МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ КЫРГЫЗСКОЙ РЕСПУБЛИКИ
КЫРГЫЗСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ им.И.АРАБАЕВА
ОСПО ИНСТИТУТА НОВЫХ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

«УТВЕРЖДАЮ»
Директор ИНИТ
КГУ им.И.Арабаева
к.т.н., и.о.доц. У.Керимов
«__» _____ 2023 г.

УЧЕБНО-МЕТОДИЧЕСКИЙ КОМПЛЕКС

по дисциплине Технология разработки программных продуктов

для студентов специальности 230109 – «Программное обеспечение вычислительной техники и автоматизированных систем

гр.ПОВ-1-21,ПОВ-2-21

форма обучения Очное

Курс 3 **Семестр** 5,6

Часов: всего 108 **, лекций:** 64 **, практ. занятий:** 44

СРС 72

Учебно-методический комплекс разработали: преподаватели Масиянова А.К., Акимбек к.Ж., Каныбекова А.К.

Рассмотрена и утверждена на заседании ОСПО ИНИТ КГУ им.И. Арабаева

Протокол № 1 **от «** 07 **»** сентября **20**23 **г.**

Зав. ОСПО ИНИТ: Н.С.Сейтказиева _____

Одобрено учебно-методическим советом ИНИТ КГУ им.И. Арабаева

Протокол № 1 **от «** 08 **»** 09 **20**23 **г.**

Председатель УМС: _____

1. Цели и задачи дисциплины

Целью дисциплины «Технология разработки программных продуктов)» является формированием у студентов знаний по методам, инструментам и процессам конструирования надежного, устойчивого и эффективного ПО для средств вычислительной техники автоматизированных и автоматических систем в рамках современных технологий разработки.

Основными задачами преподавания дисциплины являются:

- изучение методов проектирования программных средств с использованием средств автоматизации проектирования,
- изучение современных инструментальных средств для разработки ПО,
- изучение стандартов по процессам разработки, методам контроля и оценки качества ПО на всех этапах его жизненного цикла,
- изучение эвристических принципов конструирования ПО,
- изучение методов конструирования программ, устойчивых к собственным ошибкам и ошибкам данных,
- изучение принципов верификации и отладки сконструированного ПО
- изучение методов математического моделирования процессов и объектов для создания эффективной среды отладки сконструированного ПО,
- проведение экспериментов с ПО по заданной методике, проведения измерений и наблюдений за работой сконструированного ПО с анализом результатов.
- Теоретические знания закрепляются на практике в процессе выполнения лабораторных работ.

2. Место дисциплины в учебном процессе

Дисциплина относится к циклу Б.3 профессиональных дисциплин вариативной части основной образовательной программы.

Изучение данной дисциплины базируется на следующих дисциплинах:

- Вычислительная математика
- Операционные системы
- Информатика

Основные положения дисциплины должны быть использованы в дальнейшем при изучении следующих дисциплин

- Системы реального времени
- Проектирование и моделирование сетей ЭВМ

Основные положения дисциплины должны быть использованы в дальнейшем при дипломном проектировании и в профессиональной деятельности.

Требования к уровню освоения содержания дисциплины

В результате освоения дисциплины обучающийся должен

Знать:

- математические основы информатики, как науки (ПК-19),
- содержание основных этапов и тенденции развития программирования, математического обеспечения и информационных технологий (ПК-21),
- проблемы и направления развития технологий программирования (ПК-23),
- основные методы и средства автоматизации проектирования, производства испытаний и оценки качества программного обеспечения (ПК-24).

У**меть:**

- понять поставленную задачу, осваивать методы конструирования программных средств для решения практических задач(ОК-5,ПК-2)
- самостоятельно разрабатывать алгоритмы и проводить их анализ(ПК-11), - обосновывать принимаемые проектные решения, осуществлять постановку и выполнять эксперименты по проверке их корректности и эффективности, увидеть на основе анализа и корректно сформулировать точный результат (ПК-5),
- использовать принципы условий безопасности при конструировании программ и программных комплексов (ПК -22),

владеть:

- навыками использования системного моделирования при исследовании, проектировании и отладки программных систем(ПК-32),
- навыками выбора, проектирования, реализации, оценки качества и анализа эффективности программного обеспечения для решения задач в различных предметных областях(ПК – 36).

4. Объем дисциплины и виды учебной работы

Вид учебной работы	**Всего часов**	**№ семестра**		
		5	***6***	
Общая трудоемкость дисциплины	***108***			
Аудиторные занятия (Ауд)	108	54	54	
Лекции (ЛК)	64	32	32	
Практические занятия (ПЗ)		22	22	
Самостоятельная работа (СР)	**72**	**36**	**36**	
Курсовой проект (работа) – (КП, КР)	15		15	

5. Содержание дисциплины

5.1 Содержание разделов дисциплины

№	Наименование разделов	Содержание разделов
1	2	3
1	**Проблемы разработки ПО. Инженерия ПО. Конструирование ПО**	Роль ПО и компьютеров в производстве, социальной жизни и науке. Инженерия ПО. Проблемы разработки Пои пути их разрешения. Конструирование ПО
2	**Технология разработки ПО и качество ПО. Системный подход к разработке ПО. Жизненный цикл ПО. Каскадная модель**	Технология разработки ПО и качество ПО. Характеристики качества ПО. Факторы, влияющие на качество ПО. Системный подход к разработке ПО. Этапы жизненного цикла ПО. Каскадная модель жизненного цикла ПО.
3	**Стандарты и разработка ПО. Основные, вспомогательные и организационные процессы создания ПО. Процессы верификации**	Стандарты по разработке ПО. Виды и значение стандартов. Три группы процессов создания ПО. Жизненный цикл ПО и процессы верификации. Тестирование, верификация, валидация. Различие в понятиях. V образная модель жизненного цикла ПО.
4	**Спиральная модель ЖЦ ПО. «Тяжелые и легкие» технологии разработки ПО. Три вида программных разработок с точки зрения технологии их создания.**	Спиральная модель ЖЦ ПО. «Тяжелые и легкие» технологии разработки ПО. Экстремальное (XP) программирование. Три вида программных разработок с точки зрения технологии их создания.. Виды документов, выпускаемых на ПО по этапам разработки системы.
5	**Итеративный характер проектирования системы и ПО. Проектирование архитектуры ПО. Структура ПО СТС**	Стадии проектирования. Задачи, решаемые на различных стадиях проектирования системы и ПО. Цена ошибок проектирования. Проектирование, основанное на моделировании (MBSE). CASE технологии разработки ПО. Задачи и результаты архитектурного проектирования ПО. Технология Rational Rose, UML. Структура системы и структура ПО. Иерархическая структура ПО СТС.
6.	**Временная диаграмма работы системы и ПО СТС с параллельными физическими процессами.**	Цикличность решения задач управления в системах с ЦВМ. Временная диаграмма работы системы и ПО в варианте использования системы. Представления работы ПО СТС в виде набора «сечений», выполняемых последовательно. Представление работы ПО СТС в виде набора параллельных процессов.
7.	**Процессы. Контекст процесса. Взаимодействие между процессами или потоками.**	Многозадачная работа ПО СТС. Задачи и процессы. Контекст процесса. Схема вариантов совместного использования информации взаимодействующими процессами. Повышение эффективности ПО за счет параллельных вычислений. Закон Амдела.

8.	**Конструирование взаимодействия процессов во времени. «Синхронизация» процессов**	Работа с формами. Оформление формы. Задачи синхронизации «Читатели-писатели», «Обедающие философы». Технология синхронизации ПО.
9	**Конструирование ПО. Минимизация сложности ПО. Приспособленность ПО к изменениям. Проектирование «сверху вниз» и «снизу вверх».**	Конструирование ПО. Минимизация сложности ПО. Повышение приспособленность ПО к изменениям Проектирование снизу-вверх и проектирование сверху-вниз. Работа с графическими вожможностями инструментальных программ
10	**Спиральная модель разработки программного продукта**	Локализация информации в ПО и излишнее еѐ дублирование в ПО.. Определение областей вероятных изменений. Минимизация связей.
11	**Конструирование программ, устойчивых к ошибкам**	Способы обработки ошибок. Изоляция повреждений ПО, вызванных проявившейся ошибкой. Риски при использовании глобальных переменных и способы их уменьшения.
12	**Стратегии безопасности ПО и системы**	Особенности реализации и возникшие трудности: переключение между формами. Устранение ошибок
13	**Технология отладки ПО. Ошибки ПО. Статическая, динамическая, структурная, функциональная отладки**	Ошибки ПО, отладка и тестирование ПО.. Анализ обнаруживаемых в ПО ошибок и важность его проведения. Функциональная отладка .
14	**Автономная отладка и комплексная отладка ПО. Последовательность действий при отладке ПО.**	Автономная отладка (АО) и комплексная отладка программного продукта Последовательность действий при отладке ПО. и при обнаружении ошибки. Драйверы и заглушки для автономной отладки.
15	**Принципы выделения маршрутов отладки. Некоторые проектные модели оценки числа маршрутов при отладке ПО**	Принципы выделения маршрутов для отладки. Приближенный метод оценки числа вариантов для отладки ПО. Случайное дерево, как модель структуры ПО. Контроль отлаженности ПО в процессе отладки.
16	**Моделирование работы систем с целью генерации данных для комплексной отладки ПО.**	Проблема генерации данных на комплексную отладку. Имитация работы системы. Преимущества математического моделирования внешней среды.

5.2 Разделы дисциплины и виды занятий

Разделы дисциплины, изучаемые в 5 семестре

№	Наименование разделов дисциплины	неделя	Виды учебной работы, включая самостоятельную работу студентов и трудоемкость (в часах)				Формы текущего контроля успеваемости
			Всего	Аудиторная работа		Самостоятельная работа	
				ЛК	прак		
1	2	3	4	5	6	7	8
1.	**Проблемы разработки ПО. Инженерия ПО. Конструирование ПО**	1	4	2	2		ПК
2.	**Технология разработки ПО и качество ПО. Системный подход к разработке ПО. Жизненный цикл ПО.**	2	6	4	2		ПК
3.	**Стандарты и разработка ПО. Основные, вспомогательные и организационные процессы создания ПО. Процессы верификации**	3	6	4	2		ПК
4.	**Спиральная модель ЖЦ ПО. «Тяжелые и легкие» технологии разработки ПО. Три вида программных разработок с точки зрения технологии их создания.**	4	6	4	2		ПК
5.	**Итеративный характер проектирования системы и ПО. Проектирование архитектуры системы и ПО. Структура ПО СТС**	5	6	4	2	6	ПК
6.	**Временная диаграмма работы системы и ПО СТС с параллельными физическими процессами.**	6	8	4	4	10	ПК
7.	**Процессы. Контекст процесса. Взаимодействие между процессами или потоками.**	7	8	4	4		ПК
8	**Конструирование обеспечения взаимодействия процессов во времени. «Синхронизация» процессов**	8	6	4	2	10	ПК
9	**Конструирование ПО. Минимизация. Сложности, Ожидание изменений. Рефакторинг ПО. Проектирование «сверху вниз» и «снизу вверх».**	9	6	2	4	10	ПК
Итого за 5 семестр:			**54**	**32**	**22**	**36**	

№	Наименование разделов дисциплины	неделя	Виды учебной работы, включая самостоятельную работу студентов и трудоемкость (в часах)				Формы текущего контроля успеваемости
			Всего	Аудиторная работа		Самостоятельная работа	
				ЛК	**прак**		
1	2	3	4	5	6	7	8
10	**Этапы разработки программного продукта. Анализ данных. Разработка**	10	16	4		10	ПК
11	**Конструирование программ, устойчивых к ошибкам**	11	14	4	4	8	ПК
12	**Конструирование аварийной защиты в ПО на системном уровне. Автоматический контроль работы ПО встроенными средствами. Стратегии безопасности ПО и системы**	12	10	4	4	8	
13	**Технология отладки ПО. Ошибки ПО. Статическая, динамическая, структурная, функциональная отладки**	13	12	4	2	8	ПК
14	**Структурная динамическая отладка. Автономная отладка и комплексная отладка ПО. Последовательность действий при отладке ПО.**	14	4	4	2		ПК
15	**Принципы выделения маршрутов отладки. Некоторые проектные модели оценки числа маршрутов при отладке ПО**	15	12	4	4		ПК
	Контроль отлаженности ПО в процессе отладки. Обеспечение защиты информации			4	4		
16	**Моделирование работы систем с целью генерации данных для комплексной отладки ПО. Наследуемое ПО. Мобильность ПО. Реинжениринг ПО**	16	12	4	2	10	ПК
	Итого за семестр:	17	54	32	22	36	
	Всего за весь курс:		54	32	22	36	

Формы текущего контроля успеваемости- еженедельный опрос.

6. Тематический план изучения дисциплины

6.1 Практические работы

№ ЛР	Наименование лабораторных работ	Колво часов
1	3	4
1	Анализ и синтез модели для изучения свойств структуры ПО с целью определения числа маршрутов на отладку.	4
2	Составление программы для построения случайного дерева-модели графа структуры ПО с заданными случайными характеристиками. Обеспечение устойчивости к ошибкам данных	4
3	Составление программы для построения детерминированного дерева – модели графа структуры ПО с предельными характеристиками лабораторной работы №2 Обеспечение обнаружения ошибок.	6
4	Получение совокупности случайных графов – деревьев, как модели структуры ПО. Определение математического ожидания структурного параметра и числа вариантов на отладку.	6
5	Анализ сходимости значения структурного параметра с ростом числа узлов в модели графе структуры ПО. Построение одного из случайных деревьев.	6
6	Анализ экспериментальных данных по ошибкам ПО с помощью модели Джелинского – Моранды для интенсивности проявления ошибок в ПО	6
7	Составление программы для определения с помощью МНК среднего числа начального количества ошибок в ПО и среднего числа оставшихся ошибок ПО по изменению с течением времени числа обнаруженных ошибок. Обеспечение устойчивости к ошибкам данных	6
8	Составление программы для определения с помощью МНК линейной интерполяции изменения числа обнаруженных ошибок с течением времени по данным лабораторной работы №6. Обеспечение обнаружения ошибок. Сравнение результатов работ №6 и №7 и выводы.	6
Итого практических занятий		44

6.4 Самостоятельное изучение разделов дисциплины

№ раздела	Вопросы, выносимые на самостоятельное изучение
1	2
3	CASEтехнологии разработки ПО встроенных в системы компьютеров. SCADA системы.Система TRACE MODE 6 и технология конструирования ПО встроенного в систему компьютера.
5	Проектирование архитектуры ПО при помощи визуального моделирования в Rational Rose и UML.
9	Программирование в системе Mathcad 14 . Численные методы интегрирования систем дифференциальных уравнений.
14,16	Особенности разработки инструментальных средств для проведения комплексной отладки ПО систем управления. Моделирование времени.
1,2,3	Поиск и изучение дополнительной литературы и современных статей по темам изученного материала, включая электронные издания.

7. Учебно-методическое обеспечение дисциплины:

7.1 Краткое содержание лекций

Тема: Средства для создания приложений. Локальные средства разработки программ

Эти средства на рынке программных продуктов наиболее представительны и включают языки и системы программирования, а также инструментальную среду пользователя.

Язык программирования - формализованный язык для описания алгоритма решения задачи на компьютере.

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

Языки программирования можно условно разделить на следующие классы (если в качестве признака классификации взять синтаксис образования конструкций языка):

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

Другой классификацией языков программирования является их деление на языки, ориентированные на реализацию основ структурного программирования, и объектно-ориентированные языки, поддерживающие понятие объектов и их свойств и методов обработки.

Программа, подготовленная на языке программирования, проходит этап трансляции, когда происходит преобразование исходного кода программы (source code) в объектный код (object code), который далее пригоден к обработке редактором связей. Редактор связей специальная программа, обеспечи-

вающая построение загрузочного модуля (load module), пригодного к выполнению (рис. 3).

Рис. 3 - Схема процесса создания загрузочного модуля программы

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

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

Более мощным средством разработки программ являются *системы программирования.*

Системы программирования (programming system) включают:

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

Средства поддержки проектов - новый класс средств разработки программного обеспечения, предназначенный для:

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

Инструментальная среда пользователя представлена

специальными средствами, встроенными в пакеты прикладных программ, такими, как:

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

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

Интегрированные среды разработки программ

Дальнейшим развитием локальных средств разработки программ, являются интегрированные программные среды разработчиков.

Основное назначение инструментария данного вида - повышение производительности труда программистов, автоматизация создания кодов программ, обеспечивающих интерфейс программного пользователя графического типа, разработка приложений для архитектуры клиент-сервер, запросов и отчетов.

CASE-технологии

CASE-технологии - относительно новое направление, формировавшееся на рубеже 80-х годов.

CASE-технологии делятся на две группы:

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

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

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

- диаграмм потоков данных (DFD - data flow diagrams) совместно со словарями данных и спецификациями процессов;

- диаграмм "сущность-связь" (ERD - entity relationship diagrams), являющихся информационной моделью предметной области;
- диаграмм переходов состояний (STD - state transition diagrams), учитывающих события и реакцию на них системы обработки данных.

Диаграммы DFD устанавливают связь источников информации с потребителями, выделяют логические функции (процессы) образования информации, определяют группы элементов данных и их хранилища (базы данных).

Описание структуры потоков данных, определение их компонентов хранятся в актуальном состоянии в словаре данных, который выступает как база данных проекта. Каждая логическая функция может детализироваться с помощью DFD нижнего уровня согласно методам исходящего проектирования.

Этими CASE-технологиями выполняются автоматизированное проектирование спецификаций программ (задание основных характеристик для разработки программ) и ведение словаря данных.

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

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

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

В рамках CASE-технологий проект сопровождается целиком, а не только его программные коды. Проектные материалы, подготовленные в CASE-технологии, служат заданием программистам, а само программирование скорее сводится к кодированию - переводу на определенный язык структур данных и методов их обработки, если не предусмотрена автоматическая кодогенерация.

Общая характеристика технологии создания программного обеспечения

Основные этапы разработки программного обеспечения представлены на рис 1.

Рис.1 - Этапы разработки программного обеспечения.

Первый этап представляет собой постановку задачи. На этом этапе раскрывается сущность задачи, т.е. формулируется цель ее решения; определяется взаимосвязь с другими задачами; указывается периодичность решения; устанавливаются состав и формы представления входной, промежуточной и результатной информации.

Особое внимание в процессе постановки задачи уделяется детальному описанию входной, выходной (результатной) и межуточной информации.

Особенность реализации этого этапа технологического процесса заключается в том, что конечный пользователь разрабатываемой программы, хорошо знающий ее проблемную сторону, обычно хуже представляет специфику и возможности использования ЭВМ для решения задачи. В свою очередь, предметная область пользователя (особенно ее отдельные нюансы, спо-

собные оказать влияние на решение задачи) зачастую незнакома разработчику программы, хотя он знает возможности и ограничения на применение ЭВМ. Именно эти противоречия являются основной причиной возникновения ошибок при реализации данного этапа технологического процесса разработки программ, которые затем неизбежно отражаются и на последующих этапах. Отсюда вся важность и ответственность этого этапа, требующего осуществления корректной и программного постановки задачи, а также необходимости однозначного ее понимания как разработчиком программы, так и ее пользователем.

Второй этап в технологии разработки программ - математическое описание задачи и выбор метода ее решения. Наличие этого этапа обусловливается рядом причин, одна из которых вытекает из свойства неоднозначности естественного языка, на котором описывается постановка задачи. В связи с этим на нем выполняется формализованное описание задачи, т.е. устанавливаются и формулируются средствами языка математики логико-математические зависимости между исходными и результатными данными. Математическое описание задачи обеспечивает ее однозначное понимание пользователем и разработчиком программы.

Сложность и ответственность этапа математического описания задачи и выбора (разработки) соответствующего метода ее решения часто требуют привлечения квалифицированных специалистов области прикладной математики, обладающих знанием таких дисциплин, как исследование операций, математическая статистика, вычислительная математика и т.п.

Третий этап технологического процесса подготовки решения задач ЭВМ представляет собой алгоритмизацию ее решения, т.е. разработку оригинального или адаптацию (уточнение и корректировку) уже известного алгоритма.

Алгоритмизация - это сложный творческий процесс. В основу процесса алгоритмизации положено фундаментальное понятие математики и программирования - алгоритм. Алгоритм - это конечный набор правил, однозначно раскрывающих содержание и последовательность выполнения операций для систематического решения определенного класса задач за конечное число шагов.

Составление (адаптация) программ (кодирование) является завершающим этапом технологического процесса разработки программных средств. Он предшествует началу непосредственно машинной реализации алгоритма решения задачи. Процесс кодирования заключается в переводе описания алгоритма на один из доступных для ЭВМ языков программирования. В процессе составления программы для ЭВМ конкретизируются тип и структура используемых данных, а последовательность действий, реализующих алгоритм, отражается посредством конкретного языка программирования.

Этап тестирования и отладки. Оба эти процесса функционально связаны между собой, хотя их цели несколько отличаются друг от друга.

Тестирование представляет собой совокупность действий, назначенных для демонстрации правильности работы программы в заданных диапазонах изменения внешних условий и режимов эксплуатации программы. Цель тестирования заключается в демонстрации отсутствия (или выявлении) ошибок в разработанных программах на заранее подготовленном наборе контрольных примеров.

Процессу тестирования сопутствует понятие "отладка", которое подразумевает совокупность действий, направленных на устранение ошибок в программах, начиная с момента обнаружения фактов ошибочной работы программы и завершая устранением причин их возникновения.

Программного продукта своему характеру (причине возникновения) ошибки в программах делятся на синтаксические и логические.

Синтаксические ошибки в программе представляют собой некорректную запись отдельных языковых конструкций с точки зрения правил их представления для выбранного языка программирования. (ошибки выявляются автоматически)

Далее проверяется логика работы программы на исходных данных. При этом возможны следующие основные формы проявления логических ошибок:

— в какой-то момент программа не может продолжать работу (возникает программное прерывание, обычно сопровождающееся указанием места в программе, где оно произошло);
— программа работает, но не выдает всех запланированных результатов и не выходит на останов (происходит ее "зацикливание");
— программа выдает результаты и завершает свою работу, но они полностью или частично не совпадают с контрольными.

После выявления логических ошибок и устранения причин их возникновения в программу вносятся соответствующие исправления и отладка продолжается.

Программа считается отлаженной, если она безошибочно выполняется на достаточно представительном наборе тестовых данных, обеспечивающих проверку всех ее участков (ветвей).

Процесс тестирования и отладки программ имеет итерационный характер и считается одним из наиболее трудоемких этапов процесса разработки программ. Программного продукта оценкам специалистов, он может составлять от 30 до 50% в общей структуре затрат времени на разработку проектов и зависит от объема и логической сложности разрабатываемы программных комплексов.

Для сокращения затрат на проведение тестирования и отладки в настоящее время широко применяются специальные программные средства тестирования (например, генераторы тестовых данных) и приемы отладки (например, метод трассировки программ, позволяющий выявлять, все ли ветви программы были задействованы при решении задачи с заданными наборами исходных данных).

После завершения процесса тестирования и отладки программные средства вместе с сопроводительной документацией передаются пользователю для эксплуатации.

Основное назначение сопроводительной документации - обеспечить пользователя необходимыми инструктивными материалами программного продукта работе с программными средствами. Состав сопроводительной документации обычно оговаривается заказчиком (пользователем) и разработчиком на этапе подготовки технического задания на программное средство. Как правило, это документы, регламентирующие работу пользователя в процессе эксплуатации программы, а также содержащие информацию о программе, необходимую в случае возникновения потребности внесения изменений и дополнений в нее.

При передаче пользователю разработанных прикладных программных средств создается специальная комиссия, включающая в свой состав представителей разработчиков и заказчиков (пользователей). Комиссия в соответствии с заранее составленным и утвержденным обеими сторонами планом проводит работы программного продукта приемке-передаче программных средств и сопроводительной документации. Программного продукта завершении работы комиссии оформляется акт приемки-передачи.

В процессе внедрения и эксплуатации прикладных программных средств могут выявляться различного рода ошибки, не обнаруженные разработчиком при тестировании и отладке программных средств. Поэтому при реализации достаточно сложных и ответственных программных комплексов программного продукта согласованию пользователя (заказчика) с разработчиком этап эксплуатации программных средств может быть разбит на два под этапа: экспериментальная (опытная) и промышленная эксплуатация.

Смысл экспериментальной эксплуатации заключается во внедрении разработанных программных средств на объекте заказчика с целью проверки их работоспособности и удобства работы пользователей при решении реальных задач в течение достаточно длительного периода времени (обычно не менее года) Только после завершения периода экспериментальной эксплуатации и устранения выявленных при этом ошибок и учета замечаний программное средство передается в промышленную эксплуатацию.

Для повышения качества работ, оперативности исправления ошибок, выявляемых в процессе эксплуатации программных средств, также выполнения различного рода модификаций, в которых может возникнуть необходимость в ходе эксплуатации, разработчик может программного продукта договоренности с пользователем осуществлять их сопровождение.

Описанная схема технологического процесса разработки прикладных программных средств отражает их "жизненный цикл", т.е. временной интервал с момента зарождения программы до момента полного отказа от ее эксплуатации.

Современные методы и средства разработки программного обеспечения

Метод нисходящего проектирования (метод пошаговой детализации, метод иерархического проектирования, top-down-подход) .

Суть метода заключается в определении спецификаций компонентов системы путем последовательного выделения в ее составе отдельных составляющих и их постепенной детализации до уровня, обеспечивающего однозначное понимание того, что и как необходимо разрабатывать и реализовывать.

Этот метод является незаменимым при разработке сложных программного продукта характеру и больших программного продукта объему программ, когда к их разработке необходимо привлекать большое число программистов, работающих параллельно. Он программного позволяет концентрировать внимание разработчиков на наиболее ответственных частях программы, а также облегчает возможность программного постоянного контроля за ее работоспособностью программного продукта мере разработки, отладки и объединения отдельных составляющих программ за счет организации непрерывности этого процесса в течение всей разработки.

Для ускорения разработки программного комплекса часто вместо некоторых программ нижнего уровня, находящихся в процессе разработки, могут применяться специальные "программы-заглушки" Программы-заглушки требуются только на ранних стадиях разработки для того, чтобы не сдерживать общий ход создания программного комплекса. Суть программы-заглушки заключается в том, что при обращении к ней в соответствии с заданным набором исходных тестовых данных она не формирует, а выбирает результат "решения" из заранее подготовленного набора. Благодаря этому обеспечивается возможность имитировать работу на ЭВМ реально создаваемой программы, а следовательно, осуществлять проверку работоспособности программ верхнего уровня еще до того, как будут разработаны и отлажены все составляющие программы нижнего уровня.

Модульное проектирование

Реализация метода нисходящего проектирования тесно связана с другим понятием программирования - модульным проектированием, так как на практике при декомпозиции сложной программы возникает вопрос о разумном пределе ее дробления на составные части. Вместе с тем понятие модульности нельзя сводить только к представлению сложных программных комплексов в виде набора отдельных функциональных блоков.

Модуль - это последовательность логически взаимосвязанных фрагментов задачи, оформленных как отдельная часть программы. При этом программные модули должны обладать следующими свойствами:

— на модуль можно ссылаться (т.е. обращаться к нему) программного продукта имени, в том числе и из других модулей;

— программного продукта завершении работы модуль должен возвращать управление тому модулю, который его вызывал;

— модуль должен иметь один вход и выход;

— модуль должен иметь небольшой размер, обеспечивающий его обозримость.

При разработке сложных программ в них выделяют головной управляющий модуль, подчиненные ему модули, обеспечивающие реализацию отдельных функций управления, функциональную обработку (т.е. непосредственную реализацию основного назначения программного комплекса), а также вспомогательные модули, обеспечивающие сервисное обслуживание пакета (например, сбор и анализ статистики работы программы, обработка различного рода ошибочных ситуаций, обучение и выдача подсказок и т.п.).

Модульный принцип разработки программ обладает следующими преимуществами:

- большую программу могут разрабатывать одновременно несколько исполнителей и это позволяет сократить сроки ее разработки;
- программный продукт является возможность создавать и многократно использовать в дальнейшем библиотеки наиболее используемых программ;
- упрощается процедура загрузки больших программ в оперативную память, когда требуется ее сегментация
- возникает много естественных контрольных точек для наблюдения за осуществлением хода разработки программ, а в последующем для контроля за ходом исполнения программ;
- обеспечивается более эффективное тестирование программ, проще осуществляются проектирование и последующая отладка.

Преимущества модульного принципа программного построения программ особенно наглядно проявляются на этапе сопровождения и модификации программных продуктов, программного позволяя значительно сократить затраты сил и средств на реализацию этого этапа.

МОДЕЛИ РАЗРАБОТКИ ПРОГРАММНОГО ПРОДУКТА

Рассмотрим самые основные — модели разработки программного продукта (как часть жизненного цикла программного продукта). При этом сразу подчеркнём, что разработка программного продукта является лишь частью жизненного цикла программного продукта, и здесь мы говорим именно о разработке.

Моделей разработки программного продукта много, но, в общем случае, классическими можно считать каскадную, **v**-образную, итерационную, инкрементальную, спиральную и гибкую.

Знать и понимать модели разработки программного продукта необходимо затем, чтобы уже с первых дней работы понимать, что происходит вокруг, что, зачем и почему Вы делаете. Чем подробнее вы будете представлять картину происходящего на проекте, тем яснее Вам будет виден ваш собственный вклад в общее дело и смысл того, чем вы занимаетесь.

Ещё одна важная вещь, которую следует понимать, состоит в том, что никакая модель не является установкой или универсальным решением. Нет идеальной модели. Есть та, которая хуже или лучше подходит для конкретного проекта, конкретной команды, конкретных условий.

Каскадная (водопадная) модель

Сейчас представляет, скорее, исторический интерес, т.к. в современных проектах практически не применима. Она предполагает однократное выполнение каждого из фаз проекта, которые, в свою очередь, строго следуют друг за другом. Простыми словами можно сказать, что, в рамках этой модели, в любой момент времени команде «видна» лишь предыдущая и следующая фаза. В реальной же разработке программного продукта приходится «видеть весь проект целиком» и возвращаться к предыдущим фазам, чтобы исправить недоработки или что-то уточнить.

Особенности каскадной модели:

— высокий уровень формализации процессов;

— большое количество документации;

— жесткая последовательность этапов жизненного цикла без возможности возврата на предыдущий этап.

Минусы:

- Waterfall-проект должен постоянно иметь актуальную документацию. Обязательная актуализация проектной документации. Избыточная документация.
- Очень не гибкая методология.
- Может создать ошибочное впечатление о работе над проектом (например, фраза «45% выполнено» не несёт за собой никакой программного полезной информации, а является всего лишь инструментов для менеджера проекта).
- У заказчика нет возможности ознакомиться с системой заранее и даже с «Пилотом» системы.
- У пользователя нет возможности привыкать к продукту постепенно.

- Все требования должны быть известны в начале жизненного цикла проекта.
- Возникает необходимость в жёстком управлении и регулярном контроле, иначе проект быстро выбьется из графиков.
- Отсутствует возможность учесть переделку, весь проект делается за один раз.

Плюсы:

- Высокая прозрачность разработки и фаз проекта.
- Чёткая последовательность.
- Стабильность требований.
- Строгий контроль менеджмента проекта.
- Облегчает работу программного продукта составлению плана проекта и сбора команды проекта.
- Хорошо определяет процедуру программного продукта контролю качества.

«Водоворот» или каскадная модель с промежуточным контролем

В этой модели предусмотрен промежуточный контроль за счет обратных связей. Но это достоинство порождает и недостатки. Затраты на реализацию проекта при таком подходе возрастают практически в 10 раз. Эта модель, как Вы уже поняли, является незначительной модификацией предыдущей и относится к первой группе.

При реальной работе, в соответствии с моделью, допускающей движение только в одну сторону, обычно возникают проблемы при обнаружении недоработок и ошибок, сделанных на ранних этапах. Но еще более тяжело иметь дело с изменениями окружения, в котором разрабатывается (это могут быть изменения требований, смена подрядчиков, изменение политики разрабатывающей или эксплуатирующей организации, изменения отраслевых стандартов, появление конкурирующих продуктов и пр.).

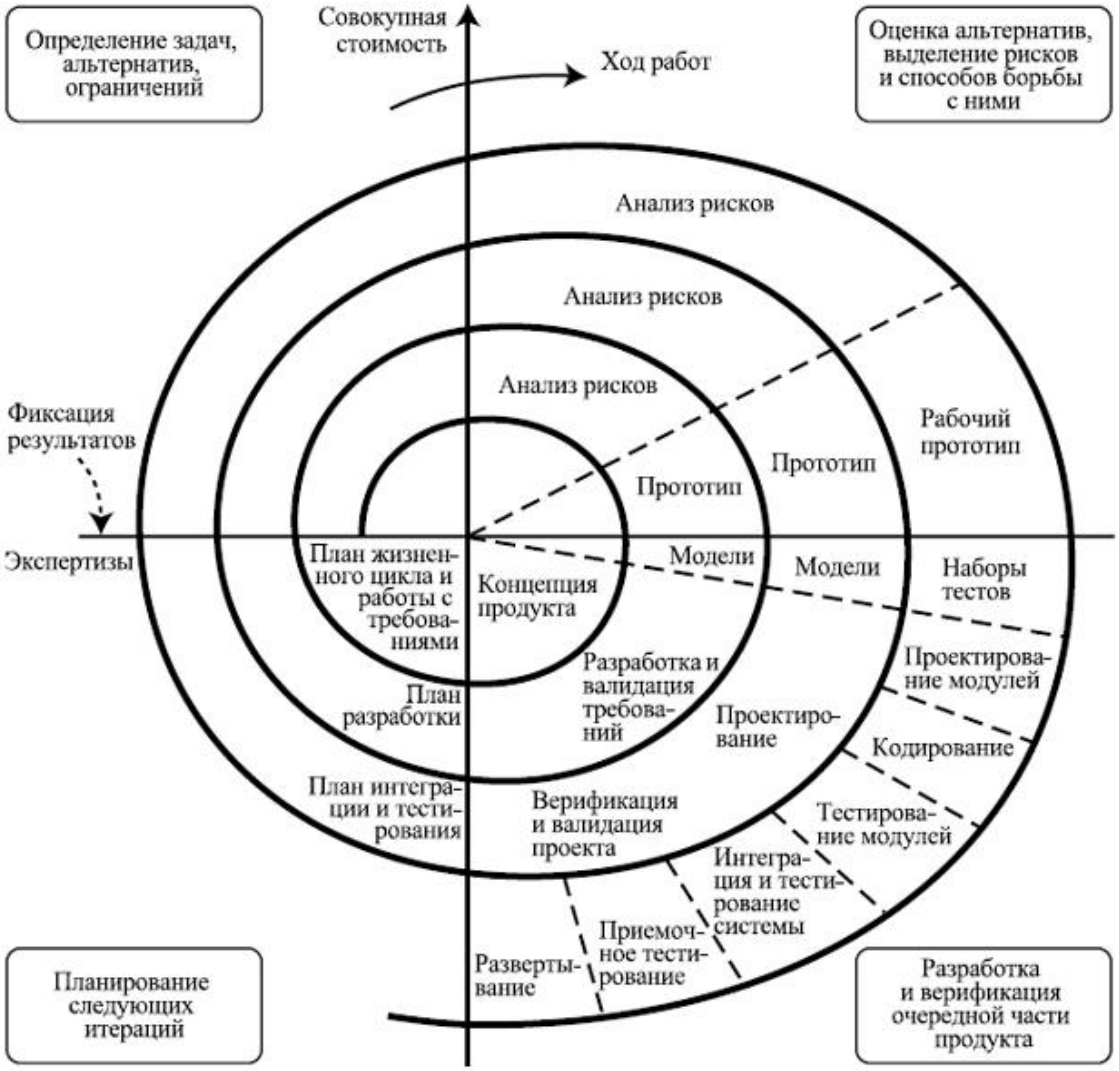
Итеративная модель

Итеративные или инкрементальные модели (известно несколько таких моделей) предполагают разбиение создаваемой системы на набор кусков, которые разрабатываются с помощью нескольких последовательных проходов всех работ или их части.

Каскадная модель с возможностью возвращения на предшествующий шаг, при необходимости пересмотреть его результаты, становится итеративной. Итеративный процесс полагает, что разные виды деятельности не привязаны намертво к определенным этапам разработки, а выполняются программного продукта мере необходимости, иногда повторяются, до тех пор, пока не будет получен нужный результат. Вместе с гибкостью и возможностью быстро реагировать на изменения, итеративные модели привносят дополнительные сложности в управление проектом и отслеживание его хода. При использовании итеративного подхода значительно сложнее становится адекватно оценить текущее состояние проекта и спланировать долгосрочное развитие событий, а также предсказать сроки и ресурсы, необходимые для обеспечения определенного качества результата.

Спиральная модель жизненного цикла программного продукта

Данная модель прекрасно сочетает в себе прототипирование и проектирование программного продукта стадиям. И из восходящей и нисходящей концепций в эту модель было взято все лучшее.



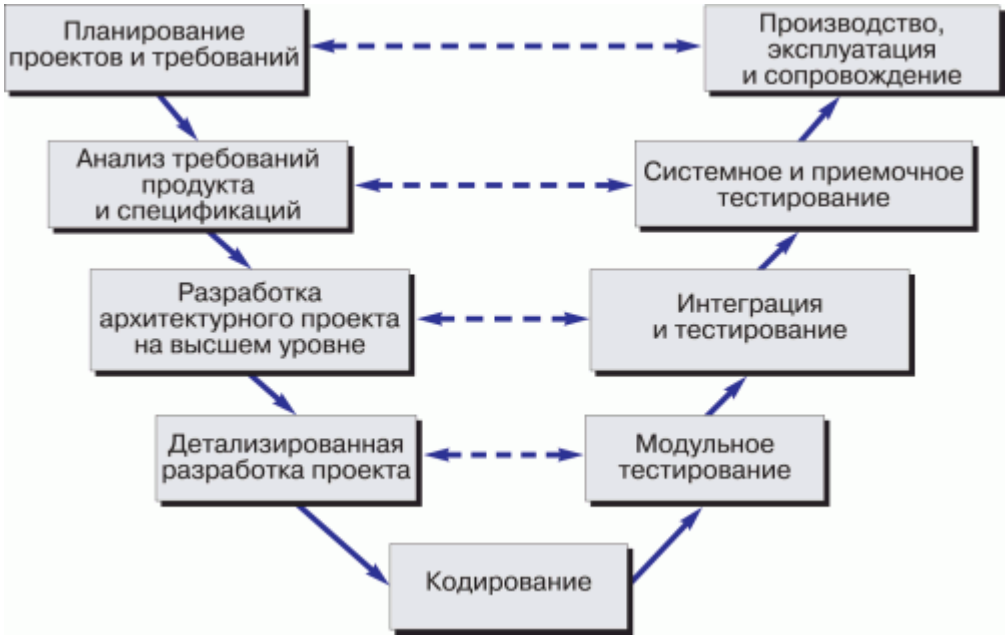
Преимущества модели:

- Результат достигается в кратчайшие сроки.
- Конкурентно способность достаточно высокая.
- При изменении требований не придется начинать все с «нуля».

Но у этой модели есть один существенный **недостаток: невозможность регламентирования стадий выполнения.**

V модель — разработка через тестирование

Данная модель имеет более приближенный к современным методам алгоритм, однако все еще имеет ряд недостатков. Является одной из основных практик экстремального программирования и полагает регулярное тестирование продукта во время разработки.



V-модель обеспечивает поддержку в планировании и реализации проекта. В ходе проекта ставятся следующие задачи:

- Минимизация рисков: **V**-образная модель делает проект более прозрачным и повышает качество контроля проекта путём стандартизации промежуточных целей и описания соответствующих им результатов и ответственных лиц. Это позволяет выявлять отклонения и риски в проекте на ранних стадиях и улучшает качество управления проектов, уменьшая риски.
- Повышение и гарантии качества: **V**-Model —стандартизованная модель разработки, что позволяет добиться от проекта результатов желаемого качества. Промежуточные результаты могут быть проверены на ранних стадиях. Универсальное документирование облегчает читаемость, понятность и проверяемость.
- Уменьшение общей стоимости проекта: ресурсы на разработку, производство, управление и поддержку могут быть заранее просчитаны и проконтролированы. Получаемые результаты также универсальны и легко прогнозируются. Это уменьшает затраты на последующие стадии и проекты.
- Повышение качества коммуникации между участниками проекта: универсальное описание всех элементов и условий облегчает взаимопонимание всех участников проекта. Таким образом, уменьшаются неточности в понимании между пользователем, покупателем, поставщиком и разработчиком.

Модель на основе разработки прототипа

Данная модель основывается на разработке прототипов и прототипирования продукта и относится ко второй группе.

Прототипирование используется на ранних стадиях жизненного цикла программного обеспечения:

- Прояснить неясные требования (прототип **UI**).
- Выбрать одно из ряда концептуальных решений (реализация сценариев).

- Проанализировать осуществимость проекта.

Классификация прототипов:

- Горизонтальные прототипы — моделирует исключительно UI, не затрагивая логику обработки и базу данных.
- Вертикальные прототипы — проверка архитектурных решений.
- Одноразовые прототипы — для быстрой разработки.
- Эволюционные прототипы — первое приближение эволюционной системы.

Agile (идеология) - манифест разработки программного обеспечения

Мы постоянно открываем для себя более совершенные методы разработки программного обеспечения, занимаясь разработкой непосредственно и помогая в этом другим. Благодаря проделанной работе мы смогли осознать, что:

- Люди и взаимодействие важнее процессов и инструментов.
- Работающий продукт важнее исчерпывающей документации.
- Сотрудничество с заказчиком важнее согласования условий контракта.
- Готовность к изменениям важнее следования первоначальному плану.

То есть, не отрицая важности того, что справа, мы всё-таки больше ценим то, что слева.

Основополагающие принципы **Agile**-манифеста

Основные идеи:

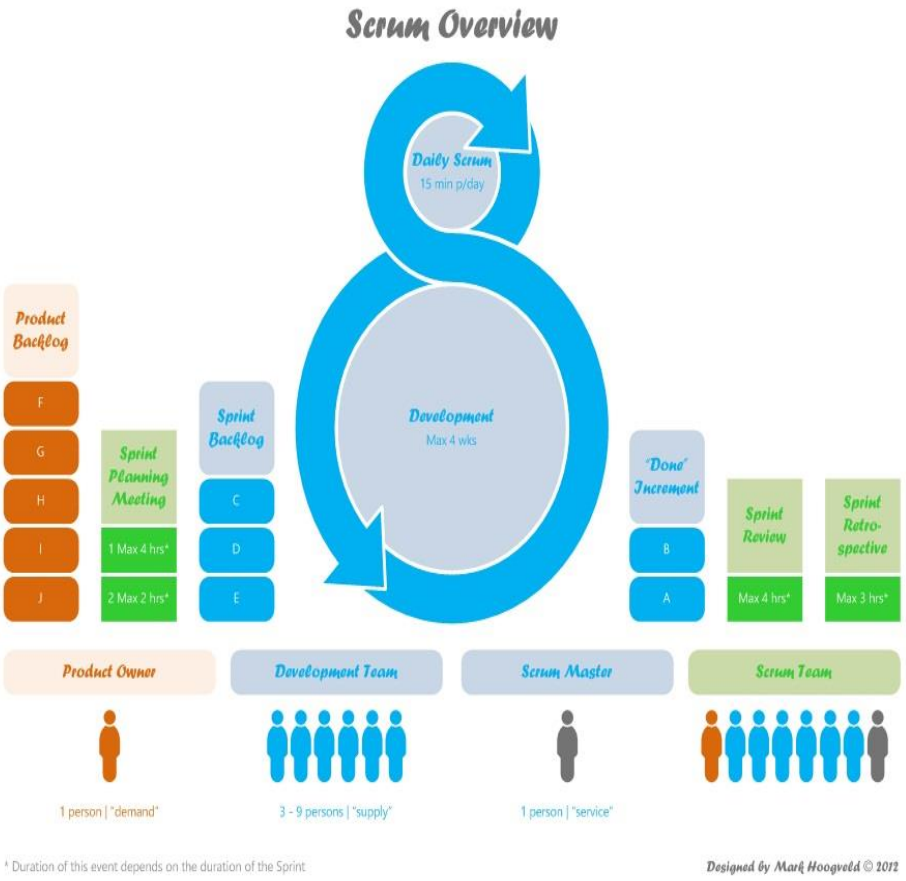
- люди и взаимодействие важнее процессов и инструментов;
- работающий продукт важнее исчерпывающей документации;
- сотрудничество с заказчиком важнее согласования условий контракта;
- готовность к изменениям важнее следования первоначальному плану.

Мы следуем таким принципам:

- Наивысшим приоритетом для нас является удовлетворение потребностей заказчика благодаря регулярной и ранней поставке ценного программного обеспечения.
- Изменение требований приветствуется даже на поздних стадиях разработки. **Agile**-процессы позволяют использовать изменения для обеспечения заказчику конкурентного преимущества.
- Работающий продукт следует выпускать как можно чаще, с периодичностью от пары недель до пары месяцев.
- На протяжении всего проекта разработчики и представители бизнеса должны ежедневно работать вместе.
- Над проектом должны работать мотивированные профессионалы. Чтобы работа была сделана, создайте условия, обеспечьте поддержку и полностью доверьтесь им.
- Непосредственное общение является наиболее практичным и эффективным способом обмена информацией как с самой командой, так и внутри команды.
- Работающий продукт — основной показатель прогресса.

- Инвесторы, разработчики и пользователи должны иметь возможность поддерживать постоянный ритм бесконечно. **Agile** помогает наладить такой устойчивый процесс разработки.
- Постоянное внимание к техническому совершенству и качеству проектирования повышает гибкость проекта.
- Простота — искусство минимизации лишней работы — крайне необходима.
- Самые лучшие требования, архитектурные и технические решения рождаются у самоорганизующихся команд.
- Команда должна систематически анализировать возможные способы улучшения эффективности и соответственно корректировать стиль своей работы.

SCRUM



Scrum (Скрам) — это не аббревиатура, этот термин взят из регби, который обозначает схватку вокруг мяча.

Сам термин **Scrum** можно определить так — это методология управления проектами, которая построена на принципах тайм-менеджмента. Основной ее особенностью является вовлеченность в процесс всех участников, причем у каждого участника есть своя определенная роль. Суть в том, что не только команда работает над решением задачи, но все те, кому интересно решение задачи. Не просто поставили задачу и расслабились, а постоянно

«работают» с командой и эта работа не означает только постоянный контроль.

Основные термины, которые используются в методологии:

- **Владелец продукта (Product owner)** — человек, который имеет непосредственный интерес в качественном конечном продукте, он программного понимает, как это продукт должен выглядеть/работать. Этот человек не работает в команде, он работает на стороне заказчика/клиента (это может быть как другая компания, так и другой отдел), но этот человек работает с командой. И это тот человек, который расставляет приоритеты для задач.
- **Scrum-мастер** — это человек, которого можно назвать руководителем проекта, хотя это не совсем так. Главное, что это человек «зараженный Scrum-бациллой» настолько, что несет ее как своей команде, так и заказчику и, соответственно, следит за тем, чтобы все принципы Scrum соблюдались.
- **Scrum-команда** — это команда, которая принимает все принципы Scrum и готова с ними работать.
- **Спринт** — отрезок времени, который берется для выполнения определенного (ограниченного) списка задач. Рекомендуется брать 2-4 недели (длительность определяется командой один раз).
- **Бэклог (backlog)** — это список всех работ. Можно сказать, это ежедневник общего пользования. Различают 2 вида бэклогов: Product-бэклог и спринт-бэклог.
    1. **Product-бэклог** — это полный список всех работ, при реализации которых мы получим конечный продукт.
    2. **Спринт-бэклог** — это список работ, который определила команда и согласовала с Владельцем продукта на ближайший отчетный период (спринт). Задания в спринт-бэклог берутся из product-бэклога.
- **Планирование спринта** — это совещание, на котором присутствуют все (команда, **Scrum**-мастер, Владелец продукта). В течение этого совещания Владелец продукта определяет приоритеты заданий, которые он хотел бы увидеть выполненными программного продукта истечении спринта. Команда оценивает программного продукта времени, сколько из желаемого они могут выполнить. В итоге получается список заданий, который не может меняться в течение спринта и к концу спринта должен быть полностью выполнен.

7.2 Содержание практических занятий

Задача 1.1

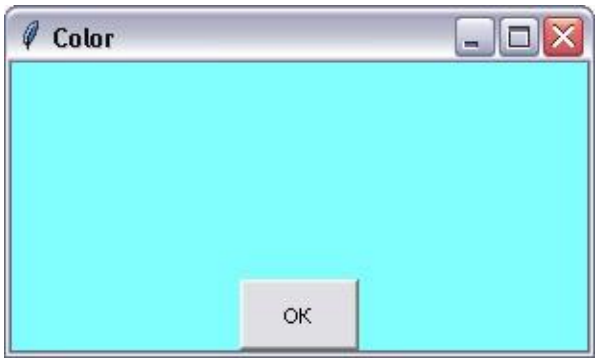
Рассмотрим простейшую задачу: изменить цвет основного окна формы на выбранный пользователем при нажатии на кнопку.

На форму наносятся компоненты – элементы управления. Часть из них являются видимыми, например, поле для ввода текста. Другие – скрытые, например, диалоговые окна. Такое окно появляется на экране только после совершения какого-либо действия.

Рассмотрим язык программирования Python.

Python – это интерпретируемый интерактивный объектно-ориентированный язык. Напрямую действия выполняются в командной строке, графический интерфейс не поддерживается. Но можно дополнительно подключить библиотеку Tkinter (Tk interface) или Qt5 (), а затем реализовать функции как графики, так и интерфейса, используя для последнего набор виджетов. В онлайн-средах эти библиотеки не поддерживаются, т.к. виджеты опираются непосредственно на функции операционной системы.

```
from tkinter import *
from tkinter import colorchooser
root = Tk()
def onChoose():
   color = colorchooser.askcolor()
   color_name=color[1]
   root.configure(background=color_name)
   root.title("Color")
   root.geometry("300x150+300+300")
btn=Button(text="OK",  padx="20", pady="8", command=onChoose)
btn.pack(side=BOTTOM) root.mainloop()
```



Рассмотрим каждую строку кода подробно:

1) подключение графической библиотеки;
2) дополнительное подключение диалогового окна выбора цвета;
3) инициализация графического окна;
4) заголовок функции пользователя;
5) присваивание переменной выбора цвета значения из диалогового окна; 6) присваивание полученного значения переменной для установки параметров цвета формы;
7) конфигурация (настройка) фонового цвета формы через полученное значение;
8) начало основной программы, установка заголовка формы;

9) установка размеров окна формы;

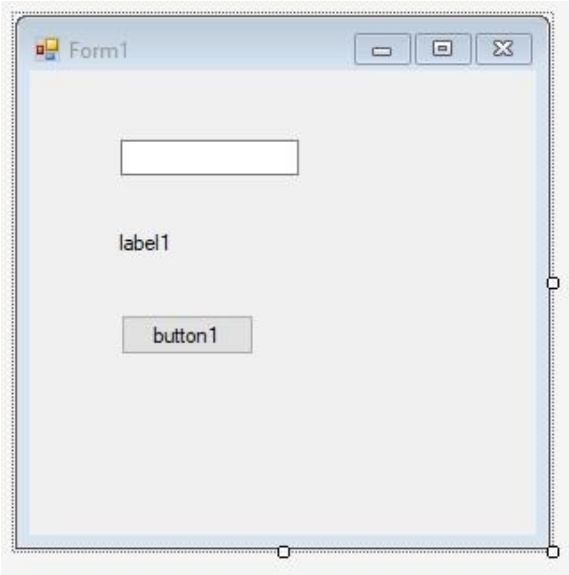
10) инициализация кнопки с параметрами относительного размера и определением действия по щелчку (т.е. вызов функции пользователя); 11) установка кнопки на форме снизу; 12) запуск окна приложения.

Для удобства работы с интерфейсом в Visual C++ и Python предусмотрено специальное приложение Qt Designer, позволяющее сохранить файл с формой, которое можно интегрировать в дальнейшую программу. Но все дальнейшие изменения придется выполнять в текстовом режиме.

Задача 1.2

Усложним задачу. Пусть на форме заданы три элемента управления: поле для ввода текста, неизменяемое поле для вывода и кнопка. При нажатии на кнопку производится расчет по формуле $\mathbf{y=x^2}$. Соответственно, x вводится в поле ввода, а результат y выводится в поле вывода.

1) Microsoft Visual C# (аналогичные манипуляции и в Visual Basic).

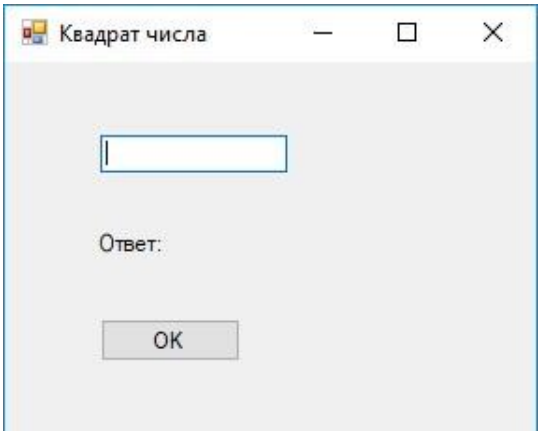


Поместим на пустую форму три компонента: TextBox, Label, Button. Установим свойства для каждого из них, включая саму форму.

Зададим заголовок формы в свойстве Text – «Квадрат числа».

Для наглядности можно запустить обработчик события загрузки формы — FormLoad(), в качестве кода которого явно записать нужные методы:

```
private void Form1_Load(object sender,
EventArgs e)        {          this.Text
= "Квадрат числа"
label1.Text = "Ответ: "
textBox1.Text = ""
button1.Text = "OK"
      }
   }
```



Но при нажатии на кнопку «OK» ничего не произойдет, т. к. у нас нет связанного с ней события.

Пока разберемся с первым обработчиком:

Будем учитывать, что C# чувствителен к регистру букв, поэтому не станем менять свойство Name (имя) у каждого компонента, оставив по умолчанию, чтобы не ошибиться в написании. this.Text – текст заголовка формы. Ключевое слово «this» означает, что все свойства и методы реализуются только в этой (данной) форме.

label1.Text – содержимое неизменяемого поля вывода. Слово «label» дословно переводится как «метка, ярлык». С помощью этого элемента мы можем программно вывести любую надпись, но изменить ее пользователем в процессе работы нельзя. Этот компонент удобно использовать для подписей блоков ввода информации и вывода результатов.

Некоторые другие свойства элемента label в C#:

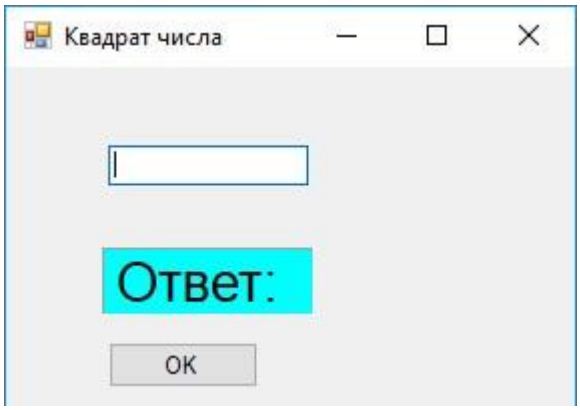
свойство	описание
AutoSize	Включает или выключает изменение размера в соответствии с размером шрифта для однострочной надписи. Значение True увеличивает или уменьшает размер компонента в соответствии с текстом, а значение False оставляет рамку неизменной.
BackColor	Цвет фона за текстом. Вместе со свойством BorderStyle (стиль обрамления) позволяет выделить фрагменты интерфейса.
ContextMenuStrip	Контекстное меню, вызываемое нажатием правой кнопки мыши. Можно запрограммировать копирование в буфер обмена содержимого этого элемента.
Cursor	Выбор значка для курсора, по умолчанию — стрелка.
Enabled	Если значение этого свойства стоит в False, то этот компонент неактивен. Для элемента Label это не столь принципиально, а для кнопок и полей ввода можно разрешать или запрещать действия.
Font	Группа свойств, отвечающих за вид и размер шрифта.
Image	Вместо однотонного фона можно поместить изображение и отдельно задать его свойства
Location	Позиция элемента форме на основе координат верхнего левого угла.
Locked	Запрещает или разрешает перемещать элемент по экрану и менять его размеры.

Size	Определяет точный размер элемента. Существуют и другие свойства, позволяющие настроить границы размера компонента.
Visible	Показать (true) или скрыть (false) элемент на экране.

Остальные функции используются реже.

Добавим значения некоторых свойств программно в существующий код:

```
... label1.BackColor =
Color.Aqua;
label1.BorderStyle =
BorderStyle.Fixed3D;
this.label1.Font = new
Font("Arial", 20F); ...
```



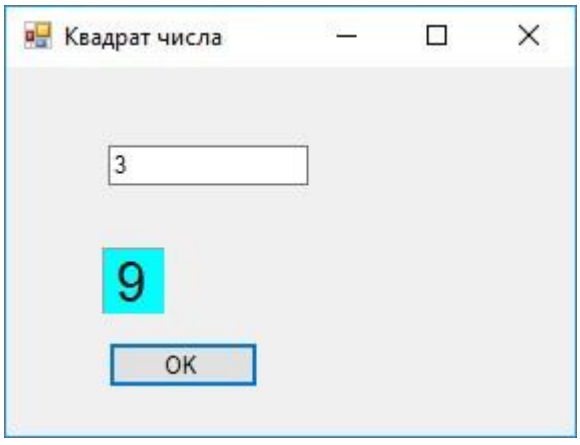
В первой строке мы поменяли свойство Color (цвет) на Aqua (бирюзовый), во второй изменили стиль рамки – fixed3D (фиксированная трехмерная граница). В языке C# нельзя использовать название шрифта в присваивании, поэтому для смены начертания и размера шрифта задается новый образец new Font. С помощью ключевого слова «new» можно добавить в коде любой элемент управления или переопределить набор параметров. TextBox1.Text — изменение текста в поле ввода.

Большинство свойств этого компонента совпадает с Label, однако, добавлены параметры, отвечающие за ввод. Строго говоря, выводить результат можно тоже с помощью этого элемента, но для защиты выходной информации от доступа пользователя лучше применять разные компоненты.

Особенностью текстового поля является то, что обрабатывать можно только строковые значения. Для работы с числовыми данными необходимо использовать функции преобразования текста в число и обратно (при выводе).

Например, подхват значения x из TextBox, расчет значения функции и вывод его в Label при обработки события нажатия на кнопку в C# имеет вид:

```
 private void button1_Click(object
sender,
EventArgs e)        {
float x =
Convert.ToSingle(textBox1.Text);
float y = x * x;
 label1.Text =
Convert.ToString(y);        }
```



Разберем строки кода.

Определим типы переменных как вещественные float (т. е. на языке математики — округленные действительные числа). Для преобразования строки ввода в числовое значение используется метод Convert.ToSingle(). Для обратного перевода числового результата в строку вывода — метод Convert.ToString(). Наша программа работает, но некорректно.

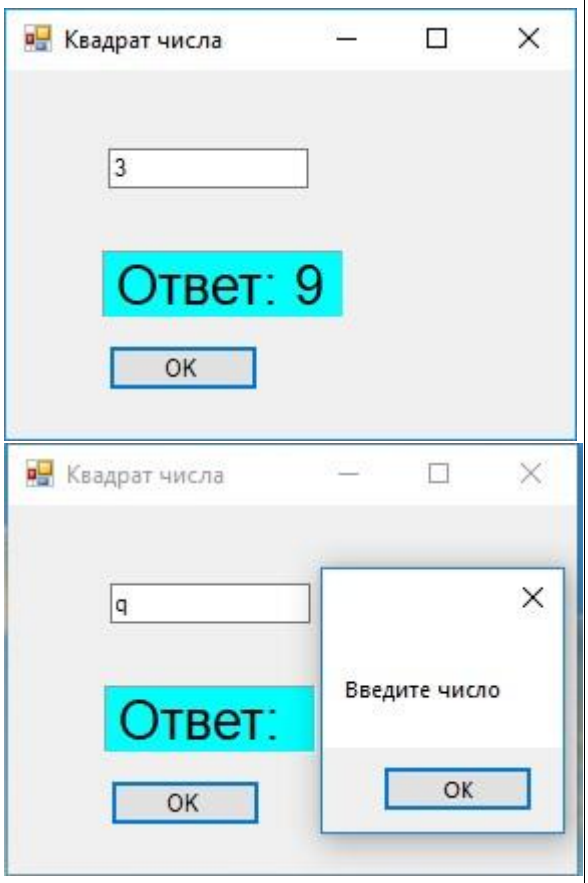
1 недостаток: пропало слово «ответ»;

2 недостаток: при вводе неправильных данных, например, буквы вместо числа, программа прекращает свою работу.

Перепишем программу с учетом замечаний:

```
 private void button1_Click(object sender,
EventArgs e)        {            float y=0;            float x;
if (float.TryParse(textBox1.Text.Trim(), out x))            {
                y = x * x;
label1.Text += Convert.ToString(y);
            }            else
 MessageBox.Show("Введите число");

        }
```



Метод .Parse используется для преобразование любого значения, преимущественно строкового, в значение определенного типа, а метод .**TryParse()** возвращает логическое значение, обозначающее произошло ли преобразование, и возвращает преобразованное значение в параметре out. В данном примере сначала заранее определили тип переменных, выходному значению присвоили начальное значение ноль. В строке **if**

(float.TryParse(textBox1.Text.Trim(), out x)) параметром метода .TryParse() является текст, находящийся в поле ввода, но для точности преобразования из него удалили крайние пробелы (с помощью метода .Trim()). Если содержимое текстового поля можно преобразовать в вещественное число, то результат поместим в переменную x. Тогда строка

x = Convert.ToSingle(textBox1.Text);

не нужна. Тогда после расчета к тексту в поле вывода добавится преобразованный в строковый тип результат, т.е.: label1.Text = label1.Text + Convert.ToString(y);

В коде программы использовался сокращенный оператор присваивания, как в языке C++.

В противном случае, когда в textbox введено не число, на экране с программой

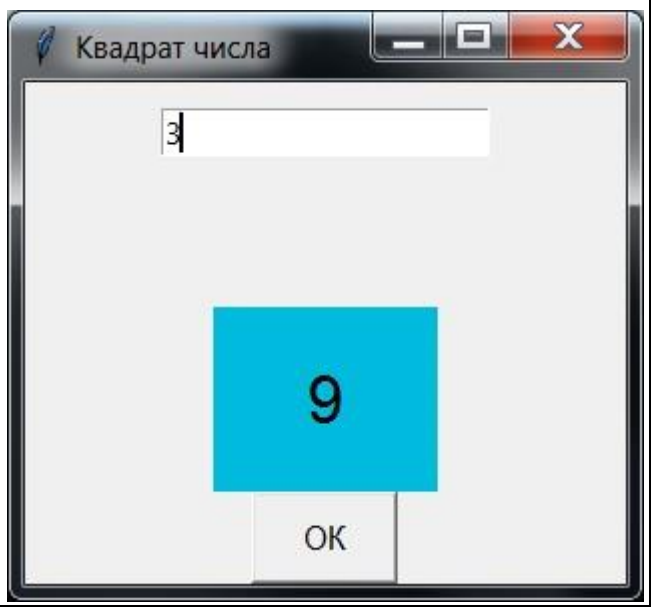
появляется диалоговое окно MessageBox с информацией о неверных данных. Этот элемент вместе с методом Show() – показать – имеет много дополнительных параметров, например, настройку заголовка окна, иллюстрации информационного сообщения и наименования кнопок. В простейшем случае достаточно одного параметра – текста сообщения, а кнопка «ОК» с функцией закрытия окна реализована по умолчанию автоматически.

В этом листинге приведены две процедуры – FormLoad() для установки параметров элементов управления и Button_Click() для проведения расчета при нажатии на кнопку.

В этой системе программирования есть удобная функция IsNumeric(), позволяющая выяснить, является ли аргумент этой функции числом или нет. Сравните этот код с подобным в Lazarus: procedure TForm1.FormCreate(Sender: TObject); begin
form1.caption := 'Квадрат числа'; Label1.caption := 'Ответ: '; edit1.Text := '';
Button1.caption := 'OK';
Label1.Color := clAqua;
Label1.BorderSpacing.InnerBorder := 1;
Label1.Font.Name:='Arial';
label1.Font.Size:=20; end;
procedure TForm1.Button1Click(Sender: TObject); var x, y: real; begin
if TryStrToFloat(edit1.Text,x) then begin y := x *x;
Label1.caption := Label1.caption + FloatToStr(y); end else
showmessage('Введите число'); end;

Эта же программа на языке Python

```
from tkinter import * import
math y = 0 def
click_button():     global y
    y =pow(int(x.get()),2)
label1=Label(text=str(y),
padx="35", pady="25",
height="1", width="2",font =
"Arial 20",
background="#0bd")
label1.pack(side=BOTTOM)
root = Tk()
root.title("Квадрат числа")
root.geometry("300x250")
btn = Button(text="OK",
padx="20", pady="8",
font="16",
command=click_button)
btn.pack(side=BOTTOM) x
= StringVar()
message_entry =
Entry(textvariable=x)
message_entry.place(relx=.5,
rely=.1, anchor="c")
root.mainloop()
```



Так, как **Python – интерпретатор,** то даже окно приложения запускается вместе с командной строкой, поэтому не обязательно добавлять в код условие правильности набора числа. При ошибочном вводе данных в командной строке можно увидеть подробное сообщение об ошибке.

Рассмотрим данный код построчно:

1) загрузка функций из графической библиотеки;
2) загрузка математической библиотеки;
3) установка начального значения вычисляемой функции;
4) заголовок функции пользователя, вычисляемой при нажатии на кнопку;
5) объявление глобальной переменной;
6) вычисление квадрата числа при выборе аргумента из текстового блока; 7) размещение результата в блоке вывода label с одновременной установкой параметров этого блока (преобразованный в текст результат, расстояния от границ окна, высота и ширина, шрифт и размер шрифта, цвет фона);
8) расположение элемента вывода на форме снизу;
9) основная программа: инициализация графического окна;
10) текст заголовка окна формы;
11) размеры формы;
12) параметры кнопки (нанесенный текст, относительное расположение, размер шрифта, привязка к процедуре – обработчику события);
13) расположение кнопки на форме снизу;
14) объявление строковой переменной;
15) связь переменной с содержимым текстового блока ввода (Entry); 16) относительное расположение текстового блока на форме; 17) открытие приложения.

2. Математические и физические задачи

Одними из самых популярных задач, решаемых на компьютере, являются задачи из области математики и физики, в которых требуется производить вычисления по формулам. Разберем классические задания по геометрии, алгебре и физике в разных системах программирования.

Задача 2.1

Постановка задачи. Собрать типовые формулы школьного курса планиметрии и стереометрии, описывающие свойства геометрических фигур, в единое приложение.

Внешнее описание: выделить 5 фигур на плоскости и 5 фигур в пространстве, установить основные свойства и измерения (площади, высоты, объемы и т.д.). Функциональная спецификация: решение по каждой фигуре производится в отдельном окне, доступ к каждому из которых осуществляется с помощью основного окна. Итого в проекте задействовано 11 форм. В файле ресурсов хранятся изображения фигур, которые используются как на главной форме (в виде изображения на кнопках), так и на второстепенных формах.

Геометрические фигуры:

1) Треугольник: ввести – 3 стороны; рассчитать - площадь, высоты, медианы, биссектрисы, углы, радиусы вписанной и описанной окружностей; изобразить – разные типы треугольника в зависимости от вычисленных углов (произвольный или прямоугольный).

2) Окружность: ввести – радиус; рассчитать – длину окружности, площадь круга; изобразить – одну окружность схематично.

3) Трапеция: ввести – 4 стороны; рассчитать – площадь, углы; изобразить – трапецию схематично.

4) Параллелограмм: ввести – 2 смежные стороны и угол (в градусах) между ними; рассчитать – площадь, диагонали; изобразить – фигуру в зависимости от равенства сторон и углов (произвольный параллелограмм, прямоугольник, квадрат, ромб).

5) Правильный многоугольник: ввести – количество сторон, длину стороны; рассчитать – площадь, углы, радиусы вписанной и описанной окружностей; изобразить – фигуру по числу сторон от треугольника до октаэдра.

6) Цилиндр: ввести – радиус и высоту; рассчитать – объем и площадь боковой поверхности; изобразить – цилиндр с нанесенными значениями радиуса и высоты.

7) Шар: ввести – радиус; рассчитать – объем и площадь сферической поверхности; изобразить – шар схематично.

8) Конус: ввести – радиус основания и высоту; рассчитать – объем и площадь боковой поверхности; изобразить – конус схематично.

9) Прямоугольный параллелепипед: ввести – длину, ширину, высоту; рассчитать – объем и площадь боковой поверхности; изобразить – параллелепипед схематично.

10) Четырехугольная пирамида: ввести – 2 стороны основания, высоту; рассчитать – объем и площадь боковой поверхности; изобразить – пирамиду схематично.

Система программирования: Microsoft Visual Studio C#.

Особенности реализации и возникшие трудности: доступ к каждой фигуре осуществляется 2 способами – через пункты меню и через кнопки на главной форме. На каждой вспомогательной форме есть кнопка «Вернуться». При переключении между формами может быть не корректная работа: закрытие основной формы, оставление приложения в «Диспетчере задач» системы. Для решения этих проблем в C# имеются методы управления появлением и скрытием форм show(), close() и hide(), собственный обработчик события системной кнопки окна «Закрыть», пункт меню «Выход» с кодом закрытия всех форм принудительно.

Достоинства приложения: проверка некорректного ввода данных для треугольника, многоугольника и др.; отсутствие лишних окон на экране. Недостатки приложения: дублирование частей кода; минимальная справочная информация.

Разработка интерфейса. В приложении использовались следующие компоненты: меню (MenuStrip), кнопки (Button), всплывающие подсказки (ToolTip), текстовые поля (TextBox), статический текст (Label), окно графического вывода (PictureBox).

В среде Microsoft Visual Studio существует редактор меню. С одной стороны, нужные пункты можно добавлять сразу на форме. С другой стороны, в контекстном меню элемента menuStrip1 (по умолчанию) можно вызвать диалоговое окно с возможностью редактирования.

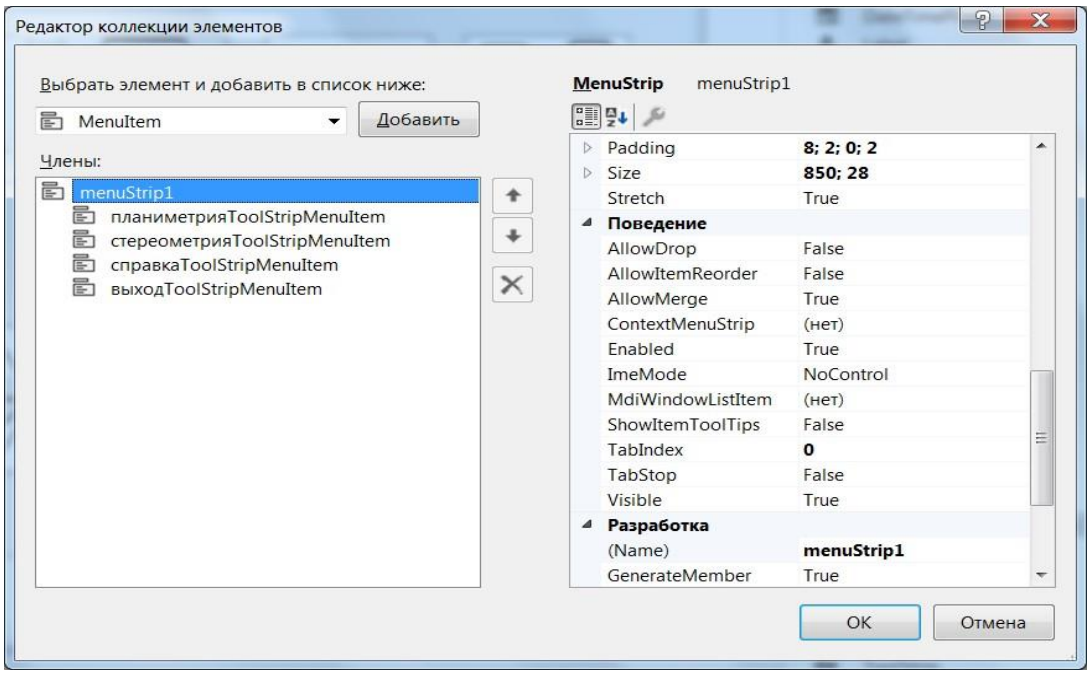


Рис. 2.1. Редактор меню C#

В этом редакторе можно менять пункты меню местами, настраивать клавиши быстрого доступа к пунктам, управлять размерами пунктов и расстоянием между ними.

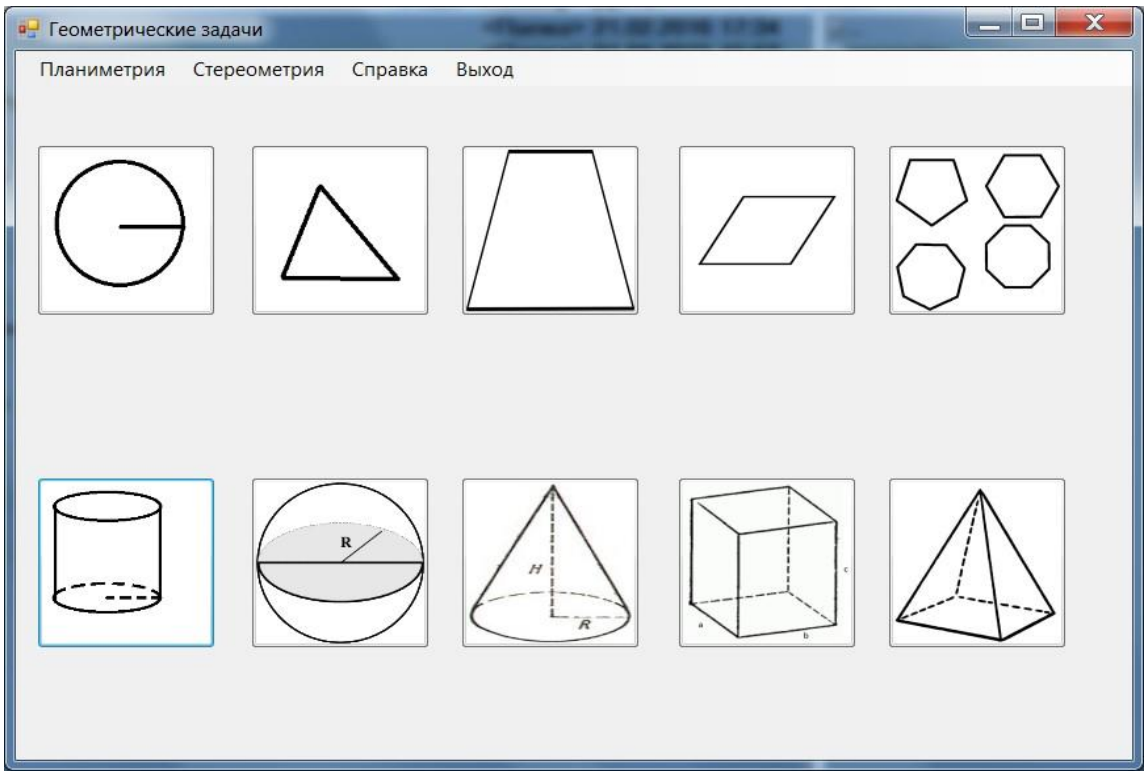
Для дополнительного выбора окна с вычисляемыми объектами на основной форме располагаются 10 кнопок. Изображение на кнопке выбирается из файла ресурсов с помощью свойства **BackgroundImage.** Для этого свойства можно установить дополнительный атрибут расположения рисунка

BackgroundImageLayout – растянуть изображение по размеру кнопки (Stretch), замостить плиткой (Tile), разместить по центру (Center), масштабировать (Zoom). Нужный параметр выбирается в зависимости от размера изображения и кнопки, при равных значениях можно оставить этот атрибут без изменений (None).

Всплывающие подсказки так же можно оформить с помощью элемента управления, но в данном приложении они устанавливаются программно в процедуре загрузки формы:

```
private void Form1_Load(object sender, EventArgs e)
    {
      ToolTip t = new ToolTip();
      t.SetToolTip(button1, "Цилиндр"); …
```

Добавить оставшиеся описания для остальных кнопок. Окончательно основная форма с выбором фигуры имеет вид:



В формах 2-11 для ввода данных пользователем применяются текстовые поля TextBox. Для того, чтобы осуществить числовую обработку данных из текстового поля, необходимо перевести тип данных из строкового в вещественный (или целочисленный) формат. В C# для этого используется команда:

r = Convert.ToDouble(textBox1.Text);

Вывод результата производится в статическое текстовое поле. Для этого сначала необходимо переконвертировать значение обратно из числового типа данных в строковый:

label3.Text = label3.Text+" "+Convert.ToString(v);

Т.к. в поле Label уже стояло значение «Объем равен», то к этому значению добавляем через пробел второе значение с результатом.

С помощью компонента PictureBox на форму можно добавить любое изображение. Если картинки будут меняться в зависимости от условия, то они устанавливаются не через окно свойств компонента, а в программе, например:

```
if (alpha == 90 || beta == 90 || gamma == 90)
            {
                pictureBox1.Image = Properties.Resources.pryamtr;
pictureBox1.SizeMode = PictureBoxSizeMode.StretchImage;
            }               else               {
                pictureBox1.Image = Properties.Resources.triangle;
pictureBox1.SizeMode = PictureBoxSizeMode.StretchImage;               }
```

В этом фрагменте программы записано условие: если один из углов равен 90 градусов, то вывести изображение прямоугольного треугольника из ресурсов, в противном случае изобразить обычный треугольник. Сразу в коде устанавливается атрибут положения изображения внутри компонента (PictureBoxSizeMode.StretchImage) – «растянуть по размеру».

Полный код с расчетом по формуле и выводом результатов выполняется при нажатии на кнопку «Рассчитать».

На каждой из вспомогательных форм располагается вторая кнопка «Вернуться на главную» со следующим кодом:

```
private void button2_Click(object sender, EventArgs e)
    {
      Form1 form1 = new Form1();          form1.Show(this);
      Hide();
    }
```

Алгоритм этого блока следующий:

1) инициализировать форму Form1;
2) показать эту форму на экране;
3) спрятать с экрана предыдущую форму (но не закрывать).

Для принудительного закрытия формы надо изменить код стандартной кнопки «Закрыть» (красный крестик). В противном случае главное окно не появится на экране.

```
private void Form2_FormClosed(object sender, FormClosedEventArgs e)
    {
      Form form1 = Application.OpenForms[0];
      form1.Show();
    }
```

Для инициализации процедуры «FormClosed» надо в свойствах элемента «Форма» (например, Form2) выбрать пункт «События» и найти событие «FormClosed». Все добавленные в проект формы автоматически нумеруются, начиная с нуля, поэтому Form1 можно не инициализировать заново, а выбрать из списка Application.OpenForms[0]. В главной форме для корректного закрытия приложения добавлен пункт меню «Выход» с кодом:

```
private void выходToolStripMenuItem_Click(object sender, EventArgs e)
    {
      Application.Exit();

    }
```

Пункт «Справка» реализован упрощенно. Это обычное диалоговое окно вывода MessageBox:

```
private void справкаToolStripMenuItem_Click(object sender, EventArgs e)
    {
      MessageBox.Show("Текст справки", "Справка", MessageBoxButtons.OK,
MessageBoxIcon.Question);
    }
```

Задача 2.2

Пусть график выводится в отдельном окне, а в результате могут присутствовать комплексные корни. Реализуем эту подзадачу с помощью системы программирования Microsoft Visual Studio Python (или приложения IDLE Python 3.6).

Разработка интерфейса: графический пользовательский интерфейс был разработан с помощью библиотеки tkinter. На форме помещаются 5 полей ввода коэффициентов (Entry) с сопровождающим текстом, статическое поле вывода результата (Label) и две кнопки – **«Решить»** и **«Построить».** При нажатии на последнюю кнопку открывается дочернее окно:
(win = Toplevel(root,relief=SUNKEN,bd=10,bg="lightblue")), в котором происходит отрисовка графика с помощью метода canvas.

Особенности реализации и возникшие трудности: по аналогии с предыдущей реализацией для решения уравнений используются четыре функции пользователя для уравнения степеней 1, 2, 3 и 4.

Для упрощения решения уравнения четвертой степени вместо метода Феррари применяется метод подбора целочисленных корней среди делителей свободного члена (внутри цикла):

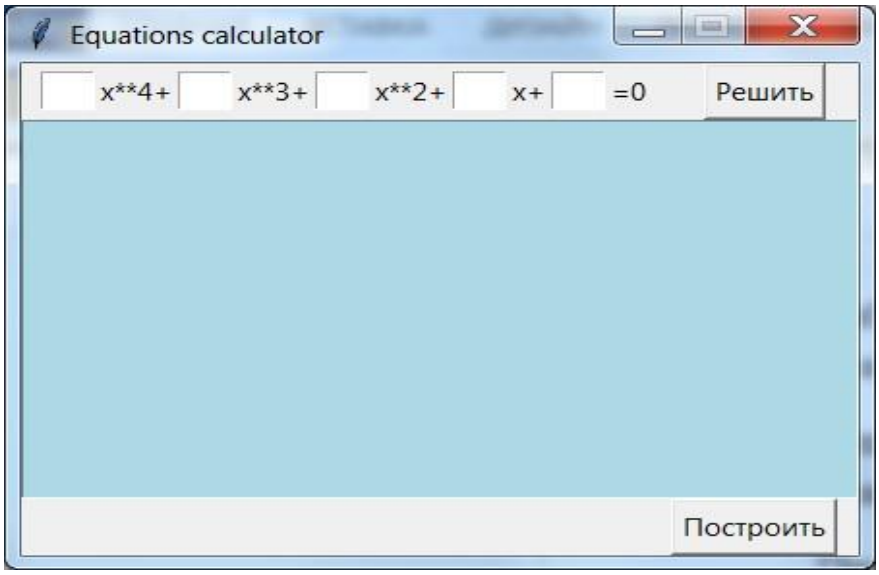
```
k=abs(round(e))
for i in range(-k,k+1):
        if int(a*i**4+b*i**3+c*i**2+d*i+e)==0:
                u=u+1
                text=text+"x%s= %s\n" %(u,i)
```

В этом фрагменте *e* – свободный член, *u* – номер корня.

Для работы с комплексными числами в языке Python достаточно подключить библиотеку **cmath**, поэтому алгоритмы решения кубического и квадратного уравнений упрощены.

При вводе коэффициентов уравнения записывается функция **handler(),** в которой происходит проверка чисел на соответствие шаблону «0.0». Таким образом, функция расчета «оборачивается» в функцию соответствия параметров **inserter().**

В случае несовпадения в текстовое поле выводится сообщение о неправильном вводе. Окончательно основная форма решения задачи имеет вид:



Приведем фрагмент кода с построением графика:

```
def click_button():
   a_val = float(a.get())
         def fu(x):
         y=a_val*x**4/1000+b_val*x**3/100+c_val*x**2/10+d_val*x+e_val*10
         return y
   win = Toplevel(root,relief=SUNKEN,bd=10,bg="lightblue")
   win.title("График")
   win.minsize(width=400,height=200)
   points=[]
   for x in range(-480,480):
      y=int(fu(x))
       x=x+180
```

```
        y=240-y
        pp=(x,y)
        points.append(pp)
        canvas=Canvas(win)
       canvas["height"]=360
       canvas["width"]=480
       canvas["background"]="#eeeeff"
       canvas["borderwidth"]=2
       canvas.pack({"side":"bottom"})
       canvas.create_line(points,fill="blue",smooth=1, width=2)
    canvas.create_line(180,360,180,0,width=2,arrow=LAST) #ось y
    canvas.create_line(0,240,480,240,width=2,arrow=LAST) #ось x
```

1) Функция обработки кнопки **«Построить».**
2) Связь значений из полей ввода с числовыми переменными. Эти строки не совсем корректны, т.к. в предыдущей функции **handler()** уже использовался ввод переменных с их проверкой. В этом фрагменте проверка не осуществляется. Целесообразно оформить эти действия в общий класс, но в этом коде ограничимся дублированием.
3) Отдельная функция ***y=fu(x),*** которую можно редактировать внутри кода.
4) Инициализация нового окна и установка свойств канвы рисования.
5) Создание массива точек и построение графика по этим точкам.
6) Вычерчивание осей координат.

8.1 Рекомендуемая литература

8.1.1 Основная литература

1. В.А.Благодатских, В.А.Волнин, К.Ф.Поскакалов. Стандартизация разработки программных средств. Учебное пособие. Москва. Финансы и статистика. 2005г, 288 стр 2.Зыль С.Н. Проектирование, разработка и анализ программного обеспечения систем реального времени. – СПб.: БХВ-Петербург,2010. – 336 с.
2. Мостовой Я.А. Лекции по технологии разработки программного обеспечения. Учебное пособие. Издательство ПГУТИ. Самара.2014г.178 с.
3. Бобровский С. Программная инженерия. Технологии Пентагона на службе российских программистов. СПБ.: Питер, 2003 – 249 с.
4. Осллэндер Д.М. Управляющие программы для механических систем: ООП систем реального времени. Пер. с англ. – М.: Бином. Лаборатория Знаний.2009. – 413с.
5. Макконел С. Совершенный код. Мастер класс/ Пер. с англ. – М. : Издательско-торговый дом» Русская редакция»; СПб.: Питер, 2005. – 896 стр.

8.1.2 Дополнительная литература:

1. Д.И. Козлов, Г.П. Аншаков, Я.А. Мостовой, А.В. Соллогуб. Управление космическими аппаратами зондирования Земли. Компьютерные технологии. - М.: Машиностроение.
2. 1998г.
3. А.Л.Фридман. Основы обьектно-ориентированной разработки программных систем.- М.: Финансы и статистика.2000.-192с.
4. Якобсон А., БучГ., Рамбо ДЖ. Унифицированный процесс разработки программного обеспечения .-Спб.: Питер. 2002.-496с.
5. Терехов А.Н. Технология программирования: учеб. Пособие/А.Н. Терехов. 2-е изд., -М.:
6. Интернет Университет информационных технологий; БИНОМ. Лаборатория знаний,2007. –5. Гецци К., Джазайери М., Мандриоли Д. Основы инженерии программного обеспечения.2-е изд.: Пер. с англ. – СПб.: БХВ-Петербург, 2005. _ 832 с. 148 стр.

8.2 Средства обеспечения освоения дисциплины

8.2.1 Методические указания и материалы по видам занятий

Методические указания и исходные данные для практических и лабораторных работ по «Технологии разработки программных продуктов».

8.2.2 Программное обеспечение современных информационно коммуникационных технологий по видам занятий

Программное обеспечение для выполнения практических работ

8.2.3 Контрольные вопросы для самопроверки

Раздел 1
Роль ПО и компьютеров в производстве, социальной жизни и науке.
Инженерия ПО
Проблемы разработки ПО

Раздел 2
Технология разработки ПО и качество ПО
Характеристики качества ПО Факторы, влияющие на качество ПО Системный подход к разработке ПО. Временной и "пространственный " аспекты системного подхода
Этапы жизненного цикла ПО. Каскадная модель жизненного цикла ПО.

Раздел 3
Три группы процессов создания ПО
Жизненный цикл ПО и процессы верификации.
Тестирование, верификация, валидация. Различие в понятиях. V образная модель жизненного цикла ПО Спиральная модель ЖЦ ПО.
«Тяжелые и легкие» технологии разработки ПО. Экстремальное (XP) программирование

Раздел 4

Стандарты по разработке ПО.
Роль и назначение стандартов, требования стандартов
Три вида программных разработок с точки зрения технологии их создания Разбиение СТС на подсистемы.
Параллельная разработка подсистем
Виды документов, выпускаемых на ПО по этапам разработки системы.
Задачи, решаемые на различных стадиях проектирования системы и ПО.

Раздел 5
Итеративный характер проектирования ПО.
Стадии проектирования.
Цена ошибок проектирования.
Проектирование, основанное на моделировании
(Model-Based Systems Engineering - MBSE) CASE технологии разработки ПО Задачи и результаты архитектурного проектирования ПО. Технология Rational Rose,UML
Структура системы, иерархия управления и структура ПО
Функциональные задачи и декомпозиция СТС Пример иерархической структуры ПО СТС

Раздел 6
Цикличность решения задач управления в системах с ЦВМ Временная диаграмма работы системы.
Представления работы ПО СТС в виде набора «сечений» диаграммы, выполняемых последовательно.
Временная диаграмма работы СТС при выполнении одной из функциональных задач
Представление работы ПО СТС в виде набора параллельных процессов.

Раздел7
Многозадачная работа ПО СТС. Причины многозадачности.
Задачи и процессы. Контекст процесса
Схема возможных вариантов совместного использования информации взаимодействующими процессами

Повышение эффективности ПО за счет параллельных вычислений

Раздел 8
Критический ресурс ЦВМ. Основное правило защиты ресурсов ЦВМ
Синхронизация процессов
Взаимное исключение процессов. Использование мьютексов
Задача синхронизации «Читатели-писатели» Задачи синхронизации. «Обедающие философы»
Технология синхронизации ПО.
Система Intel Thread Checker (ITC)

Раздел 9
Конструирование ПО
Минимизация сложности ПО и стратифицированная декомпозиция
Приспособленность ПО к изменениям
Рефакторинг ПО
Особенности конструирования программ для встроенных ЦВМ
Фиксированное распределение памяти
Проектирование снизу-вверх и проектирование сверху-вниз.
Программные заглушки и их использование
Основные понятия структурного подхода к проектированию ПО. Основные понятия объектно - ориентированного подхода к проектированию ПО.

Раздел 10
Конструирование ПО – эвристический процесс
Желательные характеристики проекта ПО
Определение объектов реального мира и искусственных объектов предметной области
Локализация информации в ПО и излишнее еѐ дублирование в ПО
Сокрытие информации – основной принцип конструирования
Определение частей объектов, видимых другим объектам
Определение областей вероятных изменений
Минимизация связей

Раздел 11
Желательные характеристики проекта ПО
Защита программ от неправильных входных данных
Процедуры обработки ошибок и «утверждения» - разница в понятиях.
Использование утверждений
Способы обработки ошибок
Изоляция повреждений ПО, вызванных проявившейся ошибкой Риски при использовании глобальных переменных и способы их уменьшения.

Раздел 12
Виды контроля работы ПО.
Контроль работы ПО встроенными средствами Эталоны для контроля работы ПО Стратегии безопасности.
Три уровня реакции ПО на обнаруженную ошибку
Отказоустойчивые системы
Перечень нештатных ситуаций
Аварийная защита

Раздел 13

Ошибки ПО, отладка и тестирование ПО.
Анализ обнаруживаемых в ПО ошибок и важность его проведения
Классификация ошибок ПО
Статическая отладка и динамическая отладка
Принцип «белого» и «черного» ящика при динамической отладке ПО.
Функциональная отладка

Раздел 14
Структурная динамическая отладка
«Особые точки» при работе программ
Автономная отладка (АО) и комплексная отладка (КО) ПО Драйверы и заглушки при автономной отладке Последовательность действий при отладке ПО.

Раздел15
Принципы выделения маршрутов при комплексной отладке Приближенный метод оценки числа вариантов для отладки ПО Регулярное и случайное дерево структуры ПО и устойчивость его структурного параметра
Контроль отлаженности ПО в процессе отладки.

Раздел 16
Проблема генерации данных на комплексную отладку
Преимущества математического моделирования внешней среды
Проблемы наследуемого ПО
Мобильность ПО

8.2.4 Критерии оценки знаний, умений и навыков

Итоговой формой контроля знаний, умений и навыков по дисциплине является **экзамен**

Экзамен проводится только при выполненных студентом лабораторных работах по билетам, которые включают 2 теоретических вопроса и задачу. Оценка знаний студентов производится по следующим критериям: оценка «*отлично*» выставляется студенту, если при ответе на поставленные вопросы он показывает владение знаниями всего программного материала, концептуальнопонятийным аппаратом, научным языком и терминологией соответствующей научной области, логически корректно и убедительно излагает свои знания.

оценка «*хорошо*» выставляется студенту, если при ответе на поставленные вопросы он показывает владение знаниями узловых проблем программы и основного содержания лекционного курса, умение пользоваться концептуально-понятийным аппаратом в процессе анализа основных проблем программы, в целом логически корректное, но не всегда точное и аргументированное изложение ответа.

оценка «*удовлетворительно*» выставляется студенту, если при ответе на поставленные вопросы он показывает владение фрагментарными, поверхностными знаниями важнейших разделов программы и содержания лекционного курса, испытывает затруднения с использованием научно-понятийного аппарата и терминологии учебной дисциплины, стремление логически определенно и последовательно изложить ответ.

оценка «*неудовлетворительно*» выставляется студенту, если он не отвечает на поставленные вопросы, либо имеет отрывочное представление учебно-программного материала.

9. Материально-техническое обеспечение дисциплины

9.1 Учебно-лабораторное оборудование

Для проведения лабораторного практикума предназначена специализированная лаборатория – «Компьютерный класс» (ауд. № 315-321).
Лабораторные работы выполняются на ПК, с установленным необходимым ПО.