**2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

Все системы контроля версий делятся на централизованные и децентрализованные. Соответственно, в централизованных системах контроля версий присутствует центральный сервер, на котором завязаны все остальные системы. В децентрализованных системах такой сервер тоже есть, но он играет только роль синхронизации работы, так как в принципе, копия версии с любого компьютера может стать основной.

Для проведения лабораторного практикума были выбраны обе системы контроля версий. На практике чаще всего используются децентрализованные, как более надёжные и простые в использовании. Тем ни менее, именно с использованием централизованных систем контроля версий на данный момент написано наибольшее количество программных комплексов, так как они исторически являлись первыми.

В БрГТУ.100529-07 91 00 приведены схемы взаимодействия программ.

**2.1 Обзор децентрализованных систем**

С приходом более совершенных IDE и языков программирования, вопрос взаимодействия с коллегами является ключевым для многих разработчиков. В основном чтобы решить эту проблему, начали появляться централизованные системы контроля версий (ЦСВ или CVS – Centralized Version Control systems). Примерами таких систем являются:

− CVS;

− Subversion;

− Perforce.

В них используется следующий метод сохранения информации: имеется один сервер, который хранит все версии файлов, а также некоторое количество клиентов, на которые поставляется информация с сервера. Диаграмма таких взаимоотношений представлена на схеме 2.1:



Схема 2.1 – Диаграмма взаимоотношений в VCS

Такой подход имеет много преимуществ, особенно относительно RCS. Основное из них в том, что разработчики могут работать с одним и тем же файлом одновременно. Кроме этого, есть ещё и другие плюсы, например, каждый разработчик в любой момент времени может знать, что делают все остальные; кроме этого, администраторы систем, имея точно такую же возможность, могут эффективнее управлять своими кадрами и всем продуктом в целом.

Но, несмотря на решение достаточно большого спектра проблем, у таких систем есть свои недостатки. Первый и наиболее существенный – единая точка отказа. Если сервер по каким-то причинам выходит из строя, то все работают только со своими локальными копиями, не имея возможности смотреть изменения других разработчиков.

**2.2 Обзор централизованных систем**

Чтобы решить эту проблему, и, в свою очередь забыть о проблемах RCC, существуют децентрализованные системы контроля версий (ДСКВ или DVCS – Distributed Version Control System). В качестве примеров можно привести следующие ДСКВ:

− git;

− Mercurial;

− Darcs;

− Bazaar.

Как следует из названия, в таких системах клиенты не просто забирают последнюю (или выбранную, но всегда одну) версию проекта: на самом деле они полностью копируют его содержание и историю. Таким образом, при таком подходе у системы появляются два больших преимущества перед VCS:

В любой момент времени разработчик может посмотреть состояние проекта в какой-то момент времени даже без наличия доступа к центральному серверу;

Каждая копия проекта это, по сути дела, полная резервная копия проекта. Таким образом, если сервер выйдет из строя, то любой разработчик сможет восстановить его состояние на любой другой машине.

Диаграмма взаимоотношений между клиентом и сервером представлена на схеме 2.2:



Схема 2.2 – Децентрализованная система контроля версий

Также при выборе системы контроля версий учитывается популярность того или иного инструмента. С целью определить наиболее используемую систему, в 2014 году был проведён опрос среди разработчиков сайтов под систему WordPress. Результаты представлены на рисунке 2.3:

Рисунок 2.3 – Популярность систем контроля версий для WordPress проектов

Основная причина в использовании git заключается в его популярности. Git является одним из основных инструментов, которые требуют заказчики, а также в том, что именно для работы с git написан самая популярная работа для работы с файлами – GitHub.

Основная причина в использовании Subversion заключается в необходимости поддержки уже существующего ПО.

**2.3 Обзор языков программирования сверх высокого уровня**

В данной курсовой работе для лучшего понимания процессов программирования при помощи систем контроля версий нам понадобится серия языков программирования, которые смогут предоставить более высокий уровень абстракции, нежели чем высокоуровневые.

Вообще, высокоуровневый язык программирования — это язык, разработанный для быстроты и удобства использования программистом. Основная черта высокоуровневых языков — это абстракция, то есть введение смысловых конструкций, кратко описывающих такие структуры данных и операции над ними, описания которых на машинном коде (или другом низкоуровневом языке программирования) очень длинны и сложны для понимания. Новой тенденцией является появление языков программирования немного более высокого уровня. Такого рода языки характеризуются наличием дополнительных структур и объектов, ориентированных на прикладное использование. Прикладные объекты, в свою очередь, требуют минимальной настройки в виде параметров и моментально готовы к использованию. Использование ультра-высокоуровневых языков программирования снижает временные затраты на разработку программного обеспечения и повышает качество конечного продукта за счет, опять-таки, уменьшения объёма исходных кодов. Распространено мнение, что программы на языках высокого уровня можно написать один раз и потом использовать на компьютере любого типа. В действительности же это верно только для тех программ, которые мало взаимодействуют с операционной системой, например, выполняют какие-либо вычисления или обработку данных. Большинство же интерактивных (а тем более мультимедийных) программ обращаются к системным вызовам, которые сильно различаются в зависимости от операционной системы. Например, для отображения графики на экране компьютера программы под Microsoft Windows используют функции Windows API, которые отличаются от используемых в системах, поддерживающих стандарт POSIX. Чаще всего для этих целей в них используется программный интерфейс X-сервера.

К настоящему времени создан целый ряд программных библиотек (например, библиотека Qt или wxWidgets), скрывающих несоответствия системных вызовов различных операционных систем от прикладных программ. Однако такие библиотеки, как правило, не позволяют полностью использовать все возможности конкретных операционных систем.

Сверхвысокоуровневый язык программирования (язык программирования сверхвысокого уровня, англ. very high-level programming language, VHLL) — язык программирования с очень высоким уровнем абстракции. В отличие от языков программирования высокого уровня, где описывается принцип «как нужно сделать», в сверхвысокоуровневых языках программирования описывается лишь принцип «что нужно сделать». Термин впервые появился в середине 1990-х годов для идентификации группы языков, используемых для быстрого прототипирования, написания одноразовых скриптов и подобных задач. К языкам сверхвысокого уровня также часто относят такие современные сценарные и декларативные (в частности функциональные) языки как Ruby и Haskell, а также Perl и предшествовавший ему мини-язык AWK.

Большой класс языков сверхвысокого уровня — это языки, используемые для специфических приложений и задач (то есть предметно-ориентированные). В связи с этой ограниченностью они могут использовать синтаксис, который никогда не используется в других языках программирования, например, непосредственно синтаксис английского языка.

Предметно-ориентированный язык (англ. Domain Specific language, DSL — «предметно-специфичный язык») — язык программирования, специализированный для конкретной области применения (в противоположность языку общего назначения, применимому к широкому спектру областей и не учитывающему особенности конкретных сфер знаний). Является ключевым понятием языково-ориентированного программирования.

Строго говоря, деление языков программирования на языки общего назначения и предметно-ориентированные весьма условно, особенно, если учесть, что формально любой протокол или формат файлов является языком. Существует масса языков общего назначения, применяемых в качестве предметно-ориентированных для определённых задач, и наоборот, предметно-ориентированных языков, применяемых в качестве языков общего назначения. Ярким примером является язык Си, разработанный в качестве кроссплатформенного ассемблера, но на практике применяемый гораздо шире. Язык ML, породивший целое семейство языков общего назначения (включая Haskell, ныне наиболее предпочитаемый разработчиками предметно-ориентированных языков в качестве базового), — изначально разрабатывался в качестве DSL для системы автоматического доказательства теорем LCF. Примером, показывающим условность классификации, служит язык БНФ (и компилятор с него Lex/Yacc): с одной стороны, это яркий пример метаязыка, с другой — он предназначен для одной конкретной задачи.

Ведущие исследователи языково-ориентированного программирования (Мартин Уорд, Пол Хьюдак, Валид Таха и другие) приводят следующие примеры предметно-специфичных языков в качестве классических:

− TeX/LaTeX для подготовки (компьютерной вёрстки) текстовых документов;

− Perl для манипулирования текстами;

− SQL для СУБД;

− Tcl/Tk для графического интерфейса пользователя;

− HTML и SGML для разметки документов.

− Verilog и VHDL для описания аппаратного обеспечения;

− Mathematica и Maple для символьных вычислений;

− AutoLisp для компьютерного моделирования (САПР);

− Prolog для задач, сформулированных в терминах исчисления предикатов;

− ML и Haskell для задач, сформулированных в терминах функций (Haskell временами определяется как DSL для денотационной семантики).

Также некоторые предметно-специфичные языки программирования являются сценарными.

Сценарный язык (язык сценариев, от англ. scripting language) — высокоуровневый язык программирования для написания сценариев (англ. script) — кратких описаний действий, выполняемых системой. Разница между программами и сценариями довольно размыта. Сценарий — это программа, имеющая дело с готовыми программными компонентами.

Согласно Джону Устерхауту, автору языка Tcl, высокоуровневые языки можно разделить на языки системного программирования (англ. system programming languages) и сценарные языки (англ. scripting languages). Последние он также назвал склеивающими языками (англ. glue languages) или языками системной интеграции (англ. system integration languages). Сценарии обычно интерпретируются, а не компилируются, хотя сценарные языки программирования один за другим обзаводятся JIT-компиляторами.

В более узком смысле под скриптовым языком может пониматься специализированный язык для расширения возможностей командной оболочки или текстового редактора и средств администрирования операционных систем.

В плане быстродействия скриптовые языки можно разделить на языки динамического разбора (sh, COMMAND.COM) и предварительно компилируемые (Perl). Языки динамического разбора считывают инструкции из файла программы минимально требующимися блоками, и исполняют эти блоки, не читая дальнейший код. Предкомпилируемые языки транслируют всю программу в байт-код и затем исполняют его. Некоторые скриптовые языки имеют возможность компиляции программы «на лету» в машинный код (т. н. JIT-компиляция).

По применению языки можно грубо разделить на три типа:

− командно-сценарные;

− прикладные сценарные;

− универсальные сценарные.

На рисунке 2.4 представлен поток компиляции и дизассемблирования, по которому можно отследить специализацию того или иного языка или множества языков:

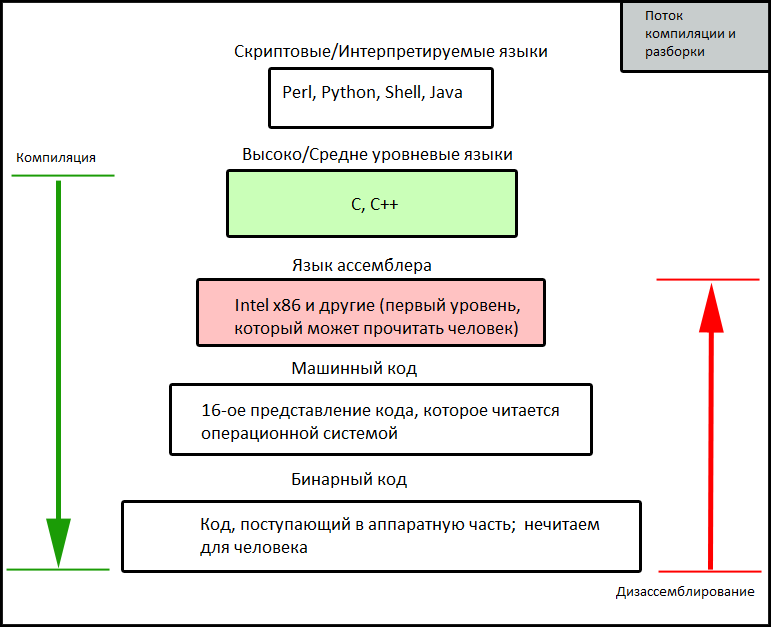


Рисунок 2.4 – Поток компиляции и дизассемблирования

Так как разрабатываемый курс лабораторных работ должен быть достаточно прост для понимая (в плане использованных языков программирования), то необходимо выбрать наиболее простые для изучения языки из выше представленных. Таким образом, для разработки в данном цикле лабораторных работ, были выбраны Tcl/tk и HTML. Также был выбран zenity – язык программирования (также может использоваться как – программа или утилита), который позволяет выводить на экран диалоговые окна GTK+ из командной строки и скриптов командной оболочки.

Особенностью всех выбранных языков является их относительная простота в освоении и дальнейшим использовании.