Интегрированные среды разработки и системы управления коллективным программированием

Рассмотрим группы инструментов, которые относятся к системам программирования:

- Инструментарий поддержки технологических процессов. Существуют инструменты, ориентированные на один процесс и применяемые только в нем, и универсальные (независимые от процесса).
- Инструментальные системы (инструментальные среды программирования, средства автоматизации разработки программ, интегрированные среды и репозитории[1] программного проекта).
- Инструментарий поддержки коллективной разработки.

1. Введение в системы программирования

1.1. Основные понятия и определения

Система программирования – часть базового программного обеспечения, поддерживающая процесс программирования. Системы программирования представляют собой единство средств статической (инструментальной) и динамической (исполнительной) поддержки.

К настоящему моменту сложилось представление о традиционном составе инструментальной системы (среды) программирования (IDE)[2], в который входят следующие программные инструменты и библиотеки:

- редактор для построения программ;
- транслятор для перевода программ с языка программирования на машинный язык;
- отладчик для проверочных запусков программ и исправления ошибок;
- библиотеки периода трансляции и периода исполнения;
- средства управления компиляцией и построением программного проекта;

[1] хранилище – место, где хранятся и поддерживаются какие-либо данные. Данные в репозитории хранятся в виде файлов, доступных для дальнейшего распространения по сети.

[2] IDE, Integrated Development Environment или Integrated Debugging Environment

• монитор, интегрирующий в себе вышеперечисленные средства и организующий функционирование системы программирования в целом.

Этот минимум, позволяющий достаточно эффективно вести процесс создания программ, был представлен еще в начале 70-х годов XX века. Однако настоящую популярность такой интегрированный набор инструментов приобрел в середине 1980-х годов благодаря компании Borland Inc. (http://www.borland.com).

Программный инструмент – это программа, предназначенная для поддержки разработки программных продуктов. Например, отладчик, облегчающий программисту выполнение отладки продукта.

Утилита – программа, обеспечивающая некоторые общие функции (например, копирование файлов, подготовку текстов, организацию перекрестных ссылок и т. п.).

Библиотеки процедур – наборы процедур различного назначения, упрощающие разработку приложений. Например, библиотека функций ввода-вывода или математических функций.

Программы предоставления дополнительных услуг – программы, предлагающие необязательные, но полезные функции (например, калькулятор или календарь).

1.2. История и эволюция

1.2.2. Основные этапы в формировании состава систем программирования.

Можно выделить три основных этапа в формировании состава систем программирования.

• Для начала 70-х годов XX века было характерно наличие "монолитных" систем программирования, состоящих из одной-двух больших программ, выполняющих множество разнообразных функций, начиная с редактирования и трансляции исходного модуля и заканчивая некоторыми сервисными возможностями.

• По мере развития отдельных функций программы происходит их выделение в самостоятельные компоненты, такие как текстовый редактор, отладчик, загрузчик.

• Следующим шагом явилось распространение инструментальной поддержки на все технологические процессы *жизненного цикла*[3] программ.

1.2.3. История развития системы программирования компании Borland Inc. на базе языка Pascal.

Практически все начало истории персональных компьютеров IBM PC связано с системой Turbo Pascal – наиболее популярной средой разработок.

За особые заслуги компании Borland Inc. в области систем программирования приведем историю развития такой системы на базе языка Pascal.

• 1983 г. - Создание Андерсом Хейлсбергом программы Turbo Pascal для операционной системы CP/M.

• 1984 г. - Перенос Turbo Pascal в MS-DOS.

• 1990 г. - Представление Turbo Pascal для Windows.

• 1992 г. - Последний выпуск Borland Pascal версии 7.0.

• 1995 г. - Появление Delphi - доведенной до логического совершенства и кульминационного пика интегрированной среды Turbo Pascal.

1.3. Классификация систем программирования

1.3.1. Классификация по ориентации на поддержку процессов.

Начнем с основной классификации систем программирования:

• Инструментарий поддержки технологических процессов:

o инструментарий, поддерживающий определенный технологический процесс (процесс-ориентированный инструментарий);

o универсальный инструментарий, который можно использовать в нескольких технологических процессах (процесс-независимый инструментарий).

• Инструментальные системы разработки и сопровождения программных продуктов:

o инструментальные среды программирования;

o средства автоматизации разработки программ (CASE-средства[4]);

[3] Жизненный цикл ПО – период времени, который начинается с момента принятия решения о необходимости создания программы и заканчивается в момент ее полного изъятия из эксплуатации.

- интегрированные среды и репозитории проекта.

- Инструментарий поддержки коллективной разработки.

1.3.2. Функциональная классификация

Еще одну классификацию систем программирования можно провести, отражая их функциональную ориентацию в технологическом процессе. Эта классификация относится к группе инструментария поддержки технологических процессов.

Рассмотрим четыре группы инструментов:

- Редакторы, поддерживающие конструирование (формирование) многочисленных программных документов.
- Анализаторы, производящие:
  - статическую обработку документов, осуществляя различные виды их контроля, выявление определенных свойств и накопление статистических данных;
  - динамический анализ программ.
- Преобразователи, позволяющие:
  - автоматически приводить документы к другой форме представления;
  - синтезировать какой-либо документ из отдельных частей.
- Инструменты поддержки процесса выполнения программы, позволяющие исполнять:
  - описания процессов или отдельных их частей, представленных в виде, отличном от машинного кода;
  - машинный код с дополнительными возможностями его интерпретации.

1.3.3. Классификация по категориям

Классификация систем программирования по категориям определяет уровень их интегрированности по выполняемым функциям и включает:

- вспомогательные программы (tools) – пакеты, решающие небольшую автономную задачу;

---

[4] CASE (англ. Computer-Aided Software Engineering) – совокупность методов и средств проектирования информационных систем с интегрированными автоматизированными инструментами, которые могут быть использованы в процессе разработки программного обеспечения. В функции CASE входят средства анализа, проектирования и программирования.

• пакеты разработчика (toolkit) – совокупность интегрированных программных средств, обеспечивающих помощь для одного из классов программных задач;

• инструментальные средства (workbench) – интеграция программных средств, которые поддерживают системный анализ, проектирование и разработку программного обеспечения. При этом используется репозиторий, содержащий всю техническую и управляющую информацию о проекте.

1.3.4. Классификация по предоставляемому интерфейсу

Системы программирования предоставляют два ярко выраженных типа программного интерфейса:

• Интерфейс командной строки (Command Line Interface – CLI). Это традиционный интерфейс систем программирования в операционной системе Unix. В современных диалектах Unix практически все инструменты имеют также надстройку с графическим пользовательским интерфейсом.

• Графический пользовательский интерфейс (Graphic User Interface – GUI). Этот интерфейс традиционен для систем программирования в Windows.

1.4. Проблемы и перспективы развития

Характеризуя современное развитие систем программирования, можно отметить сложное взаимоотношение между дифференциацией и интеграцией инструментальных средств.

Развитие и совершенствование отдельного инструментария вело к росту дифференциации, отделения и образования новых инструментов. Жесткая архитектура традиционных систем программирования часто оказывалась не в состоянии обеспечить их последующую интеграцию в гибкую согласованную систему.

Отечественные системы программирования (и трансляторы) характеризовалась в большей степени исследовательским уровнем, чем промышленным качеством. Отсутствие реального рынка и конкуренции программных продуктов привело к тому, что даже широко использовавшиеся трансляторы не соответствовали международным требованиям на программный продукт.

Воздействие сети Интернет на развитие систем программирования также велико. Перспективными направлениями развития систем программирования в Интернете являются:

- проектирование и реализация инструментария, обеспечивающего кросс-платформенную разработку приложений на языках Java и C++ в средах ОС Unix и Windows NT;
- разработка инструментария для эффективного написания программ, поддерживающих распределенные вычисления;
- разработка высокопроизводительных систем программирования;
- разработка средств отладки Интернет-приложений;
- разработка инструментария с высоким уровнем абстракции для проектирования интерфейсов;
- разработка систем коллективного программирования.

Интернет содержит достаточное количество ресурсов и для образовательных целей. Особый интерес для ВУЗов представляют свободно распространяемые программы. Многие учебные заведения используют язык Pascal для первоначального обучения программированию. В Интернете можно найти свободно распространяемые компиляторы с языка Pascal:

- Free Pascal (http://www.freepascal.org/);
- GNU Pascal (http://agnes.dida.physik.uni-essen.de/~gnu-pascal).

2. Процесс-ориентированный инструментарий

Всякий инструмент будет наилучшим образом удовлетворять своему назначению, если он предназначен для исполнения одной работы, а не многих. /Аристотель/.

Теперь подробно перечислим инструменты, относя их к одной из следующих групп.

2.1. Возникновение и исследование идеи

Для поддержки данного процесса предлагается использовать два типа инструментов.

- Поисковые информационные системы. Эти системы по сделанному запросу выдают ссылки на найденные документы. Подробно такие системы будут рассмотрены в разд. 5.3.6. Наиболее известными поисковыми системами являются:
  - поисковые системы общего назначения (например, Яndex (http://www.yandex.ru/) и Google (http://www.google.com/));

- тематические каталоги Интернета (например, Yahoo (http://www.yahoo.com/));
- базы данных патентов в Интернете (например, Questel-Orbit (http://www.questel.orbit.com/));
- индексы цитирования научных статей (например, Researchlndex (http://www.citeseer.nj.nec.com/cs)).

• Системы поиска решений. Эти системы представляют собой структурированный набор разнообразного рода эвристических приемов и методов (http://www.triz.minsk.by/index0.htm). Они объединяют предшествующий опыт различных авторов и коллективов и должны способствовать увеличению количества рождающихся идей.

2.2. Управление

Инструментарий поддержки процесса управления разделим на три группы:

• Системы управления проектами;
• Организационные средства;
• Средства оценки качества.

2.2.1. Системы управления проектами

Перечислим группы средств, предназначенных для управления проектами:

• средства описания комплекса работ проекта, связей между отдельными работами и их временных характеристик;
• средства поддержки информации о ресурсах и затратах по проекту и его отдельным работам;
• средства контроля над ходом выполнения проекта;
• графические средства представления структуры проекта и средства создания отчетов по проекту.

Система должна позволять динамически пересматривать планы и сроки работ в ходе выполнения проекта. Возможно, система будет иметь средства отслеживания назначений сотрудников.

Перечислим ряд наиболее интересных[5] систем:

• Microsoft Project (www.microsoft.com/office/project/default.htm) – самая распространенная в мире на сегодняшний день система, которая может быть рекомендована для планирования несложных

[5] http://www.aproject.ru/tools/tools.html

проектов. Пакет достаточно прост, имеет традиционные средства помощи и удобные инструменты создания отчетов;

- система Time Line (http://www.timeline.com) – также предназначена для несложных проектов. Содержит минимально необходимые функции управления проектами;
- Primavera (http://www.primavera.com) – более мощная система, применяемая для управления средними и крупными проектами.

2.2.2. Организационные средства

К организационным средствам относятся:

- электронная почта;
- электронный календарь;
- интранет.

Электронная почта может играть важную организационную роль в процессе управления проектом. Достоинства такого подхода в том, что вся информация пересылается по почте и остается "в истории". Во многих компаниях принято подкреплять почтой решения, достигнутые в результате устной договоренности. В крупных компьютерных компаниях инженер, как правило, получает по электронной почте около 100 рабочих писем в день, а менеджер от 300 до 500. В настоящее время набор почтовых программ входит в состав большинства операционных систем.

Электронный календарь представляет собой информационный пакет, в который заносятся пометки о будущих событиях типа совещаний, семинаров, крайних сроков важнейших событий. Информация может быть занесена как самостоятельно исполнителем, так и другими людьми, которым разрешен доступ (например, секретарем). Электронные календари реализованы в графических оболочках большинства современных операционных систем.

Интранет это корпоративные информационные системы, построенные на принципах, заимствованных из сети Интернет. По сути, интранет – это перенос хорошо известных сетевых технологий во внутрикорпоративные сети. Внутренняя сеть позволяет создать целостный портрет организации и обрести радикально новую систему внутренних коммуникаций и доступа к информационным ресурсам. Интранет может быть значительно "умнее" Интернета, потому что он контролируем, и в него можно инвестировать значи-

тельно больше знаний и средств. В качестве серверов интранета можно использовать хорошо известные серверы Интернета, например Apache (http://www.apache.org/).

2.2.3. Средства оценки качества

Для сравнения качества программных продуктов применяются количественные методы оценки. Среди программ оценки качества отметим Metricate компании Software Productivity Centre (http://www.spc.ca), которая анализирует все аспекты деятельности компаний по производству программного обеспечения. Это – эффективность технологических процессов, качество программного кода, уровень управления проектами, стоимость выполнения различных этапов, производительность получаемой системы, продуктивность труда разработчиков и качество готовых изделий.

2.3. Анализ требований и проектирование

2.3.1. Системы на основе структурной методологии

Для анализа требований и проектирования на основе структурной методологии могут быть применены следующие системы:

- Silverrun ModelSphere (компании magma solutions GmbH (http://www.spc.ca) – поддерживает методы DATARUN, Гейна-Сарсона, Йордона, Мартина и др.;
- Oracle Designer (компании Oracle (http://www.oracle.com) – поддерживает CASE*Method Баркера;
- ERwin (компании Computer Associates International, Inc. (http://www.cai.com) – поддерживает диаграммы функционального моделирования.
- CASE-Аналитик (компании Эйтекс) – поддерживающая подход Гейна-Сарсона.

2.3.2. Системы на основе объектно-ориентированной методологии

Укажем наиболее известную систему для анализа требований и проектирования на основе объектно-ориентированной методологии:

- Rational Rose (компании Rational Software Corporation (http://www.rational.com);

Основные достоинства и возможности системы Rational Rose:

- система прошла достаточно долгий путь развития и совершенствования. Она поддерживает язык UML[6] и ряд более ранних языковых нотаций;
- система реализована на обеих наиболее распространенных операционных системах – Unix и Windows;
- система имеет три основные модификации. Пользователь имеет возможность выбрать одну из них, устраивающую его по финансовым соображениям:
  - Enterprise – с возможностью генерации кода на языках Java, Visual BASIC, Visual C++;
  - Professional – возможность генерации кода на одном из перечисленных языков;
  - Modeler – без языковой поддержки;
- система поддерживает восстановление спецификаций из кода[7];
- система поддерживает генерацию проектной документации.

2.4. Программирование (реализация)

2.4.1. Трансляторы

Транслятор – программный инструмент, предназначенный для перевода (трансляции) программ с одного формального языка на другой. Существует несколько основных видов трансляторов:

- компиляторы;
- *де*компиляторы;
- интерпретаторы.

Обратите внимание на то, что многие другие программные инструменты могут рассматриваться как языковые трансляторы, например:

- текстовые редакторы (транслируют текст и специфические команды форматирования в некоторый образ);
- процессоры запросов (транслируют высокоуровневый язык запросов в реляционный язык (например, SQL);

---

[6] UML (Unified Modeling Language – унифицированный язык моделирования) – язык графического описания для объектного моделирования в области разработки программного обеспечения.

[7] Спецификация – техническое задание, документ, выдаваемый программистам, с указаниями о том, как должна выглядеть и работать система.

• программы доказательства теорем (преобразуют спецификацию теоремы в некоторый метаязык).

2.4.2. Компиляторы

Исходная программа преобразуется компонентами компилятора в ряд промежуточных представлений и, в конечном итоге, в программу на языке назначения. В течение этих преобразований компоненты работают с общими символьными таблицами и при необходимости обращаются к обработчику ошибок.

Для построения компилятора необходимо однозначное и точное задание входного и выходного языков. Такое задание предполагает определение правил построения допустимых конструкций (выражений) языка. Множество таких правил называют **синтаксисом** языка. Кроме того, задание должно включать описание назначения и смысла каждой конструкции языка. Такое описание называют **семантикой** языка.

Далее мы рассмотрим основные программы, входящих в состав компилятора.

Драйвер (диспетчер, монитор) – это программа, последовательно вызывающая все остальные компоненты компилятора.

Препроцессор – программа, которая выполняет модификацию данных с целью их подготовки для ввода в другую программу. Модификация может заключаться в простом переформатировании или применении макрорасширений.

Анализатор – компонент компилятора, осуществляющий последовательно:

- лексический анализ, на котором входная программа, представляющая собой поток символов, разбивается на лексемы - конструкции (слова) языка;
- синтаксический анализ, в процессе которого происходит разбор структуры программы;
- контекстный анализ, во время которого выявляются зависимости между частями программы, включая анализ типов, областей видимости, соответствие параметров и т.п.

Оптимизатор – компонент, выполняющий преобразования программы (как правило, на основе промежуточного представления) с целью улучшения ее характеристик (оптимизации) по одному или ряду параметров. Как правило, оптимизацию проводят по времени исполнения с целью получения более эффективной про-

граммы или по пространству – месту, которое файл с откомпилированной программой будет занимать.

Кодогенератор – компонент, завершающий компиляцию и порождающий на основании промежуточного представления программы либо ассемблер, либо объектный код.

2.4.3. Системы генерации трансляторов

С точки зрения существующих алгоритмов синтаксического анализа, все языки и порождающие их **грамматики**[8] делятся на два класса LL(n) и LR(n)[9]. Этот теоретический раскол в классе грамматик, распознаваемых анализаторами, построенными по принципу снизу вверх, и анализаторами, реализующими принцип сверху вниз, естественным образом наложил свой отпечаток и на практические реализации систем. В результате существуют системы с входными языками, специфицируемыми LR(n) и LL(n) грамматиками соответственно. Исторически первой системой, реализующей синтаксически управляемую трансляцию, была система **yacc**[10], входной язык которой принадлежит к классу LR(1) или, более точно, LALR(1). Система **yacc**, разработанная в 1972 году в рамках проекта Unix, успешно используется до сих пор, что, конечно, свидетельствует в пользу простоты и мощности идей, положенных в ее основу.

Однако, несмотря на простоту архитектуры этой системы, можно отметить и некоторую многословность, появляющуюся при попытке спецификации сложной грамматики на входном языке системы, а также узкую ориентированность на язык C, как язык реализации построенного по грамматике транслятора. Эти и некоторые другие ограничения и послужили основной движущей силой, давшей жизнь целому семейству систем, имеющих своим прототипом оригинальный **yacc**.

---

[8] Математические модели, использующие представление текстов в виде цепочек символов, называют формальными языками и грамматиками.

[9] Контекстно-свободные грамматики находят большое применение в информатике. Ими задаётся грамматическая структура большинства языков программирования, структурированных данных и т.д. Существуют следующие типы КС-грамматик: LL, LALR(1), LR, SLR(1).

[10] **yacc** – компьютерная программа, служащая стандартным генератором синтаксических анализаторов (парсеров) в Unix-системах. Название является акронимом «Yet Another Compiler Compiler» («ещё один компилятор компиляторов»). Yacc генерирует парсер на основе аналитической грамматики, описанной в нотации BNF (форма Бэкуса-Наура). На выходе **yacc** выдаётся код парсера на языке программирования Си.

Естественное желание разработчиков пользоваться давно написанными средствами, прошедшими не один год тестовых испытаний в реальных проектах, отрицательно сказалось на количестве систем, транслирующих входные языки класса LL(n), т.е. реализующих двойственный подход к проблеме. Тем не менее, в рамках некоторых учебных проектов был создан ряд таких систем.

2.4.4. Системы анализа корректности программного кода

Системы такого типа очень близки синтаксическому анализатору и являются средствами статического анализа программ. Обычно они разрабатываются для языков со слабым контролем типов и выполняют гораздо большее количество тщательных проверок, чем стандартные анализаторы. Это выполнение максимального количества типовых проверок и проверка соответствия параметров. Системы анализа корректности программного кода обычно выдают большое количество дополнительных предупреждений, касающихся всех подозрительных мест в программе. Ряд инструментов анализирует метрики сложности (метрики размера программ, метрики сложности потока управления программ[11] и метрики сложности потока данных программ). Во многих компаниях проверка исходных текстов программных продуктов системами анализа корректности программного кода является обязательной.

Примеры систем:

- lint (компании Sun Microsystems, Inc. (http://www.sun.com/) для языка программирования C;
- PC-lint и FlexeLint (компании Gimpel Software (http://www.gimpel.com/) анализируют программный код на языках C и C++.

2.4.5. Интерпретаторы

Интерпретатор – программа, осуществляющая непосредственное исполнение текста исходной программы пошаговым образом. Интерпретатор одновременно и транслирует и выполняет заданную программу. Существуют языки программирования, для которых интерпретация программ, написанных на них, предпочтительнее компиляции. Это языки программирования с конструкциями, поз-

11 Поток управления – множество всех возможных путей исполнения программы (представленное в виде графа).

воляющими создавать новые *под*программы или модифицировать существующие динамически, во время выполнения программы.

Примером интерпретируемого языка программирования является Lisp.

2.4.6. Декомпиляторы

Декомпилятор – программа, позволяющая по программе на языке низкого уровня (обычно это коды некоторой машины) получить программу на высокоуровневом языке, в некотором смысле ей эквивалентную. Под эквивалентностью чаще всего понимают эквивалентность внешних проявлений работы исходной программы и декомпилированной.

При решении разнообразных практических проблем в области программирования зачастую бывает полезно декомпилировать имеющиеся программы, исходные тексты которых недоступны. Эта задача является весьма сложной и пока что может быть решена лишь в частных случаях, хотя методы, применяемые для всех частных случаев, имеют много общего. Одной из важных составных частей задачи декомпиляции является анализ потоков управления. Как правило, эта часть осуществляется при помощи исследования графа управления, полученного на этапе предварительного анализа программы на низкоуровневом языке. Дуги этого графа соответствуют условным и безусловным переходам между базовыми блоками программы, и он не содержит никакой информации об управляющих конструкциях высокого уровня, из которых должна состоять декомпилированная программа. Восстановление этих структур и есть основная задача анализа потоков управления при декомпиляции. Разумеется, ее специфика определяется языком высокого уровня, на котором должна быть написана итоговая программа, т.к. разные языки, даже традиционно считаемые близкими и относящимися к одному семейству, имеют разные наборы допустимых управляющих конструкций.

Процесс, который необходимо выполнить при решении этой задачи, называют структуризацией графа управления. При структуризации каждую вершину графа управления надо сделать частью какой-либо высокоуровневой управляющей конструкции. Эти конструкции следует выбирать таким образом, чтобы не изменился порядок переходов управления между базовыми блоками.

Теория анализа потоков управления появилась давно, т.к. анализ активно применялся в компиляторах практически с момента их появления. Однако анализ потоков управления при компиляции имеет свою специфику – он используется в основном для целей оптимизации. Большинство алгоритмов анализа потоков управления, которые ориентированы на декомпиляцию, ставят своей целью уменьшение количества goto-выражений посредством введения новых логических переменных, дублирования кода (code replication) или использования конструкций высокого уровня, недоступных в большинстве широко используемых языков, таких как Pascal или C++.

Разработаны алгоритмы, использующие минимальный набор инструкций высокого уровня: циклы типов while и do...while, if- и case-выражения. Управление выполнением объемлющих конструкций из вложенных не допускается. Причем эти алгоритмы используют оператор goto лишь тогда, когда граф управления не может быть структурирован никаким иным способом в пределах перечисленных конструкций.

Анализ практических достижений в области декомпиляции языка Java выполнил Дэйв Дайер[12]. Он сравнил между собой три наиболее известных к середине 1997 года декомпилятора: DejaVu, Mocha и WingDis. Дайер предложил систему тестирования декомпиляторов, основанную на интересной классификации допускаемых ими ошибок. В соответствии с этой системой все ошибки делятся на шесть категорий. "Тяжесть" ошибки возрастает с номером категории.

Ошибки, в результате которых декомпилированная программа перестает собираться, но которые могут быть легко исправлены (например, отсутствие явного приведения типов там, где оно должно быть), относятся к первой, самой легкой категории

Генерация правильного, но нечитаемого текста на Java считается ошибкой третьей категории.

Заметим, что некоторые ошибки, проявляясь у одних декомпиляторов, полностью отсутствуют у других. Это свидетельствует о жизнеспособности такой классификации – она позволяет определить области, в которых одни декомпиляторы имеют преимущество над другими.

[12] www.javaworld.com

В настоящее время наиболее известны декомпиляторы Mocha и Crema, вхощие в состав программного продукта Jbuilder компании Borland Inc.

2.4.7. Усложнители декомпиляции

Практические успехи в области декомпиляции языков программирования (и особенно языка Java) оказались столь велики, что в последнее время были проведены значительные исследования в области защиты программ от декомпиляции.

Обфускация (запутывание кода) – приведение исходного текста или исполняемого кода программы к виду, сохраняющему ее функциональность, но затрудняющему анализ, понимание алгоритмов работы и модификацию при декомпиляции.

«Запутывание» кода может осуществляться на уровне алгоритма, исходного текста и/или ассемблерного текста. Для создания запутанного ассемблерного текста могут использоваться специализированные компиляторы, использующие неочевидные или недокументированные возможности среды исполнения программы.

Существует также специальный класс программ (шифраторов, обфускаторов), усложняющих декомпиляцию. Такие программы используют два основных подхода:

- Общее (универсальное) усложнение.
- Целенаправленное усложнение (направленное против конкретного декомпилятора).

Основная идея усложнителей заключается в ухудшении кода. Усложненный код после декомпиляции должен быть непонятен и нечитаем программистом.

2.4.8. Системы управления компиляцией и построением программ

Системы управления компиляцией и построением программ в той или иной форме присутствуют во многих инструментальных системах, а также могут использоваться как явные утилиты, работающие с файлами проекта. Фактически эти системы являются генераторами команд, т.е., используя файл описаний, они создают последовательность команд для обработки командным интерпретатором.

Примеры таких систем:

- make (компании Sun Microsystems Inc. (http://www.sun.com) входит в состав операционной системы Solaris;
- GNU Make (организации Free Software Foundations (http://www.gnu.org/software/raake/make.html);
- Jmk – Make in Java (разработка http://jmk.sourceforge.net).

Основной алгоритм перечисленных систем (далее будем называть их **make**) таков. На основе совокупности инструкций, согласно которым должна производиться сборка и построение проекта, они проверяют временн***ы***е метки файлов и, при необходимости, перекомпилируют файл. В результате существенно экономится время сборки проекта.

2.5. Тестирование и отладка

Заблуждения, заключающие в себе некоторую долю правды, самые опасные. /Адам Смит/

Мы будем изучать инструментарий, применяемый как на этапе тестирования, так и на этапе отладки. Очень часто оказывается, что один и тот же инструмент может указать на факт наличия ошибки, одновременно локализовать место ее возникновения и указать причину.

2.5.1. Тестовые мониторы

Тестовый монитор состоит из трех основных компонентов:

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

В результате запуска тестового монитора в тестовом пространстве начинают исполняться заданные *тест-сьюты*[13] из тестовой базы. На основе результатов тестирования можно сделать выводы об изменении качества программного продукта.

2.5.2. Средства отслеживания тестового покрытия

[13] Test-Suite is a set of tools used for testing software.

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

Важную роль здесь играет понятие линейного участка - фрагмента программы, на протяжении которого нет передачи управления. Если выполняется первый оператор такого участка, то будут выполнены и все остальные.

Примеры систем:

- Rational PureCoverage (компании Rational Software Corporation (http://www.rational.com/);
- Java Test Coverage and Instrumentation Toolkits (компании Glen McCluskey & Associates LLC (http://www.glenmccl.com/).

2.5.3. Средства динамического построения профиля программы (профайлеры).

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

- Компиляция и сборка программы с включенным режимом добавления кода с целью генерации информации для построения профиля.
- Запуск программы с целью получения информации, необходимой для построения профиля.
- Запуск профайлера для анализа полученной информации.

В самом простейшем случае результатом работы *профайлера* будет исходный текст программы, где для каждого линейного участка указана цифра: сколько раз этот исходный участок был исполнен. Следовательно, наиболее часто исполняемые участки надо максимально оптимизировать с целью ускорения работы всей программы.

Примеры профайлеров:

- prof (компании Sun Microsystems Inc. (http://www.sun.com/));
- gnuprof (организации Free Software Foundations (http://www.gnu.org/));

• ProDelphi (авторская разработка Гельмута Адольфа (Helmuth Adolph) (http://www.prodelphi.de/)).

2.5.4. Системы построения срезов программы

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

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

Причины возникновения данного направления очевидны. При работе с исходным текстом программы часто необходимо проследить формирование значения некоторой переменной. Это особенно актуально при отладке и сопровождении программ. В этом случае получение среза приносит существенное облегчение работы, поскольку резко снижается объем исследуемого кода. Предположим, что во время тестирования мы установили, что некоторая переменная в некоторой позиции содержит неверное значение. Получив соответствующий срез, мы локализуем проблему. В дополнение к сказанному, срезы позволяют разбить большую программу на небольшие компоненты. Это способствует более простому пониманию смысла программы, что очень актуально при сопровождении программных продуктов. Впоследствии часть этих компонентов может быть использована при создании новой программы.

В начальном понимании понятие срез (*статический срез*) учитывает все возможные пути исполнения программы.

Однако обычно на практике мы имеем дело с частным случаем, в котором программа работает некорректно, и, естественно, нас интересует причина происходящего. Таким образом, вместо того, чтобы отслеживать все множество входных данных, мы будем рассматривать срез, который сохранит свойства программы для определенного входа. Такой тип среза был обозначен термином *динамический* срез.

Большинство существующих методов вычисления динамического среза основано на так называемом обратном анализе, когда после исполнения программы записывается ее трассировка, и затем алгоритм отслеживает эту трассировку в обратном направлении для получения зависимостей по данным и по контролю, которые в свою

очередь используются для вычисления динамического среза. Существуют методы, вычисляющие динамический срез в процессе исполнения программы и не записывающие трассировку.

В существующих алгоритмах основной целью вычисления динамического среза является определение той части кода, которая влияет на значение интересующей переменной, путем выявления зависимостей по данным и по контролю в трассировке. При определении динамического среза важно найти не только части, влияющие на процесс вычисления интересующей переменной, но также и части, не влияющие на него. Чем больше таких не влияющих частей будет найдено, тем меньший динамический срез может быть получен.

2.5.5. Отладчики

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

Кроме классических отладчиков существуют еще несколько групп инструментов.

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

Средства отладки многопоточных и параллельных приложений. Это те приложения, которые в некоторой точке могут разветвиться для выполнения параллельной работы, потом попадают в точку ожидания и опять выполняются последовательно. Примеры – средства компании Sun Microsystems Inc. (http://www.sun.com):

- thd (Thread Analiser) – нитевой анализатор, который исполняется для того, чтобы отслеживать многопоточные приложения;
- locklint – анализатор корректности взаимных исключений;
- looptool – анализатор распараллеливания программ.

2.5.6. Системы отслеживания проблем (ошибок)

Системы отслеживания проблем обычно представляют собой интерфейс к обыкновенной базе данных. Наиболее сложный вопрос при разработке таких систем – отбор информации, которая связана с проблемой (ошибкой).

Вот одна из возможных структур такой информации:

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

Примерами систем отслеживания проблем являются:

- Bugzilla, применяемая для сбора информации об ошибках гипертекстового браузера Mozilla (http://bugzilla.mozilla.org/);

▪ Open Source Web Browser (авторская разработка Адама Сигеля (Adam Siegel) (http://www.asgtools.com/products/bt.html).

2.6. Ввод в действие

Основная группа систем данного процесса – это системы инсталляции. Укажем самые популярные инсталляторы для операционной системы Windows

▪ MindVision's Installer VISE – мощнейший профессиональный инсталлятор, используемый даже крупными корпорациями;

▪ WISE – известен самым маленьким размером получающегося дистрибутива. Имеет богатый язык скриптов;

▪ InstallShleld Professional/Express – старейший продукт, обладающий всеми необходимыми возможностями для создания сколь угодно сложных сценариев установки;

▪ CreateInstall – простой, легкий в освоении и компактный инсталлятор.

2.7. Сопровождение

Можно выделить следующие основные группы инструментов, используемых при сопровождении программ:

▪ связанные с технической стороной сопровождения (трансляторы, отладчики, системы управления версиями текстов и т. п.). Поскольку сопровождение представляет мини–модель жизненного цикла, то инструменты, применяемые во всех процессах жизненного цикла, могут понадобиться и здесь;

▪ связанные с анализом и пониманием кода. Это специальные средства реинжиниринга[14] и обратной инженерии. Также сюда мы относим средства восстановления спецификаций, конверторы и другие системы;

▪ связанные с организационной стороной сопровождения. Эти средства дают возможность выполнения административных функций (например, ведения базы данных ошибок) сопровождающему программный продукт.

2.8. Завершение эксплуатации

[14] Реинжиниринг программного обеспечения – процесс создания новой функциональности или устранения ошибок, путём революционного изменения, но используя уже имеющееся в эксплуатации программное обеспечение.

Основная группа систем данного процесса – это системы деинсталляции. Такие системы выполняют действие, обратное действию инсталляции, и, как правило, входят в инсталляционный пакет.

3. Универсальный инструментарий

Напомним, что в этом разделе под универсальностью инструментария будем понимать возможность его применения в нескольких процессах.

3.1. Инструменты работы с текстом

Существующие на сегодняшний момент системы можно подразделить на несколько классов. Так, некоторые из них используются как простые средства для обработки текстов, другие же представляют собой мощные автоматизированные программные комплексы. В этом разделе перечислены наиболее значимые из них.

3.1.1. Средства, базирующиеся на регулярных выражениях

Развитыми средствами определения шаблонов и осуществления поиска являются средства семейства grep в операционной системе Unix. Как grep, так и egrep могут искать участки текста, используя регулярные выражения[15].

Регулярные выражения произвели прорыв в электронной обработке текстов в конце XX века. Набор утилит (включая редактор sed и фильтр grep), поставляемых в дистрибутивах UNIX, одним из первых способствовал популяризации регулярных выражений для обработки текстов. Многие современные языки программирования имеют встроенную поддержку регулярных выражений. Среди них Perl, Java, PHP, JavaScript, языки платформы .NET Framework и др.

Эти средства используют текстовый поиск и в них не заложена семантика входного текста. А значит с их помощью нельзя отличить поиск по регулярным выражениям в произвольном тексте от поиска в исходном тексте программ. Можно провести аналогии с использованием архивирующих систем: те из них, которые предназначены для архивации текста, будут обладать гораздо меньшей

15

эффективностью для архивации потоков звука и видео, и наоборот. Поэтому такие средства обладают рядом ограничений:

- Написание конкретного шаблона для поиска в исходном тексте может быть затруднительным, а иногда и невозможным. Например, пусть требуется локализовать два последовательных оператора цикла: цикл типа for, следующий за циклом типа while, причем оба с произвольными условиями и блоками операторов, на одном и том же уровне вложенности. Возможная спецификация для grep может быть такой: **.***while**.***for. Однако такая спецификация grep отыщет не только последовательные операторы цикла, но и вложенные. Дополнительно будут найдены такие последовательные операторы, которые непосредственно не следуют друг за другом. Кроме того, оба требуемых ключевых слова могут оказаться в комментарии.
- Вторая проблема, которая может возникнуть при использовании средств семейства grep, связана со специальными символами. Обычно эти средства воспринимают символ "новая строка" как разделитель даже тогда, когда мы ищем выражение в многострочном контексте. Это особенно мешает поиску в исходном тексте, поскольку зачастую предмет поиска охватывает несколько строк. К тому же необходимость постоянно учитывать разделители (пробелы, символы "новая строка", символы табуляции) затрудняет составление регулярных выражений.

3.1.2. Средства поиска различий

Средства поиска различий предназначены для обнаружения разницы между файлами. Такая необходимость возникает достаточно часто. Например, для сравнения старой и новой версии одного файла или сравнения изменений, внесенными двумя разными людьми на основе одной исходной версии. Стандартным результатом работы таких средств являются строки файлов, которые нужно изменить для того, чтобы привести файлы в соответствие друг с другом. В большинстве случаев находится минимальный достаточный набор различий.

Примеры таких систем:

- diff (компании Sun Microsystems Inc. (http://www.sun.com) входит в состав операционной системы Solaris;
- GNU diff (организации Free Software Foundations) представляет собой совокупность нескольких утилит:

- diff - основная утилита для нахождения различий между двумя файлами;
- cmp - утилита, выдающая информацию о номерах строк, в которых в файлах есть различия;
- diff3 - утилита, находящая различия между тремя файлами;
- sdiff - утилита для интерактивного объединения двух файлов в один.

3.1.3. Средства поиска на основе шаблонов

Средства поиска на основе шаблонов в качестве входных данных получают исходный код на языке программирования и запрос на языке шаблонов (образцов). На выходе выдается информация о том, какие строки исходного программного кода соответствуют спецификации шаблона.

В запросе можно также указать поиск последовательности операторов. В таком случае запрос понимается как поиск указанной последовательности операторов в любом месте исследуемого кода. Вся последовательность воспринимается как один запрос.

Для того чтобы специфицировать запрос к исходному коду, используется *язык образцов*. Язык образцов для спецификации запроса является расширением базового языка. Расширения включают набор специальных символов, которые используются для замены синтаксических сущностей базового языка. Без учета дополнительных расширений запрос может состоять из участка программы для осуществления конкретного поиска по исходному тексту.

Запросы могут быть следующих типов:

- поиск декларации (как локальной, так и глобальной);
- поиск определения функции, спецификация кода которой описана в запросе;
- поиск группы операторов в произвольном участке кода.

3.1.4. Обозреватели и базы данных программ

Обозреватели генерируют базу данных программы, куда в качестве сущностей включаются имена файлов, имена функций, переменные и другие сущности программы с определенными отношениями между ними. Чтобы получить информацию о том, какая функция вызывает определенную процедуру или глобальные данные, используемые какой-то определенной функцией, могут быть сделаны запросы к этой базе данных программы.

Хотя такие средства очень мощны и полезны, все же они поддерживают ограниченный диапазон запросов, которые базируются на концепции "сущность-отношение-атрибут". При таком рассмотрении отсутствует структурность, присущая исходному программному коду.

3.1.5. Средства обнаружения плагиата

Инструменты для определения плагиата в программах в основном базируются на метриках Холстеда[16].

Другая категория таких инструментов использует статическое дерево исполнения (граф потока управления) программы, чтобы составить "отпечаток пальцев" программы. Далее дерево исполнения анализируется, и на основе анализа делаются выводы о том, была ли программа заимствована. Основное ограничение таких систем состоит в том, что сравнение программ в них зависит от статистической информации.

3.1.6. Текстовые редакторы

Текстовый редактор – программа, позволяющая вводить в файл символьную информацию и модифицировать ее. Можно выделить несколько разновидностей текстовых редакторов.

- Строковые редакторы, позволяющие смотреть и редактировать текст только на одной строке. При этом текст воспринимается как последовательность строк, разделенных символом "конец строки"[17].
- Потоковые редакторы, позволяющие воспринимать текст в виде потока символов. При этом признаки конца строки также являются символами.
- Экранные редакторы, позволяющие перемещать по тексту окно, в котором можно передвигать курсор и устанавливать его в нужные позиции. Многие экранные текстовые редакторы позволяют видеть на экране текст в том виде, в котором он будет напечатан (концепция What You See Is What You Get – WYSIWYG).

---

[16] Метрика размера программы. Метрика Холстеда относится к метрикам, вычисляемым на основании анализа числа строк и синтаксических элементов исходного кода программы.

[17] Системы, основанные на ASCII или совместимом наборе символов, используют или LF (перевод строки, 0x0A), или CR (возврат каретки, 0x0D) по отдельности, или последовательность CR+LF

Сейчас распространены универсальные текстовые редакторы, которые избыточны для большинства конкретных задач. Рассмотрим несколько уровней текстовых редакторов, каждый из которых обладает дополнительными возможностями по сравнению с предыдущими:

- Редакторы для составления программ. Их основные возможности - поддержка средств работы с блоками текста, поиск и замена по файлу.
- Редакторы для подготовки статей без графики и сложных формул. Добавляются возможности форматирования абзацев, расстановки переносов и разбиения текста на страницы.
- Редакторы для подготовки статей со сложными рисунками и формулами. Добавляются возможности включения графики в текст, поддержки многих кодировок и управления шрифтами.
- Редакторы для подготовки книг и журналов. Добавляются возможности работы с большими файлами и сбор оглавления.
- Редакторы для подготовки научно-технической документации. Добавляются возможности поддержки" листов изменений, колонтитулов и предметных указателей.

Редакторы для составления программ могут учитывать синтаксическую структуру обрабатываемой программы и выполнять дополнительный анализ и проверку (см. разд. 3.1.7).

Вот несколько наиболее известных и популярных текстовых редакторов:

- Microsoft Word (компании Microsoft в ОС Windows и аналогичный ему по возможностям редактор Writer в составе StarOffice[18] в Unix;
- Биллом Джоем (Bill Joy) был разработан полноэкранный текстовый редактор для Unix – **vi**. Этот редактор до сих пор является одним из самых используемых стандартных текстовых редакторов на компьютерах с операционной системой Unix.

3.1.7. Синтаксически-ориентированные редакторы

[18] Oracle Open Office (до 2010 года – StarOffice) – проприетарный офисный продукт корпорации Oracle. Apache OpenOffice (OpenOffice.org) – свободный пакет офисных приложений. InfraOffice.pro – вариант от российской компании Инфра-Ресурс. LibreOffice – независимое ответвление OpenOffice с более свободной политикой развития.

Синтаксически-ориентированные редакторы учитывают при работе структуру обрабатываемого текста. Они позволяют реализовать следующие возможности:

- предоставление пользователю возможности редактирования текста на уровне смысловых конструкций используемого языка;
- избавление пользователя от необходимости детально знать правила записи конструкций языка программирования;
- ускорение ввода программы за счет подстановки языковой конструкции целиком при нажатии одной функциональной клавиши;
- ведение непрерывного контроля правильности программы, включая синтаксис и полную статическую семантику;
- обеспечение автоматического форматирования текста программ и ее просмотра в удобном режиме.

Примеры синтаксически-ориентированных редакторов:

- EMACS (организации Free Software Foundations) – редактор, в основе которого лежит LISP-система;
- MENTOR – исторически первая система структурного редактирования;
- ALOE – генератор структурных редакторов в рамках проекта GANDALF.

3.1.8. Гипертекстовые средства

Средства исследования и просмотра с гипертекстовыми возможностями позволяют просматривать и редактировать текст (обычно – программный код), используя специальные средства браузеров, такие как выделение цветом, стилем, гипертекст и т. п. Разработаны и продолжают разрабатываться системы, целью которых является исследование и просмотр исходного текста крупных проектов с открытым исходным текстом. В качестве средства просмотра используется обычный html-браузер.

Примеры средств:

- Linux Cross-Reference - LXR (http://lxr.linux.no) – было создано как средство просмотра ядра операционной системы Linux;
- Mozilla Cross-Reference (http://lxr.mozilla.org) – средство просмотра исходных текстов браузера Mozilla;

▪ FreeBSD Unix Cross-Reference – средство просмотра текстов операционной системы FreeBSD Unix.

3.2. Системы документирования

Существуют различные генераторы документации, большинство из которых основано на различных принципах сбора информации по исходным текстам программ.

Пример такой программы:

• javadoc (компании Sun Microsystems, Inc. – генератор документации Java. Он считывает исходные тексты программ на языке Java, ищет комментарии специального вида и сопоставляет их с классами и методами, которые они описывают. Собрав все элементы, генератор придает им вид HTML-структуры и сохраняет в виде файла. Комментарии для генератора похожи на блочные комментарии в языке C, но имеют две звездочки после первой наклонной черты. Существует несколько ярлыков – символов, имеющих специальное значение (например, знак @). Так, комбинация символа и слова - @author заставит программу использовать специальные команды HTML, выделяющие имя автора в программе;

3.3. Системы разработки интерфейсов

Системы разработки интерфейсов в начале 90-х годов прошлого века составляли большую долю в инструментарии. Сейчас такие системы входят составной частью в CASE-средства. Приведем примеры некоторых инструментов, которые можно использовать для разработки интерфейсов:

- C++ Visual Studio (компании Microsoft);
- C++ Workshop Visual (компании Sun Microsystems);
- Delphi Suite (компании Borland Inc. (http://www.borland.com);
- средства построения графического интерфейса в Java (компоненты и контейнеры).

3.4. Системы управления базами данных

База данных – это последовательная совокупность данных, отображающая состояние объектов и их отношения в рассматриваемой предметной области и хранящаяся с минимальной избыточностью.

Система управления базами данных (СУБД) – совокупность языковых и программных средств, предназначенных для создания и ведения базы данных. СУБД располагается между собственно физической базой данных и пользователями системы.

Основная функция СУБД – предоставление пользователю базы данных возможности работать с ней, не вникая в детали на уровне аппаратного обеспечения. Другие функции СУБД – реализация транзакций, обеспечение безопасности данных и т.п.

По типу поддержки модели данных различают несколько классов СУБД:

- реляционные СУБД;
- объектные СУБД;
- логические СУБД.
- Значительно менее популярными в настоящее время являются СУБД, поддерживающие модели данных: иерархическую, сетевую, плоский файл, реляционную с вложенностью, семантическую и многомерную.

Заметим, что в большинстве случаев при проектировании базы данных сначала строится инфологическая модель предметной области, не зависящая от конкретной модели представления информации (и, следовательно, СУБД). Для ее создания обычно используются два источника информации:

- предметная область (в которой проводятся исследование, абстрагирование, поиск);
- информационные потребности пользователя (на основе которых выполняются описания).

К. Дж. Дейт[19] утверждает, что "и с экономической, и с теоретической точки зрения можно сказать, что реляционный подход наиболее важный (и это положение дел, по-видимому, не изменится в обозримом будущем)".

Вот некоторые наиболее известные реляционные СУБД:

- DB2 (компании IBM (http://www-4.ibm.com/software/data);
- Oracle (компании Oracle (http://www.oracle.com);
- Access (компании Microsoft (http://www.microsoft.com).

19 Кристофер Дейт (Christopher J. Date) (р. 1941) – специалист в области баз данных. Автор классического учебника «Введение в системы баз данных», который как стандартный используется во многих университетах мира.

Две следующие разработки относятся к классу расширяемых реляционных СУБД, позволяющих высококвалифицированным пользователям настроить систему заданным образом (например, определить собственные типы данных, структуры-хранения, функции и т.п.).

- Postgres (создана в Калифорнийском университете в Беркли (http://db.cs.berkeley.edu);
- Starburst (компании IBM (http://www.research.ibm.com).

3.5. Системы управления базами знаний и экспертные системы

3.5.1. Системы искусственного интеллекта

Искусственный интеллект (artificial intelligence) – одна из областей информатики, направленная на моделирование и решение задач, связанных с обработкой символьной информации, логикой и естественным языком. Искусственный интеллект ставит перед собой и более серьезную задачу построения теории интеллекта, базирующуюся на обработке информации. Отдельные ветви этой области касаются психологии, философии и лингвистики. Базы знаний и экспертные системы – два наиболее известных типа систем, связываемых с искусственным интеллектом.

К системам искусственного интеллекта можно предъявить различные требования:

- Работа с неполной и неточной информацией. Системы должны поступать "как крутой детектив". На основе неполной и неточной информации они разрабатывают первый вариант алгоритма, который потом совершенствуется на основе новой информации, получаемой системой.
- Понимание *плохо*формализуемого. Системы должны быть "как кулинарная книга". Например, "добавить по вкусу", "варить до готовности" и т.п.
- Самообучение. Системы должны не только хранить закономерности, но и показывать – как они используются при поиске решения. Например, при конструировании электрического стула система должна найти закон Ома и использовать его в расчетах.
- Понимание смысла вопроса. Например, на вопрос "Вы не скажете, который час?" в некоторых случаях вполне разумным будет ответ "Меня зовут Лена".

• Наличие возможности обучения и объяснения. Системы должны "напоминать толмача". Они должны иметь возможность обучения и объяснения.

3.5.2. Механизмы выводов на знаниях

Под механизмом вывода в экспертных системах будем понимать ту часть экспертной системы, в которой содержатся общие знания о схеме управления решением задач. Машина вывода выполняет две основные функции:

• Просмотр существующих фактов из рабочей памяти (базы данных) и правил из базы знаний и добавление в рабочую память новых фактов.

• Определение порядка просмотра и применения правил.

Вот лишь некоторые наиболее известные системы, применяющие механизм выводов:

• MYCIN (разработка Стэнфордского университета (http://www.stanford.edu) – экспертная система в области медицины;

• PROSPECTOR (компании SRI International) – экспертная система в области геологии;

• Cattell (разработка компьютерного центра РАН (http://www.ccas.ru/~posp) – экспертная система в области социальной психологии.

3.5.3. Неточный вывод на знаниях

Данные и знания в экспертных системах могут иметь неопределенность, причем для ее описания применяются разнообразные средства.

Рассмотрим один из типов неопределенности – неточность. Неточное высказывание – высказывание, истинность которого не может быть установлена с определенностью.

В качестве примеров систем, применяющих аппарат неточного вывода на знаниях, приведем:

- CubiCalc (компании HyperLogic (http://www.hyperlogic.com)
- FuziCalc (компании FuziWare Inc.).

3.6. Электронные библиотеки и инструментарий Интернета

3.6.1. Парадигма усиления информации

Рассмотрим парадигму[20] усиления информации, которая достаточно удачно описывает роль ресурсов Интернета.

Парадигма проводит параллели с принципом, положенным в основу работы транзистора: усиление сигнала при запитке базы.

Интернет играет роль "информационного усилителя", питая пользовательский запрос информацией и переводя его в совокупность структурированной информации (электронную библиотеку).

3.6.2. Профессиональный поиск информации

Результаты профессионального поиска информации играют огромную роль в методах научного познания. Этапы, соответствующие началу научного исследования, могут быть алгоритмизированы.

- Формулировка проблемы, постановка цели (что хотим достичь) и конкретных задач (что необходимо сделать для достижения цели).
- Выбор методов исследования и построение стратегии информационно-аналитического поиска. Напомним, что общие методы научного познания обычно делят на следующие три большие группы:
  - методы эмпирического исследования (наблюдение, сравнение и др.);
  - методы, используемые как на эмпирическом, так и на теоретическом уровне исследования (абстрагирование, анализ и синтез и др.);
  - методы теоретического исследования (восхождение от абстрактного к конкретному и др.).

Как мы видим, практически все методы требуют поиска информации и использования результатов поиска в качестве входных данных.

В случае простейшей (и наиболее распространенной) стратегии поиска информации могут быть выделены два основных этапа:

- Запрос информации с целью исследования некоторой предметной области.
- Выборка действительно необходимой информации и генерация отчета по исследуемой проблеме.

---

[20] Парадигма – (греч. «пример, модель, образец») – система идей и понятий.

Именно на эти этапы должны быть ориентированы системы, предназначенные для обработки информации, полученной из Интернета.

В настоящее время крупнейшим информационным ресурсом является Интернет, и в первую очередь – среда World Wide Web (WWW) – глобальная интерактивная распределенная гипертекстовая информационная система. Предпринятые попытки оценить размер информации в Интернете согласуются в том, что в среде WWW содержится более миллиарда страниц. Если предположить, что размер среднестатистической страницы составляет 4–10 Кбайт, то речь идет о терабайтах информации. Кроме поверхностной (не скрытой за поисковыми формами) части Интернета, существует скрытая (hidden, deep) часть, к которой относится множество крупных баз данных. Объем скрытой части оценивают в 500 раз больше, чем поверхностной.

3.6.3. Проблемы работы с информационными ресурсами Интернета

Можно предположить, что в Интернете можно найти информацию практически по любой тематике. Однако сделать это достаточно сложно, т.к. наиболее распространенные и известные поисковые инструменты – справочники и поисковые серверы – не позволяют эффективно структурировать результаты поиска. Кроме того, возникает задача отсеивания данных, т. е. отсечения ненужной и несвязной информации от той ее части, которая будет полезной. Существует несколько проблем, возникающих при работе с информацией в Интернете и препятствующих быстрому и качественному поиску.

- Недостаточная структурированность информации (наличие так называемых слабоструктурированных данных). Для облегчения поиска в таких документах было предложено несколько моделей представления такой информации, в том числе Stanford's Object Exchange Model[21], в рамках которой данные представляются в виде направленного графа с поименованными вершинами и дугами.
- Избыточность информации. Около трети информации в Интернете является точными или приблизительными копиями других документов.

[21] http://www.rocq.mria.fr/~simeon/semistructure/art.html

• Наличие противоречивых и недостоверных сведений. Типичным примером является включение некорректных ключевых слов в группу инструкций META, специально предназначенных для описания и индексирования документов поисковыми машинами. Некоторые поисковые машины сделали попытку решить эту проблему с помощью полнотекстовой индексации документов. Однако сетевые мастера Интернета предложили сразу несколько способов обмануть поисковый робот, например, писать белым шрифтом по белому фону все то, что ранее включалось в инструкцию META.

• Большое количество ошибок (например, опечаток, грамматических ошибок, ошибок оцифровки). Эти ошибки являются следствием неконтролируемого качества, включая отсутствие редакторского контроля над публикуемой информацией.

Укажем на две особенности поиска в Интернете, связанных с понятием человеческий фактор.

• Поведение пользователя. Пользователь не готов долго ждать результата и не готов даже искать его в предоставленной поисковой системой выборке. Практически половина пользователей не идет далее первого экрана, предложенного информационно-поисковой системой.

• Неумение делать запросы. Большинство пользователей не использует расширенные возможности поиска, такие как логические выражения. Кроме того, типичные поисковые запросы очень коротки – более 60% запросов состоят всего из 1–2 слов.

Существует необходимость в совершенствовании программного обеспечения, которое поможет пользователю в интеллектуальном поиске и отборе нужной информации.

3.6.4. Коллекции информационных ресурсов в Интернете

Электронная библиотека – система, обеспечивающая сообществу пользователей доступ (понятным для них образом) к большим репозиториям мультимедийной информации и знаний. Причем эти репозитории организованы при отсутствии каких-либо сведений о способах их применения.

Коллекция информационных ресурсов – систематизированная совокупность информационных ресурсов, объединенных по какому-либо критерию принадлежности, например, по общности содержания, источников, назначения, по кругу пользователей, способу доступа и т.д. Коллекции являются наиболее распространенной

формой организации информационных ресурсов в электронных библиотеках.

С функциональной точки зрения информационные ресурсы коллекции подразделяются на данные (информацию) и метаданные (метаинформацию). Ресурсы первого вида представляют интересующие пользователей сведения о предметной области этой коллекции. В свою очередь, метаданные коллекции характеризуют свойства самой коллекции и принадлежащих ей ресурсов в целом как сущностей реального мира.

Систематизация коллекций – задание структуры коллекции, осуществляемое на основе свойств ее предметной области и (или) свойств составляющих ее информационных ресурсов.

Систематизированный характер информационных ресурсов коллекции является принципиально важным ее свойством, отличающим коллекцию от других наборов таких ресурсов. Осмысленная систематизация информационных ресурсов не только облегчает доступ пользователей к ним, но и дает возможность целенаправленно и рациональным образом исследовать с их помощью предметную область коллекции.

Наряду с систематизированностью, к числу важных свойств любой коллекции относятся ее назначение, характеристики происхождения, способ задания состава принадлежащих ей ресурсов и правила их именования, виды используемых информационных технологий, характеристики представления информационных ресурсов и т.д.

Возможны различные подходы к заданию состава коллекций:

- явным образом – непосредственно как совокупность принадлежащих ей информационных ресурсов или как список ссылок на них (например, URL в WWW);
- неявным образом – путем спецификации в какой-либо форме критерия принадлежности информационного ресурса данной коллекции. Примером может служить задание коллекции на основе полнотекстовой документальной системы путем спецификации поискового запроса.

Систематизация коллекции с использованием свойств ее предметной области должна основываться на концептуализации предметной области. Концептуальная модель предметной области может иметь различные формы представления. Если коллекция реализуется на основе технологии баз данных, то модель представля-

ется в виде концептуальной схемы, которая отображается в среду СУБД и описывается схемой базы данных.

Для многих коллекций концептуальная модель предметной области представляется в форме классификатора, определяющего одномерное или многомерное пространство классификационных признаков. В таком случае отдельные точки или гиперплоскости этого пространства представляют классы сущностей предметной области, соответствующие ресурсам коллекции. В качестве классификационных признаков обычно используют наиболее существенные свойства этих сущностей. Для классификационных признаков может быть предопределено множество их возможных значений. При этом множество значений отдельного признака может иметь линейную или иерархическую структуру.

3.6.5. Базы данных в Интернете

Среди информационных ресурсов Интернета особый интерес вызывают базы данных, которые подразделяются на:

- текстовые (полнотекстовые, реферативные, библиографические);
- содержащие изображения и использующие средства мультимедиа;
- числовые и табличные;
- содержащие программное обеспечение.

Базы данных в Интернете рассчитаны как на массового, так и на профессионального потребителя. Баз данных первого типа – большинство. Ко второй группе относятся профессиональные информационные системы, представляющие собой специализированные базы данных и поисковые программы. Например, крупнейший мировой продавец информации – компания Questel-Orbit, разместила в Интернете базу данных патентов (http://www.qpat.com). Специализированные базы данных, как правило, имеют свой узкоспециальный интерфейс и собственную уникальную структуру. В последние годы появилась тенденция к интеграции и стандартизации профессиональных баз данных.

3.6.6. Краткая история поисковых средств Интернета

Активное внедрение средств компьютерных телекоммуникаций, глобальных компьютерных сетей и сетевых баз данных сделало сеть Интернет доступной и необходимой для многих. Интернет предоставляет своим пользователям информационные ресурсы и средства работы с ними.

Необходимость создания поисковых средств Интернета была осознана практически с момента создания сети. Исторически известны такие приложения, как Apachie (для ftp), Veronica (для Gopher) и WAIS (поиск в индексированных базах данных). На сегодняшний день наиболее развитый и удобный в использовании инструмент – поисковые системы.

Поисковая система состоит из компонентов, таких как:

- сетевой робот, занимающийся сбором информации о доступных в Интернете ресурсах. Собранная информация складывается в хранилище. Его содержимое определяет набор документов, по которым идет поиск;
- модуль индексирования, создающий индексные структуры, по которым для достижения приемлемой эффективности и производится поиск;
- поисковая машина, получающая и выполняющая запросы пользователей.

Серверы, на которых расположены поисковые системы, называют поисковыми серверами.

В число наиболее посещаемых англоязычных поисковых систем (и, соответственно, серверов) входят:

- Google (http://www.google.com);
- Yahoo (http://www.yahoo.com);
- AltaVista (http://www.altavista.com).

В русскоязычной части Интернета популярны:

- Яndex (http://www.yandex.ru);
- Rambler (http://www.rambler.ru).

Однако эти поисковые инструменты не позволяют эффективно структурировать результаты поиска. Кроме того, возникает задача отсеивания *полу*ценных данных, т.е. отсечения ненужной и несвязной информации от той ее части, которая будет полезной (см. разд. 3.6.3).

Одним из наиболее перспективных методов работы с информацией в Интернете является ее структурирование в тематические

электронные библиотеки. Активное использование персональных электронных библиотек должно служить созданию эффективной инфраструктуры для поддержки научных исследований и других сфер деятельности.

3.6.7. Искусственный интеллект и задача поиска в Интернете

Тематики исследований в двух крупных направлениях информатики – области искусственного интеллекта и Интернете – сближаются. Методы искусственного интеллекта все в большей степени ориентируются на задачи практического применения, а Интернет стремится к более сложным приложениям, требующим интеллектуального поведения.

Многие методы и приемы искусственного интеллекта нашли применение в конкретных проблемных областях. Перечислим лишь некоторые направления, которые могут с успехом применяться для разработки Интернет-приложений:

- Обработка естественного языка.
- Извлечение информации из баз данных.
- Экспертные консультирующие системы.

Понятие *онтология*[22] было введено в искусственный интеллект Томом Грубером[23], как спецификация концептуализации. Подобно формальным спецификациям программных продуктов, онтология – это описание концептов (базовых понятий, классов), свойства концептов (атрибуты, роли), отношения между концептами (зависимости, функции) и дополнительные ограничения, которые определяются аксиомами. Предполагалось, что онтологии будут использованы исключительно для облегчения взаимодействия интеллектуальных агентов (см. ниже). Затем понятие онтологии было расширено. В частности, иерархическая структура классов в объ-

[22] Одно из значений слова Онтология – это артефакт, структура, описывающая значения элементов некоторой системы. На формальном уровне онтология – это система, состоящая из набора понятий и набора утверждений об этих понятиях, на основе которых можно описывать классы, отношения, функции и индивиды. Одно из самых известных определений онтологии дал Том Грубер, звучит оно следующим образом: Онтология – это точная спецификация концептуализации. Концептуализация – это структура реальности, рассматриваемая независимо от словаря предметной области и конкретной ситуации.

[23] Gruber, 1993

ектно-ориентированном программировании представляет собой онтологию.

Большую роль в сборе информации о существующих ресурсах Интернета играют сетевые роботы, являющиеся интеллектуальными агентами. Они, начиная с некоторого множества ссылок на страницы в сети Интернет, рекурсивно обходят ресурсы, извлекая ссылки на новые ресурсы из полученных документов, до тех пор, пока не будет выполнено некоторое условие остановки.

Уточним понятия агента и интеллектуального агента.

- Агент – сущность, находящаяся в некоторой среде, от которой она получает данные, отражающие происходящие в этой среде события. Агент интерпретирует данные и исполняет команды, которые воздействуют на среду.
- Интеллектуальный агент – программно- или аппаратно- реализованная система, обладающая следующими свойствами:
  - автономность – способность функционировать без вмешательства человека;
  - общественное поведение – способность функционировать в сообществе с другими агентами;
  - реактивность – способность своевременно отвечать на изменения окружающей среды;
  - проактивность – способность агента брать на себя инициативу.

В настоящее время исследование применимости идей искусственного интеллекта к среде Интернет ведется в различных направлениях:

- Обработка поисковыми машинами запросов на естественном языке. Система получает запрос на естественном языке, с помощью грамматических и лингвистических правил сопоставляет запрос с информацией в Интернете. Удачным примером такой системы является Яndex (http://www.yandex.ru). В ней независимо от того, в какой форме употреблено в запросе слово, поиск учитывает все его формы по правилам русского языка.
- Продукционно-эвристическое распознавание естественного языка. Системы, развивающиеся в этом направлении, как правило, являются узкопрофильными и предназначены для выполнения

очень узко сформированных целей или обработки ограниченных областей данных. Пример системы – FAQFinder[24].

- Автоматическое наполнение базы знаний. Это и следующее направления используют агентную технологию. Данное направление реализовано в интеллектуальном браузере WebWatcher[25].
- Использование эвристических правил для установки приоритета. Это направление реализовано в другом интеллектуальном браузере – Letizia[26]
- Направление нейронных сетей. Основой здесь являются сети искусственных нейронов и других аналогичных конструкций, присущих нервной системе человека, в которых протекают психические процессы, опирающиеся на физиологический и биохимический уровни. Пример реализации этого направления – система работы со знаниями – Autonomy (http://www.autonomy.com).
- Принцип разделяемых знаний. Реализуется в виртуальных группах, т.е. организациях и рабочих коллективах людей, которые работают и общаются между собой в интерактивном режиме. Пример такого рода систем – AJMS[27]

Экспериментальные системы интеллектуального поиска используют одновременно мощь существующих поисковых систем и элементы искусственного интеллекта для отбора и анализа извлеченной из Интернета информации. Поисковая система по запросу пользователя обращается на крупнейшие поисковые серверы. Полученные ссылки сортируются. Анализируются html-файлы, расположенные по этим адресам. Пользователю предлагаются выдержки из текста документов, отражающие результаты поиска.

Каждый пользовательский запрос, прежде всего, преобразуется в дизъюнктивную форму со строго определенным порядком следования логических операций. Приоритеты операций расставлены следующим образом (табл. 1).

Таблица 1.

Приоритеты логических операций

Приоритет	Обозначение	Назначение

[24] http://www.infolab.nwu.edu/faqfinder

[25] http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-6/web-agent/www/project-home.html

[26] http://lcs.www.media.mit.edu/people/lieber/Lieberary/Letizia/Letizia-Intro.html

[27] http://aims.parl.com/About-AIMS.html

0	" "	Поиск по точному соответствию
1	!	Исключение лексемы[28]
2	&	Логическое И
3	|	Логическое ИЛИ

Текстовый редактор предполагает возможность фильтрации и сортировки логических абзацев. Сортировка может производиться по следующим принципам:

- по релевантности логических абзацев;
- по приоритету документов;
- по рассматриваемым поисковым серверам;
- по частоте появления документа на серверах;
- по заголовкам документов.

Таким образом, на вход генератора итогового документа подается набор логических абзацев со списками атрибутов. Синтезировать отчет можно в нескольких формах, различающихся порядком следования логических абзацев, с предоставлением пользователю возможности переключаться из одного режима в другой.

4. Инструментарий поддержки процессов некоторых технологических подходов

4.1. Системы формального преобразования и верификации программ

Поддержка технологий формального преобразования и верификации[29] программ осуществляется специальными системами. Большинство из них являются научно-исследовательскими и не имеют коммерческого применения. Одной из наиболее интересных современных работ в области формальных подходов верификации программ является B-технология. На ее основе была осуществлена разработка системы управления парижским метрополитеном "METEOR".

---

[28] последовательность допустимых символов языка программирования, имеющая смысл для транслятора.

[29] Верификация (от лат. verus – «истинный»), это способ подтверждения, проверка с помощью доказательств, каких-либо теоретических положений, алгоритмов, программ и процедур путем их сопоставления с эталонными данными, алгоритмами и программами.

Цель создания средств формального преобразования программ состояла в том, чтобы автоматизировать задачи разработки программного обеспечения, модификацию программ и исправление ошибок. Преобразования специфицировались, используя правила, левая часть которых – это образец исходного кода, а правая – действия, которые должны быть выполнены, если левая часть встретилась в тексте. Приведем примеры таких систем.

- Система TXL[30] конвертирует код, написанный на диалекте, в код на базовом языке. Это выполняется через преобразование синтаксического дерева грамматики диалекта в синтаксическое дерево базового языка, а затем получение новой программы из последнего. Для преобразования деревьев использовался рекурсивный алгоритм обхода дерева.
- Средство IntelliJ Renamer[31] – это мощное средство для преобразования программ и поиска в программном коде, разработанное для языка Java. Оно позволяет производить переименование методов, пакетов, полей, переменных и параметров методов с автоматическим исправлением вхождений. IntelliJ CodeSearch позволяет делать поиск в исходном коде на языке Java, к примеру, вызовов определенных методов или определения классов, реализующих некоторый интерфейс. Это средство корректно работает с перегруженными методами различных классов с одинаковыми именами. В этих средствах используется идея работы по синтаксическому дереву.
- Система RescueWare[32] является системой производства программ с новой бизнес-логикой, преобразуя и используя части существующих программных ресурсов. Для осуществления этих операций система предлагает разделение задачи на несколько подзадач. Сначала проходит стадия понимания программ. На этом этапе выделяются такие сущности, как программный уровень, потоки данных, карта вызовов. Далее проходит стадия изоляции определенной бизнес-логики. На этом этапе выделяются компоненты, пригодные для *пере*использования. Впоследствии созданные компоненты используются для генерации новой программной системы с учетом особенностей новых платформ. В таких подходах ключевым мо-

[30] Kozaczynsky, Hing, Engberts 1992

[31] http://www.intellij.com/products

[32] http://www.relativity.com/products/rescueware

ментом, зачастую, является преобразование программы на процедурном языке в программу на объектно-ориентированном языке. Для осуществления такого преобразования объекты должны быть идентифицированы из процедурных частей. Объектно-ориентированные программы (в сравнении с процедурно-базируемыми) считаются более простыми в сопровождении.

Существуют также системы, которые предназначены для анализа исходных кодов с целью получения высокоуровневых абстракций. Так, например, существуют средства, которые анализируют код на языке Java с целью генерации диаграмм UML. Данный подход базируется на использовании сетей рассуждений, построенных на нечеткой логике. Также существуют средства, которые предоставляют среду разработки для объектно-ориентированных языков, используя диаграммы UML. Они позволяют не только генерировать код по диаграммам, но и анализировать его с целью построения такой диаграммы. Это позволяет более наглядно представлять архитектуру разрабатываемой системы и улучшать ее понимание.

4.2. Средства сборочного программирования

Несмотря на естественность и популярность сборочного программирования, средства его поддержки практически отсутствуют или не выходят за пределы компаний по разработке программного обеспечения. Это объясняется тем, что с одной стороны существует огромное количество правовых проблем, препятствующих свободному копированию программных модулей, а с другой – низким качеством и плохой документированностью таких программ. В настоящее время наиболее популярным средством сборочного программирования являются репозитории, которые мы рассмотрим в разд. 5.4.

5. Инструментальные системы

Инструментальной системой будем называть некоторую логически связанную совокупность программных инструментов, поддерживающую разработку и сопровождение программных систем на данном языке программирования или ориентированных на какую-либо конкретную предметную область.

Эту группу характеризует в первую очередь интеграция нескольких инструментов под единой оболочкой и использование общего репозитория.

Выделяют три группы инструментальных систем:

- Инструментальные среды программирования.
- Средства автоматизации разработки программ.
- Интегрированные среды.

В идеальном варианте инструментальные системы должны распространяться на максимально возможное количество процессов и покрывать максимум стадий жизненного цикла. Однако исторически сложилось так, что инструментальные среды в большей степени связаны с процессами программирования, тестирования и отладки, а средства автоматизации разработки программ – с анализом и проектированием.

5.1. Инструментальные среды программирования

Инструментальные среды программирования обычно содержат текстовый редактор, компилятор, отладчик и средства подсказки. Кроме того, в них могут быть включены и другие инструменты, позволяющие выполнять статический и динамический анализ программ. Эти инструменты взаимодействуют между собой через обычные файлы с помощью стандартных возможностей файловой системы. Различают среды общего назначения и языково-ориентированные среды.

- Среды общего назначения содержат набор программных инструментов (например, текстовый редактор, редактор связей и т.п.), позволяющих выполнять разработку программ на разных языках программирования. Для программирования на конкретном языке программирования требуются дополнительные инструменты, ориентированные на этот язык.
- Языково-ориентированные среды предназначены для поддержки разработки программ на каком-либо одном языке программирования, причем построение такой среды базируется на знаниях об этом языке.

Инструментальная среда не обязательно должна функционировать на том компьютере, на котором должен будет применяться разрабатываемый с ее помощью программный продукт.

Особенности инструментальных сред программирования:

- поддерживают различные методологии;
- применяются в различных технологиях;

- применяются командами, работающими над различными проектами;
- используются для разработки разнообразных приложений;
- разрабатываются одной компанией. Возможность интеграции инструментов других компаний отсутствует.

Примеры инструментальных сред:

- Visual Studio (компании Microsoft);
- Forte for Solaris Developer Tools (Sun Microsystems Inc.);
- Delphi Suite (компании Borland International).

5.2. Средства автоматизации разработки программ (CASE-средства)

Средства автоматизации разработки программ – инструментарий для системных аналитиков, разработчиков и программистов, позволяющий автоматизировать процесс проектирования и разработки программного обеспечения. Первоначально под CASE-средствами понимались средства, применяемые на ранних процессах жизненного цикла. В первую очередь – на наиболее трудоемких процессах анализа и проектирования. Международный стандарт [ISO/IEC 14102:1995] определяет CASE-средства более широко – как программное средство, поддерживающее процессы жизненного цикла программного обеспечения.

CASE-средства характеризуются наличием мощных средств визуального моделирования.

Особенности средств автоматизации разработки программ:

- поддерживают единственную методологию;
- ориентируются на определенную технологию;
- предназначаются для команд, работающих над единственным проектом (так сложилось исторически);
- используются для разработки информационных систем;
- разрабатываются одной компанией. Возможность интеграции инструментов других компаний отсутствует.

Примеры CASE-средств:

- Oracle Designer (компании Oracle (http://www.oracle.com/));

• AllFusion ERwin Data Modeler (ранее ERwin) (компании Computer Associates International, Inc. (http://www.cai.com/));

• Rational Rose (компании Rational Software Corporation (http://www.rational.com/)).

5.3. Интегрированные среды

Интегрированная среда – совокупность программных инструментов, поддерживающая все процессы жизненного цикла программного обеспечения в рамках определенной технологии. Компонентами ин

разобраться в путанице -
инструментальные системы
интегрированные среды (IDE)

например, чем является Visual Studio

• инстр

• инстр

• инстр ия;

• инстр

• инструменты поддержки разработки документации.

Выделяют три уровня интеграции инструментов в интегрированных средах.

• Уровень 1. Интеграция инструментов очень слабая. Как правило, обмен информацией между ними происходит через интерфейсы экспорта и импорта.

• Уровень 2. Интеграция инструментов одной компании осуществляется на основе единого репозитория. Интеграция инструментов других компаний с первыми инструментами происходит по образцу уровня 1.

• Уровень 3. Интеграция всех инструментов осуществляется с помощью общего репозитория. При этом любой инструмент любой компании может осуществлять взаимодействие через службы взаимодействия с репозиторием.

Особенности интегрированных сред:

• поддерживают различные методологии;

• определяют технологию разработки;

• применяются командами, работающими вместе над несколькими проектами;

• как правило, используются для разработки научных и инженерных приложений;

• разрабатываются одной компанией, но имеется возможность интеграции инструментов других компаний.

Примеры интегрированных сред:

- WebSphere Studio WorkBench (компании IBM (http://www.ibra.com/software/ad/workbench/));
- CohesionWorX (компании Digital Equipment Corp.);
- SorfBench (компании Hewlett-Packard (http://www.hp.com/)).

Об аппаратной платформе интегрированных сред

Обратим внимание, что интегрированные среды практически отсутствуют на персональных компьютерах. Это можно объяснить сильным монополизмом компаний - разработчиков инструментария на рынке персональных компьютеров и их нежеланием предоставлять возможность интеграции инструментов других компаний.

5.4. Репозитории проекта

Репозитории – хранилище информации, связанной с проектом разработки программного продукта в течение всего его жизненного цикла. В современных инструментальных системах репозитории приобретают роль фундамента всей информационной среды. Важно, чтобы репозитории помогали работе инструментальных систем.

Филипп Бернштейн[33] определяет репозитории как разделяемую базу данных с информацией об артефактах проектирования, требующую некоторых дополнительных функций управления помимо предоставляемых обычными системами баз данных.

Большинство технологических подходов к разработке программного обеспечения предполагает работу с тремя основными типами информации – модельными спецификациями, интерфейсом прикладного программиста и окружением проекта. Выделим три уровня репозиториев, соответствующих этим типам.

- Модельный.
- Программного интерфейса.
- Окружения.

Уровень моделирования достаточно хорошо может быть описан универсальным языком UML. Данный язык является абстрактным, не привязанным к конкретной модели. Язык дает возмож-

[33] Philip A. Bernstein is a computer scientist, specializing in database research in the Database Group of Microsoft Research.

ность описать зависимости элементов, иерархию, взаимосвязи, свойства и т.п.

Уровень программного интерфейса разумно описывать с помощью языка определения интерфейсов IDL, обеспечивающего независимость спецификации интерфейсов от их реализации. Уровень играет не только роль промежуточного слоя, также его средства поддерживают распределенное программирование (в том числе в Интернете).

Репозиторий окружения программного проекта предназначен для хранения информации, разделяемой компонентами и подкомпонентами систем программирования в процессе их работы.

Приведем ниже основные группы и подгруппы полезной информации.

- Языково-независимая группа.
  - Инфо тладчика.
  - Инфо росмотрщика исходных текстов.
- Языково уппа (например, для языка C++).
  - Инфо аблонов.
  - Коды ункций.
  - Вирт ции.
- Группа и зитория.
  - Конт ции о версиях.
  - Конт ия к проекту.
  - Тип обработки.
  - Тип управления репозиторием.
  - Проверка цифровой подписи.

непонятно это к чему?

Основными достоинствами применения репозиториев окружения являются:

- эффективность работы с информацией;
- использование информации для целей оптимизации;
- распределенность (из которой следуют доступность, параллелизм и специализация);
- модульность, включающая независимость от конкретных инструментов (например, компиляторов);

• возможность работы с репозиторием как в архитектуре "клиент-сервер", так и в "связанном" с инструментом режиме.

Статистика отмечает, что около 80% программного обеспечения создается по уже имеющемуся. Следовательно, необходимо иметь инфраструктуру (электронную библиотеку), которая будет поддерживать архивы и достаточно интеллектуальный поиск нужных прототипов и фрагментов. Фактически, мы на более высоком уровне обращаемся к парадигме сборочного программирования, которая характеризуется стремлением к выделению независимой единицы программистского знания (модульной абстракции).

Одним из наиболее известных репозиториев является Microsoft Repository.

6. Средства поддержки коллективной разработки

Коллективные разработки часто сталкиваются с проблемами разделения ресурсов – файлов и каталогов и необходимости синхронизации времени и места.

6.1. Системы разделения файлов

Для поддержки коллективной работы с файлами применяются три основных класса систем.

- Системы управления версиями файлов.
- Системы управления пространствами пользователей.
- Системы синхронизации удаленных пространств.

Если три класса систем выпускаются одним производителем, то часто каждая последующая система в этом списке использует предыдущую систему, выступая в качестве надстройки.

О системах управления версиями

Обратим внимание, что системы управления версиями файлов следует использовать не только при коллективной, но и при индивидуальной (авторской) разработке для решения задачи отслеживания изменений между различными версиями. Мы помещаем информацию об этих системах в данном разделе с целью подчеркнуть их полную незаменимость при коллективных разработках.

Поскольку данные системы действительно очень важны, их реализации существуют в большом количестве практически для всех операционных систем. Приведем два стека таких систем (табл. 5.2).

Классы систем	**Среда операционной системы Windows**	**Среда операционной системы Unix**
Управления версиями файлов	PCS (Revision Control System) (http://www.gnu.org/software/rcs/rcs.html)	SCCS (Source Code Control System) (http://www.sun.com/)
Управления пространствами пользователей	CVS (Concurrent Versions System) (http://www.cvshome.org/)	bringover/putback (http://www.sun.com/)
Синхронизации удаленных пространств	Rsync (http://samba.ann.edu.au/rsync)	synctool (http://www.sun.com/)

6.1.1. Система управления версиями файлов

Система управления версиями файлов (VCS) предназначена для отслеживания изменений между различными версиями файлов и разделения доступа к ним. Файлы обычно содержат программы на языках программирования и тексты.

Проиллюстрируем работу с версиями на примере системы SCCS[34]. При передаче файла под ее управление, система создает новый файл (так называемый s.-файл) в подкаталоге с именем SCCS. Именно в этом дополнительном файле будут отслеживаться и храниться все изменения: /имя_файла/SCCS/s.имя_файла

Укажем операции, наиболее часто выполняемые над файлом, помещенным под систему управления версиями.

- sccs create <имя_файла> - передать файл под управление SCCS.
- sccs get <имя_файла> - извлечь последнюю версию файла, открытую только для чтения.

[34] Source Code Control System (SCCS) – первая система управления версиями, разработанная в Bell Labs в 1972 году Марком Рочкиндом (англ. Marc J. Rochkind) для компьютеров IBM System/370, работавшая под управлением OS/MVT.

Затем была создана версия для PDP-11 под управлением операционной системы UNIX. В дальнейшем SCSS была включена в состав нескольких вариантов UNIX.

- sccs edit <имя_файла> - извлечь последнюю версию для редактирования, открытую для записи, и заблокировать возможность открыть файл для редактирования остальным пользователям.
- sccs unedit <имя_файла> - отменить редактирование и исключить информацию о попытке редактирования, дать возможность открыть файл для редактирования остальным пользователям.
- sccs delta <имя_файла> - вернуть измененную версию и добавить комментарий.
- sccs delget <имя_файла> - комбинация команд delta и get.

Другие, наиболее часто применяемые операции выглядят так:

- sccs prt <имя_файла> - вывести протокол изменений файла, включая комментарии.
- sccs dif fs <имя_файла> - выполнить сравнение двух версий файла.
- sccs info <имя_файла> - перечислить все файлы текущего каталога, открытые на редактирование командой edit.

Обратим внимание на то, что системой отслеживаются именно изменения (дельты) между версиями, а не сами полные версии, что позволяет занимать небольшое количество дискового пространства. Каталог SCCS часто называют репозиторием проекта.

Как и все широко распространенные программы утилита SCCS имеет надстройки в виде графического интерфейса, например утилиту vertool (virtool?).

Другие известные SCV:

- **TortoiseSVN** – ['tɔ:tes] это бесплатный клиент для системы контроля версий Subversion, выполненный как расширение оболочки Windows и распространяемый под лицензией GPL.

Будучи клиентом Subversion, TortoiseSVN позволяет управлять файлами и папками во времени. Файлы хранятся в центральном хранилище, в котором запоминается каждое изменение, сделанное в хранимых файлах и папках. Это даёт возможность восстанавливать старые версии файлов, и изучать историю их изменения. Поэтому Subversion и другие системы контроля версий часто называют «машинами времени» для файловой системы.

В 2007 году на SourceForge.net TortoiseSVN был признан лучшим проектом в категории «инструменты и утилиты для разработчиков».

• **Borland StarTeam** – это автоматизированная система управления конфигурацией и изменениями, которая обеспечивает эффективный контроль процесса разработки. StarTeam улучшает взаимодействие между всеми сотрудниками проектных групп, предоставляя пользователям доступ к любой важной информации проекта через центральный репозиторий, который поддерживается системой управления технологическими потоками и процессами. StarTeam превосходит традиционные средства, которые предлагают только функции контроля файловых версий, поскольку предоставляет пользователям гибкое, удобное в использовании комплексное решение на основе систем управления требованиями, управления изменениями, отслеживания дефектов, контроля файловых версий, тематических обсуждений, управления задачами и управления проектом.

• **Visual Studio Team System** – набор инструментов от Microsoft для разработки программных приложений, упрощения совместной работы над проектами, инструментов для тестирования и отладки разрабатываемых программ, а также построения отчетов. Visual Studio Team System (VSTS) использует **Team Foundation Service** (TFS) в качестве хранилища данных и серверной инфраструктуры для совместной работы над проектами. TFS – это полноценное облачное решение для управления жизненным циклом приложений, реализующее все необходимые возможности, включая контроль версий и анализ кода, а также гибкое планирование и автоматизацию построений. Более того, оно доступно практически везде.

6.1.2. Система управления пространствами

Система управления рабочими пространствами пользователей предназначена для обмена результатами работы между отдельными разработчиками через объединение результатов в выделенном рабочем пространстве. Система работает на основе модели "копирование–изменение–слияние" и понятия "рабочее пространство".

Рабочее пространство – специальным образом выделенная совокупность каталогов с файлами, переданными системе управления версиями файлов. Кстати, на профессиональном жаргоне разработчики называют свои рабочие пространства – песочницами.

Две основные команды работы с этими пространствами представляют собой копирование файлов – из пространства-предка в

пространство-потомка (например, команда bringover) и обратно (например, команда putback). Обе эти команды имеют одинаковый набор параметров, позволяющих указать пространство-потомка (опция -w), пространство-предка (опция -p), а также список файлов и/или каталогов. Опция -n позволяет не выполнять реальное копирование, а лишь выдать информацию о возможных действиях.

Модель "копирование–изменение–слияние" отражает схему организации работы с пространствами. Основная последовательность действий такова:

1. Создается основное мастер-пространство, от которого разработчики порождают свои рабочие пространства.

2. После внесения изменений в своем рабочем пространстве разработчик передает их в мастер-пространство.

3. Все разработчики периодически обновляют свои рабочие пространства из мастер-пространства.

Основная проблема, которая может при этом возникнуть, – это конфликт изменений в том случае, когда разработчик передает измененные файлы в мастер-пространство, но они уже оказываются измененными другим разработчиком. Можно попытаться избежать такой ситуации, выполняя обновления как можно Чаще или установив регламент для внесений изменений.

Тем не менее если такая ситуация возникла, то требуется выполнить под управлением разработчика слияние файлов, в ходе которого конфликт будет разрешен. Процесс слияния в большинстве случаев требует интеллектуального подхода при принятии решений в формировании итогового файла. Автоматизировать можно лишь сам процесс формирования файла, например, используя утилиту слияния файлов filemerge.

6.1.3. Система синхронизации удаленных пространств

В ряде случаев может оказаться, что пространства некоторых разработчиков находятся на значительном удалении друг от друга. Очень часто пространства находятся в различных локальных сетях, каждая из которых имеет выход в Интернет. Возникают две проблемы – организация пересылки изменений и их синхронизация.

Для осуществления пересылки можно воспользоваться стандартными средствами Интернета – электронной почтой или утилитой ftp. Для облегчения синхронизации следует завести два дополнительных пространства-порта и выполнять их синхронизацию

(например, при помощи семафоров). Утилиты, выполняющие пересылку, должны тщательно сверять контрольные суммы, чтобы убедиться в корректности полученной информации. Естественно, в пересылаемый пакет информации входят только инкрементальные изменения (дельты), а не измененные файлы целиком.

6.2. Системы поддержки работы виртуальных групп

Принцип разделяемых знаний реализуется в виртуальных группах, т.е. организациях и группах людей, которые работают и общаются между собой в интерактивном режиме. Общение в виртуальных группах очень часто осуществляется через Интернет, а в качестве средств общения выступают традиционные и хорошо известные средства, такие как:

- видеоконференции;
- аудиоконференции;
- средства группового общения в реальном времени (чаты);
- средства группового общения с разным временем подключения (доски объявлений, форумы);
- электронная почта (включая списки рассылки, архивы и поиск по ним). Однако все перечисленные средства имеют определенные недостатки. Например, видео- и аудиоконференции не позволяют передавать файлы, а средства группового общения в Интернете не обеспечивают секретности от доступа любых других пользователей с одной стороны и позволяют разработчикам выступать анонимно с другой стороны.

Системы поддержки работы виртуальных групп должны хранить и обрабатывать информацию, а также помогать разработчикам осуществлять общение друг с другом. Сформулируем некоторые общие требования к таким системам:

- Необходимость регистрации пользователя при его входе в систему.
- Строго авторизованный доступ ко всем данным в системе.
- Сохранение всей переписки и прочих данных в архиве.
- Использование системы управления версиями текстов для программ и документации.

Классификация систем поддержки работы групп по предоставляемым ими возможностям общения была предложена Бобом Йохансеном (Bob Johansen).

• В одно и то же время, в одном и том же месте (same time, same place).. Это классический случай, когда разработчики имеют возможность встречаться в одном помещении в определенное время. Здесь оказываются полезными следующие системы поддержки:

o система упорядочивания доступа к данным для показов видеозаписей, слайдов и других документов;

o система проведения мозговых штурмов и процедур принятия решений. Система дает возможность вносить идеи в список сразу после их возникновения. Как правило, такие системы обладают возможностями для быстрого голосования.

• В одно и то же время, в разных местах (same time, different place). Системы должны обеспечивать общение в реальном времени. Для этого обычно применяются средства аудио- и видеоконференций.

• В разное время, в одном и том же месте (different time, same place). Это довольно редкая ситуация, когда существует комната группы, но разработчики не имеют возможности собраться в ней в одно и то же время. Разработчику должна быть предоставлена возможность доступа к репозиторию проекта и всей информации, доступной на данном этапе проекта.

• В разное время, в разных местах (different time, different place). Системы реализуют возможности ведения конференций, форумов и чатов с асинхронным подключением, возможность доступа к репозиторию проекта.

• Неограниченные возможности доступа к данным (anytime, anyplace). Эта категория оставлена автором классификации на дальнейшее развитие коммуникационных технологий, когда возможности доступа к данным станут практически неограниченными.

Обратим внимание на то, что часто тип систем поддержки работы виртуальных групп зависит от рода их деятельности.

• Группы управления. Управляющие работники компаний, находящиеся в удаленных друг от друга подразделениях, обычно используют видеоконференции.

• Группы проекта. Разработчикам требуется максимально возможное количество средств общения и доступа к репозиториям.

• Группы сопровождения. Группам необходимы средства связи и быстрый доступ к данным из любого места.

Приведем несколько наиболее известных систем поддержки виртуальных групп:

- Facilitate.com (компании Facilitate.com (http://www.facilitate.com/));
- Consensus@nyWARE (компании Soft Bicycle (http://www.softbicycle.com/));
- 3-2-1 IntraNet (компании Internet Media Inc.).

----------end----------