Некоммерческое акционерное общество
«АЛМАТИНСКИЙ УНИВЕРСИТЕТ ЭНЕРГЕТИКИ И СВЯЗИ»

Кафедра «Математического моделирования и программного обеспечения»
Специальность 6М070400 - «Вычислительная техника и программное обеспечение»

Допущен к защите
Зав. кафедрой

_____
«____»_____20__ г.

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ
пояснительная записка

Тема «Исследование парадигм программирования при междисциплинарных связях элективных дисциплин»

Магистрант_____ Колеаомешова А.С.
подпись (Ф.И.О.)

Руководитель диссертации_____ Керамбаев З.К.
подпись (Ф.И.О.)

Рецензент_____ Заурбеков Н.С.
подпись (Ф.И.О.)

Алматы, 2017 г.

Некоммерческое акционерное общество
«АЛМАТИНСКИЙ УНИВЕРСИТЕТ ЭНЕРГЕТИКИ И СВЯЗИ»

Факультет Аэрокосмических и информационных технологий
Кафедра «Математического моделирования и программного обеспечения»
Специальность 6М0704010 - «Вычислительная техника и программное обеспечение»

ЗАДАНИЕ

на выполнение магистерской диссертации

Конахметовой Ажарал Оралиевны
(фамилия, имя, отчество)

Тема диссертации :
Исследование парадигмы программирования при методах оптимизации сигналов электронных измерений

утверждена Ученым советом университета №___ от «___»_____
Срок сдачи законченной диссертации «___»_____
Цель исследования
Определение помехи электронных измерений анализ существующих методов преобразования, исследование парадигмы программирования

Перечень подлежащих разработке в магистерской диссертации вопросов или краткое содержание магистерской диссертации:
1. Изучить каналы электронных измерений специальности 6М070400 - «Вычислительная техника и прогр обеспечение»
2. Проведение численного эксперимента производительности объектов программирования

Перечень графического материала (с точным указанием обязательных чертежей)

Презентация (слайд)

Рекомендуемая основная литература

1. Морлак Курт Обзор 4 основных парадигм программирования
2. Peter Van Roy Programming Paradigms for dummies
3 Флойд Р.У Парадигмы программирования

ГРАФИК
подготовки магистерской диссертации

Наименование разделов, перечень разрабатываемых вопросов	Сроки представления научному руководителю	Примечание
1. Изучение различных парадигм программирования	01. 10. 2015 г	
2. Разработка программы для вычисления прерывистых и непрерывистых	01. 12. 2015	
3 Проведение численного эксперимента	01. 05. 2016	
4 Создание электронного каталога	01. 09 2016	

Дата выдачи задания _____

Заведующий кафедрой _____ (Байсалов М.Ж)
(подпись) (Ф.И.О.)

Руководитель диссертации _____ (Курманбаев З.А.)
(подпись) (Ф.И.О.)

Задание принял к исполнению
магистрант _____ (Конакбитова А.С.)
(подпись) (Ф.И.О.)

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Одним из многих преимуществ университета является то, что у вас есть возможность в выборе того, что вы хотите изучить. Элективные дисциплины позволяют вам быть придирчивыми и выбирать курсы университета, которые отвечают общим требованиям к образованию, помогают увеличить ваш GPA или заинтересовать у вас. Или они могут дать вам возможность исследовать новые увлечения и развить желаемые профессиональные навыки и способности. Выбор элективных дисциплин действительно очень важен. Он буквально строит всю будущую дорогу специалисту в его карьере. Задать правильный тепм, правильный план образования поможет преподавателям, студентам и даже самому университету поднять качество образования на высший уровень.

Актуальность темы. Для специальности 5В070400-«Вычислительная техника и программное обеспечение», целью которой является подготовка специалистов в области компьютерной техники и программировании программных продуктов, просто необходимы знания в области программирования. В школе при изучении языка, обучение начинает с элементарного – это изучение алфавита. При изучении языка программирования подход тот же. Но каждый язык программирования имеет свою особенность, например наличие команд обработки, функций, объектов, классов. То есть каждый язык программирования относится к той или иной парадигмы программирования. На данный момент существует 4 основных парадигм программирования, это императивная, логическая, функциональная и объектно-ориентированная парадигма.

Если изучить эти четыре парадигмы и знать отличительные черты каждой парадигмы то можно с легкостью определить особенности, отличительные черты каждой из них. Это поможет студентам с легкостью переходить от одного языка программирования к другому. Не обязательно учить язык полностью достаточно знать что функциональный язык, и что данный язык работает в режиме обработки функций.

Довольно очевидно, что таким образом студент не будет привязан лишь к одному языку программирования, а будет пробывать и другие, не потому что интересно, а потому что он будет знать как это с легкостью можно сделать.

В своей диссертации будут проведен анализ того как важно изучать элективные дисциплины и знать связь между ними при изучении другого. В данной работе будут приведены примеры, будет проанализирован существующий каталог элективных дисциплин и выявлены постреквезитные и пререквизитные связи между предметами.

Предмет исследования. Предметом исследования диссертационной работы является учебный процесс преподавания студентам элективных дисциплин по специальности 5В070400 «Вычислительная техника и программное обеспечение».

Объект исследования. Объектом исследования является каталог элективных дисциплин специальности 5В070400 «Вычислительная техника и программное обеспечение».

Целью диссертационной работы является доказать необходимость введения предмета «Парадигмы Программирования» в каталог элективных дисциплин и проведения численного эксперимента доказывающий какая из парадигм наиболее производительна.

Задачи диссертационной работы :

1. Изучение инструментов программирования для преподавания элективных дисциплин.
2. Провести анализ элективных дисциплин специальности 6М070400 – «Вычислительная техника и программное обеспечение»
3. Изучить пререквезиты и постреквезиты элективных предметов
4. Проведение численных экспериментов в выявлении наиболее производительных языков.

Научная новизна. Данного предмета нету в каталогах элективных дисциплин по данной специальности в университетах Республики Казахстан. Введение данного предмета в число элективных дисциплин поможет студентам быстро и качественно переходить от одного языка программирования к другому.

1 ИССЛЕДОВАНИЕ МЕТОДИКИ ПРЕПОДОВАНИЯ ЭЛЕКТИВНЫХ ДИСЦИПЛИН

1.1 Методика преподавания элективных дисциплин

В высшем образовании в Канаде и Соединенных Штатах курс является единицей обучения, который обычно длится один академический семестр, возглавляется одним или несколькими преподавателями (преподавателями или профессорами) и имеет фиксированный список студентов. Курс обычно является отдельным предметом. Студенты могут получить оценку и академический кредит после завершения курса.

В Соединенном Королевстве, Австралии и Сингапуре, а также в некоторых частях Канады курс представляет собой целую программу исследований, необходимых для завершения университетского диплома, а слово «единица» или «модуль» будет использоваться для обозначения академического курса как это упоминается в других частях мира, например, в Северной Америке и в Европе.

В Южной Африке, курс официально представляет собой совокупность всех курсов (в американском смысле, их часто называют «модулями») в течение года или семестра, хотя американское использование является обычным явлением[2]. На Филиппинах курс может быть отдельным предметом (обычно именуемым преподавательским составом и должностными лицами школы) или всей программой (обычно называемой студентами и аутсайдерами).

Курсы в американских университетах обычно ограничены во времени. Некоторые курсы длятся всего несколько недель, один семестр, последний учебный год (два семестра) и даже три семестра. Курс, как правило, специфичен для студентов, и инструктируется профессором.[1] Например, если человек проходит курс органической химии, то профессор будет учить студентов органической химии и тому, как это относится к их жизни. Курсы также можно назвать «факультативными». Факультативный, как правило, не обязательный курс, но есть определенное количество неспецифических факультативов, которые требуются для определенных специальностей.

Высшее образование в Казахстане состоит из высшей средней школы, профессиональной подготовки и университетского образования. Студенты поступают в высшее образование в возрасте 17 лет. В зависимости от выбранного курса и выбранного трека они могут оставаться в высшем образовании еще на 4 года или 5 лет. Среднее образование в Казахстане является бесплатным, а высшее образование может быть бесплатным в зависимости от итогового экзамена абитуриента и выбранного им специальности. Приоритет правительственного гранта на стороне инженерных специальностей. [7]

Курсы состоят из отдельных сессий, как правило, по фиксированному недельному графику.

Существуют разные форматы курсов в университетах:

– Курс лекций, где преподаватель читает лекции с минимальным взаимодействием;

– Семинар, на котором студенты готовят и представляют свою оригинальную письменную работу для обсуждения и критики;

– Коллоквиум или курс чтения, где инструктор присваивает показания для каждой сессии, которые затем обсуждаются членами;

– Учебный курс, в ходе которого один или несколько студентов работают над темой и еженедельно встречаются с преподавателем для обсуждения и руководства.

– Курс направленного индивидуального обучения, в котором студент просит создать и обозначить для себя область исследования, которая является более концентрированной и углубленной, чем стандартный курс;

– Лабораторный курс, где большинство работ проводится в лаборатории.

Многие курсы объединяют эти форматы. Курсы лекций часто включают в себя еженедельные дискуссии с небольшими группами студентов под руководством главного преподавателя, другого преподавателя или ассистен

преподавателя. Лабораторные курсы часто сочетают в себе лекции, дискуссионные секции и лабораторные занятия.[10]

Ожидается, что студенты будут выполнять различные виды работ для курса:

– Посещение курсов.

– Чтение и изучение показаний курса, указанных в курсе.

– Обсуждение материала, который они прочитали.

– Написание коротких и длинных документов на основе заданного чтения и собственных исследований в библиотеке.

– Выполнение домашних заданий или наборов задач.

– Завершение лабораторных занятий.

– Проведение викторин и экзаменов.

Точная работа зависит от дисциплины, курса и конкретного инструктора. В отличие от большинства европейских университетских курсов оценки обычно определяются всеми этими видами работ, а не только итоговым экзаменом.

Элективные дисциплины - это курс, выбранный учащимся по нескольким элективным предметам или курсам в учебной программе, в отличие от обязательного курса, который должен принять учащийся. Хотя обязательные курсы (иногда называемые «основные курсы» или «общеобразовательные курсы») считаются необходимыми для академической степени, элективные курсы, как правило, более специализированные. Количество курсов по выбору обычно меньше чем обязательных курсов. [12]

Термин «элективный» также используется для периода медицинского обучения, проводимого вне домашней медицинской школы, часто за границей. Мотивы выбора такой программы обычно является желание познакомиться с другими культурами и научиться работать в клинических ситуациях в других странах.

Как правило, университеты Северной Америки требуют, чтобы учащиеся получали как широту знаний по дисциплинам, так и глубину знаний в определенной выбранной предметной области, известной как основной. Таким образом, студенты гуманитарных или гуманитарных наук должны пройти некоторые научные курсы и наоборот. Как правило, учащиеся могут свободно выбирать свои факультативы из широкого круга курсов, предлагаемых их университетом, при условии, что студенты обладают необходимыми знаниями, чтобы понять предмет обучения. Студент специальности с основным предметом обучения английский, например, мог бы также изучать один или два года химии, биологии или физики, а также математику и иностранный язык.[6]

Элективные курсы также предлагаются в третьем и четвертом году обучения в университете, хотя выбор более ограниченный и будет зависеть от конкретной специальности, которую выбрал студент.

Целью образовательных программ высших учебных заведений является подготовка бакалавров как специалистов нового образования, способных решать социальные проблемы, творческие и профессиональные задачи образования и воспитания; обладают профессиональной компетенцией и могут творчески преподавать в учреждениях образования и воспитания.

По данным Бюро статистики труда США, две трети рабочих мест потребуют высшего образования к 2018 году. Это означает, что степени в колледже становятся критическими аспектами успешной карьеры. [11] В будущем вы не сможете получить работу своей мечты, если у вас нет формального образования, которое готовит вас к рабочему миру.

Чтобы получить максимальную отдачу от программы обучения, важно выполнить требования к курсу. Большинство университетов требуют сочетания обоих основных курсов, которые сосредоточены на вашей дипломной программе и выборных курсах, которые расширяют ваши интересы и навыки критического мышления.

К основным курсам относятся предметы, которые обучают необходимую и ценную информацию, которая напрямую связана с вашей областью обучения. Например, если вы находитесь в программе степени Учет, классы по налогообложению, коммерческому праву, финансовой отчетности и аудиту предоставят вам знания, необходимые для успеха в рабочей силе.

Элективные курсы - это курсы, которые относятся к вашей степени, но не имеют прямого отношения к вашей дипломной программе.[4] Вы можете основывать эти классы на своих интересах или брать несколько, которые помогут вам в области обучения и передачи вашей карьеры. Например, если вы работаете в программе «Компьютерная наука», и хотите узнать больше о бизнесе, чтобы стать успешным профессионалом, вы можете подумать о том,

чтобы взять вводный класс для бизнеса, чтобы развить мягкие навыки и понять, как Бизнес играет в технологической отрасли.

Эти выборные классы предоставляют студентам всестороннее образование, которое фокусируется на основополагающих предметах, таких как математика, английский и социальные науки. Они расширяют глобальную осведомленность студента, навыки критического мышления и навыки общения. Эти курсы предоставляют неоценимое понимание в областях от математики до публичных выступлений.

Курсы по выбору могут помочь вам развить навыки, которые вы хотите иметь, или расширить свои знания в специализированной области в своей области. Они могли бы просто удовлетворить ваше любопытство по поводу определенного предмета.

Во многих школах по всему миру студенты изучают основные предметы в школе - математику, английский язык и социальные исследования. Тем не менее, с ростом технологий и тем, как продвигается мир каждый год, специальные занятия для интересующихся студентов могут помочь по-разному - не только для школы, но и для студентов.[14]

Выбирая классы, отличные от обычных четырех предметов, учащиеся с большей вероятностью будут двигаться вперед, когда дело доходит до поступления в колледж и получения работы. Это также позволяет им больше ходить в школу. Существует много причин, по которым факультативы оказывают большое, положительное влияние на учащихся.

Часто студенты выбирают курсы, которые относятся к ним и их интересам. В нескольких исследованиях результаты показали, что у студентов больше шансов получить диплом в курсе, который они приняли в качестве выборного.

Есть много преимуществ для того, чтобы студенты проходили карьеру и элективные курсы. Кроме того, студент может найти то, что ему действительно интересно, о котором он никогда бы не подумал.

Когда вы начинаете учебу или магистерскую степень, вы сталкиваетесь с совершенно другой реальностью, когда речь идет о курсах, которые вы должны принять. Существуют основные курсы, которые каким-то образом связаны с программой и областью обучения, которую вы выбрали, и на выборных курсах, которые являются дополнительными и позволяют вам больше узнать о темах, выходящих за рамки конкретной программы, в которую вы участвуете, но что вы считаете интересным или важным для своего Профессиональной жизни.

С точки зрения выбора элективных курсов, их вы можете выбрать на основе своих интересов или на основе того, что может помочь вам в приобретении знаний в своей области обучения и предоставить вам критические идеи для вашей карьеры. Например, если вы находитесь в деловой программе, вы можете узнать больше об организационном поведении или психологии, чтобы подать заявку на работу, связанную с персоналом. Наличие выборки, связанной с вашей программой, может позволить вам развить критическое понимание и предысторию в том, как выровнять управление с

поведением людей и понять, как они чувствуют себя мотивированными и так далее.

Цель выборки - предоставить навыки, которые выходят за рамки основной темы вашей степени и могут позволить вам расширить ваши интересы. Кроме того, многие люди считают, что вы все равно должны принимать некоторые элективные курсы, даже если ваш курс не требует их, поскольку они могут быть очень важны для вашего образования.

Есть много причин, по которым вам следует выбирать курсы по выбору и тщательно планировать, какие из них вам нужны:

– С помощью выборки вы можете исследовать новые темы и области обучения, которые, в свою очередь, помогут вам познать свои новые интересы, о которых вы никогда не знали;

– Элективные курсы, которые находятся за пределами вашей основной программы, позволяют вам получить новую перспективу в своей собственной степени, дополняя ваши основные темы и в значительной степени влияя на ваш будущий карьерный путь.

– С академической точки зрения тот факт, что вы прошли некоторые курсы по выбору, может сделать вас более привлекательными для потенциальных работодателей, поскольку это может помочь вам дополнить ваше резюме, показывая, что у вас богатый образовательный опыт и потенциально приобретенные конкретные знания, необходимые в этой области.

– Основываясь на предыдущих моментах, факультативы расширяют ваши образовательные горизонты, предоставляют критические идеи по различным предметам и повышают ваши навыки мышления и обучения.

– В целом, факт заключается в том, что факультативы помогают вам основываться на ваших сильных основных ценностях, которые помогут вам быть более успешными в своей области.

Вы также можете найти недостатки в проведении выборных курсов. К счастью, однако, они, как правило, в значительной степени превзойдены преимуществами. Иногда студенты, которые сосредоточены и посвящены своим программам, чувствуют себя факультативными, которые не связаны с основными курсами, пустая трата времени. Да, временами, возможно, будет трудно понять смысл выбора свободного выбора. Например, если вы ориентируетесь на финансы, почему вы хотите принять выборку на социальном сетевом маркетинге? Если ничего другого, то факт, что вы принимаете факультативы, выходящие за пределы вашей области знаний, помогает вам разработать новые способы мышления, которые вы позже сможете применить, работая в своей конкретной области.

1.2 Взаимосвязь элективных дисциплин

Формирование каталога элективных дисциплин в дополнение к дисциплинам типового учебного плана является одним из этапов планирования учебного процесса. Типовые учебные планы при кредитной системе обучения

состоят из трех циклов дисциплин: общеобразовательных дисциплин; базовых дисциплин; профильных дисциплин. Каждый цикл включает перечень дисциплин обязательного компонента и компонента по выбору с указанием установленного объема кредитов. В компонент по выбору входят элективные дисциплины, предлагаемые высшим учебным заведением. Кредиты, отводимые на изучение компонента по выбору, распределяются вузами самостоятельно. Официальным документом, отражающим философию образования и модель подготовки специалиста, является учебно-методический комплекс специальности, который в своем составе помимо других имеет каталог элективных дисциплин. Этот каталог представляет собой перечень дисциплин, входящих в компонент по выбору, выбираемых обучающимися самостоятельно для создания возможности гибкого и самостоятельного всестороннего определения траектории обучения студента. В нем дается краткое содержание программ дисциплин специальности, указываются циклы дисциплин с соблюдением единой системы кодировки дисциплин, объем в кредитах, указываются пререквизиты и постреквизиты *(*Prerequisite, Postrequisite*)* – дисциплины, обязательные для освоения соответственно до и после изучения данной дисциплины Гибкость траектории изучения дисциплин обеспечивается, благодаря корректировкам с помощью пререквизитов и постреквизитов.[6]

Проведем анализ между двумя элективными дисциплинами специальности «Вычислительная техника и программное обеспечение» программы бакалавриата: объектно-ориентированное программирование и технология разработки программного обеспечения.

Т а б л и ц а 1 - Сравнение элективных дисциплин

Объектно-ориентированное программирование	*Технология разработки программного обеспечения*
Пререквезиты: Алгоритмизация и программирование, Технология программирования (Технология разработки программного обеспечения) **Постреквезиты:** Проектирование баз данных (Серверные базы данных).	**Пререквезиты**: Математический анализ, Алгоритмизация и основы программирования. **Постреквезиты:** Современные методы и средства программирования. JAVA, Современные методы и средства программирования. NET, Машинно-ориентированные языки программирования, Объектно-ориентированное программирование. C++
Цель изучения: изучение современных технологий объектно –	**Цель изучения:** изучение современных технологий программирования

ориентированного программирования на основе Net	

Пререквизиты и постреквезиты двух этих предметов пересекаются между собой, допустим изучение второго предмета невозможно без изучения первого. Взаимосвязь элективных дисциплин очевидна, поэтому успешность обучения одного предмета зависит от того как был усвоен предыдущий. Допустим невозможно начать изучение предмета «Технология разработки программного обеспечения» без понятия что такое в принципе технология разработки, как разрабатывается программное обеспечение, при помощи каких инструментов, какие языки программирования использовать, какие методы, функции языков программирования можно применить при создании программного обеспечения? Ответить на данные вопросы студенту поможет предмет объектно-ориентированное программирование, главной целью которого является проектировать, программировать и отлаживать объектно-ориентированные программы на языке C++.

Рассмотрим следующие два предмета

Таблица 2 - Сравнение элективных дисциплин

Основы информационных систем	*Машинно-ориентированные языки программирования*
Пререквезиты: Информатика, Математический анализ.	**Пререквезиты**: Информационно-коммуникационные системы, Алгоритмизация и программирование
Постреквезиты: : Проектирование баз данных, Компьютерные сети, Современные методы и средства программирования. JAVA, Современные методы и средства программирования. NET, Защита информации и информационная безопасность, Основы информационной безопасности, Безопасность компьютерных сетей и систем, Защита инфокоммуникационных систем	**Постреквезиты:** Системное программирование, Интерфейсы компьютерных систем .
Цель изучения: ознакомление с основными понятиями теории информации, изучение моделей информационных	**Цель изучения:** приобретение студентами основополагающих знаний об основных понятиях машинно -

процессов и их организации на физическом и канальном уровне.	ориентированного программирования, а также применения возможностей современного программирования на ассемблере.

С первого взгляда кажется, что данные два предмета не имеют связи, если смотреть на пререквезиты и постреквезиты. Но если посмотреть на цели изучения, то можно увидеть очевидную связь. Предмет «Машинно-ориентированные языки программирования» изучает само «железо», то на что изначально садиться процессор, вшивается машинный язык программирования то бишь ассемблер, на что устанавливается операционная система, на которую «одевают» интерфейс. При помощи теории информации мы узнаем как передаются сигналы процессору, для выполнения той или иной задачи, информация вбивается в виде каких то действий, передается как команда, и распознается как код двоичной-десятичной системы. Невозможно изучить одно не изучив другое. От того как грамотно распределяется очередность преподавания элективных дисциплин, зависит как успешно студент освоит следующую дисциплину.

1.3 Взаимосвязь элективных дисциплин по лабораторным работам.

При изучении одного предмета, студент обязан выполнить обязательный ряд задач в виде лабораторный, расчетно-графический и курсовых работ. При выполнении лабораторных работ перед студентом ставится конкретная задача освоение одной из тем. Парой в одной лабораторной, можно проследить взаимосвязь нескольких предметов. Допустим при изучении предмета «Проектирования базы данных» студент изучает написание запросов на языке pl/sql. Человек, который не имеет никаких знаний о программировании, будет долго разбираться, как писать запросы на языке pl/sql. Здесь мы видим связь как минимум с двумя предметами, это «проектирование на алгоритмических языках» и «объектно-ориентированное программирование». Также при изучении предмета «Проектирования базы данных» студенту ставится задача, установка базы данных ORACLE на платформе LINUX. При чем ставится база не просто на операционную систему, а на сервер oracle linux. Без изучения предмета «Администрирование серверных систем», «Операционные системы» и «Основы операционных систем» студент не сможет даже сделать первый шаг, поставить базу на платформу. Изучение linux-а проходят на предмети «Операционные системы», студенты полностью изучают данную ОС, установка, настройка файловой системы, настройка сети, изучение команд терминала.

Даже в выполнении лабораторных работ, мы видим четкую связь между элективными дисциплинами. Данная связь не позволяет качественно выполнить одну, не изучив вторую.

1.4 Влияние взаимосвязи элективных дисциплин на качество образования

Будущее студента как специалиста в своей сферы зависит только от того как он изучил дисциплины во время обучения в университете. Допустим приведем пример по мне. На данный момент я работаю в банке. В основном работаю с базами данных, оптимизирую запросы, автоматизирую отчеты. Но порой сталкиваюсь с проблемами, когда нужно выполнить какое-то задание. Допустим связать систему BI(business intelligence) с сайтом Национального Банка . Для того чтобы это сделать мне нужно было изучить серверные системы, интернет технологии, программирование. Это три разных предмета, которые изучались на разных курсах в университете. Возможно, если бы эти три предмета имели бы какую-нибудь взаимосвязь при выполнении лабораторных работ, мне как специалисту в будущем не пришлось бы заново изучать три разных предмета.

Возможно, при написании методических указаний к лабораторным работам нужно учитывать взаимосвязь с другими предметами, писать какие-то рекомендации при выполнении (повторение, изучение). Наверное, данный способ упростил бы студентам, а в будущем специалистам, найти работу, в которой он смог бы раскрыть все свои приобретенные навыки.

2 ОБЗОР РАЗЛИЧНЫХ ПАРАДИГМ ПРОГРАММИРОВАНИЯ

2.1 Обзор различных парадигм программирования

Парадигмы программирования - это способ классификации языков программирования на основе их функций. Языки можно разделить на несколько парадигм.

Некоторые парадигмы затрагивают главным образом последствия для модели исполнения языка, такие как разрешение побочных эффектов или последовательность действий, которые определяются моделью выполнения. Другие парадигмы касаются главным образом того, как организован код, например, группировки кода в блоки вместе с состоянием, которое модифицируется кодом. Тем не менее другие парадигмы отличаются в основном в основном стилем синтаксиса и грамматики.

```
fred(x) + fred(x) = 2*fred(x)
```

Это не обязательно относится к нефункциональному языку, например, в Pascal или C, Fred Может быть процедурой, которая имела побочный эффект, так что вызов его дважды имеет другой эффект от вызова его один раз. Например, Fred Может содержать задание:

$g:=g+1$

-где, g – это глобальная переменнаяю

Главной мотивацией написания является разработка программ, поведение которых легко анализируется математически и в любом случае легко предсказать и понять.

Тот же нефункциональный аспект сохраняется и для Java. Вызов метода fred (x) обычно будет иметь побочный эффект.

Однако было сложно разработать языки под функциональной парадигмой, которые производят программы, которые работают так же быстро, как и в соответствии с императивной парадигмой. Благодаря высокой производительности современных компьютеров это имеет меньшее значение для многих приложений, чем возможность писать правильные программы. Функциональную парадигму сложно реализовать эффективно, потому что, если место хранения когда-то используется для хранения значения, это не очевидно, когда ее можно повторно использовать - компьютер, на котором запущена программа, использующая функциональную парадигму, может потратить много усилий, чтобы определить возможность повторного использования магазина.

Еще один способ думать о функциональной парадигме - рассматривать ее как способ максимально ограничить советы, чтобы избежать вредных побочных эффектов в программе.

Императивная парадигма

Языки, которые отражают эту парадигму, распознают факт, что компьютеры имеют повторно используемую память, которая может изменить состояние. Таким образом, они характеризуются утверждениями, которые влияют на состояние машины, например:

```
x := x+1
```

Это можно понять только математически, сопоставив последовательность значений с x, скажем, x1, x2, ..., где xt обозначает значение, которое переменная x имеет в некоторый момент t. Таким образом, приведенное выше утверждение может быть переведено в математику как

$$x_{t+1} = x_t + 1$$

Подобные рассуждения обсуждаются в CS250. Это все труднее делать, поскольку изменения состояния становятся все более сложными (например, путем назначения компонентов структуры данных). Однако императивные языки можно относительно легко перевести на эффективный машинный код, и поэтому они обычно считаются высокоэффективными. Многие люди также считают императивную парадигму более естественным способом выражения себя.[10]

Языки, которые используют императивную парадигму, обычно имеют функциональные особенности – например, основные функции арифметики (сложение, вычитание ...).

Логическая парадигма

В то время как функциональная парадигма подчеркивает идею математической функции, логическая парадигма фокусируется на логике предикатов, в которой базовая концепция является отношением. Логические языки полезны для выражения проблем, когда неясно, какие функции должны быть. Таким образом, например, когда люди обеспокоены, естественно использовать отношения.

Например, рассмотрим отношения дяди: у данного человека может быть много дядей, а другой человек может быть дядей для многих племянниц и племянников.

Рассмотрим теперь, как мы можем определить отношение брата в терминах более простых отношений и свойств отца, матери, мужчины. Используя логический язык Prolog, можно сказать:

```
brother(X,Y)      /* X is the brother of Y                    */
    :-            /* if there are two people F and M for which*/
    father(F,X),  /*          F is the father of X            */
    father(F,Y),  /*    and   F is the father of Y            */
    mother(M,X),  /*    and   M is the mother of X            */
    mother(M,Y),  /*    and   M is the mother of Y            */
    male(X).      /*    and   X is male                       */
```

Математическая логика всегда играла важную роль в вычислении, поскольку логическая логика является основой конструкции логических схем, которые составляют основу любого компьютера. В логической парадигме мы используем более совершенную конструкцию, а именно предикатную логику, чтобы дать нам языки повышенной выразительной силы.

Объектно-ориентированная парадигма

Объектно-ориентированная парадигма (часто написанная О-О) фокусируется на объектах, представляемых программой, и позволяет им демонстрировать «поведение». Это контрастирует с типичным подходом в императивной парадигме, в которой, как правило, думают о работе с данными с помощью процедур. В императивной парадигме обычно данные пассивны, процедуры активны. В парадигме О-О данные объединяются с процедурами предоставления объектов, которые тем самым становятся активными. Например, в императивной парадигме можно написать процедуру, которая печатает различные виды объектов в программе. В парадигме О-О каждый объект имеет метод печати, и вы «скажите» объект для печати.

Тем не менее, можно использовать определенные не-объектно-ориентированные языки для написания объектно-ориентированных программ. Требуется способность создавать структуры данных, содержащие машинный код, или указатели на машинный код. Это возможно на языке С и на большинстве функциональных языков (где функции представлены как код).

Объекты принадлежат классам. Как правило, все объекты в данном классе будут иметь одинаковые виды поведения.

Классы обычно располагаются в какой-то иерархии классов. Эту иерархию можно представить как представляющую собой «вид» отношения. Например, вычислительной модели Университета может понадобиться классный человек для представления различных людей, входящих в университет. Подкласс класса может быть студентом; Студенты – это своего рода человек. Другой подкласс может быть профессором. И студенты, и преподаватели могут проявлять одинаковые виды поведения, поскольку они оба являются людьми. Например, они обе пьют и спят. Но есть разные виды поведения, которые характерны: например, профессора.

Язык схемы

Этот курс изучается с использованием языковой схемы, которая обеспечивает хорошую поддержку функциональной парадигмы, поскольку она содержит функциональное подмножество. Однако он также содержит императивные черты, которые означают, что мы можем взглянуть на императивную парадигму внутри Схемы. Это простой язык, но достаточно мощный, чтобы позволить нам легко реализовать расширения, которые иллюстрируют как логическую парадигму, так и объектно-ориентированную парадигму.

В частности, довольно легко реализовать объектно-ориентированную парадигму в Схеме, потому что функции, доступные на Схеме, могут быть легко присоединены к структурам данных Схемы, обеспечивающим фактически объекты, которые имеют поведение, связанное с присоединенными функциями.

2.2 Анализ языков программирования

Подобно тому, как программная инженерия (как процесс) определяется различными методологиями, также языки программирования (как модели вычислений) определяются разными парадигмами. Некоторые языки предназначены для поддержки одной парадигмы (Smalltalk поддерживает объектно-ориентированное программирование, Haskell поддерживает функциональное программирование), в то время как другие языки программирования поддерживают несколько парадигм (таких как Object Pascal, C ++, Java, C #, Scala, Visual Basic, Common Lisp, Scheme , Perl, PHP, Python, Ruby, Oz и F #). Например, программы, написанные на C ++, Object Pascal или PHP, могут быть чисто процедурными, чисто объектно-ориентированными или могут содержать элементы обеих или других парадигм. Разработчики программного обеспечения и программисты решают, как использовать эти элементы парадигмы.[4]

В объектно-ориентированном программировании программы рассматриваются как совокупность взаимодействующих объектов. В функциональном программировании программы трактуются как последовательность оценок функции без состояния. При программировании компьютеров или систем с большим числом процессоров в процессо-ориентированном программировании программы рассматриваются как наборы параллельных процессов, действующих на логически разделяемые структуры данных.

Многие парадигмы программирования хорошо известны тем методам, которые они запрещают, а тем, которые они позволяют. Например, чисто функциональное программирование запрещает использование побочных эффектов, в то время как структурированное программирование запрещает использование инструкции goto. Отчасти из-за этого новые парадигмы часто считаются доктринерскими или чрезмерно жесткими теми, кто привык к более ранним стилям. [5] Тем не менее, избегая некоторых методов, можно облегчить понимание поведения программы и доказать теорему о корректности программы.

Парадигмы программирования можно также сравнить с моделями программирования, которые позволяют задействовать модель выполнения, используя только API. Модели программирования также могут быть классифицированы в парадигмы, основанные на особенностях модели исполнения.

Для параллельных вычислений часто используется модель программирования вместо языка. Причина в том, что детали параллельного аппаратного обеспечения попадают в абстракции, используемые для программирования аппаратного обеспечения. Это заставляет программиста отображать шаблоны в алгоритме на шаблоны в модели выполнения (которые были вставлены из-за утечки аппаратных средств в абстракцию). Как следствие, ни один параллельный язык программирования не подходит для всех вычислительных задач. Таким образом, более удобно использовать базовый последовательный язык и вставлять API-вызовы в модели параллельного исполнения с помощью модели программирования. Такие модели параллельного программирования можно классифицировать в соответствии с абстракциями, отражающими аппаратное обеспечение, такими как разделяемая память, распределенная память с передачей сообщений, понятия места, видимые в коде, и так далее. Это можно рассматривать как разновидности парадигмы программирования, применимые только к параллельным языкам и моделям программирования.

Парадигмы программирования самого низкого уровня – это машинный код, который непосредственно представляет собой инструкции (содержимое программной памяти) в виде последовательности чисел, а языку ассемблера, где машинным инструкциям представлены мнемоники и адреса памяти, могут быть присвоены символические метки. Они иногда называются языками первого и второго поколения.

В 1960-е годы были разработаны языки ассемблера, поддерживающие библиотеку COPY, а также довольно сложные возможности генерирования условных макросов и предварительной обработки, CALL (подпрограммы), внешние переменные и общие разделы (globals), что позволяет значительно использовать код и изолировать его от специфики оборудования посредством использования логических операторов, таких как READ / WRITE / GET / PUT. [6]Сборка была и до сих пор используется для критичных по времени систем и часто во встроенных системах, поскольку она дает самый непосредственный контроль за работой машины.

Следующим шагом было развитие процедурных языков. Эти языки третьего поколения (первый из которых описывается как языки высокого уровня) используют лексику, связанную с решаемой проблемой. Например, Common Business Oriented Language (COBOL) – использует термины, такие как «файл», «перемещать» и «копировать».

FORmula TRANslation (FORTRAN) – с использованием терминологии математического языка, он был разработан в основном для научных и инженерных задач.

Язык ALGOrithmic (ALGOL) – ориентирован на то, чтобы быть подходящим языком для определения алгоритмов, используя терминологию на математическом языке и ориентируясь на научные и инженерные проблемы, как и FORTRAN.

Язык программирования One (PL / I) – гибридный коммерчески-научный универсальный язык, поддерживающий указатели.

Символический код инструкций для начинающих всех стран (BASIC) – он был разработан, чтобы позволить большему числу людей писать программы.

C ++ - это язык системного программирования, но он немного потерял свой путь с шаблонами, добавляющими чрезмерную сложность. Не слишком много людей выбрали C ++ для нового проекта, но большая часть программного обеспечения, которое вы используете каждый день, была написана на C ++ (ArcGIS, Windows OS, Firefox, MS Office и т. Д. И т. Д.), Поэтому он не уходит в ближайшее время.

C – является дедушкой семьи. Когда вам нужна максимальная производительность, вы используете C, это «близко к железу». Это здорово, если вам нужно закодировать драйвер устройства, но не отлично, если вам нужно создать веб-приложение. Многие живые проекты GIS с открытым кодом написаны на C, например, Very Awesome GDAL (Библиотека абстракции геопространственных данных).

SQL- используется как язык доступа к базе данных и управления. SQL лежит в основе многих операций GIS. SQL – отличный пример языка, который долгое время существовал. Почему это? Прежде всего, это декларативный, а не процедурный. То есть инструкции SQL говорят, что вы хотите, а не как вы хотите, чтобы это произошло. Поэтому детали реализации скрыты и могут меняться со временем. Это означает, что SQL будет оставаться актуальным в мире параллельной обработки, о котором мы поговорим в технологической тенденции этой недели.

Java - очень популярен для веб-программирования в целом, и это большой выбор языка программистов. Это один из претендентов на самые популярные языки GIS с открытым исходным кодом, используемые, например, в проектах GeoServer и JTS. Java является наиболее часто преподаваемым языком в университетах и, возможно, является нынешним королем холма для программирования языков в целом.

Python –часто используется как язык сценариев, хотя многие люди клянутся им и для более крупных систем. В настоящее время он получает большую видимость в качестве основного языка сценариев для ArcGIS. Это своего рода функции, такие как новый AML для ArcGIS, поскольку он обеспечивает API высокого уровня, чтобы получить все, что сделано. На мой взгляд, это был отличный выбор, потому что Python делает отличный «клеящий» язык, и с ним очень легко работать. Он имеет множество расширений, таких как SciPython и Numerical Python.

C # - был ответом Microsoft на Java и является флагманским языком программирования для .NET. Итак, если вы начали писать новое дополнение к ArcGIS, это, вероятно, лучший выбор.

Visual Basic.NET – VB.Net теперь является в основном альтернативным синтаксисом для той же среды выполнения C #. Вся сила и сложность, ни одного уважения. ☺

Flex – (от Adobe) имел репутацию одного из самых простых способов создания RIA (Rich Internet Application). API Esri Flex является одним из примеров популярной реализации, но Flex / Flash за последние пару лет выпала из-за популярности, поскольку браузеры и мобильные устройства отказались от плагина Flash.

JavaScript – текущий лидер для веб-интерфейсов пользователя. Google Maps является тяжелым пользователем JavaScript, а также является ведущим клиентом веб-карты с открытым исходным кодом (OpenLayers).

ActionScript – это язык, лежащий в основе Adobe Flash, который на некоторое время был ведущим методом разработки замечательных интерфейсов и визуализации на веб-странице. Как и в случае с Flex, он быстро упал с популярности, так как все меньше и меньше платформ поддерживают его на стороне клиента.

PHP – один из лучших способов взломать интерактивный сайт и, таким образом, довольно популярен.

Ruby – это более старый язык, который стал более популярным в последнее время. Ruby получил большую тягу благодаря Ruby on Rails, что упростило настройку приложения с поддержкой базы данных. Это было расширено для веб-карт GeoCommons (ссылка внешняя). Некоторые другие интересные сайты неогеографии, такие как OpenStreetMap и WeoGeo, также выполняются в Ruby.

Groovy, Jython, Scala – все это новые языки для виртуальной машины Java; У меня есть надежда, что они созреют в жизнеспособных вариантах для ГИС.

Avenue, VBA для ArcObjects – Зал стыда программирования ГИС. Авеню во многих отношениях красиво, поэтому мне было грустно, когда Эсри убил ее. И теперь они собираются убить VBA для ArcObjects, который никогда не был очень красивым, но был весьма полезен. Sniff.

2.3 Влияние парадигм программирования на качество образования

Изучение языка программирования для большинства учащихся в специальности вычислительной технике сродни обряда посвящения. Это важный переход, который признан недостаточным. С таким быстро изменяющимся миром, и наличием огромного количества инструментов, для специалиста важно уметь быстро и качество переходить с одного языка программирования на другой. Данная способность не достигается путем изучения большого количества языков программирования.

Языки программирования, как и естественные языки, имеют сходства, аналогии и они наследуют характеристики друг от друга. Если невозможно научиться десятку языков, вполне можно понять механизмы, которые вдохновляют и направляют на разработку и внедрение сотен различных языков. Это знание «частей» облегчает понимание «целого» нового языка и, следовательно, подкрепляет фундаментальную методологическую компетенцию в жизнь компьютерного специалиста, по крайней мере, насколько это позволяет им предвидеть инноваций и пережить технологии, которые устаревают.

Именно по этим причинам необходимо проводить курс по парадигмам программирования, так как знание основ языков программирования является во всем мире ключевым шагом на продвинутом уровне для профессионалов в области вычислительной техники(как в университете так и в работе).

Если студенты начнут свое обучение по специальности вычислительной техники с понятия что же в принципе такое программирование, какие основные виды языков программирования существуют то студентам будет легче осваивать все последующие дисциплины, которые в курсе у студентов.

Допустим императивная парадигма программирования, описывает вычисления в виде последовательности действия(инструкций), по этому свойству можно выделить языки, относящиеся к этой парадигме Fortra, C++, Pascal, Java, PHP. Особенностью структурной парадигмы программирования является наличие циклов и ветвления, к таким языкам программирования относятся Ruby, Basic, C#, JavaScript.

Процедурная парадигма программирования содержит в себе процедуры, то есть самостоятельный участок кода, которая выполняется как одна инструкция. Языки программирования, относящиеся к данной парадигме программирования C#, Java, Ruby, Python.

В модульном программировании, программы пишутся при помощи модулей, внутри которых описаны классы, функции или же императивный код. Языки программирования процедурной парадигмы Haskell, Pascal, Python.

Если посмотреть, то один и тот же язык программирования относятся к нескольким парадигмам. Не обязательно изучать все языки программирования, достаточно изучить основные особенности парадигм программирования.

3 РАЗРАБОТКА МЕТОДИКИ ПРЕПОДАВАНИЯ ЭЛЕКТИВНЫХ ДИСЦИПЛИН

3.1 Обзор существующих элективных дисциплин.

Учебно-методический каталог элективных дисциплин предназначен для увеличения эффективности и свойства подготовки будущих профессионалов методом классификации содержания и организации исследования учебной дисциплины с учетом достижений науки, техники и производства; улучшения методического обеспечения образовательного процесса; действенного планирования и организации самостоятельной учебной работы и контроля познаний студентов, оказания студентам методической помощи в усвоении учебного материала; оказания помощи педагогам в совершенствовании педагогического мастерства [13]. В связи с этим появляется вопрос о оптимальном подходе к формированию структуры УМКД. В отчете дан анализ вероятных подходов к формированию структуры учебно-методических комплексов учебных дисциплин в согласовании с требованиями ГОСО РК.

Согласно действующей в текущее время концепции образования неотъемлемой частью основной образовательной программы высшего образования являются Учебно-методические комплексы дисциплин (УМКД), которые разрабатываются на кафедрах, проходят процедуры согласования и утверждения, предусмотренные внутри-вузовскими Положениями, и обеспечивают преподавание учебных дисциплин в согласовании с требованиями ГОСО РК [14, 15, 16].

В текущее время принятого определения понятия УМКД нет, и в качестве рабочей дефиниции мы воспользуемся последующим определением «Учебно-методический комплекс дисциплины (УМКД) – это система нормативных документов и учебно-методических материалов, обеспечивающая студенту возможность усвоения содержания учебной дисциплины в соответствии с ГОСО, типовой и рабочей учебной программой по разработанной преподавателем технологии обучения» [17].

УМК дисциплины направлен на решение следующих задач [6]:

• определение места и роли учебной дисциплины в образовательной программе конкретной специальности;

• реализация междисциплинарных логических связей образовательной программы;

• распределение учебного времени по темам и видам учебных занятий;

• организация самостоятельной работы студентов в аудиторное и внеаудиторное время;

• активизация познавательной и творческой деятельности студентов;

• обеспечение взаимосвязи учебного и исследовательского процессов.

Анализ материалов, представленных на веб-сайтах вузов в открытом доступе, показал, что реализация требований ГОСО РК к структуре и содержанию УМКД осуществляется с учетом специфичности образовательного

процесса и сложившихся традиций в определенном вузе. С этой целью в каждом вузе Ученым советом принимается «Положение об УМКД», в каком детально прописываются рекомендуемая структура и содержание документа, порядок его разработки, утверждения и внесения конфигураций. Кафедры на основании принятого Положения сформировывают учебно-методические комплексы входящих в их компетенцию дисциплин, добавляя по мере надобности отдельные дополнительные элементы, отражающие специфику этих дисциплин. В нашем случае, необходимо разработать такую структуру, которая будет иметь все нужные компоненты с требованиями ГОСО РК, при этом будет удобна для внедрения в программное обеспечение.

Беря во внимание современные требования к содержанию, структуре и многофункциональным чертам УМКД, единственно рациональной и действенной в использовании формой представляется его электронная версия в виде некой локальной базы данных.

Более нередко применяемыми форматами электронных баз данных (ЭБД) при применении компьютерных технологий в сфере документооборота являются реляционные ЭБД и иерархические ЭБД. Выбор формата ЭБД определяется целями, задачками и основными требованиями к обозначенным структурам.

Целью электронной версий УМКД как базы данных является систематизированное представление совокупности нормативных, методических, учебных и контрольно-измерительных материалов, предусмотренных для использования в образовательном процессе по определенной образовательной дисциплине для данного направления подготовки бакалавров и магистров.

Специальность 5В070400 – «Вычислительная техника и программное обеспечение» включает в себя 23 элективные дисциплины по бакалавриату и 28 предметов по магистратуре. Например, для первого курса элективным дисциплинам относятся: технологии программирования и другие, а для второго курса это такими могут быть базы данных, объектно-ориентированное программирование и другие. Для третьего курса это — современные методы программирования, разработка приложений для мобильных устройств, и другие. Для четвертого курса это такими могут быть программирование web приложений и другие. Для магистрантов объем элективных дисциплин увеличивается. Между этими предметами имеются связи не только по пререквизитам и постреквизитам, но и по применяемым технологиям и по различным платформам.

Т А Б Л И Ц А 3 ЭЛЕКТИВНЫЕ ДИСЦИПЛИНЫ (по выбору) 5В070400 – “вычислительная техника и программное обеспечение”

	Цикл	**Цифровой код дисциплины**	**Наименование дисциплины**	**Семестр**	**Количество кредитов**

1 курс					
1	**ОД**	**1107**	Экологическая устойчивость и безопасность жизнедеятельности	**1**	**3**
		1107	Экологическая и техногенная безопасность	**1**	
2	**Д**	**1211**	Теория вероятностей и математическая статистика	**2**	**2**
		1211	Прикладная статистика	**2**	
3	**Д**	**1215**	Теория информации	**2**	**2**
		1215	Основы информационных систем	**2**	
4	**Д**	**1217**	Технологии программирования	**2**	**3**
		1217	Технологии разработки программного обеспечения	**2**	**3**
	икл	**Цифровой код дисциплины**	**Наименование дисциплины**	**Семестр**	**Количество кредитов**
2 курс					
1	**ОД**	**2106**	Политико-правовые и социально-духовные основы общества	**3**	**4**
		2106	Социальные институты современного общества: политика, право, религия	**3**	
2	**Д**	**2210**	Специальные главы математики	**3**	**3**
		2210	Прикладные разделы математики	**3**	
3	**Д**	**2212**	Дискретная математика	**3**	**3**
		2212	Математическая логика	**3**	
4	**Д**	**2213**	Теория электрических цепей. Линейные и нелинейные цепи	**3**	**3**
		2213	Теория электрических цепей. Методы расчета линейных и нелинейных цепей	**3**	
5	**Д**	**2214**	Инженерная и компьютерная графика	**3**	**2**

		2214	Основы компьютерного черчения	**3**	
6	**Д**	**2216**	Модели и методы теории управления	**3**	**2**
		2216	Методы оптимизации	**3**	
7	**Д**	**2218**	Объектно-ориентированное программирование.C++	**4**	**3**
		2218	Объектно-ориентированное программирование.C#	**4**	
8	**Д**	**2221**	Машинно-ориентированные языки программирования	**4**	**3**
		2221	Программирование на языке Ассемблер в среде Windows	**4**	**3**

	икл	**Цифровой код дис-циплины**	**Наименование дисциплины**	**Семестр**	**Количество кредитов**
3 курс					
1	**Д**	**3219**	Современные методы и средства программирования. JAVA	**5**	**3**
		3219	Современные методы и средства программирования. NET	**5**	
2	**Д**	**3305**	Операционные системы. Linux	**5**	**3**
		3305	Операционные системы. MAC	**5**	
3	**Д**	**3224**	Защита информации и информационная безопасность	**5**	**3**
		3224	Основы информационной безопасности	**5**	
4	**Д**	**3226**	Аналоговые устройства в цифровых системах	**5**	**3**
		3226	Микроэлектронные средства реализации аналоговых устройств	**5**	

5	**Д**	**3220**	Разработка программных приложений для мобильных устройств	**6**	**3**
		3220	Разработка программных приложений для операционных систем Android и IOS	**6**	
6	**Д**	**3223**	Беспроводные сетевые технологии	**6**	**2**
	Д	**3223**	IP телефония на базе Cisco	**6**	
7	**Д**	**3225**	Метрология, стандартизация и сертификация	**6**	**2**
	Д	**3225**	Метрология, стандартизация, сертификация и измерения на базе LabVIEW	**6**	
8	**Д**	**3227**	Управляющие вычислительные системы на основе микроконтроллеров	**6**	**3**
	Д	**3227**	Цифровые системы управления на основе микроконтроллеров	**6**	
9	**Д**	**3307**	Проектирование баз данных	**6**	**3**
	Д	**3307**	Серверные базы данных	**6**	
10	**Д**	**3209**	Теоретическая экономика и экономическая практика	**6**	**2**
	Д	**3209**	Казахстанская модель социально-экономического развития	**6**	
Блок 1.MVT 12-1 – Вычислительная техника					
11	**Д**	**3310**	Техника компьютерных и коммуникационных систем	**5**	**2**
12	**Д**	**3311**	Интерфейсы компьютерных систем	**6**	**3**
Блок 2.MVT 12-2 – Программное обеспечение					
13	**Д**	**3310**	Компьютерные сети	**5**	**2**

14	**Д**	**3311**	Проектирование и разработка пользовательских интерфейсов	**6**	**3**
	икл	**Цифровой код дисциплины**	**Наименование дисциплины**	**Семестр**	**Количество кредитов**
4 курс					
	Д	**4303**	Экономика и менеджмент	**7**	**2**
	Д	**4303**	Экономика и предпринимательская деятельность	**7**	
	Д	**4304**	Охрана труда	**7**	**2**
	Д	**4304**	Промышленная безопасность	**7**	
	Д	**4308**	СУБД Oracle	**7**	**3**
	Д	**4308**	Проектирование хранилищ данных на базе Oracle	**7**	
	Д	**4309**	Экспертные системы	**7**	**3**
	Д	**4309**	Системы искусственного интеллекта	**7**	
	Д	**4222**	Платформа облачных вычислений	**7**	**2**
	Д	**4222**	Разработка облачных бизнес-приложений	**7**	
	Д	**4306**	Защита инфокаммуникационных систем	**7**	**3**
	Д	**4306**	Безопасность компьютерных сетей и систем	**7**	
Блок 1.MVT 12-1 – Вычислительная техника					
	Д	**4312**	Архитектра и управление серверами	**7**	**3**
Блок 2.MVT 12-2 – Программное обеспечение					
	Д	**4312**	Программирование приложений Web. PHP	**7**	**3**

Т А Б Л И Ц А 2 ЭЛЕКТИВНЫЕ ДИСЦИПЛИНЫ (по выбору) 6М070400 – “вычислительная техника и программное обеспечение” (НАУЧНОЕ И ПЕДАГОГИЧЕСКОЕ НАПРАВЛЕНИЕ)

№ п/п	Цикл	Цифровой код дис-циплины	Наименование дисциплины	Семестр	Количество креди-тов
1курс					
1	БД	5205	Искусственные нейронные сети		4
		5206	Нейрокомпьютинг		
2	ПД	5312	Программные средства обработки графической информации		3
		5313	Технологии мультимедиа		
3	БД	5209	Проектирование микропроцессорных систем		4
		5210	Цифровая обработка сигналов и сигнальные процессоры		
4	БД	5207	Математическое и компьютерное моделирование		4
		5208	Анализ и моделирование бизнес-процессов		
5	ПД	5302	Методы анализа и синтеза проектных решений		3
		5303	Автоматизированные системы научных исследований и комплексных испытаний		
6	ПД	5310	Технологии высокоскоростных вычислений		4
		5311	Технологии параллельных вычислений		
№ п/п	Цикл дисциплин	Цифровой код дисциплины	Наименование дисциплины	Семестр	Кол-во кредитов
2 курс					

8	ПД	6304	Проектирование информационных систем с использованием современных СУБД		
		6305	Технологии защиты баз данных и баз знаний		
9	ПД	6306	Современные криптографические методы защиты информации		
		6307	Архитектура и инфраструктура технических средств защиты информации		
10	ПД	6308	Современные системы маршрутизации и коммутации в компьютерных сетях		
		6309	Сетевые технологии		

6М070400 – ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

(ПРОФИЛЬНОЕ НАПРАВЛЕНИЕ) ЭЛЕКТИВНЫЕ ДИСЦИПЛИНЫ (по выбору)

п/п	Цикл дисциплин	Цифровой код дис-циплины	Наименование дисциплины	еместр	ол-во кредитов
1курс					
	ПД	5302	Разработка программного обеспечения с использованием средств визуального программирования		
		5303	Проектирование пакетов прикладных программ с использованием WEB-программирования		
	ПД	5304	Проектирование информационных систем с использованием современных СУБД		
		5305	Технологии защиты баз данных и баз знаний		
	ПД	5306	Современные криптографические методы защиты информации		

		5307	Архитектура и инфраструктура технических средств защиты информации		
	Д П	5308	Микропроцессорные системы		
		5209	Цифровая обработка сигналов и сигнальные процессоры		
	Д Б	5205	Программные средства обработки графической информации		
		5206	Технологии мультимедиа		
	Д Б	5307	Математическое и компьютерное моделирование		
		5308	Анализ и моделирование бизнес-процессов		
	Д П	5310	Современные системы маршрутизации и коммутации в компьютерных сетях		
		5311	Проектирование и защита инфокоммуникационных систем		
	Д П	5312	Высокопроизводительные вычислительные системы		
		5313	Проектирование вычислительных систем на базе ПЛИС		
	Д П	5314	Системный анализ и принятие решений		
		5315	Технологии разработки распределенных приложений JAVA		

В данном каталоге присутствуют предметы, которые преподаются не в выпускающих кафедрах. В данном каталоге нету предмета парадигмы программирования, но есть довольно большое количество предметов в каталоге, который имеет непосредственное отношение к нему.

На 1 курсе бакалавриата это «Технологии программирования», на 2-ом курсе это «Объектно-ориентированное программирование.C++/C#», на 3-ем курсе «Современные методы и средства программирования. JAVA/NET», на 4-ом курсе «Экспертные системы». Каждый из предметов несет в себе той или иной язык программирования, язык программирования в свою очередь относится к определенной парадигме. Взаимосвязь или же разницу языков программирования и следовательно дисциплин помог бы выявить предмет «Парадигмы программирования». Этот предмет бы объяснял что есть 4

парадигмы, каждая парадигма имеет свою отличительную особенность, в каждой парадигме есть свои, определенные по особенностям, языки программирования. Это как начать преподовать русский язык, начав обучение без знания алфавита.

Посмотрим на примере.

1 курс – дисциплина «**Технологии программирования»**, изучаемые языки программирования – С, С++. Язык С относится к императивной парадигме, С++ к объкетно-ориентированной парадигме. Язык С – относится больше к машинному языку программирования, когда С++ включает в себя объекты и классы.

4 курс **– « Объектно-ориентированное программирование.C#»** , изучаемые языки программирования С++ и С# , оба данных языка отнросятся к объектно-ориентированной парадигме.

5 Курс – дисциплина «**Цифровые системы управления на основе микроконтроллеров**» , изучаемые языки программирования Ассемблер, относится к императивной парадигме программирования.

6 Курс – дисциплина **«Экспертные системы»**, изучаемый язык программирования Пролог, относящийся к логической парадигме прогаммирования.

7 Курс – дисциплина «**Системы искусственного интеллекта**», изучаемый язык программирования LISP, относящийся функциональной парадигме програмиирования.

То есть можно сказать, что в каталоге элективных дисциплин кафедры ММиПО есть дисциплины, которые охватывает все 4 вида парадигм программирования. Есть так же другие специальности, такие как *«Разработка программных приложений для операционных систем Android и IOS»*, при изучение которых необходимо знать парадигмы программирования, такие как : Императивная парадигма программирования- для того чтобы знать систему, в которую будет вшиваться приложение; объектно-ориентированная парадигма программирвоания – для написания программы которая сможет сооориентироваться под информационную систему устройства на которое будет создоваться мобильное приложение. То есть знание двух разных парадигм необходимо для изучения данной дисциплины. Дисциплина «парадигмы программирования» необходимв для того чтобы студенты могли четко различать языки программирования по своим особенностям, чтобы в будущем при изучении любой дисциплины, студенты могли с легкостью переходить от одного языка прогрммирования к другому.

3.2 Взаимосвязь инструментов и платформ для изучения элективных дисциплин

При изучении элективных дисциплин необходимо определить для себя инструментарий на котором будут работать студенты. На сегодняшний день существует огромное количество инстурментов для разработки. И очень важно

определить инструментарий, который будет выгоден по цене, и по техническим характеристикам, таким как : библиотека компонентов, технические характеристики для операционных систем, кроссплатформенность, поддерживаемость нескольких языков программирований.

Первым инструментом который приходит на ум, является приложение «Microsoft Visual Studio» всемиизвестной компании Microsoft.

Visual Studio – это полный набор средств разработки для создания веб-приложений ASP.NET, веб-служб XML, настольных приложений и мобильных приложений. Visual Basic, Visual C # и Visual C ++ используют одну и ту же интегрированную среду разработки (IDE), которая обеспечивает совместное использование инструментов и облегчает создание решений на смешанном языке. Кроме того, эти языки используют функциональность .NET Framework, которая обеспечивает доступ к ключевым технологиям, которые упрощают разработку веб-приложений ASP и веб-служб XML.

Рисунок 2 – Эмблема программы Microsoft visual studio

Microsoft имеет долгую историю с инструментами разработки (которые вы можете знаеть, если вы видели какой-либо телевизионный фильм или же читали какую-либо книгу про историю компании), а Visual Studio – естественная кульминация этих усилий. В течение ряда лет Microsoft поставляла отдельные средства разработки, такие как Visual C ++ и Visual Basic, но начиная с 1997 года они предложили программу разработки Visual Studio, объединив все эти среды в одно приложение. (Отдельные приложения по-прежнему доступны для покупки, но гораздо менее популярны).

Возможности приложения:

Что вы действительно можете сделать с Visual Studio? Ниже приведены некоторые из различных приложений, которые могут быть созданы с использованием Visual Studio.

– Консольные приложения: эти приложения запускаются из командной строки и не включают графический интерфейс, но отлично

подходят для небольших инструментов или всего, что будет выполняться другим приложением.

– Приложения форм Windows: это настольные приложения Windows, написанные с использованием платформы .NET; Поскольку они являются приложениями .NET, они требуют, чтобы платформа .NET была на любом компьютере, который будет запускать приложение.

– Службы Windows. – это приложения, которые запускаются в фоновом режиме во время работы вашего компьютера. Обычно это приложения, которые должны выполнять запланированные задачи или обрабатывать непрерывные сетевые запросы.

– Приложения ASP.NET: ASP.NET – это мощная технология, которая используется для создания динамических веб-приложений, часто управляемых базой данных. Многие популярные сайты написаны с использованием ASP.NET, в том числе гигантов электронной коммерции, таких как Dell.

– Веб-службы ASP.NET: ASP.NET предоставляет полную модель веб-сервисов, которая позволяет вам быстро и легко создавать веб-службы.

– Приложения для Windows Mobile: приложения Windows Mobile могут работать на устройствах, которые включают Compact Framework; К ним относятся устройства Pocket PC, а также мобильные телефоны, на которых установлена платформа Microsoft Smartphone.

– Приложения MFC / ATL / Win32: вы также можете создавать традиционные приложения MFC, ATL или Win32 с использованием C ++. Эти приложения не нуждаются во время выполнения .NET-среды выполнения, но также не включают многие из преимуществ работы с платформой .NET.

– Надстройки Visual Studio: Правильно, вы можете использовать Visual Studio для записи новых функций, которые будут добавлены в Visual Studio.

И еще: Visual Studio также включает проекты для развертывания вашего приложения, работы с базами данных, создания отчетов и т.д. Visual Studio предоставляет расширяемую модель для добавления новых проектов в Visual Studio; Многие другие приложения Microsoft теперь интегрируются непосредственно в среду IDE. Некоторые из наиболее распространенных включают SQL Server Reporting Services и Visual Studio Tools для Office.

Особенности.

Visual Studio призвана облегчить жизнь благодаря экономичным и удобным функциям; Вот некоторые из наиболее убедительных из этих функций.

– *IntelliSense:* IntelliSense является торговой маркой Visual Studio. IntelliSense просто помогает при программировании, показывая доступные классы и методы и свойства, доступные для этих классов.

– *Дизайн:* Visual Studio включает в себя визуальные дизайнеры WSYIWYG для приложений Windows, приложений ASP.NET и приложений для Windows Mobile. Эти дизайнеры значительно упрощают работу с приложением.

– *Отладка*: одна из самых важных функций Visual Studio – это возможность выполнять ваше приложение по очереди во время его выполнения. Не знаете, почему вы получаете сообщение об ошибке? Просто пройдите и посмотрите, что происходит не так.

– *Организация*: Visual Studio создана для разработки приложений, поэтому она предоставляет интуитивные методы для организации ваших различных файлов кода в проектах и ваших различных проектах в решения.

Visual Studio включает в себя слишком много функций, которые перечислены здесь; даже самые опытные разработчики не используют все функции, доступные в Visual Studio.

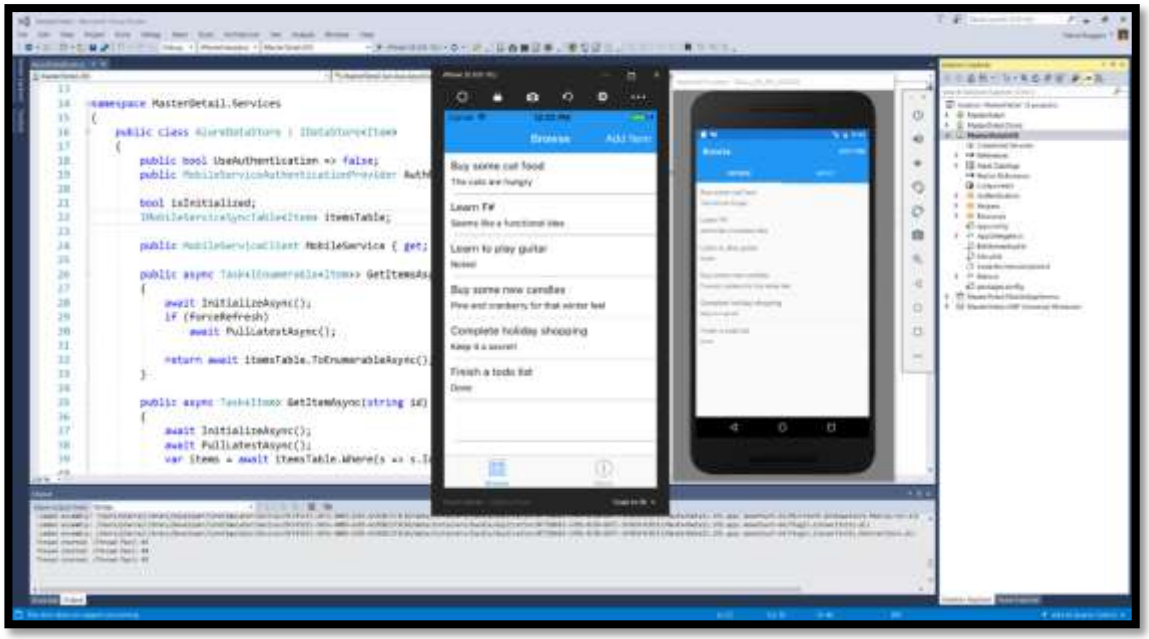

Рисунок 3 – Интерфейс Visual studio

```
Windows 7 SP1 (x86 и x64)
Windows 8 (x86 и x64)
Windows 8.1 (x86 и x64)
Windows Server 2008 R2 SP1 (x64)
Windows Server 2012 (x64)
Windows Server 2012 R2 (x64)


Требуемые компоненты: Internet Explorer 10


Требования к оборудованию:

Процессор с тактовой частотой 1,6 ГГц или большей
ОЗУ объемом 1 ГБ (1,5 ГБ для работы на виртуальной машине)
10 ГБ доступного пространства на жестком диске
Жесткий диск с частотой вращения 5400 об/мин
Видеоадаптер, соответствующий стандарту DirectX 9 и поддерживающий разрешение экрана 1024 x 768 или выше
```

Рисунок 4 – Минимальные системные требования

Eclipse

Eclipse – это платформа, разработанная с нуля для создания интегрированных инструментов для разработки веб-приложений и пользовательских приложений. По дизайну платформа сама по себе не обеспечивает множество функций конечного пользователя. Ценность платформы – это то, что она поощряет: быстрое развитие интегрированных функций на основе подключаемой модели.

Рисунок 5 – Логотип приложения

Eclipse предоставляет общую модель пользовательского интерфейса (UI) для работы с инструментами. Он предназначен для работы на *нескольких операционных системах*, обеспечивая при этом надежную интеграцию с каждой базовой ОС. Плагины могут программироваться на портативные API Eclipse и работать без изменений в любой из поддерживаемых операционных систем.

В основе Eclipse лежит архитектура для динамического обнаружения, загрузки и запуска плагинов. Платформа поддерживает логистику поиска и запуска правильного кода. Интерфейс платформы обеспечивает стандартную навигационную модель пользователя. Каждый плагин может сосредоточиться на выполнении небольшого числа задач. Какие задачи? Определение, тестирование, анимация, публикация, компиляция, отладка, диаграммирование ... единственным ограничением является ваше воображение.

Открытая архитектура

Платформа Eclipse определяет открытую архитектуру, поэтому каждая команда разработчиков плагинов может сосредоточиться на своей области знаний. Пусть эксперты хранилища построят обратные концы, а эксперты по юзабилити создают инструменты конечного пользователя. Если платформа хорошо спроектирована, новые возможности и уровни интеграции могут быть добавлены без влияния на другие инструменты.

Платформа Eclipse использует модель общего инструментария для интеграции инструментов с точки зрения конечного пользователя. Инструменты, которые вы разрабатываете, могут подключаться к рабочему месту, используя хорошо определенные крючки, называемые точками расширения.

Сама платформа построена в виде плагинов, каждый из которых определяет расширения для точек расширения плагинов нижнего уровня и, в свою очередь, определяет свои собственные точки расширения для дальнейшей настройки. Эта модель расширения позволяет разработчикам плагинов добавлять различные функциональные возможности на базовую платформу инструментов. Артефакты для каждого инструмента, такие как файлы и другие данные, координируются общей моделью ресурсов платформы.

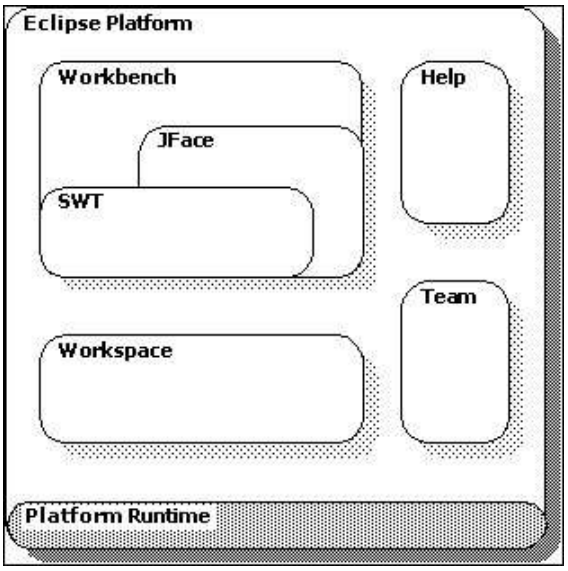

Рисунок 6 – Платформа Eclipse

Платформа дает пользователям общий способ работы с инструментами и обеспечивает интегрированное управление ресурсами, которые они создают с помощью подключаемых модулей.

Разработчики плагинов также получают эту архитектуру. Платформа управляет сложностью различных сред выполнения, таких как разные операционные системы или серверные среды рабочих групп. Разработчики плагинов могут сосредоточиться на своей конкретной задаче, а не беспокоиться об этих проблемах интеграции.

Платформа Eclipse сама по себе структурирована как подсистемы, которые реализованы в одном или нескольких плагинах. Подсистемы построены на основе небольшого механизма выполнения. На рисунке ниже изображен упрощенный вид.

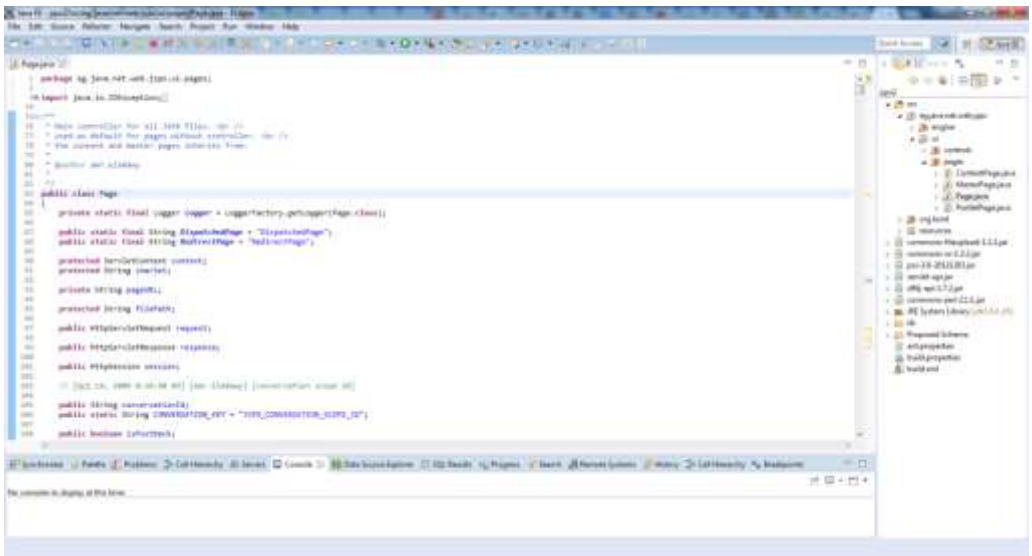

Рисунок 7 – Интерфейс программы Eclipse

Dev-C++

Dev-C ++ - это бесплатная интегрированная среда разработки (IDE) для программирования на C / C ++. Dev-C ++ разработан программным обеспечением Bloodshed. Он поставляется вместе с компилятором MinGW с открытым исходным кодом. MinGW использует GCC, сборник компиляторов GNU g ++. С Dev-C ++ вы можете легко писать Windows или консольные программы на C / C ++, вы даже можете создать установщик для своего приложения. Dev-C ++ размещен на Sourceforge. Текущая версия – 4.9.9.2 (т. Е. Бета-версия версии 5). Новостей о последних обновлениях для этой IDE нет. Кроме того, Dev-C ++ работает исключительно с окнами, порт linux больше не существует.

Dev-C ++ предоставляет интегрированную среду для написания программ. «Комплексная Среда «означает, что Dev-C ++ представляет собой комбинационную программу, состоящую из текстового редактора и C ++ компилятор. Текстовый редактор представляет собой ограниченную текстовую программу, которая позволяет вам вводить программы, вносить исправления и изменения, а также сохранять и извлекать вашу программу из дискового хранилища. Редактор интегрирован с компилятором Dev-C ++, так что вы можете легко переключаться между редактированием программы и ее компиляции и запуска. Обратите внимание, что компилятор Dev-C ++ позволяет вам

Рисунок 8 – Логотип приложения Dev C++

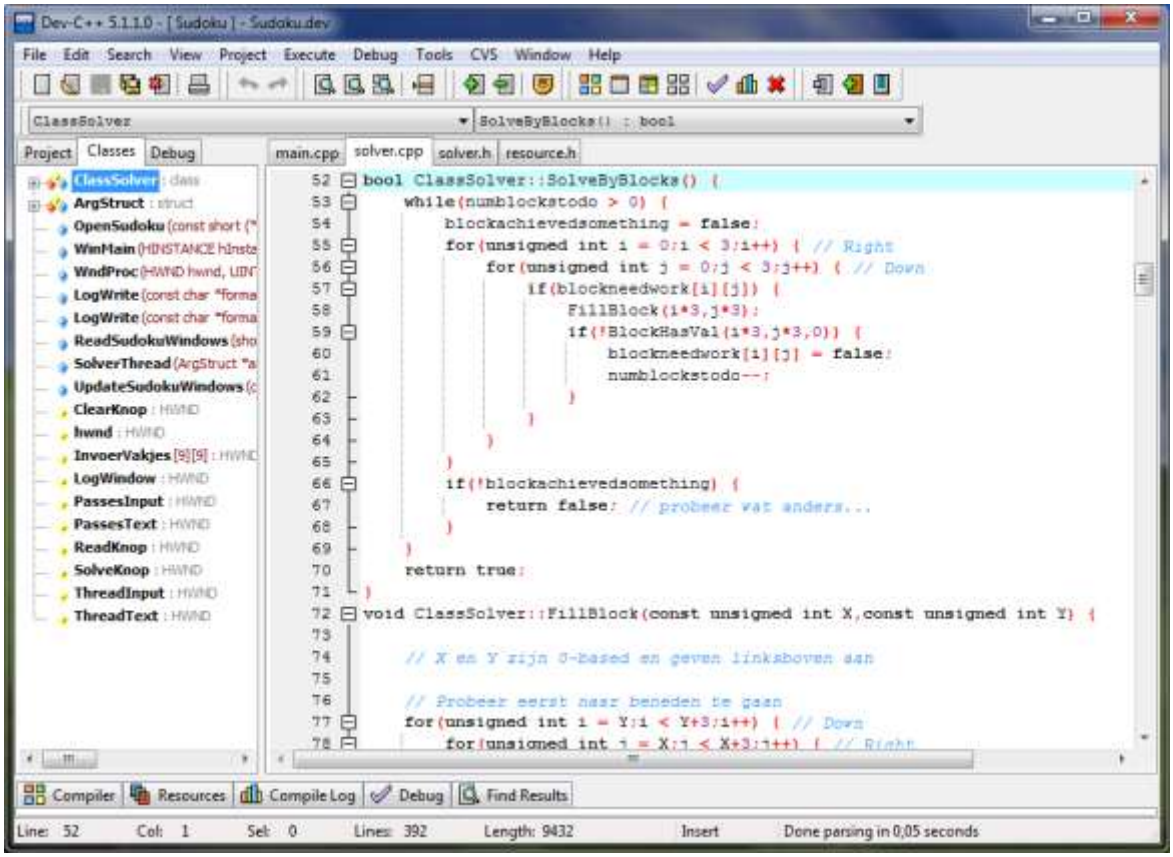

Рисунок 9 – Интрфейс программы dev c++

Среда разработки необходима для языков программирования, которые не могут использовать машинный компилятор, вшитый в систему. В основном это языки объектно-ориентированной парадигмы. Такие языки программирования как ассемблер и pascal могут работать и в командной строке windows, так как работают за счет машинного компилятора. Если смотреть по средам разработки, то наиболее удобной является Visual Studio, так как поддерживает библиотеки императивной, объектно-ориентированной функциональной парадигмы программирования.

Говоря о платформах нужно провести обзор существующих операционных систем.

Microsoft Windows

Windows является флагманской операционной системой Microsoft, стандартом де-факто для домашних и бизнес-компьютеров. Основанная на графическом интерфейсе ОС была введена в1985 году и с тех пор выпущена во многих версиях, которые описаны ниже. Microsoft начала свое сотрудничество с Биллом Гейтсом и Полом Алленом в 1975 году. Гейтс и Аллен совместно разработали Xenix (версию Unix), а также сотрудничали с BASIC-переводчиком для Altair 8800. Компания была основана в 1981 году.

Когда речь идет о операционной системе, Windows или Win – это операционная среда, созданная Microsoft, которая предоставляет интерфейс, известный как графический интерфейс пользователя (GUI), для компьютеров. Windows устраняет необходимость запоминания команд командной строки (MS-DOS) с помощью мыши для навигации по меню, диалоговым окнам, кнопкам, вкладкам и значкам. Если вы используете компьютер ПК (IBM), скорее всего, используете версию Windows. Если вы используете компьютер Apple, вы используете macOS.

Рисунок 10 – Логотип операционной системы Windows

Windows 8 – это операционная система Microsoft, выпущенная в 2012 году как часть семейства ОС Windows NT.

Windows 8 представляет собой серьезный отход от предыдущих версий Windows, поскольку он основан на языке дизайна Metro, который облегчает интерфейс пользовательского интерфейса с сенсорным экраном, подобный тем, что есть в мобильных телефонах и планшетных компьютерах. Планшет Microsoft Surface, выпущенный в начале 2013 года, запускает Windows 8.

Базовая версия Windows 8, подходящая для большинства пользователей, работает с сенсорными экранами, а также с традиционными системами клавиатуры и мыши. Версия под названием Windows RT поддерживает

архитектуру процессора ARM. Другая версия, Windows 8 Pro, ориентирована на бизнес и технических специалистов, которые хотят шифрования, виртуализации, управления ПК и подключения к домену.

Экран Windows 8 основан на плитке, каждый из которых представляет собой конкретное приложение. На панели задач нет кнопки «Пуск», хотя пользователи могут открыть экран «Пуск», коснувшись или щелкнув нижний левый угол дисплея. Предварительный выпуск Windows 8 содержит версию Metro Internet Explorer 10, а также приложения для новостей, спорта и путешествий. Он также предлагает обновленный проводник Windows, обновленный диспетчер задач и возможности доступа к изображениям.

Пользовательский интерфейс стиля Windows 8 Metro

Новая поверхность Metro с основанием на основе плитки основана на новой среде выполнения Windows (WinRT), которая работает как асинхронный API. Приложения для Metro могут быть запрограммированы в Javascript, C ++, C # и Visual Basic. Лучшее удобство использования Metro возможно только при использовании мультисенсорного экрана.

Одной из сильных сторон WinRT является простота масштабирования приложений на мобильных 7-дюймовых дисплеях, большие настольные мониторы до гигантских экранов.

Дизайн не включает в себя некоторые из известных пользовательских интерфейсов Windows Aero, таких как прозрачность, градиенты и тени. Элементы окна плоские, компактные и сведены к нескольким цветам. Закругленные углы ушли в прошлое.

Высокая производительность и частота кадров для приложений достигается за счет увеличения использования аппаратного ускорения в зависимости от поддержки процессоров и графических процессоров и DirectX 11.1.

Приложения, которые не работают на переднем плане, приостанавливаются через Windows 8, чтобы не потреблять процессорную мощность, пока приложение не будет повторно активировано.

Рисунок 11 – Интерфейс Windows 8

Минимальные системные требования:

- **Процессор.** 1 ГГц* или выше с поддержкой PAE, NX и SSE2
- **ОЗУ.** 1 ГБ (для 32-разрядной системы) или 2 ГБ (для 64-разрядной системы)
- **Место на жестком диске.** 16 ГБ (для 32-разрядной системы) или 20 ГБ (для 64-разрядной системы)
- **Видеоадаптер.** Microsoft DirectX 9 с драйвером WDDM

Операционная система Linux

Linux является одной из популярных версий операционной системы UNIX. Он является открытым исходным кодом, поскольку его исходный код свободно доступен. Он свободен в использовании. Linux был разработан с учетом совместимости с UNIX. Его функциональный список очень похож на список функций UNIX.

Рисунок 13 – Логотип операционной системы Linux

Компоненты системы Linux

Операционная система Linux имеет в основном три компонента:

– Ядро – ядро является основной частью Linux. Он отвечает за все основные действия этой операционной системы. Он состоит из различных модулей и взаимодействует непосредственно с основным оборудованием. Ядро обеспечивает требуемую абстракцию, чтобы скрыть детали аппаратного обеспечения низкого уровня в системных или прикладных программах.

– Системная библиотека. Системные библиотеки – это специальные функции или программы, в которых прикладные программы или системные утилиты обращаются к функциям Kernel. Эти библиотеки реализуют большинство функциональных возможностей операционной системы и не требуют прав доступа к коду модуля ядра.

– Системная утилита. Системные утилиты отвечают за выполнение специализированных задач индивидуального уровня.

Операционная система Linux

Режим ядра и режим пользователя

Код компонента ядра выполняется в специальном привилегированном режиме, называемом режимом ядра, с полным доступом ко всем ресурсам компьютера. Этот код представляет собой один процесс, выполняется в одном адресном пространстве и не требует какого-либо переключения контекста и, следовательно, очень эффективен и быстр. Ядро запускает каждый процесс и предоставляет системные сервисы для процессов, обеспечивает защищенный доступ к оборудованию для процессов.

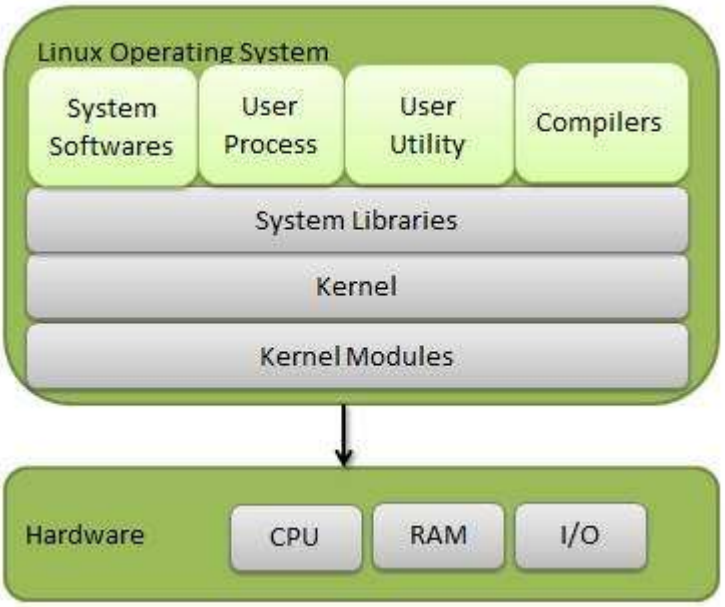

Рисунок 14 – Ядро операционной системы

Код поддержки, который не требуется для запуска в режиме ядра, находится в Системной библиотеке. Пользовательские программы и другие системные программы работают в пользовательском режиме, который не имеет доступа к системному оборудованию и коду ядра. Пользовательские программы / утилиты используют системные библиотеки для доступа к функциям ядра для получения низкоуровневых задач системы.

Основные характеристики

Ниже приведены некоторые важные функции операционной системы Linux.

– Portable – ПО для переносимости означает, что программное обеспечение может работать на разных типах оборудования одинаково. Ядро Linux и прикладные программы поддерживают их установку на любой аппаратной платформе.

– Исходный код с открытым исходным кодом – исходный код Linux свободно доступен, и это проект разработки на уровне сообщества. Несколько

команд работают в сотрудничестве, чтобы расширить возможности операционной системы Linux, и она постоянно развивается.

– Многопользовательский – Linux – многопользовательская система, так как несколько пользователей могут одновременно получать доступ к системным ресурсам, таким как memory / ram / application programs.

– Мультипрограммирование – Linux – это система мультипрограммирования, так как одновременно могут работать несколько приложений.

– Иерархическая файловая система. Linux предоставляет стандартную файловую структуру, в которой расположены системные файлы / пользовательские файлы.

– Shell – Linux предоставляет специальную программу интерпретатора, которая может использоваться для выполнения команд операционной системы. Он может использоваться для выполнения различных типов операций, для вызова прикладных программ. И т.п.

– Безопасность. Linux обеспечивает безопасность пользователей с использованием функций аутентификации, таких как защита паролем / контролируемый доступ к определенным файлам / шифрование данных.

Архитектура

На следующем рисунке показана архитектура системы Linux. Архитектура системы Linux состоит из следующих слоев:

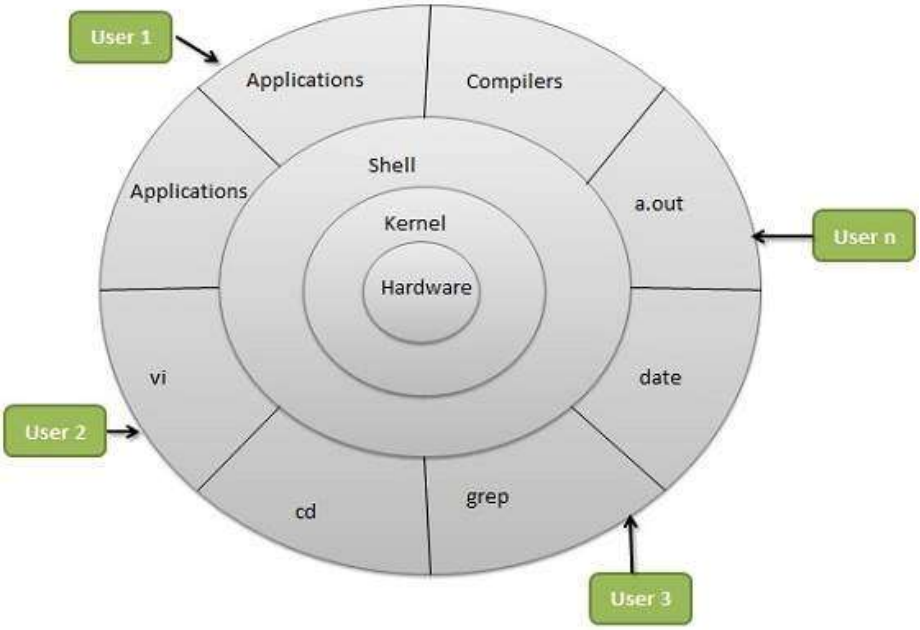

Рисунок 15 – Архитектура ядра ОС Linux

Аппаратный уровень. Аппаратное обеспечение состоит из всех периферийных устройств (RAM / HDD / CPU и т. Д.).

Ядро – это основной компонент операционной системы, взаимодействует напрямую с оборудованием, обеспечивает низкоуровневые сервисы для компонентов верхнего уровня.

Shell – интерфейс к ядру, скрывающий сложность функций ядра от пользователей. Оболочка принимает команды от пользователя и выполняет функции ядра.

Утилиты – служебные программы, которые предоставляют пользователю большую часть функциональных возможностей операционных систем.

Операционная система написана с использованием языка программирования. Операционная система представляет собой набор программ, написанных на определенном языке программирования. Да, мы можем управлять ядром с помощью языка программирования, потому что это сама программа, написанная на каком-то языке, и для ее изменения нам нужно изменить строки в программе. Теперь, по причине обучения Linux, люди изучают команды, которые понимает Linux, а не какой-то язык программирования. Каждая команда сама по себе является программой. Когда вы вводите команду, выполняется программа, имя которой совпадает с именем команды. Мы можем сделать набор таких команд, чтобы они выполнялись в предопределенной последовательности без вмешательства пользователя. Это делается для того, чтобы операционная система делала все, что захочет, без написания новых программ и с использованием существующих (команд). Вы можете создавать новые команды, используя язык программирования.

Язык программирования – это инструмент, который вы используете для преобразования программ с английского языка (или любого другого языка, который вы считаете), к тому, что может выполнить компьютер. (Программа не является кодом, это набор инструкций. «Шаги в« Как косить газон »образуют программу – косить газон для кого-то слишком глупого, чтобы сделать это без инструкций – например, о, скажем, компьютер.)

Операционная система – это действительно просто программное соединение между программами и оборудованием. (CP / M, одна из ранних микрокомпьютеров – или ПК – операционные системы, состояла из 3-х частей, BIOS, базовой системы ввода-вывода, чтобы отслеживать вход и выход, BDOS, базовую операционную систему диска, чтобы поддерживать Отслеживать файлы на диске и аппаратный интерфейс для подключения оборудования к BIOS и BDOS. (Также был встроен интерпретатор командной строки, чтобы определить, что означал yo8u при вводе «dir».)

Все остальное – программы, которые в конечном итоге переводятся на язык программирования, поэтому они могут запускаться (скрипты или интерпретируемые языки) или компилироваться в машинный язык и запускаться (или запускаться на виртуальных машинах, таких как Java). Веб-сервер, на котором вы используете эту страницу, возможно, Apache, - это программа, которая, вероятно, работает под Linux.

Зачем изучать Linux? Вы спрашиваете, зачем изучать API (интерфейс прикладного программирования – места в операционной системе, которые

программа вызывает, например, «поместить этого персонажа в это место на экране»), чтобы вы могли писать программы для запуска на компьютере Linux, Или вы спрашиваете, зачем изучать командную строку, чтобы вы могли делать что-то на Linux-компьютере с клавиатуры? Это две совершенно разные вещи.

Все знают о Windows, потому что он продается, когда мы покупаем компьютер. Windows построена на проприетарном устаревшем коде, что делает ее более открытой для вредоносного ПО и занимает больше памяти и места на жестком диске из-за необходимой защиты от вредоносных программ.

Mac OS и Linux очень похожи; Оба имеют корни в Unix, простой, но мощной и более безопасной операционной системе. Mac OS является собственностью, и она работает на своем оборудовании, что повышает цену.

Linux является открытым исходным кодом и свободен, поэтому сообщество пользователей может изучить его. Поскольку Linux работает на дешевом оборудовании, таком как ваш общий ПК с ОС Windows, поэтому он является явным победителем практически для всего. Но нужно самому загрузить и установить Linux на свой компьютер, процесс который занимает около 15 минут плюс время загрузки.

Linux имеет низкую долю продажи на рынке среди настольных и переносных компьютеров, где-то в десятках миллионов (никто точно не знает наверняка). Все остальное – суперкомпьютеры, телефоны, серверы, маршрутизаторы – сидят на Linux. Многие телефоны и планшеты работают на Android, где тоже используется база Linux.

Из-за этой низкой доли рынка трудно заставить разработчиков писать программы и драйверы, которые работают на чем угодно, кроме Windows – игр, в частности. Это меняется, поскольку все больше компаний видят преимущества Linux.

Существует несколько различных основных типов или «дистрибутивов» Linux и множество различных настольных сред, которые настраиваются для достижения внешнего вида и желания пользователя. (Windows может достичь большей части того же самого, но обычно требует дополнительных программ.)

3.3 Определение связи между платформами и средами разработки элективных дисциплин.

В своей диссертационной работе я провела анализ между элективными дисциплинами и нашла взаимосвязи при помощи постреквизитов и пререквизитов, парадигмов программирования, языков программирования а также при помощи сред разработки и платформ на которые они ставятся.

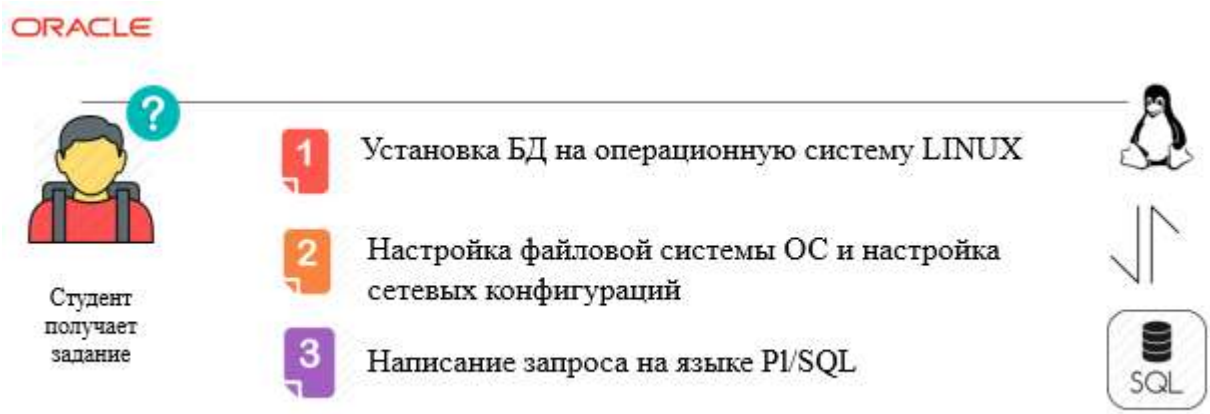

Рисунок 16 – Схема взаимосвязи элективных дисциплин.

Приведем пример взаимосвязи платформ и взаимосвязи сред разработки языков программирования между элективными дисциплинами. На 4-ом курсе специальности 5В040700 – «Вычислительная техника и программное обеспечение» изучается дисциплина «Проектирование баз данных». Данный пример я указывала в первом пункте данной диссертации, схематически наглядно видно, что при выполнении одной лабораторной по одной дисциплине, необходимо знание как минимум трех других дисциплин.

1. Установка операционной системы Linux – «Операционные системы».
2. Настройка конфигурации сети и связи между тонким и толстым клиентом – «Компьютерные сети».
3. Написание запроса в PL/Sql – «Технология программирования.»

Здесь есть взаимосвязь по платформам операционных систем.

Теперь расмотрим взаимосвязь как и по платвормам также и по средам разработкам.

Рассмотрим дисциплину «Разработка программных приложений для мобильных устройств»

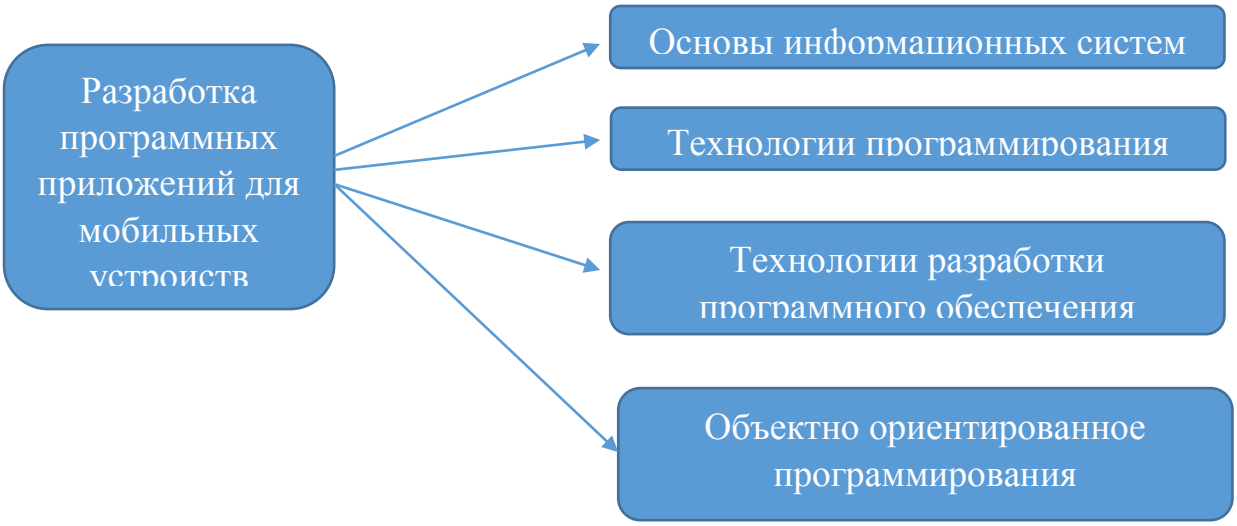

Рисунок 17 – Схема взаимосвязи элективных дисциплин.

Так как создание приложения, это создание некой информационной системы, объяснимо почему студенту необходимо изучать первую дисциплинцу. Первая дисциплина позволит студенту понять методы и модели информационных систем, понять все тонкости и все особенности информационных систем.

Технология программирования необходима для того чтобы студент изучил необходимые языки программирования для того чтобы знаь что же такое программирование и какие языки программирования существуют. Это поможет студенту определить какие языки программирования использовать студенту для создания мобильного приложения.

Технология разработки программного обеспечения позволит студенту понять что же такое программное обеспечение, для чего оно необходимо,как оно должно выглядеть какие функции оно должно нести.

Объектно-ориентировнное программирование эта дисциплина которая покажт возможности программировани, возможнсти объетно ориентированных языков, возможность классов, методов насследования и инкапсуляции.

Писать приложение можно как и на windows, так же и на linux. Конечно в плане производительности Linux выигрывает, но большинство студентов не могут совладеть с Linux-в первых порах поэтому работает на windows.

В качестве среды разработки можно использовать Microsoft Visual Studio, которое позволяет программировать как на C++, C#, Java , Php и т.д.

3.4 Теоретическое описание эксперимента и создание электронного каталога элективных дисциплин

В своей работе мы поставили задачу провести численный эксперимент и сравнить производительность языков программирования. Также мы попытались разработать каталок элективных дисциплин который будет определять технологии дисциплин, языки программирования и инструментов.

Понимание различий между языками программирования имеет решающее значение. Если для проекта выбран неправильный язык, потребуется много времени и усилий для изменения курса и повторного внедрения проекта или его части на другом языке. Как правило, это требует много усилий, несчастья и неудовлетворенности для всех: вас самих, ваших коллег, ваших клиентов и системных администраторов. Разумеется, это может быть опасно для бизнеса.

Знание того, как языки отличаются друг от друга, является ключом к принятию правильных решений. В средах могут быть разные требования к VPS с ограниченной оперативной памятью? Иногда нелегко отвечать на такие вопросы, учитывая множество ложных убеждений и слухов.

Это тестирование предназначено для демонстрации разницы между популярными языками программирования.

Надеюсь, вы считаете, что результаты этого небольшого исследования интересны.

Тестовый код вырабатывает текстовую строку, добавляя еще одну строку в цикле, пока она не вырастет до 4 мб. Каждая итерация заменяет некоторый текст. Каждый раз, когда строка становится 256 KiB, программа больше печатает количество секунд, прошедших с начала теста. Результат приложения передается по сценарию для захвата памяти (используя memstat) для каждой напечатанной строки.

Управление строками является основной функциональностью для всех языков, поэтому это позволяет корректно сравнивать языки. Обработка большой строки (-ов) разумно подчеркивает память, которая проявляет разницу между эффективностью языка.

Поскольку тестовый пример очень прост, легко реализовать его на разных языках аналогичным образом. Очевидно, что сам код не следует считать практичным, поскольку его единственная цель – создать некоторую вычислительную нагрузку для измерения. Образцы кода доступны для просмотра. Все реализации являются достаточно точными и понятными. Снова аналогичный объем работы, выполненный аналогичным образом, следует считать справедливым для сравнения.

Обработка строк была выбрана по многим причинам.

Большинство приложений не выполняют длительных вычислений. Для серьезных математических функций на любом языке недостаточно. Использование сторонних математических библиотек сделает сравнение несправедливым, несмотря на то, что сравнение библиотек будет бессмысленным, если мы хотим сравнивать языки, а не математические библиотеки.

Более того, целочисленные вычисления не подходят для тестирования, поскольку целочисленный размер может быть другим. Точность вычислений с плавающей запятой зависит от точности по умолчанию и может быть еще более зависимой от аппаратного обеспечения. Обработка строк необходима в каждом приложении, потому что строки – это просто данные. Больше данных означает больший стресс для сбора мусора и т. Д. Обработка больших строк легко сравнить, потому что все языки в этом тестировании будут выполнять ту же работу.

По сути, обработка строк очень распространена – XML (-RPC), HTML, журналы, сообщения, GUI – вся эта строка обработки на низком уровне, даже когда детали этого процесса скрыты от разработчика за API. Аппаратные средства не ускоряются. При обработке больших языков со строками делают много распределений памяти (re) и при необходимости копируют данные в память. Эффективность таких процессов является предметом этого тестирования, потому что он достаточно хорошо показывает, как разные языки.

Тесты выполняются достаточно долго, чтобы сравнить производительность и использование памяти, но не время, необходимое для запуска запуска. Вот почему запуск каждого теста один раз достаточно хорош для сравнения. Во время экспериментов я проводил каждый тест много раз и замечал лишь небольшое отклонение между результатами. Я считал, что эти отклонения являются пренебрежимо малыми (статистически незначимыми), поэтому окончательное сравнение составлено только из одного варианта тестового примера для каждого языка без сбора результатов нескольких тестов и сравнения их среднего значения. Помните, что точные цифры не слишком важны в этом тесте, потому что относительная разница проявляется очень хорошо.

Во время теста я сравнивала скорость, использование памяти и ухудшение производительности по мере роста обрабатываемых данных. Когда приложение борется с большим количеством данных, оно влияет на скорость обработки, которая важна для понимания.

Для тестирования использовались только функциональные возможности основного языка.

Изначально я хотела сравнить только основные языки перекрестной платформы, а именно PHP, Perl5, Python, Ruby и Java (Sun и OpenJDK). Тогда любопытство заставило меня включить C, C ++, Javascript («spidermonkey», Mozilla), Javascript («V8», Webkit), tcl, Lua и Java GCJ.

Хотя интересно сравнивать языки друг с другом, Javascripts, tcl и Lua выходят за рамки, поэтому я не буду сравнивать их функции.

Технически C и C ++ не должны принадлежать здесь, потому что они сильно отличаются от интерпретации языков по своей природе, однако их результаты важны для сопоставления.

Все тесты проводились на процессоре Intel Core2 Duo T7500@2.20Ghz; 2 ГБ оперативной памяти; ОС Debian GNU / Linux 2.6.32 i686

Во время тестов всегда было достаточно свободной памяти, чтобы в полной мере выполнять запуск теста без обмена и без использования ресурсоемких приложений. Однако более точные результаты могут быть собраны, если X сервер и большинство других процессов будут остановлены на время тестирования. Разница в выполнении одного и того же теста с обменом или без него или с более высоким приоритетом была ничтожно мала, если таковая имеется. Во время тестов управление питанием ЦП было отключено, поэтому оба процессора (ядра) работали с максимальной скоростью.

Значения по умолчанию использовались для всех языков, но PHP. По умолчанию PHP ограничивает максимальное использование памяти и максимальное время выполнения. Чтобы завершить тестирование, эти параметры необходимо было изменить в конфигурации времени выполнения PHP.

Время компиляции, необходимое для C, C ++ и Java, не учитывалось в этом тестировании.

Это сравнение состоит из трех частей:
Часть 1: Скорость.
Часть 2. Использование памяти.
Часть 3: Особенности языка.

Скорость.

Очевидно, что скорость выполнения важна для понимания языка. Если вы вообще не рассматриваете производительность, вам просто не нравится ваше приложение. Однако сама по себе производительность не является самой важной характеристикой, и поэтому следует учитывать и другие аспекты.

В этой таблице показано количество секунд, затраченных на выполнение Каждой стадии тестирования.

Line size Kb	Perl5	PHP	Ruby	Python	C++ (g++)	C (gcc)	Javascript (V8)	Javascript (sm)	Python3	tcl
256	2	6	7	7	7	2	3	30	17	33
512	7	23	29	32	26	8	21	131	81	141
768	16	54	75	78	60	19	51	300	201	324
1024	27	96	141	144	107	34	91	535	373	583
1280	43	153	225	232	167	53	144	842	598	921
1536	62	227	328	342	242	76	208	1220	877	1334
1792	84	318	452	476	329	104	283	1672	1211	1823
2048	109	424	597	634	431	136	370	2203	1598	2387
2304	139	549	758	815	546	173	469	2799	2039	3030
2560	171	691	941	1019	675	214	578	3463	2533	3753
2816	206	849	1143	1248	817	259	700	4198	3070	4553
3072	245	1022	1366	1497	972	309	834	4997	3659	5422
3328	288	1211	1607	1771	1142	363	979	5875	4300	6378
3584	334	1414	1869	2064	1324	423	1136	6825	4992	7409
3840	384	1634	2150	2381	1522	487	1304	7848	5729	8503
4096	437	1869	2455	2720	1731	555	1484	8928	6534	9680

Javascript (sm)	Python3	tcl	Lua	Java (openJDK)	Java (Sun)	Java (gcj)
30	17	33	49	39	38	451
131	81	141	203	162	157	1783
300	201	324	480	381	371	3937
535	373	583	886	711	696	6952
842	598	921	1423	1161	1145	10744
1220	877	1334	2090	1751	1739	15372
1672	1211	1823	2886	2489	2478	20819
2203	1598	2387	3856	3370	3358	27132
2799	2039	3030	4963	4453	4448	34302
3463	2533	3753	6198	5710	5719	42330
4198	3070	4553	7568	7146	7186	51118
4997	3659	5422	9084	8852	8983	60779
5875	4300	6378	10759	10784	10916	71275
6825	4992	7409	12594	12696	12867	82619
7848	5729	8503	14564	14861	15053	94686
8928	6534	9680	16674	17262	17426	107887

Рисунок 18 – Время выполнения кода в зависимости от размера и языка программирования

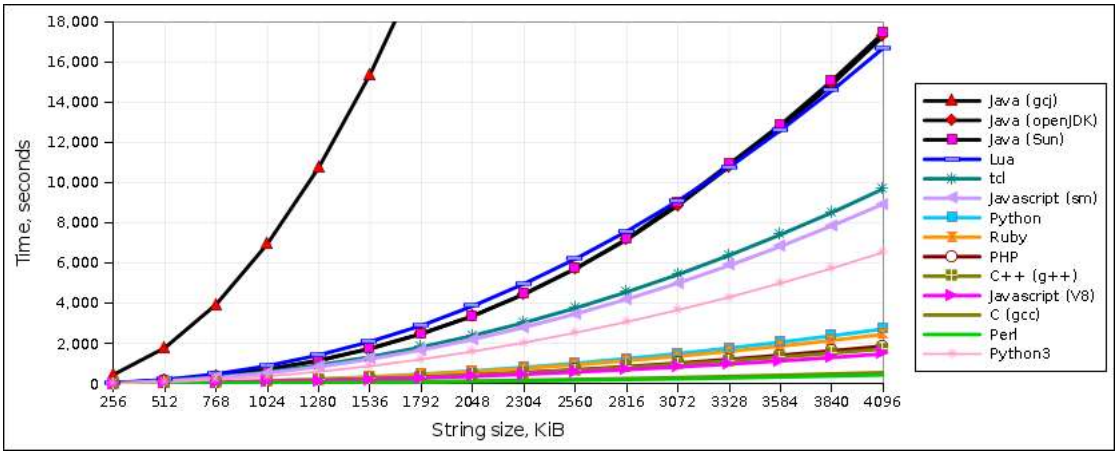

Рисунок 19 – график производительности

Как видно из графика производительности, скорость обработки замедляется по мере роста тестовой строки. Чем больше кривых графика, тем больше ухудшается производительность. График показывает, что производительность Java и Lua резко ухудшается.

Все проверенные языки хороши при манипулировании маленькими строками, но по мере того, как обработанные данные растут, разница проявляется.

Медленная группа [Java, Lua] страдает от серьезной деградации производительности.

В OpenJDK Java и Sun Java практически нет разницы в производительности. Производительность Lua очень близка к Java.

Первоначально GCJ Java-интерпретатор разбился во время теста, однако GCJ Java может скомпилировать Java-код в исполняемый файл, который завершил тест, хотя и ужасно медленный. Здесь и ниже неквалифицированная «Java» означает только основную Sun / OpenJDK Java.

Pyhon, Ruby и PHP немного медленнее, чем C ++. Это не удивительно, потому что эти языки оптимизированы достаточно хорошо.

Javascript V8 завершил тест чуть быстрее, чем C ++.

Эта группа языков показывает среднее замедление, тогда как производительность C и Perl5 является почти плоской линией на графике, что указывает на очень небольшое ухудшение. Это означает, что C и Perl5 обрабатывают увеличение количества данных на (почти) постоянной скорости.

Неожиданный результат: каким-то образом Perl5 удалось закончить быстрее, чем C. Это стало непредвиденным сюрпризом, который мне было трудно объяснить. Вероятно, Perl делает меньше перераспределения памяти для обеспечения роста строк.

Perl5 является явным победителем всего за 7 минут, необходимых для завершения теста против Java с худшим результатом, достигающим почти 5 часов, чтобы сделать то же самое. (Худший результат GCJ Java – почти 30 часов, не стоит сравнивать)

Perl5 не только превосходит по производительности, но и очень мало замедляет работу над большими данными. Это как можно ближе к C (скомпилировано для машинного кода), как это может быть для языка сценариев. Совершенно удивительно!

Интересно отметить, что при «использовании строгих»; Perl выполнил тот же тест ~ 6 секунд быстрее.

В приведенной ниже таблице Perl5 принимается за 1 и производительность других языков, измеренных в Perls, поэтому вы можете видеть, сколько раз медленнее определенного языка по сравнению с Perl5 в этом тесте. Из-за ухудшения производительности будет некорректно говорить что-то вроде «Это в два раза быстрее, чем это». Производительность некоторых языков ухудшается быстрее других, поэтому в начале этого теста Java несколько в 20 раз медленнее, чем Perl5, и в конце концов Java примерно в 40 раз медленнее (для такого же объема данных).

Очевидно, что это важная характеристика – размер имеет значение! Это соответствует наблюдению некоторых приложений Java, которые ведут себя хорошо под малой нагрузкой и экспоненциально деградируют по мере увеличения нагрузки.

Чтобы взять правильный инструмент для работы, важно понимать позицию программирования языков друг для друга. Трудное решение легче сделать, если вы считаете правильные вещи, избегая ненужных.

Есть некоторые вещи, не имеющие отношения к хорошему решению:

– Ваш любимый язык на данный момент.

– Вы можете быть очень хорошо и комфортно с языком, который вы уже знаете, но это не достаточно хорошо, чтобы не рассматривать альтернативы.

– Обучение важно.

– Личность создателя (ов) языка.

– Это просто не имеет значения, нравятся они вам или нет или даже они.

– Ваши ожидания относительно особенностей языка.

– Всегда нужно время, чтобы привыкнуть к новым вещам, особенно если они совершенно разные.

– Скорость обучения.

Некоторые языки имеют короткую начальную кривую обучения. Однако на самом деле это больше похоже на «минуту, чтобы учиться, а всю жизнь осваивать».

Эту идею лучше всего объяснил Питер Норвиг в своем учебном пособии «Обучать себя» в десятилетнем эссе. Есть некоторые вещи, которых следует избегать:

– Принимая во внимание одну единственную функцию языка.

– Рассмотрение только одной функции языка, такой как скорость или использование памяти, неизбежно приведет к неправильному решению.

– Узкие языки.

– Специализированные языки, такие как PHP, могут быть полезны только для веб-разработки. Когда вам нужно сделать что-то другое или просто расширить область действия, язык для конкретного использования может быть недостаточно хорошим.

– Непортативные языки.

– Кросс-платформенная переносимость имеет значение. Слишком много людей заперты, застряли в технологиях только с окнами, и у них мало шансов убежать.

– Бессрочная лицензия.

– Несвободные лицензии поставляются с рисками и ограничениями.

– Есть несколько важных вещей, которые следует учитывать:

– Доступность кода многократного использования

– Даже лучший язык в мире не стоит без хороших бесплатных библиотек.

– Бесплатная лицензия.

– Свобода очень важна, даже если вы не совсем понимаете, почему.

Универсальные языки.

Универсальные языки, такие как Perl5, обычно хороши практически для любой задачи. Универсальные языки более мощные по определению, что делает ваши навыки универсальными.

Хорошо переносимые языки

Спустя некоторое время программное обеспечение может быть перенесено на другую платформу или операционную систему. Переносимость гарантирует выбор. Выбор хорош.

«Чувствует себя хорошо»

По существу, ваши чувства к языку – это высшая заслуга его доброты для вас. Например, не всем людям может быть удобно с Python, но, если вы в порядке с ним, вы можете сказать, насколько комфортно это чувствует. Кодирование – это весело, если вам нравится этот язык. Fun помогает делать лучшие программы.

Однако это сравнение может быть неточным во всех аспектах, поскольку языки могут быть оптимизированы в некоторых конкретных аспектах. Например, компилятор pascal создает высокооптимизированный код для целых операций (бит-операции и т. Д.), Но может не иметь оптимизации в вычислении с плавающей запятой, в этом случае C / C ++ может создавать более оптимизированные инструкции.

4 РАЗРАБОТКА МЕТОДИКИ С УЧЕТОМ ПАРАДИГМ ПРОГРАММИРОВАНИЯ

4.2 Внедрение предмета в курс элективных дисциплин.

Предмет парадигмы программирования необходимо внести в список элективных дисциплин в первую очередь. После изучения данного предмета студенту будет легче понимать остальные языки программирования. Это увеличит качество образования и поможет преподавателям построить четкий план обучения студента таким образом, что студент сможет поэтапно переходить от одной дисциплины к другой.

Мы предлагаем каталог элективных дисциплин, который поможет преподавателям вести лабораторные работы с учетом требований знаний других элективных дисциплин.

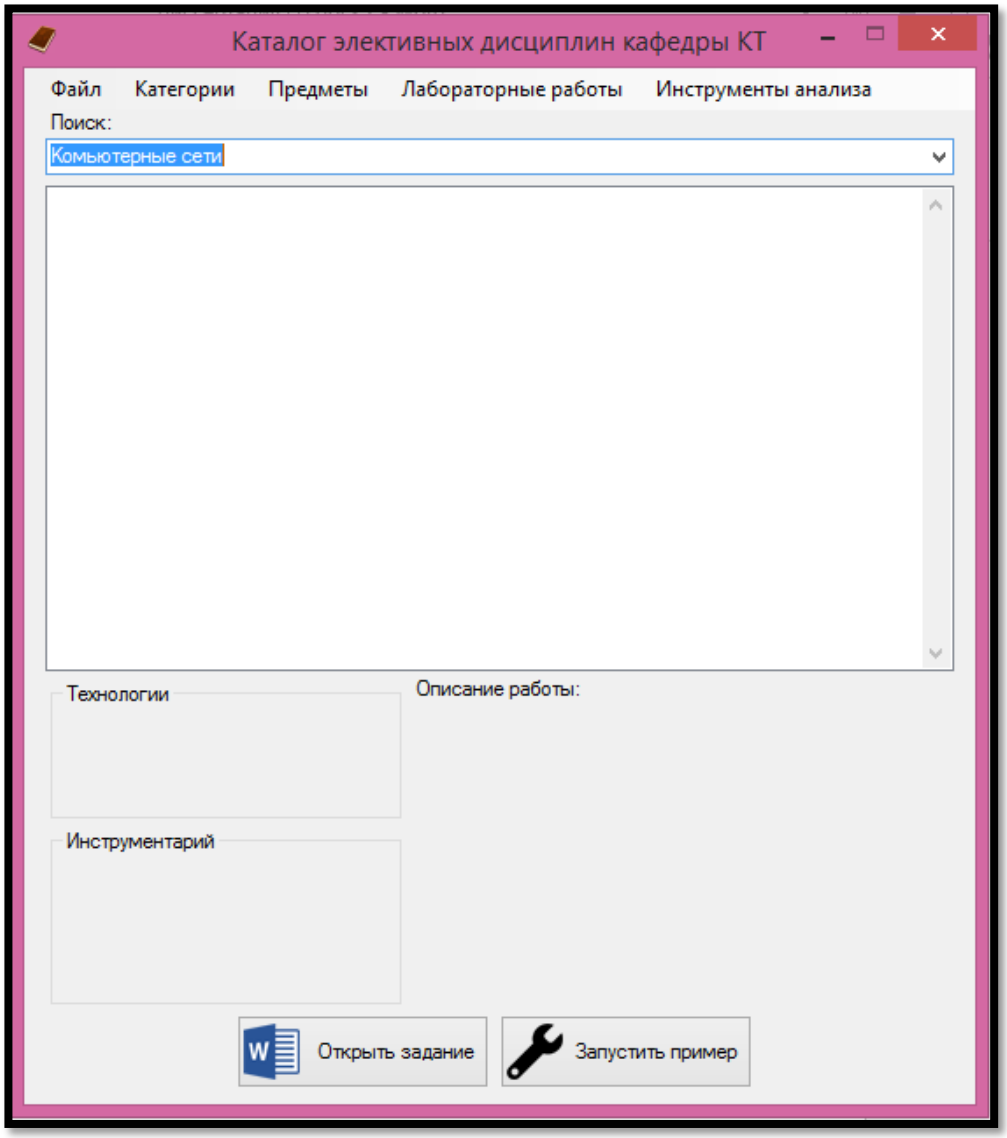

Рисунок 20 – Интерфейс приложения «Каталог элективных дисциплин кафедры компьютерных технологии»

Каталог кафедры компьютерных технологий имеет довольно простой интерфейс, который позволяет с легкостью использовать его как преподавателям, так и студентам.

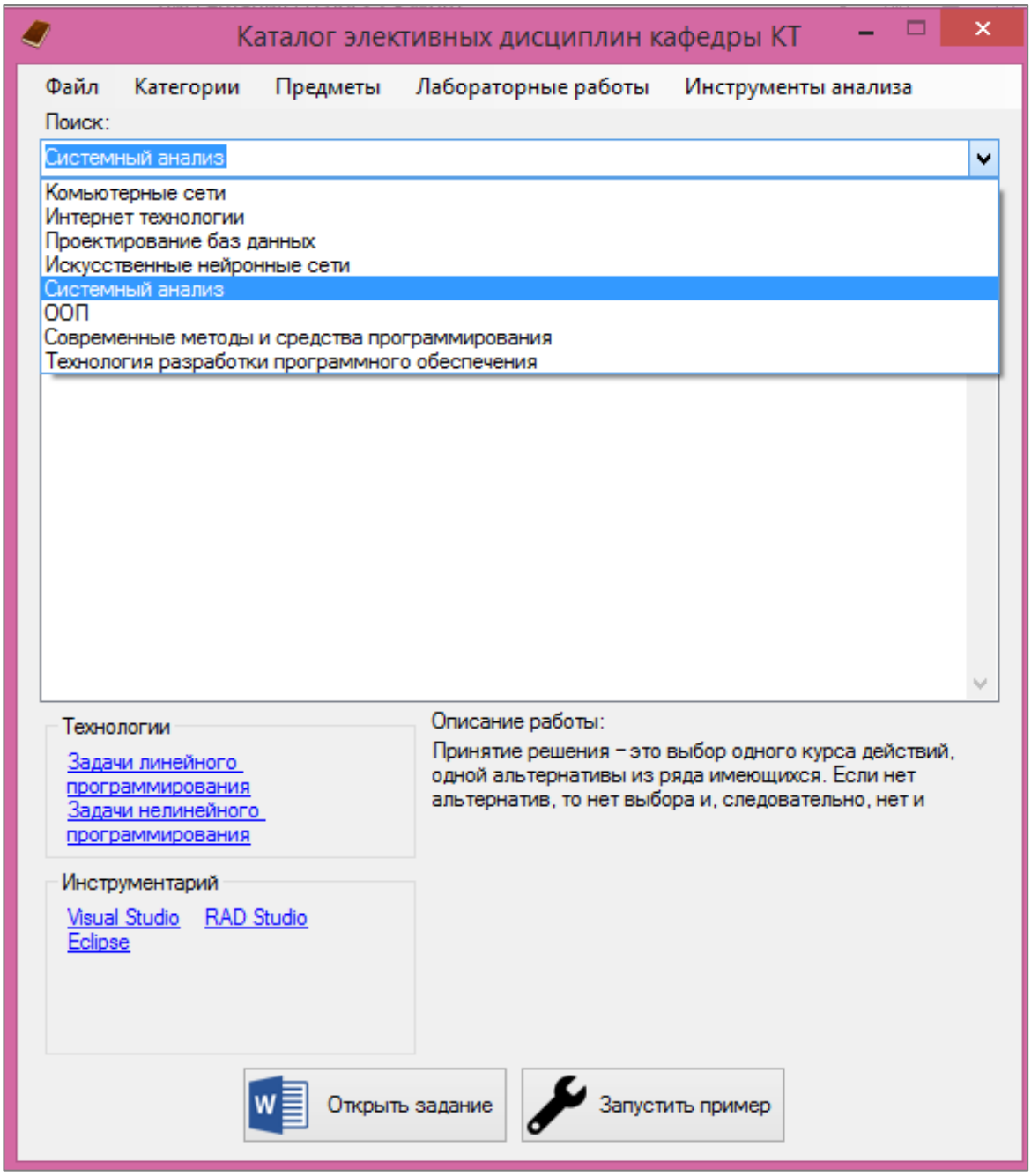

Рисунок 21 – Список дисциплин

В меню «Поиск» расположены элективные курсы, которые были добавлены для примера. По возможности данный список можно будет увеличить или же сократить

На рисунке 22 указан список лабораторных работ по элективной дисциплине, лабораторные работы могут быть добавлены или же удалены конечным пользователем. Данное действие будет добавлено далее.

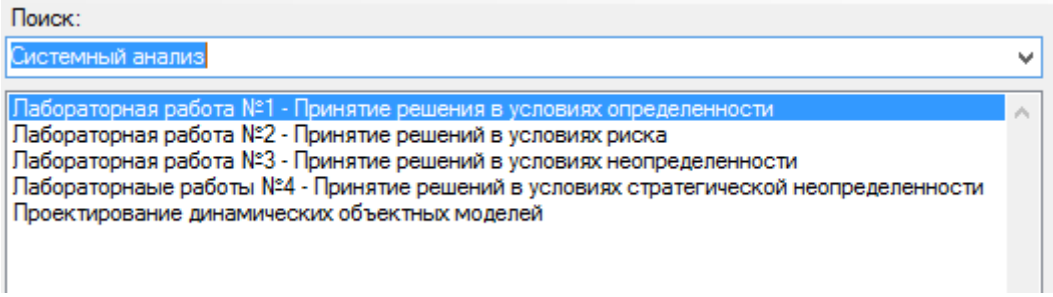

Рисунок 22 – Список лабораторных работ

Также на основной форме показаны технологии, которые нужно будет изучить и которые необходимы для изучения данной элективной дисциплине.

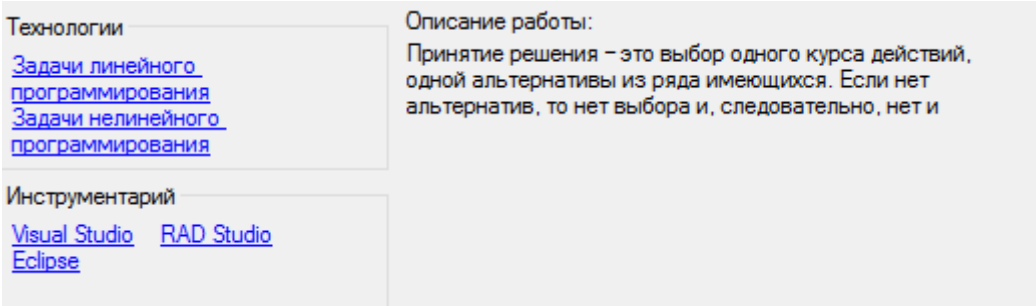

Рисунок 23 – Технологии, инструментарии и описание лабораторной работы

В поле «Инструментарий» были добавлены приложения и среды разработки необходимые для выполнения лабораторных работ. В поле «Описание работы» взято сокращенное описание, которые находятся в методических указаниях.

На рисунке 24 показаны кнопки действий «Открыть задание» и «запустить пример». В каждую лабораторную работу можно добавить задание описанное в текстовом редакторе Microsoft Word а также добавить примеры выполнения лабораторной работы на различных языках программирования.

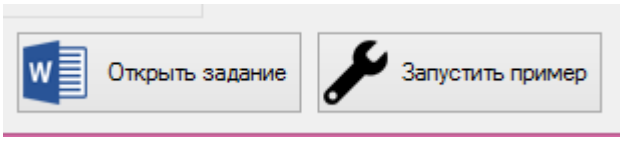

Рисунок 24 – Описание кнопок

Вы можете сами выбрать EXE – файлы, чтобы студенты наглядно могли посмотреть, как выглядит выполнение той или иной лабораторной работы. Данное действие указано на рисунке 25.

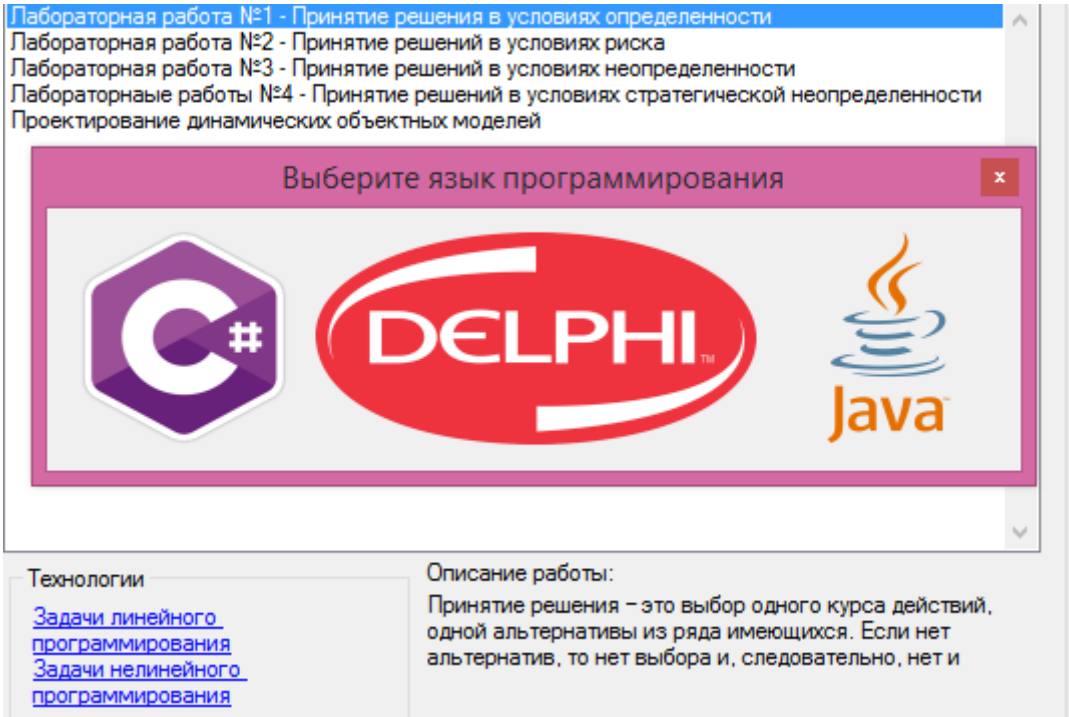

Рисунок 25 – Примеры выполнения лабораторных работ

В качестве примера использованы три основных языка программирования, которые наиболее популярны на данный момент.

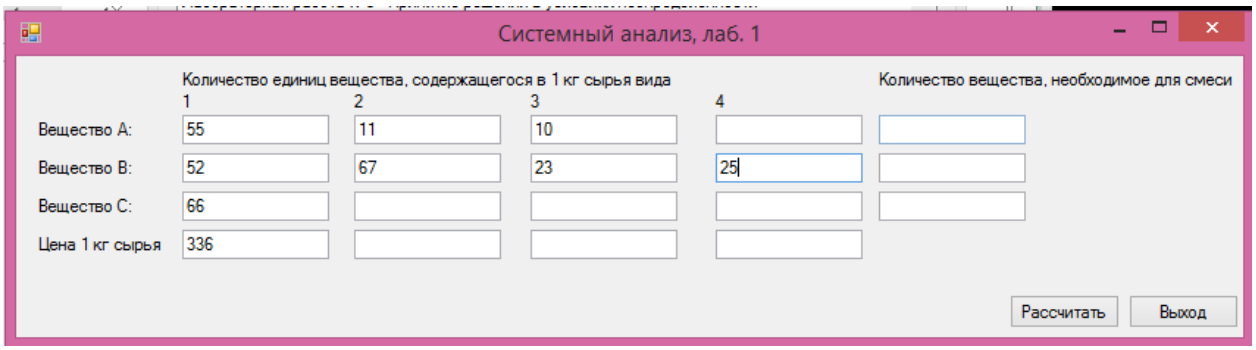

Рисунок 26 – Пример выполнения лабораторной работы на C#

На рисунке 27 показана форма работы с категориями. На донной форме расположены поля «Сферы», например, База данных, системное программирование; Название технологий, то есть список технологий используемых в зависимости от той или иной сферы, допустим при сфере «Методы проектирования» использованы технологии – задачи линейного программирования, проектирование игр и т.д. В поле инструментарий вы можете добавить новые инструменты разработки или сред программирования для той или иной технологий или же использовать существующий список.

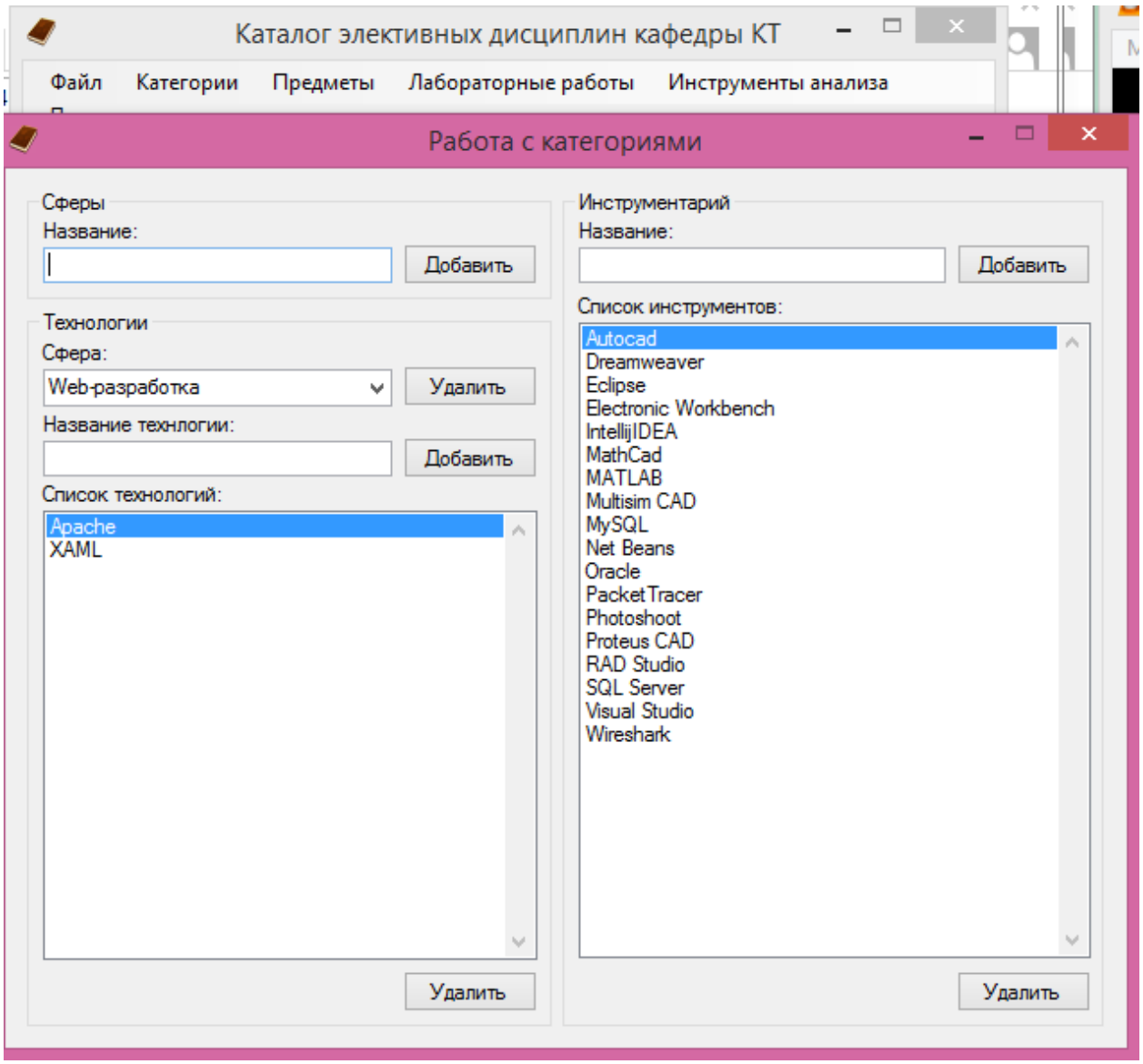

Рисунок 27 – Работа с категориями

На рисунке 28 показана форма «Добавление/Удаление предметов». На данной форме вы можете добавить предметы или же удалить их, так же вы моете задать количество кредитов, которые необходимо освоить при изучении элективной дисциплине.

Академическая кредитная система является стандартом, используемым университетами для оценки и оценки работы и усилий студентов во время их бакалавриата, магистра или доктора философии. Важно понимать, как работают кредиты и как кредиты одной академической системы конвертируются в кредиты других кредитных систем (если это возможно). Иногда студенты должны проходить подготовительные курсы, чтобы соответствовать требованиям к кредитам, необходимым для поступления в университет.

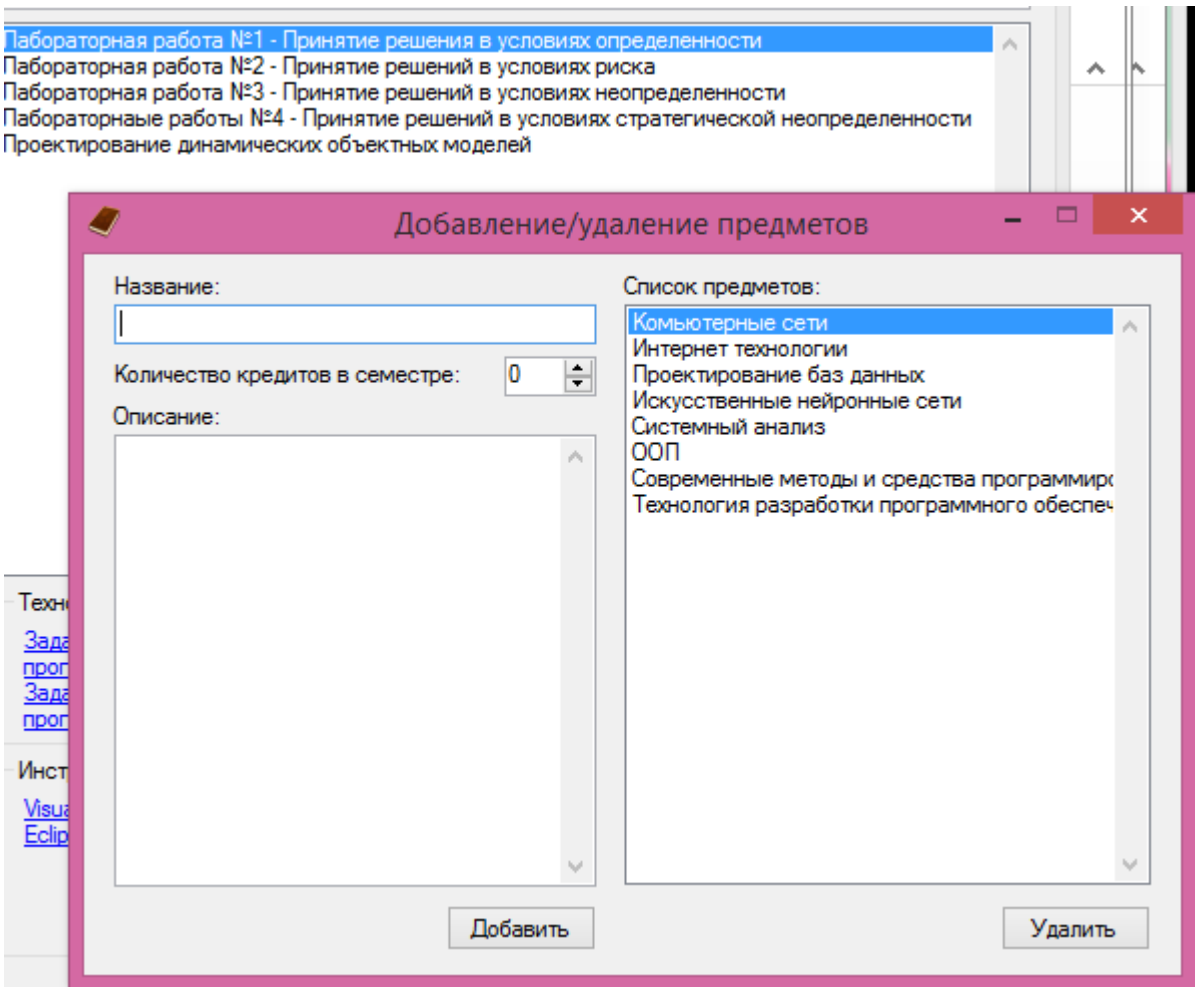

Рисунок 28 – Форма «Добавления/ Удаления предметов»

На рисунке 29 показана форма «Добавление лабораторной работы» , на которой можно непосредственно манипулировать с лабораторными работами, которые предусмотрены при изучении элективных дисциплин. На данной форме преподаватель может добавить лабораторную работу к существующему уже в каталоге предмету. Также подвязать к сфере или же технологии системы. В зависимости от сущности дисциплины/лабораторной работы выбирается необходимы инструментарий для работы. Также можно добавить файл примера и файл задания лабораторной работы. Предусмотрено добавления текста описания лабораторной работы, который преподаватель может взять из методического указания к лабораторной работы.

В целом данный каталог облегчает преподавателям работу и позволяет студентам определить какие навыки необходимы им для выполнения той или иной лабораторной работы. Каталог определяет междисциплинарные связи как по платформам выполнения лабораторных работ так же и по языкам программирования, по сферам использования данных лабораторных работ, по технологиях информационных систем.

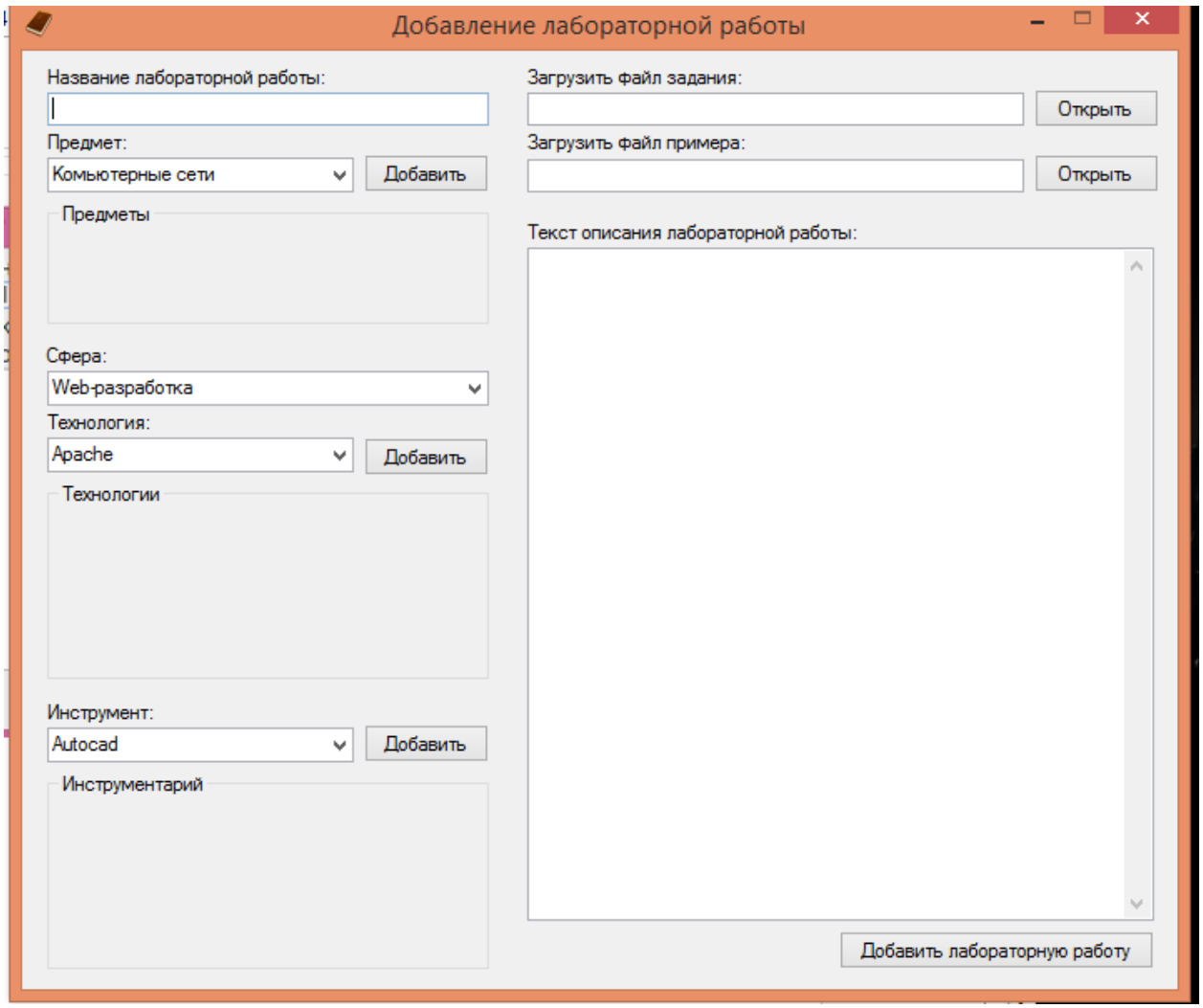

Рисунок 29 – Форма добавления лабораторной рработы

4.3 Проведение численного эксперимента

Проблема, которую мы хотим использовать, - это вычислительное разнообразие, самое простое в реализации и понимании, которое будет иметь очень высокий рост от измерения (например, экспоненциального) так, что интегральная потребность в вычислительных операциях может быть изменена в максимально возможный диапазон.

Для приблизительных оценок вполне подходит проблема вычисления числа рекурсивных чисел Фибоначчи. Эта функция довольно проста, и она будет написана на имеративном языке программирования C.

Последовательность Фибоначчи демонстрирует определенную численную картину, которая возникла как ответ на упражнение в первом тексте средней школы. Этот шаблон оказался интересным и значимым далеко за пределами его воображаемого создателя. Его можно использовать для моделирования или описания удивительного разнообразия явлений, в математике и науке, искусстве и природе. Математические идеи, к которым приводит последовательность Фибоначчи, например золотое соотношение,

спирали и автомодельные кривые, давно ценятся за их обаяние и красоту, но никто не может действительно объяснить, почему они так ясно отражаются в мире искусства и природа.

Рассмотрим элементарный пример геометрического роста – бесполого размножения, как и амебы. Каждый организм распадается на два после интервала времени созревания, характерного для вида. Этот интервал изменяется случайным образом, но в пределах определенного диапазона в соответствии с внешними условиями, такими как температура, доступность питательных веществ и т. Д. Мы можем представить себе упрощенную модель, где в идеальных условиях все амебы расщепляются после того же периода времени роста

Итак, одна амеба становится две, две становятся 4, затем 8, 16, 32 и т. Д.

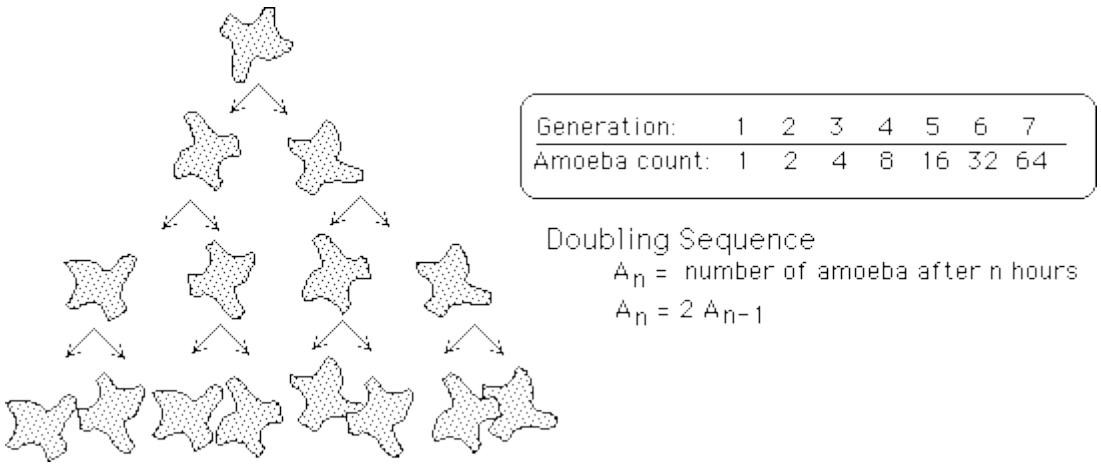

Рисунок 30 – Размножение амебы на примере чисел Фибоначи

Существуют эффективные алгоритмы для вычисления последовательности чисел Фибоначчи (циклические, слева направо). Мы будем сознательно использовать неэффективную рекурсивную реализацию (справа налево) точно в том же виде, что и выражения, написанные выше. С помощью этого алгоритма проблема просто удовлетворяет требованию высокой степени сложности вычислительной сложности, о которой говорилось выше.

Подготовка приложений к выполнению очень различна между рассматриваемыми языками: где-то это только исходный код, который подается на вход интерпретатора, в других случаях он должен компилироваться в промежуточный байт-код или компиляцию в исполняемый машинный код. Все промежуточные этапы подготовки, где они необходимы, объединены в один файл Makefile.

Многие языковые инструменты предполагают и предоставляют некоторые или другие способы оптимизации выполнения (например, уровень оптимизации, указанный компилятору). Где бы я ни знал, как оптимизировать выполнение, будет использоваться максимальный уровень оптимизации.

Выполняя команды для засекания времени, мы создадим команды вида:

```
# time nice -19 <команда_fibo> 30
```

Запуск команды происходит от имени администратора. Сроки выполнения команды системного времени (мы не будем вмешиваться в процесс измерения времени); Параметр (30, число Фибоначчи) определяет размерность задачи, количество вычислений, в зависимости от нее, увеличивается экспоненциально.

Для каждой реализации предыдущий запуск, но на самом деле они выполнялись довольно много (с числом до 10 и более), и в тексте показан средний, самый стабильный вариант (при измерении временных интервалов повторяемость всегда проблема). Мы не используем результаты первого запуска серии, чтобы обеспечить идентичные прогоны кеширования для разных серий серии.

Результаты выполнения могут радикально меняться в зависимости от версий используемых инструментов (компилятор, интерпретатор). Поэтому в результатах выполнения будет показана версия используемого программного обеспечения.

4.4 Результаты эксперимента

<u>Язык С</u>

```
$ gcc --version
gcc (GCC) 4.8.2 20131212 (Red Hat 4.8.2-7)
...
# time nice -19 ./fibo_c 30
1346269
real     0m0.013s
user     0m0.010s
sys      0m0.002s
```

Рисунок 31 – Эсперимент на языке С

Результат выполнения в реальном времени 13 секунд. Довольно очевидно, что результат, программы написанной на языке С будет наиболее производительный, мы будем ориентироваться на результат показанный на этом примере.

<u>С++</u>

```
# time nice -19 ./fibo_cc 30
1346269
 real     0m0.014s
user     0m0.012s
sys      0m0.002s
```

Рисунок 32 – Эксперимент на языке C++

Результат выполнения кода в реальном времени 14 секунд. Здесь время абсолютно равно событию реализации C в пределах статистической ошибки, что и следовало ожидать.

Java

```
# time nice -19 java fibo 30
1346269
real    0m0.176s
user    0m0.136s
sys     0m0.047s
```

Рисунок 33 – Эксперимент на языке Java

Результат выполнения кода в реальном времени 176 секунд.

Python

Было предложение исполнение данного кода в двух разных версиях.

```
$ python --version
Python 2.7.5
# time nice -19 python fibo.py 30
1346269
real    0m1.109s
user    0m1.100s
sys     0m0.005s
```

```
$ python3 --version
Python 3.3.2
# time nice -19 python3 fibo.py 30
1346269
real    0m1.838s
user    0m1.823s
sys     0m0.009s
```

Рисунок 34 – Эксперимент на двух версиях исполнения кода

Результат выполнения первой версии равен 1 минута 109 секунд, второй версии 1 минута 838 секунд. Можно определиить, что первая версия Python быстрее второй в 1.66 раз. Ряд публикаций показывает даже значимость некогда определенных функций, до 2 или 3 раз.

Но по сравнению с этим скомпилированным кодом C, Python 2 меньше в 100 (85) раз.

Ruby

```
$ ruby --version
ruby 2.0.0p353 (2013-11-22 revision 43784) [i386-linux]
# time nice -19 ruby fibo.rb 30
1346269
real    0m0.566s
user    0m0.554s
sys     0m0.009s
```

Рисунок 35 – Результат эксперимента на Ruby

Результат выполнения кода на языке Ruby равен 566 секунд, это в два раза лучше результата Питона и в 42 раза хуже чем результат Си.

Perl

```
$ perl --version
This is perl 5, version 18, subversion 2 (v5.18.2) built for i386-linux-thread-multi
...
# time nice -19 perl fibo.pm 30
1346269
real    0m2.335s
user    0m2.329s
sys     0m0.002s
```

Рисунок 36 – Результат эксперимента на Perl

Компиляция кода на языке Perl в реальном времени заняло 2 минуты и 335 секунды. По времени разница межлду Си и Перлом примерно 179 раз. Это не удивительно, так как Perl обычно работает в обработке текстовых строк а не численных выражений.

JavaScript

```
$ js -v
JavaScript-C 1.8.5 2011-03-31
# time nice -19 js fibo.js 30
1346269
real    0m0.689s
user    0m0.683s
sys     0m0.005s
```

Рисунок 37 – Результат эксперимента на Java Script

Результат выполнения скрипта в реальном времени 689 секунд. Этот результат быстрее питона примерно в два раза но хуже си в 53 раза.

PHP

```
$ php --version
PHP 5.5.9 (cli) (built: Feb 11 2014 08:25:04)
Copyright (c) 1997-2014 The PHP Group
Zend Engine v2.5.0, Copyright (c) 1998-2014 Zend Technologies
# time nice -19 php fibo.php 30
1346269
real    0m1.307s
user    0m1.292s
sys     0m0.013s
```

Рисунок 38 – Результат эксперимента на языке PHP

PhP скрипт выполняет данную последовательность за 1 минуту и 307 секунд. Данный результат в 110 раз меньше чем на языке си.

Прежде всего отметим, что эффективность исполняемого кода зависит не только от языка, на котором написан код, так и от технологии, используемой на языке (компиляция, интерпретация, компиляция в промежуточный байт-код), но также и от конкретного Компилятор и упорядоченные уровни его оптимизации. Разница в зависимости от этих факторов может достигать уровня 3 раза.

Разница в скорости выполнения эквивалентных приложений, реализованных на разных языках, может достигать трех порядков, в несколько сотен раз (между реализацией компиляции и интерпретации).

С другой стороны, ясно, что интерпретируемые языки более гибкие с точки зрения динамического набора текста, особенно в операциях с функциями более высокого порядка (элементы функционального программирования).

Сроки выполнения исполнения – неблагодарное занятие, и результаты будут очень, очень относительными. Чтобы не сравнивать численные значения, мы разлагаем полученные значения для разных языков по логарифмическим кластерам в соответствии с коэффициентом скорости относительно самого быстрого варианта GCC C:

Фактор скорости	Язык
1 ... 2	C, GCC C++, Go, Ocaml
2 ... 5	Cobj C++, PureBasic
5 ... 10	Ocaml
10 ... 20	Java
20 ... 50	Lua, Scheme, Haskell
50 ... 100	Python 2, Ruby, JavaScript, Scala
100 ... 200	Perl, Python 3, PHP
200 ... 500	bash

Рисунок 39 – Сравнение скоростей выполнения между языками программирования

Рассмотрены реализации одной и той же проблемы для сравнения скорости ее выполнения на совершенно разных языках программирования.

Из языков, рассмотренных в сравнении, только очень немногие (C, C ++, Go, PureBasic) используют технику «родной» компиляции в машинный код используемой платформы. Тем не менее, в остальном, в некоторой степени и в технологии, используется виртуальная исполняющая машина (среда исполнения). Это, очевидно, становится тенденцией последнего десятилетия. (Все ранние языки программирования – Algol, FORTRAN, COBOL, Pascal – предполагается скомпилировать его в исполняемый машинный код).

Эта тенденция связана, прежде всего, с лавинной скоростью современных процессоров: скоростью, предлагаемой процессорами, но которую нельзя потреблять и которая не нужна львиной долей потребителей компьютерной техники. Интерпретирующая реализация избыточной производительности канала приложений.

Но есть еще одна объясняемая причина для пристрастия к интерпретационной реализации: она позволяет локализовать ошибки исполняемого кода программы (ошибки программиста) внутри исполняющей системы (виртуальной машины), а не позволять им распространяться в операционную систему. Это может быть принято как реальный способ применения ... «настольных» приложений: образовательных, математических вычислений, развлечений, бытовых ... К сожалению, этот метод неприемлем для «промышленных» приложений: систем управления, электронных платежей, систем связи и телекоммуникаций – fall Приложение, эквивалентное чрезвычайной ситуации, даже если это приложение повторно поднято.

ЗАКЛЮЧЕНИЕ

Компьютерные системы и объекты программирования считаются важным предметом учебной программы университета. Как выпускники производства, обучение базовому программированию имеет важное значение для понимания или, по крайней мере, представления о том, как современные автоматизированные цифровые системы разрешили текущие производственные проблемы, например. Компьютер с числовым программным управлением, выполняющий автоматизированные задачи, такие как автоматическое фрезерование, токарные станки и сверление. Некоторым другим примером производственной системы является применение компьютерной комплексной обработки, где механический программируемый робот будет выполнять автоматизированный процесс в производственной ячейке. Разумеется, эти системы нуждаются в понимании программируемых машин и автоматизации, поэтому изучение основ компьютерных систем и предметов программирования является важным.

Чтобы оснастить будущих выпускников знаниями о программировании и о том, как машины могут решать проблемы механически, языкам программирования (PL) и парадигме программирования (PP) предлагаются инженерные специальности, где последнее важно для понимания того, как различные стили Программирование может помочь решить проблему. Эти стили программирования или парадигмы могут помочь инженерам понять проблему по-разному и разработать эффективные коды.

Введение предмета парадигмы программирования поможет студенты окунуться в мир разработки программ. Благодаря изучению данного предмета студенты смогут легко и просто переходить от одного языка программирования к другому.

Благодаря парадигмам программирования студенты и преподаватели смогут определять междисциплинарные и межплатформенные связи между элективными дисциплинами.

В данной работе были представлены доказательства и доводы необходимости внедрения данного предмета в каталог элективных дисциплин. Это поможет преподавателям определить какие навыки и знания необходимы для выполнения той или иной лабораторной работы.

Электронный каталог поможет определить необходимый инструментарий для выполнения работы, определить технологии и сферы используемые при выполнении работы.

Численный эксперимет показал что наиболее производительный язык программирования это языки программирования написанные на императивной парадигме программирования, так как они могут проводить компиляцию на машинном уровне.

СПИСОК ЛИТЕРАТУРЫ

1 А. Гриффитс : GCC. Полное руководство. Platinum Edition, М.: «ДиаСофт», 2004, ISBN 966-7992-33-0, стр. 624

2 Java™ Platform, Standard Edition 6 API Specification.

3 Учебник Python 3.1.

4 Ruby.

5 Юкихиро Мацумото : Программирование на языке Ruby. Идеология языка, теория и практика применения.

6 Том Кристиансен, Брайан Де Фой, Ларри Уолл, Джон Орвант : Программирование на Perl, 4-е издание, Сп-Б.: «Символ-Плюс», 2013, ISBN: 978-5-93286-214-8, стр. 1048.

7 Introduction to the JavaScript shell.

8 Руководство по PHP.

9 Lua 5.1 Reference Manual.

10 Справочное руководство по языку Lua 5.1.

11 Lua programming language information and resources.

12 Mendel Cooper : Advanced Bash-Scripting Guide — Искусство программирования на языке сценариев командной оболочки, перевод: Андрей Киселев.

13 Nørmark, Kurt. Overview of the four main programming paradigms. Aalborg University, 9 May 2011. Retrieved 22 September 2012.

14 Frans Coenen (1999-10-11). "Characteristics of declarative programming languages". cgi.csc.liv.ac.uk. Retrieved 2014-02-20.

15 Michael A. Covington (2010-08-23). "CSCI/ARTI 4540/6540: First Lecture on Symbolic Programming and LISP" (PDF). University of Georgia. Retrieved 2013-11-20.

16 Peter Van Roy (2009-05-12). "Programming Paradigms for Dummies: What Every Programmer Should Know" (PDF). info.ucl.ac.be. Retrieved 2014-01-27.

17 Frank Rubin (March 1987). "'GOTO Considered Harmful' Considered Harmful" (PDF). Communications of the ACM. 30 (3): 195–196. doi:10.1145/214748.315722. Archived from the original (PDF) on March 20, 2009.

18 Floyd, R. W. (1979). "The paradigms of programming". Communications of the ACM. 22 (8): 455. doi:10.1145/359138.359140.

19 "Mode inheritance, cloning, hooks & OOP (Google Groups Discussion)".

20 "Business glossary: Symbolic programming definition". allbusiness.com. Retrieved 2014-07-30.

21 "Multi-Paradigm Programming Language". developer.mozilla.org. Retrieved 21 October 2013.