета:

1) цель работы;

2) задание на лабораторную работу для своего варианта;

3) алгоритм решаемого задания с необходимыми пояснениями;

4) выводы по работе.

3. Контрольные вопросы

1. Стандартизация и метрология в разработке программного обеспечения.

2. Стандартизация информационных технологий.

Лабораторная работа № 11
«Процесс программной инженерии»

Цель работы: рассмотреть процесс программной инженерии.

1. Краткие теоретические сведения

Авторская разработка — принцип создания программных продуктов, при котором весь жизненный цикл разработки поддерживается одним единственным человеком.

Этот принцип был достаточно широко распространен в 70—80-е годы XX века. Сейчас он применяется редко [Кулаичев 1999]. Примерами авторских разработок являются операционная система Диспак (В.Ф. Тюрин), текстовый редактор Лексикон (Е. М. Веселов), трансляторы с языков Algol-68 (П. Наур) и Pascal (Н. Вирт).

Принцип авторской разработки неприменим для многих современных разработок из-за их сложности, объема и требований к качеству и сопровождению. С другой стороны, программное обеспечение, начиная с момента появления персональных компьютеров, стало продуктом массового потребления, приносящим огромный доход. В этой области быстро выросли и стали доминировать крупные компьютерные компании с развитой структурой менеджмента и мощной рекламой.

Наиболее интересен принцип авторской разработки с точки зрения применения в области наукоемких приложений. Для таких приложений характерна необходимость многолетнего изучения предметной области, практически полное отсутствие начального финансирования проекта, малая рентабельность, определяемая узким кругом пользователей.

По данным А. П. Кулаичева [Кулаичев 1999], авторская разработка может выигрывать по производительности в тридцать и более раз у коллективной разработки, что достигается за счет:

1) исключения межличностных коммуникаций, связанных с необходимостью порождения и изучения большого количества технологической документации;

2) исключения работ по разбиению проекта на составляющие, по распределению их между исполнителями, по координации деятельности исполнителей и контролю за их работой.

Объем программного продукта, выполненного методом авторских разработок в 5—20 раз меньше по сравнению с индустриальными аналогами.

Авторская разработка предполагает достижение профессионального успеха, известности и славы в одиночку. Такое вполне реально, следует только правильно выбрать профессиональную «нишу», область ведения разработки.

Коллективная разработка.

Одним из основных вопросов коллективной разработки является разделение труда — от равноправных соисполнителей до организации в виде жесткой иерархии (например, бригады главного программиста).

Известно, что первые коллективные разработки программ велись примерно так. Начальник выполнял разделение большого проекта на меньшие части и передавал далее по иерархии. Через некоторое время, теперь уже снизу вверх, шла сборка программы из написанных фрагментов. Собрать работающий программный продукт удавалось не всегда.

Технические командные роли Равноправные соисполнители.

Бригада равноправных соисполнителей обычно состоит из специалистов, занимающихся примерно подобными задачами в рамках одного проекта. Естественно, специализаций в рамках одной бригады может быть несколько. Примерный состав такой бригады разработчиков следующий:

1) инженеры-разработчики (специалисты по инженерии программирования и программисты);
2) технические писатели;
3) инженеры тестирования;
4) инженеры качества;
5) специалисты по сопровождению продукта;
6) специалисты по продажам продукта.

Тип работы определяет содержание и природу выполняемой работы. Приведем список типов работ и областей специализации на основе классификации Конгер [Conger 1994]:

1. Разработка приложений.
2. Программист.
3. Специалист по инженерии программирования.
4. Специалист по инженерии знаний.
5. Работа с приложениями.
6. Специалист по приложениям.
7. Администратор данных.
8. Администратор базы данных.
9. Техническая поддержка.
10. Системный администратор.
11. Сетевой администратор.
12. Администратор коммуникаций.
13. Обеспечение качества продукта.
14. Технический писатель.
15. Инженер тестирования.
16. Инженер качества.
17. Маркетинг.
18. Специалист по сопровождению продукта.
19. Специалист по продажам продукта.
20. Системное интегрирование.
21. Системный интегратор.

Из перечисленных специализаций очень интересна специализация системного интегратора. Основные задачи системного интегратора — предложить заказчику вариант решения его проблемы, выбрав наиболее приемлемый по цене и технике, и реализовать его. Таким образом,

системный интегратор продает решения и несет ответственность за их реализацию. Системный интегратор как профессионал должен обладать знаниями из очень многих областей — прикладное и системное программное обеспечение, администрирование систем, аппаратура, сети, экономика и т. п.

2. Порядок выполнения работы и содержание отчета

Порядок выполнения работы:

1. Категории специалистов, занятых разработкой и эксплуатацией программ.
2. Принципы и методы коллективной разработки программных продуктов.
3. Организация коллективной работы программистов.
4. Схема взаимодействия специалистов, связанных с созданием и эксплуатацией программ
5. Типы организации бригад.
6. Бригада главного программиста.
7. Обязанности членов бригады.
8. Распределение обязанностей в бригаде.

Содержание отчета:

1) цель работы;
2) задание на лабораторную работу для своего варианта;
3) алгоритм решаемого задания с необходимыми пояснениями;
4) выводы по работе.

3. Контрольные вопросы

1. Категории специалистов, занятых разработкой и эксплуатацией программ.
2. Распределение обязанностей в бригаде.

Лабораторная работа № 12
«Инструменты и методы программной инженерии»

Цель работы: рассмотреть инструменты и методы программной инженерии

1. Краткие теоретические сведения

Техники интеграции инструментов. Эти техники важны для естественного использования сочетания различных инструментов. Типичные виды интеграции инструментов включают платформы, представление, процессы, данные и управление.

Мета-инструменты.Такие средства генерируют другие инструменты. Например, классическим примероммета-инструмента является компилятор компиляторов.

Оценка инструментов. Данная тема представляется достаточно важной в силу постоянной эволюции инструментов программной инженерии.

Данная секция (подобласть) разделена на три темы: эвристические методы (heuristic methods), касающиеся неформализованных подходов; формальные методы (formal methods), обоснованные математически; методы прототипирования (prototyping methods), базирующиеся на различных формах прототипирования. Эти три темы не являются изолированными друг от друга, скорее они выделены исходя из их значимости и на основе определенных достаточно явных индивидуальных особенностей. Например, объектно-ориентированный подход может включать формальные техники и использовать прототипирование для проверки и аттестации. Так же как и инструменты, методы программной инженерии постоянно эволюционируют. Именно поэтому, в описании данной области знаний авторы SWEBOK постарались избежать, насколько это возможно, упоминания любых конкретных методологий.

Эвристические методы (Heuristic Methods).

Эта тема содержит четыре категории методов: структурные, ориентированные на данные, объектно-ориентированные и ориентированные на область применения.

Структурные методы (structured methods). При таком подходе системы строится с функциональной точки зрения, начиная с высокоуровневого понимания поведения системы с постепенным уточнением низко-уровневых деталей (такой подход, иногда, также называют “проектированием сверху-вниз”).

Методы, ориентированные на данные (data-orientedmethods). Отправной точкой такого подхода являются структуры данных, которыми манипулирует создаваемое программное обеспечение. Функции в этом случае являются вторичными.

Объектно-ориентированные методы (object-orientedmethods). Система при таком подходе рассматривается как коллекция объектов, а не функций.

Методы, ориентированные на конкретную область применения (domain-specificmethods). Такие специализированные методы разрабатываются с учетом специфики решаемых задач, например, систем реального времени, безопасности жизнедеятельности (safety) и защищенности от несанкционированного доступа (security).

2. Порядок выполнения работы и содержание отчета

Порядок выполнения работы:

1. Общая характеристика инструментальных средств разработки программ.
2. Инструменты разработки программного обеспечения.
3. Инструментальные средства программирования.
4. Инструментальные системы технологии программирования и их основные черты: комплексность, ориентированность на коллективную разработку, технологическая определенность.
5. Интегрированность.
6. Основные компоненты инструментальных систем технологии программирования: репозиторий, инструментарий, интерфейсы. CASE-средства, их назначение и применение.
7. Характеристика современных CASE-средств.

Содержание отчета:

1) цель работы;
2) задание на лабораторную работу для своего варианта;
3) алгоритм решаемого задания с необходимыми пояснениями;
4) выводы по работе.

3. Контрольные вопросы

1. Общая характеристика инструментальных средств разработки программ.
2. Характеристика современных CASE-средств.

Лабораторная работа № 13
«Качество программного обеспечения»

Цель работы: рассмотреть качество программного обеспечения.

1. Краткие теоретические сведения

Разработка ПО достигла такого уровня развития, что возникла необходимость в использовании инженерных методов оценивания результатов его проектирования на всех этапах ЖЦ, контроля степени достижения запланированных показателей качества и их метрического анализа, оценки риска и степени использования готовых компонентов для снижения стоимости разработки нового проекта. Основу инженерных методов в программировании составляет повышение качества ПО, для достижения которого сформировались методы определения требований к качеству, подходы к выбору и усовершенствованию моделей метрического анализа показателей качества и методы количественного измерения показателей качества на всех этапах ЖЦ.

Согласие, достигнутое по требованиям к качеству, наравне с четким доведением до инженеров того, что составляет качество получаемого продукта, требует обсуждения и формального определения многих аспектов. Инженеры должны понимать смысл, вкладываемый в концепцию качества, характеристики и значение качества в отношении разрабатываемого или сопровождаемого ПО. Следует отметить, что программные требования определяют требуемые характеристики качества ПО, а также влияют на методы количественной оценки и сформулированные для оценки этих характеристик соответствующие критерии приемки.

Качество ПО является предметом стандартизации. Согласно ГОСТ 2844-94 качество ПО есть совокупность свойств (показателей качества) ПО, которые обеспечивают его способность удовлетворять потребности заказчика в соответствии с его назначением. Этот стандарт регламентирует базовую модель качества и показатели, главным среди которых является надежность. Стандарт 180/1ЕС12207 определил не только основные процессы ЖЦ разработки ПО, но и организационные и дополнительные процессы, которые регламентируют инженерию, планирование и управление качеством ПО.

Согласно этому стандарту на всех этапах ЖЦ разработки ПО должен проводиться следующий контроль качества ПО:

- проверка соответствия требований проектируемому программному продукту и критериев их достижения;
- верификация и аттестация (валидация) промежуточных результатов ПО на этапах ЖЦ и измерение степени удовлетворения достигаемых показателей;
- тестирование готового ПО, сбор данных об отказах, дефектах и других ошибках, обнаруженных в системе;
- подбор моделей надежности для оценивания надежности по полученным результатам тестирования (дефекты, отказы и др.);

• оценка показателей качества, заданных в требованиях на разработку ПО.

Инспектирование качества — это процесс проверки качества, ориентированный на команду разработчиков. Он применяется на всех этапах разработки ПП.

Доказательство правильности — это математическая или логическая методика, используемая для убеждения себя и других в том, что программа делает то, что должна делать. Такое доказательство является формальным (строгим) методом.

Для любого инженерного продукта существует множество интерпретаций качества. Показатели качества могут требоваться в той или иной степени, могут отсутствовать или могут отражать определенные требования потребителя и других заинтересованных сторон, быть результатом определенного компромисса (что вполне перекликается с пониманием «приемлемого качества», менее жесткой точки зрения на обеспечение качества как достижение совершенства).

2. Порядок выполнения работы и содержание отчета

Порядок выполнения работы:

1. Функциональность и надежность как обязательные критерии качества программного продукта.
2. Корректность программ, ее составляющие, программные эталоны и методы проверки корректности.
3. Обеспечение легкости применения продукта.
4. Обеспечение мобильности, модифицируемости и интеграции программных продуктов.

Содержание отчета:

1) цель работы;
2) задание на лабораторную работу для своего варианта;
3) алгоритм решаемого задания с необходимыми пояснениями;
4) выводы по работе.

3. Контрольные вопросы

1. Разработка автоматизированного рабочего места сотрудника отдела динамики и мультимедиа.
2. Разработка автоматизированной информационной системы торговой деятельности фирмы.

Лабораторная работа № 14
«Документирование программного обеспечения»

Цель работы: рассмотреть процесс документирования программного обеспечения.

1. Краткие теоретические сведения

Когда программист-разработчик получает в той или иной форме задание на программирование, перед ним, перед руководителем проекта и перед всей проектной группой встают вопросы: что должно быть сделано, кроме собственно программы? что и как должно быть оформлено в виде документации? что передавать пользователям, а что — службе сопровождения? как управлять всем этим процессом? Кроме упомянутых вопросов есть и другие, например, что должно входить в само задание на программирование?

Качество программного обеспечения, наряду с другими факторами, определяется полнотой и качеством пакета документов, сопровождающих ПО. К программным документам относятся документы, содержащие сведения, необходимые для разработки, изготовления, сопровождения программ и эксплуатации.

Документирование программного обеспечения включает в себя:

1. Техническое задание.
2. Внешние и внутренние языки спецификации.
3. Руководство пользователя.
4. Руководство программиста.
5. Техническое задание

Техническое задание. Требование к содержанию и оформлению. Техническое задание (ТЗ) содержит совокупность требований к программным системам и может использоваться как критерий проверки и приемки разработанной программы. Поэтому достаточно полно составленное (с учетом возможности внесения дополнительных разделов) и принятое заказчиком и разработчиком, ТЗ является одним из основополагающих документов проекта программного средства.

Техническое задание на разработку ПО должно включать следующие разделы:

- введение;
- основания для разработки;
- назначение разработки;
- требования к программе;
- требования к программной документации;
- технико-экономические показатели;
- стадии и этапы разработки;
- порядок контроля и приемки;
- приложения.

В зависимости от особенностей разрабатываемого ПО стандарт допускает уточнение содержания разделов, введение новых разделов или их объединение.

В разделе «Введение» указывается наименование, краткая характеристика области применения ПО.

В разделе «Основания для разработки» указывается:

- документ (документы), на основании которых ведется разработка;
- организация, утвердившая документ, и дата утверждения;
- наименование (условное обозначение) темы разработки.

В разделе «Назначение разработки» должно быть указано функциональное и эксплуатационное назначение ПО. Например, UML – как универсальный язык моделирования. Может использоваться и для постановки технического задания.

Внешние и внутренние языки спецификации

В процессе разработки ПО появляются следующие документы:

- соглашение о требованиях;
- внешняя спецификация;
- внутренняя спецификация.

Документ «Соглашение о требованиях» должен содержать первое письменное соглашение между заказчиком и разработчиком о том, что будет сделано, и что не будет делаться при разработке и выпуске программного обеспечения.

В отличие от него спецификация предполагает наличие более точных и исчерпывающих формулировок и определений. При этом, первые два документа содержат информацию о том, что представляет собой ПО; а третий должен объяснять, как ПО устроено и как достигаются установленные для него цели и требования. Все документы имеют схожую структуру для облегчения контроля над проектом, а также для обеспечения прослеживаемости всех технических решений от требований до их реализации. По мере продвижения проекта разделы документа либо просто копируются в соответствующие разделы следующего создаваемого документа, либо расширяются описаниями технических решений текущего этапа.

2. Порядок выполнения работы и содержание отчета

Порядок выполнения работы:

1. Технологический процесс разработки программного обеспечения.
2. Стадии разработки программ и программной документации.
3. Сопровождаемая документация.
4. Основные требования к содержанию документации.
5. Правила написания технического задания к разрабатываемым программным продуктам.
6. Техническое задание и требования к его содержанию.
7. Эскизный и технический проекты.
8) Рабочий проект. Внедрение. Понятие о ЕСПД.

Содержание отчета:
1) цель работы;
2) задание на лабораторную работу для своего варианта;
3) алгоритм решаемого задания с необходимыми пояснениями;
4) выводы по работе.

3. Контрольные вопросы

1. Технологический процесс разработки программного обеспечения.
2. Требования к структуре внешней спецификации.

Лабораторная работа № 15
«Технико-экономическое обоснование проектов программных средств»

Цель работы: знать технико-экономическое обоснование проектов программных средств.

1. Краткие теоретические сведения

Выбор и формирование требований к функциональной пригодности ПС — наиболее ответственная, стратегическая задача начальных этапов технико-экономического обоснования проекта программного продукта, системного проектирования и всего последующего развития его жизненного цикла.

Жизненный цикл ПС можно разделить на две части, существенно различающиеся экономическими особенностями процессов и ресурсов, характеристиками и влияющими на них факторами.

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

Максимум трудоемкости и числа специалистов приходится на этапы программирования и тестирования компонентов, когда привлекается основная масса программистов-кодировщиков для разработки компонентов ПС. При активном использовании и совершенствовании технологий системного анализа и проектирования происходит перераспределение всех видов затрат в сторону увеличения трудоемкости начальных этапов разработки. Это дает значительное снижение использования совокупных ресурсов для всего проекта.

Менее изучены распределения необходимых ресурсов по этапам работ, с учетом реализации требуемых конкретных характеристик качества ПС. Опубликованные данные и зависимости для различных классов ПС позволяют прогнозировать совокупные затраты и другие основные технико-экономические показатели (ТЭП), планы и графики работ вновь создаваемых проектов программных продуктов.

Вторая часть ЖЦ, отражающая эксплуатацию, сопровождение, модификацию, управление конфигурацией и перенос ПС на иные платформы, в меньшей степени зависит по величине требуемых ресурсов от функциональных характеристик объекта и среды разработки.

Номенклатура работ на этих этапах более или менее определенная, но их трудоемкость и длительность могут сильно варьироваться, в зависимости от массовости и других внешних факторов распространения и применения конкретных версий программного продукта. Успех ПС у пользователей и на рынке, а также будущий процесс развития версий трудно предсказать, и он не связан напрямую с экономическими параметрами процессов разработки ПС. Определяющими становятся потребительские характеристики продукта, а их экономические особенности с позиции разработчиков и вложенные ресурсы на очередную версию отходят на второй план.

Вследствие этого в широких пределах могут изменяться трудоемкость и число специалистов, необходимое для поддержки этих этапов ЖЦ. Это затрудняет статистическое обобщение ТЭП различных проектов и прогнозирование на их основе аналогичных характеристик новой разработки.

Поэтому планы на этих этапах имеют характер общих взаимосвязей содержания работ, которые требуют распределения во времени, индивидуально для каждого проекта. В результате прогнозирование и планирование трудоемкости и длительности этапов приходится производить итерационно на базе накопления опыта и анализа развития конкретных версий ПС, а также от их успеха на рынке.

Для прогнозирования и планирования любых процессов или характеристик объектов (в частности ПС) используются исходные данные двух типов:

1) функции и номенклатура характеристик самого прогнозируемого объекта или процесса, для которого необходимо спланировать жизненный цикл;

2) характеристики прототипов и пилотных проектов, в некоторой степени подобных планируемому объекту, о которых известны реализованные планы и необходимые экономические характеристики уже завершенных аналогичных процессов или объектов.

Совместная, корректная обработка исходных данных этих двух типов позволяет при проектировании оценивать и получать новые, прогнозируемые характеристики процессов, планов и экономических показателей создания ПС.

Исходные данные первого типа отражают характеристики конкретного создаваемого объекта или процесса, доступные методы и инструментальные средства автоматизации труда при их создании. Эти данные последовательно детализируются и уточняются в процессе проектирования и дальнейшего ЖЦ ПС, что, в частности, позволяет уточнять выбор компонентов аналогичных объектов и их характеристик для исходных данных второго типа.

2. Порядок выполнения работы и содержание отчета

Порядок выполнения работы:

1. Стоимость программных средств.
2. Факторы, влияющие на стоимость программных средств.
3. Методики оценки трудоемкости разработки программного продукта.

4. Особенности продаж программных продуктов.
5. Обновление версии программных средств.
6. Способы прогнозирования рынка программного обеспечения.

Содержание отчета:

1) цель работы;
2) задание на лабораторную работу для своего варианта;
3) алгоритм решаемого задания с необходимыми пояснениями;
4) выводы по работе.

3. Контрольные вопросы

1. Стоимость программных средств.
2. Способы прогнозирования рынка программного обеспечения.