Федеральное государственное бюджетное образовательное учреждение
высшего образования «Кемеровский государственный университет»
Новокузнецкий институт (филиал)

Факультет информатики, математики и экономики
Кафедра информатики и вычислительной техники им. В. К. Буторина

О. А. Штейнбрехер

Программная инженерия

Методические указания по выполнению практических работ дисциплине для обучающихся по направлению подготовки 02.03.03 Математическое обеспечение и администрирование информационных систем Профиль «Программное и математическое обеспечение информационных систем» и направлению подготовки 09.03.03 Прикладная информатика Профиль «Прикладная информатика в экономике»

Новокузнецк
2020

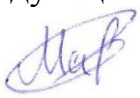
УДК [378.147.88:004.4](072)
ББК 74.484(2Рос-4Кем)я73+ 32.972я73
Ш88

Ш88 «Программная инженерия. Методические указания по выполнению практических работ : метод. указ (текст. электрон. изд.)/ О.А. Штейнбрехер ; Новокузнец. ин-т (фил.) Кемеров. гос. ун-та – Новокузнецк: НФИ КемГУ, 2020. – 21 с.

Приводятся методические указания по выполнению практических работ по разделу «Базовые понятия программной инженерии» дисциплины «Программная инженерия».

Методические указания предназначены для студентов всех форм обучения направлений 02.03.03 «Математическое обеспечение и администрирование информационных систем» и 09.03.03 «Прикладная информатика».

Рекомендовано
на заседании кафедры
информатики и вычислительной
техники им. В. К. Буторина
23 ноября 2020 года.
Заведующий кафедрой



А. В. Маркидонов

Утверждено
методической комиссией факультета
информатики, математики и экономики
17 декабря 2020 года.
Председатель методкомиссии

Г.Н. Бойченко

УДК [378.147.88:004.4](072)
ББК 74.484(2Рос-4Кем)я73+ 32.972я7

Текст представлен в авторской редакции

Оглавление

Введение

Дисциплина «Программная инженерия» относится к основной части образовательной программ 02.03.03 Математическое обеспечение и администрирование информационных систем и 09.03.03 Прикладная информатика. В ходе освоения дисциплины, обучающиеся получают и закрепляют навыки проектирования, разработки и создания программных систем, программных приложений и комплексов.

Входными требованиями для освоения дисциплины являются: знания основных понятий алгоритмизации, принципов работы программных приложений, классификации языков программирования, умения составлять алгоритмы для решения прикладных профессиональных задач, навыки программирования алгоритмов для решения прикладных профессиональных задач, обоснованного выбора языков программирования, методов и алгоритмов для решения профессиональных задач.

В результате освоения дисциплины «Программная инженерия» обучающийся должен сформировать следующие навыки.

Знать:

- понятие и модели жизненного цикла программных систем;
- принципы и методы программной инженерии, реинженерии, реверсной инженерии и рефакторинга применительно к программным системам;
- технологии, парадигмы и шаблоны проектирования и программирования.

Уметь:

- выбирать и обосновывать технологии, методы и шаблоны проектирования и программирования на всех стадиях жизненного цикла;
- применять CASE-средства проектирования и программирования на всех стадиях жизненного цикла.

Владеть:

- методами, технологиями и парадигмами проектирования и программирования для создания программных систем;
- методами обеспечения и оценки качества программных систем.

Данные методические рекомендации охватывают практические задания по разделу дисциплины «Базовые понятия программной инженерии».

Краткие теоретические сведения

Основные понятия программной инженерии

Основные понятия ПИ: программа во всех ее проявлениях (состояниях) на этапах ЖЦ, а также методы, средства и инструменты ее изготовления.

Программа – это объект разработки, который не является осязаемым (нельзя пощупать, взвесить и т.п.) человеком, а доступен пониманию ЭВМ, для которой написан. Готовая программа – это программный продукт, реализующий определённые функции (задачи) предметной области (ПрО), процесс проектирования и разработка которого осуществляется соответствующими программными методами, средствами и инструментами ТП.

Объектами разработки могут быть: модуль, программа, комплекс программ, пакет прикладных программ, система и др., т.е. объект либо сам является отдельной конструктивной единицей разработки, реализующей элементарную функцию ПрО, либо состоит из их взаимосвязанного набора.

Для объекта разработки первоначально формируется его модель, которой специфицируются реализуемые функции и элементы данных, устанавливаются их связи и отношения. Иными словами, для объекта определяется его спецификация или прототип.

Любой объект имеет начальное (исходное), промежуточное и конечное состояние. Начальное состояние – это исходная модель объекта. Промежуточное состояние – это измененное состояние, отличное от начального и конечного состояний объекта, полученное на определенном этапе программных методов и средств. Промежуточным состоянием объекта, например, является эскизный, технический и рабочий проекты.

Конечное состояние объекта – это программный продукт, готовый для исполнения требуемых от него функций. Конечным состоянием считается опытный образец, который передается в опытную эксплуатацию либо в тираж.

Метод разработки (программный метод) – это способ или планомерный подход к достижению той цели, которая становится перед объектом разработки. Наиболее распространенные методы проектирования и разработки следующие: нисходящий, восходящий, модульный, метод расширения ядра, метод сборочного программирования и др. Метод определяет стратегию проектирования и разработки программ и систем.

Технологический процесс – это взаимосвязанная последовательность операций, выполняемых при разработке объекта. Процесс предназначен для перевода объекта из одного состояния в другой соответствующими программными методами и средствами.

Для реализации набора типовых функций автоматизируемой ПрО, относящейся к СОД, АСНИ, САПР и др., используются типовые технологические процессы (ТП), которые вместе со специализированными образуют линию программ в технологии функциональноориентированного типа.

Технологическая линия (ТЛ) задает набор процессов разработки функций объекта, представленных совокупностью автоматизированных операций, которые последовательно и систематически преобразуют состояния объектов, включая заключительное его состояние – готовый программный продукт. Особенность линии – отражение определенных функций ПрО, реализуемых в виде программ с заданными показателями качества.

Для простых программ количество процессов в ТЛ меньше, чем для сложных. Независимо от сложности программного объекта основным условием выполнения процессов является их автоматизация.

Процессы имеют модельное представление, базирующееся на модели жизненного цикла программного объекта разработки.

Динамика изменения объекта может быть представлена в ТЛ последовательностью процессов, переводящих объект из одного состояния в другой путем использования средств автоматизации операций процесса. В

этом плане каждая конкретно разработанная ТЛ определяет множество допустимых состояний объектов, а на средства автоматизации возлагается функция перевода и слежения за переходом в недопустимые состояния.

Инструмент – это программное, язычное или методическое средство, применяемое для получения состояния объекта в некотором законченном виде. В зависимости от этапа жизненного цикла и состояния объекта для работы с ним могут быть языки, трансляторы, генераторы и т. п.

Процесс преобразования модели объекта из одного состояния в другое не является полностью автоматизированным, и имеется ряд систем, специально предназначенных для реализации конкретных предметных областей.

Управление разработкой. Каждая инженерная дисциплина базируется на принципах и методах конструирования (разработки) и промышленного производства продуктов, которые затрагивают как организационные, так и технические аспекты производства. Основными вопросами управления разработкой программных объектов как инженерии ведения разработки являются:

1) организация коллектива разработчиков (состав, структура, квалификация и др.);

2) планирование работ и трудозатрат и обеспечение роста производительности труда;

3) контроль хода разработки и оценка проектных решений в ходе разработки программных продуктов;

4) экономические вопросы (стоимость, ценообразование, стимулирование и др.);

5) управление качеством.

Жизненный цикл (ЖЦ) – это совокупность последовательных всех действий и состояний ПС, связанных с превращением программных систем (ПС) в продукцию производственнотехнического назначения, начиная с

анализа потребностей в автоматизации функций ПрО до создания ПС и кончая его моральным износом.

С ЖЦ связаны сроки и период разработки и эксплуатации ПС.

По срокам эксплуатации ПС разделяются на два класса:

1) с краткосрочной эксплуатацией;

2) с долгосрочной эксплуатацией.

Принципы программной инженерии

К принципам программной инженерии отнесены: принцип производственной организации, принцип обеспечения технологичности и, принцип планирования трудозатрат.

Принцип производственной организации. В отличие от творческого характера работ в процессе ТПР при определении ТЛ (анализ ПрО, выбор подходящих средств описания ПрО, корректировка и выбор ограничений на проект и т. д.) инженерные методы, используемые в процессе прикладного программирования, отличаются строгой последовательностью ТП и ТО, оформленных в производственные процессы. В них имеется специализация операций, применяемых методов и инструментов, а также исполнителей, деятельность которых учитывается в соответствии с экономическими методами оценки труда. Необходимым условием успешного использования данного принципа является планирование программных работ, управление которыми осуществляет технологическая служба предприятия, выполняющая контроль соблюдения технологии и оценку объекта разработки, планирование работ с элементами нормирования, совершенствование технологических процессов и др.

Принцип планирования. Для соблюдения сроков разработки программного объекта во многих проектах проводится планирование. Существуют различные методики оценки трудоемкости проекта. Большинство методов основано на прогнозировании объема продукта, выражаемого в числе строк (операторов, команд). Например, статистическая модель оценки трудоемкости COCOMO. На рисунке 1 представлены оценки

количества строк программного кода (LOK) на функциональную точку в зависимости от языка программирования. Обычно при расчетах учитывают оптимистические, пессимистические и реалистичные варианты исполнения.

№	Язык программирования	Ассемблер LOC	Показатель LOC на Функциональную точку
1	Basic Assembler	1	320
2	Macro Assembler	1,5	213
3	C++	6	53
4	Java	6	53
5	Oracle, Sybase	8	40
6	Oracle Developer/2000	14	23
7	Cobra	16	20
8	HTML 3,0	22	15
9	SQL (ANSI)	25	13

Рисунок 1 – Показатель LOK на функциональную точку

Предполагаемый объем делится на среднестатистическое значение производительности (число строк) одного программиста в год. В результате получалась планируемая трудоемкость.

В производственных условиях проводится текущее планирование работ, при котором решается задача составления выполнимого (достижимого) плана работ Y по ТЛ, основными данными для составления которого являются:

1) общий плановый срок разработки $[t_0, T]$, где t_0 и T – начальный и конечный сроки разработки;

2) объем работ $W = \{W_i\}$ с учетом переделок;

3) требуемые ресурсы $R = \{R_l, R_m\}$, где R_l – человеческие; R_m – материальные (технические и программные);

4) нормы потребления человеческих ресурсов по всем ТП, $(i=1..N)$, NR_l и др.

Формально план работ записывается в следующем общем виде:

$$Y = Y(t_0, T, W, R_l, R_m, NR_l, f),$$

где f – случайные факторы (ошибки при выполнении плановых работ на ТП, сбои технических средств и др.), а также факторы, связанные с появлением средств новой техники и программного обеспечения.

В качестве механизма управления разработкой может использоваться сетевой план-график работ и контроль его выполнения. В условиях производственной организации ведения разработки на основе ТЛ отсутствие его зачастую приводит к неуправляемости процесса.

Принцип обеспечения технологичности. Понятие технологичности целиком и полностью связано с наличием технологии и с полным соблюдением всех требований и правил. Технологичность – это понятие, включающее в себя технологичность ПП и технологичность процесса разработки. Технологичность ПП – это соответствие ПП потребительским возможностям и определенным функциям ПрО. Ее обеспечение основывается на заложенных в ТЛ элементах типизации, унификации и стандартизации конструктивных элементов ПП, применяемых моделях и заготовках, а также готовых повторно используемых программных объектах из фондов коллективного пользования. Технологичность определяет подготовку ПП к эксплуатации.

Технологичность разработки – это регламентированный (упорядоченный набор процессов, операций и процедур их выполнения), конструктивный (методом сборки из готового) и инструктивный (методическое и инструктивное обеспечение процессов ТЛ) порядок работы, а также организация управления разработкой ПП. Ее обеспечение основывается на применении эффективных методов ведения разработки ПП, воплощенных в ТЛ (или ТП), направленных на повышение качества и производительности труда, минимизацию затрат и времени на разработку ПП.

Принцип планирования трудозатрат. Исходя из результатов исследований можно заключить, что основное распределение трудозатрат в процессах разработки приходится на сопровождение и поддержку проекта.

Поэтому работы по планированию и нормированию в условиях производственной организации являются актуальными.

Модель жизненного цикла

Модель жизненного цикла ПС. Процесс разработки ПС определяется моделью ЖЦ, этапы которой будем отождествлять с ТП. Для каждого ТП рассматриваемого ПС фиксируются его начальное (S_0), и промежуточные (S_j) и конечное (S_R) состояния. Переход объекта из состояния S_j в S_{j+1} из ТП $_j$ обеспечивается выполнением операций ТО, входящих в этот ТП. При этом для каждого типа ПС может быть свой набор ТП и состояний S, определяемых на соответствующем множестве исходных данных, характеризующих эти состояния.

Основная цель выделяемых $ТП_i \subset$ ТЛ состоит в получении некоторого полуфабриката ПП (S_i – состояние ПС), фрагментов ЭПД для проведения экспертизы уровня и качества в соответствии с моделью качества $М_{кач}$. Каждый ТП модели ЖЦ в общем виде определяет состояние элементов ПС, состав технологических операций, обеспечивающих преобразование исходного состояния ПС и получение его конечного состояния. Таким образом, общая технологическая модель (схема) процесса разработки является отражением модели ЖЦ, способов преобразования состояний ПС и представлена в виде:

$$M_{пр} = (S, ТП_i, (ТО_j, ТМ_j), i = 0\ldots7, j = 1\ldots k.$$

Множество состояний S рассматриваемой модели включает в себя: S_0 – исходное (начальное) состояние – описание требований заказчика, предъявляемых к ПС ; S_1 – состояние, включающее в себя набор элементарных состояний, а именно, описаний функций (S_1^1), архитектуры (S_1^2), структуры данных (S_1^3) и т. п. средствами языков спецификаций L_1; S_2 – состояние соответствует техническому проекту и включает в себя описание в классе языков L_2 алгоритмов функции (S_2^1), данных (S_2^2), интерфейсов (S_2^3), гипертекстов документации (S_2^4), оценочных элементов модели качества (S_2^5) и др .; S_3 – состояние соответствует рабочему проекту и включает в себя

описание в ЯП (класс L_3) текстов программ (S_3^1), модулей (S_3^2), тестов (S_3^3) и т.п.; S_4 – состояние соответствует отлаженным элементам программного продукта; S_5 – состояние ПС после сборки ; S_6 – состояние ПС, соответствующее программному продукту, которое испытывается, проверяется на соответствие заданным функциям и значениям показателей качества ; S_7 – состояние ПС в процессе сопровождения.

Технологический процесс, входящий в состав $M_{пр}$ – промежуточный (частичный) процесс, осуществляет преобразование S_i-го состояния ПС с помощью набора $TO_j \subset ТП_i$ данной $M_{пр}$, поддерживаемых TM_j (либо один ТМ реализует несколько ТО).

Набор операций $M_{пр}$ не является фиксированным, он уточняется для каждого типа ПС и описывается в технологических документах. Среди операций могут быть типовые, используемые готовыми при создании конкретной технологии разработки ПС определенного типа. Для обеспечения технологичности разработки в их состав входят операции управления качеством разработки и формирования документации на всех процессах ЖЦ в соответствии с системой моделей документации $M_{эпд}$, определенной выше.

На рисунке 2 представлена модель общего вида частичного процесса.

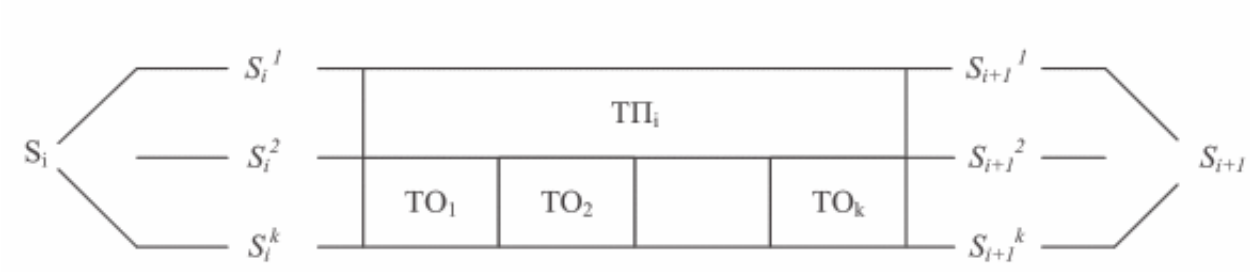


Рисунок 2 – Модель частичного технологического процесса $ТП_i$

Состояние объекта S_i определяется набором частичных его состояний S_i^1, ..., S_i^k, подаваемых на вход $ТП_i = \{TO_k\}$. Данный набор для конкретного ПС конечен.

Управление процессом преобразования состояний объекта S в S_k может быть задано в виде графовой модели (соответствует технологическому

маршруту), в вершинах которой находятся процессы (или операции), а ребра задают всевозможные переходы из одного состояния объекта в другое. Графовая модель (рисунок 3) отображает способ ведения разработки с распараллеливанием работ и возвратами (при ошибках) в предыдущие процессы.

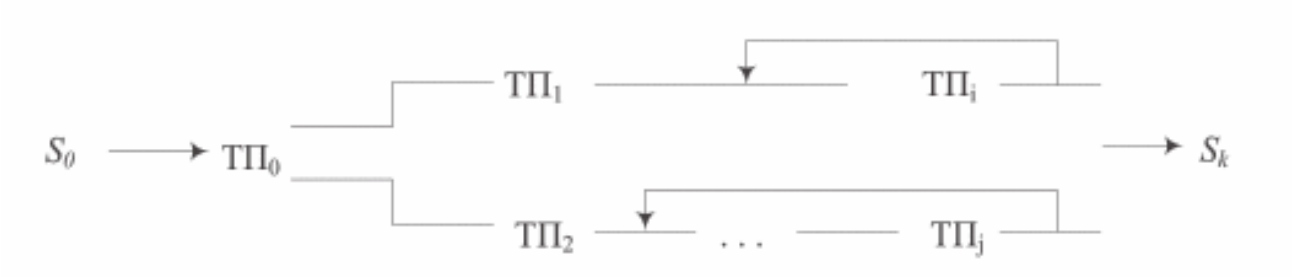


Рисунок 3 – Графовая модель технологического процесса

Каждый частичный процесс ТП, является абстрактным конечным автоматом, граф которого совпадает с технологическим маршрутом процесса, множество состояний S_i которого определено на совокупности операций $\{ТО_i\} \subset ТО$ данного процесса.

Переход объекта из состояния S_i в состояние S_{i+1} может быть только под действием технологического маршрута и осуществляется технологическим диспетчером посредством выбора из множества операций процесса, зафиксированных в карте i-го процесса, очередной операции при условии, что результат предыдущей операции проанализирован и выполнен.

Задача выполнения i-го процесса заключается в переводе автомата из некоторого промежуточного состояния объекта в другое S_{i+1} с выполнением операций преобразования, качественной или количественной оценки результата разработки объекта и формирования фрагментов ЭПД по $М_{ЭПД}$.

Реинжения, реверсная инженерия, рефакторинг

Методы систематического изменения отдельных элементов программ (модулей, компонентов, КПИ и др.) и систем программного обеспечения образуют концепцию реинженерии и реверсной инженерии, сформулированных в стандарте программной инженерии. Сущность

изменений в программных элементах и системам лежит в плоскости улучшения функциональности и повышения их качества. С помощью изменений в некотором смысле продлевается время жизни действующих, стареющих и наследуемых программ и ресурсов многоразового использования.

С общей точки зрения эволюционное развитие систем обеспечивается внешними и внутренними методами изменения компонентов, интерфейсов и/или систем. К внешним методам относятся: добавление, объединение, удаление отдельных элементов ПС. К внутренним методам – методы реинженерии, реверсной инженерии и рефакторинга. С их помощью проводится целенаправленное изменение программ или систем разного назначения.

Реинженерия – это совокупность моделей, методов и процессов изменения структуры, функций элементов ПС для достижения качества функциональности и добавления новых функций на основе требований пользователей.

Метод реинженерии используется при частичной или полной переделке отдельных функций, компонентов, некоторых фрагментов старых программ на новые при переходе к новым условиям ОС, платформ и требований к качеству продукта или процессов ЖЦ. При этом решается задача обеспечения мобильности ПС, т. е. способности ПС адаптироваться к новой платформе. Такие изменения могут выполняться путем:

1) реорганизации спецификаций и характеристик отдельных ресурсов;

2) модификации или модернизации функций ресурсов и данных;

3) определения способов взаимодействия ресурсов между собой, которые могут располагаться в разных средах и др.

При этом структура ПС может оставаться неизменной, если модификация касается только отдельной функции системы или замены ее другой.

Если в системе изменяется много функций, то процесс изменений является дорогостоящим и связан с большим риском достижения требуемого результата.

С помощью реинженерии совершенствуется системная архитектура, создается новая документация и конфигурация и тем самым обеспечивается продолжение времени жизни наследуемой, изменяемой системы.

Процессы реинженерии применяются и при изменении деловых процессов в целях уменьшения лишних видов деятельности или повышения их эффективности. Зависимость процесса использования наследуемой системы будет минимальной, если заранее будут спланированы возможные изменения в системе.

К основным процессам реинженерии относятся:

1) перевод отдельных компонентов в ЯП в новую версию в этом или в другом языке;

2) анализ документов описания структуры и функциональных возможностей системы для изменения проектных решений;

3) декомпозиция системы на более мелкие компоненты для проведения модификации и снижения сложности большой по размеру системы;

4) изменение данных, с которыми работают компоненты системы.

Преобразование исходного кода программ – это наиболее простой способ реинженерии, особенно, если оно может проводиться автоматизировано.

Если конструкции ЯП этой системы имеют различия со структурами программ и типами данных нового языка описания ПС, то может потребоваться ручное преобразование элементов ПС. На что затрачиваются значительные технические, человеческие и финансовые ресурсы.

В процессе разработки системы создается комплексная модель, по которой можно выбрать варианты реализации и модификации ПС с учетом требований к защите, безопасности и др.

Таким образом, результатом реинженерии ПС является измененная система с новыми компонентами, усовершенствованными компонентами и соответствующими связями для среды функционирования.

Реверсная инженерия – анализ и рассмотрение структуры системы и ее элементов для построения нового ПП путем обновления и восстановления системы, полученной прямой инженерией. Другими словами, изучение элементов структуры системы в целях построения нового варианта системы сводится к восстановлению исходной спецификации, ее структуры и логики компонентов, а также выбору метода перепрограммирования системы.

Сложилось два типа задач реверсной инженерии. Задачи первого типа включают в себя:

1) анализ системы для проведения изменений в структуре кода;

2) расширение функциональности ПС;

3) замену платформы, ЯП и т.п.;

4) изменение логической структуры системы и применение шаблонов проектирования для перепрограммирования;

5) изменение моделей и структур данных.

Задачи второго типа состоят в восстановлении:

1) структуры системы и компонентов, а также в выборе подходящего ЯП;

2) расширения видов интерфейсов и форматов данных для организации вычислений.

Развитию реверсной инженерии послужили переход к ООП, новые способы визуализации, измерения метрик (metric) компонентов ПС и выполнения задач:

1) обеспечение высокого качества системы, переоценка сроков, объемов, сложности и структуры разрабатываемой системы;

2) определение иерархии классов и атрибутов программных объектов в целях наследования их в новой системе;

3) идентификация классов объектов, поиск и выбор паттернов для их идентификации и фиксации их места в структуре системы.

Рефактороринг – перестройка, улучшение внутренней структуры программного кода с сохранением функциональности.

Метод рефакторинга ориентирован на изменение (замену, замещение, расширение) реализаций компонентов и их интерфейсов с учетом требований и плану конфигурации компонентов.

Каждая операция рефакторинга является базовой, атомарной функцией преобразования, которая сохраняет целостность компонента, т. е. совокупность правил, ограничений и зависимостей между составными элементами компонента, задающих компонент как единую и цельную структуру. Задача рефакторинга – сохранить функции и идентичность компонента. Добавление новых функций, характеристик и свойств продукту проводится с помощью реинженерии.

К операциям рефакторинга отнесены: добавление компонента и измененного интерфейса; замена существующей реализации компонента новой эквивалентной функциональностью и новым интерфейсом; добавление нового интерфейса к старому компоненту и др.

Используя базовый набор этих операций, формируется измененная ПС с заданной функциональностью.

Основным условием рефакторинга является обеспечение целостности компонентов и системы с сохранением свойств и характеристик объектов.

Методы сборки

Одна из главных задач современного программирования – создание теоретических и прикладных основ построения сложных программ из более простых программных элементов, которые записаны на современных языках программирования. Фактически решение этой задачи осуществляется путем объединения или интеграции разнородных программных ресурсов, включая модули, и простые программы реализации функций некоторой предметной области.

Цель метода сборки – интеграция готовых ресурсов в новые программные структуры программ или систем. Объекты сборочного программирования – готовые модули. Готовые модули и другие ресурсы накапливаются в библиотеках, в фондах алгоритмов и программ и так далее.

Метод сборки – способ соединения разноязычных объектов, основанный на теории спецификации и отображения типов и структур данных языков программирования, представленных алгебраическими системами. Сборочное программирование характеризуется построением программ из готовых «деталей» (программных ресурсов разной степени сложности) и повторным использованием программных средств. Процесс сборки ресурсов характеризуется: комплектующими деталями и узлами, схемой сборки (взаимодействия отдельных компонентов и правилами взаимодействия), технологией сборочного конвейера. Комплектующие метода – это программные элементы (модули, объекты, компоненты, сервисы, аспекты и др.).

Модуль – независимая функциональная часть программы, к которой можно обращаться как к самостоятельной единице через внешний интерфейс. Модуль – это логически законченная часть программы, выполняющая определенную функцию, обладающая свойствами завершенности, повторного использования и др. Модуль возник как обобщение понятия стандартных подпрограмм и процедур.

Объект – базовое понятие в ООП, которое имеет свойства наследования, инкапсуляции и полиморфизма. Объект — элементарная сущность, описываемая определенными свойствами (хранящимися в виде атрибутов объекта) и поведением (реализованным в виде методов). Объекты взаимодействуют между собой через сообщения и запросы.

Инкапсуляция - свойство системы, позволяющее объединить данные и методы, работающие с ними, в классе (свойство языка программирования, позволяющее пользователю не задумываться о сложности реализации используемого программного компонента, а взаимодействовать с ним

посредством предоставляемого интерфейса, а также объединить и защитить жизненно важные для компонента данные; при этом пользователю предоставляется только спецификация (интерфейс) объекта).

Наследование – позволяет создать новый класс на основе уже существующего, частично или полностью заимствуя его функциональность (позволяет описать новый класс на основе уже существующего (родительского), при этом свойства и функциональность родительского класса заимствуются новым классом).

Полиморфизм – использование объектов с одинаковым интерфейсом без информации о типе и внутренней структуре объекта (возможность объектов с одинаковой спецификацией иметь различную реализацию).

Абстракция - в объектно-ориентированном программировании это придание объекту характеристик, которые отличают его от всех объектов, четко определяя его концептуальные границы.

Класс описывает структуру свойств и поведения одного типа объектов. Каждый объект программы является экземпляром некоторого класса.

Компонент – программный объект, который реализует некоторую функциональность и является базовым понятием компонентного программирования и компонентно-ориентированной разработки. Основная форма представления компонента – каркас и контейнер. Каркас – высокоуровневая абстракция, в которой функции отделены от задач управления.

Сервис – это программный ресурс, который реализует некоторую системную функцию. Содержит независимый интерфейс с другими сервисами и ресурсами. Совокупность взаимодействующих сервисов, веб-сервисов и их интерфейсов образует сервисно-ориентированную архитектуру SOA.

Контейнер – оболочка, внутри которой реализованы функции в виде экземпляров компонентов, обеспечивает взаимодействие с сервером через

стандартные интерфейсы. Экземпляры обращаются друг к другу через системные сервисы данного контейнера или другого.

Межмодульный интерфейс – это интерфейсный модульпосредник между двумя взаимодействующими программными объектами, выполняет функции передачи и приема данных между ними. Впервые разработан язык определения интерфейсов (ЯОИ) для описания операторов вызова модулей, параметров и их типов, а также операций проверки правильности обмена данными.

Межъязычный интерфейс – совокупность средств и методов преобразования структур и типов данных между ЯП с помощью алгебраических систем и функций (и макроопределений) библиотеки интерфейса для взаимно однозначного обмена данными между разноязычными модулями через механизмы интерфейса.

Схема сборки. Под схемой сборки понимается схема взаимодействия программных ресурсов, которая определяется непосредственными обращениями к отдельным из них или последовательности их выполнения. При этом взаимодействие каждой пары объектов зависит от использования общих данных. В общем случае схеме сборки соответствует совокупность моделей, которые отображают разные типы связей между компонентами: передача управления, обмен информацией, условия общего функционирования и т. д.

Усовершенствованный метод сборки применительно к процессам программирования отдельных элементов ПС направлен на сборку специализированных технологий программирования из процессов и способов их нотаций для реализации конкретной предметной области. К настоящему времени создан стандарт спецификации процессов – язык BPMN и система реализации описаний процессов в этом языке.

К основным элементам технологической подготовки разработки (ТПР) отнесены:

1) объект разработки (его начальное, промежуточные и конечное состояния);

2) методы программирования, средства и инструменты, обеспечивающие изменение состояний объекта

3) модели технологических процессов (ТП) и ТЛ;

4) инженерные методы управления процессами разработки программ по ТЛ.

Для новых ТП и ТЛ в рамках ТПР определены классы объектов, которые разделяются на понятийные, технологические и инструментальные. В класс понятийных объектов входят модели данных, задач и программ. Задачи разбиваются на классы функций, каждой из которых определяется подмножество схем данных и их программ элементов (заготовок). В классе типовых задач определяются модели типовых и специализированных программ реализации функций ПрО.

К классу технологических объектов отнесены модели формализованного представления состояний объектов и процессов их разработки. Это система моделей для фиксации проектных решений в ходе разработки; модели процессов и линий для формализации деятельности исполнителей; модель качества программных объектов для управления качеством разработки на всех этапах ЖЦ; модель эксплуатационных документов для формирования рабочей документации в ходе разработки ПП по заданной технологии.

Определены этапы жизненного цикла ПС и дано определение восьми основным этапам жизни отдельных объектов ПС, каждый из которых отображается в одном или нескольких ТП линии. Каждый процесс ТП трактуется как вероятностный автомат, переводящий объект разработки из одного состояния в другое. Заключительным состоянием является готовый программный продукт.

К классу инструментальных объектов относятся методы программирования, средства и инструменты, из которых подбираются более

подходящие для создаваемой ТЛ. С их помощью осуществляется целенаправленное преобразование состояний объекта разработки на каждом ТП.

На рисунке 4 представлена схема вызовов модулей через интерфейсы. На схеме представлена программа С, содержащая два вызова Call A() и Call B() с параметрами. Вызовы используют интерфейсные модули-посредники A’ и B’, которые осуществляют функции преобразования данных и их передачу модулям A и B. После выполнения модулей A и B результаты преобразуются обратно к виду программу С. Если типы данных параметров – не релевантные, то в функции посредника входит прямое обратное их преобразование.

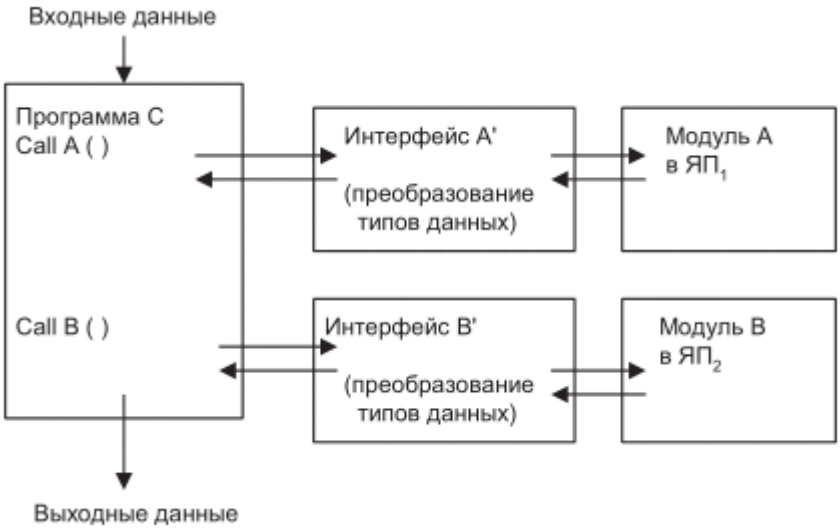


Рисунок 4 – Схема вызова модулей

Процесс сборки объектов может проводиться ручным, автоматизированным и автоматическим способами. Как правило, последний способ невозможен, связан с недостаточно формальным определением программных КПИ и их интерфейсов. Ручной способ нецелесообразен, так как сборка готовых КПИ представляет собой большой объем действий. Наиболее приемлемый способ – автоматизированная сборка, когда по заданным спецификациям программ осуществляется сборка с помощью стандартных правил сборки разнородных объектов.

Практическая работа №1. Выбор методов разработки. Составление моделей

Цель работы: получить навыки описания предметной области; изучить отличия методов разработки.

Задание.

1. Составьте описание определений, понятий и свойств предметной области.
2. Составьте спецификации на языке предметной области (неправомерные соединения свойств, зависимости, установки).
3. Определите метод разработки приложения: нисходящий, восходящий, модульный, метод расширения ядра, метод сборочного программирования и др. Для обоснования используйте свойства предметной области, срок работы приложения и другие критерии.

Примеры вариантов (предметная область).

1. Оптимальное распределение запасов
2. Построение кратчайшего маршрута
3. Учет транспортных расходов
4. Работа билетной кассы
5. Работа пропускного пункта

Ход работы.

1. Выбрать вариант работы или предложить собственный (по согласованию).
2. Определить источники для описания предметной области.
3. Составить описание (модель) предметной области
4. Определить свойства программного приложения и критерии выбора методов проектирования. Представить обоснование выбора метода проектирования.
5. Составить отчет

Практическая работа №2. Построение моделей жизненного цикла

Цель работы: получить навыки построения модели жизненного цикла.

Задание

1. Определить начальное (S_0), промежуточные (S_j) и конечное (S_R) состояния технологического процесса (жизненного цикла программного средства).
2. Определить операции, обеспечивающие переход из одного состояния в другое.
3. Составить общую технологическую модель, включающую состояния, технологические процессы и технологические процессы.
4. Построить графовую модель преобразования состояний объекта.
5. Построить технологический маршрут процесса (абстрактный конечный автомат) частичного технологического процесса.

Ход работы

1. Определить вариант работы (на основе варианта практической 1 или отличного от нее). За основу построение модели жизненного цикла взять определенный в работе 1 тип жизненного цикла: долгосрочный или краткосрочный, источники требований, последовательности и наличие этапов проектирования в выбранном методе проектирования и т.д.
2. Определить начальные, промежуточные и конечные состояния, например: S_0 – исходное (начальное) состояние – описание требований заказчика, предъявляемых к ПС ; S_1 – состояние, включающее в себя набор элементарных состояний, а именно, описаний функций ($S_1^{\,1}$), архитектуры ($S_1^{\,2}$), структуры данных ($S_1^{\,3}$) и т. п. средствами языков спецификаций L_1; S_2 – состояние соответствует техническому проекту и включает в себя описание в классе языков L_2 алгоритмов функции (S_2

^{1}), данных (S_2^{2}), интерфейсов (S_2^{3}), гипертекстов документации (S_2^{4}), оценочных элементов модели качества (S_2^{5}) и др .; S_3 – состояние соответствует рабочему проекту и включает в себя описание в ЯП (класс L_3) текстов программ (S_3^{1}), модулей (S_3^{2}), тестов (S_3^{3}) и т.п.; S_4 – состояние соответствует отлаженным элементам программного продукта; S_5 – состояние ПС после сборки ; S_6 – состояние ПС, соответствующее программному продукту, которое испытывается, проверяется на соответствие заданным функциям и значениям показателей качества ; S_7 – состояние ПС в процессе сопровождения.

3. Составить описание состояний, технологических процессов и операций для построения модели.
4. Построить граф преобразования состояний объекта.
5. Построить технологический маршрут процесса.

Практическая работа №3. Задачи реверсной инженерии, реинженерии

Цель: получение навыков решения задач прямой и реверсной инженерии, обоснованного выбора языка программирования, параметров и свойств разрабатываемого программной системы.

Задание.

1. Для имеющейся системы решить задачи реверсной инженерии первого типа – проанализировать функциональность программной системы, для расширения функциональности, проанализировать возможность и необходимость замены платформы или языка программирования, замену структуры данных.
2. Решить задачи реверсной инженерии второго типа (при анализе возможности замены языка программирования) – выбрать подходящий язык программирования, форматы данных, выбрать паттерны проектирования и определить их место в структуре системы.
3. На основе методов реинженерии провести анализ первоначальных процессов предметной области и проанализировать изменения, вносимые в нее с учетом автоматизации процессов, разрабатываемым (изменяемым) программным приложением.

Ход работы.

1. Для рассматриваемой предметной области выбрать программную систему (согласно первоначальному варианту в работе 1 или для любой программной системы, реализованной или рассмотренной обучающимся в ходе предыдущих дисциплин). Обучающиеся могут обменяться программными системами.
2. Проанализировать функциональность системы, определить возможность расширения.

3. Определить критерии выбора языка программирования для программной системы. Составить описание языков программирования и их свойств. Определить исходя из методов реверсной инженерии язык программирования для приложения.
4. Определить изменения модели и структуры данных.
5. Составить описание изменений процессов (реверсная инженерия).
6. Составить отчет.

Практическая работа №4. Технология сборки

Цель: получение навыков сборки приложений из готовых модулей – библиотек, подпрограмм, алгоритмов.

Задание.

1. Реализовать калькулятор для работы с комплексными числами (использующее библиотеку для работы с комплексными числами). Калькулятор должен позволять проводить все виды арифметических операций над комплексными числами, вычислять мнимые корни квадратичных уравнений и т.д.
2. Реализовать динамическую библиотеку (dll), содержащую методы сортировки массивов. Реализовать приложение, использующее эту библиотеку. Приложение и библиотека должны быть написаны на разных языках программирования (например, C++ и C#).
3. Используя парадигмы и стили модульного (сборочного) программирования, создайте программное приложение-сборку, где модули будут написаны в разных стилях/на разных языках. Совместная работа модулей должна быть организована через отдельный модуль-посредник. Составьте инструкцию и документацию, уделив внимание использованию модульной парадигмы. Тематика работы на выбор обучающихся.

Ход работы.

1. Реализовать приложения, согласно заданиям. Код приложения должен сопровождаться комментариями.
2. Оформить тестовые примеры и результаты работы приложения в виде отчета.
3. Составить инструкцию пользователя. Описать ограничения и другие особенности программного приложения.

Список рекомендуемой литературы

Лаврищева, Е. М. Программная инженерия. Парадигмы, технологии и case-средства: учебник для вузов / Е. М. Лаврищева. — 2-е изд., испр. — Москва : Издательство Юрайт, 2019. — 280 с. — (Университеты России). — ISBN 978-5-534-01056-5. — Текст : электронный // ЭБС Юрайт [сайт]. — URL: https://biblio-online.ru/bcode/444952 (дата обращения: 02.12.2019)

Черткова, Е. А. Программная инженерия. Визуальное моделирование программных систем : учебник для академического бакалавриата / Е. А. Черткова. — 2-е изд., испр. и доп. — Москва : Издательство Юрайт, 2019. — 147 с. — (Бакалавр. Академический курс). — ISBN 978-5-534-09172-4. — Текст : электронный // ЭБС Юрайт [сайт]. — URL: https://biblio-online.ru/bcode/437536 (дата обращения: 02.12.2019).

Лаврищева, Е. М. Программная инженерия и технологии программирования сложных систем : учебник для вузов / Е. М. Лаврищева. — 2-е изд., испр. и доп. — Москва : Издательство Юрайт, 2019. — 432 с. — (Бакалавр. Академический курс). — ISBN 978-5-534-07604-2. — Текст : электронный // ЭБС Юрайт [сайт]. — URL: https://biblio-online.ru/bcode/436514 (дата обращения: 02.12.2019).

Рыбальченко, М. В. Архитектура информационных систем : учебное пособие для вузов / М. В. Рыбальченко. — Москва : Издательство Юрайт, 2019. — 91 с. — (Университеты России). — ISBN 978-5-534-01159-3. — URL: https://biblio-online.ru/bcode/437686 (дата обращения: 18.02.2020).— Текст : электронный