МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Рубцовский индустриальный институт (филиал)
федерального государственного бюджетного образовательного
учреждения высшего образования
«Алтайский государственный технический университет им. И.И. Ползунова»
(РИИ АлтГТУ)

Кафедра «Прикладная математика»

Н.А. ЛАРИНА

ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ

Учебное пособие
для бакалавров направления
«Информатика и вычислительная техника»

Рекомендовано Рубцовским индустриальным институтом (филиалом) ФГБОУ ВО «Алтайский государственный технический университет им. И.И. Ползунова» в качестве учебного пособия для студентов, обучающихся по направлению подготовки «Информатика и вычислительная техника»

Рубцовск 2016

УДК 004.424

Ларина Н.А. Технологии программирования: Учебное пособие для бакалавров направления «Информатика и вычислительная техника» / Рубцовский индустриальный институт. – Рубцовск, 2016. – 51 с.

В пособие включены теоретический материал и практические задания в виде вопросов для самостоятельной работы студентов, примерные темы и методика выполнения курсовых работ, рабочая программа по дисциплине.

В теоретической части пособия излагаются классификация технологий программирования, методологии и раскрываются технологии программирования. В пособии отражены сведения о технологии и терминологии, применяемой для разработки автоматизированных систем обработки информации, с учётом структуры и архитектуры ИС.

Для самостоятельной работы после каждой темы, предлагается ответить на ряд вопросов, с целью самопроверки усвоения знаний.

Дисциплина изучается в рамках цикла дисциплин по выбору для обучающихся - бакалавров направления ««Информатика и вычислительная техника»» и носит обобщающий и углубляющий ранее изученный материал характер.

Рассмотрено и одобрено
на заседании НМС
Рубцовского индустриального
института.
Протокол № 4 от 08.06.16 г.

Рецензент: к.ф-м.н., заведующая кафедрой ВМФиХ Г.А. Обухова

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Технология программирования - это по определению одна из областей инженерной науки, и поэтому она несёт такую же социальную ответственность, как и другие науки.

С начала развития компьютерных технологий работу по созданию программных продуктов относили к разработке, для которой в основном требуются навыки программирования, а не знания инженерной науки. Аккредитационный совет по инженерной науке и технологии (ABET) определяет профессию инженера следующим образом: «Профессия, в которой математические и естественно - научные знания, полученные исследованиями, опытом и практикой, мудро применяются для разработки путей экономного использования природных ресурсов и сил на пользу человечеству».

Много было вложено труда в развитие инженерной науки ещё до рождения первого программного продукта. Сейчас процесс разработки программного продукта начинает требовать от своих разработчиков такой же высоты научных знаний, как это необходимо в других областях инженерной науки - физике, электронике, механике или строительстве.

Технология программирования изучает технологические процессы и порядок их прохождения - стадии (с использованием знаний, методов и средств). Знания, методы и средства могут использоваться в разных процессах и, следовательно, технологиях.

Чтобы разобраться в существующих технологиях программирования и определить основные тенденции их развития, целесообразно рассматривать эти технологии в историческом контексте, выделяя основные этапы развития программирования как науки.

Первый этап - «стихийное» программирование.

Он охватывает период от момента появления первых вычислительных машин до середины 60-х годов XX в. В этот период практически отсутствовали сформулированные технологии и программирование фактически было искусством. Первые программы имели простейшую структуру: собственно программа и обрабатываемые ею данные. Программы писали в машинных кодах. С появлением ассемблеров стало возможно использование вместо двоичных или 16-ричных кодов символических имён данных и мнемоники кодов операций. Программы стали более понятными для других.

В начале 60-х годов XX в. разразился «кризис программирования», который был вызван несовершенством технологии программирования.

Анализ причин позволил сформулировать новый подход к программированию.

Второй этап - структурный подход к программированию (60-70-е годы XX в.).

Данный подход представляет собой совокупность рекомендуемых технологических приёмов, охватывающих выполнение всех этапов разработки программного обеспечения. Поддержка принципов структурного

программирования была заложена в основу так называемых процедурных языков программирования (Pascal, C и др.). В это время начала развиваться технология модульного программирования. Эту технологию поддерживают современные версии языков Pascal, C (C++).

При увеличении размера программы возрастает сложность межмодульных интерфейсов. Для разработки программного обеспечения большого объёма было предложено использовать объектный подход.

Третий этап - объектный подход к программированию (с середины 80-х до конца 90-х годов XX в.).

Объектно-ориентированное программирование определяется как технология разработки сложного программного обеспечения, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определённого типа (класса), а классы образуют иерархию с наследованием свойств. Взаимодействие программных объектов в такой системе осуществляется путём передачи сообщений (языки: Simula, Smalltalk, Pascal, C++, Modula, Java).

Объектный подход предлагает новые способы организации программ, основанные на механизмах наследования, полиморфизма, композиции, наполнения. Но конкретная реализация в объектно-ориентированных языках программирования, таких как Pascal и C++, имеет существенные недостатки в части связи модулей.

Связи модулей нельзя разорвать, но можно попробовать стандартизовать их взаимодействие.

Четвёртый этап - компонентный подход и CASE - технологии (с середины 90-х годов XX в. до настоящего времени).

Компонентный подход предполагает построение программного продукта из отдельных компонентов физически отдельно существующих частей программного обеспечения, которые взаимодействуют между собой через стандартные двоичные интерфейсы. Это позволяет программистам разрабатывать продукты, хотя бы частично состоящие из повторно использованных частей.

Компонентный подход лежит в основе технологий, разработанных на базе COM (Component Object Model - компонентная модель объектов), и технологии разработки распределённых приложений CORBA (Common Object Request Broker Architecture - общая архитектура с посредником обработки запросов объектов).

Отличительной особенностью настоящего этапа развития технологии программирования, кроме изменения подхода, является разработка и внедрение автоматизированных технологий создания и сопровождения программного обеспечения, которые были названы CASE -технологиями (Computer - Aided Software / System Engineering - разработка программного обеспечения программных систем с использованием компьютерной поддержки).

Основные понятия профессионального программирования:

Понятие – множество ситуаций на входе кибернетической системы. Владеть понятием – значит уметь его распознать, т.е. уметь определять, принадлежит ли любая данная ситуация к множеству, характеризующему понятие, или не принадлежит.

Определение – логическая операция, заключающаяся в придании точного смысла языковому выражению. Цель определения – уточнить содержание используемых понятий.

Классификация – система, согласно которой что-либо распределяется по группам, разрядам, признакам, принципам, классам. Навести порядок в хаосе очень важно для того, чтобы лучше понимать исследуемую предметную область.

Многие определения и классификации субъективны, но, чтобы вести обсуждение какого-либо предмета, следует определить его и указать его место в картине мира.

Программа – завершённый продукт, пригодный для запуска своим автором в системе, на которой он был разработан.

Программный продукт – программа, которую любой человек может запускать, тестировать, исправлять и развивать. Такая программа должна быть написана в общем стиле, тщательно протестирована и снабжена подробной документацией.

Программный комплекс – набор взаимодействующих программ, согласованных функций и форматов, с точно определённым интерфейсом, и вместе составляющих полное средство для решения больших задач.

Программное средство – это программа или логически связанная совокупность программ на носителях данных, снабжённая программной документацией.

Алгоритм – точное и конечное описание того или иного общего метода, основанного на применении исполнимых элементарных тактов обработки.

Компьютер – *вычислитель*, он не понимает программу, а исполняет её. Наиболее естественный способ указать компьютеру ход исполнения программы – записать её в виде алгоритма. Современное слово «алгоритм» во многом аналогично таким понятиям, как рецепт, процесс, метод, способ. Алгоритм имеет пять важных свойств:

1. *Конечность*. Алгоритм всегда должен заканчиваться после выполнения конечного числа шагов.
2. *Определённость*. Каждый шаг алгоритма должен быть определён.
3. *Наличие входных данных*. Алгоритм имеет некоторое число входных данных, заданных до начала его работы или определяемых динамически во время его выполнения.
4. *Наличие выходных данных*. Алгоритм имеет одно или несколько выходных данных, имеющих определённую связь с входными данными.
5. *Эффективность*. Алгоритм обычно считается эффективным, если его операторы достаточно просты для того, чтобы их можно было точно

выполнить в течение конечного промежутка времени с помощью карандаша и бумаги.

С алгоритмом связаны области исследований:

• *Анализ алгоритмов*. Предмет этой области состоит в том, чтобы для заданного алгоритма определить рабочие характеристики. Например, часто желательно, чтобы алгоритм был быстрым.

• *Теория алгоритмов*. В этой области рассматриваются вопросы осуществления или неосуществления эффективных алгоритмов вычисления определённых величин.

• *Построение алгоритмов. В этой области рассматриваются стандартные приёмы и методы, используемые при написании алгоритмов.*

Модель – это упрощённое представление реальности. Моделью является, например, чертёж системы. Основное свойство модели в том, что она – семантически замкнутая абстракция системы. Она строится для того, чтобы лучше понять разработанную систему, визуализировать её, определить структуру или поведение. Сложные системы моделировать просто необходимо, т.к. иначе их невозможно воспринимать как единое целое.

Моделирование – метод исследования систем на основе переноса изучаемых свойств системы на объекты другой природы. Это один из методов исследования окружающей действительности. Инженерная методика изготовления моделей является устоявшейся и повсеместно принятой. Она позволяет решить несколько важных задач:

- визуализировать систему;
- определить структуру системы и её поведение;
- документировать принимаемые решения.

Моделирование – это попытка решить проблему сложности. Существует четыре принципа моделирования:

1. выбор модели оказывает определяющее влияние на подход к решению проблемы и на то, как будет выглядеть это решение.
2. Каждая модель может быть воплощена с разной степенью абстракции.
3. Лучшие модели – те, которые ближе к реальности.
4. Следует использовать совокупность нескольких моделей.

В общем случае выделяют четыре типа моделей:

•*Математические модели*, представляющие собой систему математических уравнений, адекватно описывающих изучаемое явление или объект.

•*Физические модели*, основанные на использовании эффекта масштаба в случае возможности пропорционального применения всего комплекса изучаемых средств.

•*Ситуационные модели*, представляющие собой описание ситуаций, в которых предстоит действовать изучаемому объекту.

•Электрические модели, позволяющие построить электрическую цепь, эквивалентную любому дифференциальному уравнению.

Жизненный цикл программного обеспечения – весь период его разработки и эксплуатации, начиная с момента возникновения замысла и заканчивая прекращением всех видов его использования. Существует простейшее представление жизненного цикла, который включает стадии:

- анализ;
- проектирование;
- программирование;
- тестирование и отладка;
- эксплуатация.

Жизненный цикл программного обеспечения тесно связан с технологиями программирования, о которых и пойдёт дальнейшее повествование.

Структура данных программ

Под структурой данных программ в общем случае понимают множество элементов данных, множество связей между ними, а также характер их организованности.

Под организованностью данных понимается продуманное их устройство. Простейшие структуры данных, реализуемые языками программирования, называют стандартными типами данных. Многие языки программирования позволяют на основе стандартных типов строить типы данных, определённые программистом (пользователем).

Типы данных – это не значения, а множество значений. Данные характеризуются набором операций, которые можно выполнять над этими данными, множеством значений.

Структуры данных и алгоритмы служат основой построения программ. Для компьютера все типы данных сводятся в конечном счёте к последовательности битов (байтов) и мнемоника имён им безразлична.

Для решения одной и той же задачи, но с различающимися структурами данных обычно требуются разные алгоритмы. Без предшествующей спецификации структуры данных невозможно приступить к составлению алгоритмов.

Понятие «*физическая структура данных*» отражает способ физического представления данных в памяти машины и называется ещё структурой хранения, внутренней структурой, структурой памяти или дампом.

Рассмотрение структуры данных без учёта её представления в машинной памяти называют абстрактной, или *логической, структурой данных*. В общем случае между логической и соответствующей ей физической структурами имеется различие, вследствие которого существуют правила отображения логической структуры на физическую структуру.

Структуры данных, применяемые в алгоритмах, могут быть очень сложными, поэтому правильный выбор представления данных часто служит ключом к удачному программированию и может в большей степени сказываться на производительности программы, чем детали используемого алгоритма.

Знание структур данных позволяет организовать их хранение и обработку максимально эффективным образом – с точки зрения минимизации затрат как памяти, так и процессорного времени.

Важное преимущество, которое обеспечивается структурным подходом к данным, является возможность структурирования сложной программы для достижения её понятности человеку, что сокращает количество ошибок при первоначальном кодировании и необходимо при последующем сопровождении.

Другим продуктивным технологическим приёмом, связанным со структуризацией данных, является инкапсуляция, смысл которой заключается в том, что сконструированный новый тип данных оформляется таким образом, что его внутренняя структура становится недоступной для программиста – пользователя этого типа данных. Программист, использующий такой тип данных в своей программе, может оперировать данными только через вызовы процедур.

Многовариантность реализации структур требует проектного решения о способе их реализации. При принятии проектного решения применяют такие критерии, как объём занимаемой памяти, возможный набор операций, скорость выполнения операций.

Структуры данных и алгоритмы служат основой построения программ. Над структурами данных можно выполнять пять операций: создание, уничтожение, выбор (доступ), обновление, копирование.

Важный признак структуры данных – характер упорядоченности её элементов. Существует множество способов упорядочения информации, среди которых имеются и общие, наиболее часто встречаемые и известные большинству программистов.

Любой информации (данным) присущи определённые свойства, позволяющие её правильно интерпретировать - это:

- достоверность,
- полнота,
- ценность,
- актуальность,
- ясность,
- понятность.

Важным техническим средством реализации информации является компьютер. Поэтому внедрение его в системы технического управления механизмами, состоянием здоровья человека и административного управления является закономерным. Организация информационных процессов в управлении осуществляется во многих направлениях, поэтому так важно развивать информационную культуру. Информационная культура – это соблюдение правил образования, обработки, использования и защиты информации. Соблюдение информационной культуры проявляется в выполнении определённых требований.

Для обработки информации на компьютерах необходимо программное обеспечение, которое в настоящее время рассматривается как продукт или

услуга, а поэтому и производиться оно должно по производственным правилам и нормам. Эти требования к программным продуктам привели к описаниям процессов их производства – технологиям.

Технология программирования изучает технологические процессы и порядок их прохождения – стадии. Знания, методы и средства могут использоваться в разных процессах и, следовательно, технологиях.

Технологии принято характеризовать двумя измерениями: вертикальном (процессы) и горизонтальном (стадии).

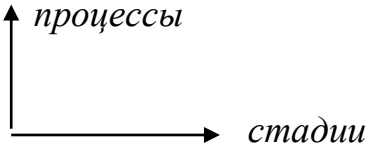

Рис. 1. Характеристика технологии

Процесс – совокупность взаимосвязанных действий, преобразующих некоторые входные данные в выходные. Процессы состоят из ряда действий, каждое из которых, в свою очередь, состоит из набора задач. Вертикальное измерение отражает статические аспекты процессов и оперирует такими понятиями, как рабочие процессы, действия, задачи, результаты деятельности исполнителя.

Стадия – это часть действий разработки ПО, ограниченная некоторыми временными рамками и завершающаяся выпуском конкретного продукта, определяемого заданными для данной стадии требованиями. Стадии состоят из этапов, которые обычно имеют итерационный характер. Стадии, объединённые в более крупные временные рамки, называют *фазами*.

Технологический подход определяется спецификой комбинации стадий и процессов, ориентированной на разные классы программного обеспечения и на особенности коллектива разработчиков.

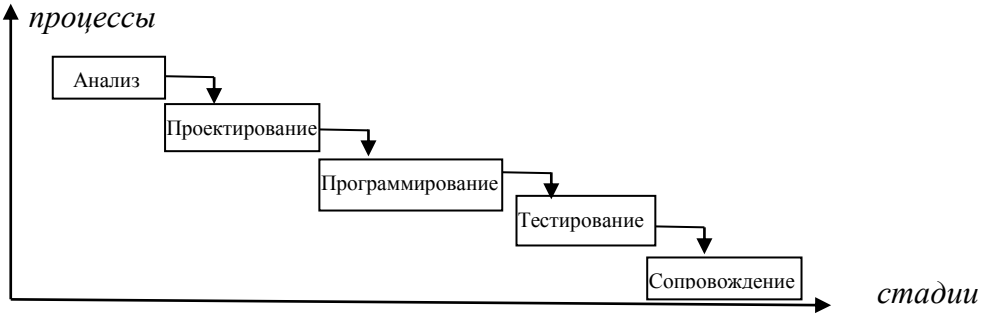

Рис. 2. Схема технологического подхода

История и эволюция технологии программирования

В рассматриваемых процессах можно выделить три этапа:

I. Осмысление опыта разработки больших систем. Понимание того, что важно не только, на каком языке программирования разрабатывается программа, но и как это делается. 1968 – 1979гг.

II. Разработка новых технологических подходов (начало 70-х годов XX века – настоящее время).

III. Принятие стандартов на состав процессов жизненного цикла ПО (середина 80-х годов XX века – настоящее время).

Рассмотрим классификацию технологических подходов, процессов и стадий, сложившуюся к настоящему времени.

КЛАССИФИКАЦИЯ ТЕХНОЛОГИЧЕСКИХ ПОДХОДОВ, ПРОЦЕССОВ, СТАДИЙ

Классификация технологических подходов

Выделим основные группы технологических подходов и перечислим подходы для каждой из них.

I. Подходы со слабой формализацией.

Применимы для очень маленьких проектов демонстрационного типа. К данной группе технологических подходов можно отнести подход «кодирование и исправление».

II. Строгие (классические, жёсткие, предсказуемые) подходы.

Для средних, крупных и гигантских проектов с фиксированным объёмом работ. К данной группе подходов относятся:

1. Каскадные технологические подходы:
   a. Классический каскадный подход;
   b. Каскадно - возвратный подход;
   c. Каскадно - итерационный подход;
   d. Каскадный подход с перекрывающимися процессами;
   e. Каскадный подход с подпроцессами;
   f. Стерильная модель.
2. Каркасные подходы.

Например, «Рациональный унифицированный процесс».

3. Генетические подходы:
   a. Синтезирующее программирование;
   b. Сборочное (расширяемое) программирование;
   c. Конкретизирующее программирование.
4. Подходы на основе формальных преобразований:
   a. Технология стерильного цеха;
   b. Формальные генетические подходы.

III. Гибкие (адаптивные, лёгкие) подходы:

Рекомендуется применять для небольших или средних проектов, в случае неясных или изменяющихся требований к системе. К данной группе относятся:

1. Ранние технологические подходы быстрой разработки:
   a. Эволюционное прототипирование;
   b. Итеративная разработка;
   c. Постадийная разработка.
2. Адаптивные подходы:
   a. Экстремальное программирование;

b. Адаптивная разработка.

3. Подходы исследовательского программирования:

a. Компьютерный дарвинизм.

Классификация технологических процессов

I. Набор классических процессов, включающий основные процессы, сложившиеся исторически в результате практического опыта разработки ПО.

II. Стандартный набор технологических процессов – основанный на стандарте ISO 12207: 1995.

Процессы классического набора фактически являются подмножеством стандартного, выступая в нём как процессы или действия процессов.

В классическом наборе девять основных технологических процессов:

1 Возникновение и исследование идеи;
2 Управление;
3 Анализ требований;
4 Проектирование;
5 Программирование;
6 Тестирование и отладка;
7 Ввод в действие;
8 Эксплуатация и сопровождение;
9 Завершение эксплуатации.

Процессы жизненного цикла (ЖЦ), определяемые международным стандартом ISO 12207:[ISO/IEC 12207: 95], делятся на три группы:

1 Основные процессы:
- Приобретение;
- Поставка;
- Разработка;
- Эксплуатация;
- Сопровождение.

2 Вспомогательные процессы:
- Документирование;
- Управление конфигурацией;
- Обеспечение качества;
- Верификация;
- Аттестация;
- Совместная оценка;
- Аудит;
- Разрешение проблем.

3 Организационные процессы:
- Управление;
- Создание инфраструктуры;
- Усовершенствование;
- Обучение.

Классификация технологических стадий

Технологические стадии выделяются исходя из соображений разумного рационального планирования и организации работ. Существует два основных варианта формирования временных промежутков, поддерживаемых технологическими подходами.

В первом варианте формируются фазы, отражающие крупные временные этапы. Например, начальная фаза, середина, кризис, окончание.

Во втором варианте – определяются стадии, отражающие названия классических процессов (или их подмножества, или их надмножества), большая часть времени которых проходит в данной стадии. К примеру, отобразим на рисунке 3.

ПРОБЛЕМЫ И ПЕРСПЕКТИВЫ РАЗВИТИЯ ТЕХНОЛОГИЙ ПРОГРАММИРОВАНИЯ

Развитие и совершенствование технологических подходов ведётся по двум направлениям, которые достаточно сильно отличаются друг от друга своими задачами.

I. *Максимизация качества*. Задачи этого направления: надёжность, чёткость и формализация, ориентированы на военные разработки и системы реального времени. Направление поддерживается строгими технологическими подходами, обеспечивающими предсказуемость.

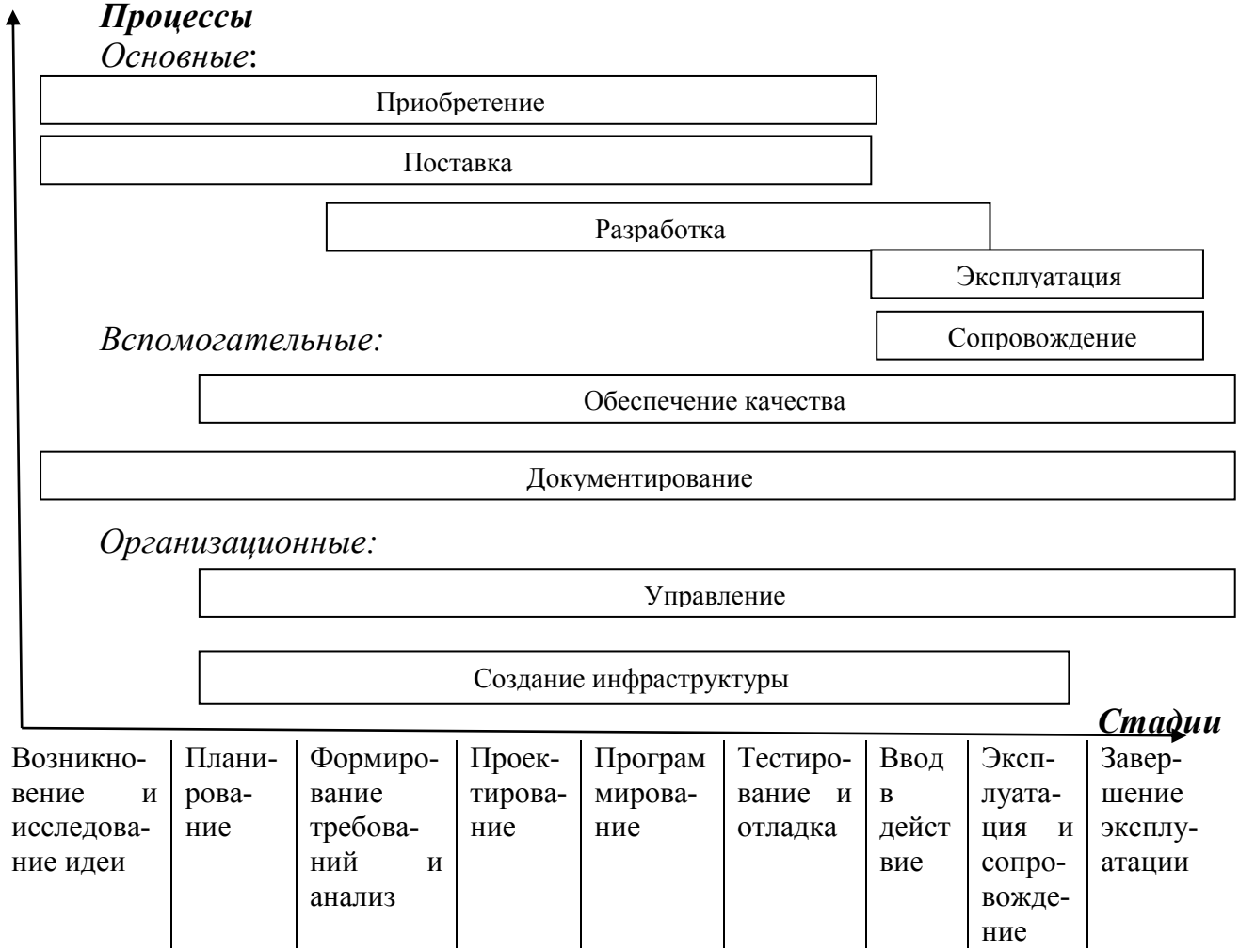

Рис. 3. Схема технологических стадий

II. *Максимизация скорости разработки.* Это направление больше обращено к искусству, импровизации и поиску. Гибкие и адаптивные технологические подходы поддерживают этот путь разработки.

Особенность разработки определяется появлением новых классов систем. С точки зрения технологий, задача разработки интернет-приложений является очень специфичной. Базироваться на классических технологических подходах ей не позволяют особенности:

- задача, как правило, не будет чётко определена и специфицирована;
- заказчики не смогут точно сформулировать свои требования;
- по окончании работы над системой она устареет;
- система должна правильно взаимодействовать с независимо разработанными программами.

Ещё одно важное и *перспективное направление*, связанное с технологией программирования, – *исследование человеческих и социальных факторов в информатике и программировании.*

Предполагается, что основные усилия в исследовании разработки ПО будут направлены на процесс проектирования. Это объясняется тем, что в больших проектах основной объём работ приходится на проектирование, которое должны выполнять талантливые творческие специалисты.

Информационные продукты и услуги

В Федеральном законе «Об информации, информатизации и защите информации» введено понятие информационных ресурсов.

Определение: «Информационные ресурсы (ИР) – это отдельные документы, массивы документов, документы и массивы документов в информационных системах (библиотеках, архивных фондах, банках данных и других информационных системах)».

Информационные ресурсы следует понимать как знания, которые материализовались в виде документов, баз данных, баз знаний, программ, алгоритмов и т.д., поэтому их следует рассматривать как стратегические ресурсы общества. ИР являются базой для создания информационных продуктов.

Определение: «Информационный продукт (ИП) – совокупность данных, сформированная производителем для распространения в вещественной или невещественной формах. ИП распространяется с помощью информационных услуг.

Определение: **«***Информационная услуга* (ИУ) – предоставление в распоряжение пользователя ИП.

Информационные услуги не сводятся к компьютерным (например, библиотека), но всё больше и больше к этому приближаются. В настоящее время оказание информационных услуг практически невозможно без создания и ведения баз данных (БД).

Виды информационных услуг классифицируются в зависимости от вида ИП следующим образом:

- выпуск информационных изданий;
- ретроспективный поиск информации – целенаправленный поиск по заявке и пересылка результатов;
- предоставление первоисточника или копии;

- традиционные услуги научно – технической информации:
    - обзоры;
    - переводы;
- дистанционный доступ;
    - непосредственный доступ;
    - косвенный доступ (бюллетени, справочная служба);
    - downloading: часть центральной БД (результат отбора по критериям поиска) загружается на ПК пользователя для дальнейшей работы;
    - регулярный поиск;
- оказание информационных услуг:
    - связь;
    - программное обеспечение;
    - создание информационных систем;
    - обработка данных на вычислительном центре и т.д.

Рынок информационных продуктов и информационных услуг.

Определение: «Рынок ИП и ИУ – система экономических, правовых, организационных отношений по торговле продуктами интеллектуального труда на коммерческой основе. На этом рынке действуют: поставщики ИП и ИУ; потребители ИП и ИУ.

Структура рынка информационных продуктов и информационных услуг

В структуре рынка ИП и ИУ можно выделить следующие компоненты:

- технологическая составляющая;
- нормативно – правовая составляющая;
- информационная составляющая;
- организационная составляющая.

В результате развития рынка ИП и ИУ формируется инфраструктура информационного рынка.

Определение: «Инфраструктура информационного рынка» – совокупность секторов, каждый из которых объединяет группы, предлагающие однородные информационные продукты и услуги.

Подходы к определению инфраструктуры рынка ИП и ИУ различны.

Пример на рисунке 4.

Рис. 4. Схема инфраструктуры рынка

МЕТОДОЛОГИЯ И ТЕХНОЛОГИЯ РАЗРАБОТКИ ИНФОРМАЦИОННЫХ СИСТЕМ

Методология создания информационных систем заключается в организации процесса построения информационной системы и обеспечении управления этим процессом для того, чтобы гарантировать выполнение требований как к самой системе, так и к характеристикам процесса разработки.

Основные задачи методологии создания ИС:

- обеспечить создание ИС, отвечающей целям и задачам предприятия;
- гарантировать создание системы с заданными параметрами, в срок за оговоренный бюджет;
- упростить сопровождение, модификацию и расширение ИС;
- обеспечить открытость, переносимость и масштабируемость системы;
- дать возможность использования в разрабатываемой системе ранее применяемых средств информационных технологий.

Методологии, технологии и инструментальные средства проектирования (CASE- средства) составляют основу проекта любой информационной системы. Методология реализуется через конкретные технологии и поддерживающие документы, методики и инструментальные средства, которые обеспечивают набор процессов жизненного цикла ИС.

Основное содержание технологии проектирования составляют технологические инструкции, включающие описания последовательности технологических операций, условий, в зависимости от которых выполняется та или иная операция и описаний самих операций.

Технология проектирования состоит из:

- заданной последовательности выполнения технологических операций проектирования;
- критериев и правил для оценки результатов логических операций;
- графических и текстовых средств (нотаций) для описания системы.

Каждая технологическая операция должна обеспечиваться материальными и информационными ресурсами:

- исходными или полученными от предыдущих операций данными;
- методическими материалами, инструкциями, нормативами и стандартами;
- программными и техническими средствами;
- исполнителями.

Общие требования к технологии проектирования, разработки и сопровождения ИС:

- поддерживать полный жизненный цикл ИС;
- гарантировать достижение целей разработки систем с необходимым качеством и в установленные сроки;
- обеспечивать возможность разделения крупных проектов на подпроекты.
- технология должна обеспечивать возможность ведения работ по проектированию отдельных подсистем небольшими группами (3-7 человек);

- обеспечивать минимальное время получения работоспособной системы;
- предусматривать возможность управления конфигурацией проекта, ведения его версий, возможность автоматического выпуска проектной документации для каждой версии проекта;
- обеспечивать независимость выполняемых проектных решений от средств реализации системы.

Декомпозиция позволяет повысить эффективность работ. Подсистемы должны быть слабо связаны по данным.

Методология RAD – Rapid Application Development

Методология разработки ИС, основанная на использовании средств быстрой разработки приложений, приобрела название методологии быстрой разработки приложений - RAD. Данная методология охватывает все этапы жизненного цикла современных информационных систем.

RAD – это комплекс специальных средств быстрой разработки, прикладных информационных систем.

Под RAD – методологией разработки ИС понимают разработки основанные на правилах:

- небольшая команда программистов (2-10 чел.),
- тщательно проработанный сетевой график работ на 2-6 месяцев,
- итерационная модель разработки с тесным взаимодействием с заказчиком.

Группа разработчиков должна состоять из профессионалов, имеющих опыт в анализе, проектировании, программировании и отладке ПО.

Принципы методологии RAD – сводятся к следующему:

- используется итерационная (спиральная) модель разработки;
- полное завершение работ на каждом этапе жизненного цикла обязательно;
- в процессе разработки ИС необходимо взаимодействие с заказчиком;
- необходимо применение CASE – средств и средств быстрой разработки приложений;
- применение средств управления конфигурацией для внесения изменений в проект и сопровождения готовой системы;
- необходимо использование прототипов;
- тестирование и развитие проекта осуществляются параллельно;
- разработка ведётся немногочисленной и хорошо управляемой группой профессионалов;
- необходимы грамотное руководство разработкой системы, чёткое планирование и контроль выполнения работ.

Средства RAD – дали возможность использовать иную технологию разработки приложений: ИС формируются как некие действующие модели (прототипы), чьи функции согласовываются с пользователем.

Возможность такого подхода в значительной степени является результатом применения принципов объектно-ориентированного проектирования, что

позволяет преодолеть колоссальный разрыв между реальным миром (предметной областью проблемы) и имитирующей средой.

Использование объектно-ориентированных методов позволяет создать описание (модель) предметной области в виде совокупности объектов – сущностей, объединяющих данные и методы обработки этих данных (процедуры). Каждый объект обладает своим собственным поведением и моделирует некоторый объект реального мира. С этой точки зрения объект является вполне осязаемой вещью, которая демонстрирует определённое поведение.

Применение принципов объектно-ориентированного программирования позволило создать принципиально новые средства проектирования приложений, называемые средствами визуального программирования. Визуальные инструменты позволяют создавать сложные графические интерфейсы пользователя.

Визуальные средства разработки оперируют со стандартными интерфейсными объектами – окнами, списками, текстами, которые легко связать с данными из базы данных и отобразить на экране монитора.

Другая группа объектов представляет собой стандартные элементы управления – кнопки, переключатели, флажки, меню и т.п., с помощью которых осуществляется управление отображаемыми данными.

Существует много различных визуальных средств разработки приложений. Их можно разделить на две группы: универсальные и специализированные.

Среди универсальных систем визуального программирования: Borland Delphi, Visual Basic и др. Универсальными их называют потому, что они не ориентированы на разработку только приложений БД – с их помощью могут быть разработаны приложения почти любого типа, в том числе и информационные приложения. Они могут взаимодействовать почти со всеми СУБД.

Специализированные средства разработки ориентированы только на приложения БД с привязкой к определённым СУБД: Power Builder и Visual FoxPro.

Программист получил непрерывную обратную связь с пользователями, которые могут наблюдать и корректировать результаты и свои требования. Визуальные инструменты дают возможность максимально сблизить этапы разработки ИС: анализ исходных условий, проектирование системы, разработка прототипов и окончательное формирование приложений. На каждом этапе разработчики оперируют визуальными объектами.

Логика приложения, построенного с помощью RAD, относится к событийно ориентированной, т.е. каждый объект, входящий в состав приложения, может генерировать события и реагировать на события, генерируемые другими объектами.

Разработчик реализует логику приложения путём определения обработчика каждого события, т.о., управление объектами осуществляется с помощью событий.

При использовании методологии быстрой разработки приложений жизненный цикл ИС состоит из четырёх фаз:

- анализ и планирование требований;
- проектирование;
- построение;
- внедрение.

Несмотря на все свои достоинства, методология RAD не может претендовать на универсальное применение. Она наиболее эффективна при выполнении сравнительно небольших проектов, разрабатываемых для вполне определённого предприятия.

Смешанные методологии

Это методологии, которые включают объединение методов нескольких методологий. Чаще объединяются методологии функционального и логического программирования. Ведутся исследования в области объединения объектно-ориентированного и логического программирования.

Классы задач, поддерживаемых методологиями

Выигрыш от применения различных методологий можно оценивать:

- эффективностью ПО на современных ПК,
- общими затратами на разработку ПО.

Выделяются две ветви развития языков, поддерживающих методологии.

- Языки, ориентированные на скорость исполнения кода программы (обычно компилируемые).
- Языки, ориентированные на высокий уровень и простоту программирования (и компилируемые, и интерпретируемые).

Ядра методологий

Методология императивного программирования – подход, характеризующийся принципом последовательного изменения состояния вычислителя пошаговым образом. При этом управление изменениями полностью определено и полностью контролируемо.

Императивное программирование – это исторически первая поддерживаемая аппаратно методология программирования, ориентированная на классическую фон-неймановскую модель.

В данной методологии применяется метод изменения состояний, который заключается в последовательном изменении состояний. Метод поддерживается концепцией алгоритма.

Другой метод управления потоком исполнения, который заключается в пошаговом контролируемом управлении. Метод поддерживается концепцией потока исполнения.

Императивное программирование основано на описании последовательного изменения состояний вычислителя, т.е. состоянием значения всех ячеек памяти. Единственная структура данных – последовательность ячеек с линейно упорядоченными адресами.

В качестве математической модели императивное программирование использует машину Тьюринга-Поста – абстрактное вычислительное устройство.

Основные синтаксические понятия группы операторов:

1. Атомарные операторы (присваивание, переход, вызов подпрограммы и т.п.).
2. Структурные операторы (составной, выбора, цикла и т.д.).

Императивные языки программирования: Fortran (1954 г.), Algol (1960 г.), Pascal (1970 г.), C (1972 г.), ICON (1974 г.).

Императивное программирование наиболее пригодно для решения задач, в которых последовательное исполнение каких-либо команд является естественным. К примеру, управление аппаратными средствами. С ростом сложности задач императивные программы становятся всё менее и менее читаемыми.

Методология объектно-ориентированного программирования – подход, использующий объектную декомпозицию, при которой статические структуры системы описываются в терминах объектов и связей между ними, а поведение системы описывается в терминах обмена сообщениями между объектами.

Объектное мышление порождено моделированием и представлением данных, графическим пользовательским интерфейсом и системным программированием, связанным с понятием «процесс».

Объектно-ориентированное программирование основано на методах:

- объектно-ориентированной декомпозиции, который заключается в выделении объектов и связей между ними и поддерживается концепциями: инкапсуляции, наследования и полиморфизма;
- абстрактных типов данных, который поддерживается концепцией абстрагирования;
- пересылки сообщений – заключается в описании поведения системы в терминах обмена сообщениями между объектами и поддерживается концепцией сообщения.

Вычислительная модель чистого объектно-ориентированного программирования поддерживает явно только одну операцию – посылка объекту сообщения. Сообщения, возможно объекты, могут иметь параметры в виде объектов.

Объект имеет набор обработчиков сообщений (набор методов). Объекты состоят из полей – персональных переменных, значениями которых являются ссылки на другие объекты.

В синтаксисе чистых объектно-ориентированных языков всё может быть записано в форме посылки сообщений объектам. Объект называют экземпляром указанного класса. Класс в объектно-ориентированных языках описывает структуру и функциональные множества объектов с подобными характеристиками, атрибутами и поведением. Объект принадлежит к некоторому классу и обладает своим собственным внутренним состоянием. Методы – функциональные свойства, которые можно активизировать.

Объектно-ориентированные языки программирования содержат конструкции, позволяющие определять объекты, принадлежащие классам и обладающие свойствами инкапсуляции, наследования и полиморфизма. Эти языки можно выделить в три группы (табл. 1):

Данная методология является мощным средством для моделирования отношений между объектами практически в любой предметной области, особенно элементами графического интерфейса.

Методология функционального программирования – способ составления программ, в которых единственным действием является вызов функции, единственным способом деления программы на части – введение имени для функции и задание для этого имени выражения, вычисляющего значения функции, а единственным правилом композиции – оператор суперпозиции функции.

Таблица 1

Объектно-ориентированные языки программирования

	Чистые	Гибридные	Урезанные
1960			
	Simula (1962)		
1970			
	Smalltalk (1792)		
	Beta (1975)		
1980			
		C++ (1983)	
		Object Pascal (1984)	
	Self (1986)		
1990			
	Cecil (1992)		
			Java (1995)
2000			C# (2000)

Функциональная методология одна из старейших. Происхождением она связана с лямбда-исчислением, изобретённым ещё в начале 30-х годов XX века логиком Алонзо Черчем.

Функциональная методология ассоциируется с языком Lisp (Дж. Маккарти, конец 50-х годов XX века). Эта методология в основном используется теоретиками программирования и является средством лабораторных исследований искусственного интеллекта.

Методология использует методы:

- аппликативности – программа есть выражение, составленное из применения функций к аргументам, что поддерживается концепцией функции;
- рекурсивного поведения – заключается в самоповторяющемся поведении, возвращающемся к самому себе, что поддерживается концепцией рекурсии;
- настраиваемости - заключается в том, что можно легко порождать новые программные объекты по образцу, как значения соответствующих выражений.

Функциональное программирование – это одна из альтернатив императивному подходу. В функциональном программировании отсутствует

понятие времени. В качестве математической модели функциональное программирование использует лямбда-исчисление Черча.

При описании функционального программирования рассматривают так называемое «расширенное лямбда – исчисление» с соответствующей грамматикой.

Так как порядок вычисления подвыражений не имеет значения («состояния» у функциональной программы нет), функциональное программирование может быть естественным образом реализовано на платформах, поддерживающих параллелизм. «Потоковая модель» функционального программирования (Филд, Харрисон 1993г.) является естественным представлением функциональных программ в терминах систем взаимодействующих процессов.

Функциональные языки программирования – языки, в которых единственным действием является вызов функции, единственным способом деления программы на части является введение имени для функции и задание для этого имени выражения, вычисляющего значение функции, а единственным правилом композиции – оператор суперпозиции функции.

Основные функциональные языки:

1950
Lisp (1958)
1960
РЕФАЛ (1968)
1970
Scheme (1975)
FP (1977) ML (1978)
1980
Miranda (1985) Standard ML (1985)
1990 Haskell (1990, 1998)
2000

Функциональное программирование обычно применяется для решения задач, которые трудно сформулировать в терминах последовательных операций. К этой категории относятся практически все задачи, связанные с искусственным интеллектом: обработка естественного языка, экспертные консультирующие системы, проблемы зрительного восприятия и т.п.

Методология логического программирования – подход, согласно которому программа содержит описание проблемы в терминах фактов и логических формул, а решение проблемы система выполняет с помощью механизмов логического вывода.

Логическое программирование берёт начало в конце 60-х годов XX века. Корделл Грин предложил использовать резолюцию как основу логического программирования. Алан Колмероэ разработал язык логического программирования Prolog в 1971 году. В основе логических языков лежит теория хорновских дизъюнктов. Логическое программирование было

популярно в середине 80-х годов XX века, когда оно было положено в основу проекта разработки программного и аппаратного обеспечения вычислительных систем пятого поколения.

Методы логического программирования:

- единообразного применения механизма логического доказательства ко всей программе;
- унификации – механизм сопоставления с образом для создания и декомпозиции структур данных.

Логическое программирование – это программирование в терминах фактов и правил вывода.

В логике теории задаются при помощи аксиом и правил вывода. То же в базисном языке логического программирования Prolog. Аксиомы принято называть фактами, а правила вывода принято ограничить по форме до так называемых «дизъюнктов Хорна».

Логические языки программирования содержат конструкции, позволяющие выполнить описание проблемы в терминах фактов и логических формул, а решение проблемы выполняет система с помощью механизмов логического вывода. Язык Prolog является родоначальником семейства ему подобных языков, в котором можно выделить три ветви.

Таблица 2

Логические языки программирования

	Модификации языка	Функциональное направление	Параллельное направление
1950			
1960			
1970	Prolog (1971)		
1980	Prolog II (1980) IC-Prolog (1982) Mprolog (1983) T- Prolog (1983) LQF (1984)	LOGLISP (1982) LCA (1982) LEAF (1985)	 Concurrent Prolog (1983) PARLOG (1983) GHC (1986)
1990	Goedel (1992)	 Mercury (1993)	

Класс задач логического программирования практически совпадает с классом задач функционального программирования.

Методология программирования в ограничениях – подход, в котором в программе определяется тип данных решения, предметная область решения и ограничения на значение искомого решения.

Методология предлагает архитектуру, интегрирующую компонент ограничения и программный компонент. Компонент ограничений обеспечивает основные операции и состоит из системы выводов на фундаментальных

свойствах системы ограничений. Операции, окружающие компонент ограничений, реализуются программно-языковым компонентом.

Методология возникла в начале 80-х годов XX века как перспективная область исследований на пересечении символьных вычислений, искусственного интеллекта, исследования операций и интегральной арифметики. В ней применяется метод описательной модели вычислений, который заключается в том, что программа на языке программирования содержит описание понятий и задач. Метод поддерживается концепцией модели. Вычислительная модель программирования в ограничениях – это программирование в терминах постановок задач.

Языки программирования в ограничениях позволяют в программе определить тип данных решения, предметную область решения и ограничения на значение искомого решения. Эти языки получили наибольшую известность, как и многие другие, в 80-х годах XX века.

Таблица 3

Языки программирования в ограничениях

1960	Sketchpad (1963)		
1980	УТОПИСТ (1980) OPS5 (1987)	Thinglab (1980) Bertrand (1988)	IDEAL (1981)
1990	OPL (1998)		

Класс задач программирования в ограничениях – задачи исследования операций и искусственного интеллекта. В таких задачах часто используется некоторое пространство решений, сужением которого достигается результат. Такое сужение можно естественным образом представить как ограничения.

Специфика методологий

Методология структурного императивного программирования – подход, заключающийся в задании хорошей топологии императивных программ, в том числе отказ от использования глобальных данных и оператора безусловного перехода, разработке модулей с сильной связностью и обеспечении их независимости от других модулей. Подход основан на принципах построения:

- последовательная декомпозиция алгоритма задачи сверху вниз;
- использование структурного кодирования.

Данная методология – это важнейшее развитие императивной методологии. Разработчиком подхода считается Эдсгер Дейкстра. Методы и концепции:

- метод алгоритмической декомпозиции сверху вниз – пошаговая детализация постановки задачи, начиная с наиболее общих задач, что обеспечивает хорошую структурированность;

•метод модульной организации частей программы – заключается в разбиении программы на модули, что поддерживается концепцией модуля;

•метод структурного кодирования – это использование при кодировании трёх основных управляющих конструкций, отказ от безусловного перехода, что поддерживается концепцией управления.

Практически на всех языках, поддерживающих императивную методологию, можно разрабатывать программы с использованием методологии структурного императивного программирования.

Класс задач для данной методологии соответствует классу задач для императивной методологии, при этом удаётся разрабатывать более сложные программы, т.к. их легко воспринимать и анализировать.

Методология императивного параллельного программирования – подход, в котором предлагается использование явных конструкций для параллельного использования выбранных фрагментов программ.

Вычислительные задачи часто имеют огромные объёмы. Анализ эффективности их решений позволяет ситуацию значительно улучшить, если использовать для вычислений не одно, а несколько вычислительных устройств одновременно. Разработка аппаратных многопроцессорных архитектур привела к широким исследованиям в этой области. Другая причина возникновения данной методологии – появление сложных программ, требующих поддержки явного параллелизма (например, операционные системы). Такой процесс можно сравнивать с решением N заданий K обучающимися одновременно, распределив их между собой.

В этой методологии используется метод синхронизации исполняемого кода: использование специальных автоматических операций для осуществления взаимодействия между одновременно исполняемыми фрагментами кода. Метод поддерживается концепцией примитивов синхронизации.

Модель вычислений параллельного программирования можно рассматривать в контексте модели императивного программирования. Есть несколько вычислителей, которые имеют общие элементы состояния. Так, если на шаге вычислений с некоторым номером элемент состояния одного вычислителя принимал данное значение, то и для любого другого вычислителя на шаге вычисления с тем же номером соответствующий элемент состояния также должен принимать данное значение.

Синтаксическое взаимодействие параллельных процессов лучше всего представлять как работу сети нескольких устройств, соединённых каналами, по которым текут данные. Спроектировав в таких терминах, например, систему параллельных процессов для быстрого суммирования большого количества чисел, можно описать эту систему одновременной обработки приложений, процессов, подпрограмм, циклов и операторов. Каждый вычислитель производит типичные для его вычислительной модели операции (например, императивный вычислитель будет переходить из состояния в состояние). Когда процесс встречает инструкцию «принять значение из канала», он входит в

состояние ожидания, пока канал пуст. Как только в канале появляется значение, процесс считывает его и продолжает работу.

Суперпозиция задач, как правило, реализуется через систему процессов, обменивающихся между собой информацией о результатах своих вычислений.

Параллельные языки программирования используют явные конструкции для параллельного исполнения выбранных фрагментов программ. Существует несколько языковых подходов к программированию для параллельных вычислительных систем:

1. программирование на параллельном языке программирования:
   a. универсальном (например, Ada);
   b. языке для конкретных типов компьютеров, позволяющих эффективно транслировать программы на параллельном языке именно в эту архитектуру (например, язык Occam изначально разрабатывался для транспьютеров);
2. программирование на широко распространённом языке программирования (например, C, C++, Pascal), который расширен языковыми (на уровне языка программирования) распараллеливающими конструкциями;
3. программирование с использованием дополнительных указаний компилятору на уровне языка прагм (например, по стандарту OpenMP);
4. программирование на широко распространённом языке программирования с использованием высокоуровневых коммуникационных библиотек и интерфейсов для организации межпроцессорного взаимодействия, где конструкции параллелизма выносятся с языкового уровня на уровень операционной системы;
5. применение средств автоматического распараллеливания последовательных программ такими инструментами, как компиляторы.

Таблица 4

Параллельные языки программирования

1960	Algol-68 (1968)
1970	Concurrent Pascal (1972) Modula-2 (1978) CSP (1978)
1980	Edison (1980) Ada (1979, 1983) Occam (1982) Concurrent Prolog (1983) Linda (1985)
1990	Oblig (1993)

Данная методология может очень эффективно применяться для обработки больших однородных массивов данных, которые часто встречаются в реализации вычислительных и статистических методов. Методология параллельного программирования также успешно применяется при моделировании, в операционных системах реального времени.

Методология логического параллельного программирования.

Логическое программирование допускает естественную параллельную реализацию. Язык Concurrent Prolog (Э. Шапиро) развивает логический подход к абстрактным спецификациям и включает несколько идей Дейкстры (в том числе идею охраняемого предложения). При этом подходе предметная область описывается как формальная теория в некотором логико-математическом языке. Когда используется абстрактная модель вычисления логических программ, то предполагается, что они легко поддаются параллельному выполнению. Вычисление логических программ – это доказательство целевого утверждения с использованием аксиом из программы. Пространство поиска вывода может быть описано И/ИЛИ деревом, где И-узлы соответствуют конъюнкции целей, ИЛИ-узлы соответствуют различным путям выбора единичных целей из программы.

В языке Concurrent Prolog применяется особое прочтение логических программ – поведенческое. В поведенческом прочтении единичная цель есть аналог процесса, конъюнкция целей – аналог системы процессов, а общее применение целевых функций – просто каналы связи. Предложение читается поведенчески: процесс *A* может заменять себя системой процессов *B1 и B2 и ... Bn.* Процесс оканчивается заменой себя пустой системой.

Другие методологии

Перечислим ряд других, менее исследованных и менее популярных методологий:

- методология программирования, управляемого потоком данных, - подход заключается в том, что операции срабатывают не последовательно, а в зависимости от готовности данных;
- методология доступ-ориентированного программирования – подход, в котором функции с переменными связываются таким образом, что при доступе к переменной процедура будет вызываться анатомически;
- методология нейросетевого программирования – подход, заключающийся в том, что на основе знаний, полученных от экспертов, создаётся программа на нейронном языке программирования, которая затем компилируется в эквивалентную нейронную сеть из аналоговых нейронов.

Технологический язык

Технологический язык – это некоторый достаточно свободный язык (стиль) оформления текстов, логических утверждений, мыслей, спецификаций на программные проекты, формальных записей алгоритмов и программ с постепенным многоуровневым уточнением любых применяемых в них обозначений, описанием организации работ коллектива исполнителей и т.д. В некотором смысле технологический язык является чистым национальным языком профессиональной (технической, математической, экономической и т.д.) прозы, лексиконом, в котором любые неформальные обозначения последовательно детализируются формальными отношениями (связями) между ними. Программирование в технологическом языке – это программирование в естественном языке в графической или линейной текстовой форме с использованием произвольных профессиональных обозначений, наиболее

естественных и удобных в соответствующей предметной области на каждом шаге её формализации (решение соответствующей задачи, построение программы и пр.). Программируются не обозначения, а отношения между ними. Система автоматизации технологического языка (в отличие от традиционных автоматизированных систем программирования) формирует операционную среду решения задачи, на выходе которой будет документально обоснованная схема её решения на всём интервале работы программиста.

Технологический язык программирования заключён в некоторую форму, оболочку (графическое представление, поле спецификаций, поле абстракций, кадры проектирования, порядок их заполнения и т.д.), которая обуславливает рациональные действия отдельного или коллектива программистов при изготовлении соответствующего программного изделия.

Языки программирования, включая традиционные, широко распространённые – Pascal, C и другие, служат в этом случае лишь некоторым «наполнителем» соответствующей технологической формы (оболочки) на определённом (небольшом) этапе создания программного изделия – на этапе кодирования фрагментов алгоритмов для работы машины.

Технологический язык программирования сформировался в результате обобщения предшествующего опыта.

Основным понятием технологического языка является чертёж. Чертёж программы или модуля может быть оформлен в соответствии с правилами «Единой системы конструкторской документации» (ЕСКД) – заключён в рамку с основной надписью (угловым штампом).

Чертёж – это поле одного из форматов: А3, А4, А5. В чертеже программы различают официальную и рабочую части. В официальной части записывают: название чертежа; фамилию исполнителя; дату оформления (внесения изменений в чертёж); номер листа в проекте; номер уровня, слоя и варианта проектирования; номер предыдущего и следующего листов (чертежей) в дереве проекта. Остальная официальная часть информации, в соответствии с ЕСКД, определяется и регламентируется для обязательного заполнения в процессе технологической подготовки работ и формирования соответствующей технологической линии производства программ.

Поле спецификаций
Рабочее поле
Поле обстракций

Рис. 4. Схема чертежа

Технологический язык программирования имеет две важные особенности:

1. большинство информации в официальной части чертежа, а также его формирование и оформление осуществляются автоматически средствами поддержки соответствующего технологического комплекса программиста;
2. средства поддержки языка ориентируют пользователя на безбумажную технологию работы за экраном рабочего места программиста.

Рабочую часть чертежа делят на три поля: спецификаций, рабочее и абстракций.

Размеры полей не оговариваются и могут быть произвольными, а также поля могут отсутствовать. Например, весь чертёж может состоять только из спецификаций, в которых записаны техническое задание или инструкция пользователю. Спецификация может занимать несколько листов (чертежей), расположенных подряд или иерархически, соответственно структуре текста.

Поле спецификаций предназначено для описания на естественном неформальном языке цели и задачи, которые ставит перед собой программист на данном шаге построения своей программы, изображённой в рабочем поле чертежа. Если это алгоритм, цели и задачи которого определены на более высоких уровнях иерархии проектирования, то в поле спецификаций описывается, что (а не как) делает алгоритм, описываются данные на его входе и выходе и в чём состоит процесс их преобразования, почему в рабочем поле он выполнен так, а не иначе. Язык спецификаций должен быть естественным, точным и однозначным. В спецификации допускаются иллюстрации, помогающие понять цели, мотивы и обоснования принимаемых проектных решений. Информация, которую заполняет программист в поле спецификаций, предназначена прежде всего для него самого (это не документация, не инструкция пользователю), для концентрации его внимания на данном шаге выполнения работы и в соответствии с этим правильного заполнения рабочего поля впоследствии.

Рабочее поле содержит запись описательной и исполнительной частей программы на данном уровне её проектирования.

Описательная часть обычно предшествует исполнительной. Форма записи описательной части может быть произвольной: линейная запись на одном из языков программирования (допускается смесь языков и неточное соблюдение синтаксиса); просто список переменных; использованных в рабочем поле; таблицы входных или выходных данных; графы, отображающие их структуру, и т.д. Форма записи должна быть такой, чтобы наиболее естественно отобразить данные, используемые на этом шаге (уровне) проектирования.

Исполнительная часть обычно изображается в графическом виде, хотя допускается и традиционная линейная запись. Можно использовать любой язык (естественный или машинный, алгоритмический), любой алфавит (славянский или латинский), профессиональный язык любой группы пользователей (математиков, физиков, экономистов, медиков и пр.). Идентификаторы и записи могут быть многострочными.

В поле абстракций программист определяет все понятия и обозначения, использованные им в рабочем поле, аналогично тому, как это делается в математических текстах после наречия «где:».

Поля спецификаций и абстракций играют важную взаимодополняющую роль – они документируют процесс мышления программиста, делают его целенаправленным и концентрированным. В поле спецификаций записывается

информация до выполнения данного шага проектирования, а в поле абстракций – после выполнения.

Очень важно, что поля спецификаций, абстракций и рабочее заполняются программистом за экраном дисплея по безбумажной схеме. Это позволяет просто производить коррекцию и постоянное улучшение записей чертежа в соответствии с итерационным процессом мышления программиста в режиме проб и ошибок. В целом то, что каждый мыслительный акт работы программиста, оставаясь свободным и творчески не ограниченным по существу и языку выражения, задокументирован по определённой единой и обязательной для всех форме: с графической записью алгоритмов, с сохранением порядка и хронологии определения по безбумажной схеме делает программные проекты долгоживущими и легко сопровождаемыми.

Ещё одним важным элементом технологического языка, которого нет в традиционных языках программирования, является понятие технологического фрейма проектирования. Это понятие имеет некоторую статическую структуру (например, информация, выводимая на экран; подсказка программисту и т.п.) и допустимые операции над ней (что можно дописывать, куда можно перейти, какие действия допустимы и т.п.). Примерами технологических фреймов являются описанные поля спецификаций, рабочие и абстракций, кроме того, к фреймам относятся: паспорт разработчика, список текущих, запланированных и выполненных работ, поле телеграмм и так далее. Одним из фреймов может быть также обычный транслятор или компилятор традиционного языка программирования вместе с определённой технологической дисциплиной его использования.

Каждый технологический фрейм определяется и компонуется вместе с другими в некоторую технологическую сеть на этапе технологической подготовки работ. Средства поддержки технологического языка, в отличие от традиционных трансляторов и компиляторов языков программирования, позволяют писать и вводить в систему новые фреймы, а также компоновать из них соответствующую технологическую линию.

КОЛЛЕКТИВНАЯ РАЗРАБОТКА ПРИЛОЖЕНИЙ

Создание крупных информационных систем требует согласованной работы целой группы программистов. Несколько лет назад проблемы организации взаимодействия отдельных разработчиков при создании крупных проектов были актуальны в основном для крупных фирм – производителей программного обеспечения. Однако с появлением и развитием систем быстрой разработки приложений (RAD) ситуация изменилась. Внедрение средств RAD позволяет повысить производительность труда как отдельных программистов, так и рабочих групп. Благодаря этому полный цикл разработки крупных проектов может выполняться существенно меньшими коллективами. Т. о., проблемы обеспечения согласованной работы отдельных программистов, выполняющих разработку крупного проекта, стали актуальными и для небольших рабочих групп.

Системы обеспечения коллективной разработки приложений применимы к широкому спектру задач. Основной из них является обеспечение управляемости и контролируемости процессов разработки и сопровождения приложения. Для этого необходимо обеспечить выполнение минимум двух функций:

- регистрации всех изменений, вносимых в проект;
- централизованного хранения файлов проекта.

Под проектом понимается множество файлов с исходными текстами программ, а также файлов ресурсов и всех прочих файлов (исполняемые файлы, библиотеки, объектные модули), необходимых для выполнения компиляции и запуска приложения.

Управление проектом

Из-за больших объёмов проектов разработка программного обеспечения ведётся коллективом специалистов. Работая в коллективе, отдельные специалисты должны взаимодействовать друг с другом, обеспечивая целостность проекта, что достаточно трудно.

Одной из главных составляющих, необходимых для производства программного обеспечения, являются люди. Несомненно, в первую очередь оцениваются технические навыки персонала. Однако эти навыки необходимо применять для решения проблем в нужное время и в нужном месте. Это предполагает комбинацию работы в команде и лидерства.

Говорят, что «не бывает технических провалов, а бывают только провалы в управлении». Хотя это и не совсем правильно, но для успешного завершения деятельности по разработке управление играет очень важную роль.

Руководитель должен сделать так, чтобы технические изыскания разработчиков имели нужное направление. Диктатура в управлении может вызвать негодование коллектива и привести к потере мотивации. Однако излишнее попустительство может привести к потере времени и напрасно проделанной работе. Решением данной проблемы является лидерство, выяснение истинных желаний и потребностей людей, активное их распределение и объединение в попытке достижения успеха. Лидер проекта должен варьировать степень своей ответственности в управлении, основываясь на величине проекта. В больших проектах их обязанность состоит в основном в управлении. В малых же проектах лидеры должны обеспечивать как общее управление, так и личное участие непосредственно в разработке.

Одной из обязанностей руководителя проекта является проведение совещаний. Так как команды, как правило, не очень хорошо могут создавать артефакты «с ходу» (особенно это касается проектирования), очень полезно иметь заготовку - шаблон в качестве основы дискуссии. Например, ответственный за проектирование может подготовить эскиз архитектуры или руководитель проекта может подготовить предварительное распределение работ. Эти эскизные проекты не должны быть очень конкретны, чтобы оставалась возможность для творческого вклада участников команды.

Хорошей практикой считается предварительная подготовка повестки дня совещания. Обсуждение рисков также следует, по мере необходимости, включать в повестку совещаний (на первой трети проекта для каждого совещания).

Метод главного программиста

Важное место среди вопросов технологии программирования занимает организация коллектива программистов. Наиболее распространены две формы организации коллектива:

- метод главного программиста;
- функциональная.

Метод главного программиста лучше учитывает специфику индивидуального труда программиста. Суть метода заключается в следующем. Для разработки программного проекта формируется группа главного программиста. В состав коллектива, кроме самого программиста, входят: помощник (заместитель), администратор библиотеки, программист – технолог инструментальных средств и несколько (6-7человек) проблемных программистов.

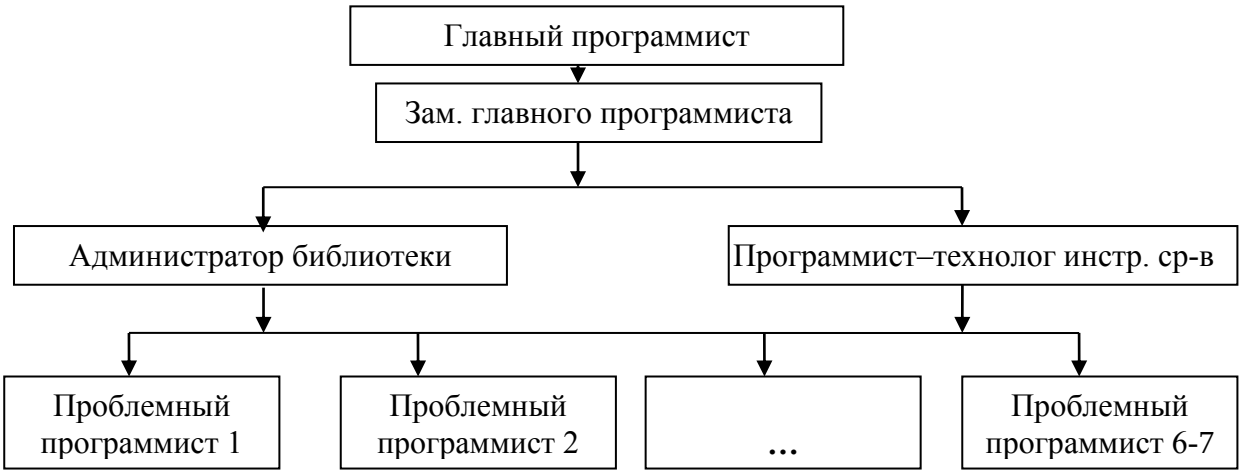

Рис. 5. Структура метода главного программиста

Всю работу проводит в основном главный программист. Все остальные ассистируют и помогают ему. Главный программист выполняет все работы по проектированию системы на верхних уровнях, следит за результатами кодирования и отладки рабочих модулей, проверяет полноту тестов, отвечает за стыковку модулей, определяет структуру, руководит составлением документации программной системы и т.д. Он единолично отвечает за весь проект, за интерфейсы между отдельными работами. Поэтому весь проект получается логически более стройным, чем при функциональной организации работ, где каждая функциональная группа отвечает за интерфейс по входу и выходу своей программы перед другими разработчиками группы.

Заместитель главного программиста дублирует руководителя по всем вопросам организации работ и, в первую очередь, отвечает за проведение

единой технической и технологической политики внутри коллектива, обеспечивает концептуальное единство и стройность проекта.

Проблемные программисты (исполнители), если задача перед соответствующим коллективом стоит большая, могут быть организованы в некоторую иерархическую схему под управлением ответственных исполнителей, которая удовлетворяет эффективности управления: 5-8 исполнителей.

В любом случае, руководитель и его заместитель в непосредственном программировании не участвуют, ответственные исполнители участвуют частично, в зависимости от уровня проекта, степени автоматизации технологического процесса и порядка, установленного в соответствии с технологической линией. Задачи остальных компонентов вытекают из их должности.

Функциональная форма организации коллектива программистов

Функциональная форма организации коллектива в чистом виде соответствует разделению коллектива на группы, каждая из которых решает свою функциональную задачу. Каждая функциональная группа отвечает за интерфейс по входу и выходу своей программы перед другими группами разработчиков. В этом слабое место функциональной организации работ программистов, так как на интерфейсы и согласования между группами тратится очень много усилий и времени.

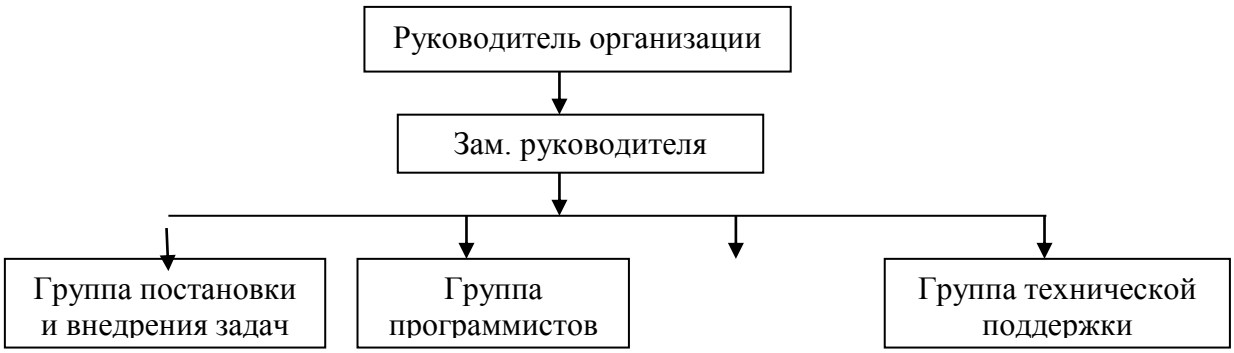

Рис. 6. Схема функциональной формы организации коллектива

Возможна и другая организация коллектива разработчиков, например, применение двух рассмотренных методов одновременно.

Тестирование и отладка

Любой заказчик хочет получить надёжное программное изделие, которое полностью удовлетворяет его потребности. Различные уровни надёжности обеспечиваются разными инженерными подходами к тестированию. За достаточные средства можно достичь уровня надёжности, как у такой-то фирмы, или даже уровня, необходимого для атомной энергетики или космических исследований.

Уровень надёжности программных изделий определяется инженерией тестирования и напрямую связан с затратами как денежных средств, так и времени проекта.

Известно, что при создании типичного программного проекта около 50% общего времени и более 50-60% стоимости расходуется на проверку (тестирование) разрабатываемой программы или системы. Кроме того, доля стоимости тестирования в общей стоимости программ имеет тенденцию возрастать при увеличении сложности программных изделий и повышении требований к их качеству.

Учитывая это, при выборе способа тестирования программ следует чётко выделять определённое (по возможности не очень большое) число правил отладки, обеспечивающих высокое качество программного продукта и снижающих затраты на его разработку.

Тестирование осуществляется путём исполнения тестов. Аксиомы тестирования, выдвинутые ведущими программистами:

- хорош тот тест, для которого высока вероятность обнаружения ошибки;
- главная проблема тестирования – решить, когда закончить (обычно решается просто – кончаются деньги);
- невозможно тестировать свою собственную программу;
- необходимая часть тестов – описание выходных результатов;
- избегайте невоспроизводимых тестов;
- готовьте тесты как для правильных, так и для неправильных данных;
- не тестируйте «с лёту»;
- детально изучите результаты каждого теста;
- по мере обнаружения всё большего числа ошибок в некотором модуле или программе, растёт вероятность обнаружения в ней ещё большего числа ошибок;
- тестируют программы лучшие умы;
- считают тестируемость главной задачей разработчиков программы;
- не изменяй программу, чтобы облегчить тестирование;
- тестирование должно начинаться с постановки целей.

Если в программе ставят комментарии на месте вызова модуля вместо использования заглушки, то исключают возможность проверки типов данных, а также часто забывают снимать комментарии. Для поиска «забытых» комментариев необходима трудоёмкая отладка.

Среди приёмов тестирования стоит выделить также так называемую *отладочную печать*. Если отладочные печати изымаются из текста, то утяжеляется сопровождение, поэтому никогда не изымайте отладочную печать.

Тестирование программ охватывает ряд видов деятельности:

- постановку задачи;
- проектирование тестов;
- написание тестов;
- тестирование тестов;
- выполнение тестов;
- изучение результатов тестирования.

Наиболее важной деятельностью является проектирование тестов.

Проектирование комплексного теста. В комплексном тесте должны проводиться следующие виды тестирования:

- работоспособности;
- стрессов;
- предельного объёма вводимых данных;
- конфигурации различных технических средств;
- совместимости;
- защиты;
- требуемой памяти;
- производительности;
- настройки;
- надёжности;
- средства восстановления при отказе;
- удобства обслуживания;
- программной документации;
- психологических факторов;
- удобства эксплуатации.

Чтобы построить разумную стратегию тестирования, надо разумно сочетать искусство проектирования тестов и научные подходы.

Часто оказывается, что один и тот же инструмент отладки может указать на факт наличия ошибки, одновременно локализовать место её возникновения и указать причину. Большая часть инструментов связана с технической стороной вопроса.

Тестовые мониторы. Тестовый монитор состоит из трёх основных компонентов.

1. Ядро системы, содержащее основные программы тестирования и оформления результатов.
2. Собственно тестовая база, включающая исходный тест и эталонные результаты пропуска тестов.
3. Тестовое пространство, обычно содержащее специфический настроечный файл.

В результате запуска тестового монитора в тестовом пространстве начинают исполняться заданные сюиты из тестовой базы. На основе результатов тестирования можно сделать выводы об изменении качества программного продукта.

Средства отслеживания тестового покрытия. Системы такого типа предназначены для выявления тестового покрытия программы, в том числе участков кода, пропущенных при тестировании.

Важную роль здесь играет понятие линейного участка – *фрагмента* программы, на протяжении которого нет передачи управления. Если выполняется первый оператор такого участка, то будут выполнены и все остальные. Часто данные системы используют в качестве вспомогательного средства профайлер.

Средства динамического построения профиля программы. Построение профиля программы позволяет обнаружить фрагменты кода, исполняемые при запуске программы. Полученная информация к размышлению позволяет выявить в программе те места, которые надо оптимизировать. Работа со средствами динамического построения профиля программы включает три этапа.

1. Компиляция и сборка программы с включённым режимом добавления кода с целью генерации информации для построения профиля.
2. Запуск программы с целью получения информации, необходимой для построения профиля.
3. Запуск профайлера для полученной информации.

В простейшем случае результатом работы профайлера будет исходный текст программы, где для каждого линейного участка указано число: сколько раз этот исходный участок был исполнен. Следовательно, часто используемые участки надо максимально оптимизировать с целью ускорения работы всей программы.

Системы построения срезов программы. Программный срез состоит из всех операторов, которые могут влиять на значение некоторой переменной в некоторой позиции. Срезы были введены для использования:

- в отладке программ;
- в тестировании программ;
- в понимании смысла программ.

Во время работы с исходным текстом программы часто необходимо проследить формирование значений. Это особенно актуально во время отладки и сопровождения программ. В этом случае получение среза приносит существенное облегчение работы, так как резко снижает объём исследуемого кода. Предположим, что во время тестирования установлено, что некоторая переменная в некоторой позиции содержит неправильное значение. Получив соответствующий срез, локализуем проблему. Срезы позволяют разделить большую программу на небольшие компоненты, что способствует более простому пониманию смысла программы при сопровождении.

Отладчики – программы, помогающие анализировать поведение наблюдаемой программы, обеспечивая её трассировку. При этом большинство отладчиков позволяют выполнять остановки в указанных точках или в заданных условиях, выводить текущие значения переменных, ячеек памяти, регистров процессора и, при необходимости, изменять эти значения. Нужные для отладчика данные порождает компилятор при включённом режиме генерации отладочной информации. Это данные о типах, начале и конце блоков, специфических конструкциях и т.п.

Кроме классических отладчиков существуют ещё несколько групп инструментов.

- *Средства динамической отладки распределения памяти* – специальный инструментарий, позволяющий анализировать распределение памяти в процессе исполнения программы.

• *Средства отладки многопоточных и параллельных приложений.* Это приложения, которые в некоторой точке могут разветвиться для выполнения параллельной работы, потом попадают в точку ожидания и опять выполняются последовательно.

Примеры – средства компании Sun Microsystems:

• thd (Thread Analiser) – нитевой анализатор, который исполняется для того, чтобы наблюдать многопоточные приложения;

• locklint – анализатор взаимных исключений;

• looptool – анализатор распараллеливания программ.

Системы наблюдения проблем (ошибок) обычно представляют собой интерфейс к обыкновенной базе данных. Наиболее сложный вопрос при разработке таких систем – отбор информации, которая связана с проблемой (ошибкой). Одна из возможных структур такой информации:

- основная информация об ошибке:
  - номер ошибки;
  - краткое описание проблемы;
  - ключевые слова;
  - приоритет;
  - состояние;
- место возникновения ошибки:
  - программный продукт;
  - версия продукта;
  - компонент;
  - особенность;
- персоналии:
  - ответственный менеджер;
  - ответственный инженер;
  - инженер, выставивший ошибку;
  - адреса рассылки;
- анализ ошибки и исправление:
  - подробное описание ошибки;
  - тест, на котором проблема проявляется;
  - предлагаемое исправление;
  - комментарии;
  - информация для пользователей;
- влияние исправления:
  - интеграция регрессионного теста в тестовую базу данных;
  - влияние исправления на документацию программного продукта;
  - влияние исправления на другие компоненты.

Примерами систем наблюдения проблем являются:

• Bugzilla, применяемая для сбора информации об ошибках гипертекстового браузера Mozilla;

• Open Source Web Browser – авторская разработка Адама Сигеля.

УЧЕБНЫЕ ДЕЛОВЫЕ ИГРЫ

Учебная деловая игра (УДИ) – это форма воссоздания предметного и социального содержания будущей профессиональной деятельности специалиста, моделирования тех самых отношений, которые характерны для этой деятельности как целого. В ней воспроизводятся наиболее типичные профессиональные ситуации в сжатом виде. В условиях (УДИ) обучающийся приобретает не только компетенции специалиста, но и социальные. Это в определённой мере репетиция производственной и общественной деятельности.

Модификации УДИ:

1. Имитационные игры – на занятии имитируется деятельность какой-либо организации или подразделения (деятельность коллектива разработчиков ПО), события (деловые совещания, обсуждение плана …), обстановка, условия (сдача ПО Заказчику).

2. Использование ролей – отрабатывается тактика поведения, действий, выполнение функций и т.д. Модель – пьеса ситуации, между участниками распределяются роли с «обязательным содержанием» (руководитель разработки, помощник руководителя, старшие групп разработчиков и их заместители, разработчики).

Основная цель всех УДИ – научить участников ориентироваться в различных ситуациях, учитывать возможности и состояния других людей, устанавливать с ними контакты, влиять на их интересы и пр. Необходимо разработать сценарий, где описываются функции и обязанности действующих лиц, их задачи.

Методика проведения УДИ:

- Этап подготовки аудитории, участников и экспертов. Определяется режим работы, формулируется главная цель, обосновывается постановка проблемы. Готовятся пакеты материалов.
- Этап изучения области автоматизации, установок и других материалов. Собирается дополнительная информация, проводятся консультации. Участники игры контактируют между собой.
- Этап проведения – процесс игры. С момента игры никто не имеет права вмешиваться и изменять её ход. Только ведущий (не обязательно преподаватель) может корректировать действия участников, если они отклоняются от главной цели игры.
- Этап анализа, обсуждения и оценки результатов игры. Выступает ведущий, эксперт, все обмениваются мнениями. Идёт защита участниками своих решений и выводов.

Принципы организации УДИ:

1. Принцип имитационного моделирования конкретных условий и динамики производства, а также игрового моделирования содержания профессиональной деятельности специалиста.

2. Принцип проблемности содержания игры и процесса её развёртывания.
3. Принцип совместной деятельности участников на уровне взаимодействия каждого с каждым.
4. Принцип диалогического общения и взаимодействия партнёров по игре. «Вопросное» состояние участников игры.
5. Принцип двуплановости игровой деятельности. Серьёзная деятельность реализуется в «несерьёзной» игровой форме, что позволяет участникам интеллектуально и эмоционально раскрепоститься, снять «зажатость», проявить инициативу, творчество.

Правила для участников игры:

- уважать право другого на собственное мнение, даже если оно резко расходится с твоим собственным;
- вдумчиво и терпеливо выслушивать своего товарища, стараясь понять его;
- не отрицать сразу всё сказанное оппонентом в споре, а заметить сильные и слабые стороны в его словах, аргументации;
- взвешивать всё, что сам собираешься сказать;
- быть готовым признать правоту другого;
- не горячиться и не торопиться с выводами;
- постоянно находиться в «вопросном» состоянии.

Эксперт или ведущий заранее готовит график и табло игры.

ЦЕЛЬ И СОДЕРЖАНИЕ ДИСЦИПЛИНЫ

Цель изучения дисциплины: согласно требованиям к обязательному минимуму содержания основной профессиональной образовательной программы высшего образования по направлению подготовки бакалавра направления «Информатика и вычислительная техника» целью дисциплины является: формирование компетенций ОК-6, ПК-1, ПК-2, ПК-3, путём теоретической и практической подготовки обучающихся в области технологий программирования, в такой степени, чтобы они могли выбирать необходимые аппаратные, алгоритмические, программные и технологические решения для постановки, разработки и реализации конкретных задач.

Она призвана обобщить знания, полученные студентами - бакалаврами направления «Информатика и вычислительная техника» в процессе обучения. Отобразить полную картину работы будущего специалиста.

За период изучения дисциплины «Технологии программирования» используются следующие формы привлечения студентов к самостоятельной и творческой деятельности:

- для лабораторных, курсовых и самостоятельных занятий студентам всей группы предлагается разработать одно творческое задание – проект АСУ клиент-серверной архитектуры на выбранную тему;
- работа выполняется с использованием метода учебная деловая игра (УДИ);

•элементами научного поиска являются сбор, обработка, анализ данных области автоматизации, поиск, разработка и решение математических моделей, оформление научно-практической документации и презентации.

Специфика изучения данной дисциплины выражается следующими положениями:

- Необходимо первоначальное изучение работы с приложениями Microsoft Office (Word, Excel, PowerPoint); Delphi, умение программировать базы данных в объектно – ориентированной среде.
- Обследование объекта автоматизации, изучение литературы и её обзор, актуализация и выявление цели и задач предстоящей работы, организация коллектива разработчиков.
- Постановка и реализация программы требуют от студентов большого количества дополнительных самостоятельных знаний и умений, например, из области сетевых и Internet-технологий, дизайна, менеджмента проекта и др., которые студенты изучают по ходу разработки.

При решении практических задач студенты могут разработать собственные варианты и модификации функций и подсистем. Предполагается коллективная разработка собственной автоматизированной системы обработки информации клиент-серверной архитектуры.

Курсовая работа предполагает разработку, программирование, заполнение, стыковку, документирование, отладку и защиту коллективно разработанной автоматизированной системы.

В курсовой работе должны быть использованы методы коллективной разработки программного обеспечения и современные информационно-коммуникационные технологии.

В отчёте по курсовой работе должны быть разделы, соответствующие ГОСТам:

- постановка задачи (ТЗ) с описанием проблемы, цели, задач, выходных и входных структур данных, требования к аппаратно – программной среде разработки и эксплуатации ПП;
- описание элементов БД и их связей;
- алгоритмы решений;
- необходимые руководства пользователей;
- контрольные примеры, доказывающие правильность работы системы.

Используется метод проектов во время коллективной разработки приложения. Использование мультимедийного и компьютерного оборудования во время разработки и защиты проекта.

В данном курсе каждый студент имеет возможность испытать свой накопленный теоретический и практический опыт - проверить: готов ли он начать самостоятельную профессиональную карьеру.

Пусть всё задуманное у Вас получится лучшим образом.

В течение семестра предстоит *посетить и активно прослушать лекции (34 часа):*

	Модуль 1 «Введение и классификации» (12ч.)
	Введение в технологию программирования (2ч.).
Лекция 1.	Понятие «технология». Типы и виды технологий программирования. Технологические процессы и стадии.
	Понятие методологии (2ч.).
Лекция 2.	Определение методологии. Основные методологии. Атрибуты и ядра методологий. Топологическая специфика методологий. Методология объектно-ориентированного проектирования и программирования.
Лекция 3.	Понятие классификации технологических подходов к организации обработки различных видов информации (2ч.).
Лекция 4-5.	Группы подходов. Подходы со слабой формализацией. Классические подходы. Гибкие (адаптивные) подходы (4ч.).
	Классификация технологических процессов (2ч.)
Лекция 6.	Набор классических процессов. Набор стандартный (ISO12207:1995).
	Модуль 2 «Методологии и технологии» (12ч.)
Лекция 7.	Классификация технологических стадий (2ч.).
Лекция 8-9.	Технологические подходы (4ч.). Проблемы и перспективы развития технологических подходов. Характеристика технологических подходов программирования.
	Методология и технология программирования ИС (2ч.).
Лекция 10.	Методологии, технологии и инструментальные средства проектирования (CASE- средства).
Лекция 11-12.	Методология RAD-Rapid Application Development (4ч.). Основные особенности методологии RAD. Объектно-ориентированный подход. Визуальное программирование. Событийное программирование.
	Модуль 3 «Методы коллективной разработки систем» (10ч.)
Лекция 13.	Коллективная разработка приложений по обработке информации (2ч.). Структура средств коллективного проектирования. Хранилища данных. Интенсификация проекта и его составляющих в TeamSource.
Лекция 14.	Вычислительные сети (2ч.). Интернет-технологии обработки информации. Способы коммутации данных. Протоколы обмена данными в эталонных моделях взаимодействия открытых систем.
	Архитектура вычислительных систем (2ч.).
Лекция 15.	Сущность и задачи управления. Управляющие воздействия. Обратная связь в управлении. Иерархия в системе управления. Оптимальное управление и его критерии.
Лекция 16.	Системы отслеживания проблем и ошибок при обработке и передаче информации. Доказательство свойств программ (2ч.).

Программирование защиты информации (2ч.).

Лекция 17. Угрозы безопасности информации. Обеспечение достоверности информации. Обеспечение сохранности информации и её конфиденциальности.

Выполнить лабораторные работы (34 часа):

Модуль 1 «Введение и классификации» (12ч.):
Лабораторная № 1 Разработка интерфейса (2ч.).
Лабораторная № 2. Разработка верхнего уровня архитектуры АС (2ч.).
Лабораторная № 3. Реализация алгоритма взаимодействия структур АС.(2ч.).
Лабораторная № 4. Программирование основных функций (2ч.).
Лабораторная № 5. Программирование основных функций (2ч.).
Лабораторная № 6 Программирование таблиц, запросов БД АС (2ч.).
Модуль 2 «Методологии и технологии» (12ч.):
Лабораторная № 7 Программирование таблиц, запросов БД АС (2ч.).
Лабораторная № 8. Заполнение таблиц БД (2ч.).
Лабораторная № 9. Оформление запросов к БД (2ч.).
Лабораторная № 10 Разработка форм отчётности (2ч.).
Лабораторная № 11 Проектирование отчётов (2ч.).
Лабораторная № 12 Стыковка элементов проекта (2ч.).
Модуль 3 «Методы коллективной разработки систем» (10ч.):
Лабораторная № 13. Сборка проекта (2ч.).
Лабораторная № 14. Отладка проекта (2ч.).
Лабораторная № 15.Написание инструкций и сборка документации на АС (2ч.).
Лабораторная № 16. Подготовка презентаций (2ч.).
Лабораторная № 17. Презентация коллективного проекта (2ч.).

Выполнить, оформить документацию и презентовать защиту групповой курсовой работы

Курсовая работа предполагает разработку, программирование, заполнение базы данных, стыковку модулей, отладку системы и защиту коллективно разработанной АС, в которой каждый обучающийся получает своё задание, что отражается в сетевом графике работ над системой.

В курсовой работе должны быть использованы методы коллективной разработки программного обеспечения, технологии программирования и современные информационно-коммуникационные технологии.

В отчёте о коллективной курсовой работе должны быть разделы, соответствующие ГОСТам:

- постановка задачи (ТЗ);
- описание выходной и входной информации;
- описание БД;
- алгоритм решения;
- необходимые руководства пользователей;
- список литературы;

- результаты работы;
- контрольный пример, доказывающий правильность работы системы.

Организация самостоятельной работы студентов по дисциплине

Самостоятельная работа студентов составляет 35часов

№ п/п	Перечень самостоятельных занятий	Объем СРС час/вес	Рекомендуемая литература
1	Изучение первоисточников специальной литературы, области автоматизации и ГОСтов	10/0,29	[1, 2, 3]
2	Подготовка к текущему контролю и лабораторным работам	12/0,34	[1-3, 5, 7, 9, 10-14]
3	Изучение дополнительной литературы	5/0,14	[4, 6, 8, 10-12]
4	Курсовая работа	8/0,23	[1-14]
	ИТОГО:	35/1,00	
	экзамен	41/1,00	

Самостоятельная работа студентов по дисциплине «Технологии программирования» состоит в изучении специальной литературы, подготовке к очередным лекциям, лабораторным и курсовой работам, контрольным точкам и экзамену.

Привлечение обучающихся к самостоятельной и коллективной творческой деятельности обеспечивается за счет элементов научного творчества, являющихся обязательными при изучении дисциплины. При этом используются следующие формы привлечения студентов к самостоятельной творческой деятельности:

Ежедневная подготовка требует от студента глубокого изучения теоретического материала лекций.

На организацию СРС по выполнению курсовых работ отводятся определенные часы из расчёта общей нагрузки. Теоретическая база подкрепляется самостоятельным освоением рекомендованной кафедрой литературой в читальных залах или библиотеке РИИ, работой с Internet-ресурсами.

При решении практических задач студенты могут разработать собственные варианты и модификации функций и подсистем. Предполагается коллективная разработка студентами собственной автоматизированной системы обработки информации клиент-серверной архитектуры.

Перечень учебно-методических разработок по дисциплине приведен в списке литературы настоящего пособия. Каждый обучающийся обязан получить в библиотеке по 1 экземпляру учебно-методической разработки, иметь их на лекционных и лабораторных занятиях, активно использовать при получении заданий от преподавателя.

Разъяснения по поводу применения модульно-рейтинговой системы квалиметрии при оценке деятельности обучающегося приведены в памятках дисциплины, которые выдаются каждому студенту в начале семестра.

Для самостоятельной работы студентов кафедрой «ПМ»:

1) Издано настоящее методическое пособие.
2) Скомплектован фонд учебников по технологиям программирования в библиотеке РИИ.
3) Скомплектована и размещена в электронной библиотеке РИИ электронная библиотека учебников по дисциплине «Технологии программирования».

График контроля

Контрольное испытание	Время проведения	Вес в итоговом рейтинге (балл)
Лекции	1-17 неделя	0,1(10)
Лабораторные работы	1-17 неделя	0,2(20)
Курсовая работа	выдача – 2неделя, защита - 17 неделя	0,3(30)
Экзамен	сессия	0,4(40)
	Итого:	1 (100)

Примечания:

1. Любая контрольная точка, выполненная после срока без уважительной причины, оценивается на 10 % ниже. Максимальная оценка в этом случае 90 баллов. Контрольная точка, выполненная после начала сессии, оценивается 25 баллами.
2. К экзамену допускаются обучающиеся, имеющие не более одной задолженности по контрольным точкам. При наличии одной задолженности на экзамене выдаётся дополнительное задание.
3. «АВТОМАТЫ» по дисциплине «Технологии программирования» не выставляются.

Карта компетенций дисциплины «ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ»

Код компетенции по ФГОС ВО	*Содержание компетенции (или ее части)*
ОК-6	способность работать в коллективе, толерантно воспринимая социальные, этнические, конфессиональные и культурные различия
ПК-1	способность разрабатывать модели компонентов информационных систем, включая модели баз данных и модели интерфейсов «человек - электронно-вычислительная машина»
ПК-2	способность разрабатывать компоненты аппаратно-программных комплексов и баз данных, используя современные инструментальные средства и технологии программирования
ПК-3	способность обосновывать принимаемые проектные решения, осуществлять постановку и выполнять эксперименты по проверке их корректности и эффективности

Компонентный состав дисциплины

Модуль дисциплины (раздел, тема)	Результаты освоения дисциплины	Технология формирования компетенций	Средство и технологии оценки	Объём в ЗЕТ
Модуль 1 «Введение и классифи-кации»	Знает: понятие «технология», методология, классификацию технологических подходов, типы и виды технологий программирования, технологические процессы и стадии. Основные методологии.	Лекции, лабораторные работы, курсовая работа, самостоятельная работа с рекомендованной литературой, подготовка к экзамену.	Защита лабораторных и курсовой работ, сдача экзамена.	1,2
	Умеет: разрабатывать архитектуру программы, реализовывать алгоритмы взаимодействия структур АС, таблиц, запросов БД.	Лекции, лабораторные работы, курсовая работа, самостоятельная работа с рекомендованной литературой, подготовка к экзамену.	Защита лабораторных и курсовой работ, сдача экзамена.	
	Владеет: разработкой интерфейса, программированием основных функций БД.	Лекции, лабораторные работы, курсовая работа, самостоятельная работа с рекомендованной литературой, подготовка к экзамену.	Защита лабораторных и курсовой работ, сдача экзамена.	
Модуль 2 «Методы коллектив-ной разработки систем»	Знает: классификацию технологических стадий, подходов. Средства проектирования: (CASE-средства), методологию RAD, объектно-ориентированный подход, визуальное и событийное программирование.	Лекции, лабораторные работы, курсовая работа, самостоятельная работа с рекомендованной литературой, подготовка к экзамену.	Защита лабораторных и курсовой работ, сдача экзамена.	1,2
	Умеет: разрабатывать и заполнять таблицы БД, запросы, формы, отчёты.	Лекции, лабораторные работы, курсовая работа, самостоятельная работа с рекомендованной литературой, подготовка к экзамену.	Защита лабораторных и курсовой работ, сдача экзамена.	
	Владеет: программированием таблиц, запросов БД АС, стыковкой элементов проекта.	Лекции, лабораторные работы, курсовая работа, самостоятельная работа с рекомендованной литературой, подготовка к экзамену.	Защита лабораторных и курсовой работ, сдача экзамена.	

Модуль 3 «Методологии и технологии»	Знает: структуру средств коллективного проектирования, Интернет-технологии обработки информации, архитектуру ВС, систем-проблем и ошибок при обработке и передаче, доказательство свойств программ, програм-мирование защиты информации.	Лекции, лабораторные работы, курсовая работа, самостоятельная работа с рекомендованной литературой, подготовка к экзамену.	Защита лабораторных и курсовой работ, сдача экзамена.	1,3
	Умеет: оформлять инструкции, готовить и проводить презентацию проекта.	Лекции, лабораторные работы, курсовая работа, самостоятельная работа с рекомендованной литературой, подготовка к экзамену.	Защита лабораторных и курсовой работ, сдача экзамена.	
	Владеет: методами сборки проекта и его отладки.	Лекции, лабораторные работы, курсовая работа, самостоятельная работа с рекомендованной литературой, подготовка к экзамену.	Защита лабораторных и курсовой работ, сдача экзамена.	
Экзамен				0,3
			Итого	4,0

Формы и содержание текущей и промежуточной аттестации по дисциплине

Форма текущей аттестации в 6 семестре – защита лабораторных работ.

Формы промежуточной аттестации в 6 семестре – курсовая работа и экзамен.

С целью закрепления теоретических знаний, полученных по дисциплине «Технологии программирования», обучающиеся выполняют лабораторные и курсовую работы.

График контроля
6 семестр

Контрольное испытание	Время проведения	Вес в итоговом рейтинге (балл)	Примечания
Лекции	1-17 недели	0,1(10)	Оценивается отсутствие, пропуски занятий, активность студентов на лекции.
Лабораторные работы	1-17 недели	0,3(30)	17 лабораторных работ. Оценивается согласно критериям лабораторной работы
Курсовая работа	17 неделя	0,3(30)	Оценивается согласно критериям курсовой работы
Экзамен	сессия	0,3(30)	Оценивается согласно критериям экзамена
	Итого:	1 (100)	

Учебно-методическая карта дисциплины «Технологии программирования» для направления подготовки «Информатика и вычислительная техника»

на 6 семестр

График аудиторных занятий, СРС, текущих и промежуточных аттестаций

Наименование вида работ	Номер недели																
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1 Аудиторные занятия 68 час.																	
- лекции	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
- лабораторные занятия	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
- практические занятия																	
- семинарские занятия																	
2 Самостоятельная работа 35час.:																	
- подготовка к лекциям	постоянно																
- подготовка к лабораторным работам	постоянно																
Курсовой проект (КП)																	
Курсовая работа (КР)		ВР	КР	КР	КР	КР	КР	КР	КР	КР	КР	КР	КР	КР	КР	ЗР 0,3	КР
Расчётное задание (РЗ)																	
Реферат (Р)																	
Другие виды СРС (итоговая работа)																	
3 Формы текущей аттестации:																	
Коллоквиум (КЛ)																	
Контрольная работа (К)																	
Контрольный опрос (КО)																	
Защита лабораторной работы (ЗР)	зр	зр	зр	зр	зр	зр	зр	зр	зр	зр	зр	зр	зр	зр	зр	зр	Зр 0,3
Другие виды аттестации																	
Посещение занятий	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	ПЗ 0,1
4 Формы промежуточной аттестации																	
Зачёт	Не предусмотрен																
Экзамен	Во время сессии, вес - 0,3																

Методические рекомендации обучающимся по изучению дисциплины

При изучении дисциплины обучающийся должен рационально планировать свое рабочее время, отводить в своем рабочем графике время на проработку лекций, подготовку к семинарам и т.д. После каждого лекционного занятия обучающемуся необходимо детально проработать материал лекции, при необходимости обращаясь к основной и дополнительной литературе, периодическим изданиям, обращаясь за консультациями к преподавателю.

- Разъяснения по поводу работы с модульно-рейтинговой системой приведены в памятке для студента;
- Вопросы к тестам текущей и итоговой аттестации приведены ниже.
- Методические рекомендации по выполнению лабораторных и курсовых работ приведены в данном методическом указании;

Возможные задания для курсовой работы (6 семестр):

- Автоматизированная система управления «Школа»
- АСУ «ДОМ».
- АС «Библиотека».
- Интегрированная оболочка для разработки программ для школьников.
- Оболочка для разработки тестов и их проведения и т.п.

Задача каждого студента определяется постановкой общей задачи для группы и сетевым графиком работ.

Типовые вопросы, задаваемые студенту при сдаче курсовой работы:

1. Дать определение понятию «технология».
2. Что является основой для сжатия данных?
3. Охарактеризовать понятия: информация, основные свойства информации, основные формы представления информации.
4. Какие технологии обработки графической информации Вам известны?
5. В чём смысл технологии обработки мультимедийной информации?
6. Дать классификацию автоматизированных информационных систем.
7. Технологии обработки финансово-экономической, статистической информации.
8. Описать понятие информационной системы.
9. Как можно проанализировать данные с помощью географических карт?
10. Назовите основные структурные компоненты процесса обмена информацией.
11. Дать определение методологии обработки информации. Перечислить основные.
12. Перечислить основные информационные процессы.
13. В чём смысл объектно-ориентированного проектирования?
14. Дать классификацию технологических процессов.
15. Каковы Проблемы и перспективы развития технологических подходов.
16. Классифицировать технологические стадии.
17. В чём смысл объектно-ориентированного проектирования?
18. Какие процессы входят в набор классических?
19. Перечислить основные структурные компоненты процесса обмена информацией.
20. В чём смысл гибких (адаптивных) подходов?

21. Какие процессы входят в стандартный набор (ISO12207:1995)?
22. Чем характеризуются технологические подходы со слабой формализацией?
23. Объяснить методологии, технологии и инструментальные средства проектирования (CASE- средства).
24. Что такое информация? Её типы и виды.
25. Охарактеризовать классические технологические подходы к процессу обработки информации.
26. Дать классификацию технологических подходов к организации обработки различных видов информации.
27. Перечислить и раскрыть суть основных особенностей методологии RAD.

Типовые вопросы для проведения экзамена:

1 Визуальное программирование.
2 Генетические подходы.
3 Графические языки программирования.
4 Группа строгих технологических подходов.
5 Каскадно-возвратный подход программирования.
6 Каскадные технологические подходы.
7 Каскадный подход в программировании.
8 Классификация технологических процессов программирования.
9 Классификация технологических стадий разработки программы.
10 Классифицировать технологические подходы программирования.
11 Классические технологические подходы программирования.
12 Классический набор процессов проектирования и программирования.
13 Метод пошаговой детализации в структурном программировании.
14 Методологии, технологии и инструментальные средства проектирования (CASE- средства).
15 Методы описания алгоритма программы.
16 Методы отладки программы.
17 Методы преодоления барьеров между разработчиком и пользователями.
18 Объектно-ориентированное программирование.
19 Перечислить и раскрыть суть основных особенностей методологии RAD.
20 Подход «сверху вниз».
21 Подход «снизу вверх».
22 Подходы исследовательского программирования.
23 Подходы на основе формальных преобразований.
24 Подходы со слабой формализацией.
25 Понятие объектно-ориентированного программирования.
26 Постадийная разработка программы.
27 Проблемы и перспективы развития программирования.
28 Проблемы и перспективы развития технологических подходов.
29 Программирование и стандарты.
30 Связь между программированием и документированием.
31 Смысл объектно-ориентированного проектирования и программирования.
32 Событийное программирование.
33 Стандартный набор (ISO12207:1995) процессов разработки ПО.
34 Технологии обработки мультимедийной информации.

35 Технологические подходы быстрой разработки программ.
36 Технология программирования.
37 Технология стерильного цеха.
38 Характеристика гибких (адаптивных) подходов программирования.
39 Характеристика технологических подходов со слабой формализацией.
40 Цели структурного программирования.
41 Что такое технология программирования?
42 Эволюционное прототипирование.
43 Экстремальное программирование.

Шкала оценок и правила вычисления рейтинга

В АлтГТУ принята 100-балльная шкала оценок. Именно эти оценки учитываются при подсчете рейтингов. Традиционная шкала будет использоваться только в зачетных книжках. Соответствие оценок на лабораторных работах: 100 - 26 баллов – «зачтено», 25 и менее баллов – «не зачтено».

Соответствие оценок при защите курсовой работы: 75 баллов и выше – «отлично», 50-74 балла – «хорошо», 25-49 баллов – «удовлетворительно», менее 25 баллов – «неудовлетворительно».

Соответствие оценок на экзамене: 75 баллов и выше – «отлично», 50-74 балла – «хорошо», 25-49 баллов – «удовлетворительно», менее 25 баллов – «не удовлетворительно».

Успеваемость студента оценивается с помощью текущего рейтинга (во время внутрисеместровых аттестаций), семестрового рейтинга (перед сессией) и итогового рейтинга (после сессии). Во всех случаях рейтинг вычисляется по формуле: $R_0 = \frac{\sum R_i \cdot p_i}{\sum p_i}$, где R_i – оценка за i-ю контрольную точку. p_i – вес этой контрольной точки. Суммирование проводится по всем контрольным точкам с начала семестра до момента вычисления рейтинга.

Пример расчета рейтинга.

студент Иванов получил следующие оценки:

1 модуль: – 95 б. Пусть

2 модуль: – 80 б.

3 модуль: – 70 б.

К зачёту студент получил 85 баллов.

На 1-й аттестации (7 неделя) его текущий рейтинг равен:

$$R_{Oaё1} = \frac{95 \cdot 0,4}{0,4} = 95.$$

На 2-й аттестации (14 неделя):

$$R_{Oaё2} = \frac{(95+80) \cdot 0,4}{0,4 \cdot 2} = 87,5.$$

Перед началом сессии вычисляется семестровый рейтинг:

$$R_{Ñâi} = \frac{(95+80) \cdot 0,4 + 70 \cdot 0,2}{0,4 \cdot 2 + 0,2} = 84.$$

Итоговый рейтинг рассчитывается после экзамена по формуле:

$$R_{Итог} = R_{Сем} \cdot 0,8 + R_{Зач} \cdot 0,2 = 84 \cdot 0,8 + 70 \cdot 0,2 = 81.$$

В ведомость и зачетку выставляется оценка «зачтено» и восемьдесят один балл.

Примечание:

Если обучающийся не планирует «защищать» лабораторную работу, но выполняет её правильно, он может сдать преподавателю её в письменном виде, аккуратно оформленную. В этом случае максимальный балл составляет 60 баллов.

СПИСОК ЛИТЕРАТУРЫ

Основная литература

1. Метрология, стандартизация, сертификация в АСУ: [Текст]: Учеб. пособие / И.В. Баранников, А.В. Ландер.-М.: Изд-во Горного ун-та, 2008. - 91с.(10 экз.)
2. Терехов, А.Н. Технология программирования / А.Н. Терехов. - 2-е изд. - М.: Интернет-Университет Информационных Технологий, 2007. - 149 с. http://biblioclub.ru/index.php?page=book&id=233491&sr=1 (ЭР)

Дополнительная литература

3. Информационная технология. [текст]: Комплекс стандартов и руководящих документов на автоматизированные системы (ГОСТ 34). – М., 1991 (2 экз)
4. Ларина Н.А. Технологии программирования. Методические указания: [текст] / Н.А. Ларина. – Рубцовск: Рубцовский индустриальный институт, 2010. – 48 с. (30 экз.)
5. Норенков И.П. Основы автоматизированного проектирования: [текст]: Учеб. для вузов. – М.: Изд-во МГТУ им. Н.Э. Баумана, 2000. – 360 с.(5 экз.)
6. Основы современных компьютерных технологий: [Текст]: Учебник / Под ред. А.Д. Хомоненко. – СПб.: Корона-принт, 2005. – 672 с. (1 экз. каф.)
7. Острейковский В.А. Информатика: [Текст]: Учебник / В.А. Острейковский – М.: Высш. шк., 2009. – 511 с. (15 экз.)
8. Степанов А.Н. Информатика: [текст]: Учебник/ А.Н. Степанов. -6-е изд. - СПб.: Питер, 2010. - 720 с. (15 экз.)
9. Черепашков, А.А. Компьютерные технологии, моделирование и автоматизированные системы в машиностроении: Учебник [текст] / А.А. Черепашков, Н.В. Носов. – М.: Ин - Фолио, 2009. - 640 с. (3 экз.)

Программное обеспечение и Интернет-ресурсы

10. http://tests.specialist.ru/ – центр компьютерного обучения МГТУ им. Н.Э. Баумана.
11. http://www.microinform.ru/default.asp – учебный центр «Микроинформ» по компьютерным технологиям.
12. www.citforum.ru – учебный сайт по технике и новым технологиям.

Учебно-методические материалы и пособия для студентов, используемые при изучении дисциплины

13. Камаев В.В. Технологии программирования: Учебник / В.А. Камаев, В.В. Костерин. - 2-е изд., перераб. и доп. -М.: Высш. шк., 2006. - 454с. (15 экз.)
14. Логинов В.Н. Информационные технологии управления: [Текст]: Учеб. пособ. / В. Н. Логинов. – М.: Кнорус, 2010. – 240 с. (15 экз.)

Ларина Нина Александровна

ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ

Учебное пособие для бакалавров направления «Информатика и вычислительная техника»

Редактор Е.Ф. Изотова

Подписано к печати 06.10.2016г. Формат 60x84/16.
Усл. печ. л. 3,19. Тираж 40 экз. Заказ 161577. Рег. № 30.

Отпечатано в ИТО Рубцовского индустриального института
658207, Рубцовск, ул. Тракторная, 2/6.