МИНИСТЕРСТВО ВЫСШЕГО И СРЕДНЕГО СПЕЦИАЛЬНОГО ОБРАЗОВАНИЯ РЕСПУБЛИКИ УЗБЕКИСТАН

ТАШКЕНТСКИЙ ГОСУДАРСТВЕННЫЙ ЭКОНОМИЧЕСКИЙ УНИВЕРСИТЕТ

ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ

ТАШКЕНТ-2007

МИНИСТЕРСТВО ВЫСШЕГО И СРЕДНЕГО СПЕЦИАЛЬНОГО ОБРАЗОВАНИЯ РЕСПУБЛИКИ УЗБЕКИСТАН

ТАШКЕНТСКИЙ ГОСУДАРСТВЕННЫЙ ЭКОНОМИЧЕСКИЙ УНИВЕРСИТЕТ

Д.П.ХАШИМОВА, Ш.Т.НАСРИДИНОВА

ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ

Рекомендовано в качестве учебного пособия Координационным Советом Министерства высшего и среднего специального образования Республики Узбекистан для направления образования «Информатика и информационные технологии»

ТАШКЕНТ – 2007

АННОТАЦИЯ

Развитие в XXI веке информационных технологий требует специальных знаний и умений. Исходя из этого, в последние годы возрастают потребность в квалифицированных специалистах в области информационных технологий и требования к уровню их подготовки. Такой специалист должен уметь формулировать требования к программным средствам, оценивать их качество и эффективность, выбирать программные средства, наиболее соответствующие запросам пользователей, а также участвовать в разработке новых программных изделий и адаптации готовых программных продуктов к конкретным условиям их применения. Данное учебное пособие охватывает материал курса «Технологии программирования», для специальности «Информатика и информационные технологии» и "Экономика(Информационные системы в экономике)" .

АННОТАЦИЯ

Ахборот технологияларнинг XXI асрда ривожланиши маҳсус билимлар ва махоратни талаб қилади. Шу муносабат билан, охирги йилларда ахборотлар технологияси соҳасида малакалий мутаҳассиларга ва уларни таёргарлик даражасига талаблар ошди. Бу мутаҳассис дастурий воситаларга талабларни шаклантира олиши, сифат ва самарасини баҳолаолиши, фойдаланувчиларнинг талабларига мос тушувчи дастурий воситаларни танлаб олиши, баъзи бир холлада янги дастурий маҳсулотларни ишлаб чиқиши ёки тайёр дастурий таъминотларни аниқ фойдаланиш шароятларига мослаштира олиши керак. Ушбу ўқув қўлланма "Информатика ва ахборот технологиялари" ва "Иқтисодиёт (Иқтисодиётда ахборот тизимлари)" мутахассисликлари учун "Дастурлаш технологиялари" курси бўйича материалларни қамраб олади.

ANNOTATION

The development of information technologies in the 21 century requires special knowledge and skills. So recently necessity in skilled specialists is increasing especially in the sphere of information technologies and requirements for their training. Such a specialist must form requirements for requirements for programme means, estimate their quality and effect choose programme means, mostly convenient to users' requirements, and participate in working out of new programme units and adoption ready programme products for concrete conditions of their use. The work covers the course material named "Programming technologies", for specialty "Informatics and information technologies" and "Economic (Information systems of economic)".

СОДЕРЖАНИЕ

Введение 9
1. Роль информационных технологий в повседневной жизни общества. 9
2. Технология программирования как технология разработки программных средств. 10
3. Технология программирования и информатизация общества. 11

Глава 1. Программные продукты и их классификация 16
1. Основные понятия программного обеспечения 16
2.Характеристика программного продукта 17
3. Классы программных продуктов 21
4. Системное программное обеспечение 22
5. Инструментарий технологии программирования 26
6. CASE-технология создания информационных систем 30

Глава 2. Пакеты прикладных программ 33
1. Определение пакета прикладных программ. 33
2. Классификация пакета прикладных программ. 34
3. Определение предметной области. 37
4. Модель предметной области. 38

Глава 3. Жизненный цикл программного изделия, стадии разработки 40
1. Понятие жизненного цикла программного изделия. 40
2. Содержание этапов разработки программного изделия 46

Глава 4. Этапы проектирования и создания программ 51
1. Этапы проектирования программ. 51
2. Постановка задачи. 51
3. Проектирование программы. 53
4. Построение модели. 56
5. Разработка, реализация и анализ алгоритма. 57
6. Тестирование и документирование программы. 58

Глава 5. Модульное программирование 63
1. Понятие модульного программирования. 63
2. Основные характеристики программного модуля. 65
3. Методы разработки структуры программы 67
4. Типовая структура программного продукта, состоящего из программных модулей. 74

Глава 6. Структурное программирование 78
1. Понятие структурного программирования. 78
2. Пошаговая детализация. 80
3. Понятие о псевдокоде. 82
4. Контроль программного продукта. 84

Глава 7. Методы проектирования сверху-вниз, HIPO-технология и главного программиста 86
1. Проектирование (программирование) сверху-вниз 86

2. HIPO - технология 86
3. Метод главного программиста. 88
Глава 8. Проектирование интерфейса пользователя 91
1. Диалоговый режим. 91
2. Графический интерфейс пользователя 92
Глава 9. Тестирование и отладка программного средства 97
1. Основные понятия. 97
2. Принципы и виды отладки программного средства. 97
3. Заповеди отладки программного средства. 99
4. Автономная отладка программного средства. 100
5. Комплексная отладка программного средства. 103
Глава 10. Документирование процесса создания программных продуктов 107
1. Составление технического задания на программирование. 107
2. Технический проект. 109
3. Рабочая документация (рабочий проект). Ввод в действие. 109
Глава 11. Обеспечение функциональности и надежности программного средства 112
1. Обеспечение надежности – основной мотив разработки программных средств. 112
2. Обеспечение завершенности программного средства. 114
3. Обеспечение точности, автономности и устойчивости программного средства. 115
4. Обеспечение защищенности программного средства. 117
Глава 12. Основные экономические категории программного обеспечения 124
1. Понятие программного изделия. 124
2. Понятие качества программного изделия и связанные с ним характеристики 126
3. Экономическая эффективность программного изделия 128
Глава 13. Объектный подход к разработке программных средств 134
1. Объекты и отношения в программировании. Сущность объектного подхода к разработке программных средств. 134
2. Особенности объектного подхода к разработке внешнего описания программного средства. 137
3. Особенности объектного подхода на этапе конструирования программного средства. 141
ГЛОССАРИЙ ПО УЧЕБНОЙ ДИСЦИПЛИНЕ 144
СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 150

Мундарижа

Кириш 9

1. Ахборот технологияларнинг жамиятнинг кундалик хаётдаги ўрни 9

2. Дастурлаш технологияси дастурий воситаларнинг ишлаб чиқиш технологияси сифатида 10

3. Дастурлаш технологияси ва жамиятнинг ахборотлаштириш 11

1-боб. Дастурий маҳсулотлар ва уларнинг таснифи 16

1. Дастурий таъминотнинг асосий тушунчалари 16

2.Дастурий махсулотнинг тавсифи 17

3. Дастурий махсулотларнинг синфлари 21

4. Тизимли дастурий таъминот 22

5. Дастурлаш технологиярининг қуроллари 26

6. Ахборот тизимларни яратувчи CASE-технологияси 30

2-боб. Амалий дастурлар пакетлари(АДП) 33

1. Амалий дастурлар пакетларни тушунчаси. 33

2. Амалий дастурлар пакетларни таснифи. 34

3. Предмет соҳани таърифи. 37

4. Предмет соҳани модели. 38

3-боб. Дастурий маҳсулотнинг мавжуд бўлиш даври, ишлаб чиқиш босқичлари 40

1. Дастурий маҳсулотнинг мавжуд бўлиш даври тушунчаси. 40

2. Дастурий маҳсулотнинг мавжуд бўлиш даври босқичларининг мазмуни 46

4-боб. Дастурларни лойиҳалаштириш ва ишлаб чиқиш босқичлари 51

1. Дастурларни лойиҳалаштириш босқичлари. 51

2. Масаланинг қўйилиши. 51

3. Дастурни лойиҳалаштириш. 53

4. Моделни тузиш. 56

5. Алгоритмни ишлаб чиқиш, амалга ошириш ва таҳлил. 57

6. Тестирование и документирование программы. 58

5-боб. Модули дастурлаш 63

1. Модуллли дастурлаш тушунчаси. 63

2. Дастурий модулнинг асосий характеристикалари. 65

3. Дастур структурасини ишлаб чиқиш усуллари 67

4. Дастурий модуллардан иборат дастурий маҳсулотнинг намунавий тузилиши. 74

6-боб. Таркибий дастурлаш 78

1. Таркибий дастурлаш тушунчаси. 78

2. Босқичма-босқич деталлаштириш. 80

3. Псевдокод тушунчаси. 82

4. Дастурий модулни назорат қилиш. 84

7-боб. Юқоридан-пастга лойиҳалаштириш, HIPO-технологияси 86

ва бош дастурчи усули
1. Юқоридан-пастга лойиҳалаштириш (дастурлаш) 86
2. HIPO - технология 86
3. Бош дастурчи усули.. 88
8-боб. Фойдаланувчининг интерфейсини лойиҳалаштириш 91
1. Мулоқат режими. 91
2. Фойдаланувчининг график интерфейси. 92
9-боб. Дастурий воситанинг тестдан ўтказилиши ва созланиши 97
1. Асосий тушунчалар. 97
2. Дастурий воситани созлаш тамойиллари ва турлари. 97
3. Дастурий воситаларни созлаш қойидалари. 99
4. Дастурий воситаларни автоном созлаш. 100
5. Дастурий воситаларни комплексли созлаш. 103
10-боб. Дастурий маҳсулотларни яратиш жараёнини хужжатлаштириш 107
1. Дастурлашга техник вазифани тузиш. 107
2. Техник лойиҳа. 109
3. Ишчи ҳужжатлар (ишчи лойиҳа) ва ишга тушириш. 109
11-боб. Дастурий воситанинг функционаллиги ва ишончилилигини таъминлаш 112
1. Ишончилилигини таъминлаш – дастурий воситаларни ишлаб чиқиш асосий сабаби. 112
2. Дастурий воситанинг якунийлигини таъминлаш. 114
3. Дастурий воситанинг аниқлигини, автономлигини ва мустаҳкамлигини таъминлаш. 115
4. Дастурий воситанинг ҳимояланганлигини таъминлаш. 117
12-боб. Дастурий таъминотнинг асосий иқтисодий категорияси 124
1. Дастурий маҳсулот тушунчаси.. 124
2. Дастурий маҳсулотнинг сифати ҳақида тушунча ва у билан боғлик бўлган ҳусусиятлар 126
3. Дастурий маҳсулотнинг иқтисодий самарадорлиги 128
13-боб. Дастурий воситаларни ишлаб чиқишга объектли ёндошиш 134
1. Дастурлашда объектлар ва муносабатлар. Дастурий воситаларни ишлаб чиқишга объектли ёндошишнинг моҳияти. 134
2. Дастурий воситанинг ташқи таърифини ишлаб чиқишга объектли ёндошишнинг хусусиятлари. 137
3. Дастурий воситанинг конструктирлаш босқичида объектли ёндошишнинг хусусиятлари. 141
ЎҚУВ ФАНИ БЎЙИЧА ГЛОССАРИЙ 144
ФОЙДАЛАНГАН АДАБИЁТЛАР РЎЙХАТИ 150

CONTENTS

Introduction 9
1. The role of info techs in burden life of society 9
2. Programming technology as technology for working out of programmer means. 10
3. Programming technology and information process of society. 11
Chapter 2. Programme products and their classification. 16
1. Main ideas of programmer supple. 16
2. Characters of a programme product. 17
3. Pogramme products' classes. 21
4. System programme supple. 22
5. Instruments of programming technology 26
6. CASE-technology for creation of information systems. 30
Chapter 2. Supplementary programmes' sets. 33
1. Supplementary programmes' sets defimition. 33
2. Classification of supplementary programmes' sets . 34
3. Concept of a subject field. 37
4. The model of the subject field. 38
Chapter 3. Life cycle of a programme unit stages of work out 40
1. The idea of life cycle of a programme unit. 40
2. Contents of stage of work out of a programme unit. 46
Chapter 4. Stage of design and creation of programmes 51
1. Stages of design of programmes. 51
2. Putting the task. 51
3. Design of a programme. 53
4. Setting a model. 56
5. Working out, realization and analysis of algorithm. 57
6. Testing and documentation of a programme. 58
Chapter 5. Module programming. 63
1. Module programming idea. 63
2. Main characteristics of a programme module. 65
3. Methods of working out of structure of a programme. 67
4. Type structure of a programme product, consisting of a programme product, consisting of programme models. 74
Chapter 6. Structural programming 78
1. Structural programming idia. 78
2. Step-by- step details. 80
3. Psevdocode. 82
4. Testing of a programme product. 84
Chapter 7. Methods of design up-to-down, HIPO technology and chief programmer. 86
1. Programming up-to-down. 86
2. HIPO technology. 86
3. Methods of chief programmer. 88

Chapter 8. Interface user design. 91
1. Dialogue model. 91
2. Graphic interface of user. 92
Chapter 9. Testing and adjustment of programme means 97
1. Main concepts. 97
2. Principles and types of adjustment of programme means. 97
3. Rules of adjustments of adjustments of programme means. 99
4. Auto adjustments of programme means. 100
5. Complex adjustments of programme means. 103
Chapter 10. Documentation of programme creation process 107
1. Working up technical task for programming. 107
2. Technical project. 109
3. Working project introduction. 109
Chapter 11. Functional supple and reliability of program means. 112
1. Supple of reliability – a mean reason for working out of program means. 112
2. Supple for completing of program means. 114
3. Supple of sharpness, independent and stability of program means. 115
4. Supple of dependence of programme means. 117
Chapter 12. Mean economical categories programme means 124
1. Programme unit. 124
2. The idea of quality of a programme unit and as characteristic. 126
3. Economical effect a programme unit. 128
Chapter 13. Objective approach to the working out of programme means. 134
1. Objects and relations in programming of object approach to the working out of programme means. 134
2. Peculiarities of the object approach to the working out of out description of programme means. 137
3. Peculiarities of the object approach on the stage of designing of programme means. . 141
GLOSSARY ON ACADEMIC SUBJECT 144
LITERATURE LIST 150

ВВЕДЕНИЕ

1. Роль информационных технологий в повседневной жизни общества.
2. Технология программирования как технология разработки программных средств.
3. Технология программирования и информатизация общества.

1. Роль информационных технологий в повседневной жизни общества.

Сегодня во всем мире развитие информационные технологий является одним из основных факторов благосостояния и экономического роста страны. Поэтому информационные технологии становятся одним из основных приоритетов государственной политики Узбекистана. Во исполнение Указа Президента страны И.А. Каримова «О дальнейшем развитии компьютеризации и внедрении информационно-коммуникационных технологий» от 30 мая 2002 года принимаются действенные меры по развитию информационных технологий. С этой целью разработана и реализуется «Программа развития компьютеризации и информационно-коммуникационных технологий на 2002-2010 годы», утвержденная постановлением Кабинетом Министров Республики Узбекистан №200 от 6 июня 2002 года. А также принят 11 февраля 2004 году Закон Республики Узбекистан «Об информатизации»[3]. В этом Законе отмечается, что одним из основных направлений государственной политики в области информатизации является: «стимулирование развития производства программных продуктов»[1].

Исходя из этого, в последние годы возрастают потребность в квалифицированных специалистах в области информационных технологий и требования к уровню их подготовки. Такой специалист должен уметь формулировать требования к программным средствам, оценивать их качество и эффективность, выбирать программные средства, наиболее соответствующие запросам пользователей, а также участвовать в разработке новых программных изделий и адаптации готовых программных продуктов к конкретным условиям их применения.

Данное учебное пособие охватывают материал курса «Технологии программирования», для специальности «Информатика и информационные технологии» и «Экономика(Информационные системы в экономике)».

Основная цель обучения дисциплины заключается в том, чтобы дать студентам фундаментальные теоретические знания по технологии проектирования программного обеспечения, сформировать практические навыки по применению методов разработки и методов тестирования программных средств.

Основные задачи обучения дисциплины заключаются в следующем:

- ознакомить студентов с возможностями существующих средств разработки программного обеспечения;
- обучить приемам и методам современных технологий проектирования программного обеспечения;

[1] Закон Республики Узбекистан «Об информатизации».// «Народное слово», 2004 г., 11-февраля.

- освоить методы оценок затрат на разработку программных средств.

В представленном учебном пособии по дисциплине «Технологии программирования» изложены методы и этапы проектирования программ, типовые подходы к разработке программного обеспечения и оценке его экономической эффективности. Весь материал подается без всякой привязки к конкретным языкам программирования, что позволяет интерпретировать теоретические положения и практические рекомендации применительно к конкретной вычислительной технике. Помимо этого, приведены термины, используемые в системном и прикладном программном обеспечении, понятия программного изделия, его качества и экономической эффективности. Так же изложены вопросы организации разработки программных изделий, оценки их эффективности.

2. Технология программирования как технология разработки программных средств.

В соответствии с обычным значением слова «технология» под **технологией программирования** (programming technology) будем понимать совокупность производственных процессов, приводящую к созданию требуемого программного средства, а также описание этой совокупности процессов. Другими словами, технологию программирования мы будем понимать здесь в широком смысле как технологию разработки программных средств, включая в нее все процессы, начиная с момента зарождения идеи этого средства, и, в частности, связанные с созданием необходимой программной документации. Каждый процесс этой совокупности базируется на использовании каких-либо методов и средств, например, компьютер (в этом случае будем говорить о компьютерной технологии программирования).

В литературе имеются и другие, несколько отличающиеся, определения технологии программирования. Эти определения обсуждаются в работе. Используется в литературе и близкое к технологии программирования понятие **программной инженерии**, определяемой как систематический подход к разработке, эксплуатации, сопровождению и изъятию из обращения программных средств. Главное различие между **технологией программирования** и **программной инженерией** как дисциплинами для изучения заключается в способе рассмотрения и систематизации материала. В технологии программирования акцент делается на изучении процессов разработки программных средств (технологических процессов) и порядке их прохождения – методы и инструментальные средства разработки программных средств используются в этих процессах (их применение и образуют технологические процессы). Тогда как в программной инженерии изучаются различные методы и инструментальные средства разработки программных средств с точки зрения достижения определенных целей – эти методы и средства могут использоваться в разных технологических процессах (и в разных технологиях программирования).

Не следует также путать **технологию программирования** с **методологией программирования**. В технологии программирования методы рассматриваются «сверху» – с точки зрения организации технологических процессов, а в

методологии программирования методы рассматриваются «снизу» – с точки зрения основ их построения (методология программирования определяется как совокупность механизмов, применяемых в процессе разработки программного обеспечения и объединенных одним общим философским подходом).

3.Технология программирования и информатизация общества.

Технологии программирования играло разную роль на разных этапах развития программирования. По мере повышения мощности компьютеров и развития средств и методологии программирования росла и сложность решаемых на компьютерах задач, что привело к повышенному вниманию к технологии программирования. Резкое удешевление стоимости компьютеров и, в особенности, стоимости хранения информации на компьютерных носителях привело к широкому внедрению компьютеров практически во все сферы человеческой деятельности, что существенно изменило направленность технологии программирования. Человеческий фактор стал играть в ней решающую роль. Сформировалось достаточно глубокое понятие качества программных средств, причем предпочтение стало отдаваться не столько эффективности программных средств, сколько удобству работы с ним для пользователей (не говоря уже о его надежности). Широкое использование компьютерных сетей привело к интенсивному развитию распределенных вычислений, дистанционного доступа к информации и электронного способа обмена сообщениями между людьми. Компьютерная техника из средства решения отдельных задач все более превращается в средство информационного моделирования реального и мыслимого мира, способное просто отвечать людям на интересующие их вопросы. Начинается этап глубокой и полной информатизации (компьютеризации) человеческого общества. Все это ставит перед технологией программирования новые и достаточно трудные проблемы.

Сделаем краткую характеристику развития программирования по десятилетиям. На рисунке 1 приведена эволюция развития технологии программирования.

В 50-е годы мощность компьютеров (первого поколения) была невелика, а программирование для них велось, в основном, в машинном коде. Решались, главным образом, научно-технические задачи (счет по формулам), задание на программирование содержало, как правило, достаточно точную постановку задачи. Использовалась интуитивная технология программирования: почти сразу приступали к составлению программы по заданию, при этом часто задание несколько раз изменялось (что сильно увеличивало время и без того итерационного процесса составления программы), минимальная документация оформлялась уже после того, как программа начинала работать. Тем не менее, именно в этот период родилась фундаментальная для технологии программирования концепция модульного программирования, ориентированная на преодоления трудностей программирования в машинном коде. Появились первые языки программирования высокого уровня, из которых только ФОРТРАН пробился для использования в следующие десятилетия.

Рис 1. Эволюция развития технологии программирования

В 60-е годы можно было наблюдать бурное развитие и широкое использование языков программирования высокого уровня (АЛГОЛ 60, ФОРТРАН, КОВОЛ и др.), значение которых в технологии программирования явно преувеличивалась. Надежда на то, что эти языки решат все проблемы, возникающие в процессе разработки больших программ, не оправдалась. В результате повышения мощности компьютеров и накопления опыта программирования на языках высокого уровня быстро росла сложность решаемых на компьютерах задач, в результате чего обнаружилась ограниченность языков, проигнорировавших модульную организацию программ. И только ФОРТРАН, бережно сохранивший возможность модульного программирования, гордо прошествовал в следующие десятилетия (все его ругали, но его пользователи отказаться от его услуг не могли из-за грандиозного накопления фонда программных модулей, которые с успехом использовались в новых программах)[4]. Кроме того, было понято, что важно не только то, на каком языке мы программируем, но и то, как мы программируем. Это было уже началом серьезных размышлений над методологией и технологией программирования. Появление в компьютерах 2-го поколения прерываний привело к развитию мультипрограммирования и созданию больших программных систем. Широко стала использоваться коллективная разработка, которая поставила ряд серьезных технологических проблем.

В 70-е годы получили широкое распространение информационные системы и базы данных. К середине 70-ых годов стоимость хранения одного бита информации на компьютерных носителях стала меньше, чем на традиционных носителях. Это резко повысило интерес к компьютерным системам хранения данных. Началось интенсивное развитие технологии программирования, прежде всего, в следующих направлениях:

- обоснование и широкое внедрение нисходящей разработки и структурного программирования,
- развитие абстрактных типов данных и модульного программирования (в частности, возникновение идеи разделения спецификации и реализации модулей и использование модулей, скрывающих структуры данных),
- исследование проблем обеспечения надежности и мобильности программных средств,
- создание методики управления коллективной разработкой программных средств,
- появление инструментальных программных средств (программных инструментов) поддержки технологии программирования.

80-е годы характеризуются широким внедрением персональных компьютеров во все сферы человеческой деятельности и тем самым созданием обширного и разнообразного контингента пользователей программных средств. Это привело к бурному развитию пользовательских интерфейсов и созданию четкой концепции качества программных средств. Появляются языки программирования (например, Ада), учитывающие требования технологии программирования. Развиваются методы и языки спецификации программных средств. Начинается

бурный процесс стандартизации технологических процессов и, прежде всего, документации, создаваемой в этих процессах. Выходит на передовые позиции объектный подход к разработке программных средств. Создаются различные инструментальные среды разработки и сопровождения программных средств. Развивается концепция компьютерных сетей.

90-е годы знаменательны широким охватом всего человеческого общества международной компьютерной сетью, персональные компьютеры стали подключаться к ней как терминалы. Это поставило ряд проблем (как технологического, так и юридического и этического характера) регулирования доступа к информации компьютерных сетей. Остро встала проблема защиты компьютерной информации и передаваемых по сети сообщений. Стали бурно развиваться компьютерная технология (CASE-технология) разработки программных средств и связанные с ней формальные методы спецификации программ. Начался решающий этап полной информатизации и компьютеризации общества.

Краткие выводы

В данной теме дано определение роли информационных технологий в жизни общества. Понятие технологии программирования. Различие между технологией программирования и программной инженерией. Различие между технологией программирования и методологией программирования. Основная цель предмета «Технология программирования». Основные задачи предмета. Объект изучения. Роль Технологии программирования на разных этапах развития программирования. Краткая характеристика развития программирования по десятилетиям.

Ключевые слова и определения

Технология программирования (programming technology) - совокупность производственных процессов, приводящую к созданию требуемого программного средства, а также описание этой совокупности процессов.

Программная инженерия определяется как систематический подход к разработке, эксплуатации, сопровождению и изъятию из обращения программных средств.

В **методологии программирования** методы рассматриваются «снизу» – с точки зрения основ их построения (методология программирования определяется как совокупность механизмов, применяемых в процессе разработки программного обеспечения и объединенных одним общим философским подходом).

Вопросы для обсуждения и контроля:

1. Какова цель обучения дисциплины «Технология программирования»?
2. Что означают термином «Технология программирования»?
3. В чем, на Ваш взгляд, различие между технологией программирования и программной инженерией?
4. В чем, на Ваш взгляд, заключается отличие технологии программирования и методологии программирования?
5. Какую роль играла технология программирования на различных этапах развития программирования?

Рекомендуемая литература

1. Закон Республики Узбекистан «Об информатизации», «Народное слово», 2004 г., 11-февраля.

2. Указ Президента Республики Узбекистан «О дальнейшем развитии компьютеризации и внедрении информационно-коммуникационных технологий».// «Народное слово», 2002 г., 1-июня.

Глава 1. ПРОГРАММНЫЕ ПРОДУКТЫ И ИХ КЛАССИФИКАЦИЯ

1. Основные понятия программного обеспечения
2. Характеристика программного продукта
3. Классы программных продуктов
4. Системное программное обеспечение
5. Инструментарий технологии программирования
6. CASE-технология создания информационных систем

1. Основные понятия программного обеспечения

Возможности компьютера как технической основы системы обработки данных связана с используемым программным обеспечением (программами).

Программа *(program, routine)* - упорядоченная последовательность команд (инструкций) компьютера для решения задачи.

Программное обеспечение *(software)* - совокупность программ обработки данных и необходимых для их эксплуатации документов.

Программы предназначены для машинной реализации задач. Термины задача *приложение* имеют очень широкое употребление в контексте информатики и программного обеспечения.

Задача *(problem, (task)* - проблема, подлежащая решению.

Приложение *(application)* - программная реализация на компьютере решения задачи.

Таким образом, задача означает проблему, подлежащую реализации с использован» средств информационных технологий, а приложение - реализованное на компьютере *ре*шение по задаче. Приложение, являясь синонимом слова "программа", считается более удачным термином и широко используется в информатике.

Термин *задача* употребляется также в сфере программирования, особенно в режиме мультипрограммирования и мультипроцессорной обработки, как единица работы вычислительной системы, требующая выделения вычислительных ресурсов (процессорного времени, основной памяти и т.п.). В данной главе этот термин употребляется в смысле первого определения.

Существует большое число разнообразных классификаций задач. С позиций специфики разработки и вида программного обеспечения будем различать два класса задач -технологические и функциональные.

Технологические задачи ставятся и решаются при организации технологического процесса обработки информации на компьютере. Технологические задачи являются основой для разработки сервисных *средств программного обес-*

печим виде утилит, сервисных программ, библиотек процедур и др., применяемых для обеспечения работоспособности компьютера, разработки других программ и обработки данных функциональных задач.

Функциональные задачи требуют решения при реализации функций управления в рамках информационных систем предметных областей. Например, управление деятельностью торгового предприятия, планирование выпуска продукции, управление перевозкой грузов и т. п. Функциональные задачи в совокупности образуют предметную область и полностью определяют ее специфику.

Предметная (прикладная) область *(application domain)* - совокупность связанных между собой функций, задач управления, с помощью которых достигается выполнение поставленных целей

2.Характеристика программного продукта

Все программы по характеру использования и категориям пользователей можно разделить на два класса (рис.1) -утилитарные программы и программные продукты (изделия)[21].

Утилитарные программы ("программы для себя") предназначены для удовлетворения нужд их разработчиков. Чаще всего утилитарные программы выполняют роль сервиса в технологии обработки данных либо являются программами решения функциональных задач, не предназначенных для широкого распространения.

Программные продукты (изделия) предназначены для удовлетворения потребностей пользователей, широкого распространения и продажи.

В настоящее время существуют и другие варианты легального распространения программных продуктов, которые появились с использованием глобальных или региональных телекоммуникаций:

Рис.1. Классификация программ по категориям пользователей

Freeware- бесплатные программы, свободно распространяемые, поддерживаются самим пользователем, который правомочен вносить в них необходимые изменения;

Shareware - некоммерческие (условно-бесплатные) программы, которые могут использоваться, как правило, бесплатно. При условии регулярного использования подобных продуктов осуществляется взнос определенной суммы.

Ряд производителей использует *OEM-программы* (Original Equipement Manufacturer), т.е. встроенные программы, устанавливаемые на компьютеры или поставляемые Месте с вычислительной техникой.

Программный продукт должен быть соответствующим образом подготовлен к эксплуатации, иметь необходимую техническую документацию, предоставлять сервис и гарантию надежной работы программы, иметь товарный знак изготовителя, а также желательно наличие кода государственной регистрации. Только при таких условиях созданный программный комплекс может быть назван программным продуктом.

Программный продукт - комплекс взаимосвязанных программ для решения определенной проблемы (задачи) массового спроса, подготовленный к реализации как любой вид промышленной продукции. Путь от "программ для себя" до программных продуктов достаточно долгий, он связан изменениями технической и программной среды разработки и эксплуатации программ, с появлением и развитием самостоятельной отрасли - информационного бизнеса, для которого характерны разделение труда фирм - разработчиков программ, их дальнейшая специализация, формирование рынка программных средств и информационных услуг. Программные продукты могут создаваться как:

- индивидуальная разработка под заказ;
- разработка для массового распространения среди пользователей.

При индивидуальной разработке фирма-разработчик создает оригинальный программный продукт, учитывающий специфику обработки данных для конкретного заказчика.

При разработке для массового распространения фирма-разработчик, с одной стороны, обеспечить универсальность выполняемых функций обработки данных, с другой, гибкость и настраиваемость программного продукта на условия конкретного применения. Отличительной особенностью программных продуктов должна быть их системность - функциональная полнота и законченность реализуемых функций обработки, применяемых в совокупности.

Программный продукт разрабатывается на основе промышленной технологии выполнения проектных работ с применением современных инструментальных средств программирования. Специфика заключается в уникальности процесса разработки алгоритмов и программ, зависящего от характера обработки информации и используемых инструментальных средств. На создание программных продуктов затрачиваются значительные ресурсы - трудовые, материальные, финансовые; требуется высокая квалификация разработчиков. Как правило, программные продукты требуют сопровождения, которое осуществляется специализированными фирмами - распространителями программ (дистрибьютерами), реже - фирмами-разработчиками. Сопровождение программ массового применения сопряжено с большими трудозатратами - исправление обнаруженных ошибок, создание новых программ и т.п.

Сопровождение программного продукта – поддержка работоспособности программного продукта, переход на его новые версии, внесение изменений, исправление обнаруженных ошибок и т.п.

Программные продукты в отличие от традиционных программных изделий не имеют строго регламентированного набора качественных характеристик, задаваемых при создании программ, либо эти характеристики невозможно

заранее точно указать или оценить, одни и те же функции обработки, обеспечиваемые программным средством, могут различную глубину проработки. Даже время и затраты на разработку программных продуктов не могут быть определены с большой степенью точности заранее. Основными характеристиками програм являются:

- алгоритмическая сложность (логика алгоритмов обработки информации);
- состав и глубина проработки реализованных функций обработки;
- полнота и системность функций обработки;
- объем файлов программ;
- требования к операционной системе и техническим средствам обработки со сторон программного средства;
- объем дисковой памяти;
- размер оперативной памяти для запуска программ;
- тип процессора;
- версия операционной системы;
- наличие вычислительной сети и др.

Программные продукты имеют многообразие *показателей качества,* которые отражают следующие аспекты:

- насколько хорошо (просто, надежно, эффективно) можно использовать программный продукт;
- насколько легко эксплуатировать программный продукт;
- можно ли использовать программный продукт при изменении условия его применения и др.

Приведем характеристики качества программных продуктов:

Мобильность программных продуктов означает их независимость от технического комплекса системы обработки данных, операционной среды, сетевой технологии обработки данных, специфики предметной области и т.п. Мобильный (многоплатформный) программный продукт может быть установлен на различных моделях компьютеров и операционных систем, без ограничений на его эксплуатацию в условиях вычислительной сети. Функции обработки такого программного продукта пригодны для массового использования без каких- либо изменений.

Надежность работы программного продукта определяется бессбойностью и устойчивостью в работе программ, точностью выполнения предписанных функций обработки, возможностью диагностики возникающих в процессе работы программ ошибок.

Эффективность программного продукта оценивается как с позиций прямого назначения - требований пользователя, так и с точки зрения расхода вычислительных ресурсов, необходимых для его эксплуатации.

Расход вычислительных ресурсов оценивается через объем внешней памяти для размещения программ и объем оперативной памяти для запуска программ.

Учет человеческого фактора означает обеспечение дружественного интерфейса для работы конечного пользователя, наличие контекстно-зависимой подсказки или обучающей системы в составе программного средства, хорошей

документации для освоения и использования заложенных в программном средстве функциональных возможностей, анализ и диагностику возникших ошибок и др.

Модифицируемость программных продуктов означает способность к внесению изменений, например расширение функций обработки, переход на другую техническую базу обработки и т.п.

Коммуникативность программных продуктов основана на максимально возможной их интеграции с другими программами, обеспечении обмена данными в общих форматах представления (экспорт/импорт баз данных, внедрение или связывание объектов обработки и др.).

В условиях существования рынка программных продуктов важными характеристиками являются:

-стоимость;

-количество продаж;

-время нахождения на рынке (длительность продаж);

-известность фирмы-разработчика и программы;

-наличие программных продуктов аналогичного назначения.

Программные продукты массового распространения продаются по ценам, которые учитывают спрос и конъюнктуру рынка (наличие и цены программ-конкурентов). Большое значение имеет проводимый фирмой маркетинг, который включает:

• формирование политики цен для завоевания рынка;

• широкую рекламную кампанию программного продукта;

• создание торговой сети для реализации программного продукта (так называемые дилерские и дистрибьютерские центры);

• обеспечение сопровождения и гарантийного обслуживания пользователей программного продукта, создание горячей линии (оперативный ответ на возникающие в процессе эксплуатации программных продуктов вопросы);

• обучение пользователей программного продукта.

Спецификой программных продуктов (в отличие от большинства промышленных изделий) является также и то, что их эксплуатация должна выполняться на правовой основой - лицензионные соглашения между разработчиком и пользователями с соблюдением авторских прав разработчиков программных продуктов.

3. Классы программных продуктов

Программные продукты можно классифицировать по различным признакам. Рассмотрим классификацию, в которой основополагающим признаком является сфера (область) использования программных продуктов:

- аппаратная часть автономных компьютеров и сетей ЭВМ;
- функциональные задачи различных предметных областей;
- технология разработки программ.

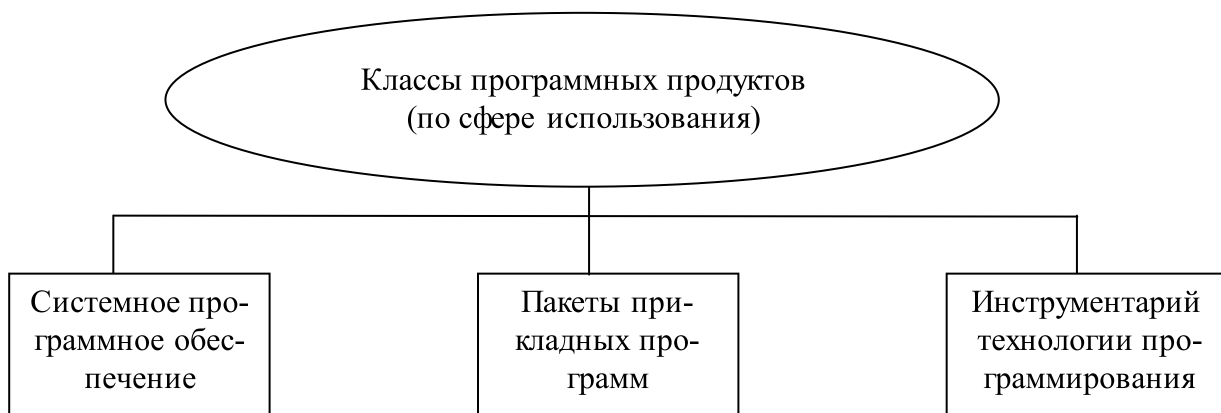


Рис. 2. Классы программных продуктов

Для поддержки информационной технологии в этих областях выделим соответственно три класса программных продуктов, представленных на рис.1.

* системное программное обеспечение;
* пакеты прикладных программ;
* инструментарий технологии программирования.

Системное программное обеспечение направлено:

* на создание операционной среды функционирования других программ;
* на обеспечение надежной и эффективной работы самого компьютера и вычислительной сети;
* на проведение диагностики и профилактики аппаратуры компьютера и вычислительных сетей;
* на выполнение вспомогательных технологических процессов (копирование, архивирование, восстановление файлов программ и баз данных и т.д.).

Данный класс программных продуктов тесно связан с типом компьютера и является его неотъемлемой частью[26]. Программные продукты в основном ориентированы на квалифицированных пользователей - профессионалов в компьютерной области: системного программиста, администратора сети, прикладного программиста, оператора. Однако знание базовой технологии работы с этим классом программных продуктов требуется и конечным пользователям персонального компьютера, которые самостоятельно не только работают со своими программами, но и выполняют обслуживание компьютера, программ и данных.

Программные продукты данного класса носят общий характер применения, независимо от специфики предметной области. К ним предъявляются высокие требования по надежности и технологичности работы, удобству и эффективности использования.

Пакеты прикладных программ (ППП) служат программным инструментарием решения функциональных задач и являются самым многочисленным классом программных продуктов. В данный класс входят программные продукты, выполняющие обработку информации различных предметных областей.

Установка программных продуктов на компьютер выполняется квалифицированными пользователями, а непосредственную их эксплуатацию осуществляют, как правило, конечные пользователи - потребители информации, во многих случаях, деятельность которых весьма далека от компьютерной области. Данный класс программных продуктов может быть весьма специфичным для отдельных предметных областей.

Инструментарий технологии программирования обеспечивает процесс разработки программ и включает специализированные программные продукты, которые являются инструментальными средствами разработчика. Программные продукты иного класса поддерживают все технологические этапы процесса проектирования, программирования (кодирования), отладки и тестирования создаваемых программ. Пользователями технологии программирования являются системные и прикладные программисты.

Инструментарий технологии программирования - совокупность программ и программных комплексов, обеспечивающих технологию разработки, отладки и внедрения создаваемых программных продуктов.

4. Системное программное обеспечение

На рис. 2. представлена структура системного программного обеспечения - базового программного обеспечения, которое, как правило, поставляется вместе с компьютером, сервисного программного обеспечения, которое может быть приобретено дополнительно.

Базовое программное обеспечение (base software) - минимальный набор программных средств, обеспечивающих работу компьютера.

Сервисное программное обеспечение - программы и программные комплексы, которые расширяют возможности базового программного обеспечения и организуют более удобную среду работы пользователя.

Базовое программное обеспечение

В базовое программное обеспечение входят:

- операционная система;
- операционные оболочки (текстовые и графические);
- сетевая операционная система.

Операционная система предназначена для управления выполнением пользовательских программ, планирования и управления вычислительными ресурсами ЭВМ.

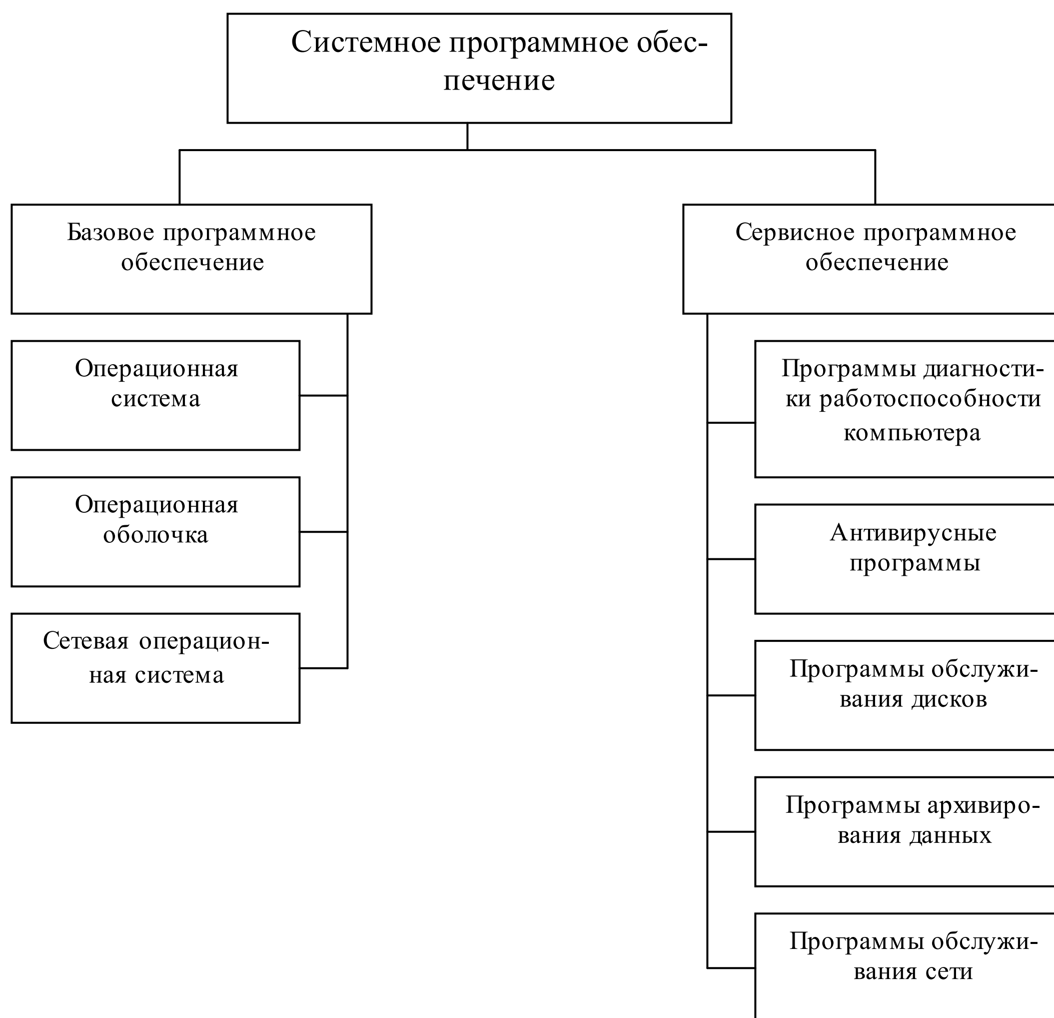


Рис. 2. Классификация системного программного обеспечения компьютера

В секторе программного обеспечения и операционных систем ведущее положение занимают фирмы IBM, Microsoft, UNISYS, Nowell. Доход от продаж операционных систем в среднем превышает 20 млрд. дол. в год. Рассмотрим наиболее распространенные типы операционных систем.

Операционные системы для персональных компьютеров делятся на:

* одно- и многозадачные (в зависимости от числа параллельно выполняемых прикладных процессов);

* одно- и многопользовательские (в зависимости от числа пользователей, одновременно работающих с операционной системой);

* непереносимые и переносимые на другие типы компьютеров;

* несетевые и сетевые, обеспечивающие работу в локальной вычислительной сети ЭВМ.

Большое значение сегодня имеет применение 32-разрядных операционных систем для персональных компьютеров:

* OS/2 во всех модификациях (IBM);

* WINDOWS NT во всех модификациях (Microsoft);
* Unix во всех модификациях;
* Next step3.2 (Next);
* SCO Open Desktop 3.0 (Santa Cruz Operation)
* Solaris 2.1 (SunSoft) - x86;
* UnixWare Personal Edition 1.0 (Novell).

Наиболее традиционное сравнение ОС осуществляется по следующим характеристикам процесса обработки информации:

- управление памятью (максимальный объем адресуемого пространства, типы памяти, технические показатели использования памяти);
- функциональные возможности вспомогательных программ (утилит) в составе операционной системы;
- наличие компрессии диска;
- возможность архивирования файлов;
- задержка многозадачного режима работы;
- задержка сетевого программного обеспечения;
-наличие качественной документации;
- условия и сложность процесса инсталляции.

Сетевые операционные системы - комплекс программ, обеспечивающих обработку, передачу и хранение данных в сети. Сетевая ОС предоставляет пользователям различные виды сетевых служб (управление файлами, электронная почта, процессы управления сетью и др.), поддерживает работу в абонентских системах. Сетевые операционные системы используют архитектуру клиент-сервер или одноранговую архитектуру. Вначале сетевые операционные системы поддерживали лишь локальные вычислительные сети (ЛВС), сейчас эти операционные системы распространяются на ассоциации локальных сетей.

Операционные оболочки - специальные программы, предназначенные для облегчения общения пользователя с командами операционной системы. Операционные оболочки имеют текстовый и графический варианты интерфейса конечного пользователя.

Эти программы существенно упрощают задание управляющей информации для выполнения команд операционной системы, уменьшают напряженность и сложность работы конечного пользователя.

Сервисное программное обеспечение

Расширением базового программного обеспечения компьютера является набор сервисных, дополнительно устанавливаемых программ, которые можно классифицировать по функциональному признаку следующим образом (рис 3):

- программы диагностики работоспособности компьютера;
- антивирусные программы, обеспечивающие защиту компьютера, обнаружение и восстановление зараженных файлов;
- программы обслуживания дисков, обеспечивающие проверку качества поверхности магнитного диска, контроль сохранности файловой системы на

логическом и физическом уровнях, сжатие дисков, создание страховых копий дисков, резервирование данных на внешних носителях и др.;

• программы архивирования данных, которые обеспечивают процесс сжатия информации в файлах с целью уменьшения объема памяти для ее хранения;

• программы обслуживания сети.

Рис. 3. Классификация сервисного программного обеспечения по функциональному признаку

Эти программы часто называются утилитами. Утилиты - программы, служащие для выполнения вспомогательныхс операций обработки данных или обслуживания компьютеров (диагностики, тестирования аппаратных и программных средств, оптимизации использования дискового пространства, восстановления разрушенной на магнитном диске информации и т.п.).

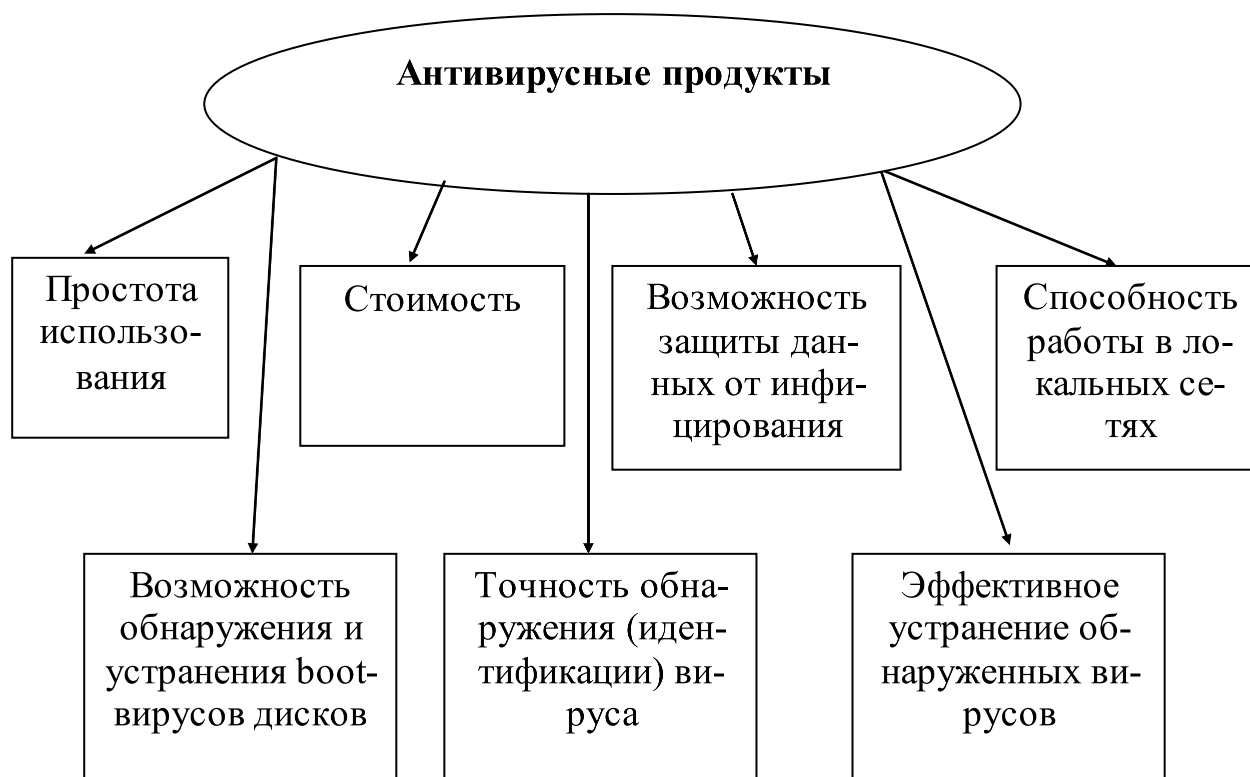


Рис. 4. Критерии оценок антивирусных программ

Антивирусные продукты оцениваются по ряду критериев. На рисунке 4 приведены критерии оценки антивирусных программ.

5. Инструментарий технологии программирования

Состав и назначение инструментария технологии программирования

В настоящее время бурно развивается направление, связанное с технологией создания программных продуктов. Это обусловлено переходом на промышленную технологию производства программ, стремлением к сокращению сроков, трудовых и материальных затрат на производство и эксплуатацию программ, обеспечению гарантированного уровня их качества. Это направление часто называют программотехникой. Программотехника (software engineering) - технология разработки, отладки, верификации и внедрения программного обеспечения. Инструментарий технологии программирования - программные продукты поддержки (обеспечения) технологии программирования.

В рамках этих направлений сформировались следующие группы программных продуктов:

• средства для создания приложений, включающие:

-локальные средства, обеспечивающие выполнение отдельных работ по созданию программ (рис 5);

Рис. 5. Классификация инструментария технологии программирования

-интегрированные среды разработчиков программ, обеспечивающие выполнение
комплекса взаимосвязанных работ по созданию программ;

• CASE-технология (Computer-Aided System Engineering), представляющая методы анализа, проектирования и создания программных систем и предназначенная для автоматизации процессов разработки и реализации информационных систем.

Средства для создания приложений

Средства для создания приложений - совокупность языков и систем программирования, а также различные программные комплексы для отладки и поддержки создаваемых программ[27].

Языки программирования если в качестве признака классификации взять синтаксис образования его конструкций, можно условно разделить на классы (рис. 6):

Рис. 6. Классификация языков программирования по конструкции

- машинные языки (computer language) - языки программирования, воспринимаемые аппаратной частью компьютера (машинные коды);
- машинно-ориентированные языки (computer-oriented language) - языки программирования, которые отражают структуру конкретного типа компьютера (ассемблеры);
- алгоритмические языки (algorithmic language) - не зависящие от архитектуры компьютера языки программирования для отражения структуры алгоритма (Паскаль, Фортран, Бейсик и др.);
- процедурно-ориентированные языки (procedure-oriented language) - языки программирования, где имеется возможность описания программы как совокупности процедур (подпрограмм);
- проблемно-ориентированные языки (universal programming language) - языки программирования, предназначенные для решения задач определенного класса (Лисп, РПГ, Симула и др.);
- интегрированные системы программирования.

Другой классификацией языков программирования является их деление на языки, ориентированные на реализацию основ структурного программиро-

вания, и объектно-ориентированные языки, поддерживающие понятие объектов и их свойств и методов обработки.

Программа, подготовленная на языке программирования, проходит этап трансляции, когда происходит преобразование исходного кода программы в объектный код, который далее пригоден к обработке редактором связей. Редактор связей - специальная программа, обеспечивающая построение загрузочного модуля, пригодного к выполнению.

Трансляция может выполняться с использованием средств компиляторов или интерпретаторов. Компиляторы транслируют всю программу, но без ее выполнения. Интерпретаторы, в отличие от компиляторов, выполняют пооператорную обработку и выполнение программы.

Существуют специальные программы, предназначенные для трассировки и анализа выполнения других программ, так называемые отладчики. Лучшие отладчики позволяют осуществить трассировку (отслеживание выполнения программы в пооператорном варианте), идентификацию места и вида ошибок в программе, "наблюдение" за изменением значений переменных, выражений и т.п. Для отладки и тестирования правильности работы программ создается база данных контрольного примера.

Системы программирования включают:

- компилятор;
- интегрированную среду разработчика программ;
- отладчик;
- средства оптимизации кода программ;
- набор библиотек (возможно с исходными текстами программ);
- редактор связей;
- сервисные средства (утилиты) для работы с библиотеками, текстовыми и двоичными файлами;
- справочные системы;
- документатор исходного кода программы;
- систему поддержки и управления проектом программного комплекса.

Средства поддержки проектов - новый класс программного обеспечения, предназначен для:

- отслеживания изменений, выполненных разработчиками программ;
- поддержки версий программы с автоматической разноской изменений;
- получения статистики о ходе работ проекта.

Инструментальная среда пользователя представлена специальными средствами, встроенными в пакеты прикладных программ, такими, как:

- библиотека функций, процедур, объектов и методов обработки;
- макрокоманды;
- клавишные макросы;
- языковые макросы;
- программные модули-вставки;
- конструкторы экранных форм и отчетов;
- генераторы приложений;
- языки запросов высокого уровня;

- языки манипулирования данными;
- конструкторы меню и многое другое.

Средства отладки и тестирования программ предназначены для подготовки разработанной программы к промышленной эксплуатации.

6. CASE-технология создания информационных систем

Средства CASE-технологии - относительно новое, сформировавшееся на рубеже 80-х направление. Массовое применение затруднено крайне высокой стоимостью и предъявляемыми требованиями к оборудованию рабочего места разработчика[26].

CASE-технология - программный комплекс, автоматизирующий весь технологический процесс анализа, проектирования, разработки и сопровождения сложных программных систем.

Средства CASE-технологии делятся на две группы:

- встроенные в систему реализации - все решения по проектированию и реализации привязаны к выбранной системе управления базами данных (СУБД);
- независимые от системы реализации - все решения по проектированию ориентированы на унификацию начальных этапов жизненного цикла и средств их документирования, обеспечивают большую гибкость в выборе средств реализации.

Основное достоинство CASE-технологии - поддержка коллективной работы над проектом за счет возможности работы в локальной сети разработчиков, экспорта/импорта любых фрагментов проекта, организационного управления проектом.

Некоторые CASE-технологии ориентированы только на системных проектировщиков и предоставляют специальные графические средства для изображения различного вида моделей.

Другой класс CASE-технологий поддерживает только разработку программ, включая:

- автоматическую генерацию кодов программ на основании их спецификаций;
- проверку корректности описания моделей данных и схем потоков данных;
- документирование программ согласно принятым стандартам и актуальному состоянию проекта;
- тестирование и отладку программ.

Кодогенерация программ выполняется двумя способами: создание каркаса программ и создание полного продукта. Каркас программы служит для последующего ручного варианта редактирования исходных текстов, обеспечивая возможность вмешательства программиста; полный продукт не редактируется вручную.

В рамках CASE-технологий проект сопровождается целиком, а не только его программные коды. Проектные материалы, подготовленные в CASE-

технологии, служат заданием программистам, а само программирование скорее сводится к кодированию -переводу на определенный язык структур данных и методов их обработки, если не предусмотрена автоматическая кодогенерация.

Большинство CASE-технологий использует также метод "прототипов" для быстрого создания программ на ранних этапах разработки. Кодогенерация программ осуществляется автоматически - до 85 - 90% объектных кодов и текстов на языках высокого уровня, а в качестве языков наиболее часто используются Ада, Си, Кобол.

Краткие выводы

В данной теме дано определение понятия программного обеспечения, классов программных продуктов: системное программное обеспечение, пакеты прикладных программ; инструментарий технологии программирования. Раскрыта сущность характеристики системного программного обеспечения, базового программного обеспечения и сервисного программного обеспечения. Инструментарии технологии программирования и его классификация: средства для создания приложений, средства для создания информационных систем (CASE-технология).

Ключевые слова и определения

Задача (problem, (task) - проблема, подлежащая решению.

Программное обеспечение (software) - совокупность программ обработки данных и необходимых для их эксплуатации документов.

Программный продукт - комплекс взаимосвязанных программ для решения определенной проблемы (задачи) массового спроса, подготовленный к реализации как любой вид промышленной продукции.

Программа (program, routine) - упорядоченная последовательность команд (инструкций) компьютера для решения задачи.

Приложение *(application)* - программная реализация на компьютере решения задачи

Утилитарные программы ("программы для себя") предназначены для удовлетворения нужд их разработчиков. Чаще всего утилитарные программы выполняют роль сервиса в технологии обработки данных либо являются программами решения функциональных задач, не предназначенных для широкого распространения.

Freeware- бесплатные программы, свободно распространяемые, поддерживаются самим пользователем, который правомочен вносить в них необходимые изменения;

Shareware - некоммерческие (условно-бесплатные) программы, которые могут использоваться, как правило, бесплатно.

CASE-технология - программный комплекс, автоматизирующий весь технологический процесс анализа, проектирования, разработки и сопровождения сложных программных систем.

Вопросы для обсуждения и контроля:

1. Как можно классифицировать программы по категориям пользователей?
2. Какие основные характеристики программ вам известны?
3. Дайте определение программного продукта?
4. Какие показатели качества программного продукта вам известны?
5. Назовите основные характеристики качества программных продуктов?
6. Назовите качественные характеристики программного продукта?
7. Назовите составляющие инструментария технологии программирования.
8. Раскройте сущность средств для создания приложений.
9. Что вы понимаете под CASE- технология?
10. Назовите программные продукты для создания приложений.
11. Какие сетевые операционные системы вы знаете?

Рекомендуемая литература

1. Балдин К.В., Уткин В.Б. Информационные системы в экономике: Учебник. – М.: Издательско-торговая корпорация «Дашков и К», 2005. 25-33 стр.
2. Чернев Д.А. Технология разработки программного обеспечения: (Учебное пособие).-Т.:, “Mehnat”, 2004.7-8 стр.
3. Информатика: Учебник. / Под ред. Н.В.Макаровой. 3-е перераб. изд. - М.: ФиС, 2004. –298-308 стр.
4. Ғуломов С. С., Шермухамедов А. Т., Бегалов Б.А. «Иқтисодий информатика». Тошкент. "Ўзбекистон", 1999. 267-276 бетлар.
5. Благодатских В.А., Енгибарян М.А., Ковалевская Е.В. Экономика, разработка и использование программного обеспечения ЭВМ. - М.: Финансы и статистика, 1995. 5-14 стр.

Глава 2 . ПАКЕТЫ ПРИКЛАДНЫХ ПРОГРАММ

1. Определение ППП.
2. Классификация ППП.
3. Определение предметной области.
4. Модель предметной области.

1. Определение пакет прикладных программ.

Термин "**Пакет прикладных программ**" применяется к комплексам программ различной сложности и назначения. Ранее отмечалось, что нельзя провести четкую границу между программным изделием, являющимся прикладной программой, и пакет прикладных программ(ППП). По мере разработки все большего числа пакетов программ появлялись и новые определения того, что следует понимать под пакетом программ.

По современным взглядам, **пакет прикладных программ** - это совокупность совместимых программ для решения определенного класса задач. ППП всегда ориентируется на пользователей определенной квалификации, как в программировании, так и в той области, к которой относятся задачи, решаемые с применением этого ППП[27].

Совместимость программ, составляющих ППП, означает возможность их взаимного использования, общность структуры управляющих данных и используемых информационных массивов. Кроме того, ППП следует рассматривать как самостоятельное программное изделие, как особый вид прикладного ПО.

Исходя из определения, можно выделить следующие общие свойства ППП:

- Пакет состоит из нескольких программных единиц.
- Пакет предназначен для решения определенного класса задач.
- В пределах своего класса пакет обладает определенной универсальностью, т.е. позволяет решать все или почти все задачи этого класса.
- В пакете предусмотрены средства управления, позволяющие выбирать конкретные возможности из числа предусмотренных в пакете.
- Пакет допускает настройку на конкретные условия применения.
- Пакет разработан с учетом возможности его использования за пределами той организации, в которой он создан, и удовлетворяет общим требованиям к программному изделию.
- Документация и способы применения пакета ориентированы на пользователя, имеющего определенный уровень квалификации в той области знаний, к которой относятся решаемые пакетом задачи.

Поскольку ППП предназначен для решения определенного класса задач, можно говорить о функциональном назначении пакета.

2. Классификация ППП

В зависимости от функционального назначения выделяют ППП, расширяющие возможности ОС, например, для построения многопользовательских систем, работы с удаленными абонентами, реализации специальной организации файлов, упрощения работы с ОС и т.п. Примерами таких пакетов служат пакет СРВ, реализующий режим разделения времени в ОС ЕС ЭВМ, пакет Norton Commander для облегчения работы с операционной системой MS DOS на персональных ЭВМ.

Среди пакетов, предназначенных для решения прикладных задач пользователей, иногда выделяют методо-ориентированные и проблемно-ориентированные пакеты. **Методо-ориентированный пакет** предназначен для решения задачи пользователя одним из нескольких методов, предусмотренных в пакете, причем метод либо назначается пользователем, либо выбирается автоматически на основе анализа входных данных. Пример такого пакета - пакет математического программирования, позволяющий решить задачу выпуклого программирования либо методом штрафных функций, либо одним из вариантов методов возможных направлений.

Проблемно-ориентированные пакеты предназначены для решения групп (последовательностей) задач, использующих общие данные. Это наиболее многочисленная группа пакетов. Проблемная ориентация может выражаться в общем характере операций, выполняемых пакетом. Типичные примеры таких пакетов - текстовые редакторы, табличные процессоры, пакет линейного программирования.

Проблемная ориентация может быть представлена и общей прикладной проблемой, решение которой распадается на отдельные задачи, для каждой из которых в пакете предусмотрен свой алгоритм. Типичные примеры - пакет для проведения расчетов межотраслевых балансов, пакеты, используемые в различных системах автоматизации проектирования.

В последние годы получили распространение так называемые **интегрированные пакеты**, представляющие собой пакеты широкого назначения, объединяющие текстовый редактор, процессор электронных таблиц, систему управления базой данных, пакет графического отображения данных (деловую графику) и средства обмена данными с удаленными абонентами. рис.1 показан вариант классификации пакетов по их функциональному назначению.

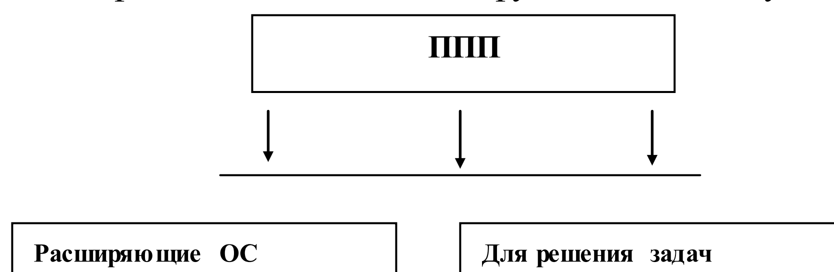


Рис.1. Вариант классификации ППП по функциональному назначению

При определении пакета программ было отмечено, что пакет состоит из нескольких программных единиц. Такие программные единицы обычно называют **программными модулями**. Пакет предназначен для решения задач определенного класса. Этот класс задач обычно называют **предметной областью пакета**. Применительно к ППП для решения расчетных задач предметная область определяет некоторую структуру данных, т.е. организацию входных, промежуточных и выходных данных. Говорят, что пакет использует информационную базу, соответствующую своей предметной области[27].

Для реализации выбранных пользователем конкретных действий пакет должен воспринимать от пользователя управляющую информацию. Эта управляющая информация представляется на формальном языке - входном языке пакета. Описание конкретного задания пользователя на входном языке пакета называют **программой на входном языке (ПВЯ).**

Решение каждой задачи в пакете сводится к выполнению соответствующего алгоритма. Программные модули пакета, реализующие алгоритмы решения задач, предусмотренных в пакете, будем называть обрабатывающими модулями. Обрабатывающие модули выполняют преобразование данных, составляющих информационную базу пакета.

Для того чтобы преобразовать задание пользователя в последовательность вызовов обрабатывающих модулей, в пакет должны входить управляющие модули.

Чтобы обеспечить взаимодействие пакета с пользователем и управляющих модулей пакета с информационной базой и обрабатывающими модулями, в состав пакета включаются обслуживающие модули.

Таким образом, ППП можно рассматривать как объединение входного языка, информационной базы, управляющих, обслуживающих и обрабатывающих программных модулей. Совокупность обрабатывающих модулей часто называют функциональным наполнением пакета. Управляющие и обслуживающие модули называются системной частью пакета, или системным наполнением пакета.

Взаимодействие составных частей пакета схематически показано на рис.2. Средствами операционной системы запускается головной управляющий модуль пакета (ведущий модуль). Затем организуются прием задания пользователя, представляемого в форме программ на входном языке (ПВЯ), и выполнение этого задания путем вызова в нужной последовательности обрабатывающих и обслуживающих модулей.

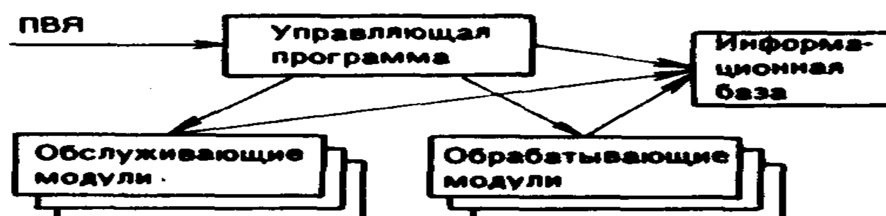


Рис.2. Составные части ППП

Под способом применения ППП будем понимать организацию взаимодействия пользователя с пакетом при решении задач. Выбор способа применения ППП зависит от многих факторов, из которых наиболее существенными являются возможности ОС и выбранного языка программирования, объемы обрабатываемых данных, продолжительность решения задачи, частота использования ППП, особенности квалификации пользователей пакета и требования к оперативности решения задач (допустимому времени ожидания результатов расчетов)[21].

Способы применения существующих в настоящее время ППП весьма разнообразны, однако можно выделить некоторые типовые режимы, определяемые построением самого пакета и особенностями используемой ЭВМ и ОС.

Простейший режим с точки зрения построения ППП сводится к использованию отдельных программ пакета как подпрограмм некоторой главной программы, составляемой пользователем на каком-либо языке программирования, например ПЛ/1 или Си. В этом случае ППП состоит только из обрабатывающих модулей и может рассматриваться как расширение библиотеки подпрограмм используемого языка программирования.

Следующий по сложности реализации режим предполагает, что вся управляющая информация для конкретного выполнения пакета передается в виде законченной программы на входном языке при запуске пакета. Дальнейшая работа пакета проходит без участия пользователя. Такой режим по аналогии с соответствующим режимом ОС часто называют **пакетным.** Пакетный режим удобен, когда требуется решать много однотипных задач с использованием одной и той же программы на входном языке, когда время, затрачиваемое на решение каждой задачи, достаточно велико, когда программа на входном языке сложна и имеет значительный объем.

Большинство ППП, применяемых на персональных ЭВМ, ориентировано на **диалоговое взаимодействие** с пользователем в ходе решения задач.

Простейший диалоговый режим (вариант диалогового взаимодействия) состоит в том, что пользователь инициирует выполнение пакета, вводит задание в форме программы на входном языке и на этом заканчивает управление выполнением пакета. Фактически этот режим отличается от пакетного только возможностью исправления ошибок в ПВЯ, повторного запуска пакета при неудачах.

Более сложный вариант диалогового режима, называемый также режимом сопровождения, предусматривает возможность динамического управления выполнением пакета. Управляющая информация вводится по частям и формируется пользователем в процессе работы с пакетом на основе анализа промежуточных результатов. Такая работа в большинстве случаев более естественна для пользователя, в частности, при использовании пакетов редактирования текстов, при работе с электронными таблицами, при решении многих расчетных задач.

3. Определение и модель предметной области

Область науки или деятельности, к которой относятся задачи, решаемые с применением ППП, называют **предметной областью пакета**. Иначе говоря, предметная область определяется совокупностью задач, решаемых пакетом. Такое содержательное описание предметной области несет полезную информацию для пользователя пакета, но оно недостаточно конкретно для проектирования и разработки ППП.

Разработчик ППП фактически имеет дело с некоторым упрощенным отображением предметной области, с некоторой моделью предметной области.

Под **математической моделью** обычно понимают совокупность некоторых объектов (переменных) и связей (отношений) между этими объектами.

Модель предметной области ППП можно представить, как совокупностью данных (переменных), используемых в пакете при решении задач, и связей между этими данными[28].

Данное (переменная) как часть модели предметной области характеризуется содержательным названием, отображающим его роль в предметной области. Такое название определяется в содержательных терминах предметной области, привычных для пользователя, например "Валовая продукция отрасли", "Цена изделия", "Коэффициент прямых затрат". Данное, поле названия, обычно имеет и уникальное имя (идентификатор), которое и используется при описании модели, тогда как содержательное название необходимо только для связи с пользователем пакета. В процессе вычислений данное получает значение, которое может использоваться для получения значений других данных.

Каждое данное принадлежит к определенному типу данных. Здесь под типом данного понимается совокупность его свойств, в том числе множество допустимых значений, набор операций, которые могут выполняться над данным. С типом данного связана форма представления значений данного в памяти ЭВМ.

Таким образом, каждое используемое в пакете данное в модели предметной области характеризуется именем, типом значением. Имя и тип являются фиксированными атрибутами данного, значение имеет динамический характер. В исходном состоянии данное может не иметь значения (говорят, что значение данного не определено), в процессе вычислений данное может получать значение, изменять значение и терять значение.

Между данными в модели предметной области устанавливаются связи (отношения). Характер этих связей разнообразен и в значительной части определяется семантикой решаемых задач.

Совокупность данных в модели предметной области представляет информационную базу пакета. Данные в информационной базе связаны между собой - образуют некоторую структуру данных. Характер этих связей определяется при разработке информационной базы пакета и обычно не изменяется в процессе функционирования пакета. Будем называть такие связи связями по определению. Таким образом, связи по определению - это связи, устанавливаемые в информационной базе при построении модели предметной области пакета.

Иной характер носят связи, реализуемые обрабатывающими модулями пакета. Эти связи предопределены и потенциально присутствуют в модели предметной области, но реализуются только по прямому или косвенному указанию пользователя в процессе решения конкретной задачи, в ходе работы пакета. Такие связи будем называть **функциональными**.

Работа пакета (решение задач пользователя) в модели предметной области представляется изменением значений данных. В начале работы пакета должны быть установлены (приняты по умолчанию, заданы или введены пользователем) значения некоторых данных, значения остальных данных являются неопределенными. Затем в соответствии с требованиями пользователя выполняются некоторые обрабатывающие модули, в результате чего получают значения некоторые не определенные ранее данные или изменяются значения данных, которые уже имели значения.

Таким образом, данные могут получать новые значения только двумя способами: либо в результате ввода пользователем нового значения, либо в результате выполнения обрабатывающего модуля.

Количество допустимых типов данных и сам перечень типов являются важными характеристиками МПО и всего пакета.

По способу присваивания конкретных значений, данные можно разделить на следующие группы.

1. Данное имеет постоянное значение, которое может устанавливаться при загрузке пакета и в процессе работы пакета не изменяется (и не может быть изменено средствами, доступными пользователю пакета). Примерами таких данных служат различные физические константы, справочные таблицы.

2. Данное имеет некоторое фиксированное значение в момент загрузки пакета (так называемое значение по умолчанию), а в ходе загрузки пакета это значение может изменяться по указанию пользователя или в результате выполнения обрабатывающих модулей.

3. Данное не имеет значения до тех пор, пока пользователь не предпримет действий по определению значения этого данного. Поскольку действия пользователя, по предположению, ограничены вводом значений данных и запросами на выполнение обрабатывающих модулей, то из данных этой группы можно выделить такие данные, значения которых не вычисляются ни одним из обрабатывающих модулей. Эти данные могут быть только входными, и если их значения требуются для решения задачи, пользователь должен сам эти значения задавать. Возможна и ситуация, когда одно и то же данное в зависимости от решаемой пользователем задачи может рассматриваться либо как входное, либо как вычисляемое при работе пакета по заданию пользователя.

Таким образом, при построении модели предметной области необходимо установить, какие типы данных будут использоваться в пакете и какие способы присваивания значений должны быть реализованы, затем выбрать имена данных и для каждого данного определить

Краткие выводы

В данной теме дано определение Пакета прикладных программ и его классификация по функциональному назначению. Также взаимосвязь между составными частями пакета. Кроме того, дается определение предметной области и рассматривается одна из моделей предметной области .

Ключевые слова и определения

Термин "ППП"означает комплексы программ различной сложности и назначения.

ППП - это совокупность совместимых программ для решения определенного класса задач. ППП всегда ориентируется на пользователей определенной клалификации как в программировании, так и в той области, к которой относятся задачи, решаемые с применением этого ППП.

Методо-ориентированный пакет предназначен для решения задачи пользователя одним из нескольких методов, предусмотренных в пакете, причем метод либо назначается пользователем, либо выбирается автоматически на основе анализа входных данных.

Проблемно-ориентированные пакеты предназначены для решения групп (последовательностей) задач, использующих общие данные.

Область науки или деятельности, к которой относятся задачи, решаемые с применением ППП, называют **предметной областью пакета**

ПВЯ – входной язык ППП, средство управления работой ППП

Вопросы для обсуждения и контроля:

1. Что означает термин ППП? Назовите виды ППП.
2. Какие подходы используются для различных описаний предметной области ППП?
3. Что такое входной язык ППП?
4. Что является основной структурной единицей ППП?
5. На какие группы можно разделить данные по способу присваивания конкретных значений?
6. Какие связи можно назвать функциональными?
7. Что Вы понимаете под математической моделью предметной области.

Рекомендуемая литература

1.Информатика: Учебник. / Под ред. Н.В.Макаровой. 3-е перераб. изд. - М.: ФиС, 2004. 326-334 стр.

2.Могилев А.В. Информатика: Учеб.пособие для студентов пед.вузов /А.В. Могилев, Н.И. Пак, Е.К.Хеннер; Под.ред. Е.К. Хеннера- 2-е изд., стер. – М.: Издательский центр «Академия», 2003.148-154 стр.

3.Гуломов С.С., Алимов Р.Х., Лутфуллаев Х.С. ва бошқалар. Ахборот тизимлари ва технологиялари. Тошкент.: ”Шарқ”, 2000 й. 292 бет.

4.Ғуломов С. С., Шермухамедов А. Т., Бегалов Б.А. «Иқтисодий информатика». Тошкент. "Ўзбекистон", 1999. 276-289 б.

Глава 3. ЖИЗНЕННЫЙ ЦИКЛ ПРОГРАММНОГО ИЗДЕЛИЯ, СТАДИИ РАЗРАБОТКИ

1. Понятие жизненного цикла программного изделия.
2. Содержание этапов разработки программного изделия

1. Понятие жизненного цикла программного изделия

Под жизненным циклом ПС (software life cycle) понимают весь период его разработки и эксплуатации (использования), начиная от момента возникновения замысла ПС и кончая прекращением всех видов его использования. Жизненный цикл охватывает довольно сложный процесс создания и использования ПС (software process). Этот процесс может быть организован по-разному для разных классов ПС и в зависимости от особенностей коллектива разработчиков.

В настоящее время можно выделить 5 основных подходов к организации процесса создания и использования ПС.

- Водопадный подход. При таком подходе разработка ПС состоит из цепочки этапов. На каждом этапе создаются документы, используемые на последующем этапе. В исходном документе фиксируются требования к ПС. В конце этой цепочки создаются программы, включаемые в ПС.
- Исследовательское программирование. Этот подход предполагает быструю (насколько это возможно) реализацию рабочих версий программ ПС, выполняющих лишь в первом приближении требуемые функции. После экспериментального применения реализованных программ производится их модификация с целью сделать их более полезными для пользователей. Этот процесс повторяется до тех пор, пока ПС не будет достаточно приемлемо для пользователей. Такой подход применялся на ранних этапах развития программирования, когда технологии программирования не придавали большого значения (использовалась интуитивная технология). В настоящее время этот подход применяется для разработки таких ПС, для которых пользователи не могут точно сформулировать требования (например, для разработки систем искусственного интеллекта).
- Прототипирование. Этот подход моделирует начальную фазу исследовательского программирования вплоть до создания рабочих версий программ, предназначенных для проведения экспериментов с целью установить требования к ПС. В дальнейшем должна последовать разработка ПС по установленным требованиям в рамках какого-либо другого подхода (например, водопадного).
- Формальные преобразования. Этот подход включает разработку формальных спецификаций ПС и превращение их в программы путем корректных преобразований. На этом подходе базируется компьютерная технология (CASE-технология) разработки ПС.
- Сборочное программирование. Этот подход предполагает, что ПС конструируется, главным образом, из компонент, которые уже существуют.

Должно быть некоторое хранилище (библиотека) таких компонент, каждая из которых может многократно использоваться в разных ПС. Такие компоненты называются повторно используемыми (reusable). Процесс разработки ПС при данном подходе состоит скорее из сборки программ из компонент, чем из их программирования .

В нашем курсе лекций мы, в основном, будем рассматривать водопадный подход с некоторыми модификациями. Во-первых, потому, что в этом подходе приходиться иметь дело с большинством процессов программной инженерии, а, во-вторых, потому, что в рамках этого подхода создается большинство больших программных систем. Именно этот подход рассматривается в качестве индустриального подхода разработки программного обеспечения. Исследовательское программирование исходит из взгляда на программирование как на искусство. Оно применяется тогда, когда водопадный подход не применим из-за того, что не удается точно сформулировать требования к ПС. В нашем курсе мы этот подход рассматривать не будем. Прототипирование рассматривается как вспомогательный подход, используемый в рамках других подходов, в основном, для прояснения требований к ПС. Компьютерной технологии (включая обсуждение жизненного цикла ПС, созданного по этой технологии) будет посвящена отдельная лекция. Сборочное программирование мы в нашем курсе рассматривать не будем, хотя о повторно используемых программных модулях мы говорить будем, обсуждая свойства программных модулей [30].

Программы любого вида характеризуются жизненным циклом, состоящим из отдельных этапов:

a) маркетинг рынка программных средств, спецификация требований к программному продукту;

b) проектирование структуры программного продукта;

c) программирование (создание программного кода), тестирование, автономная и комплексная отладка программ;

d) документирование программного продукта, подготовка эксплуатационной и технологической документации;

e) выход на рынок программных средств, распространение программного продукта;

f) эксплуатация программного продукта пользователями;

g) сопровождение программного продукта;

h) снятие программного продукта с продажи, отказ от сопровождения.

Маркетинг и спецификация программного продукта предназначены для изучения требований к создаваемому программному продукту, а именно:

- определение состава и назначения функций обработки, данных программного продукта;
- установление требований пользователя к характеру взаимодействия с программным продуктом, типу пользовательского интерфейса (система меню, использование манипулятора мышь, типы подсказок, виды экранных документов и т.п.);

• требования к комплексу технических и программных средств для эксплуатации программного продукта и т.д.

На данном этапе необходимо выполнить формализованную постановку задачи. Если программный продукт создается не под заказ и предполагается выход на рынок программных средств, маркетинг выполняется в полном объеме: изучаются программные продукты-конкуренты и аналоги, обобщаются требования пользователей к программному продукту, устанавливается потенциальная емкость рынка сбыта, дается прогноз цены и объема продаж. Кроме того, важно оценить необходимые для разработки программного продукта материальные, трудовые и финансовые ресурсы, ориентировочные длительности основных этапов жизненного цикла программного продукта.

Если программный продукт создается, как заказное программное изделие на данном этапе также важно правильно сформулировать и документировать задание на его разработку. Ошибочно понятое требование к программному продукту может привести к нежелательным результатам в процессе его эксплуатации. Проектирование структуры программного продукта связано с алгоритмизацией процесса обработки данных, детализацией функций обработки, разработкой структуры программного продукта (архитектуры программных модулей), структуры информационной базы (базы данных) задачи, выбором методов и средств создания программ - технологии программирования.

Программирование, тестирование и отладка программ являются технической реализацией проектных решений и выполняются с помощью выбранного инструментария разработчика (алгоритмические языки и системы программирования, инструментальные среды разработчиков и т.п.).

Для больших и сложных программных комплексов, имеющих развитую модульную структуру построения, отдельные работы данного этапа могут выполняться параллельно, обеспечивая сокращение общего времени разработки программного продукта. Важная роль ежит используемым при этом инструментальным средствам программирования и программ, поскольку они влияют на трудоемкость выполнения работ, их стоимость, качество создаваемых программ.

Документирование программного продукта является обязательным для работ, выполняемых, как правило, не самим разработчиком, а лицом, связанным с распространением и внедрением программного продукта. Документация должна содержать необходимые сведения по установке и обеспечению надежной работы программного продукта, поддерживать пользователей при выполнении функций обработки, определять порядок комплексирования программного продукта с другими программами. Успех распространения и эксплуатации программного продукта в значительной степени зависит от качества его документации.

На машинном уровне программного продукта, как правило, создаются:

- автоматизированная контекстно-зависимая помощь (HELP);

- демонстрационные версии, работающие в активном режиме по типу обучающих систем (электронный учебник) или пассивном режиме (ролик,

мультфильм) - для демонстрации функциональных возможностей программного продукта и информационной технологии его использования.

Выход программного продукта на рынок программных средств связан с организацией продаж массовому пользователю. Этот этап должен по возможности быть коротким, для продвижения программных продуктов применяются стандартные приемы маркетинга: реклама, увеличение числа каналов реализации, создание дилерской и дистрибьютерной сети, ценовая политика - продажа со скидками, сервисное обслуживание и др.

Требуется постоянная программа маркетинговых мероприятий и поддержки программных продуктов. Как правило, для каждого программного продукта существует своя форма кривой продаж, которая отражает спрос (рис. 1).

Вначале продажа программного продукта идет вверх - возрастающий участок кривой. Затем наступает стабилизация продаж программного продукта. Фирма-разработчик стремится к максимальной длительности периода стабильных продаж на высоком уровне. Далее происходит падение объема продаж, что является сигналом к изменению маркетинговой политики фирмы в отношении данного программного продукта, требуется модификация данного продукта изменение цены или снятие с продажи.

Рис. 1. Кривая продаж программного продукта

Эксплуатация программного продукта «идет» параллельно с его сопровождением, при этом эксплуатация программ может начинаться и в случае отсутствия сопровождения или продолжаться в случае завершения сопровождения еще какое-то время. После снятия программного продукта с продажи определенное время также может выполняться его сопровождение. В процессе эксплуатации программного продукта производится устранение обнаруженных ошибок.

Снятие программного продукта с продажи и отказ от сопровождения происходят, как правило, в случае изменения технической политики фирмы-разработчика, неэффективности работы программного продукта, наличия в нем неустранимых ошибок, отсутствия спроса.

Длительность жизненного цикла для различных программных продуктов неодинакова. Для большинства современных программных продуктов длительность жизненного цикла измеряется в годах (2-3 года). Хотя достаточно часто

встречаются на компьютерах и давно снятые с производства программные продукты.

Особенность разработки программного продукта заключается в том, что на начальных этапах принимаются решения, реализуемые на последующих этапах. Допущенные ошибки, например, при спецификации требований к программному продукту, приводят к огромным потерям на последующих этапах разработки или эксплуатации программного продукта и даже к неуспеху всего проекта. Так, при необходимости внесения изменений в спецификацию программного продукта следует повторить в полном объеме все последующие этапы проектирования и создания программного продукта.

Понятие жизненного цикла программного изделия.

Сущность развития ПИ во времени отражает объективная экономическая категория "цикл жизни". Предпосылкой для введения этой категории в сферу программных разработок послужило определение ПИ как изделия, имеющего самостоятельное значение, процессы проектирования и изготовления которого аналогичны процессам, связанным с производством (изготовлением). Как и любое изделие, ПИ имеет свой **цикл жизни**, т.е. интервал времени от начального момента возникновения объективной необходимости в ПИ до момента изъятия его из эксплуатации. Следует подчеркнуть, что жизненный цикл программного изделия заканчивается в результате его морального, а не физического износа (что типично для промышленных изделий). В свою очередь, говорят, что ПИ морально устарело, если оно перестает удовлетворять актуальным требованиям, а дальнейшая его модификация не представляется возможной или не выгодна, что влечет за собой необходимость в разработке нового ПИ.

В результате анализа проблем, требующих решения в процессе создания, изучения опыта разработок и использования различных ПИ было установлено, что независимо от специфических условий, видов и требований к конкретным ПИ можно выделить некоторую типовую последовательность этапов разработки и функционирования ПИ.

В общем виде за период своего жизненного цикла ПИ проходит три фазы: разработку, использование, сопровождение. **Фаза разработки** начинается с анализа осуществимости проекта, а далее путем последовательной трансформации преобразования от требований пользователя в форму, доступную для реализации на ЭВМ. К тому же на протяжении всей фазы закладываются основные характеристики качества будущего ПИ, которые проявляются на стадии его эксплуатации. На эту фазу приходятся, как правило, 50% стоимости ПИ и 32% трудозатрат.

Фаза использования начинается тогда, когда изделие передается пользователю, находится в действии и используется эффективно. В фазе использования обычно выполняются обучение персонала, внедрение, настройка, сопровождение и, возможно, расширение ПИ.

Фазу сопровождения также называют фазой продолжающейся разработки. Практиками признано, что эта часть жизненного цикла (ЖЦ) должна приниматься во внимание с момента начала разработки с целью совершенство-

вания ПИ в соответствии с потребностями пользователя. Процесс сопровождения, продолжающийся параллельно собственно эксплуатации ПИ, практически в течение всего времени его использования состоит из выявления и устранения ошибок в программах и изменения их функциональных возможностей. Взаимосвязь фаз ЖЦ в общем виде может быть представлена следующей схемой (рис. 2).

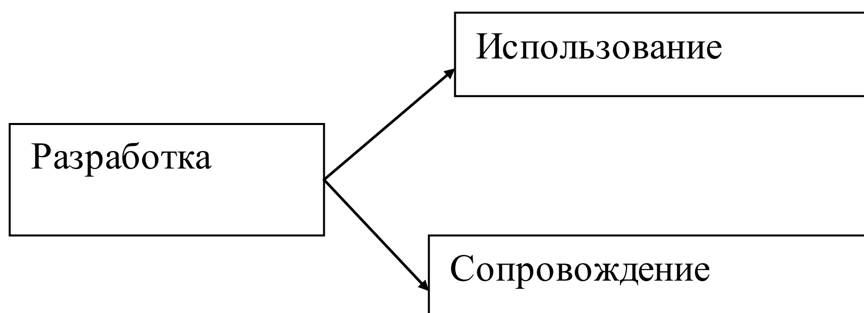


Рис. 2. Взаимосвязь фаз жизненного цикла

Структура трудозатрат на различные виды деятельности по сопровождению такова, что на изменение функциональных возможностей ПИ уходит около 78% времени, а на выявление ошибок - 17%.

Подходы к определению жизненного цикла.

Уточняет общую схему ЖЦ подход, связанный с определением основных стадий ЖЦ ПИ, по которым в литературе встречаются различия в терминологии, но единый по существу. ЖЦ ПИ образует четыре основные стадии (рис 3).

1- **Исследование и проектирование**. Изучение и анализ существующих ПИ, документирование; анализ осуществимости; определение и спецификация требований к новому ПИ; концептуальное проектирование ПИ.

2. **Реализация**. Детализация проекта, кодирование, тестирование, установление эксплуатационных процедур.

3. **Внедрение** (сдача) в опытную эксплуатацию. Приемочные тесты, обучение пользователей.

4. **Эксплуатация и сопровождение**. Периодические процедуры обработки информации, рабочие прогоны программ, измерение производительности и других характеристик ПИ, сопровождение и модификация по мере появления новых требований.

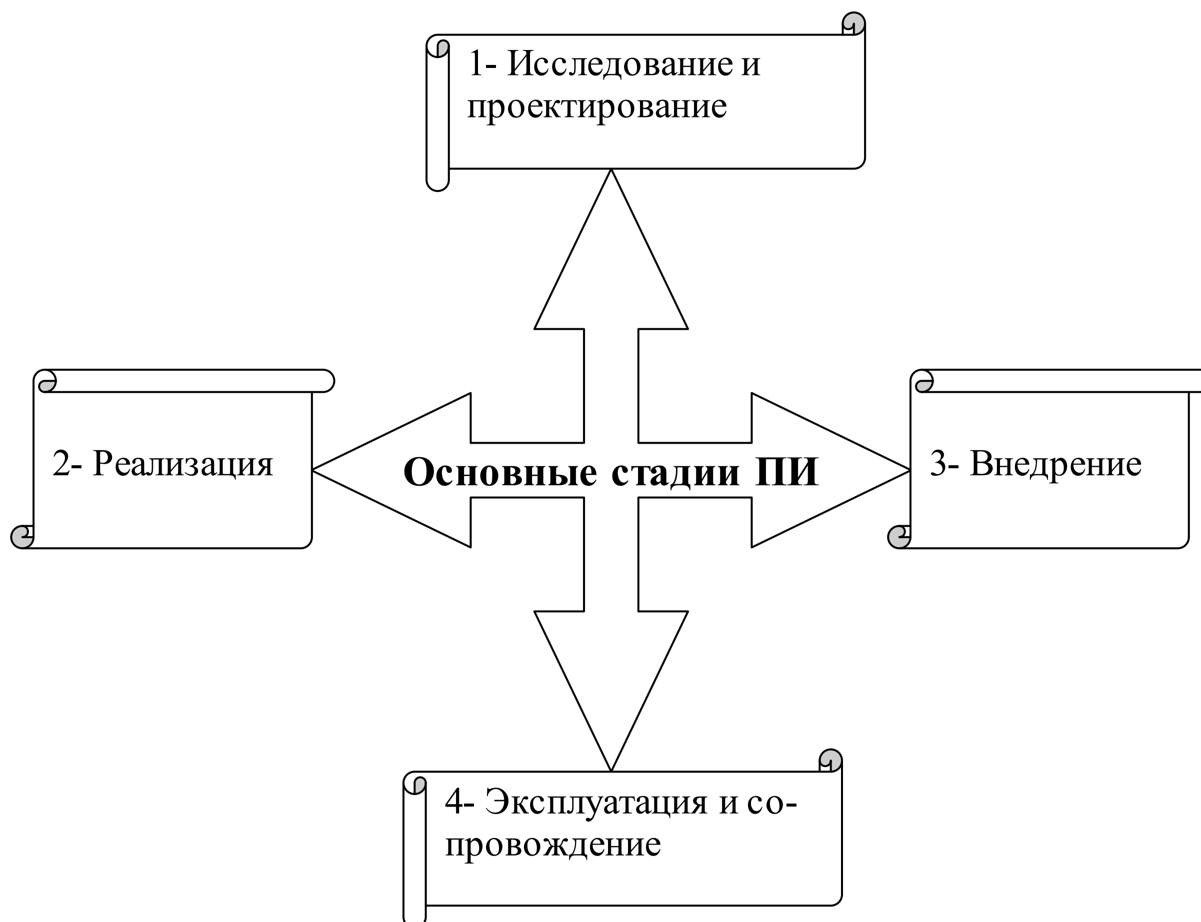


Рис. 3. Основные стадии жизненного цикла программного изделия.

Начальные этапы определяют качество проекта и успех разработки программных средств (ПС) в целом. Для начальных этапов разработки ПС типичны ситуации: выявление и четкое формулирование проблемы в условиях значительной неопределенности; выбор стратегии исследования и разработок; точное определение ПИ как системы (границы, входы, выходы, набор компонентов); выявление целей развития и функционирования ПИ; выявление функций и состава вновь создаваемого ПИ[28].

2. Содержание этапов разработки программного изделия

В силу значимости фазы разработки в ЖЦПИ рассмотрим подробнее содержание отдельных ее этапов.

В укрупненном виде жизненный цикл ПИ можно представить состоящим из следующих этапов: определение требований к системе; определение требований к ПО; предварительное проектирование; анализ и детальное проектирование; кодирование и отладка; тестирование ПИ и системы; эксплуатация и сопровождение.

Исходным этапом создания и является этап **разработки требований**, в процессе которого проводятся поисковые, исследовательские работы, формируется комплекс требований, выражающий потребности пользователя в кон-

кретном ПИ. На данном этапе будущий комплекс программ тщательно анализируется с учетом выполняемых им функций и основных свойств, обосновывается целесообразность их разработки, предварительно оцениваются трудовые и стоимостные затраты и сроки создания, вырабатываются рекомендации по выбору инструментальных средств и методов, которые предполагается использовать в процессе разработки программ. Обязательным в содержании данного этапа является также формирование требований к качеству программ в соответствии с условиями их функционирования и реализации конкретных функций. Выполнение этих работ в процессе формирования требований и формирование необходимых эксплуатационных свойств в ПИ на данном и последующих этапах их разработки позволят предотвратить дополнительные расходы, вызванные модификацией программ при их внедрении и сопровождении.

Следующим этапом в процессе создания программ является **этап проектирования**, в процессе которого требования пользователей формируются в более точном и конкретном виде Проектирование программ охватывает комплекс работ по разработке структуры программ и их компонентов; выбору языка программирования и конкретной конфигурации комплекса технических средств, на котором предполагается реализация разрабатываемых программ. В процессе проектирования решается задача выбора оптимальной структуры программ, определяющая содержание и характер работ на последующих этапах разработки. На данном этапе качество ПИ обеспечивается конкретными решениями и зависит в основном от организации управления разработкой, квалификации специалистов, использования прогрессивных методов, приемов, правил и средств проектирования программ.

После проектирования программ следует их кодирование. На практике эти этапы, как правило, частично перекрываются, т.е. за проектированием отдельных модулей выполняется их программирование, а затем, возможно, и предварительная проверка правильности функционирования разработанного модуля.

Программирование характеризуется большим числом разнообразных правил, приемов, методов и средств его выполнения, применение которых зависит от квалификации, опыта и индивидуальных особенностей программистов. Этот этап наиболее автоматизирован в процессе разработки ПИ. В настоящее время существуют десятки языков программирования и средств автоматизации, облегчающие труд программистов и повышающие его производительность, а также создающие предпосылки унификации и стандартизации процесса создания ПИ. К тому же использование современных приемов программирования, средств автоматизации, проведение различных видов проверок и контроля программирования способствуют предотвращению и выявлению значительного числа ошибок, что сокращает время и расходы на этапе отладки и тестирования программ, повышает их качество.

Этап **отладки и тестирования программ**, следующий за этапом программирования, имеет целью выявление и устранение ошибок в них, а также определение, в какой мере разработанные программы удовлетворяют требова-

ниям, сформулированным в спецификациях. Работы по отладке и тестированию программ характеризуются большой степенью повторяемости и являются наиболее утомительными и дорогостоящими. В связи с этим уделяется большое внимание разработке и использованию различных системных и инструментальных средств, автоматизирующих выполнение работ на данном этапе, что позволяет повысить качество разрабатываемых программ и снизить трудоемкость их создания. По оценкам специалист отладку и тестирование программ затрачивается до половины общих средств на разработку программ, что, тем не менее, не исключает наличие в них ошибок.

В связи с тем, что каждая из названных стадий может быть представлена различным набором составляющих ее этапов, в настоящее время существует значительное количество моделей ЖЦ. Разнообразие моделей с теоретической точки зрения объясняется тем, что термин "**жизненный цикл**" тождествен методу жизненного цикла, а сам метод включает:

• идентификацию количества фаз (этапов) с указанием результатов, получаемых на каждой из них;

• определение числа функций и видов работ, которые должны быть выполнены на каждой фазе;

• отображение результатов этапов разработки, т.е. подготовка документации, которая должна быть выдана в конце каждой фазы.

С практической точки зрения отсутствие в использовании единой модели объясняется тем, что ЖЦ моделируется либо порождается под воздействием определенной внешней среды, а не наоборот. К основным объектам, формирующим внешнюю среду, относят организационно-технологические условия разработки ПИ; условия их сопровождения и эксплуатации; технические условия, структуру и режим использования ПИ; опыт и организацию коллектива разработчиков; класс решаемых задач с учетом их типа и прикладной области, и наконец, количество этапов и их наполнение в ЖЦ зависят от выделенных ресурсов (временных, трудовых, стоимостных).

Обратим внимание, что во всех приведенных вариантах схем жизненного цикла не выделен момент (что является очевидным и необходимым дл" промышленных изделий) их изготовления (тиражирования). Это можно объяснить двумя причинами. Первая - невозможность разделить во времени и пространстве процесс разработки и собственно производство ПС. Вторая - ПС, как уже отмечалось, не имеет физического износа. Обладая свойством фиксироваться на различных носителях, ПС не теряет и не меняет при этом свое содержание. Эта особенность позволяет организовать их тиражирование и многократное применение с минимальными техническими и временными затратами. Приведенные схемы, помимо различий в количестве этапов различаются еще и последовательностью их выполнения во времени, имея в виду начало каждого следующего этапа только после завершения предыдущего либо частичное наложение последующего этапа на предыдущий. В этом смысле наиболее известна и имеет широкое практическое применение каскадная модель. По мнению автора, характерными чертами каскадной модели являются следующие:

• завершение каждой фазы верификации и подтверждение, цель ко-

торых - снизить уровень неопределенности, связанный с разработкой ПИ;

• циклическое повторение реализованных этапов по возможности с более раннего.

Следует отметить, что указанные характеристики являются подходом к решению проблемы разработки ПС: сложное переплетение создания, этапов и распределения работ, размытые их границы.

Данная модель одна из немногих, имеющих глубокую теоретическую и практическую проработку и их экономическое обоснование. Каждый из этапов подразделяется на виды работ и функции, выполняемые в каждую работу, и выполняется в соответствии с конечной целью этапа и подцелями работ и функций (рис. 4).

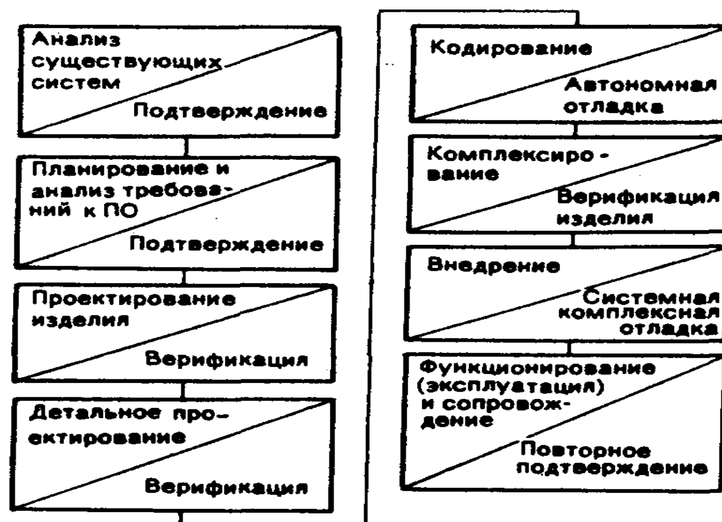


Рис. 4. Этапы создания ПИ

Завершение разработки ПС - это особый вид трудовой деятельности. В отличие от аналогичного процесса в сфере материального производства разработка программ характеризуется высокой динамичностью и значительной долей неопределенности получения результатов в установленные сроки. При разработке ПС сложно переплетаются создание, этапы и распределение работ. Этапы имеют размытые границы начала и завершения. По созданию форм и результатам труда процесс разработки программ близок к научно-исследовательским и опытно конструкторским работам. В нем реализуются функции исследования, творческого поиска и обоснования принимаемых решений.

Краткие выводы

В данной теме дано определение понятия жизненного цикла программного изделия, также его фаз и их взаимосвязь. Кроме того, даны подходы к определению жизненного цикла и подробно приведены содержание каждого из этапов разработки программного изделия.

Ключевые слова и определения

Жизненный Цикл Программного Изделия (ЖЦПИ) – схема или временная модель отражающая основные этапы существования ПС от разработки до завершения эксплуатации пользователями.

Верификация – проверка правильности кода или иных конструкций в тексте программы.

Комплексирование – совместная отладка модулей программы в виде единого программного комплекса.

Внедрение – комплекс мероприятий по установке программы на ЭВМ пользователя, ее тестирование на реальных данных и обеспечение функционирования для решения задач.

Моральный износ ПС – нецелесообразность эксплуатации ПС в реальных условиях в связи с изменением среды эксплуатации или иных внешних факторов.

Вопросы для обсуждения и контроля:

1. Что означает термин Жизненный Цикл программного Обеспечения?
2. Назовите основные этапы (фазы) ЖЦПИ.
3. Какие подходы используются для различных моделей ЖЦПИ?
4. Как соотносятся во времени фазы сопровождения и модернизации ПС?
5. На каком этапе ЖЦПИ осуществляется определение основных характеристик ПС?
6. На каком этапе производится написание кода программы и ее тестирование?
7. Перечислите основные стадии разработки ПС.
8. Какие виды верификации программы осуществляются на этапе кодирования?

Рекомендуемая литература

1.Информатика: Учебник. / Под ред. Н.В.Макаровой. 3-е перераб. изд. - М.: ФиС, 2004. 326-334 стр.

2.Могилев А.В. Информатика: Учеб.пособие для студентов пед.вузов /А.В. Могилев, Н.И. Пак, Е.К.Хеннер; Под.ред. Е.К. Хеннера- 2-е изд., стер. – М.: Издательский центр «Академия», 2003.148-154 стр.

3.Гуломов С.С., Алимов Р.Х., Лутфуллаев Х.С. ва бошқалар. Ахборот тизимлари ва технологиялари. Тошкент.: ”Шарқ”, 2000 й. 292 бет.

4.Ғуломов С. С., Шермухамедов А. Т., Бегалов Б.А. «Иқтисодий информатика». Тошкент. "Ўзбекистон", 1999. 276-289 б.

Глава 4. ЭТАПЫ ПРОЕКТИРОВАНИЯ И СОЗДАНИЯ ПРОГРАММ

1. Этапы проектирования программ.
2. Постановка задачи.
3. Проектирование программы.
4. Построение модели.
5. Разработка, реализация и анализ алгоритма.
6. Тестирование и документирование программы.

1. Этапы проектирования программ

Современный подход к проектированию программ основан на декомпозиции задачи, которая в свою очередь основана на использовании абстракций. Целью при декомпозиции является создание модулей, которые представляют собой небольшие, относительно самостоятельные программы, взаимодействующие друг с другом по хорошо определенным и простым правилам. Если эта цель достигнута, то разработка отдельных модулей может осуществляться различными людьми независимо друг от друга, при этом объединенная программа будет функционировать правильно[21].

Разработка любой программы или программной системы начинается с определения требований к ней для конкретного набора пользователей и заканчивается эксплуатацией системы этими пользователями.

Существуют различные подходы и технологии разработки алгоритмов и программ. Хотя программирование в значительной степени искусство, тем не менее, можно систематизировать и обобщить накопленный профессиональный опыт. По современным взглядам проектирование и разработку программ целесообразно разбить на ряд последовательных этапов:

1) постановка задачи;
2) проектирование программы;
3) построение модели;
4) разработка алгоритма;
5) реализация алгоритма;
6) анализ алгоритма и его сложности;
7) тестирование программы;
8) документирование.

Кратко остановимся на каждом из этих этапов.

2. Постановка задачи

При постановке задачи для крупных компьютерных программ необходимо провести следующие работы (рис 1).

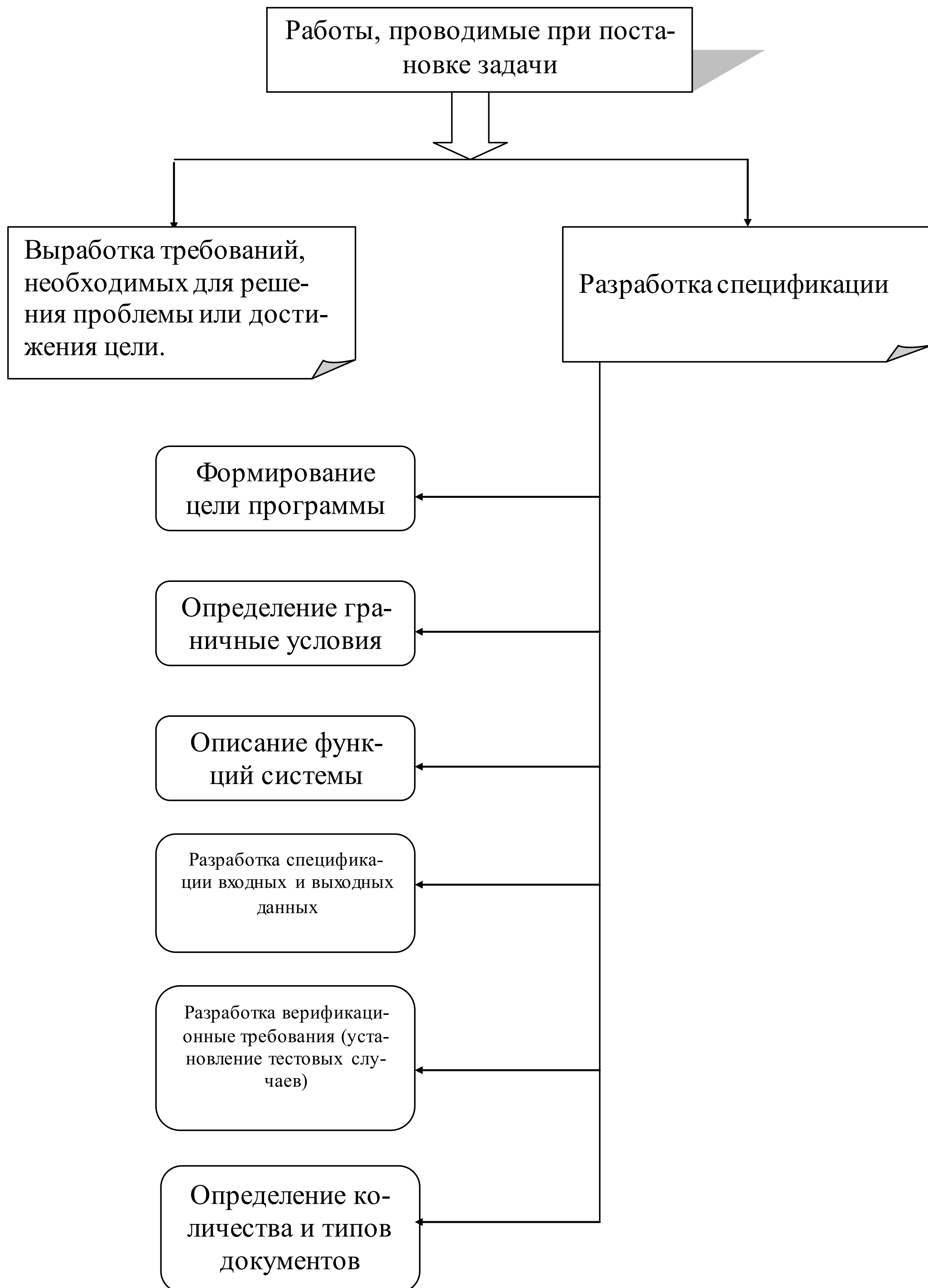


Рис. 1. Работы, проводимые при постановке задачи

В ходе этой работы выявляются свойства, которыми должна обладать система в конечном виде (замысел), описываются функции системы, характеристики интерфейса.

Чтобы приступить к решению задачи необходимо точно ее сформулировать. В первую очередь, это означает определение исходных и выходных данных, т.е. ответы на вопросы: а) что дано; б) что нужно найти. Дальнейшая детализация постановки задачи представляет собой ответы на серию вопросов такого рода:

- как определить решение;
- каких данных не хватает и все ли они нужны;
- какие сделаны допущения и т.п.

3. Проектирование программы

Сначала производится проектирование архитектуры программной системы. Это предполагает первичную (общую) стадию проектирования и заканчивается декомпозицией спецификаций в структуру систем. Обычно на модульном уровне по каждому модулю разрабатывается специфика модуля:

- имя/цель - дается имя модулю и предложение о функции модуля с формальными параметрами;
- неформальное описание - обзор действий модуля;
- ссылки - какие модули ссылаются на него и на какие модули ссылается данный модуль;
- вход/выход - формальные и фактические параметры, глобальные, локальные и связанные (общие для ряда модулей) переменные;
- примечания - полезные комментарии общего характера по модулю.

Следующим шагом является детальное проектирование. На этом этапе происходит процедурное описание программы, выбор и оценка алгоритма для реализации каждого модуля. Входной информацией для проектирования являются требования и спецификации системы.

Для проектирования программ существует различные подходы и методы. Современный подход к проектированию основан на декомпозиции, которая, в свою очередь, основана на использовании абстракции. Целью при декомпозиции является создание модулей, которые взаимодействуют друг с другом по определенным и простым правилам. Декомпозиция используется для разбиения программы на компоненты, которые затем могут быть объединены.

Методы проектирования архитектуры делятся на две группы:

1)ориентированные на обработку и

2) ориентированные на данные.

Методы, ориентированные на обработку, включают следующие общие идеи.

а) Модульное программирование.

Основные концепции:

- каждый модуль реализует единственную независимую функцию;
- имеет единственную точку входа/выхода;
- размер модуля минимизируется;
- каждый модуль разрабатывается независимо от других модулей;
- система в целом построена из модулей.

Исходя из этих принципов, каждый модуль тестируется отдельно, затем после кодирования и тестирования происходит их интеграция и тестируется вся система.

б) Функциональная декомпозиция.

Подобна стратегии «разделяй и управляй». Практически является декомпозицией в форме пошаговой детализации и концепции скрытия информации. Каждый модуль характеризуется субъективным решением проектировщика, связь осуществляется с помощью хорошо организованных интерфейсов.

в) Проектирование с использованием потока данных.

Использует поток данных как генеральную линию проектирования программы. Содержит элементы структурного проектирования сверху-вниз с пошаговой детализацией:

- экспертиза потоков данных и отображение графа потока данных;
- анализ входных, центральных и выходных преобразующих поток данных элементов;
- формирование иерархической структуры программы;
- детализация и оптимизация структуры программы.

г) Технология структурного анализа проекта.

Она основана на структурном анализе с использованием специальных графических средств построения иерархических функциональных связей между объектами системы. Данная технология эффективна на ранних стадиях создания системы, когда диаграммы просты и читаемы.

Методы проектирования, основанные на использовании структур данных, описаны ниже.

а) Методология Джексона.

Здесь структура данных - ключевой элемент в построении проекта. Структура программы определяется структурой данных, подлежащих обработке. Программа представляется как механизм, с помощью которого входные данные преобразуются выходные. В методе предусматривается:

- разработка и изображение структуры входных и выходных данных;
- изображение структуры программы путем соединения изображений этих структурных элементов;
- определение дискретных операций над структурами данных;
- построение алгоритмов обработки структур данных.

б) Методология Уорнера.

Подобна предыдущей методологии, но процедура проектирования более детализирована. Используются следующие виды представления проекта:

• диаграммы организации данных (описывают входные и выходные данные);
• диаграммы логического следования (логический поток этих данных);
• список инструкций (команды, используемые в проекте);
• псевдокод (описание проекта);
• определение входных данных системы;
• организация входных данных в иерархическую структуру;
• детальное определение формата элементов входного файла;
• то же самое для выходных данных;
• спецификация программы: чтение, ветвление, вычисление, выходы, вызовы подпрограмм;
• составление диаграммы (по типу блок-схем) указывающие логическую последовательность инструкций.

в) Метод иерархических диаграмм.

В этом методе определяется связь между входными, выходными данными и процессом обработки с помощью иерархической декомпозиции системы (без детализации). По сути используются три элемента: вход, обработка, выход[21].

Алгоритм проектирования по этому методу заключается в следующих шагах:

- начать с наивысшего уровня абстракции, определив вход, выход, обработку;
- соединить каждый элемент входа и выхода с соответствующей обработкой;
- документировать каждый элемент системы, используя диаграммы;
- детализировать диаграммы, используя шаги 1-3

г) Объектно-ориентированная методология проектирования.

Основанная на концепции упрятывания информации и абстрактных типов данных. Рассматриваются данные, модули и системы в качестве объектов. Каждый объект содержит некоторую структуру данных с набором процедур, знающих как работать с этими данными. По этой методологии создаются абстракции по заданной проблемной области:

• определение проблемы;
• развитие неформальной стратегии, удовлетворяющей требованиям к системе;
• формализация стратегии;
• создание объектов и их атрибутов;
• определение операций над объектами;
• установка интерфейсов;
• реализация операций.

4. Построение модели

Построение модели в большинстве случаев является непростой задачей. Чтобы приобрести опыт в моделировании, необходимо изучить как можно больше известных и удачных моделей.

При построении моделей, как правило, используют два принципа: дедуктивный (от общего к частному) и индуктивный (от частного к общему).

При дедуктивном подходе (рис.2) рассматривается частный случай общеизвестной фундаментальной модели. Здесь при заданных предположениях известная модель приспосабливается к условиям моделируемого объекта. Например, можно построить модель свободно падающего тела на основе известного закона Ньютона $ma = mg - F_{сопр}$ и в качестве допустимого приближения принять модель равноускоренного движения для малого промежутка времени.

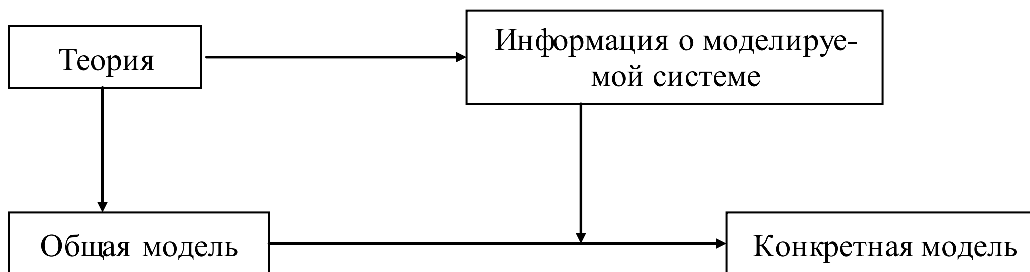


Рис 2. Схема построения модели при дедуктивном способе

Индуктивный способ (рис. 3) предполагает выдвижение гипотез, декомпозицию сложного объекта, анализ, затем синтез. Здесь широко используется подобие, аналогичное моделирование, умозаключение с целью формирования каких-либо закономерностей в виде предположений о поведении системы.

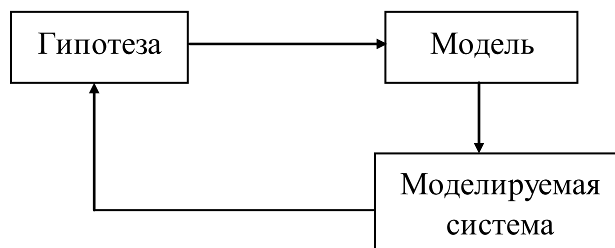


Рис 3. Схема построения модели при индуктивном способе

На рисунке 4 приведена технология построения модели при индуктивном способе.

Рис 4. Технология построения модели при индуктивном способе

5. Разработка, реализация и анализ алгоритма

Разработка алгоритма - самый сложный и трудоемкий процесс, но и самый интересный в творческом отношении. Выбор метода разработки зависит от постановки задачи, ее модели. На этом этапе необходимо провести анализ правильности алгоритма, что очень непросто и трудоемко. Наиболее распространенная процедура доказательства правильности алгоритма - это прогон его на множестве различных тестов. Однако, не гарантирует того, что не может существовать случая, в котором программа «не сработает». В общей методике доказательства правильности алгоритма предполагают, что алгоритм описан в виде последовательности шагов. Для каждого шага предлагается некое обоснование его правильности для всех подходящих входных (условиях до данного шага) и выходных данных (условиях после этого шага). Затем предлагается до-

казательство конечности алгоритма с окончательными исходными входными и выходными данными.

На этапе реализации алгоритма происходит конструирование и реализация алгоритма, включая:

• кодирование;
• интеграцию;
• тестирование (сертификацию).

По сути проводится перевод проекта в форму программы для конкретного компьютера, сборка системы и ее прогон при тестовых и нормальных условиях для подтверждения ее работы в соответствии со спецификациями системы. Этот этап зависит от того, какой язык программирования выбран, на каком компьютере алгоритм будет реализован. С этим связаны выбор типов данных, вводимых структур данных, связь с окружающей средой и т.п. Важно осознавать интерактивность, вид транслятора (компилятор или интерпретатор), наличие библиотек подпрограмм, модулей и объектов.

Анализ алгоритма и его сложности необходим для оценки ресурсов компьютеров, на которых он будет работать, времени обработки конкретных данных, приспособления в работе в локальных сетях и телекоммуникациях. Хотелось бы также иметь для данной задачи количественный критерий для сравнения нескольких алгоритмов с целью выбора более простого и эффективного среди них.

6. Тестирование и документирование программы

Перед началом эксплуатации программы необходим этап ее отладки и тестирования.

Тестирование - это процесс исполнения программ с целью выявления (обнаружения) ошибок. Тестирование - процесс деструктивный, поэтому считается, что тест удачный, если обнаружена ошибка. Хорошим считается тест, который имеет большую вероятность обнаружения еще не выявленной ошибки. Удачным считается тест, который обнаруживает еще не выявленную ошибку. Существуют различные способы тестирования программ.

Тестирование программы как «черного ящика» (стратегия «черного ящика» определяет тестирование с анализом входных данных и результатов работы программы). Критерием исчерпывающего входного тестирования является использование всех возможных наборов входных данных.

Тестирование программы как «белого ящика» заключается в стратегии управления логикой программы, позволяет использовать ее внутреннюю структуру. Критерием выступает исчерпывающее тестирование всех маршрутов и управляющих структур программы.

Разумная и реальная стратегия тестирования - сочетание моделей «черного» и «белого ящиков».

Принципы тестирования:

• описание предполагаемых значений выходных данных или результатов должно быть необходимой частью тестового набора;

• тесты для неправильных и непредусмотренных входных данных следует разрабатывать так же тщательно, как для правильных и предусмотренных;

• необходимо проверять не только делает ли программа то, для чего она предназначена, но и не делает ли она то, что не должна делать;

• нельзя планировать тестирование в предположении, что ошибки не будут обнаружены;

• вероятность наличия необнаруженных ошибок в части программы пропорциональна числу ошибок, уже обнаруженных в этой части;

• тестирование   процесс творческий.

При разработке программ очень полезным бывает метод «ручного тестирования» без компьютера на основе инспекции и сквозного просмотра (тестирование «всухую»).

Инспекция и сквозной просмотр - это набор процедур и приемов обнаружен ошибок при чтении текста.

На рисунке 5 приведены основные типы ошибок, встречающихся при программировании.

При проектировании процедуры тестирования предусматривают серии тестов, имеющих наивысшую вероятность обнаружения большинства ошибок. Для целей исчерпывающего тестирования создают эквивалентные разбиения входных параметров, причем предусматривают два класса: правильные входные данные и неправильные (ошибочные входные значения). Для каждого класса эквивалентности строят свой тест. Классом эквивалентности тестов можно назвать такое множество тестов, что выполнение алгоритма на одном из них гарантирует аналогичный результат прогона для других.

Сам процесс тестирования может быть пошаговым и/или монолитным. В том и в другом случае используют стратегии нисходящего тестирования, - начиная с верхнего, головного модуля, и затем подключая последовательно другие модули (аппарат заглушек), и восходящего тестирования, начиная с тестирования отдельных модулей.

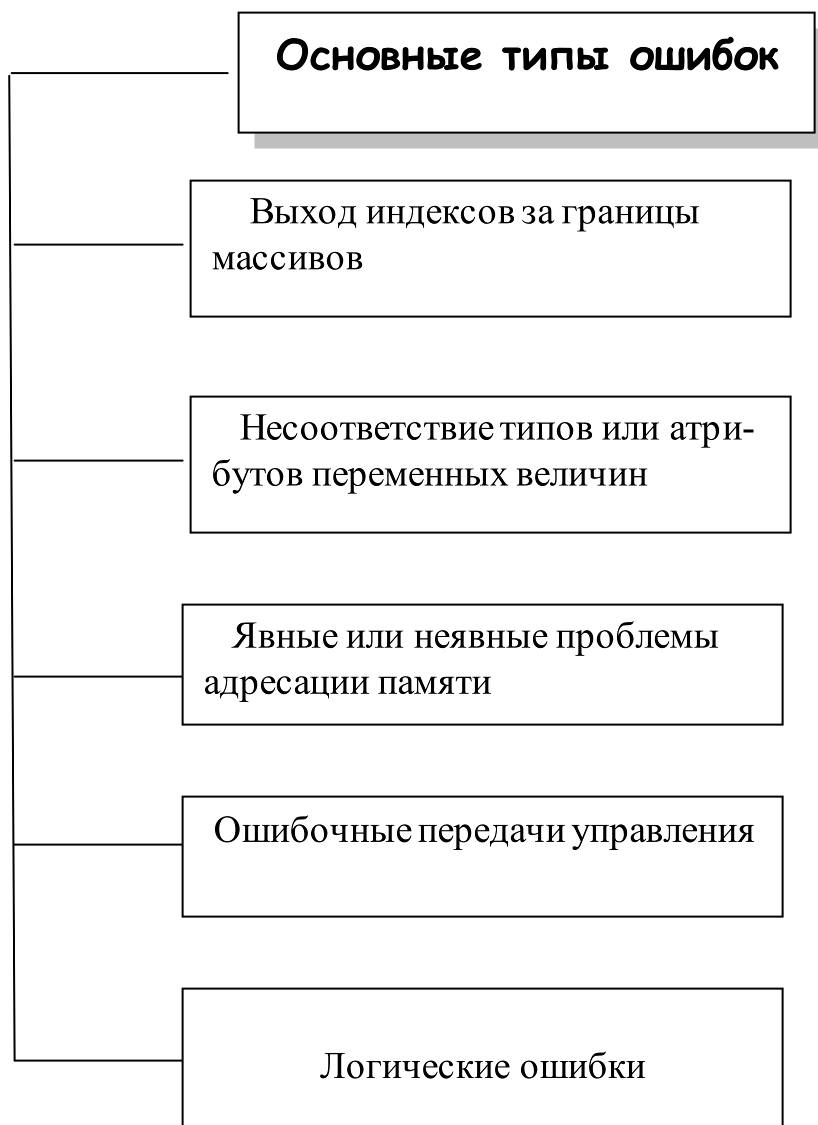


Рис. 5. Основные типы ошибок

В процессе отладки программы используют метод грубой силы - использование выводов промежуточных данных по всей программе (трассировка) или использование автоматических средств. Например, в Турбо-Паскале имеется в наличии мощный аппарат автоматической отладки программ (режим DEBUG).
Есть золотое правило программистов - оформляй свои программы в том виде, в ком бы ты хотел видеть программы, написанные другими. К каждому конечному продукту необходимо документированное сопровождение в виде (help), файлового текста (readme.txt).

Краткие выводы

В данной теме раскрыто сущность этапов проектирования и разработки программ, постановка задачи. Работы, проводимые при постановке задачи для крупных компьютерных программ. Проектирование программ. Разработка специфики модуля. Методы проектирования архитектуры: ориентированные на обработку, ориентированные на данные. Методы, ориентированные на обра-

ботку: модульное программирование, функциональная декомпозиция, проектирование с использованием потока данных, технология структурного анализа проекта.

Ключевые слова и определения

Модульное программирование - каждый модуль тестируется отдельно, затем после кодирования и тестирования происходит их интеграция и тестируется вся система.

Функциональная декомпозиция - подобна стратегии «разделяй и управляй». Практически является декомпозицией в форме пошаговой детализации и концепции скрытия информации. Каждый модуль характеризуется субъективным решением проектировщика, связь осуществляется с помощью хорошо организованных интерфейсов.

Проектирование с использованием потока данных - использует поток данных как генеральную линию проектирования программы, содержит элементы структурного проектирования сверху-вниз с пошаговой детализацией.

Технология структурного анализа проекта - основана на структурном анализе с использованием специальных графических средств построения иерархических функциональных связей между объектами системы.

Методология Джексона - структура программы определяется структурой данных, подлежащих обработке. Программа представляется как механизм, с помощью которого входные данные преобразуются выходные.

Методология Уорнера - подобна предыдущей методологии, но процедура проектирования более детализирована.

Метод иерархических диаграмм - в этом методе определяется связь между входными, выходными данными и процессом обработки с помощью иерархической декомпозиции системы (без детализации). По сути используются три элемента: вход, обработка, выход.

Дедуктивный подход - рассматривается частный случай общеизвестной фундаментальной модели.

Индуктивный способ - предполагает выдвижение гипотез, декомпозицию сложного объекта, анализ, затем синтез.

Тестирование - это процесс исполнения программ с целью выявления (обнаружения) ошибок.

Вопросы для обсуждения и контроля:

1. Из каких этапов состоит процесс разработки программ?
2. Какие действия необходимо произвести на этапе реализации алгоритма?
3. На каком этапе разработки программ решается вопрос использовать подпрограммы или нет?

4. В чем особенность разработки программ с использованием подпрограмм?
5. Как выбрать модель задачи?
6. Что происходит на этапе реализации алгоритма?
7. Что такое тестирование программы?
8. В чем отличие тестирования «белого» и «черного» ящика?
9. Какое документированное сопровождение необходимо к каждому конечному продукту?
10. Какие работы включает этап постановка задачи?
11. Какие действия производятся при проектировании архитектуры программной системы?
12. Какие действия производятся при детальном проектировании?
13. В чем различие между методами, ориентированными на обработку и методами проектирования, основанными на использовании структур данных?
14. Какой метод проектирования, по Вашему мнению, является наиболее эффективным?

Рекомендуемая литература

1. Чернев Д.А. Технология разработки программного обеспечения: (Учебное пособие).-Т.:, “Mehnat”, 2004.47-51 стр.
2. Информатика: Учебник. / Под ред. Н.В.Макаровой. 3-е перераб. изд. - М.: ФиС, 2004. –296-308 стр.
3. Информатика: Базовый курс / С.В.Симонович и др.- Санкт-Петербург: Питер, 2003. 599-605 стр.

Глава 5. МОДУЛЬНОЕ ПРОГРАММИРОВАНИЕ

1. Понятие модульного программирования.
2. Основные характеристики программного модуля.
3. Методы разработки структуры программы
4. Типовая структура программного продукта, состоящего из программных модулей.

1. Понятие модульного программирования

Модульное программирование основано на понятии модуля — логически взаимосвязанной совокупности функциональных элементов, оформленных в виде отдельных программных модулей.

Модуль характеризуют:

• один вход и один выход — на входе программный модуль получает определенный набор исходных данных, выполняет содержательную обработку и возвращает один набор результатных данных, т.е. реализуется стандартный принцип IPO (Input — Process — Output) — вход-процесс-выход;

• функциональная завершенность — модуль выполняет перечень регламентированных операций для реализации каждой отдельной функции в полном составе, достаточных для завершения начатой обработки;

• логическая независимость — результат работы программного модуля зависит только от исходных данных, но не зависит от работы других модулей;

• слабые информационные связи с другими программными модулями — обмен информацией между модулями должен быть по возможности минимизирован;

• обозримый по размеру и сложности программный элемент.

Таким образом, модули содержат определение доступных для обработки данных, операции обработки данных, схемы взаимосвязи с другими модулями.

Каждый модуль состоит из спецификации и тела. Спецификации определяют правила использования модуля, а тело — способ реализации процесса обработки.

Модульная структура программных продуктов

Принципы модульного программирования программных продуктов во многом сходны с принципами нисходящего проектирования. Сначала определяются состав и подчиненность функций, а затем — набор программных модулей, реализующих эти функции.

Однотипные функции реализуются одними и теми же модулями. Функция верхнего уровня обеспечивается главным модулем; он управляет выполнением нижестоящих функций, которым соответствуют подчиненные модули.

При определении набора модулей, реализующих функции конкретного алгоритма, необходимо учитывать следующее:

• каждый модуль вызывается на выполнение вышестоящим модулем и, закончив работу, возвращает управление вызвавшему его модулю;

• принятие основных решений в алгоритме выносится на максимально "высокий" по иерархии уровень;

• для использования одной и той же функции в разных местах алгоритма создается один модуль, который вызывается на выполнение по мере необходимости. В результате дальнейшей детализации алгоритма создается функционально-модульная схема (ФМС) алгоритма приложения, которая является основой для программирования (рис. 1).

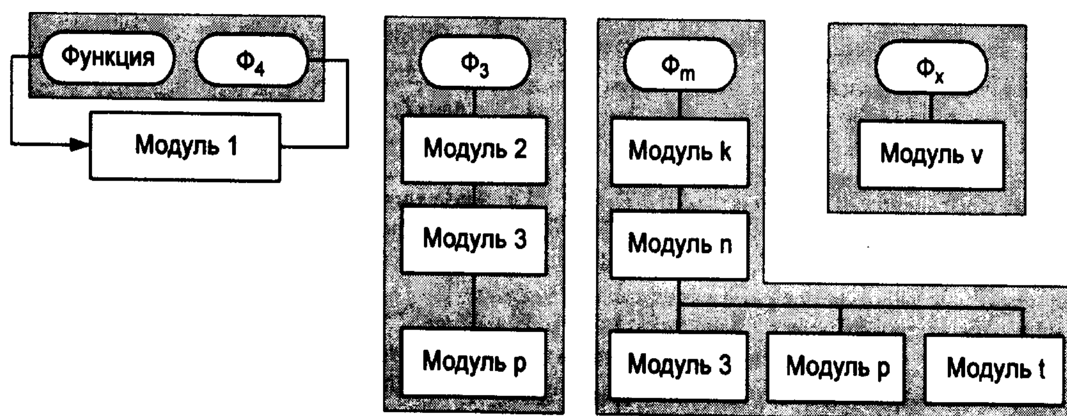


Рис. 1. Функционально-модульная структура приложения

Пример 1. Некоторые функции могут выполняться с помощью одного и того же программного модуля (например, функции $Ф_1$ и $Ф_2$).

• Функция $Ф_3$ реализуется в виде последовательности выполнения программных модулей.

• Функция $Ф_m$ реализуется с помощью иерархии связанных модулей.

• Модуль n управляет выбором на выполнение подчиненных модулей.

• Функция $Ф_x$ реализуется одним программным модулем.

Состав и вид программных модулей, их назначение и характер использования в программе в значительной степени определяются инструментальными средствами. Например, применительно к средствам СУБД отдельными модулями могут быть:

- экранные формы ввода и/или редактирования информации базы данных;
- отчеты генератора отчетов;
- макросы;
- стандартные процедуры обработки информации;
- меню, обеспечивающее выбор функции обработки и др.

Алгоритмы большой сложности обычно представляются с помощью схем двух видов:

• обобщенной схемы алгоритма — раскрывает общий принцип функционирования алгоритма и основные логические связи между отдельными модуля-

ми на уровне обработки информации (ввод и редактирование данных, вычисления, печать результатов)

- детальной схемы алгоритма — представляет содержание каждого элемента обобщенной схемы с использованием управляющих структур в блок-схемах алгоритма, псевдокода либо алгоритмических языков высокого уровня.

2. Основные характеристики программного модуля

Не всякий программный модуль способствует упрощению программы. Выделить хороший с этой точки зрения модуль является серьезной творческой задачей. Для оценки приемлемости выделенного модуля используются некоторые критерии. Так, Хольт предложил следующие два общих таких критерия:

- хороший модуль снаружи проще, чем внутри;
- хороший модуль проще использовать, чем построить.

Майерс предлагает для оценки приемлемости программного модуля использовать более конструктивные его характеристики:

- размер модуля,
- прочность модуля,
- сцепление с другими модулями,
- рутинность модуля (независимость от предыстории обращений к нему).

Размер модуля измеряется числом содержащихся в нем операторов или строк. Модуль не должен быть слишком маленьким или слишком большим. Маленькие модули приводят к громоздкой модульной структуре программы и могут не окупать накладных расходов, связанных с их оформлением. Большие модули неудобны для изучения и изменений, они могут существенно увеличить суммарное время повторных трансляций программы при отладке программы. Обычно рекомендуются программные модули размером от нескольких десятков до нескольких сотен операторов.

Прочность модуля – это мера его внутренних связей. Чем выше прочность модуля, тем больше связей он может спрятать от внешней по отношению к нему части программы и, следовательно, тем больший вклад в упрощение программы он может внести. Для оценки степени прочности модуля Майерс предлагает упорядоченный по степени прочности набор из семи классов модулей. Самой слабой степенью прочности обладает модуль, *прочный по совпадению*. Это такой модуль, между элементами которого нет осмысленных связей. Такой модуль может быть выделен, например, при обнаружении в разных местах программы повторения одной и той же последовательности операторов, которая и оформляется в отдельный модуль. Необходимость изменения этой последовательности в одном из контекстов может привести к изменению этого модуля, что может сделать его использование в других контекстах ошибочным. Такой класс программных модулей не рекомендуется для использования. Вообще говоря, предложенная Майерсом упорядоченность по степени прочности классов модулей не бесспорна. Однако, это не очень существенно, так как толь-

ко два высших по прочности класса модулей рекомендуются для использования. Эти классы мы и рассмотрим подробнее.

Функционально прочный модуль – это модуль, выполняющий (реализующий) одну какую-либо определенную функцию. При реализации этой функции такой модуль может использовать и другие модули. Такой класс программных модулей рекомендуется для использования.

Информационно прочный модуль – это модуль, выполняющий (реализующий) несколько операций (функций) над одной и той же структурой данных (информационным объектом), которая считается неизвестной вне этого модуля. Для каждой из этих операций в таком модуле имеется свой вход со своей формой обращения к нему. Такой класс следует рассматривать как класс программных модулей с высшей степенью прочности. Информационно прочный модуль может реализовывать, например, абстрактный тип данных.

В модульных языках программирования как минимум имеются средства для задания функционально прочных модулей (например, модуль типа FUNCTION в языке ФОРТРАН). Средства же для задания информационно прочных модулей в ранних языках программирования отсутствовали. Эти средства появились только в более поздних языках. Так в языке программирования Ада средством задания информационно прочного модуля является пакет.

Сцепление модуля – это мера его зависимости по данным от других модулей. Характеризуется способом передачи данных. Чем слабее сцепление модуля с другими модулями, тем сильнее его независимость от других модулей. Для оценки степени сцепления Майерс предлагает упорядоченный набор из шести видов сцепления модулей. Худшим видом сцепления модулей является *сцепление по содержимому*. Таким является сцепление двух модулей, когда один из них имеет прямые ссылки на содержимое другого модуля (например, на константу, содержащуюся в другом модуле). Такое сцепление модулей недопустимо. Не рекомендуется использовать также *сцепление по общей области* – это такое сцепление модулей, когда несколько модулей используют одну и ту же область памяти. Такой вид сцепления модулей реализуется, например, при программировании на языке ФОРТРАН с использованием блоков COMMON. Единственным видом сцепления модулей, который рекомендуется для использования современной технологией программирования, является *параметрическое сцепление* (сцепление по данным по Майерсу) – это случай, когда данные передаются модулю либо при обращении к нему как значения его параметров, либо как результат его обращения к другому модулю для вычисления некоторой функции. Такой вид сцепления модулей реализуется на языках программирования при использовании обращений к процедурам (функциям).

Рутинность модуля – это его независимость от предыстории обращений к нему. Модуль будем называть *рутинным*, если результат (эффект) обращения к нему зависит только от значений его параметров (и не зависит от предыстории обращений к нему). Модуль будем называть *зависящим от предыстории*, если результат (эффект) обращения к нему зависит от внутреннего состояния этого модуля, изменяемого в результате предыдущих обращений к нему. Майерс не

рекомендует использовать зависящие от предыстории (непредсказуемые) модули, так как они провоцируют появление в программах хитрых (неуловимых) ошибок. Однако такая рекомендация является неконструктивной, так как во многих случаях именно зависящий от предыстории модуль является лучшей реализаций информационно прочного модуля. Поэтому более приемлема следующая (более осторожная) рекомендация:

- всегда следует использовать рутинный модуль, если это не приводит к плохим (не рекомендуемым) сцеплениям модулей;
- зависящие от предыстории модули следует использовать только в случае, когда это необходимо для обеспечения параметрического сцепления;
- в спецификации зависящего от предыстории модуля должна быть четко сформулирована эта зависимость таким образом, чтобы было возможно прогнозировать поведение (эффект выполнения) данного модуля при разных последующих обращениях к нему.

В связи с последней рекомендацией может быть полезным определение внешнего представления (ориентированного на информирование человека) состояний зависящего от предыстории модуля. В этом случае эффект выполнения каждой функции (операции), реализуемой этим модулем, следует описывать в терминах этого внешнего представления, что существенно упростит прогнозирование поведения данного модуля.

3. Методы разработки структуры программы

Как уже отмечалось выше, в качестве модульной структуры программы принято использовать древовидную структуру, включая деревья со сросшимися ветвями. В узлах такого дерева размещаются программные модули, а направленные дуги (стрелки) показывают статическую подчиненность модулей, т.е. каждая дуга показывает, что в тексте модуля, из которого она исходит, имеется ссылка на модуль, в который она входит. Другими словами, каждый модуль может обращаться к подчиненным ему модулям, т.е. выражается через эти модули. При этом модульная структура программы, в конечном счете, должна включать и совокупность спецификаций модулей, образующих эту программу. *Спецификация* программного модуля содержит

- синтаксическую спецификацию его входов, позволяющую построить на используемом языке программирования синтаксически правильное обращение к нему (к любому его входу),
- функциональную спецификацию модуля (описание семантики функций, выполняемых этим модулем по каждому из его входов).

Функциональная спецификация модуля строится так же, как и функциональная спецификация ПС.

В процессе разработки программы ее модульная структура может поразному формироваться и использоваться для определения порядка программирования и отладки модулей, указанных в этой структуре. Поэтому можно говорить о разных методах разработки структуры программы. Обычно в литературе

обсуждаются два метода: метод восходящей разработки и метод нисходящей разработки[17].

Метод *восходящей разработки* заключается в следующем. Сначала строится модульная структура программы в виде дерева. Затем поочередно программируются модули программы, начиная с модулей самого нижнего уровня (листья дерева модульной структуры программы), в таком порядке, чтобы для каждого программируемого модуля были уже запрограммированы все модули, к которым он может обращаться. После того, как все модули программы запрограммированы, производится их поочередное тестирование и отладка в принципе в таком же (восходящем) порядке, в каком велось их программирование. Такой порядок разработки программы на первый взгляд кажется вполне естественным: каждый модуль при программировании выражается через уже запрограммированные непосредственно подчиненные модули, а при тестировании использует уже отлаженные модули. Однако, современная технология не рекомендует такой порядок разработки программы. Во-первых, для программирования какого-либо модуля совсем не требуется наличия текстов используемых им модулей – для этого достаточно, чтобы каждый используемый модуль был лишь специфицирован (в объеме, позволяющем построить правильное обращение к нему), а для тестирования его возможно (и даже, как мы покажем ниже, полезно) используемые модули заменять их имитаторами (заглушками). Во-вторых, каждая программа в какой-то степени подчиняется некоторым внутренним для нее, но глобальным для ее модулей соображениям (принципам реализации, предположениям, структурам данных и т.п.), что определяет ее концептуальную целостность и формируется в процессе ее разработки. При восходящей разработке эта глобальная информация для модулей нижних уровней еще не ясна в полном объеме, поэтому очень часто приходится их перепрограммировать, когда при программировании других модулей производится существенное уточнение этой глобальной информации (например, изменяется глобальная структура данных). В-третьих, при восходящем тестировании для каждого модуля (кроме головного) приходится создавать ведущую программу (модуль), которая должна подготовить для тестируемого модуля необходимое состояние информационной среды и произвести требуемое обращение к нему. Это приводит к большому объему «отладочного» программирования и в то же время не дает никакой гарантии, что тестирование модулей производилось именно в тех условиях, в которых они будут выполняться в рабочей программе.

Метод *нисходящей разработки* заключается в следующем. Как и в предыдущем методе сначала строится модульная структура программы в виде дерева. Затем поочередно программируются модули программы, начиная с модуля самого верхнего уровня (головного), переходя к программированию какого-либо другого модуля только в том случае, если уже запрограммирован модуль, который к нему обращается. После того, как все модули программы запрограммированы, производится их поочередное тестирование и отладка в таком же (нисходящем) порядке. При этом первым тестируется головной модуль программы, который представляет всю тестируемую программу и поэтому тестируется при «естественном» состоянии информационной среды, при котором

начинает выполняться эта программа. При этом те модули, к которым может обращаться головной, заменяются их имитаторами (так называемыми заглушками). Каждый *имитатор модуля* представляется весьма простым программным фрагментом, который, в основном, сигнализирует о самом факте обращения к имитируемому модулю, производит необходимую для правильной работы программы обработку значений его входных параметров (иногда с их распечаткой) и выдает, если это необходимо, заранее запасенный подходящий результат. После завершения тестирования и отладки головного и любого последующего модуля производится переход к тестированию одного из модулей, которые в данный момент представлены имитаторами, если таковые имеются. Для этого имитатор выбранного для тестирования модуля заменяется самим этим модулем и, кроме того, добавляются имитаторы тех модулей, к которым может обращаться выбранный для тестирования модуль. При этом каждый такой модуль будет тестироваться при «естественных» состояниях информационной среды, возникающих к моменту обращения к этому модулю при выполнении тестируемой программы. Таким образом, большой объем «отладочного» программирования при восходящем тестировании заменяется программированием достаточно простых имитаторов используемых в программе модулей. Кроме того, имитаторы удобно использовать для того, чтобы подыгрывать процессу подбора тестов путем задания нужных результатов, выдаваемых имитаторами. При таком порядке разработки программы вся необходимая глобальная информация формируется своевременно, т.е. ликвидируется весьма неприятный источник просчетов при программировании модулей. Некоторым недостатком нисходящей разработки, приводящим к определенным затруднениям при ее применении, является необходимость абстрагироваться от базовых возможностей используемого языка программирования, выдумывая абстрактные операции, которые позже нужно будет реализовать с помощью выделенных в программе модулей. Однако способность к таким абстракциям представляется необходимым условием разработки больших программных средств, поэтому ее нужно развивать.

Особенностью рассмотренных методов восходящей и нисходящей разработок (которые мы будем называть *классическими*) является требование, чтобы модульная структура программы была разработана до начала программирования (кодирования) модулей. Это требование находится в полном соответствии с водопадным подходом к разработке ПС, так как разработка модульной структуры программы и ее кодирование производятся на разных этапах разработки ПС: первая завершает этап конструирования ПС, а второе – открывает этап кодирования. Однако эти методы вызывают ряд возражений: представляется сомнительным, чтобы до программирования модулей можно было разработать структуру программы достаточно точно и содержательно. На самом деле это делать не обязательно, если несколько модернизировать водопадный подход. Ниже предлагаются конструктивный и архитектурный подходы к разработке программ, в которых модульная структура формируется в процессе программирования (кодирования) модулей.

Конструктивный подход к разработке программы представляет собой модификацию нисходящей разработки, при которой модульная древовидная структура программы формируется в процессе программирования модулей. Разработка программы при конструктивном подходе начинается с программирования головного модуля, исходя из спецификации программы в целом. При этом спецификация программы принимается в качестве спецификации ее головного модуля, который полностью берет на себя ответственность за выполнение функций программы. В процессе программирования головного модуля, в случае, если эта программа достаточно большая, выделяются подзадачи (внутренние функции), в терминах которых программируется головной модуль. Это означает, что для каждой выделяемой подзадачи (функции) создается спецификация реализующего ее фрагмента программы, который в дальнейшем может быть представлен некоторым поддеревом модулей. Важно заметить, что здесь также ответственность за выполнение выделенной функции несет головной (может быть, и единственный) модуль этого поддерева, так что спецификация выделенной функции является одновременно и спецификацией головного модуля этого поддерева. В головном модуле программы для обращения к выделенной функции строится обращение к головному модулю указанного поддерева в соответствии с созданной его спецификацией. Таким образом, на первом шаге разработки программы (при программировании ее головного модуля) формируется верхняя начальная часть дерева, например, такая, которая показана на рис. 2.

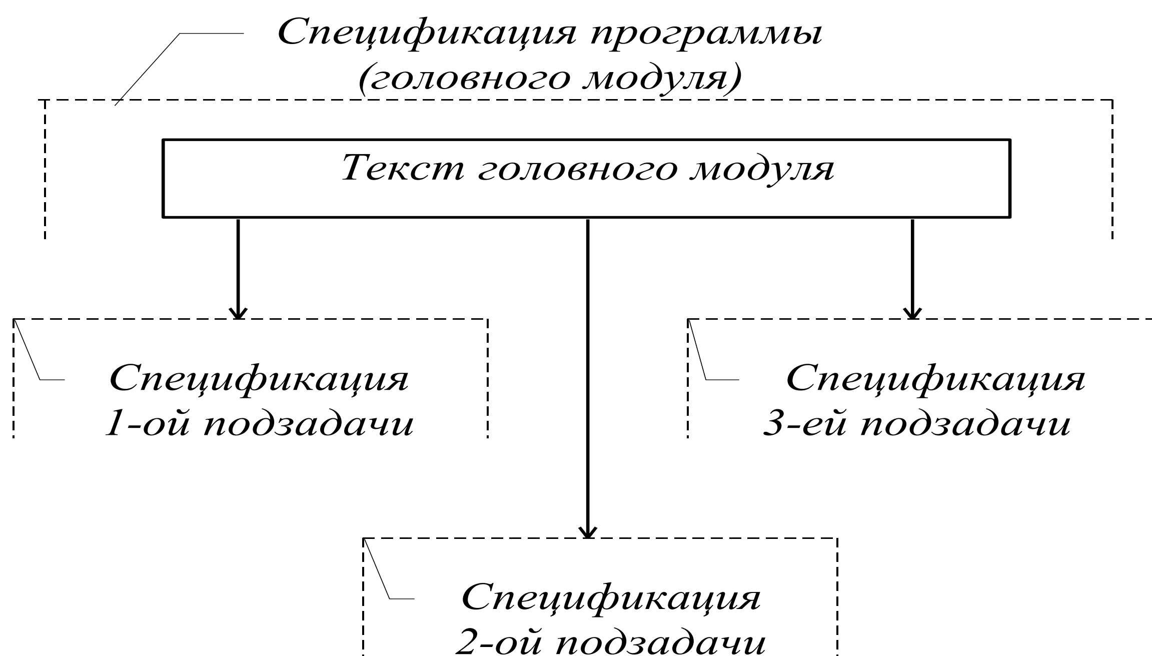


Рис. 2. Первый шаг формирования модульной структуры программы при конструктивном подходе.

Аналогичные действия производятся при программировании любого другого модуля, который выбирается из текущего состояния дерева программы из

числа специфицированных, но пока еще не запрограммированных модулей. В результате этого производится очередное доформирование дерева программы, например, такое, которое показано на рис. 3.

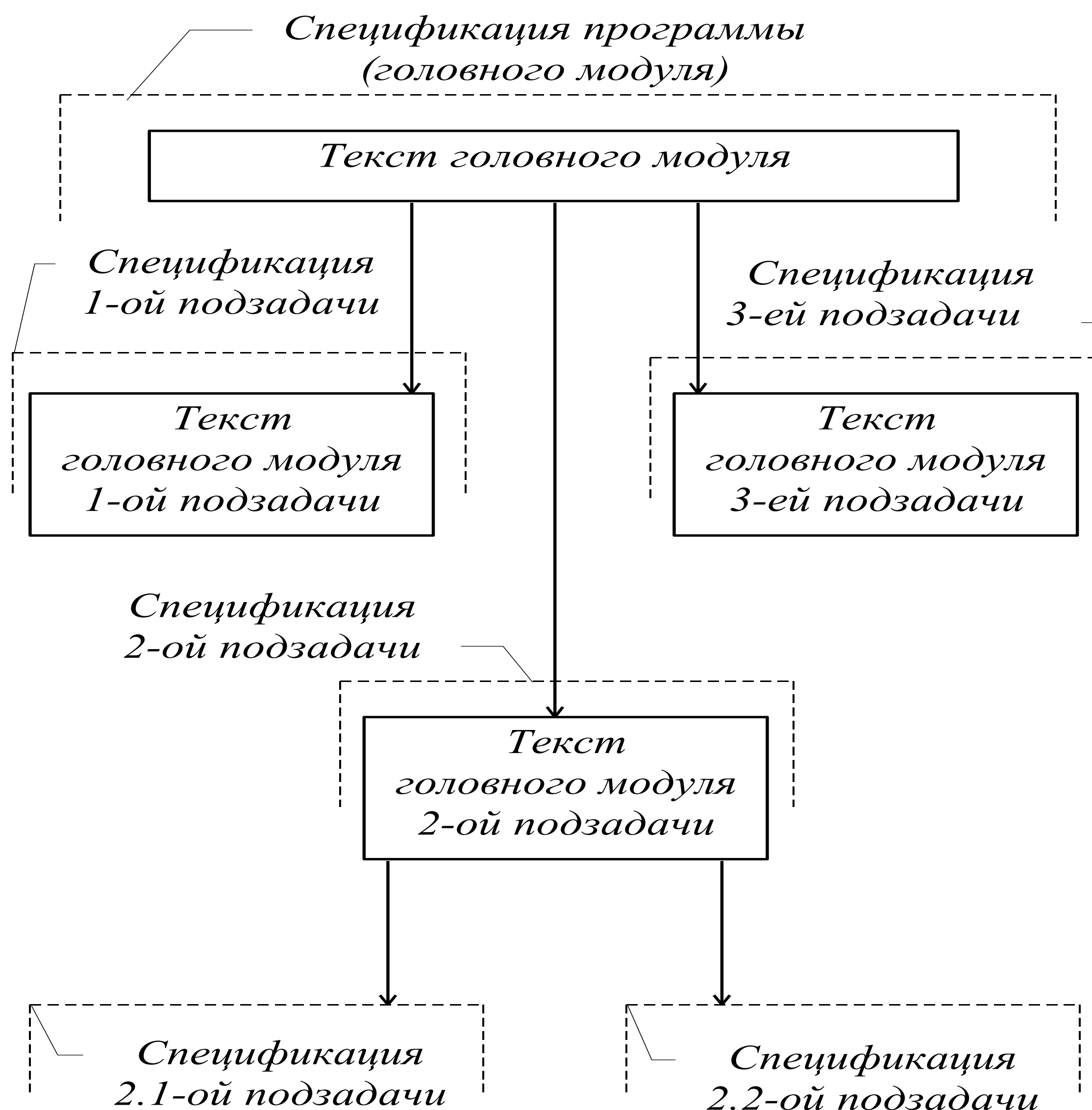


Рис. 3. Второй шаг формирования модульной структуры программы при конструктивном подходе.

Архитектурный подход к разработке программы представляет собой модификацию восходящей разработки, при которой модульная структура программы формируется в процессе программирования модуля. Но при этом ставится существенно другая цель разработки: повышение уровня используемого языка программирования, а не разработка конкретной программы. Это означает, что для заданной предметной области выделяются типичные функции, каждая из которых может использоваться при решении разных задач в этой области, и специфицируются, а затем и программируются отдельные программные модули, выполняющие эти функции. Так как процесс выделения таких функций связан с накоплением и обобщением опыта решения задач в заданной предмет-

ной области, то обычно сначала выделяются и реализуются отдельными модулями более простые функции, а затем постепенно появляются модули, использующие ранее выделенные функции. Такой набор модулей создается в расчете на то, что при разработке той или иной программы заданной предметной области в рамках конструктивного подхода могут оказаться приемлемыми некоторые из этих модулей. Это позволяет существенно сократить трудозатраты на разработку конкретной программы путем подключения к ней заранее заготовленных и проверенных на практике модульных структур нижнего уровня. Так как такие структуры могут многократно использоваться в разных конкретных программах, то архитектурный подход может рассматриваться как путь борьбы с дублированием в программировании. В связи с этим программные модули, создаваемые в рамках архитектурного подхода, обычно параметризуются для того, чтобы усилить применимость таких модулей путем настройки их на параметры.

В классическом методе нисходящей разработки рекомендуется сначала все модули разрабатываемой программы запрограммировать, а уж затем начинать нисходящее их тестирование, что опять-таки находится в полном соответствии с водопадным подходом. Однако такой порядок разработки не представляется достаточно обоснованным: тестирование и отладка модулей может привести к изменению спецификации подчиненных модулей и даже к изменению самой модульной структуры программы, так что в этом случае программирование некоторых модулей может оказаться бесполезно проделанной работой. Нам представляется более рациональным другой порядок разработки программы, известный в литературе как метод *нисходящей реализации*, что представляет некоторую модификацию водопадного подхода. В этом методе каждый запрограммированный модуль начинают сразу же тестировать до перехода к программированию другого модуля.

Все эти методы имеют еще различные разновидности в зависимости от того, в какой последовательности обходятся узлы (модули) древовидной структуры программы в процессе ее разработки. Это можно делать, например, по слоям (разрабатывая все модули одного уровня, прежде чем переходить к следующему уровню). При нисходящей разработке дерево можно обходить также в лексикографическом порядке (сверху вниз, слева направо). Возможны и другие варианты обхода дерева. Так, при конструктивной реализации для обхода дерева программы целесообразно следовать идеям Фуксмана, которые он использовал в предложенном им методе вертикального слоения. Сущность такого обхода заключается в следующем. В рамках конструктивного подхода сначала реализуются только те модули, которые необходимы для самого простейшего варианта программы, которая может нормально выполняться только для весьма ограниченного множества наборов входных данных, но для таких данных эта задача будет решаться до конца. Вместо других модулей, на которые в такой программе имеются ссылки, в эту программу вставляются лишь их имитаторы, обеспечивающие, в основном, сигнализацию о выходе за пределы этого частного случая. Затем к этой программе добавляются реализации некоторых других модулей (в частности, вместо некоторых из имеющихся имитаторов), обес-

Рис. 4. Классификация методов разработки структуры программ.

печивающих нормальное выполнение для некоторых других наборов входных данных. И этот процесс продолжается поэтапно до полной реализации требуемой программы. Таким образом, обход дерева программы производится с целью кратчайшим путем реализовать тот или иной вариант (сначала самый простейший) нормально действующей программы. В связи с этим такая разновид-

ность конструктивной реализации получила название метода *целенаправленной конструктивной реализации*. Достоинством этого метода является то, что уже на достаточно ранней стадии создается работающий вариант разрабатываемой программы. Психологически это играет роль допинга, резко повышающего эффективность разработчика. Поэтому этот метод является весьма привлекательным.

Подводя итог сказанному, на рисунке 4 представлена общая классификация рассмотренных методов разработки структуры программы.

4. Типовая структура программного продукта, состоящего из программных модулей

Для контроля структуры программы можно использовать три метода:

- статический контроль,
- смежный контроль,
- сквозной контроль.

Статический контроль состоит в оценке структуры программы, насколько хорошо программа разбита на модули с учетом значений рассмотренных выше основных характеристик модуля.

Смежный контроль сверху – это контроль со стороны разработчиков архитектуры и внешнего описания ПС. Смежный контроль снизу – это контроль спецификации модулей со стороны разработчиков этих модулей.

Сквозной контроль – это мысленное прокручивание (проверка) структуры программы при выполнении заранее разработанных тестов. Является видом динамического контроля так же, как и ручная имитация функциональной спецификации или архитектуры ПС.

Следует заметить, что указанный контроль структуры программы производится в рамках водопадного подхода разработки ПС, т.е. при классическом подходе. При конструктивном и архитектурном подходах контроль структуры программы осуществляется в процессе программирования (кодирования) модулей в подходящие моменты.

Порядок разработки программного модуля.

При разработке программного модуля целесообразно придерживаться следующего порядка:

- изучение и проверка спецификации модуля, выбор языка программирования;
- выбор алгоритма и структуры данных;
- программирование (кодирование) модуля;
- шлифовка текста модуля;
- проверка модуля;
- компиляция модуля.

Первый шаг разработки программного модуля в значительной степени представляет собой смежный контроль структуры программы снизу: изучая спецификацию модуля, разработчик должен убедиться, что она ему понятна и достаточна для разработки этого модуля. В завершении этого шага выбирается язык программирования: хотя язык программирования может быть уже предопределен для всего ПС, все же в ряде случаев (если система программирования это допускает) может быть выбран другой язык, более подходящий для реализации данного модуля (например, язык ассемблера).

На втором шаге разработки программного модуля необходимо выяснить, не известны ли уже какие-либо алгоритмы для решения поставленной и или близкой к ней задачи. И если найдется подходящий алгоритм, то целесообразно им воспользоваться. Выбор подходящих структур данных, которые будут использоваться при выполнении модулем своих функций, в значительной степени предопределяет логику и качественные показатели разрабатываемого модуля, поэтому его следует рассматривать как весьма ответственное решение.

На третьем шаге осуществляется построение текста модуля на выбранном языке программирования. Обилие всевозможных деталей, которые должны быть учтены при реализации функций, указанных в спецификации модуля, легко могут привести к созданию весьма запутанного текста, содержащего массу ошибок и неточностей. Искать ошибки в таком модуле и вносить в него требуемые изменения может оказаться весьма трудоемкой задачей. Поэтому весьма важно для построения текста модуля пользоваться технологически обоснованной и практически проверенной дисциплиной программирования. Впервые на это обратил внимание Дейкстра, сформулировав и обосновав основные принципы структурного программирования. На этих принципах базируются многие дисциплины программирования, широко применяемые на практике. Наиболее распространенной является дисциплина пошаговой детализации.

Следующий шаг разработки модуля связан с приведением текста модуля к завершенному виду в соответствии со спецификацией качества ПС. При программировании модуля разработчик основное внимание уделяет правильности реализации функций модуля, оставляя недоработанными комментарии и допуская некоторые нарушения требований к стилю программы. При шлифовке текста модуля он должен отредактировать имеющиеся в тексте комментарии и, возможно, включить в него дополнительные комментарии с целью обеспечить требуемые примитивы качества. С этой же целью производится редактирование текста программы для выполнения стилистических требований.

Шаг проверки модуля представляет собой ручную проверку внутренней логики модуля до начала его отладки (использующей выполнение его на компьютере), реализует общий принцип, сформулированный для обсуждаемой технологии программирования, о необходимости контроля принимаемых решений на каждом этапе разработки ПС.

И, наконец, последний шаг разработки модуля означает завершение проверки модуля (с помощью компилятора) и переход к процессу отладки модуля.

Краткие выводы

В данной теме рассмотрены цели разработки структуры программы, характеристики оценки приемлемости выделенного модуля: размер модуля, прочность модуля, сцепление с другими модулями, рутинность модуля, а также дано понятие программного модуля и основные характеристики программного модуля. Статический контроль состоит в оценке структуры программы.

Ключевые слова и определения

Модуль — логически взаимосвязанная совокупность функциональных элементов, оформленных в виде отдельных программных модулей.

Размер модуля измеряется числом содержащихся в нем операторов или строк.

Прочность модуля – это мера его внутренних связей.

Функционально прочный модуль – это модуль, выполняющий (реализующий) одну какую-либо определенную функцию.

Информационно прочный модуль – это модуль, выполняющий (реализующий) несколько операций (функций) над одной и той же структурой данных (информационным объектом), которая считается неизвестной вне этого модуля.

Сцепление модуля – это мера его зависимости по данным от других модулей.

Рутинность модуля – это его независимость от предыстории обращений к нему.

Метод восходящей разработки заключается в следующем. Сначала строится модульная структура программы в виде дерева. Затем поочередно программируются модули программы, начиная с модулей самого нижнего уровня (листья дерева модульной структуры программы), в таком порядке, чтобы для каждого программируемого модуля были уже запрограммированы все модули, к которым он может обращаться[32].

Метод нисходящей разработки заключается в следующем. Как и в предыдущем методе сначала строится модульная структура программы в виде дерева. Затем поочередно программируются модули программы, начиная с модуля самого верхнего уровня (головного), переходя к программированию какого-либо другого модуля только в том случае, если уже запрограммирован модуль, который к нему обращается.

Конструктивный подход к разработке программы представляет собой модификацию нисходящей разработки, при которой модульная древовидная структура программы формируется в процессе программирования модулей. Разработка программы при конструктивном подходе начинается с программирования головного модуля, исходя из спецификации программы в целом.

Архитектурный подход к разработке программы представляет собой модификацию восходящей разработки, при которой модульная структура программы формируется в процессе программирования модуля.

Вопросы для обсуждения и контроля:

1. Какую цель преследует модульное программирование?
2. Что Вы понимаете под понятием программного модуля?
3. В первую очередь на что направлено структуризация программ?
4. Какие цели преследует структуризация программных продуктов?
5. Приведите типовую структуру программного продукта?
6. Какие модули можно различить?
7. Какие критерии используются для оценки приемлемости выделенного модуля?
8. Какие характеристики используются для оценки приемлемости выделенного модуля?
9. Как определяется размер модуля?
10. Что Вы понимаете под прочностью модуля?
11. Какие классы прочных модулей Вы знаете?
12. Что Вы понимаете под сцеплением модуля?
13. Что Вы понимаете под рутинностью модуля?
14. Какие виды модулей Вы знаете?
15. Какими характеристиками должен обладать программный модуль?
16. В чем заключается метод восходящей разработки?
17. В чем отличие метода восходящей и нисходящей разработки?
18. Нарисуйте первый шаг формирования модульной структуры программы при конструктивном подходе?
19. Что собой представляет архитектурный подход к разработке программы?
20. Назовите методы контроля структуры программы.

Рекомендуемая литература

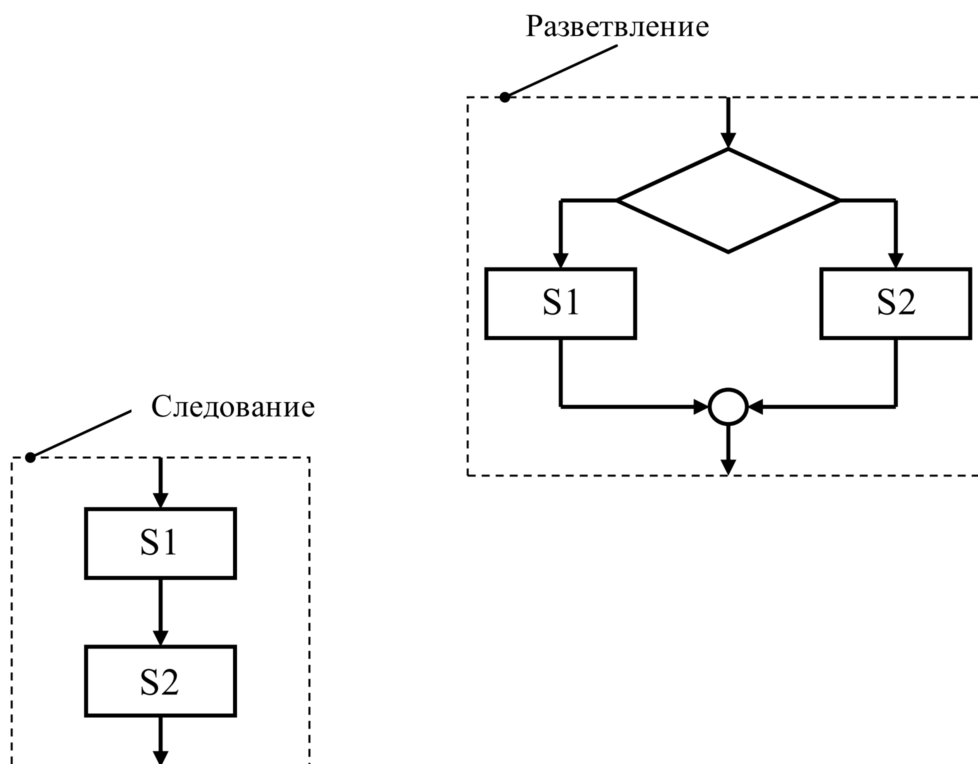
1. Балдин К.В., Уткин В.Б. Информационные системы в экономике: Учебник. – М.: Издательско-торговая корпорация «Дашков и К», 2005. 159 стр.
2. Чернев Д.А. Технология разработки программного обеспечения: (Учебное пособие).-Т.:, “Mehnat”, 2004.99 стр.
3. Информатика: Учебник. / Под ред. Н.В.Макаровой. 3-е перераб. изд. - М.: ФиС, 2004. –648-662 стр.
4. Благодатских В.А., Енгибарян М.А., Ковалевская Е.В. Экономика, разработка и использование программного обеспечения ЭВМ. - М.: Финансы и статистика, 1995. 161-168 стр.
5. Г.Майерс. Надежность программного обеспечения. М.: Мир, 1980. - С. 92-113.

Глава 6. СТРУКТУРНОЕ ПРОГРАММИРОВАНИЕ

1. Понятие структурного программирования.
2. Пошаговая детализация.
3. Понятие о псевдокоде.
4. Контроль программного продукта.

1. Понятие структурного программирования

При программировании модуля следует иметь в виду, что программа должна быть понятной не только компьютеру, но и человеку: и разработчик модуля, и лица, проверяющие модуль, и текстовики, готовящие тесты для отладки модуля, и сопроводители ПС, осуществляющие требуемые изменения модуля, вынуждены будут многократно разбирать логику работы модуля [32]. В современных языках программирования достаточно средств, чтобы запутать эту логику сколь угодно сильно, тем самым, сделать модуль трудно понимаемым для человека и, как следствие этого, сделать его ненадежным или трудно сопровождаемым. Поэтому необходимо принимать меры для выбора подходящих языковых средств и следовать определенной дисциплине программирования. В связи с этим Дейкстра и предложил строить программу как композицию из нескольких типов управляющих конструкций (структур), которые позволяют сильно повысить понимаемость логики работы программы. Программирование с использованием только таких конструкций назвали *структурным*.



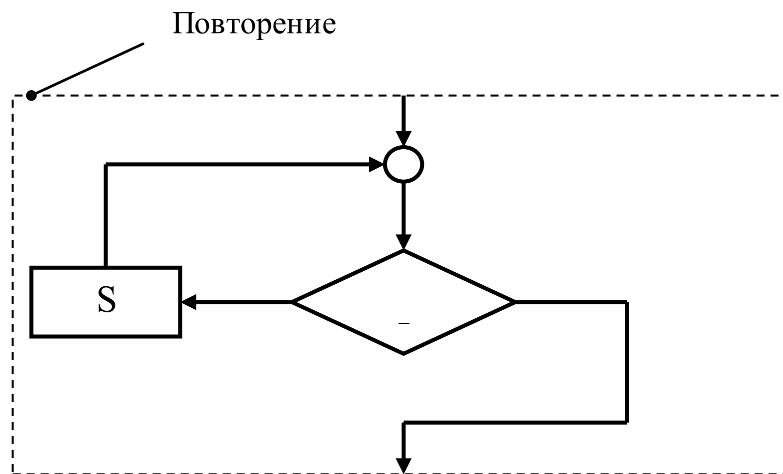


Рис. 1. Основные управляющие конструкции структурного программирования.

Основными конструкциями структурного программирования являются: следование, разветвление и повторение (см. Рис. 1). Компонентами этих конструкций являются обобщенные операторы (узлы обработки) S, S1, S2 и условие (предикат) P. В качестве обобщенного оператора может быть либо простой оператор используемого языка программирования (операторы присваивания, ввода, вывода, обращения к процедуре), либо фрагмент программы, являющийся композицией основных управляющих конструкций структурного программирования. Существенно, что каждая из этих конструкций имеет по управлению только один вход и один выход. Тем самым, и обобщенный оператор имеет только один вход и один выход.

Весьма важно также, что эти конструкции являются уже математическими объектами (что, по существу, и объясняет причину успеха структурного программирования). Доказано, что для каждой неструктурированной программы можно построить функционально эквивалентную (т.е. решающую ту же задачу) структурированную программу. Для структурированных программ можно математически доказывать некоторые свойства, что позволяет обнаруживать в программе некоторые ошибки. Этому вопросу будет посвящена отдельная лекция.

Структурное программирование иногда называют еще "программированием без GO TO". Однако дело здесь не в операторе GO TO, а в его беспорядочном использовании. Очень часто при воплощении структурного программирования на некоторых языках программирования (например, на ФОРТРАНе) оператор перехода (GO TO) используется для реализации структурных конструкций, что не нарушает принципов структурного программирования. Запутывают программу как раз "неструктурные" операторы перехода, особенно переход к оператору, расположенному в тексте модуля выше (раньше) выполняемого оператора перехода. Тем не менее, попытка избежать оператора перехода в некоторых простых случаях может привести к слишком громоздким структурированным программам, что не улучшает их ясность и содержит опасность появ-

ления в тексте модуля дополнительных ошибок. Поэтому можно рекомендовать избегать употребления оператора перехода всюду, где это возможно, но не ценой ясности программы.

К полезным случаям использования оператора перехода можно отнести выход из цикла или процедуры по особому условию, "досрочно" прекращающего работу данного цикла или данной процедуры, т.е. завершающего работу некоторой структурной единицы (обобщенного оператора) и тем самым лишь локально нарушающего структурированность программы. Большие трудности (и усложнение структуры) вызывает структурная реализация реакции на возникающие исключительные (часто ошибочные) ситуации, так как при этом требуется не только осуществить досрочный выход из структурной единицы, но и произвести необходимую обработку (исключение) этой ситуации (например, выдачу подходящей диагностической информации). Обработчик исключительной ситуации может находиться на любом уровне структуры программы, а обращение к нему может производиться с разных нижних уровней. Вполне приемлемой с технологической точки зрения является следующая "неструктурная" реализация реакции на исключительные ситуации. Обработчики исключительных ситуаций помещаются в конце той или иной структурной единицы и каждый такой обработчик программируется таким образом, что после окончания своей работы производит выход из той структурной единицы, в конце которой он помещен. Обращение к такому обработчику производится оператором перехода из данной структурной единицы (включая любую вложенную в нее структурную единицу).

2. Пошаговая детализация

Структурное программирование дает рекомендации о том, каким должен быть текст модуля. Возникает вопрос, как должен действовать программист, чтобы построить такой текст. Часто программирование модуля начинают с построения его блок-схемы, описывающей в общих чертах логику его работы. Однако современная технология программирования не рекомендует этого делать без подходящей компьютерной поддержки. Хотя блок-схемы позволяют весьма наглядно представить логику работы модуля, при их ручном кодировании на языке программирования возникает весьма специфический источник ошибок: отображение существенно двумерных структур, какими являются блок-схемы, на линейный текст, представляющий модуль, содержит опасность искажения логики работы модуля, тем более, что психологически довольно трудно сохранить высокий уровень внимания при повторном ее рассмотрении. Исключением может быть случай, когда для построения блок-схем используется графический редактор и они формализованы настолько, что по ним автоматически генерируется текст на языке программирования (как, например, это делается в Р-технологии).

В качестве основного метода построения текста модуля современная технология программирования рекомендует *пошаговую детализацию*. Сущность этого метода заключается в разбиении процесса разработки текста модуля на

ряд шагов. На первом шаге описывается общая схема работы модуля в обозримой линейной текстовой форме (т.е. с использованием очень крупных понятий), причем это описание не является полностью формализованным и ориентировано на восприятие его человеком. На каждом следующем шаге производится уточнение и детализация одного из понятий (будем называть его *уточняемым*), в каком либо описании, разработанном на одном из предыдущих шагов. В результате такого шага создается описание выбранного уточняемого понятия либо в терминах базового языка программирования (т.е. выбранного для представления модуля), либо в такой же форме, что и на первом шаге с использованием новых уточняемых понятий. Этот процесс завершается, когда все уточняемые понятия будут *уточнения* (т.е. в конечном счете будут выражены на базовом языке программирования). Последним шагом является получение текста модуля на базовом языке программирования путем замены всех вхождений уточняемых понятий заданными их описаниями и выражение всех вхождений конструкций структурного программирования средствами этого языка программирования.

Пошаговая детализация связана с использованием частично формализованного языка для представления указанных описаний, который получил название *псевдокода*. Этот язык позволяет использовать все конструкции структурного программирования, которые оформляются формализовано, вместе с неформальными фрагментами на естественном языке для представления обобщенных операторов и условий. В качестве обобщенных операторов и условий могут задаваться и соответствующие фрагменты на базовом языке программирования.

Головным описанием на псевдокоде можно считать внешнее оформление модуля на базовом языке программирования, которое должно содержать:

- начало модуля на базовом языке, т.е. первое предложение или заголовок (спецификацию) этого модуля ;
- раздел (совокупность) описаний на базовом языке, причем вместо описаний процедур и функций – только их внешнее оформление;
- неформальное обозначение последовательности операторов тела модуля как одного обобщенного оператора (см. ниже), а также неформальное обозначение тела каждого описания процедуры или функции как одного обобщенного оператора;
- последнее предложение (конец) модуля на базовом языке .

Внешнее оформление описания процедуры или функции представляется аналогично. Впрочем, если следовать Дейкстре, раздел описаний лучше также представить здесь неформальным обозначением, произведя его детализацию в виде отдельного описания.

Неформальное обозначение обобщенного оператора на псевдокоде производится на естественном языке произвольным предложением, раскрывающим в общих чертах его содержание. Единственным формальным требованием к оформлению такого обозначения является следующее: это предложение должно занимать целиком одно или несколько графических (печатных) строк и за-

вершаться точкой (или каким-либо другим знаком, специально выделенным для этого).

```
Следование:
        обобщенный_оператор
        обобщенный_оператор
  Разветвление:
        ЕСЛИ  условие  ТО
          обобщенный_оператор
        ИНАЧЕ
          обобщенный_оператор
        ВСЕ ЕСЛИ
  Повторение:
        ПОКА  условие  ДЕЛАТЬ
          обобщенный_оператор
        ВСЕ ПОКА
```

Рис. 2. Основные конструкции структурного программирования на псевдокоде.

3. Понятие о псевдокоде

Для каждого неформального обобщенного оператора должно быть создано отдельное описание, выражающее логику его работы (детализирующее его содержание) с помощью композиции основных конструкций структурного программирования и других обобщенных операторов. В качестве заголовка такого описания должно быть неформальное обозначение детализируемого обобщенного оператора. Основные конструкции структурного программирования могут быть представлены в следующем виде (см. рис. 2). Здесь условие может быть либо явно задано на базовом языке программирования в качестве булевского выражения, либо неформально представлено на естественном языке некоторым фрагментом, раскрывающим в общих чертах смысл этого условия. В последнем случае должно быть создано отдельное описание, детализирующее это условие, с указанием в качестве заголовка обозначения этого условия (фрагмента на естественном языке).

```
Выход из повторения (цикла):
      ВЫЙТИ
Выход из процедуры (функции):
      ВЕРНУТЬСЯ
Переход на обработку исключительной ситуации:
      ВОЗБУДИТЬ  имя_исключения
```

Рис. 3. Частные случаи оператора перехода в качестве обобщенного оператора.

В качестве обобщенного оператора на псевдокоде можно использовать указанные выше частные случаи оператора перехода (см. рис. 3). Последовательность обработчиков исключительных ситуаций (исключений) задается в конце модуля или описания процедуры
(функции). Каждый такой обработчик имеет вид:

```
ИСКЛЮЧЕНИЕ имя_исключения
  обобщенный_оператор
ВСЕ ИСКЛЮЧЕНИЕ
```

Отличие обработчика исключительной ситуации от процедуры без параметров заключается в следующем: после выполнения процедуры управление возвращается к оператору, следующему за обращением к ней, а после выполнения исключения управление возвращается к оператору, следующему за обращением к модулю или процедуре (функции), в конце которого (которой) помещено данное исключение.

Рекомендуется на каждом шаге детализации создавать достаточно содержательное описание, но легко обозримое (наглядное), так чтобы оно размещалось на одной странице текста. Как правило, это означает, что такое описание должно быть композицией пяти-шести конструкций структурного программирования. Рекомендуется также вложенные конструкции располагать со смещением вправо на несколько позиций (см. рис. 4). В результате можно получить описание логики работы по наглядности вполне конкурентное с блок-схемами, но обладающее существенным преимуществом – сохраняется линейность описания.

```
УДАЛЕНИЕ В ФАЙЛЕ ЗАПИСЕЙ ДО ПЕРВОЙ,
УДОВЛЕТВОРЯЮЩЕЙ ЗАДАННОМУ ФИЛЬТРУ:

  УСТАНОВИТЬ НАЧАЛО ФАЙЛА.
  ПОКА НЕ КОНЕЦ ФАЙЛА ДЕЛАТЬ
    ПРОЧИТАТЬ ОЧЕРЕДНУЮ ЗАПИСЬ.
    ЕСЛИ ОЧЕРЕДНАЯ ЗАПИСЬ УДОВЛЕТВОРЯЕТ
            ФИЛЬТРУ ТО
      ВЫЙТИ
    ИНАЧЕ
      УДАЛИТЬ ОЧЕРЕДНУЮ ЗАПИСЬ ИЗ ФАЙЛА.
    ВСЕ ЕСЛИ
  ВСЕ ПОКА
  ЕСЛИ ЗАПИСИ НЕ УДАЛЕНЫ ТО
    НАПЕЧАТАТЬ "ЗАПИСИ НЕ УДАЛЕНЫ".
  ИНАЧЕ
    НАПЕЧАТАТЬ "УДАЛЕНО N ЗАПИСЕЙ".
  ВСЕ ЕСЛИ
```

Рис. 4. Пример одного шага детализации на псевдокоде.

Идею пошаговой детализации приписывают иногда Дейкстре. Однако Дейкстра предлагал принципиально отличающийся метод построения текста модуля, который нам представляется более глубоким и перспективным. Во-первых, вместе с уточнением операторов он предлагал постепенно (по шагам) уточнять (детализировать) и используемые структуры данных. Во-вторых, на каждом шаге он предлагал создавать некоторую виртуальную машину для детализации и в ее терминах производить детализацию всех уточняемых понятий, для которых эта машина позволяет это сделать. Таким образом, Дейкстра предлагал, по существу, детализировать по горизонтальным слоям, что является перенесением его идеи о слоистых системах на уровень разработки модуля. Такой метод разработки модуля поддерживается в настоящее время пакетами языка АДА и средствами объектно-ориентированного программирования.

4.Контроль программного модуля

Применяются следующие методы контроля программного модуля:

- статическая проверка текста модуля;
- сквозное прослеживание;
- доказательство свойств программного модуля.

При статической проверке текста модуля этот текст просматривается с начала до конца с целью найти ошибки в модуле. Обычно для такой проверки привлекают, кроме разработчика модуля, еще одного или даже нескольких программистов. Рекомендуется ошибки, обнаруживаемые при такой проверке исправлять не сразу, а по завершению чтения текста модуля.

Сквозное прослеживание представляет собой один из видов динамического контроля модуля. В нем также участвуют несколько программистов, которые вручную прокручивают выполнение модуля (оператор за оператором в той последовательности, какая вытекает из логики работы модуля) на некотором наборе тестов.

Доказательству свойств программ посвящена следующая лекция. Здесь следует лишь отметить, что этот метод применяется пока очень редко.

Краткие выводы

В данной теме рассмотрены понятие структурного программирования, основные конструкции структурного программирования: следование, разветвление и повторение, программированием без GOTO. Пошаговая детализация и понятие о псевдокоде. Подробно описаны внешнее оформление модуля на базовом языке программирования: начало модуля на базовом языке, раздел (совокупность) описаний на базовом языке, неформальное обозначение последовательности операторов тела модуля как одного обобщенного оператора, последнее предложение (конец) модуля на базовом языке Контроль программного модуля.

Ключевые слова

Структурное программирование, следование, разветвление и повторение, программирование без GOTO, пошаговая детализация и понятие о псевдокоде, начало модуля на базовом языке, раздел (совокупность) описаний на базовом языке, контроль программного модуля.

Вопросы для обсуждения и контроля:

1. Дайте характеристику методу структурного программирования.
2. Назовите основные управляющие конструкции структурного программирования
3. Назовите источник ошибок при кодировании блок-схемы на язык программирования.
4. Раскройте сущность метода пошаговой детализации.
5. Какой формализованный язык используют при пошаговой детализации?
6. Как осуществляется статическая проверка текста модуля.
7. Раскройте сущность сквозного прослеживания.
8. Дайте доказательство свойств программного модуля.
9. Раскройте содержание головного описания на псевдокоде.
10. Покажите основные конструкции структурного программирования на псевдокоде.
11. Назовите частные случаи оператора перехода в качестве обобщенного оператора.

Рекомендуемая литература:

1. Балдин К.В., Уткин В.Б. Информационные системы в экономике: Учебник. – М.: Издательско-торговая корпорация «Дашков и К», 2005. 159 стр.
2. Чернев Д.А. Технология разработки программного обеспечения: (Учебное пособие).-Т.:, “Mehnat”, 2004.99 стр.
3. Информатика: Учебник. / Под ред. Н.В.Макаровой. 3-е перераб. изд. - М.: ФиС, 2004. –648-662 стр.
4. Благодатских В.А., Енгибарян М.А., Ковалевская Е.В. Экономика, разработка и использование программного обеспечения ЭВМ. - М.: Финансы и статистика, 1995. 161-168 стр.

Глава 7. МЕТОДЫ ПРОЕКТИРОВАНИЯ СВЕРХУ-ВНИЗ, HIPO-ТЕХНОЛОГИЯ И ГЛАВНОГО ПРОГРАММИСТА

1. Проектирование (программирование) сверху-вниз.
2. HIPO - технология
3. Метод главного программиста.

1. Проектирование (программирование) сверху-вниз

Программирование сверху-вниз — это некоторая многоуровневая дисциплина написания программ. На верхнем уровне исходный алгоритм представляется в виде некоторой иерархической схемы, элементы которой описываются на естественном для данной проблемы языке. Каждое такое описание можно рассматривать как последовательность комментариев, заготовок или команд некоторой гипотетической проблемно-ориентированной машины. Каждая команда такой машины моделируется или расписывается командами другой гипотетической машины более низкого уровня до команд реальной машины или операторов соответствующего языка программирования. Проектирование программной системы проводится таким образом, что описание системы на верхнем уровне не зависит от описания (работы) блоков на более низких уровнях.

Вся система проектируется и отлаживается по уровням сверху-вниз. Каждый нижний уровень отлаживается на тестах, полученных и проверенных на предыдущем верхнем уровне. В отличие от известной техники блок-схем каждый уровень оформляется как программа и отлаживается на ЭВМ обычным образом. В целом технология программирования сверху-вниз позволяет:

1) начать программирование почти одновременно и параллельно с разработкой соответствующего алгоритма;

2) формально, в виде программы некоторой гипотетической машины, фиксировать каждый этап разработки соответствующего алгоритма;

3) легко модифицировать программу по уровням путем замены одной гипотетической машины подходящей другой;

4) упрощать отладку программ путем рассредоточения ее по уровням и проведения, независимо от нижерасположенных уровней детализации.

2. HIPO - технология

HIPO-технология (Hierarchical Input Process Output) — это многоуровневая дисциплина проектирования и документирования программ. Для этой цели в HIPO-технологии разработаны шаблоны, бланки или типовые диаграммы. Таких диаграмм в HIPO-технологии обычно три. Первый тип диаграмм вспомогательный — он играет роль, аналогичную оглавлению всего проекта. Диаграммы первого типа заполняются в конце процесса проектирования при оформлении документации и в самом процессе активно не участвуют. Второй

тип диаграмм задает иерархию связи и сборки диаграмм третьего типа (IPO-диаграмм)

IPO-диаграммы являются основными в HIPO - технологии . В этих диаграммах выделены три участка, три колонки в первой колонке слева записывается входная информация (что на входе алгоритма), в последней—выходная (что на выходе), а в средней описан процесс (алгоритм, который информацию на входе преобразует в выходную). Язык заполнения IPO-диаграмм не оговаривается и может быть любым. В этом языке называется “что” (а не “как”) делает каждый программный модуль на данном уровне проектирования. Каждая IPO-диаграмма соответствует одному уровню (этапу, шагу) проектирования. На одной диаграмме должно быть не более 6—7 программных блоков, уточняющих название (характер работы) данной диаграммы. Это заставляет проектировщика системы разбить ее по уровням на небольшое число (не более 6—7) подсистем и настроить для всей системы некоторое дерево рассмотрении, проектирования. Название, присваиваемое программному блоку на одной диаграмме, должно быть кратким, общепонятным и обязательно сохраняться без изменений на всех диаграммах системы. При необходимости каждое название может быть уточнено, расширено и пояснено в специальной области IPO-диаграмм, которая называется областью спецификаций. В этой области могут быть записаны некоторые рекомендации по тому, “как” реализуется блок на данном уровне рассмотрения.

Все IPO-диаграммы имеют строго формализованную систему ссылок, которая задается наглядно на HIPO-диаграммах второго типа. Если какой-либо блок уточняется другими IPO-диаграммами, то ему присваивается соответствующий номер. Если блок не имеет уточняющего номера связи, то его детализация закончена и он может быть непосредственно закодирован на соответствующем языке программирования. Формализованная система ссылок позволяет проектировщику и администратору легко следить за состоянием и влиять на процесс разработки системы. Согласно HIPO-технологии процесс проектирования системы заканчивается только после окончания заполнения всех IPO-диаграмм проекта и увязки их друг с другом. Поэтому процесс проектирования программ сильно затягивается, зато процесс кодирования, отладки и выпуска документации осуществляется почти на автомате и может проводиться менее квалифицированным персоналом.

Очень важным в HIPO-технологии является явное указание данных на входе и выходе каждого шага алгоритма (процесса). То, что в обычной технологии удерживается программистом “в уме” 'при построении соответствующей программы, в HIPO-технологии указано явно, связано с процессом обработки и помещено структурно в одно и то же место — слева и справа проектного листа. Этим HIPO-технология существенно отличается от описанных выше методов и традиционного программирования с помощью, например, блок-схем.

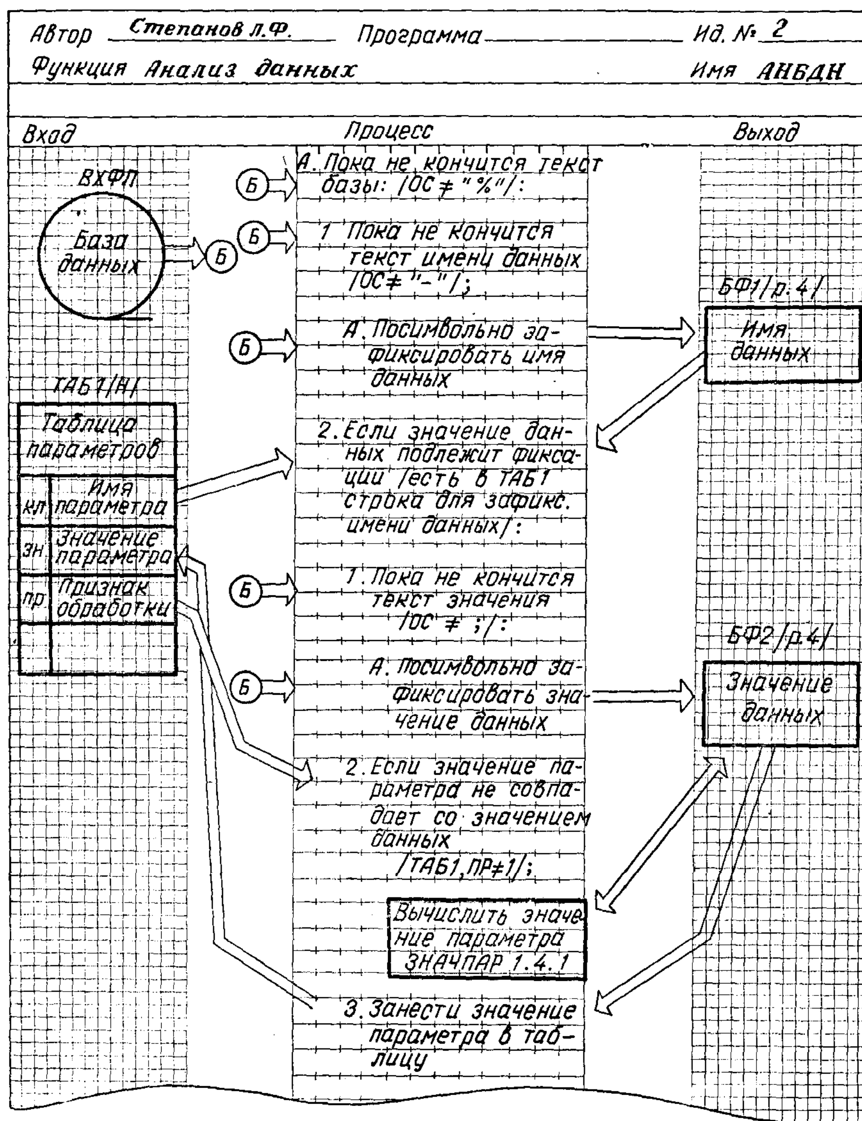


Рис.1. Пример заполнения 1РО-диаграммы

Описанные технологии являются технологиями индивидуального программирования. В них очень большую роль играют человеческий фактор и способ организации коллектива программистов для выполнения соответствующего проекта[[16].

3. Метод главного программиста.

Этот метод получил наибольшее распространение из известных методов объединения программистов. Он был разработан главным программистом фирмы *1ВМ* в процессе обобщения опыта работы по операционной системе *1ВМ/360*. **Метод главного программиста** заключается в том, что реализация программного проекта от начала до конца (проектирования, программирования, отладки, выпуска технической документации и инструкций пользователю) осуществляется под руководством главного программиста, который стоит во

главе бригады (3— 10 человек), нацеленной на реализацию данного программного проекта. В задачу бригады входит, например, ввод программ в машину, исправление синтаксических ошибок, обеспечение работ с инструментальными программами, создание проверочных примеров (тестов), переписывание, редактирование и выпуск документации, подбор и подготовка кадров и т. д.

Если программный проект большой, то состав бригады расширяется. В нее включают 1—3 ассистентов (заместителей) главного программиста, которые вместе с ним обсуждают, разрабатывают и оценивают проект, по многим вопросам взаимодействуют с остальными членами бригады, осуществляя проведение в жизнь линии главного программиста. Ассистент ищет альтернативные стратегии проектирования, оппонирует все действия главного программиста. Ассистент всегда может заменить главного, но он не отвечает ни за одну часть программы; за главным программистом всегда остается единоличное право принимать окончательное решение.

Краткие выводы

В данной теме рассмотрены понятие технологии проектирования ПО, модульное проектирование (программирование), структурное проектирование, а также проектирование (программирование) сверху-вниз. Подробно описаны понятия HIPO – технологии и IPO-диаграмм. Кроме того приведено описание метода главного программиста.

Ключевые слова и определения

Программирование сверху-вниз - это некоторая многоуровневая дисциплина написания программ. На верхнем уровне исходный алгоритм представляется в виде некоторой иерархической схемы, элементы которой описываются на естественном для данной проблемы языке.

HIPO-технология (Hiегагсhiса1 Input Pгocess Output) - это многоуровневая дисциплина проектирования и документирования программ..

Метод главного программиста заключается в том, что реализация программного проекта от начала до конца (проектирования, программирования, отладки, выпуска технической документации и инструкций пользователю) осуществляется под руководством главного программиста, который стоит во главе бригады (3— 10 человек)

Вопросы для обсуждения и контроля:

1. Что представляет из себя технология разработки программного обеспечения?
2. Назовите основные технологии разработки программного обеспечения
3. Что представляет собой программный модуль?
4. Какие типы структур используются в структурном проектировании ПС?
5. Что Вы понимаете под модульным программированием?
6. Какие виды диаграмм применяются в HIPO- технологии?
7. В чем заключается метод главного программиста?
8. Перечислите основные элементы HIPO-технологии.

Рекомендуемая литература

1.Балдин К.В., Уткин В.Б. Информационные системы в экономике: Учебник. – М.: Издательско-торговая корпорация «Дашков и К», 2005. 159 стр.

2.Чернев Д.А. Технология разработки программного обеспечения: (Учебное пособие).-Т.:, “Mehnat”, 2004.99 стр.

3.Информатика: Учебник. / Под ред. Н.В.Макаровой. 3-е перераб. изд. - М.: ФиС, 2004. –648-662 стр.

4.Благодатских В.А., Енгибарян М.А., Ковалевская Е.В. Экономика, разработка и использование программного обеспечения ЭВМ. - М.: Финансы и статистика, 1995. 161-168 стр.

Глава 8. ПРОЕКТИРОВАНИЕ ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ

1. Диалоговый режим.
2. Графический интерфейс пользователя

1. Диалоговый режим

Большинство программных продуктов, особенно прикладного характера, ориентированных на конечного пользователя, работают в *диалоговом режиме* взаимодействия с пользователем таким образом, что ведется обмен *сообщениями,* влияющими на обработку данных [22].

В диалоговом режиме под воздействием пользователя осуществляются запуск функций (метров) обработки, изменение свойств объектов, производится настройка параметров выдачи информации на печать и т.п.

Системы, поддерживающие диалоговые процессы, классифицируются на:

- системы с *жестким сценарием диалога* — стандартизированное представление информации обмена;
- *дескрипторные системы* — формат ключевых слов сообщений;
- *тезаурусные системы* — семантическая сеть дескрипторов, образующих словарь системы (аналог — гипертекстовые системы);
- системы с *языком деловой прозы* — представление сообщений на языке, естественном для профессионального пользования.

Наиболее просты для реализации и распространены диалоговые системы с жестким сценарием диалога, которые представлены в виде:

- *меню* — диалог инициируется программой; пользователю предлагается выбор альтернативы функций обработки из фиксированного перечня; предоставляемое меню может быть иерархическим и содержать вложенные подменю следующего уровня;
- действия *запрос-ответ* — фиксирован перечень возможных значений, выбираемых из списка, или ответы *типа Да/Нет;*
- *запрос по формату* — с помощью ключевых слов, фраз или путем заполнения экранной формы с регламентированным по составу и структуре набором реквизитов осуществляется подготовка сообщений.

Диалоговый процесс управляется согласно созданному *сценарию,* для которого определяются:

- точки (момент, условие) начала диалога;
- инициатор диалога — человек или программный продукт;
- параметры и содержание диалога — сообщения, состав и структура меню, экранные формы и т.п.;
- реакция программного продукта на завершение диалога. Описание сценария диалога выполняют:
- *блок-схема,* в которой предусмотрены блоки выдачи сообщений и обработки полученных ответов;

- *ориентированный граф,* вершины которого — сообщения и выполняемые действия, дуги — связь сообщений; словесное описание;
- специализированные объектно-ориентированные языки построения сценариев.

Для создания диалоговых процессов и интерфейса конечного пользователя наиболее подходят объектно-ориентированные инструментальные средства разработки программ.

В составе инструментальных средств СУБД содержатся *построители меню,* с помощью которых создается ориентированная на конечного пользователя совокупность режимов и команд в виде *главного меню* и *вложенных подменю.* Конструктор *экранных форм* СУБД используется для разработки форматов экранного ввода и редактирования данных базы данных и входной информации, управляющей работой программного продукта .

В ряде СУБД и электронных таблиц, текстовых редакторов существуют различные типы *диалоговых окон,* содержащих разнообразные объекты управления:

- тексты сообщения;
- поля ввода информации пользователя;
- списки возможных альтернатив для выбора;
- кнопки и т.п.

В среде электронных таблиц и текстовых редакторов имеются возможности настройки главных меню (удаление ненужных, добавление новых режимов и команд), создания системы подсказок с помощью встроенных средств и языков программирования

2. Графический интерфейс пользователя

Графический интерфейс пользователя — ГИП является обязательным компонентом большинства современных программных продуктов, ориентированных на работу конечного пользователя. К графическому интерфейсу пользователя предъявляются высокие требования как с чисто инженерной, так и с художественной стороны разработки, при его разработке ориентируются на возможности человека.

Наиболее часто графический интерфейс реализуется в интерактивном режиме работы пользователя для программных продуктов, функционирующих в среде Windows, и строится в виде системы спускающихся *меню* с использованием в качестве средства манипуляции мыши и клавиатуры. Работа пользователя осуществляется с *экранными формами,* содержащими *объекты управления, панели инструментов* с *пиктограммами* режимов и команд обработки.

Пример 1. Средствами редактора диалогов Microsoft Word Dialog Editor построено диалоговое окно, обеспечивающее графический интерфейс пользователя (рис. 1). К числу типовых объектов управления графического интерфейса относятся:

✓ метка (1aЬe1) — постоянный текст, не подлежащий изменению при работе пользователя с экранной формой (например, слова *Фамилия Имя Отчество);*

✓ текстовое окно (text Ьox) — используется для ввода информации произвольного вида, отображения хранимой информации в базе данных (например, для ввода фамилии студента);

Карточка студента

Фамилия Имя Отчество

Поле ввода текста

Семейное положение

Замужем, женат

Не замужем, холост

Факультативы для посещения

Информатика

Высшая математика

Физкультура

Спортивная секция

Предметы по выбору

Поле ввода текста

Рис. 1. Пример графического интерфейса пользователя.

✓ рамка (frame) — объединение объектов управления в группу по функциональному или другому принципу (например, для изменения их параметров);

✓ командная кнопка (command button) — обеспечивает передачу управляющего воздействия, например, кнопки <Сапсе1>, <ОК>, <0тмена>; выбор режима обработки типа <Ввод>, <Удаление>, <Редактирование>, <Выход> и др.;

✓ кнопка-переключатель <option button> — для альтернативного выбора кнопки из группы однотипных кнопок (например, семейное положение);

✓ помечаемая кнопка <chek button> — для аддитивного выбора несколько кнопок из группы однотипных кнопок (например, факулыпатив для посещения);

✓ окно-список (list box) — содержит список альтернативных значений для выбора (например, «Спортивная секция»);

✓ комбинированное окно (combo box) — объединяет возможности окна-списка и текстового окна (например, «Предметы по выбору» — можно указать новый предмет или выбрать один из предлагаемого списка);

✓ линейка горизонтальной прокрутки — для быстрого перемещения внутри длинного списка или текста по горизонтали;

✓ линейка вертикальной прокрутки — для быстрого перемещения внутри длинного списка или текста по вертикали;

✓ окно-список каталогов;

✓ окно-список накопителей;

✓ окно-список файлов и др.

Стандартный графический интерфейс пользователя должен отвечать ряду требований: - поддерживать информационную технологию работы пользователя с программным продуктом содержать привычные и понятные пользователю пункты меню, соответствующие функциям обработки, расположенные в естественной последовательности использования

- ориентироваться на конечного пользователя, который общается с программой на *внешнем* уровне взаимодействия;

- удовлетворять правилу "шести" — в одну линейку меню включать не более 6 понятий, каждое из которых содержит не более 6 опций;

- графические объекты сохраняют свое стандартизованное назначение и по возможности местоположение на экране.

Краткие выводы

В данной теме дано понятие диалоговой режима работы программного обеспечения, классификация систем, поддерживающие диалоговые процессы. Кроме того, приведен пример графического интерфейса пользователя, также перечень типовых объектов управления графического интерфейса.

Ключевые слова и определения

Системы с жестким сценарием диалога — стандартизированное представление информации обмена.

Дескрипторные системы — формат ключевых слов сообщений.

Тезаурусные системы — семантическая сеть дескрипторов, образующих словарь системы (аналог — гипертекстовые системы).

Системы с языком деловой прозы — представление сообщений на языке, естественном для профессионального пользования.

Меню — диалог инициируется программой; пользователю предлагается выбор альтернативы функций обработки из фиксированного перечня; предоставляемое меню может быть иерархическим и содержать вложенные подменю следующего уровня.

Запрос-ответ — фиксирован перечень возможных значений, выбираемых из списка, или ответы *типа Да/Нет*.

Запрос по формату — с помощью ключевых слов, фраз или путем заполнения экранной формы с регламентированным по составу и структуре набором реквизитов осуществляется подготовка сообщений.

Метка (label) — постоянный текст, не подлежащий изменению при работе пользователя с экранной формой (например, слова *Фамилия Имя Отчество)*.

Текстовое окно (text Ьох) — используется для ввода информации произвольного вида, отображения хранимой информации в базе данных (например, для ввода фамилии студента).

Рамка (frame) — объединение объектов управления в группу по функциональному или другому принципу (например, для изменения их параметров).

Командная кнопка (command button) — обеспечивает передачу управляющего воздействия, например, кнопки <Сапсе1>, <ОК>, <0тмена>; выбор режима обработки типа <Ввод>, <Удаление>, <Редактирование>, <Выход> и др.

Кнопка-переключатель <option button> — для альтернативного выбора кнопки из группы однотипных кнопок (например, семейное положение).

Помечаемая кнопка <chek button> — для аддитивного выбора несколько кнопок из группы однотипных кнопок (например, факулыпатив для посещения).

Окно-список (list box) — содержит список альтернативных значений для выбора (например, «Спортивная секция»).

Комбинированное окно (combo box) — объединяет возможности окна-списка и текстового окна (например, «Предметы по выбору» — можно указать новый предмет или выбрать один из предлагаемого списка).

Вопросы для обсуждения и контроля:

1. Какие программные продукты пользуются устойчивым спросом на нашем рынке программных продуктов?
2. Какие требования предъявляют пользователи к интерфейсу пользователя?
3. Какие диалоговые системы и графический интерфейс используются для разработки программных продуктов?
4. Каким сценарием пользуются при управлении диалоговыми процессами?
5. Каким требованиям должен отвечать, по Вашему мнению, стандартный графический интерфейс?
6. Как Вы думаете, не лучше ли для маленьких фирм производить программы с минимальным графическим интерфейсом?

Рекомендуемая литература

1. Информатика: Учебник. / Под ред. Н.В.Макаровой. 3-е перераб. изд. - М.: ФиС, 2004.

2. Могилев А.В. Информатика: Учеб.пособие для студентов пед.вузов /А.В. Могилев, Н.И. Пак, Е.К.Хеннер; Под.ред. Е.К. Хеннера- 2-е изд., стер. – М.: Издательский центр «Академия», 2003.

3. Гуломов С.С., Алимов Р.Х., Лутфуллаев Х.С. ва бошқалар. Ахборот тизимлари ва технологиялари. Тошкент.: ”Шарқ”, 2000 й.

4. Fуломов С. С., Шермухамедов А. Т., Бегалов Б.А. «Иқтисодий информатика». Тошкент. "Ўзбекистон", 1999.

Глава 9. ТЕСТИРОВАНИЕ И ОТЛАДКА ПРОГРАММНОГО СРЕДСТВА

Лишь та - ошибка, что не исправляется.
Конфуций

1. Основные понятия.
2. Принципы и виды отладки программного средства.
3. Заповеди отладки программного средства.
4. Автономная отладка программного средства.
5. Комплексная отладка программного средства.

1. Основные понятия

Отладка ПС – это деятельность, направленная на обнаружение и исправление ошибок в ПС с использованием процессов выполнения его программ. *Тестирование* ПС – это процесс выполнения его программ на некотором наборе данных, для которого заранее известен результат применения или известны правила поведения этих программ. Указанный набор данных называется *тестовым* или просто *тестом*. Таким образом, отладку можно представить в виде многократного повторения трех процессов: тестирования, в результате которого может быть констатировано наличие в ПС ошибки, поиска места ошибки в программах и документации ПС и редактирования программ и документации с целью устранения обнаруженной ошибки. Другими словами:

Отладка = Тестирование + Поиск ошибок + Редактирование.

В зарубежной литературе отладку часто понимают только как процесс поиска и исправления ошибок (без тестирования), факт наличия которых устанавливается при тестировании. Иногда тестирование и отладку считают синонимами. В нашей стране в понятие отладки обычно включают и тестирование, поэтому мы будем следовать сложившейся традиции. Впрочем, совместное рассмотрение в данной лекции этих процессов делает указанное разночтение не столь существенным. Следует, однако, отметить, что тестирование используется и как часть процесса аттестации ПС[21].

2. Принципы и виды отладки программного средства

Успех отладки ПС в значительной степени предопределяет рациональная организация тестирования. При отладке ПС отыскиваются и устраняются, в основном, те ошибки, наличие которых в ПС устанавливается при тестировании. Как было уже отмечено, тестирование не может доказать правильность ПС, в лучшем случае оно может продемонстрировать наличие в нем ошибки. Другими словами, нельзя гарантировать, что тестированием ПС практически выполнимым набором тестов можно установить наличие каждой имеющейся в ПС

ошибки. Поэтому возникает две задачи. Первая задача: подготовить такой набор тестов и применить к ним ПС, чтобы обнаружить в нем по возможности большее число ошибок. Однако чем дольше продолжается процесс тестирования (и отладки в целом), тем большей становится стоимость ПС. Отсюда вторая задача: определить момент окончания отладки ПС (или отдельной его компоненты). Признаком возможности окончания отладки является полнота охвата пропущенными через ПС тестами (т.е. тестами, к которым применено ПС) множества различных ситуаций, возникающих при выполнении программ ПС, и относительно редкое проявление ошибок в ПС на последнем отрезке процесса тестирования. Последнее определяется в соответствии с требуемой степенью надежности ПС, указанной в спецификации его качества.

Для оптимизации набора тестов, т.е. для подготовки такого набора тестов, который позволял бы при заданном их числе (или при заданном интервале времени, отведенном на тестирование) обнаруживать большее число ошибок в ПС, необходимо, во-первых, заранее планировать этот набор и, во-вторых, использовать рациональную стратегию планирования (проектирования) тестов. Проектирование тестов можно начинать сразу же после завершения этапа внешнего описания ПС. Возможны разные подходы к выработке стратегии проектирования тестов, которые можно условно графически разместить (см. рис. 1) между следующими двумя крайними подходами. Левый крайний подход заключается в том, что тесты проектируются только на основании изучения спецификаций ПС (внешнего описания, описания архитектуры и спецификации модулей). Строение модулей при этом никак не учитывается, т.е. они рассматриваются как черные ящики. Фактически такой подход требует полного перебора всех наборов входных данных, так как в противном случае некоторые участки программ ПС могут не работать при пропуске любого теста, а это значит, что содержащиеся в них ошибки не будут проявляться. Однако тестирование ПС полным множеством наборов входных данных практически неосуществимо. Правый крайний подход заключается в том, что тесты проектируются на основании изучения текстов программ с целью протестировать все пути выполнения каждой программ ПС. Если принять во внимание наличие в программах циклов с переменным числом повторений, то различных путей выполнения программ ПС может оказаться также чрезвычайно много, так что их тестирование также будет практически неосуществимо.

Рис. 1. Спектр подходов к проектированию тестов.

Оптимальная стратегия проектирования тестов расположена внутри интервала между этими крайними подходами, но ближе к левому краю. Она включает проектирование значительной части тестов по спецификациям, но она требует также проектирования некоторых тестов и по текстам программ. При этом в первом случае эта стратегия базируется на принципах:

- на каждую используемую функцию или возможность – хотя бы один тест,
- на каждую область и на каждую границу изменения какой-либо входной величины – хотя бы один тест,
- на каждую особую (исключительную) ситуацию, указанную в спецификациях, – хотя бы один тест.

Во втором случае эта стратегия базируется на принципе: каждая команда каждой программы ПС должна проработать хотя бы на одном тесте.

Оптимальную стратегию проектирования тестов можно конкретизировать на основании следующего принципа: для каждого программного документа (включая тексты программ), входящего в состав ПС, должны проектироваться свои тесты с целью выявления в нем ошибок. Во всяком случае, этот принцип необходимо соблюдать в соответствии с определением ПС и содержанием понятия технологии программирования как технологии разработки надежных ПС. В связи с этим Майерс даже определяет разные виды тестирования в зависимости от вида программного документа, на основании которого строятся тесты. В нашей стране различаются два основных вида отладки (включая тестирование): автономную и комплексную отладку ПС. *Автономная* отладка ПС означает последовательное раздельное тестирование различных частей программ, входящих в ПС, с поиском и исправлением в них фиксируемых при тестировании ошибок. Она фактически включает отладку каждого программного модуля и отладку сопряжения модулей. *Комплексная* отладка означает тестирование ПС в целом с поиском и исправлением фиксируемых при тестировании ошибок во всех документах (включая тексты программ ПС), относящихся к ПС в целом. К таким документам относятся определение требований к ПС, спецификация качества ПС, функциональная спецификация ПС, описание архитектуры ПС и тексты программ ПС.

3. Заповеди отладки программного средства

В этом разделе даются общие рекомендации по организации отладки ПС. Но сначала следует отметить некоторый феномен, который подтверждает важность предупреждения ошибок на предыдущих этапах разработки: по мере роста числа обнаруженных и исправленных ошибок в ПС *растет* также относительная вероятность существования в нем необнаруженных ошибок. Это объясняется тем, что при росте числа ошибок, обнаруженных в ПС, уточняется и наше представление об общем числе допущенных в нем ошибок, а значит, в какой-то мере, и о числе необнаруженных еще ошибок.

Ниже приводятся рекомендации по организации отладки в форме заповедей.

Заповедь 1. Считайте тестирование ключевой задачей разработки ПС, поручайте его самым квалифицированным и одаренным программистам; нежелательно тестировать свою собственную программу.
Заповедь 2. Хорош тот тест, для которого высока вероятность обнаружить ошибку, а не тот, который демонстрирует правильную работу программы.
Заповедь 3. Готовьте тесты как для правильных, так и для неправильных данных.
Заповедь 4. Документируйте пропуск тестов через компьютер; детально изучайте результаты каждого теста; избегайте тестов, пропуск которых нельзя повторить.
Заповедь 5. Каждый модуль подключайте к программе только один раз; никогда не изменяйте программу, чтобы облегчить ее тестирование.
Заповедь 6. Пропускайте заново все тесты, связанные с проверкой работы какой-либо программы ПС или ее взаимодействия с другими программами, если в нее были внесены изменения (например, в результате устранения ошибки).

3. Автономная отладка программного средства

При автономной отладке ПС каждый модуль на самом деле тестируется в некотором программном окружении, кроме случая, когда отлаживаемая программа состоит только из одного модуля. Это окружение состоит из других модулей, часть которых является модулями отлаживаемой программы, которые уже отлажены, а часть – модулями, управляющими отладкой (*отладочными* модулями, см. ниже). Таким образом, при автономной отладке тестируется всегда некоторая программа (*тестируемая программа*), построенная специально для тестирования отлаживаемого модуля. Эта программа лишь частично совпадает с отлаживаемой программой, кроме случая, когда отлаживается последний модуль отлаживаемой программы. В процессе автономной отладки ПС производится наращивание тестируемой программы отлаженными модулями: при переходе к отладке следующего модуля в его программное окружение добавляется последний отлаженный модуль. Такой процесс наращивания программного окружения отлаженными модулями называется *интеграцией* программы. Отладочные модули, входящие в окружение отлаживаемого модуля, зависят от порядка, в каком отлаживаются модули этой программы, от того, какой модуль отлаживается и, возможно, от того, какой тест будет пропускаться [27].

При восходящем тестировании это окружение будет содержать только один отладочный модуль (кроме случая, когда отлаживается последний модуль отлаживаемой программы), который будет головным в тестируемой программе. Такой отладочный модуль называют *ведущим* (или драйвером). Ведущий отладочный модуль подготавливает информационную среду для тестирования отлаживаемого модуля (т. е. формирует ее состояние, требуемое для тестирования этого модуля, в частности, путем ввода некоторых тестовых данных), осуществляет обращение к отлаживаемому модулю и после окончания его работы выдает необходимые сообщения. При отладке одного модуля для разных тестов могут составляться разные ведущие отладочные модули.

При нисходящем тестировании окружение отлаживаемого модуля в качестве отладочных модулей содержит *отладочные имитаторы* (заглушки) некоторых еще не отлаженных модулей. К таким модулям относятся, прежде всего, все модули, к которым может обращаться отлаживаемый модуль, а также еще не отлаженные модули, к которым могут обращаться уже отлаженные модули (включенные в это окружение). Некоторые из этих имитаторов при отладке одного модуля могут изменяться для разных тестов.

На практике в окружении отлаживаемого модуля могут содержаться отладочные модули обоих типов, если используется смешанная стратегия тестирования. Это связано с тем, что как восходящее, так и нисходящее тестирование имеет свои достоинства и свои недостатки.

К *достоинствам восходящего тестирования* относятся:

- простота подготовки тестов,
- возможность полной реализации плана тестирования модуля.

Это связано с тем, что тестовое состояние информационной среды готовится непосредственно перед обращением к отлаживаемому модулю (ведущим отладочным модулем).

Недостатками восходящего тестирования являются следующие его особенности:

- тестовые данные готовятся, как правило, не в той форме, которая рассчитана на пользователя (кроме случая, когда отлаживается последний, головной, модуль отлаживаемой программ);
- большой объем отладочного программирования (при отладке одного модуля приходится составлять много ведущих отладочных модулей, формирующих подходящее состояние информационной среды для разных тестов);
- необходимость специального тестирования сопряжения модулей.

К *достоинствам нисходящего тестирования* относятся следующие его особенности:

- большинство тестов готовится в форме, рассчитанной на пользователя;
- во многих случаях относительно небольшой объем отладочного программирования (имитаторы модулей, как правило, весьма просты и каждый пригоден для большого числа, нередко – для всех, тестов);
- отпадает необходимость тестирования сопряжения модулей.

Недостатком нисходящего тестирования является то, что тестовое состояние информационной среды перед обращением к отлаживаемому модулю готовится косвенно – оно является результатом применения уже отлаженных модулей к тестовым данным или данным, выдаваемым имитаторами. Это, во-первых, затрудняет подготовку тестов и требует высокой квалификации тестовика (разработчика тестов), а во-вторых, делает затруднительным или даже невозможным реализацию полного плана тестирования отлаживаемого модуля. Указанный недостаток иногда вынуждает разработчиков применять восходящее тестирование даже в случае нисходящей разработки. Однако чаще применяют некоторые модификации нисходящего тестирования, либо некоторую

комбинацию нисходящего и восходящего тестирования. Исходя из того, что нисходящее тестирование, в принципе, является предпочтительным, остановимся на приемах, позволяющих в какой-то мере преодолеть указанные трудности.

Прежде всего, необходимо организовать отладку программы таким образом, чтобы как можно раньше были отлажены модули, осуществляющие ввод данных, – тогда тестовые данные можно готовить в форме, рассчитанной на пользователя, что существенно упростит подготовку последующих тестов. Далеко не всегда этот ввод осуществляется в головном модуле, поэтому приходится в первую очередь отлаживать цепочки модулей, ведущие к модулям, осуществляющим указанный ввод. Пока модули, осуществляющие ввод данных, не отлажены, тестовые данные поставляются некоторыми имитаторами: они либо включаются в имитатор как его часть, либо вводятся этим имитатором.

При нисходящем тестировании некоторые состояния информационной среды, при которых требуется тестировать отлаживаемый модуль, могут не возникать при выполнении отлаживаемой программы ни при каких входных данных. В этих случаях можно было бы вообще не тестировать отлаживаемый модуль, так как обнаруживаемые при этом ошибки не будут проявляться при выполнении отлаживаемой программы ни при каких входных данных. Однако так поступать не рекомендуется, так как при изменениях отлаживаемой программы (например, при сопровождении ПС) не использованные для тестирования отлаживаемого модуля состояния информационной среды могут уже возникать, что требует дополнительного тестирования этого модуля (а этого при рациональной организации отладки можно было бы не делать, если сам данный модуль не изменялся). Для осуществления тестирования отлаживаемого модуля в указанных ситуациях иногда используют подходящие имитаторы, чтобы создать требуемое состояние информационной среды. Чаще же пользуются модифицированным вариантом нисходящего тестирования, при котором отлаживаемые модули перед их интеграцией предварительно тестируются отдельно (в этом случае в окружении отлаживаемого модуля появляется ведущий отладочный модуль, наряду с имитаторами модулей, к которым может обращаться отлаживаемый модуль). Однако, представляется более целесообразной другая модификация нисходящего тестирования: после завершения нисходящего тестирования отлаживаемого модуля для достижимых тестовых состояний информационной среды следует его отдельно протестировать для остальных требуемых состояний информационной среды.

Часто применяют также комбинацию восходящего и нисходящего тестирования, которую называют методом *сандвича*. Сущность этого метода заключается в одновременном осуществлении как восходящего, так и нисходящего тестирования, пока эти два процесса тестирования не встретятся на каком-либо модуле где-то в середине структуры отлаживаемой программы. Этот метод при разумном порядке тестирования позволяет воспользоваться достоинствами как восходящего, так и нисходящего тестирования, а также в значительной степени нейтрализовать их недостатки.

Весьма важным при автономной отладке является тестирование сопряжения модулей. Дело в том, что спецификация каждого модуля программы, кроме головного, используется в этой программы в двух ситуациях: во-первых, при разработке текста (иногда говорят: тела) этого модуля и, во-вторых, при написании обращения к этому модулю в других модулях программы. И в том, и в другом случае в результате ошибки может быть нарушено требуемое соответствие заданной спецификации модуля. Такие ошибки требуется обнаруживать и устранять. Для этого и предназначено тестирование сопряжения модулей. При нисходящем тестировании тестирование сопряжения осуществляется попутно каждым пропускаемым тестом, что считают достоинством нисходящего тестирования. При восходящем тестировании обращение к отлаживаемому модулю производится не из модулей отлаживаемой программы, а из ведущего отладочного модуля. В связи с этим существует опасность, что последний модуль может приспособиться к некоторым "заблуждениям" отлаживаемого модуля. Поэтому, приступая (в процессе интеграции программы) к отладке нового модуля, приходится тестировать каждое обращение к ранее отлаженному модулю с целью обнаружения несогласованности этого обращения с телом соответствующего модуля (и не исключено, что виноват в этом ранее отлаженный модуль). Таким образом, приходится частично повторять в новых условиях тестирование ранее отлаженного модуля, при этом возникают те же трудности, что и при нисходящем тестировании [28].

Автономное тестирование модуля целесообразно осуществлять в четыре последовательно выполняемых шага.

Шаг 1. На основании спецификации отлаживаемого модуля подготовьте тесты для каждой возможности и каждой ситуации, для каждой границы областей допустимых значений всех входных данных, для каждой области изменения данных, для каждой области недопустимых значений всех входных данных и каждого недопустимого условия.

Шаг 2. Проверьте текст модуля, чтобы убедиться, что каждое направление любого разветвления будет пройдено хотя бы на одном тесте. Добавьте недостающие тесты.

Шаг 3. Проверьте текст модуля, чтобы убедиться, что для каждого цикла существуют тесты, обеспечивающие, по крайней мере, три следующие ситуации: тело цикла не выполняется ни разу, тело цикла выполняется один раз и тело цикла выполняется максимальное число раз. Добавьте недостающие тесты.

Шаг 4. Проверьте текст модуля, чтобы убедиться, что существуют тесты, проверяющие чувствительность к отдельным особым значениям входных данных. Добавьте недостающие тесты.

4. Комплексная отладка программного средства

Как уже было сказано выше, при комплексной отладке тестируется ПС в целом, причем тесты готовятся по каждому из документов ПС. Тестирование этих документов производится, как правило, в порядке, обратном их разработ-

ке. Исключение составляет лишь тестирование документации по применению, которая разрабатывается по внешнему описанию параллельно с разработкой текстов программ – это тестирование лучше производить после завершения тестирования внешнего описания. Тестирование при комплексной отладке представляет собой применение ПС к конкретным данным, которые в принципе могут возникнуть у пользователя (в частности, все тесты готовятся в форме, рассчитанной на пользователя), но, возможно, в моделируемой (а не в реальной) среде. Например, некоторые недоступные при комплексной отладке устройства ввода и вывода могут быть заменены их программными имитаторами.

Тестирование архитектуры ПС. Целью тестирования является поиск несоответствия между описанием архитектуры и совокупностью программ ПС. К моменту начала тестирования архитектуры ПС должна быть уже закончена автономная отладка каждой подсистемы. Ошибки реализации архитектуры могут быть связаны, прежде всего, с взаимодействием этих подсистем, в частности, с реализацией архитектурных функций (если они есть). Поэтому хотелось бы проверить все пути взаимодействия между подсистемами ПС. При этом желательно хотя бы протестировать все цепочки выполнения подсистем без повторного вхождения последних. Если заданная архитектура представляет ПС в качестве малой системы из выделенных подсистем, то число таких цепочек будет вполне обозримо.

Тестирование внешних функций. Целью тестирования является поиск расхождений между функциональной спецификацией и совокупностью программ ПС. Несмотря на то, что все эти программы автономно уже отлажены, указанные расхождения могут быть, например, из-за несоответствия внутренних спецификаций программ и их модулей (на основании которых производилось автономное тестирование) функциональной спецификации ПС. Как правило, тестирование внешних функций производится так же, как и тестирование модулей на первом шаге, т.е. как черного ящика.

Тестирование качества ПС. Целью тестирования является поиск нарушений требований качества, сформулированных в спецификации качества ПС. Это наиболее трудный и наименее изученный вид тестирования. Ясно лишь, что далеко не каждый примитив качества ПС может быть испытан тестированием. Завершенность ПС проверяется уже при тестировании внешних функций. На данном этапе тестирование этого примитива качества может быть продолжено, если требуется получить какую-либо вероятностную оценку степени надежности ПС. Однако, методика такого тестирования еще требует своей разработки. Могут тестироваться такие примитивы качества, как точность, устойчивость, защищенность, временная эффективность, в какой-то мере – эффективность по памяти, эффективность по устройствам, расширяемость и, частично, независимость от устройств. Каждый из этих видов тестирования имеет свою специфику и заслуживает отдельного рассмотрения. Мы здесь ограничимся лишь их перечислением. Легкость применения ПС (критерий качества, включающий несколько примитивов качества) оценивается при тестировании документации по применению ПС.

Тестирование документации по применению ПС. Целью тестирования является поиск несогласованности документации по применению и совокупностью программ ПС, а также выявление неудобств, возникающих при применении ПС. Этот этап непосредственно предшествует подключению пользователя к завершению разработки ПС (тестированию определения требований к ПС и аттестации ПС), поэтому весьма важно разработчикам сначала самим воспользоваться ПС так, как это будет делать пользователь. Все тесты на этом этапе готовятся исключительно на основании только документации по применению ПС. Прежде всего, должны тестироваться возможности ПС как это делалось при тестировании внешних функций, но только на основании документации по применению. Должны быть протестированы все неясные места в документации, а также все примеры, использованные в документации. Далее тестируются наиболее трудные случаи применения ПС с целью обнаружить нарушение требований относительности легкости применения ПС.

Тестирование определения требований к ПС. Целью тестирования является выяснение, в какой мере ПС не соответствует предъявленному определению требований к нему. Особенность этого вида тестирования заключается в том, что его осуществляет организация-покупатель или организация-пользователь ПС как один из путей преодоления барьера между разработчиком и пользователем. Обычно это тестирование производится с помощью контрольных задач – типовых задач, для которых известен результат решения. В тех случаях, когда разрабатываемое ПС должно придти на смену другой версии ПС, которая решает хотя бы часть задач разрабатываемого ПС, тестирование производится путем решения общих задач с помощью как старого, так и нового ПС (с последующим сопоставлением полученных результатов). Иногда в качестве формы такого тестирования используют *опытную* эксплуатацию ПС – ограниченное применение нового ПС с анализом использования результатов в практической деятельности. По существу, этот вид тестирования во многом перекликается с испытанием ПС при его аттестации, но выполняется до аттестации, а иногда и вместо аттестации.

Краткие выводы

В данной теме рассмотрены основы отладки программного средства, тестирование программного средства, принципы и виды отладки программного средства, спектр подходов к проектированию тестов, заповеди отладки программного средства, автономная отладка программного средства, комплексная отладка и тестирование программного средства. А также подробно описаны тестирование архитектуры программного средства, тестирование внешних функций, тестирование качества программного средства, тестирование документации по применению программного средства, тестирование определения требований к программному средству.

Ключевые слова и определения

Отладка ПС – это деятельность, направленная на обнаружение и исправление ошибок в ПС с использованием процессов выполнения его программ.

Автономная отладка ПС означает последовательное раздельное тестирование различных частей программ, входящих в ПС, с поиском и исправлением в них фиксируемых при тестировании ошибок.

Тестирование ПС – это процесс выполнения его программ на некотором наборе данных, для которого заранее известен результат применения или известны правила поведения этих программ. Указанный набор данных называется *тестовым* или просто *тестом*.

Комплексная отладка означает тестирование ПС в целом с поиском и исправлением фиксируемых при тестировании ошибок во всех документах (включая тексты программ ПС), относящихся к ПС в целом.

Тестирование архитектуры *ПС*. Целью тестирования является поиск несоответствия между описанием архитектуры и совокупностью программ ПС.

Тестирование внешних функций. Целью тестирования является поиск расхождений между функциональной спецификацией и совокупностью программ ПС.

Тестирование качества ПС. Целью тестирования является поиск нарушений требований качества, сформулированных в спецификации качества ПС.

Тестирование документации по применению ПС. Целью тестирования является поиск несогласованности документации по применению и совокупностью программ ПС, а также выявление неудобств, возникающих при применении ПС.

Тестирование определения требований к ПС. Целью тестирования является выяснение, в какой мере ПС не соответствует предъявленному определению требований к нему.

Вопросы и задания для обсуждения и контроля:

1. Что из себя представляет процесс тестирования программы?
2. В чем разница между тестированием и отладкой программы?
3. Назовите преимущества и недостатки восходящего тестирования.
4. Назовите преимущества и недостатки нисходящего тестирования.
5. Назовите заповеди отладки программ?
6. В чем отличие между автономной отладкой и комплексной?
7. Назовите шаги автономного тестирования модуля.

Рекомендуемая литература

1. Гагарина Л.Г., Виснадул Б.Д., Игошин А.В. Основы технологии разработки программных продуктов: Учебное пособие. - М.: ФОРУМ: ИНФРА-М, 2006.

2. С.С.Ғуломов, Б.А.Бегалов, Н.Р.Зайналов, Р.А.Дадабаева, А.Э.Давронов. Дастурлаш технологиялари. Олий ўқув юртлари учун ўқув қўлланма. – Т.: ТДИУ, 2006.

3. Гвоздева В.А. Введение в специальность программиста: Учебник. – М.: ФОРУМ: ИНФРА – М, 2005.

4. Козырев А.А. Информационные технологии в экономике и управлении: Учебник. Второе издание. – СПб.:Изд-во Михайлова В.А., 2001. 360 с.

5. Благодатских В.А., Енгибарян М.А., Ковалевская Е.В. Экономика, разработка и использование программного обеспечения ЭВМ. – М.: Финансы и статистика, 1995.

Глава 10. ДОКУМЕНТИРОВАНИЕ ПРОЦЕССА СОЗДАНИЯ ПРОГРАММНЫХ ПРОДУКТОВ

1. Составление технического задания на программирование.
2. Технический проект.
3. Рабочая документация (рабочий проект). Ввод в действие.

При традиционной неавтоматизированной разработке программ независимо от принятого метода проектирования и используемого инструментария выполняют следующие работы.

1. Составление технического задания на программирование

Данная работа соответствует этапу анализа и спецификации программ жизненного цикла программных продуктов.

При составлении технического задания требуется:

- определить платформу разрабатываемой программы — тип операционной системы (например, для IBM PC-совместимых машин делается выбор операционной среды:
MS DOS, Windows, Windows NT либо Unix, 0S/2);
- оценить необходимость сетевого варианта работы программы (определяется программное обеспечение (ПО) вычислительной сети — Windows NT, допустимая номенклатура программного обеспечения сетевой обработки);
- определить необходимость разработки программы, которую можно переносить на различные платформы;
- обосновать целесообразность работы с базами данных под управлением СУБД.

На этом же этапе выбирают методы решения задачи; разрабатывают обобщенный алгоритм решения комплекса задач, функциональную структуру алгоритма или состав объектов, определяют требования к комплексу технических средств системы обработки информации, интерфейсу конечного пользователя.

2. Технический проект

На данном этапе выполняется комплекс наиболее важных работ, а именно:

- с учетом принятого подхода к проектированию программного продукта разрабатывается детальный алгоритм обработки данных или уточняется состав объектов и их свойств, методов обработки, событий, запускающих методы обработки;
- определяется состав общесистемного программного обеспечения, включающий базовые средства (операционную систему, модель СУБД, электронные таблицы, методо-ориентированные и функциональные ППП промышленного назначения и т.п.)

• разрабатывается внутренняя структура программного продукта, образованная отдельными программными модулями;

• осуществляется выбор инструментальных средств разработки программных модулей

Работы данного этапа в существенной степени зависят от принятых решений по технической части системы обработки данных и операционной среде, от выбранных инструментальных средств проектирования алгоритмов и программ, технологии работ.

Пример 18.2. Для создания MS DOS-приложений может быть использован язык программирования Visial Basicfor DOS standart, Fortan 5.1, Visual C++ *for Windows* Если необходима переносимость программ на другие ЭВМ или другие операционные платформы, выбирается среда Windows NT.

При разработке программ, работающих в среде Windows, возможно применение технологии OLE 2.0 для создания приложений, включающих *объекты* других приложений. Определяется способ использования объектов: *внедрение (embedding) или связывание (linking)* .

Приложение может работать с базами данных различных СУБД для этого служит стандартная технология интерфейса Open Database Connektivity (ODBC). Работа в режиме телекоммуникаций обеспечивается стандартной технологией Messaging Application Program Interfase (MAP1).

3. Рабочая документация (рабочий проект). Ввод в действие.

На данном этапе осуществляется *адаптация базовых средств* программного (операционной системы, СУБД, методо-ориентированных ППП, инструментальных сред конечного пользователя — текстовых редакторов, электронных таблиц и т.п.). Выполняется разработка программных модулей или методов обработки объектов — собственно *программирование* или создание программного кода. Проводятся автономная и комплексная отладка программного продукта, *испытание* работоспособности программных модулей и базовых программных средств. Для комплексной отладки готовится *контрольные* который позволяет проверить соответствие возможностей программного продукта заданным спецификациям.

Основной результат работ этого этапа — также создание эксплуатационной *документации* на программный продукт:

• *описание применения* — дает общую характеристику программного изделие с указанием сферы его применения, требований к базовому программному обеспечению комплексу технических средств;

• *руководство пользователя* — включает детальное описание функциональных возможностей и технологии работы с программным продуктом. Данный вид документации ориентирован на *конечного* пользователя и содержит необходимую информацию самостоятельного освоения и нормальной работы пользователя (с учетом квалификации пользователя);

- *руководство программиста (оператора)*—указывает особенности установки (инсталляции) программного продукта и его внутренней структуры – состав и назначения модулей, правила эксплуатации и обеспечения надежной и качественной работы программного продукта.

В ряде случаев на данном этапе для программных продуктов массового применения создаются *обучающие системы, демоверсии, гипертекстовые системы помощи.*

Готовый программный продукт сначала проходит *опытную эксплуатацию* (пробный рынок продаж), а затем сдается в *промышленную эксплуатацию* (тиражирование и распространение программного продукта).

Краткие выводы

Составление технического задания на программирование. Требования, определяемые при составлении технического задания. Технический проект. Работы, выполняемые на этапе технический проект. Рабочая документация. Эксплуатационная документация (описание применения, руководство пользователя, руководство программиста, обучающие системы, гипертекстовые системы помощи). Внедрение.

Ключевые понятия и определения

Составление технического задания на программирование- на этом этапе выбирают методы решения задачи; разрабатывают обобщенный алгоритм решения комплекса задач, функциональную структуру алгоритма или состав объектов, определяют требования к комплексу технических средств системы обработки информации, интерфейсу конечного пользователя.

Технический проект – на этом этапе разрабатывается детальный алгоритм обработки данных, определяется состав общесистемного программного обеспечения, разрабатывается внутренняя структура программного продукта, осуществляется выбор инструментальных средств разработки программных модулей

Рабочая документация (рабочий проект)- на данном этапе осуществляется *адаптация базовых средств* программного (операционной системы, СУБД, методо-ориентированных ППП, инструментальных сред конечного пользователя — текстовых редакторов, электронных таблиц и т.п.).

Описание применения — дает общую характеристику программного изделие с указанием сферы его применения, требований к базовому программному обеспечению комплексу технических средств;

Руководство пользователя — включает детальное описание функциональных возможностей и технологии работы с программным продуктом. Данный вид документации ориентирован на *конечного* пользователя и содержит необходимую информацию самостоятельного освоения и нормальной работы пользователя (с учетом квалификации пользователя);

Руководство программиста (оператора)—указывает особенности установки (инсталляции) программного продукта и его внутренней структуры – состав и назначения модулей, правила эксплуатации и обеспечения надежной и качественной работы программного продукта.

Вопросы для обсуждения и контроля:

1. Назовите этапы создания программных продуктов.
2. Какие действия выполняются на этапе составление технического задания на программирование?
3. Что разрабатывается на этапе технический проект?
4. Какая документация составляется на этапе Рабочая документация?
5. Какие действия осуществляются на этапе рабочая документация?

Рекомендуемая литература

1.Чернев Д.А. Технология разработки программного обеспечения: (Учебное пособие).-Т.:, “Mehnat”, 2004.47-51 стр.

2.Информатика: Учебник. / Под ред. Н.В.Макаровой. 3-е перераб. изд. - М.: ФиС, 2004. –651-653 стр.

3.Информатика: Базовый курс / С.В.Симонович и др.- Санкт-Петербург: Питер, 2003. 601-605 стр.

4.Благодатских В.А., Енгибарян М.А., Ковалевская Е.В. Экономика, разработка и использование программного обеспечения ЭВМ. - М.: Финансы и статистика, 1995. 134-136 стр.

Глава 11. ОБЕСПЕЧЕНИЕ ФУНКЦИОНАЛЬНОСТИ И НАДЕЖНОСТИ ПРОГРАММНОГО СРЕДСТВА

1. Обеспечение надежности – основной мотив разработки программных средств.
2. Обеспечение завершенности программного средства.
3. Обеспечение точности программного средства.
4. Обеспечение автономности программного средства.

1. Обеспечение надежности – основной мотив разработки программных средств

Рассмотрим теперь общие принципы обеспечения надежности ПС, что, как мы уже подчеркивали, является основным мотивом разработки ПС, задающим специфическую окраску всем технологическим процессам разработки ПС. В технике известны четыре подхода обеспечению надежности:

- предупреждение ошибок;
- самообнаружение ошибок;
- самоисправление ошибок;
- обеспечение устойчивости к ошибкам.

Целью подхода предупреждения ошибок – не допустить ошибок в готовых продуктах, в нашем случае – в ПС. Проведенное рассмотрение природы ошибок при разработке ПС позволяет для достижения этой цели сконцентрировать внимание на следующих вопросах:

- борьба со сложностью,
- обеспечение точности перевода,
- преодоление барьера между пользователем и разработчиком,
- обеспечение контроля принимаемых решений.

Этот подход связан с организацией процессов разработки ПС, т.е. с технологией программирования. И хотя, как мы уже отмечали, гарантировать отсутствие ошибок в ПС невозможно, но в рамках этого подхода можно достигнуть приемлемого уровня надежности ПС.

Остальные три подхода связаны с организацией самих продуктов технологии, в нашем случае – программ. Они учитывают возможность ошибки в программах. Самообнаружение ошибки в программе означает, что программа содержит средства обнаружения отказа в процессе ее выполнения. Самоисправление ошибки в программе означает не только обнаружение отказа в процессе ее выполнения, но и исправление последствий этого отказа, для чего в программе должны иметься соответствующие средства. Обеспечение устойчивости программы к ошибкам означает, что в программе содержатся средства, позволяющие локализовать область влияния отказа программы, либо уменьшить его неприятные последствия, а иногда предотвратить катастрофические последствия отказа. Однако, эти подходы используются весьма редко (может быть, относительно чаще используется обеспечение устойчивости к ошибкам). Свя-

зано это, во-первых, с тем, что многие простые методы, используемые в технике в рамках этих подходов, неприменимы в программировании, например, дублирование отдельных блоков и устройств (выполнение двух копий одной и той же программы всегда будет приводить к одинаковому эффекту – правильному или неправильному). А, во-вторых, добавление в программу дополнительных фрагментов приводит к ее усложнению (иногда – значительному), что в какой-то мере мешает методам предупреждения ошибок.

4. Методы борьбы со сложностью.

Известны два общих метода борьбы со сложностью систем:

- обеспечения независимости компонент системы;
- использование в системах иерархических структур.

Обеспечение независимости компонент означает разбиение системы на такие части, между которыми должны остаться по возможности меньше связей. Одним из воплощений этого метода является модульное программирование. Использование в системах иерархических структур позволяет локализовать связи между компонентами, допуская их лишь между компонентами, принадлежащими смежным уровням иерархии. Этот метод, по существу, означает разбиение большой системы на подсистемы, образующих малую систему. Здесь существенно используется способность человека к абстрагированию.

5. Обеспечение точности перевода.

Обеспечение точности перевода направлено на достижение однозначности интерпретации документов различными разработчиками, а также пользователями ПС. Это требует придерживаться при переводе определенной дисциплины. Майерс предлагает использовать общую дисциплину решения задач, рассматривая перевод как решение задачи. Лучшим руководством по решению задач он считает книгу Пойа "Как решать задачу". В соответствии с этим весь процесс перевода можно разбить на следующие этапы:

- Поймите задачу;
- Составьте план (включая цели и методы решения);
- Выполните план (проверяя правильность каждого шага);
- Проанализируйте полученное решение.

Подробно обсуждать этот вопрос мы здесь не будем.

6. Преодоление барьера между пользователем и разработчиком.

Как обеспечить, чтобы ПС выполняла то, что пользователю разумно ожидать от нее? Для этого разработчикам необходимо правильно понять, во-первых, чего хочет пользователь, и, во-вторых, его уровень подготовки и окружающую его обстановку. Ясное описание соответствующей сферы деятельности пользователя или интересующей его проблемной области во многом облегчает достижение разработчиками этой цели. При разработке ПС следует привлекать пользователя для участия в процессах принятия решений, а также тщательно освоить особенности его работы (лучше всего – побывать в его "шкуре").

7. Контроль принимаемых решений.

Обязательным шагом в каждом процессе (этапе) разработки ПС должна быть проверка правильности принятых решений. Это позволит обнаруживать и исправлять ошибки на самой ранней стадии после ее возникновения, что, во-первых, существенно снижает стоимость ее исправления и, во-вторых, повышает вероятность правильного ее устранения.

С учетом специфики разработки ПС необходимо применять везде, где это возможно,

- смежный контроль,
- сочетание как статических, так и динамических методов контроля.

Смежный контроль означает, проверку полученного документа лицами, не участвующими в его разработке, с двух сторон: во-первых, со стороны автора исходного для контролируемого процесса документа, и, во-вторых, лицами, которые будут использовать полученный документ в качестве исходного в последующих технологических процессах. Такой контроль позволяет обеспечивать однозначность интерпретации полученного документа.

Сочетание статических и динамических методов контроля означает, что нужно не только контролировать документ как таковой, но и проверять, какой процесс обработки данных он описывает. Это отражает одну из специфических особенность ПС (статическая форма, динамическое содержание).

2. Обеспечение завершенности программного средства

Завершенность ПС является общим примитивом качества ПС для выражения и функциональности и надежности ПС, причем для функциональности она является единственным примитивом.

Функциональность ПС определяется его функциональной спецификацией. Завершенность ПС как примитив его качества является мерой того, в какой степени эта спецификация реализована в разрабатываемом ПС. Обеспечение этого примитива в полном объеме означает реализацию каждой из функций, определенной в функциональной спецификации, со всеми указанными там деталями и особенностями. Все рассмотренные ранее технологические процессы показывают, как это может быть сделано.

Однако в спецификации качества ПС могут быть определены несколько уровней реализации функциональности ПС: может быть определена некоторая упрощенная (начальная или стартовая) версия, которая должна быть реализована в первую очередь; могут быть также определены и несколько промежуточных версий. В этом случае возникает дополнительная технологическая задача: организация наращивания функциональности ПС. Здесь важно отметить, что разработка упрощенной версии ПС не есть разработка его *прототипа*. Прототип разрабатывается для того, чтобы лучше понять условия применения будущего ПС, уточнить его внешнее описание. Он рассчитан на избранных пользователей и поэтому может сильно отличаться от требуемого ПС не только выполняемыми функциями, но и особенностями пользовательского интерфейса. Упрощенная же версия разрабатываемого ПС должна быть рассчитана на *практически полезное* применение любыми пользователями, для которых

предназначено это ПС. Поэтому главный принцип обеспечения функциональности такого ПС заключается в том, чтобы с самого начала разрабатывать ПС таким образом, как будто требуется ПС в полном объеме, до тех пор, когда разработчики будут иметь дело непосредственно с теми частями или деталями ПС, реализацию которых можно отложить в соответствии со спецификацией его качества. Тем самым, и внешнее описание и описание архитектуры ПС должно быть разработано в полном объеме. Можно отложить лишь реализацию тех программных подсистем (определенных в архитектуре разрабатываемого ПС), функционирования которых не требуется в начальной версии этого ПС. Реализацию же самих программных подсистем лучше всего производить методом целенаправленной конструктивной реализации, оставляя в начальной версии ПС подходящие имитаторы тех программных модулей, функционирование которых в этой версии не требуется. Допустима также упрощенная реализация некоторых программных модулей, опускающая реализацию некоторых деталей соответствующих функций. Однако такие модули с технологической точки зрения лучше рассматривать как своеобразные их имитаторы (хотя и далеко продвинутые).

Достигнутый при обеспечении функциональности ПС уровень его завершенности на самом деле может быть не таким, как ожидалось, из-за ошибок, оставшихся в этом ПС. Можно лишь говорить, что требуемая завершенность достигнута с некоторой вероятностью, определяемой объемом и качеством проведенного тестирования. Для того чтобы повысить эту вероятность, необходимо продолжить тестирование и отладку ПС. Однако, оценивание такой вероятности является весьма специфической задачей, которая пока еще ждет соответствующих теоретических исследований.

3. Обеспечение точности, автономности и устойчивости программного средства

Обеспечение этого примитива качества связано с действиями над значениями вещественных типов (точнее говоря, со значениями, представляемыми с некоторой погрешностью). Обеспечить требуемую точность при вычислении значения той или иной функции – значит получить это значение с погрешностью, не выходящей за рамки заданных границ. Видами погрешности, методами их оценки и методами достижения требуемой точности (т.н. *приближенными вычислениями*) занимается вычислительная математика. Здесь мы лишь обратим внимание на некоторую структуру погрешности: погрешность вычисленного значения (*полная погрешность*) зависит

- от *погрешности используемого метода* вычисления (в которую мы включаем и неточность используемой модели),
- от погрешности представления используемых данных (от т.н. *неустранимой погрешности*),
- от *погрешности округления* (неточности выполнения используемых в методе операций).

Вопрос об автономности программного средства решается путем принятия решения о возможности использования в разрабатываемом ПС какого-либо подходящего базового программного обеспечения. Надежность имеющегося в распоряжении разработчиков базового программного обеспечения для целевого компьютера может не отвечать требованиям к надежности разрабатываемого ПС. Поэтому от использования такого программного обеспечения приходиться отказываться, а его функции в требуемом объеме приходится реализовывать в рамках разрабатываемого ПС. Аналогичное решение приходится принимать при жестких ограничениях на используемые ресурсы (по критерию эффективности ПС). Такое решение может быть принято уже в процессе разработки спецификации качества ПС, иногда – на этапе конструирования ПС.

Обеспечение устойчивости программного средства обеспечивается с помощью так называемого *защитного программирования*. Вообще говоря, защитное программирование применяется при программировании модуля для повышения надежности ПС в более широком смысле. Как утверждает Майерс, «защитное программирование основано на важной предпосылке: худшее, что может сделать модуль, – это принять неправильные входные данные и затем вернуть неверный, но правдоподобный результат». Для того, чтобы этого избежать, в текст модуля включают проверки его входных и выходных данных на их корректность в соответствии со спецификацией этого модуля, в частности, должны быть проверены выполнение ограничений на входные и выходные данные и соотношений между ними, указанные в спецификации модуля. В случае отрицательного результата проверки возбуждается соответствующая исключительная ситуация. Для обработки таких ситуаций в конец этого модуля включаются фрагменты второго рода – обработчики соответствующих исключительных ситуаций. Эти обработчики помимо выдачи необходимой диагностической информации, могут принять меры либо по исключению ошибки в данных (например, потребовать их повторного ввода), либо по ослаблению влияния ошибки (например, во избежание поломки устройств, управляемых с помощью данного ПС, при аварийном прекращении выполнения программы осуществляют мягкую их остановку).

Применение защитного программирования модулей приводит к снижению эффективности ПС как по времени, так и по памяти. Поэтому необходимо разумно регулировать степень применения защитного программирования в зависимости от требований к надежности и эффективности ПС, сформулированных в спецификации качества разрабатываемого ПС. Входные данные разрабатываемого модуля могут поступать как непосредственно от пользователя, так и от других модулей. Наиболее употребительным случаем применения защитного программирования является применение его для первой группы данных, что и означает реализацию устойчивости ПС. Это нужно делать всегда, когда в спецификации качества ПС имеется требование об обеспечении устойчивости ПС. Применение защитного программирования для второй группы входных данных означает попытку обнаружить ошибку в других модулях во время выполнения разрабатываемого модуля, а для выходных данных разрабатываемого модуля –

попытку обнаружить ошибку в самом этом модуле во время его выполнения. По существу, это означает частичное воплощение подхода самообнаружения ошибок для обеспечения надежности ПС. Этот случай защитного программирования применяется крайне редко – только в том случае, когда требования к надежности ПС чрезвычайно высоки.

4. Обеспечение защищенности программных средств

Различают следующие виды защиты ПС от искажения информации:

- защита от сбоев аппаратуры;
- защита от влияния «чужой» программы;
- защита от отказов «своей» программы;
- защита от ошибок оператора (пользователя);
- защита от несанкционированного доступа;
- защита от защиты.

4.1 Защита от сбоев аппаратуры. В настоящее время этот вид защиты является не очень злободневной задачей (с учетом уровня достигнутой надежности компьютеров). Но все же полезно знать ее решение. Это обеспечивается организацией т.н. «двойных или тройных просчетов». Для этого весь процесс обработки данных, определяемый ПС, разбивается по времени на интервалы так называемыми «опорными точками». Длина этого интервала не должна превосходить половины среднего времени безотказной работы компьютера. В начале каждого такого интервала во вторичную память записывается с некоторой контрольной суммой копия состояния изменяемой в этом процессе памяти («опорная точка»). Для того, чтобы убедиться, что обработка данных от одной опорной точки до следующей (т.е. один «просчет») произведена правильно (без сбоев компьютера), производится два таких «просчета». После первого «просчета» вычисляется и запоминается указанная контрольная сумма, а затем восстанавливается состояние памяти по опорной точке и делается второй «просчет». После второго «просчета» вычисляется снова указанная контрольная сумма, которая затем сравнивается с контрольной суммой первого «просчета». Если эти две контрольные суммы совпадают, второй просчет считается правильным, в противном случае контрольная сумма второго «просчета» также запоминается и производится третий «просчет» (с предварительным восстановлением состояния памяти по опорной точке). Если контрольная сумма третьего «просчета» совпадет с контрольной суммой одного из первых двух «просчетов», то третий просчет считается правильным, в противном случае требуется инженерная проверка компьютера.

4.2. Защита от влияния «чужой» программы. При появлении мультипрограммного режима работы компьютера в его памяти может одновременно находиться в стадии выполнения несколько программ, попеременно получающих управление в результате возникающих прерываний (т.н. квазипараллельное выполнение программ). Одна из таких программ (обычно: операционная система) занимается обработкой прерываний и управлением мультипрограмм-

ным режимом. Здесь под «чужой» программой понимается программа (или какой-либо программный фрагмент), выполняемая параллельно (или квазипараллельно) по отношению к защищаемой программе (или ее фрагменту). Этот вид защиты должна обеспечить, чтобы эффект выполнения защищаемой программы не зависел от того, какие программы выполняются параллельно с ней, и относится, прежде всего, к функциям операционных систем.

Различают две разновидности этой защиты:

- защита от отказов «чужой» программы,
- защита от злонамеренного влияния «чужой» программы.

Защита от отказов «чужой» программы означает, что на выполнение функций защищаемой программой не будут влиять отказы (проявления ошибок), возникающие в параллельно выполняемых программах. Для того чтобы управляющая программа (операционная система) могла обеспечить защиту себя и других программ от такого влияния, аппаратура компьютера должна реализовывать следующие возможности:

* защиту памяти,
* два режима функционирования компьютера: привилегированный и рабочий (пользовательский),
* два вида операций: привилегированные и ординарные,
* корректную реализацию прерываний и начального включения компьютера,
* временно́е прерывание.

Защита памяти означает возможность программным путем задавать для каждой программы недоступные для нее участки памяти. В привилегированном режиме могут выполняться любые операции (как ординарные, так и привилегированные), а в рабочем режиме – только ординарные. Попытка выполнить привилегированную операцию, а также обратиться к защищенной памяти в рабочем режиме вызывает соответствующее прерывание. К привилегированным операциям относятся операции изменения защиты памяти и режима функционирования, а также доступа к внешней информационной среде. Корректная реализация прерываний и начального включения компьютера означает обязательную установку привилегированного режима и отмену защиты памяти. В этих условиях управляющая программа (операционная система) может полностью защитить себя от влияния отказов других программ. Для этого достаточно, чтобы

- все точки передачи управления при начальном включении компьютера и при прерываниях принадлежали этой программе,
- она не позволяла никакой другой программе работать в привилегированном режиме (при передаче управления любой другой программе должен включаться только рабочий режим),
- она полностью защищала свою память (содержащую, в частности, всю ее управляющую информацию, включая так называемые вектора прерываний) от других программ.

Тогда никто не помешает ей выполнять любые реализованные в ней функции защиты других программ (в том числе и доступа к внешней информационной

среде). Для облегчения решения этой задачи часть такой программы помещается в постоянную память. Наличие временнóго прерывания позволяет управляющей программе защититься от зацикливания в других программах (без такого прерывания она могла бы просто лишиться возможности управлять).

Защита от злонамеренного влияния «чужих» программ означает, что изменение внешней информационной среды, предоставленной защищаемой программе, со стороны другой, параллельно выполняемой программы будет невозможно или сильно затруднено без ведома защищаемой программы. Для этого операционная система должна обеспечить подходящий контроль доступа к внешней информационной среде. Необходимым условием обеспечение такого контроля является обеспечения защиты от злонамеренного влияния «чужих» программ хотя бы самой операционной системы. В противном случае такой контроль можно было бы обойти путем изменения операционной системы со стороны «злонамеренной» программы.

Этот вид защиты включает, в частности, и защиту от т.н. «компьютерных вирусов», под которыми понимают фрагменты программ, способные в процессе своего выполнения внедряться (копироваться) в другие программы (или в отдельные программные фрагменты). «Компьютерные вирусы», обладая способностью к размножению (к внедрению в другие программы), при определенных условиях вызывают изменение эффекта выполнения «зараженной» программы, что может привести к серьезным деструктивным изменениям ее внешней информационной среды. Операционная система, будучи защищенной от влияния «чужих» программ, может ограничить доступ к программным фрагментам, хранящимся во внешней информационной среде. Так, например, может быть запрещено изменение таких фрагментов любыми программами, кроме некоторых, которые знает операционная система, или, другой вариант, может быть разрешено только после специальных подтверждений программы (или пользователя).

4.3. Защита от отказов «своей» программы. Обеспечивается надежностью ПС, на что ориентирована вся технология программирования, обсуждаемая в настоящем курсе лекций.

4.4. Защита от ошибок пользователя. Здесь идет речь не об ошибочных данных, поступающих от пользователя ПС, – защита от них связана с обеспечением устойчивости ПС, а о действиях пользователя, приводящих к деструктивному изменению состояния внешней информационной среды ПС, несмотря на корректность используемых при этом данных. Защита от таких действий, частично, обеспечивается выдачей предупредительных сообщений о попытках изменить состояние внешней информационной среды ПС с требованием подтверждения этих действий. Для случаев же, когда такие ошибки совершаются, может быть предусмотрена возможность восстановления состояния отдельных компонент внешней информационной среды ПС на определенные моменты времени. Такая возможность базируется на ведении (формировании) архива состояний (или изменений состояния) внешней информационной среды.

4.5. Защита от несанкционированного доступа. Каждому пользователю ПС предоставляет определенные информационные и процедурные ресурсы

(услуги), причем у разных пользователей ПС предоставленные им ресурсы могут отличаться, иногда очень существенно. Этот вид защиты должен обеспечить, чтобы каждый пользователь ПС мог использовать только то, что ему предоставлено (санкционировано). Для этого ПС в своей внешней информационной среде может хранить информацию о своих пользователях и предоставленным им правах использования ресурсов, а также предоставлять пользователям определенные возможности формирования этой информации. Защита от несанкционированного доступа к ресурсам ПС осуществляется с помощью т.н. *паролей* (секретных слов). При этом предполагается, что каждый пользователь знает только свой пароль, зарегистрированный в ПС этим пользователем. Для доступа к выделенным ему ресурсам он должен предъявить ПС свой пароль. Другими словами, пользователь как бы "вешает замок" на предоставленные ему права доступа к ресурсам, "ключ" от которого имеется только у этого пользователя.

Различают две разновидности такой защиты:

- простая защита от несанкционированного доступа,
- защита от взлома защиты.

Простая защита от несанкционированного доступа обеспечивает защиту от использования ресурсов ПС пользователем, которому не предоставлены соответствующие права доступа или который указал неправильный пароль. При этом предполагается, что пользователь, получив отказ в доступе к интересующим ему ресурсам, не будет предпринимать попыток каким-либо несанкционированным образом обойти или преодолеть эту защиту. Поэтому этот вид защиты может применяться и в ПС, которая базируется на операционной системе, не обеспечивающей полную защиту от влияния «чужих» программ.

Защита от взлома защиты – это такая разновидность защиты от несанкционированного доступа, которая существенно затрудняет преодоление этой защиты. Это связано с тем, что в отдельных случаях могут быть предприняты настойчивые попытки взломать защиту от несанкционированного доступа, если защищаемые ресурсы представляют для кого-то чрезвычайную ценность. Для такого случая приходится предпринимать дополнительные меры защиты. Во-первых, необходимо обеспечить, чтобы такую защиту нельзя было обойти, т. е. должна действовать защита от влияния «чужих» программ. Во-вторых, необходимо усилить простую защиту от несанкционированного доступа использованием в ПС специальных программистских приемов, в достаточной степени затрудняющих подбор подходящего пароля или его вычисление по информации, хранящейся во внешней информационной среде ПС. Использование обычных паролей оказывается недостаточной, когда речь идет о чрезвычайно настойчивом стремлении добиться доступа к ценной информации. Если требуемый пароль («замок») в явном виде хранится во внешней информационной среде ПС, то "взломщик" этой защиты относительно легко может его достать, имея доступ к этому ПС. Кроме того, следует иметь в виду, что с помощью современных компьютеров можно осуществлять достаточно большой перебор возможных паролей с целью найти подходящий.

Защититься от этого можно следующим образом. Пароль (секретное слово или просто секретное целое число) X должен быть известен только владельцу защищаемых прав доступа и он не должен храниться во внешней информационной среде ПС. Для проверки прав доступа во внешней информационной среде ПС хранится другое число Y=F(X), однозначно вычисляемое ПС по предъявленному паролю X. При этом функция F может быть хорошо известной всем пользователям ПС, однако она должна обладать таким свойством, что восстановление слова X по Y практически невозможно: при достаточно большой длине слова X (например, в несколько сотен знаков) для этого может потребоваться астрономическое время. Такое число Y будем называть *электронной* (*компьютерной*) *подписью* владельца пароля X (а значит, и защищаемых прав доступа).

Другой способ защиты от взлома защиты связан с защитой сообщений, пересылаемых по компьютерным сетям. Такое сообщение может представлять команду на дистанционный доступ к ценной информации, и этот доступ отправитель сообщения хочет защитить от возможных искажений. Например, при осуществлении банковских операций с использованием компьютерной сети. Использование компьютерной подписи в такой ситуации недостаточно, так как защищаемое сообщение может быть перехвачено «взломщиком» (например, на «перевалочных» пунктах компьютерной сети) и подменено другим сообщением с сохранением компьютерной подписи (или пароля).

Защиту от такого взлома защиты можно осуществить следующим образом. Наряду с функцией F, определяющей компьютерную подпись владельца пароля X, в ПС определены еще две функции: Stamp и Notary. При передаче сообщения отправитель, помимо компьютерной подписи Y=F(X), должен вычислить еще другое число S=Stamp(X,R), где X – пароль, а R – текст передаваемого сообщения. Здесь также предполагается, что функция Stamp хорошо известна всем пользователям ПС и обладает таким свойством, что по S практически невозможно ни восстановить число X, ни подобрать другой текст сообщения R с заданной компьютерной подписью Y. При этом передаваемое сообщение (вместе со своей защитой) должно иметь вид:

(R Y S)

причем Y (компьютерная подпись) позволяет получателю сообщения установить истинность клиента, а S как бы скрепляет защищаемый текст сообщения R с компьютерной подписью Y. В связи с этим будем называть число S *электронной* (*компьютерной*) *печатью*. Функция Notary(R,Y,S).проверяет истинность защищаемого сообщения: (R,Y,S).
Эта позволяет получателю сообщения однозначно установить, что текст сообщения R принадлежит владельцу пароля X.

4.6. Защита от защиты. Защита от несанкционированного доступа может создать нежелательную ситуацию для самого владельца прав доступа к ресурсам ПС – он не сможет воспользоваться этими правами, если забудет (или потеряет) свой пароль («ключ»). Для защиты интересов пользователя в таких

ситуациях и предназначена защита от защиты. Для обеспечения такой защиты ПС должно иметь привилегированного пользователя, называемого *администратором ПС*. Администратор ПС должен, в частности, отвечать за функционирование защиты ПС: именно он должен формировать контингент пользователей данного экземпляра ПС, предоставляя каждому из этих пользователей определенные права доступа к ресурсам ПС. В ПС должна быть привилегированная операция (для администратора), позволяющая временно снимать защиту от несанкционированного доступа для пользователя с целью фиксации требуемого пароля («замка»).

Краткие выводы

В данной теме рассмотрены обеспечение автономности программного средства, обеспечение устойчивости программного средства, обеспечение защищенности программных средств. Подробно описаны защита от сбоев аппаратуры, защита от влияния «чужой» программы, защита от отказов «своей» программы, защита от ошибок пользователя, защита от несанкционированного доступа, защита от защиты.

Ключевые слова и определения

Самообнаружение ошибки в программе означает, что программа содержит средства обнаружения отказа в процессе ее выполнения.

Самоисправление ошибки в программе означает не только обнаружение отказа в процессе ее выполнения, но и исправление последствий этого отказа, для чего в программе должны иметься соответствующие средства.

Обеспечение устойчивости программы к ошибкам означает, что в программе содержатся средства, позволяющие локализовать область влияния отказа программы, либо уменьшить его неприятные последствия, а иногда предотвратить катастрофические последствия отказа.

Обеспечение независимости компонент означает разбиение системы на такие части, между которыми должны остаться по возможности меньше связей. Одним из воплощений этого метода является модульное программирование.

Обеспечение точности перевода направлено на достижение однозначности интерпретации документов различными разработчиками, а также пользователями ПС.

Смежный контроль означает, проверку полученного документа лицами, не участвующими в его разработке, с двух сторон: во-первых, со стороны автора исходного для контролируемого процесса документа, и, во-вторых, лицами, которые будут использовать полученный документ в качестве исходного в последующих технологических процессах.

Вопросы для обсуждения и контроля

1. Какие подходы по обеспечению надёжности известны в технике?
2. Какие методы борьбы со сложностью Вам известны?
3. Назовите этапы процесса перевода?
4. Охарактеризуйте виды контроля принимаемых решений.
5. Приведите структуру принимаемого решения.
6. Какие пути автономности программного средства Вам известны?
7. Какие факторы могут повлиять на работоспособность программного средства?
8. Какие уровни конфиденциальности информации Вы знаете?
9. Как Вы думаете, есть ли необходимость в защите программного средства?
10. Какие методы защиты информации Вы знаете?

Рекомендуемая литература

1. Гагарина Л.Г., Виснадул Б.Д., Игошин А.В. Основы технологии разработки программных продуктов: Учебное пособие. - М.: ФОРУМ: ИНФРА-М, 2006.
2. С.С.Ғуломов, Б.А.Бегалов, Н.Р.Зайналов, Р.А.Дадабаева, А.Э.Давронов. Дастурлаш технологиялари. Олий ўқув юртлари учун ўқув қўлланма. – Т.: ТДИУ, 2006.
3. Гвоздева В.А. Введение в специальность программиста: Учебник. – М.: ФОРУМ: ИНФРА – М, 2005.
4. Козырев А.А. Информационные технологии в экономике и управлении: Учебник. Второе издание. – СПб.:Изд-во Михайлова В.А., 2001.
5. Благодатских В.А., Енгибарян М.А., Ковалевская Е.В. Экономика, разработка и использование программного обеспечения ЭВМ. – М.: Финансы и статистика, 1995.
6. А.Н. Лебедев. Защита банковской информации и современная криптография // Вопросы защиты информации, 2(29), 1995.

Глава 12. ОСНОВНЫЕ ЭКОНОМИЧЕСКИЕ КАТЕГОРИИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

1. Понятие программного изделия.
2. Понятие качества программного изделия и связанные с ним характеристики
3. Экономическая эффективность программного изделия

1. Понятие программного изделия.

Общественное разделение труда привело к превращению сферы производства программных средств (программ) в одну из отраслей производственного труда, в составную часть общественного материального производства.

Программные средства создаются производителями чаще всего не для собственного потребления, а для продажи на внутреннем и внешнем рынках. Эти средства предназначены для удовлетворения потребностей пользователя по автоматизации управления.

Подтверждением этого факта является существование специализированных организаций по производству программных средств, их рекламе, продаже, поддержке их в актуальном состоянии.

Программное средство (ПС), предназначенное для продажи, существенно отличается от ПС для собственного потребления тем, что оно разрабатывается для обработки множества наборов данных с учетом конкретных условий самых разнообразных потребителей. Это ПС должно быть самым тщательным образом документировано для того, чтобы им могли пользоваться не только разработчики, но и широкий круг потребителей. Кроме того, должно быть проведено тестирование ПС с использованием вариантов исходных данных.

Исходя из вышеизложенного будем называть ПС, предназначенное для продажи, программным изделием и приведем его определение. **Программное изделие** (ПИ) - это программа на носителе данных, являющаяся продуктом промышленного производства (ГОСТ 19.004-80).

Процесс производства ПИ предполагает решение достаточно сложных организационно-экономических проблем. К ним относятся проблемы технологии разработки с решением таких функций управления, как планирование, учет, контроль, анализ и регулирование.

Кроме того, важное место должно отводиться вопросам нормирования труда производителей. Это связано со спецификой труда разработчиков ПС, а именно большой элемент творчества в работе, сложность измерения и оценки многих видов работ в процессе разработки программных изделий.

Специфические особенности ПИ проявляются также в отсутствии некоторых видов работ в процессе их создания и эксплуатации по сравнению с любым другим видом промышленной продукции, что показано в табл. 1.

Таблица 1

ЭТАПЫ СОЗДАНИЯ И ЭКСПЛУАТАЦИИ ПРОГРАММНЫХ ИЗДЕЛИЙ

Наименование этапов	Содержание работы	
	Продукция производственно-технического назначения	Программное изделие
1	2	3
Разработка	Определение требований пользователя Определение конструктивных элементов Проектирование конструктивных элементов Изготовление опытного образца и его испытание Создание технологии массового производства	Определение требований пользователя Определение конструктивных элементов Проектирование конструктивных элементов Реализация и тестирование
	Массовое производство	--
Ввод в эксплуатацию	Поставка пользователю	Копирование Поставка пользователю
Эксплуатация и обслуживание	Техническое обслуживание (ремонт) Возвращение изделия на доработку Расширение функциональных возможностей Физический износ Моральный износ	-- Сопровождение Сопровождение -- Моральный износ

Основные требования, предъявляемые к программному изделию:

ПИ как продукция производственно-технического назначения должно отвечать ряду требований.

1. Как и любая продукция, ПИ должно создаваться в соответствии с государственными отраслевыми стандартами (ГОСТами), согласованными с Государственным комитетом по вычислительной технике.

2. ПИ должно иметь установленную цену, согласованную с ведущими организациями-разработчиками ПС. В условиях рыночной экономики возможно существование договорных цен на ПИ.

3. При реализации ПИ должны быть особо оговорены вопросы совершенствования (модернизации) ПИ организациями-поставщиками.

4. Тщательное документирование ПИ обеспечивает возможность их применения пользователями различной квалификации. Состав и количество доку-

ментации, сопровождающей ПИ, определяются в соответствии с ГОСТами на ПИ.

2. Понятие качества программного изделия и связанные с ним характеристики

Прежде чем говорить о качестве ПИ, полезно напомнить смысловое содержание термина "качество продукции". Качество продукции - совокупность свойств продукции, обусловливающих ее способность удовлетворять определенной потребности в соответствии с ее назначением". Так как ПИ является одним из видов продукции производственно-технического назначения, то приведенное выше определение качества продукции можно однозначно отнести к ПИ.

После приобретения программами статуса продукции производственно-технического назначения вопросы качества ПИ стали особенно актуальными. Процесс разработки требует больших материальных и трудовых затрат, в связи с чем поставка некачественной программной продукции приводит к значительным потерям, причем не редко потери носят катастрофический характер. Подтверждением последнего может служить провал советской космической экспедиции на Венеру (проект "Фобос"). Из-за сбоев в программном обеспечении спутник изменил траекторию движения и не достиг намеченной цели.

Наиболее полный свод характеристик качества ПИ представлен в виде дерева, в котором более элементарные характеристики являются составными частями более обобщенных. Стрелки на рисунке указывают на логическое отношение следования (например, если программа удобна в эксплуатации, то она обязательно понятна, оцениваема и модифицируема).

Потребителя ПИ интересует:

• насколько хорошо (просто, надежно, эффективно) можно использовать данное ПИ в его исходном виде (исходная полезность);

• насколько удобно ПИ в эксплуатации (для понимания, модифицирования, повторных испытаний), можно использовать ПИ при изменении условий его применения (удобство эксплуатации).

Каждая из характеристик нижнего уровня может быть разбита на более конкретные свойства, которые раскрывают ее содержание.

Например, чтобы ПИ было мобильным, необходимо обеспечить максимальную независимость от типа ЭВМ (машино-независимость) и максимальную независимость от операционной системы.

Чтобы обеспечить свойство модифицируемости, ПИ должно быть хорошо структурировано, документировано, открыто к расширению существующих и добавлению новых функций.

Выделим совокупность характеристик качественного ПИ, количественная оценка которых позволяет определить, обладает ли данное ПИ тем или иным свойством.

Начнем с определения понятности ПИ, так как это наиболее комплексная характеристика качественного ПИ, включающая в себя ряд сопутствующих характеристик. Каждое ПИ должно создаваться с учетом требований пользователя, определенных в техническом задании. Кроме того, пользователь должен иметь возможность достаточно легко вникнуть в смысл документации, сопровождающей ПИ, понять его функциональное назначение. Все это может быть обеспечено, если ПИ описано ясным, лаконичные языком, без излишних повторений, с необходимыми ссылками.

Информативность *ПИ* - это одна из характеристик, обеспечивающая понятность ПИ. Говорят, что ПИ обладает характеристикой информативности, если оно содержит информацию, обеспечивающую понимание назначения ПИ, принятых ограничений, смыслового значения результатов работы отдельных компонентов ПИ.

Открытость *ПИ* дает возможность понять назначение каждого оператора ПИ при чтении ее текста, другими словами, каждый из идентификаторов должен нести смысловую нагрузку.

Согласованность, как характеристика качественного ПИ, бывает внутренней и внешней. Внутренняя согласованность должна обеспечивать единую терминологию, единую трактовку понятий и значений. Особое значение эта характеристика приобретает при создании программных комплексов, когда над проектом работает группа специалистов, и в процессе работы необходимы контакты по взаимоувязке программных модулей. Внешняя согласованность обеспечивается однозначным соответствием создаваемого ПИ требованиям, изложенным в техническом проекте на его разработку.

Структурированность ПИ делает его понятным для пользователя. Она предполагает создание ПИ в соответствии с определенными требованиями: использование при программировании четырех базовых конструкций, подробное комментирование текста программ, использование модульного программирования, ограничение на объем модулей и др. Перечисленные выше характеристики качественного ПИ в комплексе обеспечивают понятность ПИ.

Следующая характеристика качественного ПИ, которая также является комплексной, - **надежность**. Надежности ПИ будет посвящен параграф данного учебника. Остановимся на характеристиках, которые тесно связаны с ней.

Завершенность характеризует ПИ, которое включает все необходимые для функционирования программные компоненты, достаточно проработанные для выполнения заданных функций.

Характерным примером незавершенности ПИ может служить отсутствие в нем одного из модулей, обеспечивающего какой-либо дополнительный (не основной) режим работы. Например, модуль, обеспечивающий просмотр исходного документа при вводе данных с определенной страницы (предполагается, что документ многостраничный). Кроме того, примером незавершенности ПИ может служить некомплектная документация либо отсутствие в каком-либо из сопровождающих документов отдельных разделов.

Точность - характеристика, определяющая точность результатов расчета в соответствии с их назначением. Например, если в программе ведутся расчеты

по банковским операциям, то разумная точность - 3 знака после запятой с последующим округлением до двух знаков. Если в программе производятся расчеты по биологическим экспериментам на молекулярном уровне, то может потребоваться точность до 10 - 12.

Кроме вышеперечисленных характеристик качественного ПИ, можно отметить и ряд других.

Эффективность - выполнение требуемых функций при минимальных затратах ресурсов. Причем под ресурсами подразумеваются объем оперативной памяти, время работы центрального процессора, объем внешней памяти, пропускная способность канала. Часто характеристика эффективности вступает в противоречие с другими характеристиками качественного ПИ.

Например, ПИ будет более эффективным по времени работы, если будет состоять из меньшего количества модулей, чем это требует характеристика структурированности, так как на вызов модулей затрачивается относительно много машинного времени. Поэтому необходимость повышения эффективности ПИ за счет снижения других характеристик желательно оговаривать в техническом проекте на разработку ПИ.

Модифицируемость. Эта характеристика отражает возможность внесения изменений в ПИ без значительных затрат времени на последующую отладку. Эта характеристика тесно связана с характеристикой расширяемости, которая предполагает модификацию ПИ в части увеличения объема памяти либо числа функциональных модулей.

Последняя характеристика качественного ПИ, которую следует отметить,- **оцениваемость**. В случае, если ПИ обладает такой характеристикой, это обеспечивает возможность оценки качества ПИ. Для этого необходимо разработать критерий оценки и способ проверки соответствия ПИ этому критерию. Следует отметить, что ПИ, разбитое на модули по функциональному признаку, наиболее легко оценить, проверяя работу каждого из модулей, чем ПИ, состоящее из модулей, выделенных случайным образом.

В заключение следует отметить, что в каждом конкретном случае при оценке качества ПИ пользователь должен подбирать определенный набор характеристик, удовлетворяющих его требованиям. После этого производится определение значений конкретных показателей, о которых будет сказано в следующем параграфе. На основе найденных значений определяется некоторый интегральный показатель, который дает возможность оценить преимущества или недостатки приобретаемого ПИ.

3. Экономическая эффективность программного изделия

Эффективность - одно из наиболее общих экономических понятий, не имеющих пока, по-видимому, единого общепризнанного определения. По нашему мнению, это одна из возможных характеристик качества системы, а

именно ее характеристика с точки зрения соотношения затрат и результатов функционирования системы.

В дальнейшем будем понимать под экономической эффективностью ПИ меру соотношения затрат и результатов функционирования ПИ. К основным показателям экономической эффективности относятся: экономический эффект, коэффициент экономической эффективности капитальных вложений, срок окупаемости капитальных вложений.

Экономический эффект - результат внедрения какого-либо мероприятия, выраженный в стоимостной форме, в виде-экономии от его осуществления. Так, для организаций (предприятий), использующих ПИ, основными источниками экономии являются:

• улучшение показателей их основной деятельности, происходящее в результате использования ПИ;

• сокращение сроков освоения новых ПИ за счет их лучших эргономических характеристик;

• сокращение расхода машинного времени и других ресурсов на отладку и сдачу задач в эксплуатацию;

• повышение технического уровня, качества и объемов вычислительных работ;

• увеличение объемов и сокращение сроков переработки информации;

• повышение коэффициента использования вычислительных ресурсов, средств подготовки и передачи информации;

• уменьшение численности персонала, в том числе высококвалифицированного, занятого обслуживанием программных средств, автоматизированных систем, систем обработки информации, переработкой и получением информации;

• снижение трудоемкости работ программистов при программировании прикладных задач с использованием новых ПИ в организации - потребителе ПИ;

• снижение затрат на эксплуатационные материалы.

Коэффициент экономической эффективности капитальных вложений показывает величину годового прироста прибыли, образующуюся в результате производства или эксплуатации ПИ, на один Сум единовременных капитальных вложений.

Срок окупаемости (величина, обратная коэффициенту эффективности) - показатель эффективности использования капиталовложений - представляет собой период времени, в течение которого произведенные затраты на ПИ окупаются полученным эффектом.

Определение эффективности ПИ основано на принципах оценки экономической эффективности производства и использования в народном хозяйстве новой техники. Основные положения разработаны на основе и в развитие методики определения экономической эффективности использования в народном хозяйстве новой техники, изобретений и рационализаторских предложений [12] с учетом специфики ПИ.

На различных стадиях жизненного цикла ПИ и в зависимости от целей расчета рассчитываются и документально оформляются следующие виды экономического эффекта: предварительный, потенциальный, гарантированный и фактический.

Предварительный экономический эффект рассчитывается до выполнения разработки на основе данных технических предложений и прогноза использования. Предварительный эффект является элементом технико-экономического обоснования разработки ПИ и используется при планировании разработки и внедрения ПИ.

Потенциальный экономический эффект рассчитывается по окончании разработки на основе достигнутых технико-экономических характеристик и прогнозных данных о максимальных объемах использования ПИ в народном хозяйстве. Потенциальный эффект используется при оценке деятельности организаций-разработчиков ПИ.

Гарантированный экономический эффект рассчитывается в виде гарантированного экономического эффекта для конкретного объекта внедрения и общего гарантированного внедрения по ряду объектов.

Гарантированный экономический эффект для конкретного объекта внедрения рассчитывается после окончания разработки для одного программы внедрения на основе данных о гарантированном разработчиком удельном эффекте от применения ПИ и гарантированных пользователем сроках и годовом объеме использования ПИ. Гарантированный эффект от одного внедрения ПИ рассчитывается при оформлении договорных отношений между организацией-разработчиком и организацией-пользователем.

Гарантированный общий экономический эффект рассчитывается при постановке ПИ на производство на основе обобщения фактических показателей использования ПИ (по ряду объектов внедрения), а также данных об объемах внедрения ПИ, соответствующих возможностям изготовления, внедрения и сопровождения. Гарантированный общий эффект служит для разработки и утверждения экономически обоснованной цены на программную продукцию, выбора варианта производства и внедрения ПИ.

Фактический **экономический эффект** рассчитывается на основе данных учета и сопоставления затрат и результатов при конкретных применениях ПИ. Фактический эффект рассчитывается от одного программы внедрения конкретного ПИ на конкретном объекте, а также **как** общий экономический эффект от использования конкретного ПИ на всех объектах внедрения за расчетный период. Фактический эффект используется для оценки деятельности организаций, разрабатывающих, внедряющих и использующих ПИ, для определения размеров отчислений в фонды экономического стимулирования, а также для анализа эффективности функционирования ПИ и выработки технических предложений по совершенствованию ПИ и условий его применения.

Показатели экономической эффективности ПИ определяются:

- экономической оценкой результатов влияния ПИ на конечный результат их использования (основное направление анализа и расчета показателей эффективности для прикладных ПИ);

- экономической оценкой результатов влияния на технологические процессы подготовки, передачи, переработки данных в вычислительных системах (основное направление анализа и расчета показателей эффективности - для ПИ организации вычислительных процессов и эксплуатации средств вычислительной техники и ПИ, расширяющих функции операционных систем);
- экономической оценкой результатов влияния ПИ на технологический процесс создания новых ПИ (основное направление анализа и расчета показателей эффективности - для инструментально-технологических средств разработки и производства программного обеспечения).

При необходимости определения экономической эффективности ПИ, входящих в АСУ, САПР, АСНИ и другие системы через оценку влияния ПИ на конечные результаты функционирования этих систем в народном хозяйстве, доля эффекта от ПИ оценивается по коэффициенту долевого участия ПИ в показателях эффективности автоматизированных систем. Эти показатели рассчитываются по результатам 'основной деятельности организаций (предприятий, научных учреждений) на основе соответствующих общегосударственных, отраслевых и ведомственных методик. Долевой коэффициент участия ПИ в показателях эффективности автоматизированных систем в зависимости от условий расчета может определяться как отношение годовых приведенных затрат на ПИ к годовым приведенным затратам на систему, как отношение капитальных вложений в ПИ к капитальным вложениям в систему, как отношение трудозатрат на разработку ПИ к трудозатратам на разработку системы, а также методом экспертных оценок по взаимному соглашению разработчиков, изготовителей и пользователей.

Краткие выводы

В данной теме рассмотрена разница между программным изделием и программным средством, основные требования, предъявляемые к программному изделию. Кроме того, дана характеристика понятия качества программного изделия и связанных с ним характеристик. Также приведены основные показатели экономической эффективности.

Ключевые слова и определения:

Программное изделие (ПИ) - это программа на носителе данных, являющаяся продуктом промышленного производства (ГОСТ 19.004-80).

Информативность ПИ - это одна из характеристик, обеспечивающая понятность ПИ.

Открытость ПИ дает возможность понять назначение каждого оператора ПИ при чтении ее текста.

Структурированность ПИ делает его понятным для пользователя. Она предполагает создание ПИ в соответствии с определенными требованиями: использование при программировании четырех базовых конструкций, подробное комментирование текста программ, использование модульного программирования, ограничение на объем модулей и др.

Модифицируемость. Эта характеристика отражает возможность внесения изменений в ПИ без значительных затрат времени на последующую отладку.

Эффективность - выполнение требуемых функций при минимальных затратах ресурсов.

Экономический эффект - результат внедрения какого-либо мероприятия, выраженный в стоимостной форме, в виде-экономии от его осуществления. Так, для организаций (предприятий), использующих ПИ.

Коэффициент экономической эффективности капитальных вложений показывает величину годового прироста прибыли, образующуюся в результате производства или эксплуатации ПИ, на один Сум единовременных капитальных вложений.

Срок окупаемости (величина, обратная коэффициенту эффективности) - показатель эффективности использования капиталовложений - представляет собой период времени, в течение которого произведенные затраты на ПИ окупаются полученным эффектом.

Предварительный экономический эффект рассчитывается до выполнения разработки на основе данных технических предложений и прогноза использования.

Потенциальный экономический эффект рассчитывается по окончании разработки на основе достигнутых технико-экономических характеристик и прогнозных данных о максимальных объемах использования ПИ.

Фактический экономический эффект рассчитывается на основе данных учета и сопоставления затрат и результатов при конкретных применениях ПИ.

Вопросы для обсуждения и контроля:

1. Что понимают под программным средством (ПС), программным изделием (ПИ)?
2. Перечислите основные характеристики качественного ПИ.
3. Что понимают под экономической эффективностью ПС?
4. Что показывает коэффициент экономической эффективности капитальных вложений, срок окупаемости?
5. Что такое экономический эффект?
6. Какие существуют виды экономического эффекта?
7. Когда рассчитываются: предварительный экономический эффект; потенци-

альный экономический эффект; гарантированный экономический эффект; фактический экономический эффект?
8. Что определяют показатели экономической эффективности программного изделия?

Рекомендуемая литература

1. Вигерс Коре Разработка требований к программному обеспечению. - М.: Издательско-торговый дом “Русская редакция”, 2004. – 400-410с.
2. Козырев А.А. Информационные технологии в экономике и управлении: Учебник. Второе издание. – СПб.:Изд-во Михайлова В.А., 2001. 329-330 стр.
3. Благодатских В.А., Енгибарян М.А., Ковалевская Е.В. Экономика, разработка и использование программного обеспечения ЭВМ. - М.: Финансы и статистика, 1995. 15-24 стр.

Глава 13. ОБЪЕКТНЫЙ ПОДХОД К РАЗРАБОТКЕ ПРОГРАММНЫХ СРЕДСТВ

1. Объекты и отношения в программировании. Сущность объектного подхода к разработке программных средств.
2. Особенности объектного подхода к разработке внешнего описания программного средства.
3. Особенности объектного подхода на этапе конструирования программного средства.

1. Объекты и отношения в программировании. Сущность объектного подхода к разработке программных средств.

Окружающий нас мир состоит из объектов и отношений между ними. Согласно В. Далю объект (предмет) – это все, что представляется чувствам (объект вещественный) или уму (объект умственный). Таким образом, *объект* воплощает некоторую сущность и имеет некоторое состояние, которое может изменяться со временем как следствие влияния других объектов, находящихся с первым в каких-либо отношениях. Он может иметь внутреннюю структуру: состоять из других объектов, также находящихся между собой в некоторых отношениях. Исходя из этого, можно построить иерархическое строение мира из объектов. Однако, при каждом конкретном рассмотрении окружающего нас мира некоторые объекты считаются неделимыми, причем в зависимости от целей рассмотрения такими (неделимыми) могут приниматься объекты разного уровня иерархии. *Отношение* связывает некоторые объекты: можно считать, что объединение этих объектов обладает некоторым свойством. Если отношение связывает n объектов, то такое отношение называется n-местным (n-арным). На каждом месте объединения объектов, которые могут быть связаны каким-либо конкретным отношением, могут находиться разные объекты, но вполне определенные (в этом случае говорят: объекты определенного класса). Одноместное отношение называется *простым свойством объекта* (соответствующего класса). Многоместное отношение объектов будем называть *ассоциативным свойством объекта*, если этот объект участвует в этом отношении. Состояние объекта может быть изучено по значению простых или ассоциативных свойств этого объекта. Множество всех объектов, которые обладают каким-то общим набором свойств, называется *классом объектов*.

В процессе познания или изменения окружающего нас мира мы всегда принимаем в рассмотрение ту или иную упрощенную модель мира (*модельный мир*), в которую включаем объекты и отношения некоторых интересующих нас классов из окружающего нас мира. Каждый объект, имеющий внутреннюю структуру, может представлять свой модельный мир, включающий объекты этой структуры и отношения, которые их связывают. Таким образом, окружающий нас мир, можно рассматривать (в некотором приближении) как иерархическую структуру модельных миров.

В настоящее время в процессе познания или изменения окружающего нас мира широко используется компьютерная техника для обработки различного рода информации. В связи с этим применяется компьютерное (информационное) представление объектов и отношений. Каждый объект информационно может быть представлен некоторой структурой данных, отображающей его состояние. Простые свойства этого объекта могут задаваться непосредственно в виде отдельных компонент этой структуры, либо специальными функциями над этой структурой данных. Ассоциативные свойства (n-местные отношения для $n>1$) можно представить либо в активной форме, либо в пассивной форме. В активной форме n-местное отношение представляется некоторым программным фрагментом, реализующим либо n-местную функцию (определяющую значение свойства соответствующего объединения объектов), либо процедуру, осуществляющую по состоянию представлений объектов, связываемых представляемым отношением, изменение состояний некоторых из них. В пассивной форме такое отношение может быть представлено некоторой структурой данных (в которую могут входить и представления объектов, связываемых этим отношением), интерпретируемую на основании принятых соглашений по общим процедурам, независящим от конкретных отношений (например, реляционная база данных). В любом случае представление отношения определяет некоторые действия по обработке данных.

При исследовании модельного мира пользователи могут по-разному получать (или захотеть получать) информацию от компьютера.

В одних случаях пользователей может интересовать получение информации об отдельных свойствах определенных объектов или результаты какого-либо взаимодействия между некоторыми объектами модельного мира. Для удовлетворения таких запросов разрабатываются соответствующие ПС, которые выполняют интересующие пользователей функции, или подходящие информационные системы, способные выдавать информацию об интересующих пользователей отношениях. В начальный период развития компьютерной техники (при не достаточно высокой мощности компьютеров) такой подход к исследованию модельного мира был вполне естественным. Именно он и провоцировал *функциональный* (*реляционный*) подход к разработке ПС, который был подробно рассмотрен в предшествующих лекциях. Сущность этого подхода состоит в систематическом использовании *декомпозиции функций* (*отношений*) для описания и построения структуры ПС (включая тексты программ). При этом сами объекты модельного мира, с которыми связаны заказываемые и реализуемые функции, представлялись фрагментарно (в том объеме, который необходим для выполнения этих функций) и в форме, удобной для реализации этих функций. Тем самым обеспечивалась эффективная реализация требуемых функций, но не создавалось цельного и адекватного компьютерного представления модельного мира, интересующего пользователя. Попытки даже незначительного расширения объема и характера информации об этом модельном мире, которую можно получить от ПС, могло потребовать серьезной модернизации этого ПС.

В других случаях пользователя может интересовать наблюдение за изменением состояний объектов модельного мира в результате их взаимодействий. Это требует использования подходящих информационных моделей таких объектов, создания программных средств, моделирующих процессы взаимодействия объектов модельного мира, и предоставление пользователю доступа к этим информационным моделям (к *пользовательским о*бъектам). С помощью традиционных методов разработки это оказалось довольно трудоемкой задачей. Наиболее полно отвечает решению этой задачи *объектный* подход к разработке ПС. Сущность его состоит в систематическом использовании *декомпозиции объектов* при описании и построении ПС. При этом функции (отношения), выполняемые таким ПС, будут выражаться через отношения объектов других уровней, т.е. их декомпозиция будет существенно зависеть от декомпозиции объектов.

С точки зрения разработчиков ПС следует различать следующие категории объектов (и, соответственно, их классов):

- объекты модельного (вещественного или умственного) мира,
- информационные модели объектов реального мира (будем называть их *пользовательскими объектами*),
- объекты процесса выполнения программ,
- объекты процесса разработки ПС (*технологические объекты программирования*).

Кроме того, в зависимости от способа представления в компьютере модельного мира и характера взаимодействия с ним со стороны пользователя следует различать пассивные и активные объекты. *Пассивный объект* представляет собой некоторый фрагмент информационной среды, который способен хранить разные данные определенного типа (представляющие разные *состояния* этого объекта) и с которым связан некоторый набор операций (*применимых* к этому объекту). Операции над таким объектом применяются под воздействием некоторой *внешней* по отношению к этому объекту *активной силы*, исходящей либо от пользователя, либо от какого-либо программного фрагмента в процессе его выполнения. *Активный объект* представляет собой такое расширение пассивного объекта, в котором фрагмент информационной среды способен также хранить и программные фрагменты, способные находиться в процессе выполнения (в *активном состоянии*). Активный объект, у которого какие-либо программные фрагменты находятся в активном состоянии, способен воспринимать сообщения или сигналы из операционной среды, в которую он погружен, и самостоятельно выполнять некоторые операции как реакцию на эти сообщения или сигналы. Таким образом, можно считать, что активный объект обладает *внутренней активной силой*.

Когда говорят об объектно-ориентированном подходе к разработке ПС, имеют в виду объектный подход с ориентацией на описание объектов модельного мира и построением их информационных моделей, причем используются, в основном, активные объекты. При этом многие процессы разработки ПС приобретают специфические («объектные») черты:

- использование системы понятий, позволяющих описывать объекты и их классы,
- декомпозиция объектов является основным средством упрощения ПС,
- использование внепрограммных абстракций для упрощения процессов разработки,
- предпочтение (приоритет) разработки структуры данных перед реализацией функций.

Основные из этих специфических особенностей разработки ПС покажем в рамках водопадной модели технологии.

2. Особенности объектного подхода к разработке внешнего описания программного средства.

При объектном подходе этап внешнего описания ПС оказывается существенно более емким и содержательным по сравнению с реляционным подходом.

Определение требований заключается в неформальном описании модельного мира, который пользователь собирается изучать или просто использовать с помощью требуемого ПС. При этом повышается роль прототипирования, которое при этом подходе часто окупается уменьшением объема работы на последующих этапах разработки ПС.

Спецификация качества сохраняет свое содержание.

Существенно изменяется содержание процесса спецификации требований: вместо разработки функциональной спецификации ПС создается формальное описание модельного мира, состоящее из трех частей:

- объектной модели,
- динамической модели,
- функциональной модели.

Назначение этих частей можно образно определить следующим образом [15.3]: объектная модель определяет то, с чем что-то случается; динамическая модель определяет, когда это случается; функциональная модель определяет то, что случается.

Объектная модель показывает статическую объектную структуру модельного мира, который должно представлять разрабатываемое ПС (программная система). Она включает определения используемых классов объектов и отношений между этими классами, а также определение используемых объектов этих классов и отношения между этими объектами.

Обычно *класс объектов* в объектной модели представляется в виде тройки (Имя класса, Список атрибутов, Список операций).

Каждый атрибут представляется некоторым именем и может принимать значения определенного типа. По существу, атрибут класса выражает некоторое простое свойство объектов этого класса. Представление некоторых простых свойств объектов атрибутами весьма удобно, особенно когда по значениям этих

атрибутов осуществляется классификация объектов. Операции, указываемые в представлении класса, отражают другие свойства объектов этого класса (как простые, так и ассоциативные). Они показывают, что можно делать с объектами этого класса (или что могут делать сами эти объекты).

В объектной модели отношение между объектами обобщаются в отношения между этими классами, к которым относятся эти объекты. При этом используются, как правило, только одноместные и двуместные отношения между объектами. Более сложные отношения приводит к неоправданному усложнению объектных моделей, а с другой стороны, такие отношения всегда могут быть сведены к двуместным за счет определения дополнительных классов. Одноместные отношения называют *атрибутами*, причем некоторые атрибуты объекта получаются из атрибутов класса присвоением им конкретных значений. Отношение между двумя (и более) объектами называют *связями*, а их обобщение (отношение между классами) обычно называют *ассоциацией*. Ассоциации играют важную роль в объектной модели – они определяют допустимые связи между объектами. Различают следующие виды ассоциаций:

- взаимодействия состояний объектов,
- агрегирования (структурирования) объектов,
- абстрагирования (порождения) классов.

Ассоциация «взаимодействие», по существу, означает, что объекты классов, находящихся в таком отношении, могут быть параметрами некоторых операций. Ассоциация «агрегирование» означает, что объект одного из классов, находящихся в таком отношении, включает (или может включать) в себя (как часть) объекты другого из этих классов. Ассоциация «абстрагирование» означает, что один из классов, находящихся в таком отношении, наследует свойства другого из этих классов и может обладать также и другими (дополнительными) свойствами.

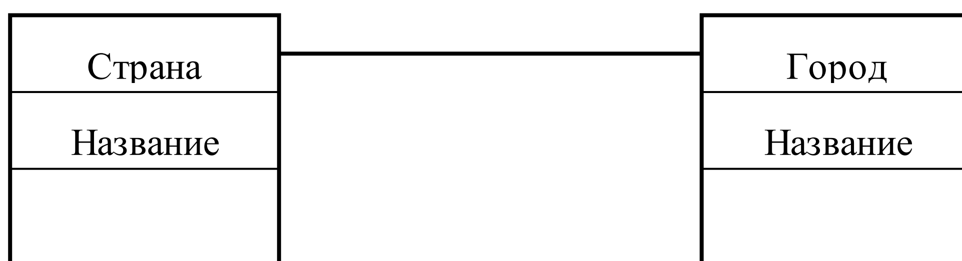


Рис. 1. Пример отношения между классами объектов.

Для представления объектной модели часто используются графические языки спецификации объектов (например, язык UML). На таких языках классы и объекты задаются прямоугольниками, которых указывается специфицирующая их информация. Для задания отношений между двумя классами соответствующие им прямоугольники связываются линией, снабженной различными

графическими значками и некоторыми надписями. Графические значки специфицируют характер (вид) отношения между этими классами, а надписи обеспечивают полную идентификацию этого отношения (делают его конкретным). Например, на рис. 1 заданное отношение между классами Страна и Город имеет характер «один к одному». Более конкретно это отношение означает, что каждый объект класса Страна обязательно связан отношением «имеет столицей» с одним и только одним объектом класса Город, и этот объект класса Город не связан таким отношением ни с каким другим объектом класса Страна.

Для описания декомпозиции объектов используется отношение вида агрегирования. Например, отношение «программа состоит из одного или нескольких модулей» представлено на рис. 2.

Рис. 2. Пример отношения агрегирования между классами объектов.

Следует заметить, что объектная модель полностью включает описание внешней информационной среды при реляционном подходе.

Динамическая модель показывает допустимые последовательности изменений состояний объектов из объектной модели модельного мира, который должно представлять разрабатываемое ПС (программная система). Она описывает последовательности операций в ответ на внешние сигналы (взаимодействия) без рассмотрения того, что эти операции делают. Динамическая модель необходима, если в соответствующей объектной модели имеются активные объекты.

Основные понятия динамической модели: события и состояния объектов. Под *событием* здесь понимается элементарное воздействие одного объекта на другого, происходящее в определенный момент времени. Одно событие может логически предшествовать другому или быть не связанным с другим. Другими словами, события в динамической модели *частично упорядочены*. Под *состоянием объекта* здесь понимается совокупность значений атрибутов объекта и представления текущих связей этого объекта с другими объектами. Состояние объекта связывается с интервалом времени между некоторыми двумя событиями, на которые реагирует этот объект. Объект *переходит* из одного состояния в другое в результате реакции на некоторое событие (в конце интервала, связанного с этим состоянием).

В связи с этим в динамической модели для каждого класса активных объектов строится своя *диаграмма состояний*. Она представляет собой граф, вершинами которого являются состояния, а дугами – переходы между этими со-

стояниями, обозначаемые именами событий. Некоторые переходы могут быть связаны с условиями, разрешающими эти переходы. *Условие* – это предикат, зависящий от значений некоторых атрибутов объекта. Каждое условие указывается на дуге, переходом по которой управляет это условие. Существенно, что в диаграмме состояний с некоторыми состояниями или событиями связываются определенные операции. Операция, связываемая с событием, обозначает реакцию объекта на это событие и считается, что она выполняется мгновенно (в точке некоторого временного интервала). Такая операция называется *действием*. Операция, связываемая с состоянием, выполняется в рамках временного интервала, с которым связано это состояние (т.е. имеет продолжительность, ограниченную этим интервалом). Такая операция называется *деятельностью*. Диаграмма состояний определяет управление активизацией указанных операций. Таким образом, диаграмма состояний описывает поведение одного класса объектов.

Динамическая модель в целом объединяет все диаграммы состояний с помощью событий между классами.

Функциональная модель показывает, как вычисляются выходные значения из входных без указания порядка, в котором эти значения вычисляются. Она определяет все операции, условия и ограничения, используемые в объектной и динамической моделях (*внешние операции*). Функциональная модель соответствует определению *внешних функций* при реляционном подходе к разработке ПС.

Для определения крупных операций в функциональной модели используются *потоковые диаграммы* (*диаграммы потоков данных*), позволяющие выразить эти операции через более простые операции. Основными понятиями потоковых диаграмм являются *процессы*, *объекты* и *потоки данных*. Потоковая диаграмма – это граф, вершинами которого являются объекты или процессы, а дугами – потоки данных. Процессы преобразуют данные, поступающие от одних объектов и направляемые для хранения в другие объекты. Эти процессы представляют *внутренние* операции, через которые выражается операция, представляемая данной потоковой диаграммой. Объекты могут быть пассивными (*хранилищами данных*) и активными (*агентами*). Пассивные объекты используются только для хранения данных, а активные объекты используются как для хранения, так и для преобразования данных. Потоки данных определяют допустимые направления перемещения данных и типы перемещаемых данных.

Процессы могут выражаться терминальными операциями (определяемые непосредственно) или с помощью других потоковых диаграмм. Таким образом, потоковые диаграммы являются иерархическими.

Терминальные операции определяются так же, как и при реляционном подходе. Впрочем, и диаграммы потоков данных используются при реляционном подходе.

Таким образом, основным содержанием этапа внешнего описания при объектном подходе является *объектное моделирование*. При этом широко используются формальные языки спецификаций, в том числе и графические. Од-

ним из наиболее употребительных в настоящее время таких языков является язык UML.

3. Особенности объектного подхода на этапе конструирования программного средства.

На этапе конструирования при объектном подходе продолжается процесс объектного моделирования: уточняются модели, построенные на этапе внешнего описания, в терминах описания программных систем и производится дальнейшая декомпозиция объектов.

В процессе разработки объектной архитектуры ПС выделяются все объекты, с информационными моделями которых собирается непосредственно работать пользователь, и завершается их программная спецификация, а так же определяется их пользовательский интерфейс. Такие объекты мы будем называть *пользовательскими*. Классы таких объектов или отдельные активные объекты образуют архитектурные подсистемы. Определяется метод взаимодействия между этими подсистемами.

В случае использования активных объектов основным широким классом архитектур при объектном подходе является *коллектив параллельно действующих программ*, причем здесь роль программ выполняют как раз эти активные объекты. Типичной архитектурой такого класса является архитектура «клиент-сервер». В такой системе один из активных объектов, называемый *сервером*, выполняет определенные программные услуги по запросам других активных объектов, называемых *клиентами*. Такой запрос передается серверу с помощью сообщения от клиента, результат выполнения сервером запроса передаются соответствующему клиенту с помощью другого сообщения.

Дальнейшая разработка структуры программных подсистем и их кодирование на языках программирования может осуществляться уже в рамках реляционного подхода на ориентированных на него языках программирования – пользователь внутреннюю организацию этих подсистем уже «не видит». Однако, во многих случаях существуют сильные аргументы за то, чтобы продолжить объектную декомпозицию этих подсистем. Объектная структура этих подсистем может быть существенно более понятной *разработчику*, чем их структура при реляционном подходе. Кроме того, продолжение объектной декомпозиции и использование основных понятий и методов объектного подхода при дальнейшей разработке ПС представляется «естественным», так как весь процесс разработки становится единообразным (концептуально целостным). При этом приходится использовать языки программирования уже другого типа – объектно-ориентированные. Объекты, возникающие в программах при такой декомпозиции архитектурных подсистем, мы будем называть *объектами процесса выполнения программ*.

Ключевые слова и определения:

Объект воплощает некоторую сущность и имеет некоторое состояние, которое может изменяться со временем как следствие влияния других объектов, находящихся с первым в каких-либо отношениях.

Множество всех объектов, которые обладают каким-то общим набором свойств, называется **классом объектов**.

Пассивный объект представляет собой некоторый фрагмент информационной среды, который способен хранить разные данные определенного типа (представляющие разные *состояния* этого объекта) и с которым связан некоторый набор операций (*применимых* к этому объекту).

Активный объект представляет собой такое расширение пассивного объекта, в котором фрагмент информационной среды способен также хранить и программные фрагменты, способные находиться в процессе выполнения (в *активном состоянии*).

Объектная модель показывает статическую объектную структуру модельного мира, который должно представлять разрабатываемое ПС (программная система).

Динамическая модель показывает допустимые последовательности изменений состояний объектов из объектной модели модельного мира, который должно представлять разрабатываемое ПС (программная система).

Под **событием** здесь понимается элементарное воздействие одного объекта на другого, происходящее в определенный момент времени.

Функциональная модель показывает, как вычисляются выходные значения из входных без указания порядка, в котором эти значения вычисляются. Она определяет все операции, условия и ограничения, используемые в объектной и динамической моделях (*внешние операции*).

Вопросы для обсуждения и контроля:

1. Что такое менеджер программного средства?
2. Что такое ординарный пользователь программного средства?
3. Что такое администратор программного средства?
4. Что такое руководство по инсталляции программного средства?
5. Что такое руководство по управлению программным средством?
6. Что такое руководство по сопровождению программного средства?

Рекомендуемая литература

1.Delphi 7. Учебный курс / С.И.Бобровский.-СПб.: Питер, 2003. 17-19, 99-129 стр.

2.Фаронов В.В. Delphi 4.0. Учебный курс. М.: Нолидж,1998. 10-22 стр.

3.Сван Т. Основы программирования в Delphi для Windows 95. – К.: «Диалектика», 10-19, 25-26, 29-38 стр

4. J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorenzen. Objekt-Oriented Modeling and Design. – Prentice Hall. 1991.

5. Г.Буч. Объектно-ориентированное проектирование с примерами применения: пер. с англ. – М.: Конкорд, 1992.

6. М. Фаулер, К. Скотт. UML в кратком изложении. - М.: Мир, 1999.

7. В.Ш.Кауфман. Языки программирования. Концепции и принципы. – М.: Радио и связь, 1993.

ГЛОССАРИЙ ПО УЧЕБНОЙ ДИСЦИПЛИНЕ

Активный объект представляет собой такое расширение пассивного объекта, в котором фрагмент информационной среды способен также хранить и программные фрагменты, способные находиться в процессе выполнения (в *активном состоянии*).

Базовое программное обеспечение (base software) — минимальный набор программных средств, обеспечивающих работу компьютера.

Верификация – проверка правильности кода или иных конструкций в тексте программы.

Внедрение – комплекс мероприятий по установке программы на ЭВМ пользователя, ее тестирование на реальных данных и обеспечение функционирования для решения задач.

Дедуктивный подход - рассматривается частный случай общеизвестной фундаментальной модели.

Дескрипторные системы — формат ключевых слов сообщений.

Динамическая модель показывает допустимые последовательности изменений состояний объектов из объектной модели модельного мира, который должно представлять разрабатываемое ПС (программная система).

Жизненный Цикл Программного Изделия (ЖЦПИ) – схема или временная модель, отражающая основные этапы существования ПС от разработки до завершения эксплуатации пользователями.

Задача (problet, (task) - проблема, подлежащая решению.

Запрос-ответ — фиксирован перечень возможных значений, выбираемых из списка, или ответы *типа Да/Нет*.

Запрос по формату — с помощью ключевых слов, фраз или путем заполнения экранной формы с регламентированным по составу и структуре набором реквизитов осуществляется подготовка сообщений.

Индуктивный способ - предполагает выдвижение гипотез, декомпозицию сложного объекта, анализ, затем синтез.

Инструментарий технологии программирования — совокупность программ и программных комплексов, обеспечивающих технологию разработки, отладки и внедрения создаваемых программных продуктов.

Информативность ПИ - это одна из характеристик, обеспечивающая понятность ПИ.

Кнопка-переключатель <option button> — для альтернативного выбора кнопки из группы однотипных кнопок (например, семейное положение).

Командная кнопка (command button) — обеспечивает передачу управляющего воздействия, например, кнопки <Сапсе1>, <ОК>, <0тмена>; выбор режима обработки типа <Ввод>, <Удаление>, <Редактирование>, <Выход> и др.

Комбинированное окно (combo box) — объединяет возможности окна-списка и текстового окна (например, «Предметы по выбору» — можно указать новый предмет или выбрать один из предлагаемого списка);

Комплексирование – совместная отладка модулей программы в виде единого программного комплекса.

Коэффициент экономической эффективности капитальных вложений показывает величину годового прироста прибыли, образующуюся в результате производства **или** эксплуатации ПИ, на один Сум единовременных капитальных вложений.

Меню — диалог инициируется программой; пользователю предлагается выбор альтернативы функций обработки из фиксированного перечня; предоставляемое меню может быть иерархическим и содержать вложенные подменю следующего уровня.

Метка (Label) — постоянный текст, не подлежащий изменению при работе пользователя с экранной формой (например, слова *Фамилия Имя Отчество)*.

Метод главного программиста заключается в том, что реализация программного проекта от начала до конца (проектирования, программирования, отладки, выпуска технической документации и инструкций пользователю) осуществляется под руководством главного программиста, который стоит во главе бригады (3— 10 человек)

Метод иерархических диаграмм - в этом методе определяется связь между входными, выходными данными и процессом обработки с помощью иерархической декомпозиции системы (без детализации). По сути используются три элемента: вход, обработка, выход.

Методология Джексона - структура программы определяется структурой данных, подлежащих обработке. Программа представляется как механизм, с помощью которого входные данные преобразуются выходные.

Методология Уорнера - подобна предыдущей методологии, но процедура проектирования более детализирована.

Методо-ориентированный пакет предназначен для решения задачи пользователя одним из нескольких методов, предусмотренных в пакете, причем метод либо назначается пользователем, либо выбирается автоматически на основе анализа входных данных.

Модифицируемость. Эта характеристика отражает возможность внесения изменений в ПИ без значительных затрат времени на последующую отладку.

Модуль - это отдельная, функционально законченная программная единица, которая структурно идентифицируется (или оформляется) стандартным образом по отношению к компилятору и по отношению к объединению ее с другими аналогичными единицами и загрузке.

Модульное программирование - каждый модуль тестируется отдельно, затем после кодирования и тестирования происходит их интеграция и тестируется вся система.

Моральный износ ПС – нецелесообразность эксплуатации ПС в реальных условиях в связи с изменением среды эксплуатации или иных внешних факторов.

Объект воплощает некоторую сущность и имеет некоторое состояние, которое может изменяться со временем как следствие влияния других объектов, находящихся с первым в каких-либо отношениях.

Объектная модель показывает статическую объектную структуру модельного мира, который должно представлять разрабатываемое ПС (программная система).

Окно-список (list box) — содержит список альтернативных значений для выбора (например, «Спортивная секция»).

Операционные оболочки — специальные программы, предназначенные для облегчения общения пользователя с командами операционной системы.

Открытость ПИ дает возможность понять назначение каждого оператора ПИ при чтении ее текста.

Отладка ПС – это деятельность, направленная на обнаружение и исправление ошибок в ПС с использованием процессов выполнения его программ.

Пакет прикладных программ- это совокупность совместимых программ для решения определенного класса задач. ППП всегда ориентируется на пользователей определенной квалификации как в программировании, так и в той области, к которой относятся задачи, решаемые с применением этого ППП.

Пассивный объект представляет собой некоторый фрагмент информационной среды, который способен хранить разные данные определенного типа (представляющие разные *состояния* этого объекта) и с которым связан некоторый набор операций (*применимых* к этому объекту).

ПВЯ – входной язык ППП, средство управления работой ППП.

Помечаемая кнопка <chek button> — для аддитивного выбора несколько кнопок из группы однотипных кнопок (например, факулыпатив для посещения).

Потенциальный экономический эффект рассчитывается по окончании разработки на основе достигнутых технико-экономических характеристик и прогнозных данных о максимальных объемах использования ПИ.

Предварительный экономический эффект рассчитывается до выполнения разработки на основе данных технических предложений и прогноза использования.

Предметная область пакета - область науки или деятельности, к которой относятся задачи, решаемые с применением ППП.

Приложение (application) - программная реализация на компьютере решения задачи.

Программа (program, routine) - упорядоченная последовательность команд (инструкций) компьютера для решения задачи.

Программирование сверху-вниз - это некоторая многоуровневая дисциплина написания программ. На верхнем уровне исходный алгоритм представляется в виде некоторой иерархической схемы, элементы которой описываются на естественном для данной проблемы языке.

Программное изделие (ПИ) - это программа на носителе данных, являющаяся продуктом промышленного производства.

Программное обеспечение (software) - совокупность программ обработки данных и необходимых для их эксплуатации документов.

Программный продукт - комплекс взаимосвязанных программ для решения определенной проблемы (задачи) массового спроса, подготовленный к реализации как любой вид промышленной продукции.

Программотехника (software engineering) — технология разработки, отладки, верификации и внедрения программного обеспечения.

Проблемно-ориентированные пакеты предназначены для решения групп (последовательностей) задач, использующих общие данные.

Проектирование с использованием потока данных - использует поток данных как генеральную линию проектирования программы, содержит элементы структурного проектирования сверху-вниз с пошаговой детализацией.

Рабочая документация (рабочий проект)- на данном этапе осуществляется адаптация базовых средств программного (операционной системы, СУБД, методо-ориентированных ППП, инструментальных сред конечного пользователя — текстовых редакторов, электронных таблиц и т.п.).

Рамка (frame) — объединение объектов управления в группу по функциональному или другому принципу (например, для изменения их параметров).

Руководство пользователя — включает детальное описание функциональных возможностей и технологии работы с программным продуктом. Данный вид документации ориентирован на *конечного* пользователя и содержит необходимую информацию самостоятельного освоения и нормальной работы пользователя (с учетом квалификации пользователя);

Руководство программиста (оператора)—указывает особенности установки (инсталляции) программного продукта и его внутренней структуры – состав и назначения модулей, правила эксплуатации и обеспечения надежной и качественной работы программного продукта.

CASE-технология — программный комплекс, автоматизирующий весь технологический процесс анализа, проектирования, разработки и сопровождения сложных программных систем.

Сервисное программное обеспечение — программы и программные комплексы, которые расширяют возможности базового программного обеспечения и организуют более удобную среду работы пользователя.

Сетевые операционные системы — комплекс программ, обеспечивающих обработку, передачу и хранение данных в сети.

Системы с жестким сценарием диалога — стандартизированное представление информации обмена;

Системы с языком деловой прозы — представление сообщений на языке, естественном для профессионального пользования.

Событие - элементарное воздействие одного объекта на другого, происходящее в определенный момент времени.

Составление технического задания на программирование- на этом этапе выбирают методы решения задачи; разрабатывают обобщенный алгоритм решения комплекса задач, функциональную структуру алгоритма или состав

объектов, определяют требования к комплексу технических средств системы обработки информации, интерфейсу конечного пользователя.

Средства для создания приложений — совокупность языков и систем программирования, а также различные программные комплексы для отладки и поддержки создаваемых программ.

Срок окупаемости (величина, обратная коэффициенту эффективности) - показатель эффективности использования капиталовложений - представляет собой период времени, в течение которого произведенные затраты на ПИ окупаются полученным эффектом.

Структурированность ПИ делает его понятным для пользователя. Она предполагает создание ПИ в соответствии с определенными требованиями: использование при программировании четырех базовых конструкций, подробное комментирование текста программ, использование модульного программирования, ограничение на объем модулей и др.

Структурное программирование основано на фиксации для программиста допустимых структур.

Тезаурусные системы — семантическая сеть дескрипторов, образующих словарь системы (аналог — гипертекстовые системы);

Текстовое окно (text Ьох) — используется для ввода информации произвольного вида, отображения хранимой информации в базе данных (например, для ввода фамилии студента).

Тестирование - это процесс исполнения программ с целью выявления (обнаружения) ошибок.

Технический проект – на этом этапе разрабатывается детальный алгоритм обработки данных, определяется состав общесистемного программного обеспечения, разрабатывается внутренняя структура программного продукта, осуществляется выбор инструментальных средств разработки программных модулей.

Технология программирования (programming technology) - совокупность производственных процессов, приводящую к созданию требуемого программного средства, а также описание этой совокупности процессов.

Технология структурного анализа проекта - основана на структурном анализе с использованием специальных графических средств построения иерархических функциональных связей между объектами системы.

Утилитарные программы ("программы для себя") предназначены для удовлетворения нужд их разработчиков.

Утилиты — программы, служащие для выполнения вспомогательных с операций обработки данных или обслуживания компьютеров

Фактический экономический эффект рассчитывается на основе данных учета и сопоставления затрат и результатов при конкретных применениях ПИ.

Функциональная декомпозиция - подобна стратегии «разделяй и управляй». Практически является декомпозицией в форме пошаговой детализации и концепции скрытия информации. Каждый модуль характеризуется субъективным решением проектировщика, связь осуществляется с помощью хорошо организованных интерфейсов.

Функциональная модель показывает, как вычисляются выходные значения из входных без указания порядка, в котором эти значения вычисляются. Она определяет все операции, условия и ограничения, используемые в объектной и динамической моделях (*внешние операции*).

Экономический эффект - результат внедрения какого-либо мероприятия, выраженный в стоимостной форме, в виде-экономии от его осуществления. Так, для организаций (предприятий), использующих ПИ.

Эффективность - выполнение требуемых функций при минимальных затратах ресурсов.

HIPO - технология (Hierarchica1 Input Process Output) - это многоуровневая дисциплина проектирования и документирования программ..

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Законы Республики Узбекистан

1.Закон Республики Узбекистан «Об электронном документообороте».// «Народное слово», 2004 г., 30 апреля.

2. Закон Республики Узбекистан «Об электронной коммерции»// «Народное слово», 2004 г., 30 апреля.

3.Закон Республики Узбекистан «Об информатизации»// «Народное слово», 2004 г., 11-февраля.

2. Указы и Постановления Президента Республики Узбекистан

4.Указ Президента Республики Узбекистан от 14 июня 2005 г. «О мерах по ускорению реализации приоритетных направлений в сфере углубления рыночных реформ и дальнейшей либерализации экономики».// «Народное слово», 2005 г., 15-июня.

5.Указ Президента Республики Узбекистан «О дальнейшем развитии компьютеризации и внедрении информационно-коммуникационных технологий».// «Народное слово», 2002 г., 1-июня.

3. Постановления Кабинета Министров Республики Узбекистан

6.Постановление Кабинета Министров Республики Узбекистан «О совершенствовании системы подготовки кадров в сфере информационных технологий».// «Народное слово», 2005 г., 3-июня.

7.Постановление Кабинета Министров Республики Узбекистан «О дальнейшем развитии компьютеризации и внедрении информационно-коммуникационных технологий». //«Народное слово», 2002 г., 8-июня.

4. Труды Президента Республики Узбекистан

8.Каримов И.А. «Эришилган ютуқларни мустаҳкамлаб, янги марралар сари изчил ҳаракат қилишимиз лозим». //«Халқ сўзи», 2006 й., 11-февраль.

9. Каримов И.А. Ўзбекистон буюк келажак сари. – Тошкент.: Ўзбекистон, 1998, 528 б.

10. Каримов И.А. Ўзбекистон XXI аср бўсағасида: ҳавфсизликка таҳдид, барқарорлик шартлари ва тараққиёт кафолатлари.–Т.:Ўзбекистон, 1997.

5. Нормативно-правовые документы Министерств Республики Узбекистан

11. Олий таълим. Меъёрий ҳужжатлар тўплами: /С.С.Ғуломов таҳрири остида; Тузувчилар: Б.Х.Рахимов, Ш.Д.Жонбоев ва бошқ. – Т.: «Шарқ», 2001. – 672 б.

12. Положение о порядке создания и использования электронных баз данных в государственных организациях республики. Утверждено Государственным комитетом Республики Узбекистан по науке и технике, от 29.20.1995.

13. Положение о порядке и правилах создания, внедрения и эксплуатации локальных, ведомственных, региональных и других информационно-вычислительных сетей на территории республики. Утверждено Государственным комитетом Республики Узбекистан по науке и технике, от 30.01.1995.

6. Учебники

14. Балдин К.В., Уткин В.Б. Информационные системы в экономике: Учебник.–М.:Издательско-торговая корпорация «Дашков и К»,2005.-395 с.

15. Информатика: Учебник. / Под ред. Н.В.Макаровой. 3-е перераб. изд. - М.: ФиС, 2004. –768с.

16. Информатика. Серия: Учебник / Под ред. П.П.Беленького. - Ростов Н/Д: Феникс, 2003. –448с.

17. Семокин И.Г., Шестоков А.П. Основы программирования. 2-е изд, стер. Учебник. - М.: Издательский центр Академия, 2003. – 432 с.

18. Ғуломов С. С., Шермухамедов А. Т., Бегалов Б.А. «Иқтисодий информатика». Тошкент. "Ўзбекистон", 1999. - 528 б.

19. Автоматизированные информационные технологии в экономике: Учебник/ Под ред. Проф. Г.А. Титоренко. – М.:Компьютер, ЮНИТИ, 1998. – 400 с.

20. Благодатских В.А., Енгибарян М.А., Ковалевская Е.В. Экономика, разработка и использование программного обеспечения ЭВМ. - М.: Финансы и статистика, 1995.-288 с.

7. Учебные пособия

21. С.С.Ғуломов, Б.А.Бегалов, Н.Р.Зайналов, Р.А.Дадабаева, А.Э.Давронов. Дастурлаш технологиялари. Олий ўқув юртлари учун ўқув қўлланма. - Тошкент.: ТДИУ, 2006 й., 192 б.

22. Ахборот технологиялари асосида дарсларни ташкил қилиш йўллари. Услубий қўлланма. Тошкент, ТДИУ. – 2005, 26 б.

23. Иқтисодий масалаларни шахсий компьютерларда ечиш технологиялари. Услубий қўлланма. Тошкент, ТДИУ. – 2005, 95 б.

24. Голиш Л.В. Технологии обучения на лекциях и семинарах в экономическом вузе. Учебное пособие. Ташкент, ТГЭУ. – 2005, 203 с.

25. Хошимова Д.П., Насридинова Ш.Т., Кобилов А.У. «Технология программирования». Сборник задач – Ташкент: ТГЭУ, 2005 г. 25 с.

26. Аляев Ю.А. и др. Практикум по алгоритмизации и программированию на язык Паскаль: Учеб. пособ. - М.: ФиС, 2004. – 528с

27. Вигерс Коре Разработка требований к программному обеспечению. - М.: Издательско-торговый дом "Русская редакция", 2004. – 576с.

28. СмирновА.А. Применение прикладного программного обеспечения.- Учебно-практическое пособие -М.: Моск. Гос. Ун.-т экон.,статистики и информатики,2004.-156с.

29. Информатика: Базовый курс / С.В.Симонович и др.- Санкт-Петербург: Питер, 2003. – 640 стр.

30. Могилев А.В., Пак Н.И., Хеннер Е.К. Информатика: Учеб. пособ. – М., 2003. – 816 с.

31. Зуев Е.А. Turbo Pascal. Практическое программирование – М.: «Издательство ПРИОР», 1999. 336 с.

32. Фаронов В.В. Турбо Паскаль 7.0. Начальный курс. Учебное пособие. – М.: «Нолидж», 1999. 616 с.

8. Монографии и научные статьи

33. Интернет-порталы: содержание и технологии: Сб. науч. Ст. Вып.1/Редкл.: А.Н. Тихонов (преп) и др.; ГНИИ ИТТ «Информика». – М.: Просвещение, 2003.

34. Титарев Д.Л., Титатев Л.Г., Феданов А.Н. Портал университета// В сб..: Интернет-порталы: содержание и технологии: Сб. науч. Ст. Вып.1/ Редкол.: А.Н. Тихонов (пред.) и др.; ГНИИ ИТТ «Информика». – М.: Просвещение, 2003.

35. Бегалов Б.А. Технология процессов формирования информационно-коммуникационного рынка. Монография. – Ташкент: Фан, 2000.

9. Докторские, кандидатские диссертации

36. Охунов Д.М. Исследование и разработка маркетинговых автоматизированных информационных систем предприятий. Диссертация на соискание кандидата экономических наук. Ташкент, ТГЭУ, 2005, 138 с.

37. Бегалов Б.А. Ахборот-коммуникациялар бозорининг шаклланиш ва ривожланиш тенденцияларини эконометрик моделлаштириш. Иктисод фанлари доктори илмий даражаси даъвогарлигига диссертация иши. Тошкент, ТДИУ, 2001, 330 б.

10. Сборник статей научно-практических конференции

38. «Ахборот-коммуникациялар технологиялари асосида электрон ўқув адабиётларини яратиш: тажриба, муаммо ва истиқболлар» мавзуидаги республика илмий-амалий анжумани, Тошкент, 2004 йил, 28-апрель.

39. «Етук мутахассисларни тайёрлашда замонавий педагогик технологиялар ва интерактив усулларнинг самарадорлиги» мавзусидаги II анжуман маърузалари тезислари. Тошкент, 2003 йил.

40. "Иктисодчи кадрлар тайёрлаш сифатини таъминлашда ахборот-коммуникациялар технологиялари", Республика илмий-амалий анжумани, Тошкент, 2003, 15-16 май.

41. Применение INTERNET в учебном процессе» мавзусидаги халқаро илмий-амалий конференция материаллари, 2002 йил, Москва-Тошкент.

11. Газеты и журналы

42. Информационные ресурсы России.
43. Информационные технологии.
44. Научно-техническая информация.
45. Ўзбекистон иқтисодий ахборотномаси.

12. Сборник статистических данных

46. Мониторинг развития информационно-коммуникационных технологий в Узбекистане. 2003 – 2005 гг. Ташкент. 70 с.

47. Деловая среда в Узбекистане глазами представителей малого бизнеса. Ташкент. 2002-2005 гг. 170 с.

13. Интернет сайты

48. http://www.borlpasc.narod.ru - сайт «Турбо Паскаль».
49. http://ad.cctpu.edu.ru - сайт кафедры «Информатики и проектирования систем» Томского Политехнического Университета
50. http://www.turbopascal.tk - сайт языка Турбо Паскаль.

51. http://www.rxlib.com - сайт о современных программных продуктах и компьютерных технологиях.

52. http://www.iite.ru - сайт ЮНЕСКО “Информационные технологии в образовании».

53. http://diamond.stup.ac.ru/ENG/F4/Direct/4.html - российский образовательный сайт «Новые информационные технологии в образовании».

14. Электронные учебники и учебные пособия виртуальной библиотеки

54. Б.А.Бегалов. Введение в базы данных. Электронное учебное пособие. Программисты: А.Бобожонов, У.Муслимов.Ташкент. ТГЭУ. 2004 г.

55. Ахборот тизимлари ва технологиялари. Электрон дарслик. Компьютер дастурчилари: С.Аҳмедов, А.Рапопорт. Тошкент. ТДИУ. 2003 й.

56. С.С.Ғуломов, А.Т.Шермуҳамедов, Б.А.Бегалов. Иқтисодий информатика. Электрон дарслик. Компьютер дастурчилари: О.Сидиқов, С.Аҳмедов. Тошкент. ТДИУ. 2002 й.