УВАЖАЕМЫЕ СТУДЕНТЫ! Изучите приведенную лекцию, законспектируйте основные сведения о технологиях JAVA и .NET

Ответы на вопросы, фотоотчет, предоставить преподавателю на e-mail: r.bigangel@gmail.com до 30.01.2023.

При возникновении вопросов по приведенному материалу обращаться по следующему номеру телефона: (072)111-37-59, (Viber, WhatsApp), vk.com: https://vk.com/daykini

ВНИМАНИЕ!!! При отправке работы, не забывайте указывать ФИО студента, наименование дисциплины, дата проведения занятия (по расписанию).

Лекция 3
Тема: Классы инструментальных средств разработки программных продуктов.

Цель: Ознакомится с категориями современных инструментальных средств.

Введение

В процессе разработки программного обеспечения используется большое количество самого разностороннего программного обеспечения (ПО). Данный курс лекций рассматривает когда и что используется на протяжении всего этапа разработки приложений.

Прежде чем приступить к рассмотрению средств разработки, которые могут быть применены для создания программ, необходимо определиться с основными понятиями, терминами, которые будут использоваться в статье. В соответствии с тематикой статьи базовым термином для нас, конечно же, является «средства разработки программ». Применительно к области разработки программного обеспечения данное определение может звучать следующим образом:

Средства разработки программного обеспечения – *совокупность приемов, методов, методик, а также набор инструментальных программ (компиляторы, прикладные/системные библиотеки и т.д.), используемых разработчиком для создания программного кода Программы, отвечающего заданным требованиям.*

С учетом данного определения термин «Разработка программ» будет звучать следующим образом:

Разработка программ – *сложный процесс, основной целью которого является создание, сопровождение программного кода, обеспечивающего необходимый уровень надежности и качества. Для достижения основной цели разработки программ используются средства разработки программного обеспечения.*

ПС, предназначенное для поддержки разработки других ПС, будем называть *программным инструментом* разработки ПС, а устройство компьютера,

специально предназначенное для поддержки разработки ПС, будем называть *аппаратным инструментом* разработки ПС.

Для простоты, все используемые инструментальные средства можно разделить на 4 группы. Рассмотрим подробнее каждую из групп.

Необходимые

Необходимые инструментальные средства - это те, без которых в принципе невозможно получить исполняемый код; К этой группе можно отнести:

- редакторы текстов;
- компиляторы и ассемблеры;
- компоновщики или редакторы связей (linkers);

Часто используемые

Это средства, использования которых, в отличие от необходимых, можно избежать. Но без них процесс разработки весьма затрудняется и удлиняется; Из часто используемых средств стоит назвать:

- утилиты автоматической сборки проекта;
- отладчики;
- программы создания инсталляторов;
- редакторы ресурсов;
- профилировщики;
- программы поддержки версий;
- программы создания файлов помощи (документации).

Специализированные

Эти инструментальные средства используются в исключительных случаях, решают довольно специфичные задачи:

- программы отслеживания зависимостей;
- дизассемблеры;
- декомпиляторы;
- программы отслеживания активности системы и изменений, происходящих в системе;
- программы-вериферы и контейнеры (создают виртуальную среду для отдельных классов программ, в которой можно исследовать поведение программы);

Интегрированные среды разработки

Интегрированные среды разработки содержат большую часть из приведенных выше программ и позволяют упростить процесс создания приложений. По большому счету, среда разработки - это программа, которая собирает вместе несколько инструментальных средств из первой и второй (а иногда и третьей) групп.

1. Основные средства, используемые на разных этапах разработки программ

В зависимости от предметной области и задач, поставленных перед разработчиками, разработка программ может представлять собой достаточно

сложный, поэтапный процесс, в котором задействовано большое количество участников и разнообразных средств. Для того, чтобы определить, когда и в каких случаях какие средства применяются, выделим основные этапы разработки программного обеспечения. Наибольший интерес для проблематики рассматриваемого вопроса представляют следующие этапы разработки:

1. Проектирование приложения.
2. Реализация программного кода приложения.
3. Тестирование приложения.

Здесь сознательно опущены этапы, связанные с написанием технического задания, планирования сроков, бюджета и т.д. Причина этого заключается в том, что на данных этапах, за редким исключением, практически не используются специфические средства разработки.

2. Средства проектирования приложений

На этапе проектирования приложения в зависимости от сложности разрабатываемого программного продукта, напрямую зависящего от предъявляемых требований, выполняются следующие задачи проектирования:

1. Анализ требований.
2. Разработка архитектуры будущего программного обеспечения.
3. Разработка устройств основных компонент программного обеспечения.
4. Разработка макетов Пользовательских интерфейсов.

Результатом проектирования обычно является «Эскизный проект» (Software Design Document) или «Технический проект» (Software Architecture Document). Задача «Анализ требований» обычно выполняется с использованием методов системологии (анализа и синтеза) с учетом экспертного опыта проектировщика. Результатом анализа обычно является содержательная или формализованная модель процесса функционирования программы. В зависимости от сложности процесса для построения данных моделей могут быть применены различные методы и вспомогательные средства. В общем случае для описания моделей обычно применяются следующие нотации (в скобках приведены программные средства, которые могут быть использованы для получения моделей):

- BPMN (Vision 2003 + BPMN, AcuaLogic BPMN, Eclipse, Sybase Power Designer).
- Блок-схемы (Vision 2003 и многие другие).
- ER-диаграмы (Visio 2003, ERWin, Sybase Power Designer и многие другие).
- UML-диаграмы (Sybase Power Designer, Rational Rose и многие другие).
- макеты, мат-модели и т.д.

Иногда, когда разрабатываемый программный продукт предназначен для автоматизации какой-либо сложной деятельности задача Анализа (Моделирования) выполняется до составления технических требований к будущему продукту. Результаты анализа позволяют сформировать обоснованные требования к той или иной функциональности разрабатываемой программы и просчитать реальную выгоду от внедрения разрабатываемого продукта. Более того, иного получается так, что по результатам анализа первоначальные цели и задачи автоматизации кардинально меняются или по результатам оценки

эффективности разработки и внедрения принимается решение продукт не разрабатывать.

Целью второй и третьей задачи из приведенного списка задач является разработка модели (описания) будущей системы, понятной для кодировщика – человека, который пишет код программы. Здесь огромное значение имеет то, какую парадигму программирования (парадигму программирования также необходимо рассматривать как средство разработки) необходимо использовать при написании программы. В качестве примера основных парадигм необходимо привести следующее:

- Функциональное программирование;
- Структурное программирование;
- Императивное программирование;
- Логическое программирование;
- Объектно-ориентированное программирование (прототипирование; использование классов; субъективно-ориентированное программирование).

Выбор её во многом зависит от сложившихся привычек, опыта, традиций, инструментальных средств, которыми располагает коллектив разработчиков. Иногда разрабатываемый программный продукт настолько сложен, что для решения ряда задач в разных компонентах системы используются разные парадигмы. Необходимо отметить, что выбор того или иного подхода накладывает ограничения на средства, которые будут применены на этапе реализации программного кода. Результатом решения данной задачи в зависимости от подхода могут быть (в скобках приведены программные средства, которые могут быть использованы для их получения):

- диаграмма классов и т.д (Ration Rose, Sybase PowerDisigner и многие другие).
- описание модулей структур и их программного интерфейса (например, Sybase PowerDisigner и многие другие).

Разработка макетов пользовательских интерфейсов подразумевает создание наглядного представления того, как будут выглядеть те или иные видеоформы, окна в разрабатываемом приложении. Решение данной задачи основывается на применение средств дизайнера, которые в данной статье рассматриваться не будут.

3. Средства реализации программного кода

На этапе реализации программного кода выполняется кодирование отдельных компонент программы в соответствии с разработанным техническим проектом. Средства, которые могут быть применены, в значительной степени зависит от того, какие подходы были использованы во время проектирования и, кроме этого, от степени проработанности технического проекта. Тем не менее, среди средств разработки программного кода необходимо выделить следующие основные виды средств (в скобках приведено примеры средств): • методы и методики алгоритмирования.

- языки программирования (C++,Си, Java, C#, php и многие другие);
- средства создания пользовательского интерфейса (MFC, WPF, QT, GTK+ и т.д.)
- средства управления версиями программного кода (cvs, svn, VSS).

- средства получения исполняемого кода (MS Visual Studio, gcc и многие другие).
- средства управления базами данных (Oracle, MS SQL, FireBird, MySQL и многие другие).
- отладчики (MS Visual Studio, gdb и т.д.).

4. Средства тестирования программ

Основными задачами тестирования является проверка соответствия функциональности разработанной программы первоначальным требованиям, а также выявление ошибок, которые в явном или неявном виде проявляются во время работы программы. Среди основных работ по тестированию можно выделить следующее:

- Тестирование на отказ и восстановление.
- Функциональное тестирование.
- Тестирование безопасности.
- Тестирование взаимодействия.
- Тестирование процесса установки.
- Тестирование удобства пользования.
- Конфигурационное тестирование.
- Нагрузочное тестирование.

Среди основных видов средств, которые могут быть применены для выполнения поставленных работ можно привести следующие:

- средства анализа кода, профилирования (Code Wizard – ParaSoft, Purify – Rational Softawre. Test Coverage – Semantic и т.д.);
- средства для тестирования функциональности (TEST – Parasoft, QACenter – Compuware, Borland SilkTest и т.д.);
- средства для тестирования производительности (QACenter Performance – Compuware и т.д).

Текстовые редакторы

Как вам должно быть известно, текстовые редакторы относятся к необходимым инструментальным средствам. Это так, потому что не имея текстового редактора, мы не сможем редактировать исходный текст на языке программирования. Важно понимать, что в этом контексте под текстовым редактором понимается не что-то вроде MicrosoftWord с его многообразием функций для форматирования текста, таблиц и т.п.. Текстовый редактор для программирования должен обеспечивать немного другой функционал:

- возможность сохранения документов в требуемом формате (поддержка нужных расширений и т.п.);
- возможность подсветки синтаксиса (выделение ключевых слов, констант и т.п.);
- возможность копирования, вставки, замены фрагментов текста;
- возможность автодополнения вводимого текста, отображения вариантов для вызываемых функций и т.п.;
- возможности автоформатирования вводимого текста (автоматически создавать отступы в теле функции и т.п.).

Важно понимать, что даже без всех вышеперечисленных функций, редактором можно пользоваться при написании программ, если он способен редактировать обычные текстовые документы. как-правило, это единственное требование, определяющее принципиальную пригодность текстового редактора. Всё остальное - не что иное как улучшения, направленные на повышение производительности труда опытных разработчиков, либо на облегчение написания приложений новичками.

Поскольку работа с текстовым редактором занимает значительную часть всего времени разработки, правильный выбор текстового редактора может существенно ускорить или замедлить процесс редактирования исходных текстов. Кроме того, практически все интегрированные среды разработки используют свой редактор. Все эти редакторы имеют общие черты, но есть и отличия, по которым и формируется окончательное мнение человека о среде разработки в целом. Таким образом, выбор редактора - индивидуален для каждого конкретно взятого человека. Пока одни прекрасно справляются с редактированием исходных текстов в vi (консольный текстовый редактор, довольно тяжелый для использования неподготовленным людям), другие не могут написать простейшее приложение в самом «навороченном» редакторе, который делает за них половину рутиной работы.

Если не рассматривать консольные редакторы (которые все еще популярны под операционными системами семейства UNIX), то можно с уверенностью сказать, что все редакторы, поддерживающие GUI (Graphicaluserunterface, графический интерфейс пользователя) имеют схожий пользовательский интерфейс.

Пожалуй, нет смысла описывать тут процесс открытия и сохранения документов в текстовом редакторе. Сделаем акцент на другой момент при работе с исходными текстами программ. Дело в том, что русские символы могут кодироваться по-разному. Существует несколько однобайтных кодировок (например: cp-1251, cp-866, koi8-r) и несколько двухбайтных (utf-8, utf-16). Получается, что если сохранить файл в одной кодировке, а открыть, подразумевая, что он в другой - русский текст сложно будет прочитать. Это - основной источник проблем при работе с текстовыми редакторами. Несмотря на повсеместный переход на UTF-8, проблема всё еще актуальна и нужно помнить о том, какая у вас кодовая страница и поддерживает ли её редактор. Таким образом, практически обязательным требованием для текстового редактора является поддержка разных кодировок текста и способность преобразования текста из одной в другую.

Компиляторы, интерпретаторы, компоновщики

Итак, у нас есть исходный текст, написанный на языке программирования. Он составлен всоответствиями с правилами языка и готов к дальнейшей обработке. Каким образом компьютер может выполнить этот исходный текст? Тут нам на помощь приходит программа, именуемая интерпретатор или компилятор. Обе эти программы выполняют схожие задачи (преобразование исходных текстов в машинный код), но подходят к ним по-разному.

Компилятор

Компилятор преобразует сразу весь исходный текст. На выходе, как правило, получается один или несколько файлов объектного кода. Будем считать, что объектный код - что-то вроде полуфабриката, который практически готов к употреблению процессором. В некоторых языках программирования, процесс компиляции может управляться так называемыми директивами компиляции - специальными указаниями о том, как компилировать отдельные блоки текста. В общей сложности, компиляция программы состоит из следующих этапов:

- Лексический анализ. На этом этапе весь текст исходного файла преобразуется в последовательность лексем.
- Синтаксический (грамматический) анализ. Последовательность лексем преобразуется в дерево разбора.
- Семантический анализ. Дерево разбора обрабатывается с целью установления его смысла — например, привязка идентификаторов к их декларациям, типам, проверка совместимости, определение типов выражений и т. д. Результат обычно называется «промежуточным представлением».
- Оптимизация. Выполняется удаление лишних конструкций и упрощение кода с сохранением его смысла. Оптимизация может быть на разных уровнях и этапах. Так же, оптимизация может заключаться в использовании группы команд более современного процессора.
- Генерация кода. Из промежуточного представления порождается код на целевом языке.

Таким образом, «пропустив» исходный текст черезкомпилятор мы (при отсутствии ошибок на этапах анализа) получим объектный код, который почти готов к выполнению. Компиляторы одних языков программирования (например, C) генерируют код для конкретного типа процессора (напримерIntel 80386), в то время как компиляторы других языков программирования (например Java) генерируют код для выполнения в специальной виртуальной машине. Почему компилятор не генерирует сразу готовый код?

Во-первых, чтобы была возможность создания программ на нескольких языках программирования. Часть файлов объектного кода может быть получена с использованием одного языка программирования, а остальные - с использованием другого.

Во-вторых, чтобы можно было компилировать большие и сложные программы. Чем больше приложение, тем больше в нем потенциально повторяющегося кода. Чтобы не писать и не компилировать этот код снова и снова, создают что-то вроде библиотеки функций, которая компилируется отдельно. Таким образом, сначала компилируются библиотеки, потом - оставшаяся часть программы.

Далее, в процессе компоновки, будет получен итоговый исполняемый файл, но об этом позже.

Интерпретатор

Интерпретатор тоже преобразует текст на языке высокого уровня, но делает это построчно. Как только первая строка преобразована - она тут же запускается на выполнение. У такого подхода есть как достоинства, так и недостатки.

Достоинства интерпретаторов:

- Бóльшая переносимость интерпретируемых программ — программа будет работать на любой платформе, на которой есть соответствующий интерпретатор.

- Как правило, более совершенные и наглядные средства диагностики ошибок в исходных кодах.
- Упрощение отладки исходных кодов программ.
- Меньшие размеры кода по сравнению с машинным кодом, полученным после обычных компиляторов.

Недостатки интерпретаторов:

- Интерпретируемая программа не может выполняться отдельно без программы-интерпретатора. Сам интерпретатор при этом может быть очень компактным.
- Интерпретируемая программа выполняется медленнее, поскольку промежуточный анализ исходного кода и планирование его выполнения требуют дополнительного времени в сравнении с непосредственным исполнением машинного кода, в который мог бы быть скомпилирован исходный код.
- Практически отсутствует оптимизация кода, что приводит к дополнительным потерям в скорости работы интерпретируемых программ.

Чтобы избавиться от части недостатков, для некоторых языков программирования применяют интерпретаторы компилируемого типа. Они построчно выполняют код, который создал компилятор. Если вы помните, выше мы говорили, что некоторые компиляторы дают на выходе код не конкретного процессора, а какой-либо виртуальной машины. Это как раз тот случай.

Компоновщик

Компоновщик, известный так же как редактор связей, (linker, linkeditor) занимается созданием готового исполняемого файла из «кусочков» объектного кода. Выше, в разделе компиляции мы рассматривали ситуацию, когда проще выполнить компиляцию по частям. Если был применен такой подход, то окончательную сборку выполняет компоновщик. Как-правило, компоновщик является частью компилятора.

Для связывания модулей компоновщик использует таблицы имён, созданные компилятором в каждом из объектных модулей. Такие имена могут быть двух типов:

- Определённые или экспортируемые имена — функции и переменные, определённые в данном модуле и предоставляемые для использования другим модулям;
- Неопределённые или импортируемые имена — функции и переменные, на которые ссылается модуль, но не определяет их внутри себя.

Работа компоновщика заключается в том, чтобы определить и связать ссылки на неопределённые имена в каждом модуле. Для каждого импортируемого имени находится его определение в других модулях и после этого упоминание имени заменяется на его адрес.

Обычно компоновщик не выполняет проверку типов и количества параметров процедур и функций. Если надо объединить объектные модули программ, написанные на языках со строгой типизацией, то необходимые проверки должны быть выполнены дополнительной утилитой перед запуском редактора связей.