МИНИСТЕРСТВО ПО РАЗВИТИЮ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И КОММУНИКАЦИЙ РЕСПУБЛИКИ УЗБЕКИСТАН

ТАШКЕНТСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ ИМЕНИ МУХАММАДА АЛ-ХОРАЗМИЙ

ФАКУЛЬТЕТ ИНФОРМАЦИОННАЯ БЕЗОПАСНОСТЬ

Кафедра **Криптология**

Методическое пособие по выполнению практических работ учебной дисциплины «Безопасность программных средств» для студентов направлении информационной безопасности

Ташкент - 2022

Авторы: Преподаватели кафедры "Криптологии" профессор D.Sc. Д.Я. Иргашева, доцент PhD. З.Т. Худойкулов, О.О. Турсунов, И.С. Олимов. Методическое пособие по выполнению практических работ по учебной дисциплине «Безопасность программных средств» для студентов бакалавриатуры специальностей информационной безопасности - Ташкент: ТУИТ. 2022.-92 с.

Безопасность программных средств – целью изучения дисциплины является развивать навыков программирования программного обеспечения с точки зрения безопасности. Формирования навыков разработки безопасных программных средств, знаний о моделях и методологиях жизненного цикла разработки программного обеспечения, о принципах и подходов к решению безопасного коддинга, а также формирование умений по применению современных технологий, выбора средств и инструментов тестирования безопасности программных средств.

Методическое пособие для студентов Ташкентского университета информационных технологий имени Мухаммада аль-Хорезми. Руководство рекомендовано к публикации решением Научно-методического совета Ташкентского университета информационных технологий имени Мухаммада аль-Хорезми (протокол № «___» «_____» «___» 20__ г.

Практическая работа №1

Тема: Практическое применение жизненного цикла разработки программного обеспечения

Цель работы: Ознакомиться с жизненным циклом и моделями разработки программного обеспечения, определить достоинства и недостатки моделей, выбрать модель построения информационной системы индивидуального проектного задания и программные средства, составить план реализации индивидуального проектного задания.

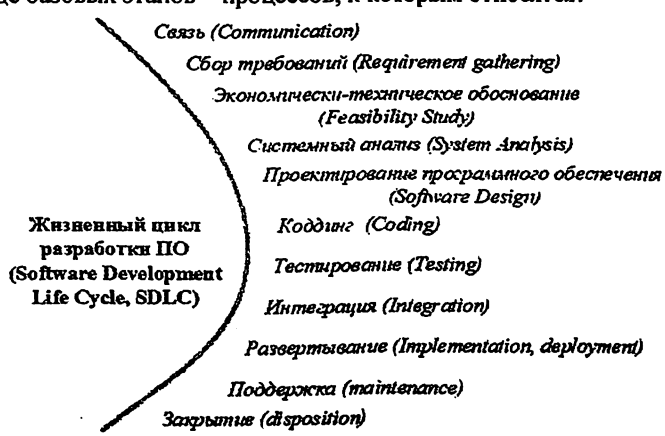
Теоритическая часть

Разработка информационной системы является весьма сложной задачей, процесс решения которой разбивается на определённое количество этапов. К числу ключевых этапов относятся: анализ требований, проектирование, реализация, тестирование и внедрение. Объединение этих этапов в один процесс приводит к понятию жизненного цикла ИС.

Жизненный цикл (ЖЦ) информационной системы – непрерывный процесс, который начинается с момента принятия решения о необходимости создания системы и заканчивается в момент её полного изъятия из эксплуатации.

Модель жизненного цикла программного обеспечения представляет собой совокупность упорядоченных во времени, взаимосвязанных и объединенных в стадии работ, выполнение которых необходимо и достаточно для создания ПО, соответствующего заданным требованиям

Наиболее общим представлением жизненного цикла ПО является модель в виде базовых этапов – процессов, к которым относятся:



1.1 – рисунок. Жизненный цикл разработки ПО.

Модель жизненного цикла ИС – структура, описывающая процессы, действия и задачи, которые осуществляются в ходе разработки, функционирования и сопровождения программного обеспечения в течение всей жизни ИС, от определения требований до завершения её использования.

В дальнейшем, рассматривая различные модели ЖЦ, для простоты будем использовать следующий набор этапов:

1. Анализ (разработка требований).
2. Проектирование (создание проекта).
3. Реализация (программирование).
4. Тестирование (исправление ошибок).
5. Внедрение (ввод в эксплуатацию).

К настоящему времени наибольшее распространение получили следующие основные модели ЖЦ:

– Каскадная (waterfall). Простая в реализации, подходит для коротких проектов с нулевым риском и фиксированными требованиями.

– V-образная. Базируется на каскадной. Эта модель SDLC подразумевает контроль качества на каждом из этапов. Подходит для проектов, для которых ошибки могут иметь фатальные последствия, когда критически важна точность выполнения требований.

– Модель эволюционного прототипирования. Опять же, в основе – waterfall. При прохождении каждого этапа сразу же происходят необходимые доработки проекта на основе отзывов клиента (заказчика). Создается несколько прототипов, один из которых в итоге дорабатывается и отправляется в продакшн.

– Итеративная и инкрементальная. Решение разрабатывается модулями при реализации серии циклов. А при работе над каждым модулем используется все та же каскадная модель.

– Спиральная. Включает в себя черты предыдущих. Используется для сложных, крупных проектов с частыми релизами, подходит для ПО с неясными требованиями.

– Гибкие методологии. Их более 50. И многие, опять же, могут включать черты рассмотренных выше.

Начиная с этапа разработки SDLC для тестирования приложения безопасность обеспечивается с помощью:

– применения практик безопасного программирования,
– проведения автотестов,
– динамического анализа кода,
– организации пентестов (испытаний на проникновение),
– функционального тестирования,
– тестирования протоколов,
– мониторинга угроз, реагирования на инциденты (после развертывания, в ходе эксплуатации).

Типичный жизненный цикл разработки программного обеспечения состоит из следующих этапов:

Этап 1: планирование и анализ потребностей

Анализ требований является наиболее важным и фундаментальным этапом в SDLC. Он выполняется старшими членами команды при участии клиентов, отдела продаж, исследований рынка и экспертов в данной области.

Эта информация затем используется для планирования базового проектного подхода и проведения технико-экономического обоснования продукта в экономической, эксплуатационной и технической областях.

Планирование требований по обеспечению качества и выявление рисков, связанных с проектом, также выполняется на этапе планирования. Итогом технико-экономического обоснования является определение различных технических подходов, которые можно использовать для успешной реализации проекта с минимальными рисками.

Этап 2: определение требований

Как только анализ требований будет выполнен, следующим шагом будет четкое определение и документирование требований к продукту и их утверждение от клиента или аналитиков рынка. Это делается с помощью документа SRS (Спецификация требований к программному обеспечению), который содержит все требования к продукту, которые должны быть спроектированы и разработаны в течение жизненного цикла проекта.

Этап 3: проектирование архитектуры продукта

SRS – это ориентир для разработчиков продукта, чтобы предложить лучшую архитектуру для продукта, который будет разработан. На основании требований, указанных в SRS, обычно предлагается несколько подходов к проектированию архитектуры продукта, которые документируются в спецификации DDS – документ проектирования.

Эта DDS рассматривается всеми важными заинтересованными сторонами и на основе различных параметров, таких как оценка рисков, надежность продукта, модульность дизайна, бюджетные и временные ограничения, для продукта выбран наилучший подход к проектированию.

Подход к проектированию четко определяет все архитектурные модули продукта, а также его связь и представление потока данных с внешними и сторонними модулями (если таковые имеются). Внутренний дизайн всех модулей предлагаемой архитектуры должен быть четко определен с мельчайшими деталями в DDS.

Этап 4: Создание или разработка продукта

На этом этапе SDLC начинается фактическая разработка и сборка продукта. Программный код генерируется в соответствии с DDS на этом этапе. Если дизайн выполнен детально и организованно, генерация кода может быть выполнена без особых хлопот.

Разработчики должны следовать руководящим принципам кодирования, определенным их организацией, и для генерации кода используются такие инструменты программирования, как компиляторы, интерпретаторы, отладчики и т. Д. Для кодирования используются различные языки программирования высокого уровня, такие как C, C ++, Pascal, Java и PHP. Язык программирования выбирается в зависимости от типа разрабатываемого программного обеспечения.

Этап 5: Тестирование продукта

Этот этап обычно является подмножеством всех этапов, так как в современных моделях SDLC тестирование в основном затрагивает все этапы SDLC. Однако этот этап относится только к этапу тестирования продукта, на котором дефекты продукта регистрируются, отслеживаются, исправляются и повторно тестируются до тех пор, пока продукт не достигнет стандартов качества, определенных в SRS.

Этап 6: развертывание на рынке и сопровождение

Как только продукт протестирован и готов к развертыванию, он официально выпускается на соответствующем рынке. Иногда развертывание продукта происходит поэтапно в соответствии с бизнес-стратегией этой организации. Продукт может быть сначала выпущен в ограниченном сегменте и протестирован в реальной бизнес-среде (UAT-Пользовательское тестирование).

Затем, основываясь на отзывах, продукт может быть выпущен как есть или с предлагаемыми улучшениями в сегменте таргетинга. После того, как продукт выпущен на рынок, его обслуживание выполняется для существующей клиентской базы.

Практическая часть

Модель водопада – Применение

Каждое разработанное программное обеспечение отличается и требует соответствующего подхода SDLC, основанного на внутренних и внешних факторах. Некоторые ситуации, в которых использование модели Водопад является наиболее подходящим:

– Требования очень хорошо задокументированы, понятны и зафиксированы.

– Определение продукта является стабильным.

– Технология понятна и не динамична.

– Здесь нет двусмысленных требований.

– Достаточные ресурсы с необходимым опытом доступны для поддержки продукта.

– Проект короткий.

Требования очень хорошо задокументированы, понятны и зафиксированы.

Определение продукта является стабильным.

Технология понятна и не динамична.

Здесь нет двусмысленных требований.

Достаточные ресурсы с необходимым опытом доступны для поддержки продукта.

Проект короткий.

Итерационная модель – приложение

Как и другие модели SDLC, у итеративной и инкрементальной разработки есть некоторые специфические приложения в индустрии программного обеспечения. Эта модель чаще всего используется в следующих сценариях:

– Требования к полной системе четко определены и понятны.

– Основные требования должны быть определены; однако некоторые функции или запрошенные улучшения могут со временем развиваться.

– Есть время для ограничения рынка.

– Новая технология используется и изучается командой разработчиков во время работы над проектом.

– Ресурсы с необходимыми наборами навыков недоступны и планируется использовать на контрактной основе для конкретных итераций.

– Есть некоторые особенности и цели высокого риска, которые могут измениться в будущем.

– Требования к полной системе четко определены и понятны.

– Основные требования должны быть определены; однако некоторые функции или запрошенные улучшения могут со временем развиваться.

– Есть время для ограничения рынка.

Новая технология используется и изучается командой разработчиков во время работы над проектом.

Ресурсы с необходимыми наборами навыков недоступны и планируется использовать на контрактной основе для конкретных итераций.

Есть некоторые особенности и цели высокого риска, которые могут измениться в будущем.

Применение спиральной модели

Спиральная модель широко используется в индустрии программного обеспечения, так как она синхронизирована с естественным процессом разработки любого продукта, то есть обучением со зрелостью, которое сопряжено с минимальным риском для клиента, а также для фирм-разработчиков.

Следующие указатели объясняют типичное использование спиральной модели

– При наличии бюджетных ограничений важна оценка рисков.

– Для проектов со средней и высокой степенью риска.

– Долгосрочная приверженность проекту из-за потенциальных изменений экономических приоритетов, так как требования меняются со временем.

– Клиент не уверен в своих требованиях, как правило, так.

– Требования сложны и нуждаются в оценке, чтобы получить ясность.

– Новая линейка продуктов, которая должна выпускаться поэтапно, чтобы получить достаточное количество отзывов клиентов.

– Значительные изменения ожидаются в продукте в течение цикла разработки.

При наличии бюджетных ограничений важна оценка рисков.

Для проектов со средней и высокой степенью риска.

Долгосрочная приверженность проекту из-за потенциальных изменений экономических приоритетов, так как требования меняются со временем.

Клиент не уверен в своих требованиях, как правило, так.

Требования сложны и нуждаются в оценке, чтобы получить ясность.

Новая линейка продуктов, которая должна выпускаться поэтапно, чтобы получить достаточное количество отзывов клиентов.

Значительные изменения ожидаются в продукте в течение цикла разработки.

V- Модель — Применение

Применение V-модели практически совпадает с моделью водопада, поскольку обе модели имеют последовательный тип. Требования должны быть очень четкими до начала проекта, потому что возвращение и внесение изменений обычно обходится дорого. Эта модель используется в области медицинского развития, поскольку является строго дисциплинированной областью.

Следующие указатели являются одними из наиболее подходящих сценариев для использования приложения V-Model.

– Требования четко определены, четко задокументированы и зафиксированы.
– Определение продукта является стабильным.
– Технология не динамична и хорошо понимается командой проекта.
– Здесь нет двусмысленных или неопределенных требований.
– Проект короткий.

Требования четко определены, четко задокументированы и зафиксированы.
Определение продукта является стабильным.
Технология не динамична и хорошо понимается командой проекта.
Здесь нет двусмысленных или неопределенных требований.
Проект короткий.

Задание.

1. Ознакомиться с теоретическими сведениями по практической работе
2. Определить достоинства и недостатки моделей ЖЦ
3. Выбрать и обосновать выбор модели ЖЦ для выполнения индивидуального проектного задания.
4. Сформировать план построения ИС индивидуального проектного задания, с использованием программных средств.

Контрольные вопросы:

1. Что отражает модель ЖЦ ИС?
2. Организационные процессы внедрения ИС;
3. События, происходящие с системой в процессе ее создания и использования;
4. Процесс проектирования ИС

Практическая работа №2

Тема: Формирования требований к различным программным средствам

Цель работы: Ознакомиться с жизненным циклом и моделями разработки программного обеспечения, определить достоинства и недостатки моделей, выбрать модель построения информационной системы индивидуального проектного задания и программные средства, составить план реализации индивидуального проектного задания.

Теоритическая часть

Проекты программных средств различаются по уровню сложности, масштабу и необходимому качеству. Они имеют различное назначение, содержание и относятся к разным областям применения. Поэтому существует потребность в четко организованном процессе, методах формализации и управления требованиями к конкретным программным продуктам. Чаще всего проблемами, с которыми встретились не достигшие своих целей проекты программных продуктов, являются: ***недостаток информации от пользователя или заказчика о функциях проекта, неполные, некорректные требования, а также многочисленные изменения требований и спецификаций***. Это происходит потому, что зачастую разработчики и заказчики считают, что "даже если мы не очень точно знаем, чего хотим достичь, лучше быстрее приступить к реализации проекта, так как мы и так выбились из графика и нам некогда размышлять. Мы можем уточнить требования позднее". Подобный подход приводит к ***хаотическим, неупорядоченным действиям*** при разработке ПС, когда никто не знает, что на самом деле хочет заказчик и пользователь, и как в действительности функционирует созданная на данный момент система и/или программный продукт.

Формализация и управление требованиями – это систематический метод выявления, организации и документирования требований к системе и/или ПС, а также процесс, в ходе которого вырабатывается и обеспечивается *соглашение* между заказчиком и выполняющими проект специалистами, в условиях меняющихся требований к системе – рис. 6.1. Развитие программной инженерии, её обозримое будущее, связаны с непрерывным возрастанием сложности, поэтому разработкой ПС должны заниматься *хорошо организованные* и обученные коллективы – *команды* разработчиков. Каждый член команды в той или иной степени должен привлекаться к управлению и формализации требований к проекту. Команде необходимо выработать профессиональные приемы для понимания потребностей пользователей, управления масштабом ПС, структурой и построением системы, удовлетворяющей эти потребности.

Команда должна применить методы и процессы для того, чтобы ***понять решаемую проблему заказчика до начала разработки ПС***. Для этого следует использовать ***метод анализа, выявления и освоения проблемы и интересов заказчика***:

- достигнуть соглашения между заказчиком и разработчиком по определению проблемы, целей и задач проекта;

- выделить основные причины – проблемы, являющиеся её источниками и стоящие за основной проблемой проекта системы и ПС;

- выявить заинтересованных лиц и пользователей, чье коллективное мнение и оценка в конечном итоге определяет успех или неудачу проекта;

- определить, где приблизительно находятся область и границы возможных решений проблем;

- понять ограничения, которые будут наложены на проект, команду и решения проблем.

В результате необходимо ***предложить заказчику возможные решения рассматриваемой проблемы***. Для анализа проблемы можно использовать разнообразные методы. В частности, моделирование бизнес-процессов, специальный ***метод анализа проблем***, который достаточно хорошо зарекомендовал себя для сложных информационных систем, осуществляющих поддержку ключевых инфраструктур. Выявленные на данном этапе прецеденты, могут позднее применяться для определения совокупности требований к ПС. Методы системного анализа, позволяют осуществить разбиение сложной системы на подсистемы. Этот процесс помогает понять, каким общим целям должно служить ПС. При этом часто оказывается, что в чем-то усложняются требования, создавая новые подсистемы, для которых в свою очередь нужно заниматься разработкой и пониманием требований.

Понимание потребностей пользователей необходимый организационный этап, так как разработчики редко получают совершенные спецификации требований к создаваемой системе, они должны сами ***добывать*** информацию, необходимую им для успешной работы. Термин ***выявление требований*** точно отражает данный процесс, в котором разработчики должны играть активную роль. Чтобы помочь команде решить эти проблемы, лучше понять потребности пользователей и других заинтересованных лиц, целесообразно использовать ***методы***:

- интервьюирования и анкетирования – создание структурированного интервью; проведение интервью с пользователями и/или заинтересованными лицами; подведение итогов совокупности интервью, формулированиенаиболее часто упоминавшихся потребностей заказчика и пользователей;

- совещания, посвященные анализу и синтезу требований – формулирование и определение целей программного продукта; ознакомление с ними всех участников проекта и установление, что они с ними согласны; если это не так, следует остановиться и уточнениями добиться согласия; обязательно убедиться в согласии заказчиков;

- мозговой штурм и отбор идей, чтобы: выявить и/или уточнить функции проекта; отсечь нецелесообразные идеи; провести классификацию функций, чтобы определить приоритеты, риски, трудоемкости реализации фун-кций;

- анализ иллюстративных прецедентов в приложении к концепции требований (или системному проекту), чтобы их функции были наглядны и понятны;

- по возможности выявление или создание временных прототипов на основе первичных требований.

Хотя ни один из методов не является универсальным, каждый из них позволяет лучше понять потребности пользователей и тем самым превратить "неясные" требования, в требования, которые "лучше известны и понятны". Каждый из этих методов эффективен в определенных ситуациях, однако целесообразно отдавать предпочтение совещанию, посвященному требованиям, и мозговому штурму.

Для сложных систем требуются ***стратегии управления информацией о требованиях***. Для этого применяется информационная иерархия; она начинается с потребностей пользователей, описанных с помощью функций, которые затем превращаются в более подробные требования к ПС, выраженные посредством прецедентов или традиционных форм описания и стандартизированных характеристик. Эта иерархия отражает уровень абстракции при рассмотрении ***взаимосвязи области проблемы и области решения***. Концепцию требований, модифицированную в соответствии с конкретным содержанием комплекса программ, необходимо иметь в ***каждом*** проекте. В требованиях к ПС следует указывать, ***какие функции*** должны осуществляться, а ***не то, как*** они могут реализоваться. Они используются для задания функциональных и конструктивных требований, а также ограничений ресурсов проектирования. Нужно использовать набор характеристик для установления и оценки качества ПС и содержащихся в нем компонентов. Если необходимо, документация требований может дополняться одним или несколькими более формальными, либо более структурированными методами спецификации требуемого качества.

Без ***лидера – руководителя проекта***, который будет утверждать требования к ПС, согласовывать и поддерживать потребности заказчика и команды разработчиков, нельзя быть уверенным в том, что будут приняты необходимые четкие, скоординированные решения. Возможно возникновение флюктуаций требований, задержек, а также принятие неоптимальных решений, вызванное приближением срока окончания проекта. Руководитель должен создать совет по контролю за изменениями, который призван помогать лидеру в принятии действительно сложных решений и гарантировать, что изменения и утверждения требований будут приниматься только после их обсуждения.

Концепция требований к проекту (или системный проект) должна быть "***живым***" документом, чтобы было легче его использовать и пересматривать. Следует сделать концепцию ***официальным каналом*** изменения функций так, чтобы проект всегда имел достоверный, соответствующий современному состоянию документ. Лидер должен нести персональную ответственность за концепцию, регулярно (вместе с командой)

оценивать состояние дел и готовить отчеты и запросы, способствующие этим действиям. Процесс отслеживания спецификаций требований к ПС, должен помогать убедиться, что концепция надлежащим образом реализуется в детализированных требованиях к компонентам ПС. Руководство процессом контроля за изменениями, должно гарантировать, что перед тем, как разрешить внесение существенных изменений в систему, производится оценка рентабельности поступивших предложений (см. лекцию 16).

Проекты, как правило, инициируются с ***объемом функциональных возможностей, значительно превышающим*** тот, который разработчик может реализовать, обеспечив приемлемое качество при заданных ресурсах. Тем не менее, необходимо ограничиваться, чтобы иметь возможность предоставить в срок ***достаточно целостный и качественный продукт.*** Существуют различные методы задания очередности выполнения (приоритетов) требований и понятие базового уровня – совместно согласованного представления о том, в чем будут состоять ключевые функции системы как продукта проекта – понятие состава требований, задающих ***ориентир*** для принятия решений и их оценки.

Если ***масштаб проекта и сопутствующие требования заказчика превышают реальные ресурсы***, в любом случае придется ограничиваться в функциях и качестве ПС. Поэтому следует определять, что ***обязательно должно*** быть сделано в версии программного продукта при имеющихся ресурсах проекта. Для этого придется вести переговоры. Нельзя ожидать, что данный процесс полностью решит проблемы масштаба и требований к ПС. Но такие шаги окажут заметное воздействие на размеры проблемы, позволят разработчикам ПС сконцентрировать свои усилия на критически важных подмножествах требований и функций и, в несколько приемов, предоставить высококачественные системы, удовлетворяющие или даже превосходящие основные ожидания пользователей. Привлечение заказчика к решению проблемы управления масштабом и функциями повышает взаимные обязательства сторон, способствует росту взаимопонимания и доверия между заказчиком и разработчиками. Имея достаточное определение функций продукта (концепцию) и сократив масштаб проекта до разумного уровня, можно надеяться на успех в следующих фазах или версиях проекта.

На основе выполненных оценок трудозатрат рекомендуется определять базовый уровень для каждой версии концепции требований, используя атрибут "номер версии", достигнуть ***соглашения с заказчиком относительно масштаба,*** а также принять жесткие решения по масштабу версии. Итеративная разработка и управление изменениями, используя базовый уровень, позволяет контролировать, что все предложенные функции записаны и ничего не потеряно. Целесообразна ***система управления запросами на изменения требований***, чтобы фиксировать все изменения и гарантировать, что они поступают через эту систему в совет по контролю за изменениями. На всех этапах развития спецификации требований к ПС, следует задавать полный набор характеристик функционального и

конструктивного поведения программного продукта и подробные прецеденты для основных функций системы.

В требованиях должны быть полно и сжато зафиксированы потребности пользователей в таком виде, чтобы разработчик мог ***построить удовлетворяющее их ПС и его компоненты***. Кроме того, требования должны быть достаточно конкретными, чтобы можно было определить, когда они удовлетворены. Зачастую именно разработчики обеспечивают эту конкретизацию. Существуют различные возможности организации и документирования этих требований. Вся разработка должна вытекать из требований, а все спецификации ПС и компонентов должны найти отражение в процессах и продуктах разработки. Сложный программный комплекс следует пересматривать и обновлять на протяжении всего жизненного цикла проекта.

Проектирование и реализация корректной (правильной) системы, адекватной требованиям – весьма трудоемкая задача. Один из полезных методов состоит в использовании прототипов требований и прецедентов в качестве основы архитектуры и реализации проекта. Постоянно отслеживать эволюцию функций и требований, а также их реализацию позволяет ***верификация*** (см. лекцию 13). Её поддержка осуществляется путем использования методов ***трассировки***, что позволяет связывать друг с другом части проекта, проводить трассировку требований к прецедентам и функциям и обратно. С помощью трассировки можно удостовериться в том, что:

- все элементы требований проекта учтены;
- все реализованные элементы проекта служат заданной цели и требованиям.

Хотя верификация является аналитическим методом, рекомендуется помнить о важности размышлений и интуиции. Нельзя ограничиваться механическим применением верификации. Основное внимание при её проведении уделяется тестированию и использованию трассировки для отбора компонентов системы, нуждающихся в тестировании. ***Методы проверки корректности требований*** призваны гарантировать, что:

- все элементы требований могут надлежащим образом и достаточно полно тестироваться;
- все тесты служат цели проекта и не являются избыточными.

Чтобы принять решение о том, какая часть системы нуждается в верификации и проверке корректности и в каком объеме, целесообразно применять ***анализ и оценку рисков***. Это позволяет определить, для каких элементов неправильная реализация требований недопустима, а также разработать план действий по верификации и проверке правильности, основываясь на результатах этих оценок. На этом этапе следует привлекать к управлению требованиями и анализу их корректности, специалистов по тестированию, подключая их к планированию тестов с самого начала проекта (см. лекцию 13) . Группа тестирования должна разработать тестовые

процедуры и сценарии, которые трассируются к прецедентам, а также функциональным и конструктивным требованиям.

Построение ***корректных требований к системе*** во многом зависит от надлежащей обработки изменений. Изменения – неотъемлемая часть жизни ПС, это нужно учитывать ***при создании планов***, а также необходимо разработать процесс, с помощью которого можно управлять изменениями требований. Управление изменениями дает уверенность в том, что создаваемая система ***является корректной*** и, более того, ***будет правильной*** и в дальнейшем. Специалистам по независимой гарантии качества, должна отводиться роль в отслеживании и оценке процесса управления требованиями и плана их тестирования, а также периодических испытаний по окончании значительных этапов, чтобы проверить правильность результатов работы и обеспечить постоянное участие заказчиков.

Практическая часть

Процессы разработки требований к характеристикам сложных программных средств

При планировании, разработке и реализации ***требований к характеристикам качества ПС*** необходимо, в первую очередь, учитывать следующие основные факторы:

- функциональную пригодность (функциональность) конкретного проекта ПС;
- возможные конструктивные характеристики качества комплекса программ, необходимые для улучшения функциональной пригодности;
- доступные ресурсы для создания и обеспечения всего жизненного цикла ПС с требуемым качеством.

Эти факторы целесообразно применять последовательно, итерационно на каждом из основных этапов ЖЦ ПС. При первоначальном определении требований к функциональной пригодности и к конструктивным характеристикам качества, заданные заказчиком ограничения ресурсов не всегда могут учитывать ряд особенностей проекта, что обусловит недопустимое снижение (или завышение) требований к некоторым характеристикам качества ПС. Кроме того, возможно, что некоторые характеристики противоречивы или принципиально нереализуемы в данном проекте. В результате ***не сбалансированные требования*** к качеству и доступные ресурсы проявятся как риски – ущерб в виде потерь в качестве или в перерасходовании ресурсов. Для устранения или снижения рисков до допустимых пределов потребуется изменение требований к функциональной пригодности и/или к конструктивным характеристикам. Таким образом, целесообразно анализ и разработку требований к качеству ПС проводить в ***два этапа***: предварительно максимизируя функциональную пригодность и конструктивные характеристики качества, а затем минимизируя риски снижения требуемого качества или используемых ресурсов.

Разработку и утверждение требований к характеристикам и атрибутам качества без учета рисков, целесообразно проводить ***итерационно***

на этапах системного и детального проектирования ПС. Полная и однократная формализация требований к характеристикам качества в начале жизненного цикла сложного ПС обычно невозможна, прежде всего, из-за разных представлений заказчика и разработчиков о деталях его назначения, функций и возможностей реализации при доступных ресурсах. Чем крупнее и сложней проект ПС и соответственно выше его стоимость, тем тщательнее следует разрабатывать требования к его характеристикам качества и распределять ресурсы на их реализацию (см. лекцию 12). Поэтому при средней и относительно невысокой сложности ПС, во многих случаях, можно удовлетвориться подготовкой требований к качеству с подробностью анализа, соответствующей системному проектированию. Для крупномасштабных и особо сложных проектов необходим более детальный анализ факторов при разработке требований и их оптимизация по критерию качество/затраты. Поэтому ***на этапах проектирования последовательно должны определяться требования***:

- при проектировании концепции – предварительные требования к назначению, функциональной пригодности и к номенклатуре необходимых конструктивных характеристик качества ПС;
- при предварительном проектировании – требования к шкалам и мерам применяемых атрибутов характеристик качества с учетом общих ограничений ресурсов;
- при детальном проектировании – подробные требования к атрибутам качества с детальным учетом и распределением реальных ограниченных ресурсов, а также, возможно, их оптимизация по критерию качество/затраты.

В зависимости от сложности проекта окончательным результатом работ при предварительном или детальном проектировании должны быть детализированные и утвержденные требования к функциям, свойствам и значениям атрибутов качества ПС, которые в обоих случаях достаточны для его полноценного рабочего проектирования и последующей, эффективной эксплуатации (без учета рисков). В реальных проектах при подготовке требований к качеству ПС могут исключаться этапы проектирования, однако содержание и последовательность работ по анализу и детализации требований к характеристикам и атрибутам качества целесообразно сохранять. Эти ***требования закрепляются в контракте и техническом задании***, по которым разработчик впоследствии должен отчитываться перед заказчиком при завершении проекта или его версии. Однако на последующих этапах жизненного цикла и при конфигурационном управлении, требования могут изменяться по согласованию между заказчиком и разработчиком, которые чаще всего приурочиваются к подготовке новой версии программного продукта. Для этого необходим мониторинг требований и реализаций характеристик качества в течение всего ЖЦ ПС.

Выбор требований к характеристикам и атрибутам качества при проектировании программных средств начинается с ***определения исходных данных***. Для корректного выбора и установления требований к

характеристикам качества, прежде всего, необходимо ***определить особенности проекта***:

- класс, назначение и основные функции создаваемого ПС;
- комплект стандартов и их содержание, которые целесообразно использовать при выборе характеристик качества ПС;
- состав потребителей характеристик качества ПС, для которых важны соответствующие атрибуты качества;
- реальные ограничения всех видов ресурсов проекта.

Этап разработки концепции проекта целесообразно начинать с формализации и обоснования набора исходных данных, отражающих общие особенности класса, потребителей и этапов жизненного цикла проекта ПС, каждый из которых влияет на выбор определенных характеристик качества комплекса программ. Из конструктивных характеристик и атрибутов качества, прежде всего, следует выделять, те на которые в наибольшей степени воздействует ***класс, назначение и функции ПС***. Для этого первоначально целесообразно использовать классификацию программных средств по стандарту ISO **12182** и всю базовую номенклатуру характеристик и субхарактеристик качества, ***стандартизированных в*** **ISO 9126** (см. лекцию 11). Их описания желательно предварительно упорядочить по приоритетам с учетом особенностей назначения и сферы применения конкретного ПС. Обычно наиболее сильное влияние функции ПС оказывают на требования к атрибутам характеристик: Защищенность, Надежность и Эффективность. В то же время, характеристики Сопровождаемость и Мобильность относительно слабо связаны с назначением и конкретной функциональной пригодностью ПС для оперативных пользователей. Их меры и шкалы определяются не столько конкретными функциями комплекса программ, сколько его архитектурой и приспособленностью интерфейсов к модификации и переносу на иные операционные и аппаратные платформы.

Требования стандартов к функциональной пригодности должны выполняться для любых классов и назначения ПС. Однако номенклатура учитываемых требований к конструктивным характеристикам качества существенно зависят от назначения и функций комплексов программ. Так, например, при проектировании:

- систем управления объектами в реальном времени наибольшее значение имеет защищенность, корректность и надежность функционирования стабильного комплекса программ и менее важно может быть качество обеспечения сопровождения и конфигурационного управления, способность к взаимодействию и практичность;
- административных систем кроме корректности, важно обеспечивать практичность применения, комфортное взаимодействие с пользователями и внешней средой и может не иметь особого значение эффективность использования вычислительных ресурсов и обеспечение мобильности комплекса программ;

- операционных систем наиболее жесткие требования предъявляются к корректности, эффективности использования вычислительных ресурсов и защищенности, и не всегда могут учитываться мобильность и унификация способности к взаимодействию её компонентов;

- пакетов прикладных программ для вычислений или моделирования процессов, возможно не учитывать их мобильность, защищенность и временную эффективность, но особенно важна корректность.

Далее необходимо ***выделить и ранжировать по приоритетам потребителей***, которым необходимы определенные показатели качества ПС с учетом их специализации и профессиональных интересов. Широкая номенклатура характеристик, представленная в стандарте **ISO 9126** определяет разнообразные требования, из которых следует селектировать и выбирать те, которые необходимы ***с позиции потребителей этих данных***:

- заказчиков, для которых важно регламентировать и оценивать ПС по значениям требований к характеристикам, заданным и утвержденным в контракте, техническом задании и спецификациях требований и определяющих, прежде всего, назначение, функции и сферу применения программного продукта;

- пользователей, для которых необходима функциональная пригодность, корректность, надежность и другие показатели качества при оперативном использовании комплекса программ по основному назначению;

- разработчиков, для которых особенно важны: ясность и конкретность описаний требований к функциям и характеристикам ПС, его возможная архитектура и интерфейсы между компонентами и с внешней средой;

- специалистов сопровождающих и модифицирующих ПС, которые отдают приоритет характеристикам, поддерживающим сопровождение и конфигурационное управление версиями комплекса программ и его компонентов;

- лицам, ответственным за инсталляцию и реализацию программного продукта в различных аппаратных и операционных средах, для которых наиболее важны характеристики мобильности.

В таблице представлен ***пример ранжирования*** по степени важности на три уровня (высокая, средняя, низкая) основных стандартизированных характеристик качества ПС для разных категорий специалистов. Первые две группы потребителей характеристик качества, заинтересованы, преимущественно, в реализации функций, проявляющихся в процессе использования конечного, готового программного продукта. Для этих потребителей при выборе важно выделять и по возможности формализовать внешние, эксплуатационные характеристики на завершающих этапах ЖЦ версий ПС. К ним относятся, прежде всего, высокие приоритеты для надежности, эффективности и практичности. Для заказчиков приоритетными могут быть также сопровождаемость и мобильность, которые для оперативных пользователей ПС обычно являются второстепенными.

Последние три группы потребителей интересуют преимущественно характеристики ПС на промежуточных этапах ЖЦ, на которых проявляются, в основном, внутренние структурные, технологические свойства комплекса программ, влияющие также на сопровождаемость и мобильность. Их можно не представлять в составе эксплуатационной документации для оперативных пользователей и отражать только в технологической документации разработчиков, специалистов по сопровождению и переносу программ и данных, а также поставлять заказчикам по специальному запросу. Они должны формализоваться для обеспечения возможности контроля системой качества предприятия или проекта, а также для технологической поддержки реализации этих характеристик качества в течение всего ЖЦ ПС. Для этих потребителей надежность и практичность отходят на второй план, однако, ресурсная эффективность может оставаться высоко приоритетной.

Приоритеты потребителей при выборе требований к качеству отражаются не только на выделении важнейших для них критериев и ранжировании приоритетов других характеристик, но также на возможности исключении из анализа некоторых характеристик качества, которые для данного потребителя не имеют значения. Представленное в таблице 6.1 ранжирование может детализироваться и изменяться в зависимости от функций ПС и реальных ресурсов, доступных для обеспечения их жизненного цикла. Эти приоритеты должны обобщаться и учитываться заказчиком проекта при подготовке контракта и технического задания. После определения назначения и функций ПС подготовка исходных данных и концепции проекта должны завершаться ***выделением номенклатуры приоритетных конструктивных атрибутов характеристик качества***, имеющих достаточное влияние на функциональную пригодность ПС для определенных потребителей.

Состав концепции основных требований к программному средству:

- описание обобщенных результатов обследования и изучения существующей системы и внешней среды;
- описание целей, назначения программного продукта и потребностей заказчика и потенциальных пользователей к нему в заданной среде применения;
- перечень базовых стандартов предполагаемого проекта программного продукта;

Общие требования к характеристикам комплекса задач ПС:

- цели создания программного продукта и назначение комплекса функциональных задач;
- перечень объектов среды применения ПС (технологических объектов управления, подразделений предприятия и т. П.), при управлении которыми должен решаться комплекс задач;
- периодичность и продолжительность решения комплекса задач;
- связи и взаимодействие комплекса задач с внешней средой и другими компонентами системы;

– распределение функций между персоналом, программными и техническими средствами при различных ситуациях решения требуемого комплекса функциональных задач;

Требования к входной информации:

– источники информации и их идентификаторы;

– перечень и описание входных сообщений (идентификаторы, формы представления, регламент, сроки и частота поступления);

– перечень и описание структурных единиц информации входных сообщений или ссылка на документы, содержащие эти данные;

– - требования к выходной информации:

– потребители и назначение выходной информации;

– перечень и описание выходных сообщений;

– регламент и периодичность их выдачи;

– допустимое время задержки решения определенных задач;

- описание и оценка преимуществ и недостатков разработанных альтернативных вариантов функций в концепции создания проекта ПС;

- сопоставительный анализ требований заказчика и пользователей к программному продукту и набора функций в концепции ПС для удовлетворения требований заказчика и пользователей;

- обоснование выбора оптимального варианта требований к содержанию и приоритетам комплекса функций ПС в концепции;

- общие требования к структуре, составу компонентов и интерфейсам с внешней средой;

- ожидаемые результаты и возможная эффективность реализации выбранного варианта требований в концепции ПС;

- ориентировочный план реализации выбранного варианта требований концепции ПС;

- общие требования к составу и содержанию документации проекта ПС;

- оценка необходимых затрат ресурсов на разработку, ввод в действие и обеспечение функционирования ПС;

- предварительный состав требований, гарантирующих качество применения ПС;

- предварительные требования к условиям испытаний и приемки системы и ПС.

Спецификация требований к системе и к комплексу программ на этапе детального проектирования:

- требования проекта системы к комплексу программ, как к целому в общей архитектуре системы;

- требования к унификации интерфейсов и базы данных комплекса программ;

- требования и обоснование выбора проектных решений уровня системы, состава компонентов системы, описание функций системы и ПС с точки зрения пользователя;

- спецификация требований верхнего уровня комплекса программ, производные требования к компонентам ПС и требования к интерфейсам между системными компонентами, элементами конфигурации ПС и аппаратуры;

- описание распределения системных требований по компонентам ПС с учетом требований, которые обеспечивают заданные характеристики качества;

- требования к архитектуре системы, содержащей идентификацию и функции компонентов системы, их назначение, статус разработки, аппаратные и программные ресурсы;

- требования совместного целостного функционирования компонентов ПС, описание и характеристики их динамических связей;

- требования анализа трассируемости функций компонентов программного средства к требованиям проекта системы;

- требования для системы или/и подсистем и методы, которые должны быть использованы для гарантии того, что каждое требование к комплексу программ будет выполнено и прослеживаемо к конкретным требованиям системы:

* к режимам работы;

* к производительности системы;

* к внешнему и пользовательскому интерфейсу системы;

* к внутреннему интерфейсу компонентов и к внутренним данным системы;

* по возможности адаптации ПС к внешней среде;

* по обеспечению безопасности системы, ПС и внешней среды;

* по обеспечению защиты, безопасности и секретности данных;

* по ограничениям доступных ресурсов проекта ПС;

* по обучению и уровню квалификации персонала;

* по возможностям средств аттестации результатов и компонентов, включающие в себя демонстрацию, тестирование, анализ, инспекцию и требуемые специальные методы для контроля функций и качества конкретной системы или компонента ПС.

Задание.

1. Ознакомиться с теоретическими сведениями по практической работе

2. Определить достоинства и недостатки моделей ЖЦ

3. Выбрать и обосновать выбор модели ЖЦ для выполнения индивидуального проектного задания.

4. Сформировать план построения ИС индивидуального проектного задания, с использованием программных средств.

Контрольные вопросы:

1. Что отражает модель ЖЦ ИС?

2. Организационные процессы внедрения ИС;

3. События, происходящие с системой в процессе ее создания и использования;

Практическая работа №3

Тема: Реализовать проектирования программного обеспечения

Цель работы: Ознакомиться и изучить вопросы проектирования программного обеспечения.

Теоритическая часть

Технический проект – образ намеченного к созданию объекта, представленный в виде его описания, схем, чертежей, расчетов, обоснований, числовых показателей.

Технический проект

Цель технического проекта – определение основных методов, используемых при создании информационной системы, и окончательное определение ее сметной стоимости.

Техническое проектирование подсистем осуществляется в соответствии с утвержденным техническим заданием.

Технический проект программной системы подробно описывает:

- выполняемые функции и варианты их использования;
- соответствующие им документы;
- структуры обрабатываемых баз данных;
- взаимосвязи данных;
- алгоритмы их обработки.

Технический проект должен включать данные об объемах и интенсивности потоков обрабатываемой информации, количестве пользователей программной системы, характеристиках оборудования и программного обеспечения, взаимодействующего с проектируемым программным продуктом.

При разработке технического проекта оформляются:

- ведомость технического проекта. Общая информация по проекту;
- пояснительная записка к техническому проекту. Вводная информация, позволяющая ее потребителю быстро освоить данные по конкретному проекту;
- описание систем классификации и кодирования;
- перечень входных данных (документов). Перечень информации, которая используется как входящий поток и служит источником накопления;
- перечень выходных данных (документов). Перечень информации, которая используется для анализа накопленных данных;
- описание используемого программного обеспечения. Перечень программного обеспечения и СУБД, которые планируется использовать для создания информационной системы;
- описание используемых технических средств. Перечень аппаратных средств, на которых планируется работа проектируемого программного продукта;
- проектная оценка надежности системы. Экспертная оценка надежности с выявлением наиболее благополучных участков программной системы и ее узких мест;

– ведомость оборудования и материалов. Перечень оборудования и материалов, которые потребуются в ходе реализации проекта.

Структурная схема

Структурной называют схему, отражающую состав и взаимодействие по управлению частями разрабатываемого программного обеспечения. Структурная схема определяется архитектурой разрабатываемого ПО.

Функциональная схема

Функциональная схема – это схема взаимодействия компонентов программного обеспечения с описанием информационных потоков, состава данных в потоках и указанием используемых файлов и устройств.

Разработка алгоритмов

Метод пошаговой детализации реализует нисходящий подход к программированию и предполагает пошаговую разработку алгоритма.

Структурные карты

Методика структурных карт используется на этапе проектирования ПО для того, чтобы продемонстрировать, каким образом программный продукт выполняет системные требования. Структурные карты Константайна предназначены для описания отношений между модулями.

Техника структурных карт Джексона основана на методе структурного программирования Джексона, который выявляет соответствие между структурой потоков данных и структурой программы. Основное внимание в методе сконцентрировано на соответствии входных и выходных потоков данных.

Основной причиной большинства провалов программных проектов является именно применение неадекватных методов управления его разработкой. Главными причинами провалов программных проектов являются:

1. Отсутствие требований заказчика или их неполнота, или их частые изменения.
2. Отсутствие рабочего взаимодействия с заказчиком.
3. Отсутствие опыта у разработчика.
4. Отсутствие необходимых ресурсов людских, материальных, временных.
5. Неполнота планирования, «забытые работы».
6. Ошибки в оценках трудоемкости и сроков работ.
7. Использование слишком новых, неопробованных и нестабильных технологий разработки.

На ***этапе предварительного проектирования***, после фиксирования исходных данных и приоритетов характеристик для конкретного проекта и его потребителей, может начинаться выбор требований к свойствам и значениям, а также установление и утверждение конкретных мер и шкал характеристик и атрибутов качества. Такой анализ должны проводить заказчик и некоторые потенциальные пользователи совместно со специалистами, обеспечивающими ЖЦ комплекса программ и реализацию установленных требований к

показателям качества. Этими специалистами, для каждого из выбранных атрибутов, должна быть установлена и согласована мера и шкала оценок характеристик и их атрибутов для конкретного проекта и потребителя результатов анализа. Там где кроме оценивания наличия номинальных свойств, возможны количественные измерения, при оценке реализации требований к качеству может быть полезен выбор и установление допусков на отклонения от величин требуемых спецификациями. Для показателей, представляемых качественными свойствами и признаками их наличия, желательно определить и зафиксировать в спецификациях описания допустимых условий, при которых следует считать, что данная характеристика может или должна быть реализована в проекте ПС.

Принципиальные и технические возможности, точность реализации свойств и измерения значений характеристик качества, а также ***общие ресурсы конкретного проекта***, всегда ограничены в соответствии с их содержанием и возможностями заказчика и разработчиков. Это определяет рациональные диапазоны значений каждого атрибута, которые могут быть выбраны для проекта ПС ***на основе требований заказчика***, здравого смысла, а также путем анализа пилотных проектов и прецедентов в спецификациях требований реализованных проектов. В пределах этих диапазонов целесообразно определение реальной достоверности оценивания и масштаба мер для описания соответствующего атрибута требований.

Требования к значениям характеристик качества должны быть предварительно проверены разработчиками на их реализуемость с учетом доступных ресурсов конкретного проекта и при необходимости откорректированы по составу и значениям с учетом рисков. При ограниченности ресурсов проекта ПС распределение приоритетов должно становиться более строгим и могут снижаться приоритеты характеристик, для реализации которых ресурсов недостаточно. В результате формируется полный ***набор требуемых характеристик, свойств и атрибутов, их мер и значений качества для определенных потребителей в ЖЦ ПС***.

Входные проектные данные и требования к ПС, включая установленные законодательные и регламентирующие нормативные требования, должны быть, оформлены документально, а их выбор проанализирован поставщиком на адекватность. Неполные, двусмысленные или противоречивые требования должны быть предметом урегулирования с лицами, ответственными за их предъявление. Спецификацию требований должен представить потребитель-заказчик. Однако по взаимному согласию её может подготовить поставщик-разработчик, в тесном сотрудничестве с потребителем для предупреждений разногласий путем, например, уточнения определений терминов, объяснения предпосылок и обоснования требований. Исходные требования могут быть представлены и согласованы в виде спецификации всей системы. Если программный продукт нуждается во взаимодействии с другими программными или аппаратными продуктами, то в спецификации требований должны быть оговорены непосредственно или при

помощи ссылок интерфейсы между разрабатываемыми и другими применяемыми продуктами. В этом случае должны быть разработаны процедуры, обеспечивающие четкое распределение системных требований между аппаратными и программными компонентами, а также соответствующие спецификации интерфейсов с внешней средой. При заключении контракта спецификация требований может быть определена не полностью, она может быть доработана в ходе реализации проекта.

Выходные проектные данные должны быть документально оформлены и выражены в требованиях так, чтобы их можно было проверять в ЖЦ ПС и подтвердить соответствие входным проектным требованиям. Выходные проектные данные должны содержать критерии приемки проекта заказчиком или ссылки на них, а также идентифицировать те характеристики проекта, которые являются критическими для безопасного и надежного функционирования ПС. К составу выходных проектных данных могут относиться:

- спецификация структурного проектирования;
- описание результатов и спецификации рабочего проектирования компонентов;
- исходные тексты программ и программы в объектном коде;
- комплект эксплуатационной документации и руководств для пользователей;
- комплект технологической документации для обеспечения возможности модификации и сопровождения ПС.

Результаты системного анализа и выбора номенклатуры и мер характеристик проектов ПС средней или относительно невысокой сложности должны быть документированы в спецификациях требований, согласованы с их потребителями и утверждены заказчиком проекта для последующей реализации. Для разработчиков особенно важно ***формализовать требования к качеству и согласовать их с заказчиком при утверждении контракта и технического задания на проект ПС***. Требования к характеристикам, утвержденные после предварительного проектирования, могут быть закреплены в техническом задании ***как обязательные для детального и рабочего проектирования***. Эти данные могут использоваться при последующем оценивании качества и его сопоставлении с требованиями в процессе квалификационных испытаний или сертификации ПС. Однако для крупных и дорогих проектов может потребоваться уточнение требований к качеству при детальном проектировании с позиции улучшения соотношения значений качества и затрат ресурсов, которые необходимы или допустимы для их реализации в ЖЦ ПС.

При детальном проектировании может быть целесообразно дополнительное уточнение совокупности требований выбранных характеристик и атрибутов качества сложных ПС с учетом соотношения качества и затрат ресурсов, которые могут быть весьма велики. Для заказчика и пользователей может иметь значение не только определение

функциональной пригодности, но и потенциального спроса на рынке конкретного программного продукта, а также его конкурентоспособности с другими аналогичными по функциям ПС с учетом его качества и стоимости. Это обстоятельство может определять необходимость уточнения требований к отдельным характеристикам качества не только для их реализации разработчиками в ЖЦ ПС, но также для оценивания интегрального качества готового программного продукта поставляемого на рынок.

Каждая характеристика качества и затраты ресурсов первоначально ***анализируются независимо***, что может использоваться в качестве исходных данных для их сопоставления с отдельными характеристиками аналогичных ПС, или для представления, как составляющей вектора в многомерном пространстве стандартизированных атрибутов характеристик качества. Обычно заказчики и разработчики первоначально устанавливают требования к каждой характеристике без учета относительных затрат на их достижение, а также без детального анализа их совместного влияния на полную функциональную пригодность у потребителей. Это может приводить к значительным перекосам и ***несбалансированным значениям требований*** к отдельным, взаимосвязанным характеристикам качества, на которые не рационально используются ограниченные ресурсы ЖЦ ПС, или к не адекватно низким их значениям. В проектах крупных ПС это может угрожать значительным повышением стоимости и/или снижением конкурентоспособности создаваемого программного продукта из-за недостаточного уровня отдельных показателей качества.

Атрибуты качества ПС имеют различные меры и шкалы, вследствие чего они в большинстве своем непосредственно ***не сопоставимы между собой***. Они предварительно выбираются и согласовываются с заказчиком при последовательном, почти независимом анализе каждого атрибута качества в соответствии с их мерами и шкалами, для последующего использования в контракте и техническом задании. Для обобщенного оценивания качества ПС необходим учет относительного влияния каждого атрибута на функциональную пригодность. При этом не всегда учитываются ресурсы для их реализации в конкретном ПС. Это часто приводит к выдвижению ряда не рациональных требований, которые значительно отличаются: либо по степени влияния на функциональную пригодность, либо по величине ресурсов, необходимых для их реализации. Для целенаправленного эффективного управления качеством сложного ПС при проектировании целесообразно иметь механизм объединения разнородных характеристик в некоторый интегральный показатель, отражающий их совокупное влияние на его функциональную пригодность. Таким образом, при разработке требований к характеристикам качества выявилась проблема анализа системной эффективности программного средства и обобщения его характеристик, а также ***оценивания совместного влияния различных характеристик и атрибутов качества на функциональную пригодность ПС с учетом затрат*** на их реализацию

Задание

1.На основе технического задания разработать

- Структурную схему программного продукта.
- Функциональную схему.
- Алгоритмы программы.
- Дерево диалога.

2.Разработать функциональную схему программного продукта.

3. Разработать интерфейс системы в виде дерева диалога.

5. Оформить результаты, используя MS Office в виде технического проекта.

6. Сдать и защитить работу.

Контрольные вопросы

1.Назовите этапы разработки программного обеспечения.

2.В чем заключается проектирование программного обеспечения?

3.Перечислите составляющие технического проекта.

4.Охарактеризуйте структурный подход к программированию.

5.Из чего состоят структурная и функциональная схемы?

6.Приведите понятие псевдокода.

Практическая работа №4

Тема: Реализовать угрозу SQL-инъекций в симуляторе OWASP WebGoat.

Цель работы: ознакомиться и изучить угрозу SQL-инъекций. Установка и работа на симуляторе OWASP WebGoat.

Теоритическая часть

Веб-технологии используются практически повсеместно – интернет-магазины, банки, да и просто веб-страницы какой-либо фирмы. Они могут иметь как сложные интерфейсы, так и простенькие с одним двумя полями ввода, для быстрого доступа к информации хранящейся в базе данных. Специалистов привлекают доступность с любой точки планеты, быстрая разработка и независимость от клиентской платформы. Для управления достаточно веб-браузера который давно уже стал стандартным компонентом любой ОС, включая используемых в карманных компьютерах и смартфонах.

К сожалению, доступность веб-приложения отовсюду, имеет и свой минус, приходится уделять дополнительное внимание безопасности решения. А это не простой вопрос. Ведь в формировании результирующего контента участвует несколько элементов – веб-сервер или сервер-приложений, обработчик (HTML, PHP, Python, Perl, Ruby, JavaScript, + AJAX и т.п.), база данных. И это только небольшой список технологий, которые должны работать как единое целое, выдавая результат. Причем конечных решений может быть много. Ведь список тех же веб-серверов насчитывает уже десяток вариантов. Естественно разработчик предусмотреть все возможные проблемы для разных ситуаций просто не в состоянии. В результате получаем целый букет специфический атак направленных на веб-приложения — **XSS (Cross-Site Scripting), SQL injection**, подмена сессии, подмена прототипа. Все в различных вариантах исполнения. Причем это не болезнь маленьких проектов, уязвимостям подвержены и крупные проекты, развивающиеся уже не один год.

Специалисты безопасности и разработчики должны смотреть на сеть глазами потенциального взломщика, понимать суть атак и видеть проблемные места. Разнообразные руководства, которые легко найти в интернет могут дать лишь теоретические знания. Без их практического закрепления они не очевидны, легко забываются и будут очень поверхностными. Вот для такой ситуации разработчиками OWASP (Open Web Application Security Project) создана специальная обучающая система WebGoat [1], позволяющая в наглядном виде изучать приемы взлома веб-приложений. Реализована база для проведения около 30 различных видов атак. Причем основной упор сделан именно на образовательную сторону вопроса, а не создание уязвимой платформы для опытов. В WebGoat реализованы все сопутствующие элементы – лекции, проверки знаний, лабораторная работа и результирующий экзамен. По ходу обучения ведется статистика, показывающая результат на каждом этапе. Список курсов обширен и затрагивает базовые знания по HTML, контроль доступа, и различные виды атак – XSS, различные виды Injections, Buffer Overflow, работа со CSS и скрытыми полями в формах и так далее. По

ходу обучения объясняется суть проблемы, даются все необходимые подсказки и код, этап завершается практической демонстрацией взлома с использованием уязвимости. Пройденная лекция подсвечивается зеленым флажком. Удобно, что в WebGoat уже есть все необходимое, то есть не нужно самостоятельно собирать тестовую среду, чтобы проверить все на практике.

Архитектура WebGoat проста. В базе данных содержатся все лекции. Страница выдаваемая пользователю формируется за счет нескольких сервлетов JSP (JavaServer Page), каждый из которых играет свою роль – собственно страница с заголовком и навигацией (main.jsp), список лекций (CrossSiteScripting.jsp, RoleBasedAccessControl.jsp, SQLInjection.jsp), и контент (EditProfile.jsp и другие).

К слову OWASP — некоммерческая организация, направленная на улучшение безопасности прикладных приложений, все продукты которой выпускаются под свободными лицензиями. Одна из ее разработок — брандмауэр уровня приложений **ModSecurity** защищающий веб-сервер и работающие на нем приложения от специфических атак. Поэтому кроме обучения у WebGoat еще одно предназначение – наглядно представить, как работает ModSecurity, помочь при создании и тестировании правил. Собственно этим занимается отдельный субпроект OWASP — **OWASP Securing WebGoat using ModSecurity Project**, задача которого разработка новых правил для ModSecurity позволяющих полностью защитить уязвимости представленные в WebGoat, без изменения единой строки кода в последнем. То есть в идеале, ModSecurity должен блокировать 100% атак направленных на "дырявый" WebGoat. Для этого ModSecurity настраивается в качестве прокси-сервера. Возможно, в будущем WebGoat будет также представлен и в качестве приманки для взломщика (honeypot).

Интересно, что WebGoat реализовали на Java, который очень не любим специалистами безопасности. Поэтому в дополнение к "своим" уязвимостям он может содержать еще проблемы характерные для Java приложений. Кстати название WebGoat созвучно со scapegoat (англ. Козёл отпущения), который и является эмблемой проекта.

Практическая часть

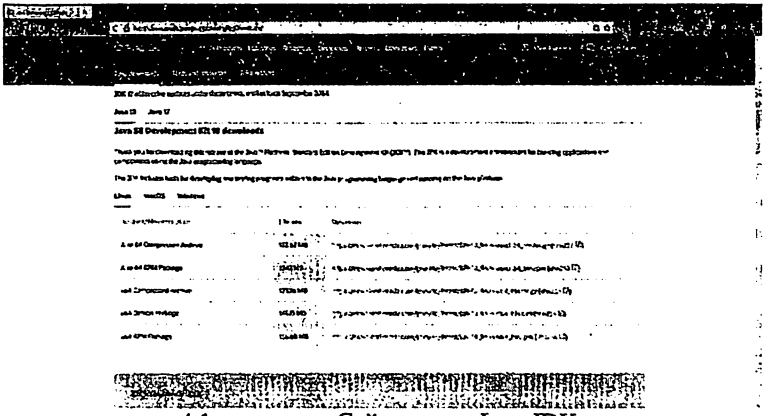
Для установки JDK на компьютер необходимо:

- Загрузить пакет JDK (Java Development Kit)
- Установить JDK (Java Development Kit)
- Добавить системную переменную JAVA_HOME

На странице загрузки выберите нужную версию Java (**для Android Studio требуется версия**) и нажмите кнопку загрузки **JDK:**

Установка пакета JDK (Java Development Kit)

Для установки Java необходимо иметь права администратора (перезагрузить компьютер, если необходимо войти с другой учетной записью).



4.1 - рисунок. Сайт скачки Java JDK.

Установка WebGoat

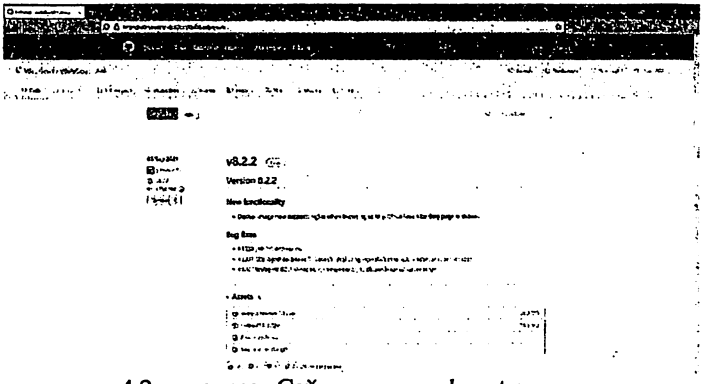
WebGoat — кроссплатформенный инструмент, его можно запустить в любой ОС, в которой будут работать Apache Tomcat и Java SDK. Для доступа к лекциям используется веб-браузер, интерфейс только английский и очевидно другого пока не предвидится.

На сайте проекта доступны инструкции по установке в Linux, Windows и Mac OS X, но они весьма поверхностны и не показывают сути. Чтобы не устанавливать WebGoat.

Как видим, в описании нас отправляют на GitHub, чтобы скачать последний релиз WebGoat.

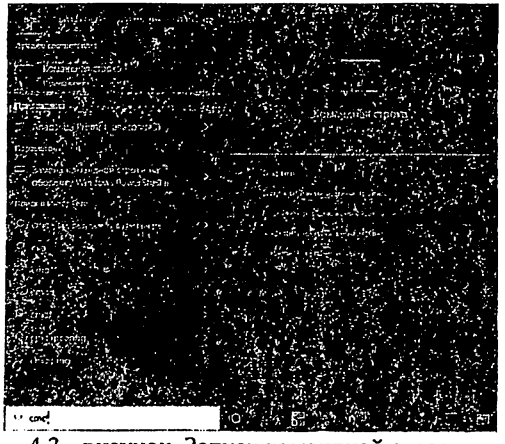
Идем по ссылке: https://github.com/WebGoat/WebGoat/releases

Выбираем самый свежий релиз и скачиваем файл webgoat-server-jar:



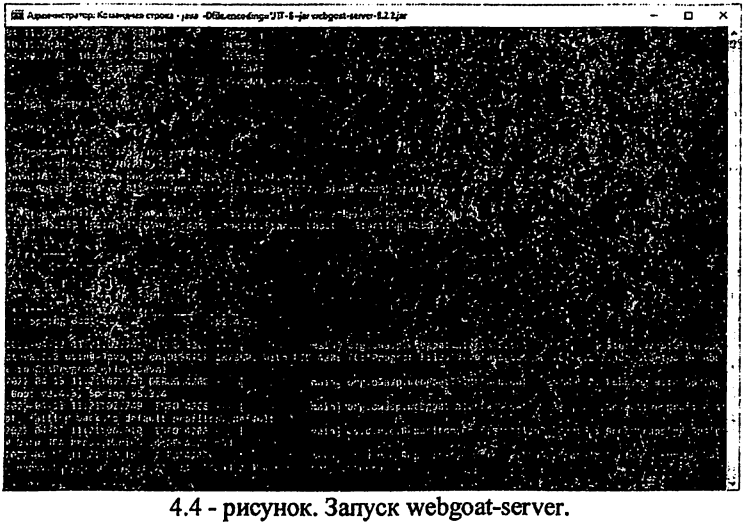
4.2 - рисунок. Сайт скачки webgoat-server.

Далее идем в папку Downloads и копируем данный архив в более приемлемое место, но можно запускать и из этой директории, кому как удобно.



4.3 - рисунок. Запуск командной строки.

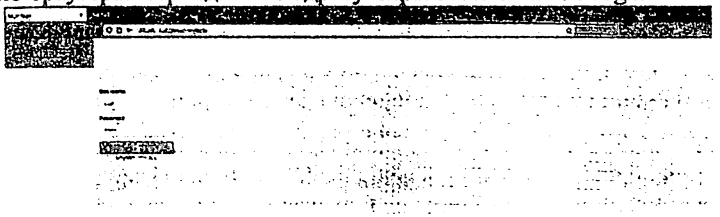
Теперь нам нужно запустить этот файл (webgoat-server и webwolf), с помощью команды: «java -jar», предварительно перейдя в директорию месторасположения:

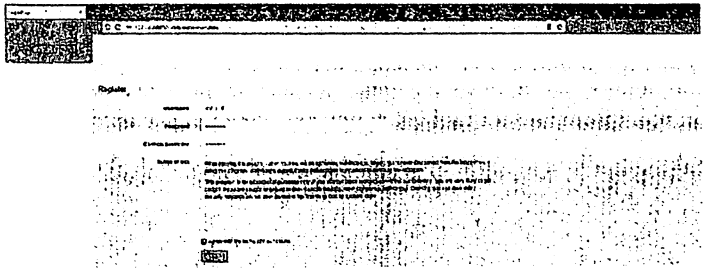


4.4 - рисунок. Запуск webgoat-server.

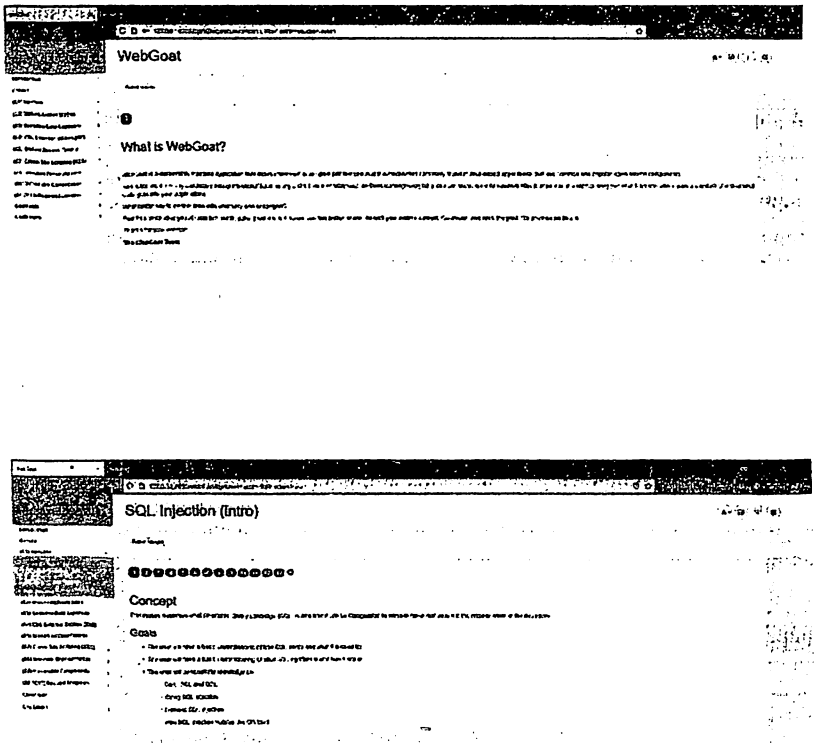
4.5 - рисунок. Запуск webwolf.

После успешного завершения зайдите на http://127.0.0.1:8080/WebGoat/ с помощью браузера. Перейдите по адресу http://127.0.0.1:9090/login.





4.6 - рисунок. Окна регистрации учетной записи.



4.7 - рисунок. Окна работы с инструментариями.

Задание

1. Ознакомится с контентом, представленными в симуляторе webgoat-server.

2. Реализовать запрос SQL-инъекции любого типа с использованием симулятора webgoat-сервера.

Контрольные вопросы

1. Что такое SQL-инъекция и ее виды.
2. Как защитить себя от этой угрозы.
3. Являются ли эти типы угроз серьезными.

Практическая работа №5

Тема: Реализация и защита угрозы TOCTOU.

Цель работы: изучить время проверки до времени использования.

Теоритическая часть

В разработке программного обеспечения время проверки до момента использования (TOCTTOU или TOCTOU, произносится "TOCK too") - это класс ошибок программного обеспечения, вызванных изменениями в системе между проверкой условия (например, учетных данных безопасности) и использованием результатов этой проверки. Это один из примеров состояния гонки.

В TOCTOU говорится, что состояние гонки может возникнуть, если состояние системы изменяется между моментом, когда некоторое условие было проверено процессом, и моментом, когда действие было предпринято на основе этого условия тем же процессом.

Атака TOCTTOU, использующая такие условия, может привести к повышению привилегий, позволяя получить несанкционированный доступ к ресурсам, например, доступ на чтение и запись, а также обойти контроль журналов и аудита. Такого рода атаки трудно обнаружить. Она требует не только поиска доказательств, но и определения того, может ли она быть вызвана TOCTTOU.

Условия гонки TOCTTOU наиболее распространены в файловых системах Unix, но уязвимы все системы. Например, в Java можно проверить существование файла и возможность доступа к нему программы с помощью метода checkAccess(), но нет никакой гарантии, что к файлу можно получить доступ после завершения проверки. Большинство атак требуют точного времени для выполнения системных вызовов, чтобы они правильно чередовались с жертвой и изменяли установленное состояние файла в промежутке между этапами проверки и работы с файлом. Однако существуют и более простые формы TOCTTOU, например, веб-администратор блокирует страницу для предотвращения редактирования после того, как пользователь уже начал ее редактировать. Правки будут приняты, если приложение не перепроверит состояние страницы.

Первопричина многих уязвимостей TOCTTOU кроется в отсутствии контроля параллелизма в API файловой системы операционной системы, поэтому эту проблему не так просто решить. Большинство ОС изменяют порядок выполнения инструкций и процессов для повышения эффективности. Программисту приходится добиваться атомарности двух операций, используя API, который для этого не предназначен. Таким образом, проблема заключается в том, чтобы состояние файловой системы, управляемое операционной системой, не могло измениться между двумя системными вызовами.

В этой работе вы напишете Java-приложение, используя API с ошибками. После эксплуатации этого приложения, вы предложите исправление.

Практическая часть

Часть 0. **Сценарий и базовая программа**

Вам нужно написать фронтенд для корзины покупок с командной строкой. Приложение

приложение (называется ShoppingCart.java) должно:

1. выводить текущий баланс пользователя
2. выводить список товаров и их цены
3. запрашивать товар для покупки
4. проверить, достаточно ли кредитов, если нет, остановить выполнение.
5. в противном случае вывести цену товара из кошелька.
6. добавить название товара в файл кошелька.
7. вывести новый баланс.
8. нормально выйти.

Вот пример:

$ java ShoppingCart
Ваш баланс: 30000 кредитов
ручка 40
автомобиль 30000
конфеты 1
книга 100
Что вы хотите купить? <вставьте название товара, например, ручку>.
Ваш новый баланс: 29960 кредитов

$ cat pocket.txt
книга
ручка

$ cat wallet.txt
29960

Мы предоставляем бэкенд здесь (API документирован в исходном коде), который включает в себя:

- Wallet.java: Взаимодействует с кошельком.
- Pocket.java: Взаимодействует с карманом.
- Store.java: Взаимодействует с опциями и ценами продукта

Часть 1: Эксплуатация программы

Цель состоит в том, чтобы атаковать программу и использовать состояние гонки в этой системе. Можете ли вы приобрести автомобиль, заплатив меньше его стоимости? Очевидно, что нельзя разрешается изменять файлы pocket.txt и wallet.txt иначе, чем через API.

- Что такое общий ресурс? Кто им пользуется?
- В чем корень проблемы?
- Объясните в деталях, как можно атаковать эту систему.

– Представьте вывод и результат программы, объяснив, как чередование позволяет достижения этих результатов.

Часть 2: Исправление API

Цель - исправить API, чтобы избежать уязвимости.

– Напишите метод Wallet, реализующий необходимые средства защиты:

– public void safeWithdraw(int valueToWithdraw) throws Exception

– Были ли другие API, страдающие от возможных гонок? Если да, пожалуйста объясните их и обновите API, чтобы устранить любые проблемы, связанные с гонками.

– Почему этих мер защиты достаточно?

– Убедитесь, что ваше решение работает вне любой IDE, которую вы используете.

Задание

1. Ваш файл ShoppingCart.java с четкими инструкциями по его компиляции и запустить его.
2. Выходные журналы и объяснения, как их получить
3. Код, обеспечивающий безопасность API с ошибками.

Контрольные вопросы

1. Что такое состояние гонки?
2. Какие типы независимых потоков существует?
3. Объясните состояние гонки на примерах?

Практическая работа №6

Тема: Моделирование угроз с помощью Microsoft Threat Modeling Tools.

Цель работы: установить и изучить средство моделирования угроз.

Теоритическая часть

Средство моделирования угроз — это основной элемент жизненного цикла разработки защищенных приложений (Майкрософт) (SDL) Он позволяет архитекторам ПО на ранних этапах выявлять и устранять потенциальные проблемы, связанные с безопасностью, когда их еще можно относительно легко и без особых затрат устранить. В результате это средство значительно снижает стоимость разработки. Кроме того, при проектировании этого средства учитывались потребности специалистов, не связанных со сферой безопасности. С его помощью все разработчики могут легко моделировать угрозы благодаря четкому руководству по созданию и анализу моделей угроз.

Это средство предоставляет пользователям следующие возможности.

– Обмен данными о разработке решений для обеспечения безопасности своих систем.

– Анализ этих решений на предмет потенциальных проблем безопасности с помощью проверенных методов.

– Рекомендации по анализу проблем, связанных с безопасностью, и методы их устранения

– Ниже описаны некоторые из возможностей (включая инновационные) предлагаемого инструментария.

– **Автоматизация** — рекомендации и отзывы о создании модели.

– **STRIDE по элементам** — интерактивный анализ угроз и их устранение.

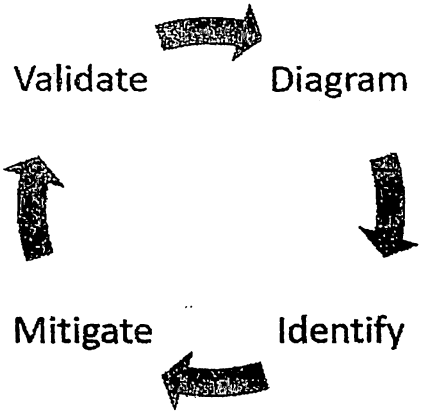
– **Отчеты** — действия по обеспечению безопасности и тестирование на этапе проверки.

– **Уникальные методы** — пользователи могут более эффективно визуализировать и оценивать угрозы.

– **Спроектированное для разработчиков с упором на защиту программного обеспечения** — разные подходы по работе с активами или злоумышленниками. Мы всегда ставим во главу угла программное обеспечение. Мы опираемся на действия, знакомые всем разработчикам и архитекторам: рисование изображений при проектировании архитектуры программного обеспечения.

– **Фокус на анализе архитектуры** — термин "моделирование угроз" может означать как требования, так и методику анализа архитектуры. В некоторых случаях он обозначает сложную комбинацию этих двух компонентов. Подход к моделированию угроз Microsoft SDL — это метод с фокусом на анализ архитектуры.

Ниже приведена диаграмма, на которой показан этот процесс.

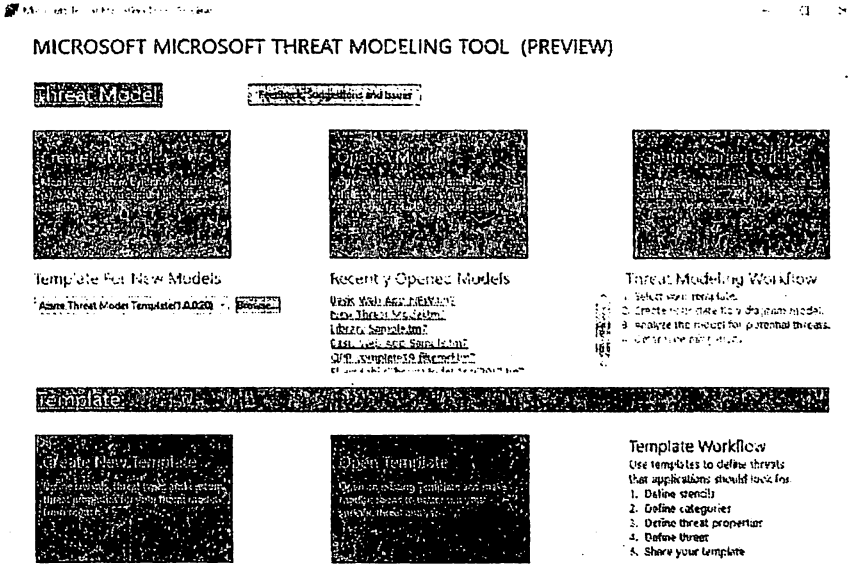


6.1 - рисунок. Диаграмма процесса.

Практическая часть

Запуск процесса моделирования угроз

При запуске средства моделирования угроз вы увидите несколько вещей, как показано на рисунке:



6.2 - рисунок. Средства моделирования угроз.

Раздел Threat model (Модель рисков)

Компонент	Сведения
Кнопка Feedback, Suggestions and	Вы перейдете на форум MSDN, где обсуждается все, что касается SDL. Таким образом вы получите

Issues (Отзывы, предложения и проблемы)	возможность прочитать о том, что в таких случаях делают другие пользователи, а также узнать способы решения проблем и рекомендации. Если вам по-прежнему не удается найти то, что ищете, отправьте письмо по адресу tmtextsupport@microsoft.com, чтобы получить помощь от нашей группы поддержки.
Создание модели	Откроется пустой холст для рисования диаграммы. Выберите шаблон, который необходимо использовать для модели.
Шаблон для новых моделей	Перед созданием модели необходимо выбрать шаблон, который будет использоваться. Наш основной шаблон — шаблон моделирования угроз Azure, который содержит наборы элементов, угрозы и способы их устранения, относящиеся только к Azure. Для универсальных моделей выберите базу знаний TM SDL в раскрывающемся меню. Хотите создать собственный шаблон или отправить его всем пользователям? Дополнительные сведения см. в нашем **репозитории шаблонов** на странице GitHub.
Открытие модели	Открываются ранее сохраненные модели рисков. Функция Recently Opened Models (Недавно открытые модели) отлично подходит, если необходимо открыть самые последние файлы. При наведении указателя мыши на выделенные модели вы увидите два способа открытия: Open From this Computer (Открыть на этом компьютере) — классический способ открытия файла с помощью локального хранилища; Open from OneDrive (Открыть в OneDrive) — команды могут использовать папки в OneDrive, чтобы сохранить и совместно использовать свои модели рисков в одном расположении с целью повышения производительности и улучшения совместной работы.
Руководство по началу работы	Откроется основная страница **средства моделирования угроз Microsoft.**

Раздел Template (Шаблон)

Компонент	Сведения
Create New Template (Создать шаблон)	Откроется пустой шаблон для создания модели. При отсутствии обширных знаний о создании шаблонов с нуля мы рекомендуем создавать модели на основе уже имеющихся.

Open Template (Открыть шаблон)	Откроются имеющиеся шаблоны, в которые нужно внести изменения.

Модель STRIDE

Чтобы помочь вам точнее формулировать эти вопросы, корпорация Майкрософт предлагает модель STRIDE, которая классифицирует разные типы угроз и упрощает обмен информацией, связанной с общей безопасностью.

Категория	Описание
Спуфинг	Предполагает незаконный доступ к данным пользователя (включая имя пользователя и пароль), используемыми для аутентификации, и их последующего применения.
Незаконное изменение	Предполагает вредоносное изменение данных. Примеры включают несанкционированные изменения, внесенные в постоянные данные, например хранящиеся в базе данных, а также изменение данных при их передаче между двумя компьютерами через открытую сеть, например Интернет.
Отказ	Речь идет о пользователях, которые отрицают выполнение действий, если другие пользователи не могут доказать обратное. Например, пользователь может выполнить незаконную операцию в системе, где отсутствует возможность трассировки запрещенных операций. Неподдельность означает способность системы учитывать угрозы отказа. Например, пользователь, который покупает товар, должен расписаться на квитанции при получении товара. Продавец может использовать эту квитанцию как доказательство того, что покупатель уже получил товар.
Раскрытие информации	Предполагает раскрытие сведений пользователям, которые не должны иметь к ним доступ, например возможность прочитать файл, к которому этим пользователям не предоставлен доступ, или возможность для злоумышленника считать данные при их передаче между двумя компьютерами.
Отказ в обслуживании	DoS-атака — это буквально отказ в обслуживании для допустимых пользователей, что делает веб-сервер временно недоступным или непригодным для использования. Следует защищаться от

	определенных типов угроз DoS-атак, просто чтобы повысить доступность и надежность системы.
Несанкционированное получение привилегий	Непривилегированный пользователь получает привилегированный доступ, т. е. достаточные полномочия для нарушения работоспособности или уничтожения всей системы. Угроза повышения привилегий включает ситуации, в которых злоумышленник эффективно обходит все средства защиты системы, сам становясь частью надежной системы. Тем самым он создает действительно опасную ситуацию.

Команда разработчиков средства моделирования угроз постоянно работает над тем, чтобы улучшить функциональность и возможности средства. В течение года могут произойти некоторые незначительные изменения, но при всех основных изменениях необходимо переписывать все руководство. Почаще просматривайте его, чтобы быть в курсе последних объявлений.

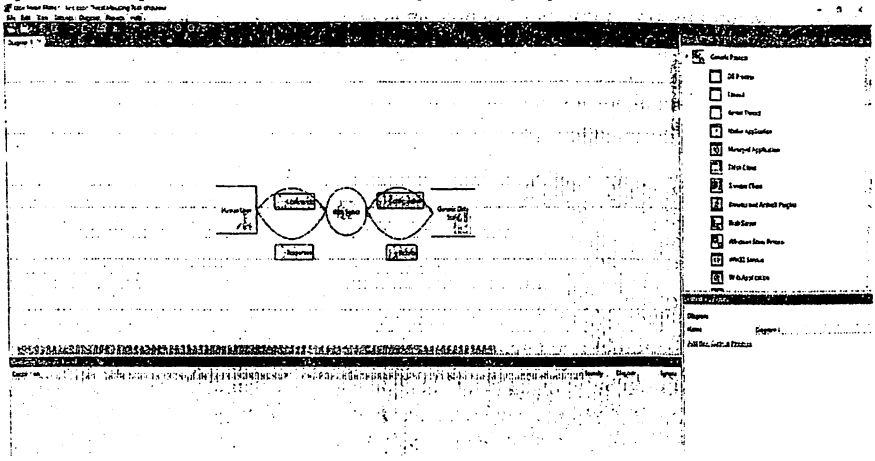
Создание модели

В этом разделе мы понаблюдаем за следующими людьми:

- Кристина (разработчик);
- Рикардо (руководитель программы);
- Ашиш (тестировщик).

Они собираются разработать свою первую модель рисков.

Рикардо: "Привет, Кристина. Я работал над диаграммой модели рисков и хотел убедиться, что мы получили правильные сведения. Поможешь мне просмотреть их?" Кристина: "Конечно. Давай посмотрим". Рикардо открывает средство и предоставляет общий доступ к нему Кристине.



6.3 - рисунок. Создание модели.

Кристина: "Хорошо, все выглядит просто, но можешь помочь мне разобраться? Рикардо: "Конечно! Вот какая здесь структура:

– наш реальный пользователь изображен в виде внешней сущности — квадрата;

– он отправляет команды на наш веб-сервер — круг;

– веб-сервер связывается с базой данных (две параллельные линии).

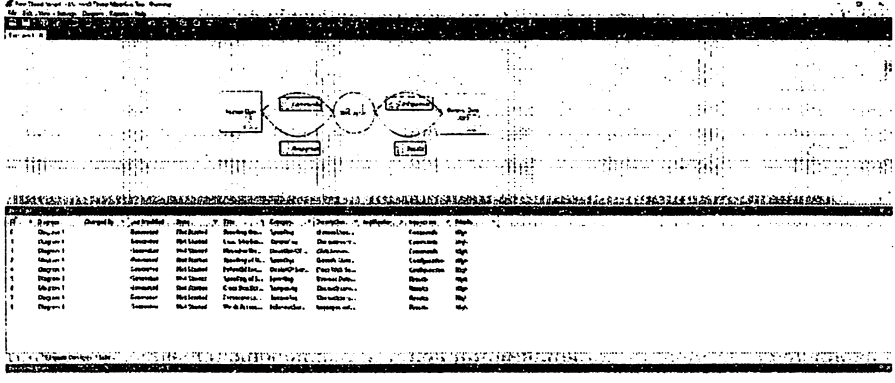
То, что Рикардо показал Кристине, — это DFD, сокращение для **диаграммы потока данных**. С помощью средства моделирования угроз пользователи могут указывать границы доверия, обозначенные красными пунктирными линиями, чтобы показать, в каких местах различные сущности находятся под контролем. Например, для проверки подлинности ИТ-администраторам требуется система Active Directory, таким образом она находится за пределами их контроля.

Кристина: "Похоже, это мне подходит. Как насчет угроз?" Рикардо: "Давай покажу".

Анализ угроз

Как только он щелкает аналитический отчет в меню значков (файл с увеличительным стеклом), открывается список созданных угроз, найденных средством моделирования угроз на основе шаблона по умолчанию, в котором используется подход SDL с названием **STRIDE (спуфинг, мошенничество, разглашение сведений, отказ, отказ в обслуживании и несанкционированное повышение привилегий)**. Идея заключается в том, что программное обеспечение подвергается прогнозируемому набору угроз, которые можно найти с помощью этих 6 категорий.

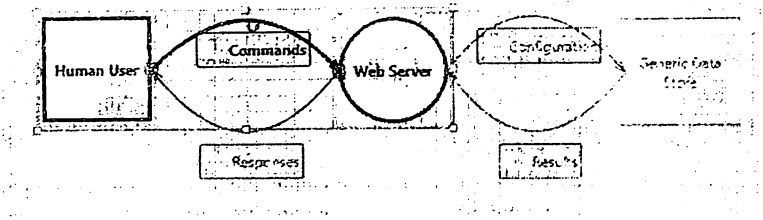
Этот подход имеет сходство с охраной дома, при которой на окна и двери устанавливается механизм блокировки перед добавлением сигнализации или преследованием вора.



6.4 - рисунок. Анализ угроз.

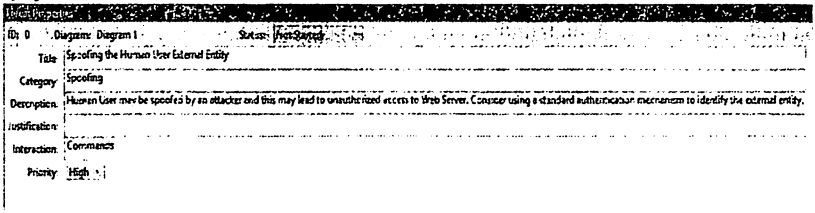
Рикардо начинает с выбора первого элемента в списке. Происходит следующее:

Во-первых, улучшается взаимодействие между двумя наборами элементов.



6.5 - рисунок. Модуль угроз.

Во-вторых, в окне свойств угроз появляются дополнительные сведения об угрозах.



6.6 - рисунок. Свойство угроз.

Созданная угроза помогает ему понять потенциальные изъяны проектирования. Классификация STRIDE дает ему представление о потенциальных векторах атаки, тогда как в дополнительном описании содержатся сведения о том, что именно идет не так, и возможные способы решения проблемы. Он может использовать изменяемые поля для записи примечаний в сведениях об обосновании или изменения порядка приоритета в зависимости от шкалы ошибок его организации.

В шаблонах Azure есть дополнительные сведения, с помощью которых пользователи могут не только выявить проблему, но и узнать способы ее устранения путем добавления описаний примеров и гиперссылок в документации Azure.

В описании содержатся сведения о важности добавления механизма проверки подлинности, чтобы пользователи не подвергались спуфингу, в результате чего выявлена первая угроза, которую необходимо устранить. Через несколько минут разговора с Кристиной они понимают важность применения управления доступом и ролей. Рикардо делает несколько заметок, чтобы убедиться в том, что они реализованы.

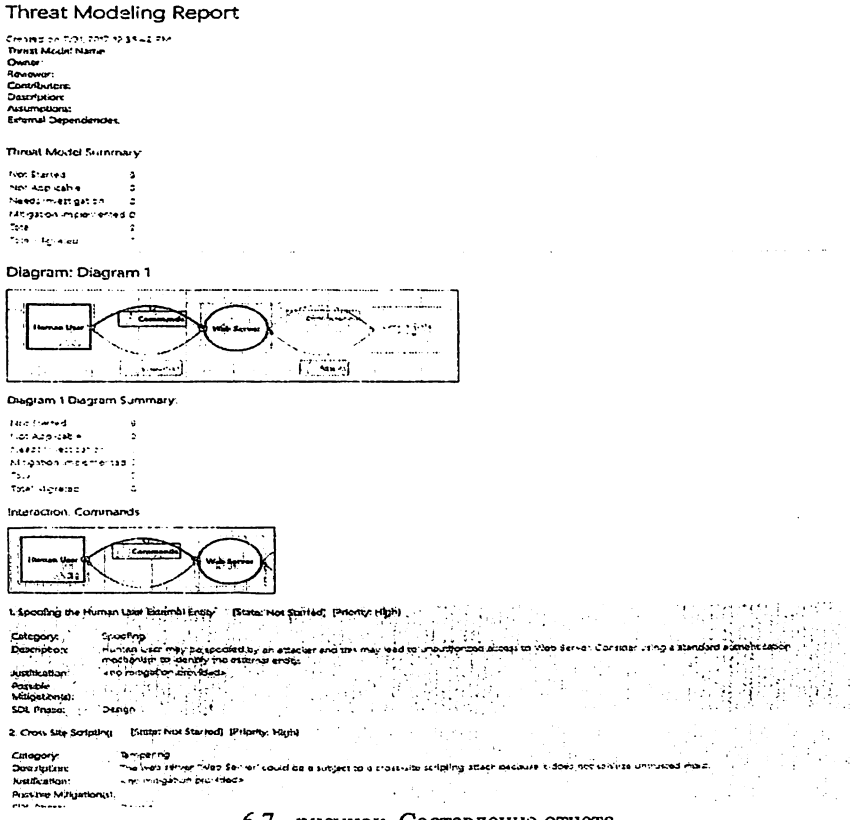
Так как Рикардо открыл угрозы в разделе "Раскрытие информации", он понял, что для плана управления доступом необходимы некоторые учетные записи только для чтения для аудита и создания отчетов. Он задался вопросом, будет ли это новая угроза, но устранения рисков были теми же, поэтому он

отметил угрозы соответствующим образом. Он также подумал о раскрытии информации и понял, что для лент резервного копирования скоро потребуется шифрование, а это задание для рабочей группы.

Для угроз, неприменимых для разработки из-за имеющихся устранений рисков или гарантий безопасности, можно выбрать значение "Неприменимо" из раскрывающегося списка "Состояние". Есть три варианта: "Не запущено" — выбор по умолчанию, "Требуется исследование" — позволяет отслеживать элементы и Mitigated (Устранено) — после полной отработки.

Совместное использование отчетов &

После того как Рикардо перейдет в список с помощью Кристина и добавит важные примечания, способы устранения и обоснования, изменения приоритета и состояния, он выбирает отчеты — > Создание полного отчета о сохранении отчета > , который выводит хороший отчет для ИТ-специалиста с коллегами, чтобы обеспечить правильную работу по обеспечению безопасности.


Threat Modeling Report

Owner:
Reviewer:
Contributors:
Description:
Assumptions:
External Dependencies:

Threat Model Summary:

Diagram: Diagram 1

Diagram 1 Diagram Summary:

Interaction: Commands

1. Spoofing the Human User External Entity (State: Not Started) (Priority: High)

Category: Spoofing

Justification:

Possible Mitigation(s):

SDL Phase: Design

2. Cross Site Scripting (State: Not Started) (Priority: High)

Category: Tampering

Justification:

Possible Mitigation(s):

6.7 - рисунок. Составление отчета.

Если вместо этого Рикардо хочет предоставить общий доступ к файлам, он сохраняет их в учетной записи OneDrive своей организации. После этого он может скопировать ссылку на документ и поделиться ею с коллегами.

Собрания на тему моделирования угроз

Когда Рикардо отправил свою модель рисков своей коллеге с использованием OneDrive, Ашиш, тестировщик, был разочарован. Казалось, Рикардо и Кристина упустили довольно много важных моментов, которые легко скомпрометировать. Его скепсис относится и к моделям рисков.

В этом сценарии после того, как Ашиш перехватил модель рисков, он назначил два собрания по моделированию угроз: одно собрание по синхронизации процесса и рассмотрению диаграмм, а затем второе собрание по обзору угроз и выходу.

На первом собрании Ашиш потратил 10 минут на объяснение всем процесса моделирования угроз SDL. Затем он начал подробно объяснять диаграмму модели рисков. В течение пяти минут был выявлен недостающий важный компонент.

Несколькими минутами позже Ашиш и Рикардо начали подробно обсуждать создание веб-сервера. Это не самый лучший вариант продолжения собрания, но все согласились, что обнаружение проблемы на раннем этапе поможет сэкономить время в будущем.

На втором собрании команда рассмотрела угрозы, способы их устранения, а также закончила работу с моделью рисков. Они поместили документ в систему управления версиями и продолжили разработку.

Задание

Моделировать один проект используя Microsoft Modeling Thread Tools. Анализировать проект и исправит недостатки проекта.

Контрольные вопросы

1. Объясните базы данных единую информационную систему библиотека.
2. Объясните информационная система онлайн-торговли.
3. Объясните информационная система мобильного приложения для банковских услуг.

Практическая работа №7

Тема: Выполнить проверку входных данных в C++.

Цель работы: изучить время проверки до времени использования.

Теоритическая часть

Если программа работает с изначально заданными данными, то она полезна только один раз, потому что результат будет всегда один и тот же. Гораздо эффективнее она становится, когда можно использовать разные данные:

- числа для вычисления;
- команды пользователя;
- изображения;
- аудио и видео;
- текст и так далее.

Команды ввода и вывода в C++

В самом начале кода каждой программы мы подключаем библиотеку iostream — Input/Output Stream (поток ввода/вывода). Именно в ней находится команда cout, что позволяет выводить данные на экран консоли. В ней же есть команда cin, которая, наоборот, запрашивает пользовательский ввод.

Большинство программ, имеющих какой-либо пользовательский интерфейс, должны обрабатывать вводимые пользователем данные. В программах, которые мы писали, мы использовали std::cin, чтобы попросить пользователя ввести текст. Поскольку ввод текста имеет произвольную форму (пользователь может вводить что угодно), пользователю очень легко ввести данные, которые не ожидаются.

При написании программ вы всегда должны учитывать, как пользователи (непреднамеренно или наоборот) будут использовать ваши программы некорректно. Хорошо написанная программа будет предвидеть, как пользователи будут использовать ее неправильно, и либо аккуратно обработает эти случаи, либо вообще предотвратит их появление (если это возможно). Программа, которая хорошо обрабатывает случаи ошибок, называется **надежной**.

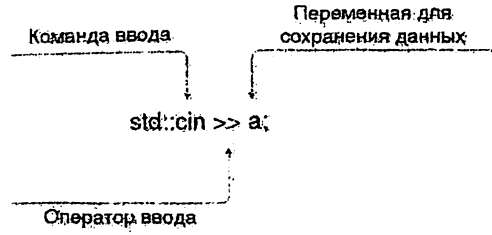
В этой работе мы подробно рассмотрим способы, которыми пользователь может вводить недопустимые данные через std::cin, и покажем вам несколько разных способов обработки таких случаев.

Процесс проверки того, соответствуют ли пользовательские входные данные тому, что ожидает программа, называется **проверкой ввода**.

Есть три основных способа проверки ввода:

- встроенный (по мере печати пользователя):
- прежде всего, не позволить пользователю вводить недопустимые данные;
- пост-запись (после печати пользователя):
- позволить пользователю ввести в строку всё, что он хочет, затем проверить правильность строки и, если она корректна, преобразовать строку в окончательный формат переменной;

– позволить пользователю вводить всё, что он хочет, позволить std::cin и operator>> попытаться извлечь данные и обработать случаи ошибок.



7.1 - рисунок. Структура ввода данных.

Некоторые графические пользовательские интерфейсы и расширенные текстовые интерфейсы позволяют проверять входные данные, когда пользователь их вводит (символ за символом). В общем случае, программист предоставляет функцию проверки, которая принимает входные данные, введенные пользователем, и возвращает true, если входные данные корректны, и false в противном случае. Эта функция вызывается каждый раз, когда пользователь нажимает клавишу. Если функция проверки возвращает истину, клавиша, которую только что нажал пользователь, принимается. Если функция проверки возвращает false, введенный пользователем символ отбрасывается (и не отображается на экране). Используя этот метод, вы можете гарантировать, что любые входные данные, вводимые пользователем, гарантированно будут корректными, потому что любые недопустимые нажатия клавиш обнаруживаются и немедленно отбрасываются. Но, к сожалению, std::cin не поддерживает этот стиль проверки.

Поскольку строки не имеют никаких ограничений на ввод символов, извлечение гарантированно завершится успешно (хотя помните, что std::cin прекращает извлечение на первом неведущем пробельном символе). После того, как строка введена, программа может проанализировать эту строку, чтобы узнать, корректна она или нет. Однако анализ строк и преобразование вводимых строк в другие типы (например, числа) может быть сложной задачей, поэтому это делается только в редких случаях.

Чаще всего мы позволяем std::cin и оператору извлечения выполнять эту тяжелую работу. В этом методе мы позволяем пользователю вводить всё, что он хочет, заставляем std::cin и operator>> попытаться извлечь данные и справиться с последствиями, если это не удастся. Это самый простой способ, о котором мы поговорим ниже.

Практическая часть

Рассмотрим следующую программу-калькулятор, в которой нет обработки ошибок:

```
#include <iostream>
double getDouble()
{
```

```
    std::cout << "Enter a double value: ";
    double x{};
    std::cin >> x;
    return x;
}
char getOperator()
{
    std::cout << "Enter one of the following: +, -, *, or /: ";
    char op{};
    std::cin >> op;
    return op;
}
void printResult(double x, char operation, double y)
{
    switch (operation)
    {
    case '+':
        std::cout << x << " + " << y << " is " << x + y << '\n';
        break;
    case '-':
        std::cout << x << " - " << y << " is " << x - y << '\n';
        break;
    case '*':
        std::cout << x << " * " << y << " is " << x * y << '\n';
        break;
    case '/':
        std::cout << x << " / " << y << " is " << x / y << '\n';
        break;
    }
}
int main()
{
    double x{ getDouble() };
    char operation{ getOperator() };
    double y{ getDouble() };
    printResult(x, operation, y);
    return 0;
}
```

Эта простая программа просит пользователя ввести два числа и математический оператор.

```
Enter a double value: 5
Enter one of the following: +, -, *, or /: *
Enter a double value: 7
5 * 7 is 35
```

Теперь подумайте, где неверный ввод пользователя может нарушить работу этой программы.

Сначала мы просим пользователя ввести несколько чисел. Что, если он введет что-то, отличающееся от числа (например, 'q')? В этом случае извлечение не удастся.

Во-вторых, мы просим пользователя ввести один из четырех возможных символов. Что, если он введет символ, отличный от ожидаемых? Мы сможем извлечь входные данные, но пока не обрабатываем то, что происходит после.

В-третьих, что, если мы попросим пользователя ввести символ, а он введет строку типа "*q hello". Хотя мы можем извлечь нужный нам символ '*', в буфере останутся дополнительные входные данные, которые могут вызвать проблемы в будущем.

Типы недопустимых входных текстовых данных

Обычно мы можем разделить ошибки ввода текста на четыре типа:

1. извлечение входных данных выполняется успешно, но входные данные не имеют смысла для программы (например, ввод 'k' в качестве математического оператора);
2. извлечение входных данных выполняется успешно, но пользователь вводит дополнительные данные (например, вводя "*q hello" в качестве математического оператора);
3. ошибка извлечения входных данных (например, попытка ввести 'q' при запросе ввода числа);
4. извлечение входных данных выполнено успешно, но пользователь выходит за пределы значения числа.

Таким образом, чтобы сделать наши программы устойчивыми, всякий раз, когда мы запрашиваем у пользователя ввод, мы в идеале должны определить, может ли произойти каждый из вышеперечисленных возможных вариантов, и если да, написать код для обработки этих случаев.

Давайте разберемся в каждом из этих случаев и в том, как их обрабатывать с помощью std::cin.

Случай ошибки 1: извлечение успешно, но входные данные не имеют смысла

Это самый простой случай. Рассмотрим следующий вариант выполнения приведенной выше программы:

```
Enter a double value: 5
Enter one of the following: +, -, *, or /: k
Enter a double value: 7
```

В этом случае мы попросили пользователя ввести один из четырех символов, но вместо этого он ввел 'k'. 'k' – допустимый символ, поэтому std::cin успешно извлекает его в переменную op, и она возвращается в main. Но наша программа не ожидала этого, поэтому она не обрабатывает этот случай правильно (и, таким образом, ничего не выводит).

Решение здесь простое: выполните проверку ввода. Обычно она состоит из 3 шагов:

1. убедитесь, что пользовательский ввод соответствует вашим ожиданиям;

2. если да, верните значение вызывающей функции;

3. если нет, сообщите пользователю, что что-то пошло не так, и попросите его повторить попытку.

Вот обновленная функция getOperator(), которая выполняет проверку ввода.

```
char getOperator()
{
  while (true) // Цикл, пока пользователь не введет допустимые данные
  {
    std::cout << "Enter one of the following: +, -, *, or /: ";
    char operation{};
    std::cin >> operation;
    // Проверяем, ввел ли пользователь подходящие данные
    switch (operation)
    {
    case '+':
    case '-':
    case '*':
    case '/':
      return operation; // возвращаем символ вызывающей функции
    default: // в противном случае сообщаем пользователю, что пошло не так
      std::cout << "Oops, that input is invalid.  Please try again.\n";
    }
  } // и попробуем еще раз
}
```

Случай ошибки 2: извлечение успешно, но с посторонними входными данными

Рассмотрим следующий вариант выполнения приведенной выше программы:

```
Enter a double value: 5*7
Как думаете, что будет дальше?
Enter a double value: 5*7
Enter one of the following: +, -, *, or /: Enter a double value: 5 * 7 is 35
```

Программа выводит правильный ответ, но форматирование испорчено. Давайте подробнее разберемся, почему.

Когда пользователь вводит "5*7" в качестве вводных данных, эти данные попадают в буфер. Затем оператор >> извлекает 5 в переменную x, оставляя в буфере "*7\n". Затем программа напечатает "Enter one of the following: +, -, *, or /:". Однако когда был вызван оператор извлечения, он видит символы "*7\n", ожидающие извлечения в буфере, поэтому он использует их вместо того, чтобы запрашивать у пользователя

дополнительные данные. Следовательно, он извлекает символ '*', оставляя в буфере "7\n".

После запроса пользователя ввести другое значение double, из буфера извлекается 7 без ожидания ввода пользователя. Поскольку у пользователя не было возможности ввести дополнительные данные и нажать Enter (добавляя символ новой строки), все запросы в выводе идут вместе в одной строке, даже если вывод правильный.

Хотя программа работает, выполнение запутано. Было бы лучше, если бы любые введенные посторонние символы просто игнорировались. К счастью, символы игнорировать легко:

```
// очищаем до 100 символов из буфера или пока не будет удален символ '\n'
std::cin.ignore(100, '\n');
```

Этот вызов удалит до 100 символов, но если пользователь ввел более 100 символов, мы снова получим беспорядочный вывод. Чтобы игнорировать все символы до следующего символа '\n', мы можем передать std::numeric_limits<std::streamsize>::max() в std::cin.ignore(). std::numeric_limits<std::streamsize>::max() возвращает наибольшее значение, которое может быть сохранено в переменной типа std::streamsize. Передача этого значения в std::cin.ignore() приводит к отключению проверки счетчика.

Чтобы игнорировать всё, вплоть до следующего символа '\n', мы вызываем

```
std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
```

Поскольку эта строка довольно длинная для того, что она делает, будет удобнее обернуть ее в функцию, которую можно вызвать вместо std::cin.ignore().

```
#include <limits> // для std::numeric_limits

void ignoreLine()
{
  std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
}
```

Поскольку последний введенный пользователем символ должен быть '\n', мы можем указать std::cin игнорировать символы в буфере, пока не найдет символ новой строки (который также будет удален).

Давайте обновим нашу функцию getDouble(), чтобы игнорировать любой посторонний ввод:

```
double getDouble()
{
  std::cout << "Enter a double value: ";
  double x{};
  std::cin >> x;
  ignoreLine();
  return x;
```

```
}
```

Теперь наша программа будет работать, как ожидалось, даже если мы введем "5*7" при первом запросе ввода – 5 будет извлечено, а остальные символы из входного буфера будут удалены. Поскольку входной буфер теперь пуст, при следующем выполнении операции извлечения данные у пользователя будут запрашиваться правильно!

Случай ошибки 3: сбой при извлечении

Теперь рассмотрим следующий вариант выполнения нашей программы калькулятора:

```
Enter a double value: a
```

Неудивительно, что программа работает не так, как ожидалось, но интересно, как она дает сбой:

```
Enter a double value: a
Enter one of the following: +, -, *, or /: Enter a double value:
```

и программа внезапно завершается.

Это очень похоже на случай ввода посторонних символов, но немного отличается. Давайте посмотрим подробнее.

Когда пользователь вводит 'a', этот символ помещается в буфер. Затем оператор >> пытается извлечь 'a' в переменную x, которая имеет тип double. Поскольку 'a' нельзя преобразовать в double, оператор >> не может выполнить извлечение. В этот момент происходят две вещи: 'a' остается в буфере, а std::cin переходит в «режим отказа».

Находясь в «режиме отказа», будущие запросы на извлечение входных данных будут автоматически завершаться ошибкой. Таким образом, в нашей программе калькулятора вывод запросов всё еще печатается, но любые запросы на дальнейшее извлечение игнорируются. Программа просто выполняется до конца, а затем завершается (без вывода результата потому, что мы не прочитали допустимую математическую операцию).

К счастью, мы можем определить, завершилось ли извлечение сбоем, и исправить это:

```
if (std::cin.fail()) // предыдущее извлечение не удалось?
{
  // да, давайте разберемся с ошибкой
  std::cin.clear(); // возвращаем нас в "нормальный" режим работы
  ignoreLine();     // и удаляем неверные входные данные
}
```

Вот и всё!

Давайте, интегрируем это в нашу функцию getDouble():

```
double getDouble()
{
  while (true) // Цикл, пока пользователь не введет допустимые данные
  {
    std::cout << "Enter a double value: ";
    double x{};
```

```
        std::cin >> x;
        if (std::cin.fail()) // предыдущее извлечение не удалось?
        {
          // да, давайте разберемся с ошибкой
          std::cin.clear(); // возвращаем нас в "нормальный" режим работы
          ignoreLine();     // и удаляем неверные входные данные
        }
        else // иначе наше извлечение прошло успешно
        {
          ignoreLine();
          return x; // поэтому возвращаем извлеченное нами значение
        }
      }
    }
```

Случай ошибки 4: извлечение успешно, но пользователь выходит за пределы значения числа

Рассмотрим следующий простой пример:

```
#include <cstdint>
#include <iostream>
int main()
{
  std::int16_t x{}; // x - 16 бит, может быть от -32768 до 32767
  std::cout << "Enter a number between -32768 and 32767: ";
  std::cin >> x;
  std::int16_t y{}; // y - 16 бит, может быть от -32768 до 32767
  std::cout << "Enter another number between -32768 and 32767: ";
  std::cin >> y;
  std::cout << "The sum is: " << x + y << '\n';
  return 0;
}
```

Что произойдет, если пользователь введет слишком большое число (например, 40000)?

Enter a number between -32768 and 32767: 40000

Enter another number between -32768 and 32767: The sum is: 32767

Проверка консольного ввода в C++

Как мы уже говорили, компьютер может только выполнять инструкции. Когда что-то идёт не по плану, он не способен самостоятельно решить, что ему делать, поэтому выдаёт ошибку.

Допустим, нам нужно, чтобы пользователь ввёл свой возраст. Мы ожидаем число вроде 8, 15 или 21, но кто-то может ввести эти числа прописью, например «двадцать один». Для программы эти варианты будут неожиданными, потому что она уже подготовила переменную типа int — строка в неё никак не влезет.

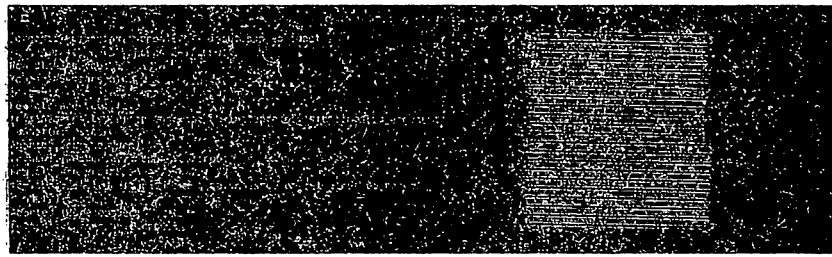
Чтобы избежать таких ошибок, любые данные от пользователей стоит принимать как строку, а потом переводить в нужный нам тип. Процесс преобразования данных из одного типа в другой называется **конвертацией.**

Конвертация данных

В C++ самый простой способ конвертировать строку в число — использовать функцию *stoi ()* или аналогичную:

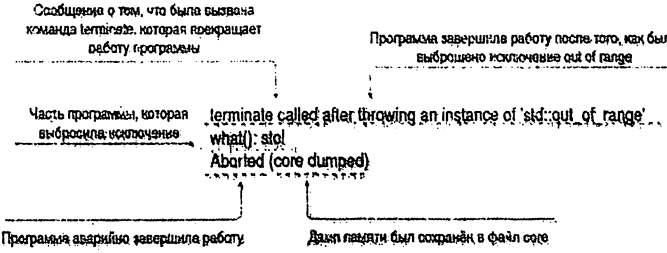
```
std::string ageInput;
int age;
std::cout << "How old are you: ";
std::cin >> ageInput;
//В круглых скобках функции указывается значение, которое нужно конвертировать
age = stoi(ageInput);
std::cout << "You are " << age << " years old. \n";
```

Вот пример корректной и некорректной конвертации:



7.2 - рисунок. Конвертации переменных.

Как видно на скриншоте, в первом случае всё прошло успешно, но в следующих случаях программа выдала ошибку. Давайте рассмотрим последнюю:



7.3 - рисунок. Проверка конвертации.

Разберём некоторые моменты подробнее:

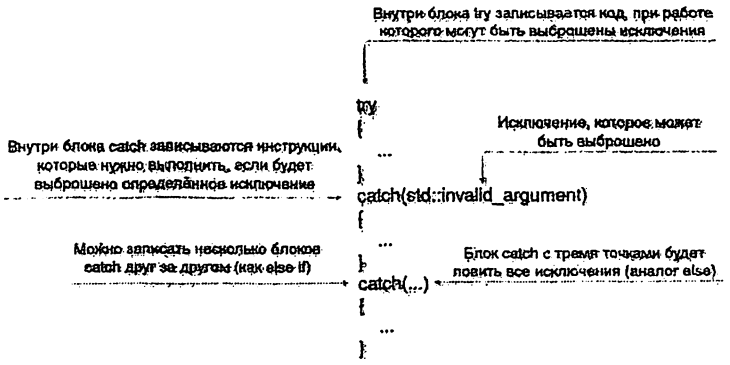
- **Исключение (exception)** — это что-то вроде ошибки, но не совсем. Разные части программы могут «выбрасывать» их при определённых условиях, чтобы сказать: что-то идёт не так; в данном случае — out of range (выход из диапазона). Это исключение было выброшено, потому что введённое число больше, чем может поместиться в переменную типа int.

- **Дамп памяти (memory dump)** — это содержимое рабочей памяти одного процесса, ядра или всей операционной системы. Дамп можно посмотреть, чтобы понять, каким было состояние программы и почему она аварийно завершила работу.

Чтобы программа не закрывалась при выбрасывании исключений, их нужно обработать.

Обработка исключений в C++

Для этого нам пригодится конструкция *try-catch*.



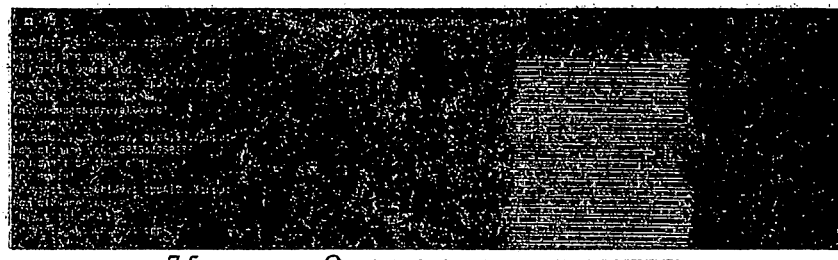
7.4 - рисунок. Конструкция *try-catch*.

Вот код программы, которая проверяет корректность введённых данных:

```
std::string ageInput;
int age = 0;
std::cout << "How old are you: ";
std::cin >> ageInput;
//Названия исключений можно посмотреть в сообщениях об ошибке
try
{
  age = stoi(ageInput);
}
catch(std::invalid_argument)
{
  //Говорим, что можно вводить только числа
  std::cout << "Only numbers are allowed! \n";
}
catch(std::out_of_range)
{
  //Говорим, что число слишком большое
  std::cout << "You can't be that old! \n";
}
catch(...)
{
```

```
    //Если будет выброшено какое-то исключение, которое не обработано выше, то говорим, что возникла неизвестная ошибка
      std::cout << "Unknown error! \n";
    }
    if(age > 0)
    {
      std::cout << "You are " << age << " years old. \n";
    }
    else
    {
      std::cout << "Try again! \n";
    }
```

Теперь можно запустить её и попробовать ввести любые значения — программа определит, какое сообщение вывести, и благополучно завершится.



7.5 - рисунок. Окно проверки входных данных.

Задание

1. Напишите простую программу, которая проверяет имя пользователя (прописные, строчные и пробелы).

2. Напишите простую программу, которая проверяет что номер телефона указан в международном формате (+998 xx xxx xx xx).

3. Напишите простую программу для проверки правильности ввода адреса электронной почты.

4. Напишите простую программу, которая проверяет что введенный пароль «хороший» (хотя бы одна заглавная буква, хотя бы один специальный символ (_, -,), хотя бы одна цифра и строчные буквы и пароль не должен начинаться со специального символа и цифры) .

Контрольные вопросы

1. Какие проблемы и уязвимости возникают при вводе неверной информации в приложения?

2. В языке программирования C++ есть функции, которые проверяют, какое входное значение

3. Какие функции должны быть проверены на входные значения?

Практическая работа №8

Тема: Выполнить блокировку файла и ограничение доступу к созданному файлу.

Цель работы: Получить практические навыки разработки программ с использованием файлового ввода/вывода.

Теоритическая часть

Работа с файлами в С (работает и в С++).

Пример работы с текстовым файлом:

```
#include <iostream.h>
#include <stdio.h>
void main( void )
{
FILE *file;
char* file_name = "file.txt";
char load_string[50] = "none";
file = fopen( file_name, "w" );
fputs( "string", file );
fclose( file );
file = fopen( file_name, "r" );
if( file != 0 )
{
fgets( load_string, 50 , file );
cout << "File not found !!!" << endl;
}
fclose(file);
}
```

Описание функций работы с файлами находятся в библиотеке stdio.h

Сначала надо создать указатель на переменную типа FILE (FILE* file;).

Открытие файла производится вызовом функции fopen (file=fopen(file_name, "w");)

Первый параметр этой функции - имя файла, второй - указывает в каком режиме должен быть открыт файл. "w" - открыть для записи, "r" - открыть для чтения, "a" - дополнение файла(это наиболее часто используемые режимы). Запись и считывание данных из файла осуществляется следующими функциями: fputc, fputs, fgetc, fgets, fprintf, fscanf(описание этих функций см. в stdio.h).

Закрытие файла осуществляется вызовом функции fclose (fclose(file);).

Работа с файлами с помощью MFC(классы CFile, CStdioFile, ...) и стандартный класс MFC CFileDialog.

В библиотеку MFC включено несколько классов для обеспечения работы с файлами. Рассматриваемые ниже классы наследуются от базового класса CFile.

Класс CFile

Класс CFile предназначен для обеспечения работы с файлами. Он позволяет упростить использование файлов, представляя файл как объект, который можно создать, читать, записывать и т.д.

Чтобы получить доступ к файлу, сначала надо создать объект класса CFile. Конструктор класса позволяет сразу после создания такого объекта открыть файл. Но можно открыть файл и позднее, воспользовавшись методом Open.

Открытие и создание файлов

После создания объекта класса CFile можно открыть файл, вызвав метод Open. Методу надо указать путь к открываемому файлу и режим его использования. Прототип метода Open имеет следующий вид:

```
virtual BOOL Open(LPCTSTR lpszFileName,
    UINT nOpenFlags, CFileException* pError=NULL);
```

В качестве параметра lpszFileName надо указать имя открываемого файла. Можно указать только имя файла или полное имя файла, включающее полный путь к нему.

Второй параметр nOpenFlags определяет действие, выполняемое методом Open с файлом, а также атрибуты файла. Ниже представлены некоторые возможеые значения параметра nOpenFlags:

CFile::modeCreate - Создается новый файл. Если указанный файл существует, то его содержимое стирается и длина файла устанавливается равной нулю.

CFile::modeNoTruncate - Этот файл предназначен для использования совместно с файлом CFile::modeCreate. Если создается уже существующий файл, то его содержимое не будет удалено.

CFile::modeRead - Файл открывается только для чтения.

CFile::modeReadWrite - Файл открывается для записи и для чтения.

CFile::modeWrite - Файл открывается только для записи.

CFile::typeText - Используется классами, порожденными от класса CFile, например CStdioFile, для работы с файлами в текстовом режиме. Текстовый режим обеспечивает преобразование комбинации символа возврата каретки и символа перевода строки.

CFile::Binary - Используется классами, порожденными от класса CFile, например CStdioFile, для работы с файлами в двоичном режиме.

Необязательный параметр pError, который является указателем на объект класса CFileException, используется только в том случае, если выполнение операции с файлом вызовет ошибку. При этом в объект, указываемый pError, будет записана дополнительная информация.

Метод Open возвращает не нулевое значение, если файл открыт и нуль в случае ошибки. Ошибка при открытии файла может случиться, например, если методу Open указан для чтения несуществующий файл.

Идентификатор открытого файла

В состав класса CFile входит элемент данных m_hFile типа UINT. В нем хранится идентификатор открытого файла. Если объект класса CFile уже создан, но файл еще не открыт, то в переменной m_hFile записана константа hFileNull.

Обычно идентификатор открытого файла непосредственно не используется. Методы класса CFile позволяют выполнять практически любые операции с файлами и не требуют указывать идентификатор файла. Так как m_hFile является элементом класса, то реализация его методов всегда имеет свободный доступ к нему.

Закрытие файлов

После завершения работы с файлом, его надо закрыть. Класс CFile имеет для этого специальный метод Close. Нужно заметить, что если был создан объект класса CFile и открыт файл, а затем объект удаляется, то связанный с ним файл закрывается автоматически с помощью деструктора.

Чтение и запись файлов

Для доступа к файлам предназначено несколько методов класса CFile: Read, ReadHuge, Write, WriteHuge, Flush. Методы Read и ReadHuge предназначены для чтения данных из предварительно открытого файла. В 32-разрядных операционных системах оба метода могут одновременно считать из файла больше 65535 байт. Спецификация ReadHuge считается устаревшей и оставлена только для совместимости с 16-разрядными операционными системами.

Данные, прочитанные из файла, записываются в буфер lpBuf. Параметр nCount определяет количество байт, которое надо считать из файла. Фактически из файла может быть считано меньше байт, чем запрошено параметром nCount. Это происходит, если во время чтения достигнут конец файла. Методы возвращают количество байт, прочитанных из файла.

Для записи в файл предназначены методы Write и WriteHuge. В 32-разрядных операционных системах оба метода могут одновременно записывать в файл больше 65535 байт. Методы записывает в открытый файл nCount байт из буфера lpBuf. В случае возникновения ошибки записи, например переполнения диска, методы вызывает обработку исключения.

Метод Flush

Когда используется метод Write или WriteHuge для записи данных на диск, они некоторое время могут находиться во временном буфере. Чтобы убедиться, что необходимые изменения внесены в файл на диске, нужно воспользоваться методом Flush.

Операции с файлами

В состав класса входят методы, позволяющие выполнять над файлами различные операции, например копирование, переименование, удаление, изменение атрибутов.

Для изменения имени файла класс CFile включает статический метод Rename, выполняющий функции этой команды. Метод нельзя использовать

для переименования каталогов. В случае возникновения ошибки метод вызывает исключение.

Для удаления файлов в классе CFile включен статический метод Remove, позволяющий удалить указанный файл. Этот метод не позволяет удалять каталоги. Если удалить файл невозможно, то метод вызывает исключение.

Чтобы определить дату и время создания файла, его длину и атрибуты, предназначен статический метод GetStatus. Существует две разновидности метода - первый определен как виртуальный, а второй - как статический метод.

Виртуальная версия метода GetStatus определяет состояние открытого файла, связанного с данным объектом класса CFile. Этот метод вызывается только тогда, когда объект класса CFile создан и файл открыт.

Статическая версия метода GetStatus позволяет определить характеристики файла, не связанного с объектом класса CFile. Чтобы воспользоваться этим методом, необязательно предварительно открывать файл.

Блокировка

В состав класса включены методы LockRange и UnlockRange, позволяющие заблокировать один или несколько фрагментов данных файла для доступа из других процессов. Если приложение пытается повторно блокировать данные, уже заблокированные раньше этим или другим приложением, вызывается исключение. Блокировка представляет собой один из механизмов, позволяющих нескольким приложениям или процессам одновременно работать с одним файлом, не мешая друг другу.

Установить блокировку можно с помощью метода LockRange. Чтобы снять установленные блокировки, надо воспользоваться методом UnlockRange. Если в одном файле установлены несколько блокировок, то каждая из них должна сниматься отдельным вызовом метода UnlockRange.

Позиционирование

Чтобы переместить указатель текущей позиции файла в новое положение, можно воспользоваться одним из следующих методов класса CFile - Seek, SeekToBegin, SeekToEnd. В состав класса CFile также входят методы, позволяющие установить и изменить длину файла, - GetLength, SetLength.

При открытии файла указатель текущей позиции файла находится в самом начале файла. Когда порция данных прочитана или записана, то указатель текущей позиции перемещается в сторону конца файла и указывает на данные, которые будут читаться или записываться очередной операцией чтения или записи в файл.

Чтобы переместить указатель текущей позиции файла в любое место, можно воспользоваться универсальным методом Seek. Он позволяет переместить указатель на определенное число байт относительно начала, конца или текущей позиции указателя.

Чтобы переместить указатель в начало или конец файла, наиболее удобно использовать специальные методы. Метод SeekToBegin перемещает указатель в начало файла, а метод SeekToEnd - в его конец.

Но для определения длины открытого файла совсем необязательно перемещать его указатель. Можно воспользоваться методом GetLength. Этот метод также возвращает длину открытого файла в байтах. Метод SetLength позволяет изменить длину открытого файла. Если при помощи этого метода размер файла увеличивается, то значение последних байт не определено.

Текущую позицию указателя файла можно определить с помощью метода GetPosition. Возвращаемое методом GetPosition 32-разрядное значение определяет смещение указателя от начала файла.

Характеристики открытого файла

Чтобы определить расположение открытого файла на диске, надо вызвать метод GetFilePath. Этот метод возвращает объект класса CString, в котором содержится полный путь файла, включая имя диска, каталоги, имя файла и его расширение.

Если требуется определить только имя и расширение открытого файла, можно воспользоваться методом GetFileName. Он возвращает объект класса CString, в котором находится имя файла. В случае, когда нужно узнать только имя открытого файла без расширения, пользуются методом GetFileTitle.

Следующий метод класса CFile позволяет установить путь файла. Это метод не создает, не копирует и не изменяет имени файла, он только заполняет соответствующий элемент данных в объекте класса CFile.

Класс CMemFile

В библиотеку MFC входит класс CMemFile, наследуемый от базового класса CFile. Класс CMemFile представляет файл, размещенный, в оперативной памяти. С объектами класса CMemFile так же, как и с объектами класса CFile. Отличие заключается в том, что файл, связанный с объектом CMemFile, расположен не на диске, а в оперативной памяти компьютера. За счет этого операции с таким файлом происходят значительно быстрее, чем с обычными файлами.

Работая с объектами класса CMemFile, можно использовать практически все методы класса CFile, которые были описаны выше. Можно записывать данные в такой файл или считывать их. Кроме этих методов в состав класса CMemFile включены дополнительные методы.

Для создания объектов класса CMemFile предназначено два различных конструктора. Первый конструктор CMemFile имеет всего один необязательный параметр nGrowBytes:

```
CMemFile(UINT nGrowBytes=1024);
```

Этот конструктор создает в оперативной памяти пустой файл. После создания файл автоматически открывается (не нужно вызывать метод Open).

Когда начинается запись в такой файл, автоматически выделяется блок памяти. Для получения памяти методы класса CMemFile вызывают стандартные функции malloc, realloc и free. Если выделенного блока памяти недостаточно, его размер увеличивается. Увеличение блока памяти файла происходит по частям по nGrowBytes байт. После удаления объекта класса CMemFile используемая память автоматически возвращается системе.

Второй конструктор класса CMemFile имеет более сложный прототип. Это конструктор используется в тех случаях, когда программист сам выделяет память для файла:

CMemFile(BYTE* lpBuffer, UINT nBufferSize, UINT nGrowBytes=0);

Параметр lpBuffer указывает на буфер, который будет использоваться для файла. Размер буфера определяется параметром nBufferSize.

Необязательный параметр nGrowBytes используется более комплексно, чем в первом конструкторе класса. Если nGrowBytes содержит нуль, то созданный файл будет содержать данные из буфера lpBuffer. Длина такого файла будет равна nBufferSize.

Если nGrowBytes больше нуля, то содержимое буфера lpBuffer игнорируется. Кроме того, если в такой файл записывается больше данных, чем помещается в отведенном буфере, то его размер автоматически увеличивается. Увеличение блока памяти файла происходит по частям по nGrowBytes байт.

Класс CMemFile позволяет получить указатель на область памяти, используемую файлом. Через этот указатель можно непосредственно работать с содержимым файла, не ограничивая себя методами класса CFile. Для получения указателя на буфер файла можно воспользоваться методом Detach. Перед этим полезно определить длину файла (и соответственно размер буфера памяти), вызвав метод GetLength.

Метод Detach закрывает данный файл и возвращает указатель на используемый им блок памяти. Если опять потребуется открыть файл и связать с ним оперативный блок памяти, нужно вызвать метод Attach.

Нужно отметить, что для управления буфером файла класс CMemFile вызывает стандартные функции malloc, realloc и free. Поэтому, чтобы не нарушать механизм управления памятью, буфер lpBuffer должен быть создан функциями malloc или calloc.

Класс CStdioFile

Тем, кто привык пользоваться функциями потокового ввода/вывода из стандартной библиотеки C и C++, следует обратить внимание на класс CStdioFile, наследованный от базового класса CFile. Этот класс позволяет выполнять буферизированный ввод/вывод в текстовом и двоичном режиме. Для объектов класса CStdioFile можно вызывать практически все методы класса CFile.

В класс CStdioFile входит элемент данных m_pStream, который содержит указатель на открытый файл. Если объект класса CStdioFile создан, но файл еще не открыт, либо закрыт, то m_pStream содержит константу NULL.

Класс CStdioFile имеет три различных конструктора. Первый конструктор класса CStdioFile не имеет параметров. Этот конструктор только создает объект класса, но не открывает никаких файлов. Чтобы открыть файл, надо вызвать метод Open базового класса CFile.

Второй конструктор класса CStdioFile можно вызвать, если файл уже открыт и нужно создать новый объект класса CStdioFile и связать с ним

открытый файл. Этот конструктор можно использовать, если файл был открыт стандартной функцией fopen. Параметр метода должен содержать указатель на файл, полученный вызовом стандартной функции fopen.

Третий конструктор можно использовать, если надо создать объект класса CStdioFile, открыть новый файл и связать его с только что созданным объектом.

Для чтения и записи в текстовый файл класс CStdioFile включает два новых метода: ReadString и WriteString. Первый метод позволяет прочитать из файла строку символов, а второй метод - записать.

Практическая часть

Примеры записи и чтения из файла

Приведем фрагменты кода, в которых демонстрируется использование стандартных диалоговых панелей выбора файла и процедуры чтения и записи в файл.

Открытие файла и чтение из него

```
CString m_Text; // создание стандартной панели выбора файла Open
 CFileDialog DlgOpen(TRUE,(LPCSTR)"txt",NULL,
 OFN_HIDEREADONLY,(LPCSTR)" Text Files (*.txt) |*.txt||");
  // отображение стандартной панели выбора файла Open
 if(DlgOpen.DoModal()==IDOK)
   { // создание объекта и открытие файла для чтения
 CStdioFile File(DlgOpen.GetPathName(),CFile::
                 modeRead|CFile::typeBinary);
 // чтение из файла строки
  CString& ref=m_Text;
  File.ReadString(ref); // передается ссылка на строку m_Text
 }
```

Запустите программу - Build / Rebuild all (будут ошибки), выберите Build / Set active configuration - Win 32 Realise, выберите пункт меню "Project", далее "Settings...", закладку "C/C++", Category - Code Generation и в пункте "Use run-time library" выберите "Multithreaded". После этого сделайте опять Build / Rebuild all и программа будет работать.

Открытие файла и запись из него

```
CString m_Text; // создание стандартной панели выбора файла SaveAs
 CFileDialog DlgSaveAs(FALSE,(LPCSTR)"txt",NULL,
 OFN_HIDEREADONLY|OFN_OVERWRITEPROMPT,
 (LPCSTR)" Text Files (*.txt) |*.txt||");
 // отображение стандартной панели выбора файла SaveAs
 if(DlgSaveAs.DoModal()==IDOK)
 { // создание объекта и открытие файла для записи
  CStdioFile File(DlgSaveAs.GetPathName(),
   CFile::modeCreate|CFile::modeWrite|CFile::typeBinary);
   // запись в файл строки
  File.WriteString((LPCTSTR)m_Text);}
```

FileIOPermission Класс

Управляет возможностью доступа к файлам и папкам.

В следующих примерах показан код, использующий FileIOPermission . После следующих двух строк кода объект f представляет разрешение на чтение всех файлов на локальных дисках клиентского компьютера. Затем в примере кода запрашивается разрешение на определение наличия у приложения разрешения на чтение файлов.

```
C#
FileIOPermission f = new FileIOPermission(PermissionState.None);
f.AllLocalFiles = FileIOPermissionAccess.Read;
try
{      f.Demand(); }
catch (SecurityException s)
{      Console.WriteLine(s.Message);      }
```

После следующих двух строк кода объект f2 представляет разрешения на чтение c:\ test_r и чтение и запись в C:\example\out.txt. Read и Write представляют разрешения на доступ к файлам и папкам, как описано выше. После создания разрешения код требует разрешения, чтобы определить, имеет ли приложение право на чтение и запись в файл.

```
C#
FileIOPermission f2 = new FileIOPermission(FileIOPermissionAccess.Read,
"C:\\test_r");
f2.AddPathList(FileIOPermissionAccess.Write |
FileIOPermissionAccess.Read, "C:\\example\\out.txt");
try
{
  f2.Demand();
}
catch (SecurityException s)
{
  Console.WriteLine(s.Message);
}
```

Задание

Создайте программное обеспечение, предоставляющее простые файловые данные на основе приведенных выше примеров, используя языки программирования Delpi, Java, C++ и C#.

Контрольные вопросы

1. Способы обеспечения конфиденциальности информации?
2. Роль алгоритмов симметричного шифрования в обеспечении конфиденциальности данных?
3. Роль алгоритмов блочного шифрования в обеспечении конфиденциальности данных?

Практическая работа №9

Тема: Необходимые навыки написания правильного кода на языке программирования C++.

Цель работы: Навыки правильного оформления кода и следования общепринятым единообразным принципам необходимо развивать на начальной стадии формирования компетенций программиста.

Теоритическая часть

Дадим определения некоторых терминов и понятий.

Выражение (expression) – комбинация операций, вызовов функций и объектов данных, вычисляемая в соответствии с правилами языка.

Идентификатор (identifier) – уникальное имя программного объекта (переменной, функции, класса, объекта и др.), позволяющее его идентифицировать в некоторой части пространства программы.

Интерпретатор (interpreter) – компьютерная программа, осуществляющая интерпретацию исходного кода программы, то есть его построчную обработку и исполнение.

Исходный код (source code) компьютерной программы– текст на языке программирования, который создается и может быть прочитан человеком. Является входом для компилятора или интерпретатора.

Класс (class) – определенный в программе именованный абстрактный тип данных, определяющий интерфейс и реализацию для своих экземпляров.

Комментарий (comment) – пояснение к исходному коду программы, которое игнорируется компилятором; комментарии включаются непосредственно в исходный код.

Компилятор (compiler) – программа, исполняющая трансляцию исходного кода программы в машинный код и последующую сборку кода исполняемой программы.

Компьютерная программа (computer program) – последовательность инструкций или деклараций на языке программирования для выполнения определённых вычислений, решения задач с помощью компьютера.

Массив (array) – встроенный тип данных языка, значения которого состоят из набора однотипных элементов, доступ к которым осуществляется по индексу.

Метод класса (class method) – инкапсулированный элемент класса, функция в классе

Объект класса (object) – экземпляр класса, переменная типа класса. При исполнении программы каждый её объект находится в определенном состоянии (атрибуты) и характеризуется поведением (методы).

Оператор (statement) – наименьшая самостоятельная единица императивного языка программирования, команда на совершение определенного действия.

Операция (operator) – запись определенного действия над аргументами операции (операндами), выполнение которого, в отличие от оператора, оставляет в месте нахождения операции непустой вычисленный результат.

Переменная (variable) – объект данных программы, связанный с областью памяти, в которой хранится его значение. Переменная обеспечивает доступ к памяти для чтения или изменения значения.

Поле класса (class field, data member) – инкапсулированный элемент данных класса, переменная в классе, атрибут класса.

Препроцессор (preprocessor) – программа, осуществляющая предварительную перед компиляцией обработку исходного кода. Обработка препроцессором – особенность компиляции программ на C/C++.

Спецификация (specification) – законченное описание поведения разрабатываемой программы (подпрограммы); как правило, выполняется на естественном языке и имеет определенную структуру.

Строковый тип (string) – стандартный тип данных языка, значениями которого являются последовательности символов алфавита (строки);

Структура (struct) – встроенный тип данных языка, значения которого состоят из набора разнотипных именованных элементов (полей).

Тип данных (data type) – категория данных, представляющая множество значений данных и операций с ними.

Функция (function) – автономная, как правило – именованная единица программного кода (подпрограмма), которую можно вызвать в программе для исполнения.

Язык программирования (programming language) – формальный искусственный язык, предназначенный для записи компьютерных программ.\

Общие замечания.

Ни один адекватный человек не использует все возможности C++ в одной программе. C++ это настолько мощный и гибкий язык, что все его возможности не то что использовать, а просто запомнить невозможно.

Я как-то сопровождал программу, где дикие украинские студенты использовали макросы, шаблоны, статическое и динамическое подключение библиотек, глобальные, внешние и статические переменные без каких либо префиксов, закрытые конструкторы, конфигурацию в константах и в XML-файле одновременно, а еще вперли туда самодельный упрощенный интерпретатор LUA. Это просто ужас. Примерно 97% кода в той программе можно выбросить и ничего не изменится.

Прикола ради, расскажу историю из своего опыта. У меня у самого была такая ситуация, когда я был наемным программистом, один мой работодатель, мягко выражаясь, кинул меня через пенис. Но, чтобы хоть какую-то часть денег забрать (как у нас это водится, с черной бухгалтерии), мне надо было закончить одну программу. Это был сервер обработки USSD запросов для операторов мобильной связи. Тогда я специально писал read only код. Например, чтобы преобразовать строку в число, в программе создавалось три параллельных потока, в одном строилось число в неинициализированном фрагменте памяти, в другом строка, третий синхронизировал те два, а чтение значений между потоками выполнялось через общий файл, проецируемый в память.

Общее оформление кода.

Альберт Эйнштейн однажды сказал: «Сделай настолько просто, насколько это возможно. Но не проще.». Друзья! Придерживайтесь этого принципа. Или Вы умнее автора приведенной выше фразы? Каждая строка кода, каждая функция, каждый класс, каждый цикл и оператор условного перехода — всё это должно быть очень и очень простым. Оно должно читаться как стихотворение, просто без рифмы. Имена переменных должны способствовать этому. Всё должно способствовать этому. Код должен легко читаться, без комментариев. Комментируйте только не очевидные приемы и константы. Когда пишите код от которого будут наследоваться, его максимально подробно комментируйте.

Структура программы.

Хорошая программа состоит из:

одного главного класса программы (обычно это ядро программы);

– набора менеджеров и брокеров, которыми управляет ядро;

– набора сущностей для хранения данных в удобном для обработки виде;

– набора сущностей из предметной области (зависит от решаемой программой задачи), которыми управляют менеджеры и брокеры;

– набора системных сущностей (для работы с файлами, звуками, сетью и т.д.), которыми управляет, непосредственно или через брокеры, ядро;

– набора внешних библиотек (если требуется), которыми управляет использующая библиотеку сущность или брокер этой сущности;

– набора графических интерфейсов (если требуются), которые взаимодействуют с ядром и никогда не обрабатывают никаких данных, а лишь принимают и отображают их;

– интерфейса для работы с оборудованием (если этого требует решаемая задача);

– интерфейса для работы с базами данных (если требуется);

– журнала сообщений;

– сторожевого таймера (если программа должна быть надежной).

Названия переменных.

Переменные называйте так:

– Закрытые и защищенные члены класса начинаются с маленькой буквы m и подчеркивания, за которым следует первое слово названия с маленькой буквы, а остальные названия в стиле CamelCase. Например: intm_usersCount, int m_width, int m_tempDataDir. Открытых переменных у класса не должно быть. Если в них есть необходимость — что-то спроектировано не так.

– Названия параметров функций пишите с большой буквы в стиле CamelCase. Например: OpenFile(const std::string FileName);

– Названия локальных переменных начинайте с маленькой буквы и продолжайте в стиле CamelCase. Например: int maxPortNumber;

– Названия глобальных переменных начинайте с буквы g с последующим знаком подчеркивания и дальше первое слово с маленькой буквы, а остальные в стиле CamelCase. Например: GameCore g_gameCore; Кстати, если у Вас в коде есть глобальные переменные — повод задуматься.

– Названия констант членов класса начинайте с символа k и подчеркивания и дальше как в обычных закрытых членах класса. Например, intk_maxFileSize. Все остальные константы (за пределами класса) лучше выносите в отдельный заголовочный файл (constants.h, например) и старайтесь определять их (константы) с помощью директивы препроцессора. При этом названия переменных пишите полностью большими буквами через подчеркивание. Например:

– #define MAX_FILE_NAME_LEN 32

Практическая часть

Соблюдайте единый Code style. Если программист приходит работать в организацию, особенно крупную, то чаще всего его знакомят с правилами оформления кода в конкретном проекте (соглашение по code style). Это не случайный каприз работодателя, а свидетельство серьёзного подхода.

Всегда используйте IDE для программирования. Ушли те времена, когда необходимо было составлять программы в блокнотах и потом вручную собирать написанный код в объектный файл. Друзья, в мире существует очень много классных программ, которые подсвечивают, код, форматирую его и автоматически его собирают в готовую программу. Кроме всего этого, IDE информирует об возможных ошибках, еще до этапа сборки программы, подсвечивают неиспользуемые переменные, а также легко настраиваются под индивидуальные требования программистов. IDE даже может сама писать код, как это удобно реализовано в MVS, например, мы объявляем оператор switch, IDE уже знает об этом и делает это сама, нам же остается только добавить нужные переменные и константы. Все эти полезности реализованы с одной единственной целью — сделать процесс составления программ намного приятнее и намного быстрее. Так пользуйтесь же всем этим!

Пользуйтесь системами контроля версий. Если вы работаете над программой долгое время, около недели или больше, не поленитесь создать репозиторий для вашей программы. Зачем это нужно? Когда разработка программы сильно затянулась, то вы уже не всегда сможете вспомнить что и когда добавляли в программный код, откуда взялась та или иная ошибка. Система контроля версий поможет вам ответить на все эти вопросы. В конце концов, если вдруг вы случайно что-то поломаете в программе, система контроля версий всегда сможет восстановить предыдущую, рабочую версию программы. Не поленитесь в этом разобраться, система контроля версий — это очень классная штука.

Размещение фигурных скобок у функций:

```
int myFunction()
{
}
```

Нужно чтобы открывающаяся и закрывающаяся скобки находились в отдельных строках, по своему опыту могу сказать, так четко просматривается область видимости функции, её тело и заголовок.

Отступы должны быть табами. Вместо пробелов, отступы лучше проставлять табами, это намного удобнее, в том смысле, что экономит время. Ведь проще нажать один раз — таб, чем несколько раз пробел. Некоторые говорят, что табами проставлять отступы — это не правильно, так как в разных редакторах длинна таба меняется. Соответственно, и отформатированный код будет выглядеть не одинаково. Возможно они и правы, но я не встречал такого, хотя я программировал на многих IDE, табы почти везде одинаковые.

Имена функций должны быть в верблюжьем регистре. Следует давать имена функциям в верблюжьем регистре, причем начинаться имя должно всегда с маленькой буквы:

```
// Правильно:
myFunction()
searchInFile()
// Неправильно:
SearchInFile()
search_in_file()
myfunction()
Myfunction()
MYFUNCTION()
```

от еще несколько полезных статей: правила именования переменных и соглашения об именовании. Обязательно прочитайте их, в дальнейшем, вам очень помогут эти знания.

Оператор if-else.

```
if () {
  // тут код
}
else {
  // и тут код
}
```

Ну во первых, всегда ставьте фигурные скобки, даже если в операторе if нужно выполнить всего одну строку кода. Во вторых, переносите оператор else, на новую строку, хотя, на первый взгляд это может показаться и не удобно. В чем плюс? А плюс в том, что если вдруг вам нужно будет закомментировать или удалить блок кода с оператором else, то вам это будет сделать очень просто. И поверьте, такие ситуации встречаются очень часто. Вот запись, как делать не следует:

```
if () {
  // тут код
} else {
  // и тут код
}
```

Оператор else стоит в строке с закрывающейся скобкой тела оператора if. Такая запись кажется намного компактнее и красивее. Но как я уже говорил, когда вам понадобится закомментировать блок кода else, вы случайно закомментируете или удалите и закрывающую скобку предыдущего блока. В результате получите ошибку, которая отберет у вас драгоценное время. Такая запись, по моему мнению, также считается не правильной:

```
if ()
{
  // тут код
}
else
{
  // и тут код
}
```

Она получается излишне большой.

Используйте оператор const как можно чаще. Это дает огромное преимущество в отлаживании работы программы. Например, вы написали программу, но она работает не правильно, то есть в код прокралась ошибка, причем эта ошибка связана с изменением значения в переменной, в которой это значение меняться не должно. Вы начинаете просматривать весь код в поисках этой ошибки, на все это уйдет не мало времени. Если бы вы сразу объявили константу а не переменную, то ошибка б ы проявилась еще на этапе компиляции и вам бы не пришлось тратить время на её поиск. Рассмотрим пример:

```
#include <iostream>
using namespace std;
int main()
{
  const int august = 8;
  int currentMonth;
  cin >> currentMonth;
  if (august == currentMonth) {
    cout << "Current month - august";
  }
  return 0;
}
```

Задание

Создайте метод возврата файла и значения без использования функций try/catch, используя стандартам кодирования.

Контрольные вопросы

1. Какие типы стандартов кодирования существуют?
3. Объясните структуру кода?

Практическая работа №10

Тема: Необходимые навыки работать с библиотекой SafeStr.

Цель работы: Сформировать новики работать с библиотекой SafeStr.

Теоритическая часть

Библиотека C String Library (SafeStr) от Messier и Viega предоставляет богатую библиотеку обработки строк для языка C, которая имеет безопасную семантику и при этом легко взаимодействует с кодом устаревших библиотек [Messier 03].

Библиотека SafeStr использует динамический подход для C, который автоматически изменяет размеры строк по мере необходимости. SafeStr выполняет это, перераспределяя память и перемещая содержимое строки всякий раз, когда операция требует увеличения размера строки. В результате использование библиотеки не должно привести к переполнению буфера.

Библиотека SafeStr построена на основе типа safestr_t. Тип safestr_t совместим с char * и позволяет структурам safestr_t преобразовываться в char * и вести себя как строки в стиле C. Тип safestr_t хранит учетную информацию (например, фактическую и выделенную длину) в памяти, непосредственно предшествующей памяти, на которую ссылается указатель.

Таблица 1. Функции SafeStr API и эквиваленты для обычных строк C

Функция SafeStr	Функция C
safestr_append( )	strcat( )
safestr_nappend( )	strncat( )
safestr_find( )	strstr( )
safestr_copy( )	strcpy( )
safestr_ncopy( )	strncpy( )
safestr_compare( )	strcmp( )
safestr_ncompare( )	strncmp( )
safestr_length( )	strlen( )
safestr_sprintf( )	sprintf( )
safestr_vsprintf( )	vsprintf( )

Обычно вы можете создавать безопасные строки любым из следующих трех способов:

SAFESTR_ALLOC( )

Выделяет строку с изменяемым размером с начальным размером выделения в байтах, как указано ее единственным аргументом. Возвращаемая строка будет пустой строкой (фактический размер равен нулю). Обычно размер, выделенный для строки, будет больше фактического размера строки. Библиотека округляет выделение памяти в большую сторону, поэтому, если вы знаете, что вам понадобится большая строка, стоит заранее выделить ее с большим начальным размером выделения, чтобы избежать перераспределения по мере роста фактической длины строки.

SAFESTR_CREATE( )

Создает строку с изменяемым размером из обычной строки в стиле C, переданной в качестве единственного аргумента. Обычно это подходящий способ преобразования строки в стиле C в безопасную строку.

SAFESTR_TEMP( )

Создает временную строку с изменяемым размером из обычной строки в стиле C, переданной в качестве единственного аргумента. SAFESTR_CREATE() и SAFESTR_TEMP() ведут себя аналогично, за исключением того, что строка, созданная с помощью SAFESTR_TEMP(), будет автоматически уничтожена следующей функцией SafeStr, которая ее использует. Единственным исключением является функция safestr_reference(), которая увеличивает счетчик ссылок на строку, позволяя ей существовать до тех пор, пока не будет вызвана функция safestr_release() или safestr_free() для уменьшения счетчика ссылок на строку.

Люди иногда не понимают, когда на самом деле следует использовать SAFESTR_TEMP(), а также как правильно ее использовать. Используйте SAFESTR_TEMP(), когда вам нужно передать константную строку в качестве аргумента функции, которая ожидает safestr_t. Прекрасным примером такого случая может быть функция safestr_sprintf() со следующей сигнатурой:

int safestr_sprintf(safestr_t *output, safestr_t *fmt, ...);

Строка, задающая формат, должна быть безопасной строкой, но, поскольку вы всегда должны использовать константные строки для спецификации формата (см. рецепт 3.2), вы должны использовать SAFESTR_TEMP(). Альтернативой является использование SAFESTR_CREATE() для создания строки перед вызовом safestr_sprintf() и немедленного ее освобождения с помощью safestr_free().

интервал я = 42; safestr_t fmt, вывод; вывод = SAFESTR_ALLOC (1); /* Вместо этого: */ fmt = SAFESTR_CREATE("Значение i равно %d.\n"); safestr_sprintf(&output, fmt, i); safestr_free (FMT); /* Вы можете сделать это: */ safestr_sprintf(&output, SAFESTR_TEMP("Значение i равно %d.\n"), i);

При использовании временных строк помните, что временная строка будет автоматически уничтожена после вызова любой API-функции SafeStr, кроме safestr_reference(), которая увеличивает счетчик ссылок на строку. Если счетчик ссылок временной строки увеличивается, строка будет выдерживать любое количество вызовов API, пока ее счетчик ссылок не уменьшится до такой степени, что она будет уничтожена. Функции API safestr_release() и safestr_free() могут использоваться взаимозаменяемо для уменьшения счетчика ссылок на строку.

Например, если вы пишете функцию, которая принимает safestr_t в качестве аргумента (который может передаваться или не передаваться как временная строка), и вы будете выполнять несколько операций над строкой, вы должны увеличить счетчик ссылок строки перед операцией. его, и снова уменьшите его, когда вы закончите. Это гарантирует, что строка не будет преждевременно уничтожена, если в функцию будет передана временная строка.

void some_function (safestr_t * base, safestr_t extra) { safestr_reference (extra); if (safestr_length(*base) + safestr_length(extra) < 17) safestr_append(base, extra); safestr_release (дополнительно); }

В этом примере, если вы пропустили вызовы safestr_reference() и safestr_release() и если extra была временной строкой, вызов safestr_length() привел бы к уничтожению строки. В результате вызов safestr_append() будет работать с недопустимым safestr_t, если общая длина base и extra будет меньше 17.

Наконец, библиотека SafeStr также отслеживает достоверность строк. Строка может быть либо доверенной, либо ненадежной. Операции, которые объединяют строки, приводят к недоверенным строкам, если любая из строк, участвующих в комбинации, не является доверенной; в противном случае результату доверяют. В API SafeStr есть несколько мест, где проверяется достоверность строки, но функция safestr_istrusted() позволяет вам проверять строки самостоятельно.

Строки, полученные в результате использования SAFESTR_CREATE() или SAFESTR_TEMP(), не являются доверенными. Вы можете использовать SAFESTR_TEMP_TRUSTED() для создания доверенных временных строк. Достоверность существующей строки можно изменить с помощью функции safestr_trust(), чтобы сделать ее надежной, или safestr_untrust(), чтобы сделать ее ненадежной.

Основная причина отслеживать достоверность строки — отслеживать поток внешних входных данных. Безопасные строки, созданные из внешних данных, изначально не должны вызывать доверия. Если вы позже проверите содержимое строки и убедитесь, что она не содержит ничего опасного, вы можете пометить строку как доверенную. Всякий раз, когда вам нужно использовать строку для выполнения какой-либо потенциально опасной операции (например, использование строки в качестве аргумента командной строки для внешней программы), проверьте надежность строки перед ее использованием и выполните соответствующую ошибку, если строка ненадежный.

Библиотека SafeStr поддерживает неизменяемые строки. Строки могут быть указаны как неизменяемые во время инициализации или путем вызова недействительным safestr_makereadonly (safestr_t s);

Неизменяемые строки нельзя изменить с помощью SafeStr API. Тем не менее, память все еще может быть перезаписана. Библиотека предотвращает только запись, инициированную с помощью функций SafeStr.

API SafeStr может помочь отслеживать надежные и ненадежные данные в стиле режима taint в Perl. Разработчик может использовать этот механизм, чтобы пометить строки, происходящие из ненадежных источников, как таковые. Строки, которые были проверены на наличие потенциально вредоносного ввода, впоследствии могут быть помечены как надежные. При изменении строки доверенное свойство этой строки устанавливается на «ненадежный», если какой-либо из операндов не является доверенным. При

создании новой строки из операций над другими строками новая строка помечается как доверенная только в том случае, если все строки, влияющие на ее значение, являются доверенными.

Свойство доверия не будет распространяться должным образом, если API SafeStr будет обойден. API SafeStr в настоящее время не предоставляет подпрограмм, проверяющих флаг доверия. Однако вы можете самостоятельно проверить флаг.

```
int safer_system(safestr_t cmd) {
if (!safestr_istrusted(cmd)) {
printf("Untrusted data in safer_system!
");
abort();
}
return system((char *)cmd);
}
```

Задание

Каждый студент создает мини-проект с использованием библиотеки SafeString и объясняет преимущества проекта, предоставляемого библиотекой.

Контрольные вопросы

1. Какие разделы содержит библиотека SafeString?
2. Какие ошибки, возникшие в библиотеке строк Arduino, были исправлены в библиотеке SafeString?
3. Объясните использование библиотеки SafeString на примерах?

Практическая работа №11

Тема: Выполнить статическое тестирование с помощью программы Helix QAC.

Цель работы: Сформировать новики выполнить статическое тестирование с помощью программы Helix QAC.

Теоритическая часть

Статическое тестирование – тип тестирования, который предполагает, что программный код во время тестирования не будет выполняться. При этом само тестирование может быть как ручным, так и автоматизированным.

Статическое тестирование начинается на ранних этапах жизненного цикла ПО и является, соответственно, частью процесса верификации. Для этого типа тестирования в некоторых случаях даже не нужен компьютер – например, при проверке требований.

Большинство статических техник могут быть использованы для «тестирования» любых форм документации, включая вычитку кода, инспекцию проектной документации, функциональной спецификации и требований.

Даже статическое тестирование может быть автоматизировано – например, можно использовать автоматические средства проверки синтаксиса программного кода.

Виды статического тестирования:

– вычитка исходного кода программы;

– проверка требований.

Инструменты статического анализа кода

- *C/C++*
- *C#*
- Java

Инструменты статического анализа предназначены для выявления дефектов в исходном коде программ. Само название говорит, что принцип их работы основан на статическом анализе кода.

Существует огромное количество инструментов статического анализа, созданных для различных языков программирования.

C/C++

- **Cppcheck.** Очень популярный бесплатный открытый проект. Изначально целью было полное отсутствие ложных срабатываний, но сейчас целью является их максимальное малое количество. Анализатор является универсальным и предназначен для анализа кода, написанного на C/C ++ во встроенных проектах и других, использующих различные языковые расширения. Однако анализатор не поддерживает все конструкции, описанные новыми стандартами языка C++. Анализ потока данных по большей части чувствителен к потоку и двунаправлен. Теоретически вы можете более или менее проверить свой код напрямую без настройки, однако на практике настройка анализа будет необходима.

- **Clang Static Analyzer.** Статический анализатор кода для языков C/C++/Objective-C, встроенный в компилятор Clang. Впрочем, во многих других компиляторах также есть хорошие встроенные анализаторы кода. Например, Visual Studio включает в себя поддержку статического анализа проектов Visual C++ (флаг компилятора /analyze). Популярность Clang Static Analyzer вызвана открытостью его кода и возможностью написания собственных расширений.
- **Clang-Tidy** - отличается от диагностических возможностей Clang Static Analyzer тем, что Clang-Tidy - линтер, проверяющий соответствие кода стандартам кодирования.
- **Frama-C.** Анализатор программ на языке Си с открытым исходным кодом.
- **Lint.** Упоминается здесь, так как представляет историческую ценность. Этот инструмент можно назвать первым статическим анализатором кода для языка Си. Названия многих современных анализаторов образуются от слова "lint" (cpplint, PC-Lint, Splint, JSLint, Rpmlint, Puppet Lint, Pylint).
- **Parasoft C/C++test.** Широко известный и популярный анализатор кода. Имеется триал.
- **PC-Lint.** Весьма гибкий инструмент статического анализа. За гибкость анализа приходится платить большим количеством настроек в конфигурационных файлах. Можно попробовать в течение 30 дней для ознакомительных целей.
- **Helix QAC.** Статический анализатор для языков C, C++. Имеется возможность запросить пробную версию.

C#

- **ReSharper.** Не является статическим анализатором в классическом понимании, так как предоставляет мало сценариев использования. Плагин для Visual Studio, проводит статический анализ кода на языке C# и не только.
- **FxCop.** Бесплатный инструмент для статического анализа кода от компании Microsoft. Производит анализ байт-кода (CIL) на соответствие рекомендациям Microsoft по проектированию приложений. На данный момент проект мертв.
- **Roslyn Analyzers.** Набор статических анализаторов кода для языков C# и Visual Basic на основе .Net Compiler Platform ("Roslyn"). Производит анализ исходного кода, в отличие от FxCop. В составе этого проекта также был произведен порт наиболее важных правил FxCop.
- **Security Code Scan.** Статический анализатор кода на основе .Net Compiler Platform ("Roslyn") для языков C# и Visual Basic для поиска паттернов ошибок, связанных с безопасностью приложений: SQL Injection, Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), XML eXternal Entity Injection (XXE) и др. Производит анализ исходного кода
- **Roslynator.** Набор статических анализаторов кода для языка C# на основе .Net Compiler Platform ("Roslyn"). Производит анализ исходного кода.

- **CodeRush.** Плагин для Visual Studio. Продукт коммерческий, но имеется пробная версия.
- **Parasoft dotTEST.** Набор инструментов для тестирования приложений .NET, включающий в себя статический анализатор кода. Работает как плагин для Visual Studio. Как и в предыдущем случае, продукт коммерческий, имеется пробная версия.

Java

- **FindBugs.** Наиболее известный бесплатный статический анализатор Java кода. Анализирует байт-код программы. На данный момент проект мертв.
- **SpotBugs.** Стал преемником FindBugs. Проект с открытым исходным кодом.
- **IntelliJ IDEA.** Среда разработки от компании JetBrains, содержащая набор инспекций кода, которые позволяют найти, подсветить и исправить аномалии в коде.
- **SonarJava.** Статический анализатор кода для языка Java для поиска "запахов" кода, ошибок и уязвимостей, разрабатываемый компанией SonarSource.

Helix QAC — это статический анализатор кода, который автоматически сканирует код на наличие нарушений (на основе правил кодирования C и C++). Это позволяет командам разработчиков обнаруживать дефекты на более ранних этапах разработки, когда их легче и дешевле исправить. А Helix QAC автоматически применяет стандарты кодирования, такие как MISRA®, что гарантирует соответствие вашего кода.

Соответствовать стандартам

Соблюдайте стандарты кодирования и отраслевые стандарты. Helix QAC может автоматически проверять ваш код. И это гарантирует, что ваш код соответствует выбранному вами стандарту кодирования. Кроме того, отчет о соответствии поможет вам визуализировать, какие части вашего кода требуют наибольшего внимания. Helix QAC также включает модули соответствия для нескольких стандартов кодирования C и C++. Кроме того, его можно настроить для поддержки пользовательских стандартов кодирования.

Улучшить качество кода

Улучшите общее качество и безопасность любого приложения. Helix QAC выявляет дефекты, которые необходимо исправить, и предоставляет подробные рекомендации, помогающие разработчикам устранять проблемы в исходном коде. Он делает это без запуска программ. Поскольку разработчики получают мгновенную контекстную обратную связь, они учатся на ошибках. И в следующий раз, когда они создадут новый код (или пересмотрят существующий код), они станут лучше.

Ускорить разработку

Ускорьте время разработки за счет устранения узких мест. Helix QAC интегрируется с системами сборки и средами непрерывной интеграции. Это позволяет своевременно и часто выявлять дефекты. Таким образом, вы избежите ошибок, исправление которых на более позднем этапе разработки

обходится дороже. И это переориентирует и ускорит существующие проверки кода. Вы даже можете настроить Helix QAC для анализа только новых изменений и более быстрого предоставления обратной связи.

Выявляйте больше дефектов раньше

Выявляйте критические дефекты, о которых компиляторы не сообщают, на ранних стадиях разработки. Helix QAC строит точную модель поведения вашего программного обеспечения. И он отслеживает значения переменных в коде, какими они были бы во время выполнения. Таким образом, этот анализ максимизирует покрытие кода. И это сводит к минимуму ложные срабатывания и ложноотрицательные результаты. Он даже распознает проблемы, вызванные чрезмерно сложным кодом.

Совместная работа над проверками кода

Совместно проверяйте код с помощью панели инструментов Helix QAC. Вы сможете комментировать диагностику, найденную Helix QAC, и назначать действия пользователям.

Уверенно повторно используйте код

Повторно используйте код с уверенностью в его качестве. Helix QAC обнаруживает проблемы с переносимостью кода. Итак, вы будете

иметь возможность повторно использовать код, которому вы доверяете. И это поможет вам быстрее развиваться в будущем.

Качество кода Монитор

Следите за качеством кода с помощью информационной панели Helix QAC. Вы сможете отслеживать показатели качества кода из одного места. Таким образом, вы будете иметь представление о тенденциях с течением времени. И вы даже можете использовать панель инструментов для создания отчетов для наших заинтересованных сторон.

Масштабирование до миллионов строк кода

Масштабируйте статический анализ в соответствии с вашей средой. Helix QAC может обрабатывать миллионы строк кода. Это гарантирует безопасность вашего продукта, независимо от того, насколько сложна кодовая база.

Задание

Каждый студент должен выполнить статический тест проекта с помощью программы Helix QAC.

Контрольные вопросы

1. Характеристики Helix QAC?
2. Что такое индикатор?
3. Helix должен объяснить пошаговое статическое тестирование с использованием программного обеспечения QAC?

Практическая работа №12

Тема: Выполнить статическое тестирование с помощью программы SonarQube

Цель работы: Навыки правильного оформления кода и следования общепринятым единообразным принципам необходимо развивать на начальной стадии формирования компетенций программиста.

Теоритическая часть

"SonarQube — это платформа с открытым исходным кодом, разработанная SonarSource, для непрерывной оценки качества кода путем статического анализа".

Она не выполняет код, а лишь просматривает его. Причем, можете мне поверить, делает это очень тщательно. Неудивительно, что у SonarQube довольно внушительный список клиентов и репутация одного из востребованных инструментов статического анализа кода.

Завершив сканирование кода, о чем будет рассказано далее, SonarQube формирует отчет, который можно посмотреть в GUI через браузер. Все обнаруженные проблемы представляют собой "интерактивные тикеты", позволяющие писать к ним комментарии, делегировать их другим пользователям, открывать или закрывать и т. д.

К подробному описанию проблемы сразу же прилагается соответствующий код.

SonarQube также объясняет суть проблемы при нажатии на соответствующую ссылку "Why is this an issue?" .

SonarQube предоставляет систематизированный отчет о качестве кода, безопасности и общий Quality Gate Status, т. е. логическое значение, свидетельствующее о готовности приложения для отправки в продакшн. Если вы добавите так называемый Sonar-scan в конвейер, то отчет будет обновляться с каждым коммитом или пул реквестом. Кроме того, он поддерживает контроль версий, каждая из которых фиксирует конкретный коммит или слияние веток в проекте.

SonarQube предоставляет следующие возможности:

- Поддержка языков Java, C, C++, C#, Objective-C, Swift, PHP, JavaScript, Python и др.
- Предоставляет отчеты о дублировании кода, соблюдении стандартов кодирования, покрытия кода модульными тестами, возможные ошибки в коде, плотность комментариев в коде, технический долг и другое.
- Сохраняет историю метрик и строит графики изменения этих метрик во времени.
- Обеспечивает полностью автоматизированный анализ: интегрируется с Maven, Ant, Gradle и распространенными системами непрерывной интеграции.
- Позволяет интегрироваться с такими IDE, как Visual Studio, IntelliJ IDEA и Eclipse с помощью плагина SonarLint.

- Обеспечивает интеграцию с внешними инструментами: JIRA, Mantis, LDAP, Fortify и т.д.
- Можно расширять существующую функциональность с помощью сторонних плагинов.
- Реализует методологию SQALE для оценки технического долга.

Практическая часть

Установка SonarQube (открытая версия для сообщества)

Установка сервера SonarQube (приложение Java) не представляет сложности. Вы можете:

- скачать ZIP-файл и распаковать его локально или
- запустить SonarQube в Docker-контейнере.

Самый простой способ — запустить Docker-контейнер. Из документации вы узнаете, как скачать ZIP-файл и локально его распаковать.

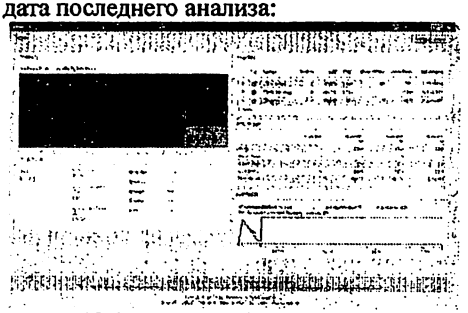
Перед выполнением следующей команды убедитесь, что Docker у вас уже локально установлен:

```
docker run -d \
--name sonarqube \
-e SONAR_ES_BOOTSTRAP_CHECKS_DISABLE=true \
-p 9000:9000 \
sonarqube:latest
```

Обратите внимание, что эта команда запустит SonarQube, но в момент удаления контейнера все данные будут потеряны. Можно выполнить ее ради эксперимента. Если же вы решите продолжить работу с SonarQube, советую закрыть этот контейнер и настроить тома данных. Имеются в виду каталоги, которые находятся в локальном компьютере и сохраняются в случае закрытия или обновления Docker-контейнера. Документация содержит пошаговую инструкцию к этому процессу.

Docker выполняет поставленную задачу, установка завершена, и инструмент готов к использованию. В браузере откройте GUI, перейдя по ссылке http://localhost:9000.

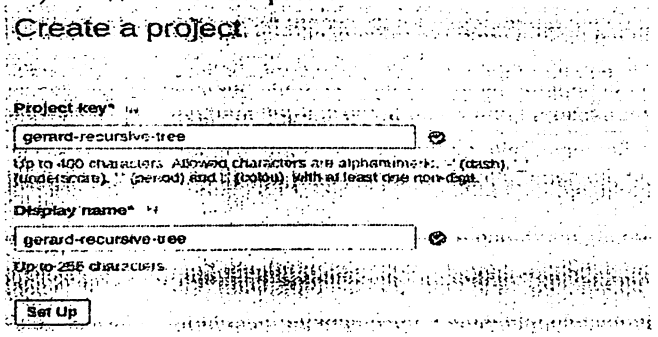
На главной странице SonarQube вы видите список проектов, добавленных в систему, с краткой статистикой по каждому проекту: версия сборки, количество строк кода, количество багов, уязвимостей и признаков "кода с душком", дата последнего анализа:



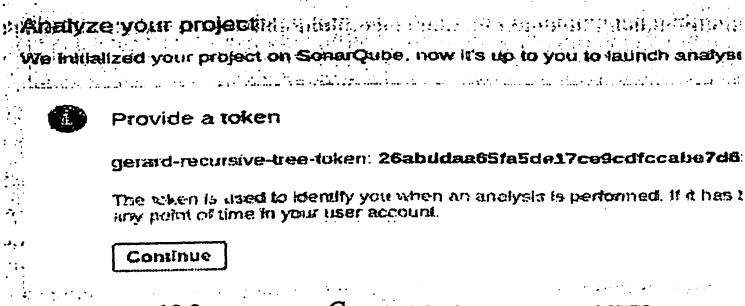
12.1 - рисунок. Главная чтраница.

Создание нового проекта

Авторизуйтесь с помощью предустановленных учетных данных (admin:admin) и создайте новый проект.



12.2 - рисунок. Именование проекта/



12.3 - рисунок. Создание токена для проекта.

Запомните ключ проекта и токен, поскольку они нам в дальнейшем понадобятся.

Первое сканирование проекта

На этом этапе просканируем код с помощью такого инструмента CLI, как sonar-scanner. Сделать это можно разными способами, подробное описание которых предоставлено на соответствующей странице документации.

Я предпочел скачать исполняемый файл и добавить его расположение в глобальную переменную PATH. Теперь команда sonar-scanner доступна из любой точки компьютера:

PATH=$PATH:/opt/sonar-scanner-4.5.0.2216-linux/bin

В корневом каталоге проекта создайте файл sonar-project.properties со следующим содержимым:

sonar.projectKey=gerard-recursive-tree

sonar.sources=components,pages,styles,typings

sonar.sourceEncoding=UTF-8

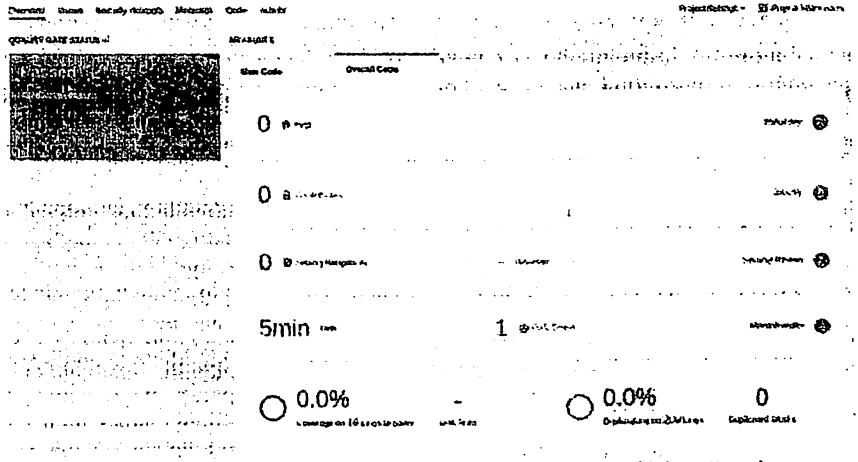
Первая строка кода содержит ключ, полученный при создании проекта SonarQube в GUI.

Настало время просканировать файлы. В корневом каталоге проекта выполните данную команду:

sonar-scanner -Dsonar.login=[PROJECT TOKEN]

Написание -Dsonar.login без пробела не является ошибкой. Значение токена должно совпадать с тем, что было получено при создании нового проекта.

Работа сканера займет несколько секунд, после чего в GUI в браузере вы сможете изучить отчет. Если проект содержит высоко качественный код, то результаты будут представлены в следующем виде:



12.4 – рисунок. Статус проверки качества ""passed" (пройден).

При наличии в коде ошибок, уязвимостей, проблемных зон в системе безопасности, запашков или дублированных строк отчет примет совершенно другой вид. Поскольку GUI прост и понятен, я не буду вдаваться в детали.

Задание

Установите и настройте SonarQube. Проведите в нем примерный анализ проекта. Редактировать результаты анализа.

Контрольные вопросы

1. Что такое SonarQube (Сонар)?
2. При качественной разработке программ SonarQube программисты анализируют на основе какие единиц качества?
3. Пошагово объясните, как использовать SonarQube?

Практическая работа №13

Тема: Тестирование безопасности системы с помощью программы OWASP ZAP

Цель работы: Использовать OWASP ZAP для обнаружения уязвимостей веб-приложений в конвейере CI/CD.

Теоритическая часть

Прокси-сервер OWASP Zed Attack Proxy (ZAP) является одним из самых популярных бесплатных инструментов безопасности в мире и активно поддерживается сотнями международных добровольцев. Это может помочь вам автоматически находить уязвимости безопасности в ваших веб-приложениях во время разработки и тестирования ваших приложений. Это также отличный инструмент для опытных тестировщиков пера, который можно использовать для ручного тестирования безопасности.

Проблема

Веб-приложения имеют базовую аутентификацию, Логины пользователей и проверку формы, которая останавливает сканер на полпути

Решение

Используйте тестовые сценарии Selenium для управления ZAP. Проект может уже включать в себя сценарии selenium для функционального тестирования. Активные проверки активно изменяют записанные запросы и ответы для определения дальнейших уязвимостей

Общее тестирование безопасности можно разделить на следующие типы:

- **Оценка уязвимости** - как правило , через сканирование формы инструмента для оценки безопасности системы
- **испытание на проницаемость** - система анализируется и форма имитации атаки, для оценки их безопасности и обороноспособности
- **Тест выполнения** - для оценки безопасности системы путем тестирования конечного пользователя
- **Обзор кода** - код системы должны быть оценены и проанализированы для определения безопасности системы и обороноспособность

Тест Проницаемость, тестеры играют роль внешних атак, цель состоит в том, чтобы сломать систему для того, чтобы достичь цели кражи данных или аналогичных атак отказа в обслуживании.

Тест Преимущества проницаемости редко вопросы ложной безопасности (другие виды испытаний, проблемы безопасности, представленные в большинстве случаев не представляют собой реальную опасность подвергнуться нападению).

Однако тестирование проницаемости также очень трудоемкие тесты.

Проницаемость тесты также используется для проверки механизмов системы обороны для проверки планов реагирования, а также определить, следовать политике безопасности.

Автоматизированное тестирование на проникновение является важной частью проверки непрерывной интеграции, он помогает найти новые риски в

области безопасности, но также может быстро найти безопасности появляются в быстрой итерации назад.

Проницаемость тесты обычно следуют следующим этапам:

• исследовать - научиться тестируемой системы. Научитесь элемент содержит всю систему , которая будет использоваться в системе, которая включает в себя терминал, программное обеспечение и системы , для которых установлены патчи и тому подобное.

Этот процесс тестеры будет искать скрытый контент в системе, известные угрозы безопасности, а также выполнение других уязвимостей.

• **Атака** - тестеры попытки использовать уязвимости известных или предполагаемых рисков для того , чтобы доказать существование этих рисков и уязвимостей.

• **Отчеты** - тестеры сообщать результаты испытаний, включая обнаружение уязвимости системы и как эксплуатировали уязвимости, степень сложности , чтобы использовать эти уязвимости.

Настройка CI/CD

Давайте создадим конвейер CI, который запустит ZAP в безголовом режиме, запустим наши функциональные тесты, которые будут выполнять два типа сканирования (активное/пассивное), сохранят результаты предупреждений о сканировании в отчетах HTML и отключат сервер.

Шаги CI/CD:

- Начать ЗАП
- Запуск скриптов Selenium (Пассивное сканирование)
- Дождитесь завершения пассивного сканирования
- Начать Активное Сканирование
- Дождитесь завершения активного сканирования
- Получать оповещения и сообщать

Zed Attack Proxy (ZAP) является свободным, открытым исходным кодом инструмент тестирования на проникновение, разработан и поддерживается организацией OWASP.

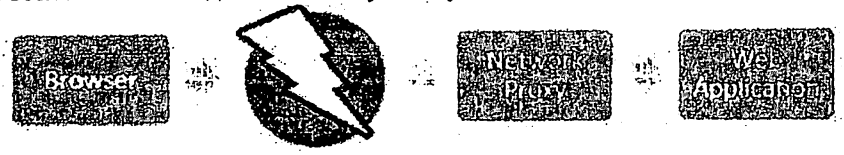
ZAP специально разработан для тестирования безопасности веб-приложений, и имеет очень сильную гибкость и масштабируемость.

ZAP ядро механики знакомы каждый «промежуточный прокси-сервер.» Он создал тестером между браузером и веб-приложений услуг, так что он может перехватывать и обзор обмена информацией между браузером и прикладных услуг.

В случае необходимости, ЗАП изменение может быть завершено, и пакет повторной передачи.

13.1 - рисунок.

Если уже существует сетевой прокси, ZAP также может быть выполнен с возможностью подключения к существующим агентам.



13.2 - рисунок.

ZAP предлагает широкий выбор различных уровней квалификации функций безопасности - эксперты тестирования безопасности от разработки, чтобы проверить новичок безопасности для.

ZAP поддерживает все основные операционные системы и Docker, не форсирует использование связывания ОС.

Установка и настройка

ZAP предусматривает установку пакета Windows, Linux и Mac OS / X.

Docker также зеркало, адрес загрузки можно найти ниже.

устанавливать

В первую очередь, установлен в системе в соответствии с системой проницаемости тест ZAP. Скачать по следующему адресу для вашей системы ZAP установочного пакета и выполнить:

https://www.zaproxy.org/download/

Примечание ZAP требуют более JDK8 версии для запуска, поэтому необходимо предварительно установить JDK8 или более поздней версии на вашей системе.

Docker версия не требует установки Java.

После завершения установки, вы можете начать ZAP.

Держите сессию

Первый раз, когда вы начинаете ZAP, будет предложено, если вы хотите сохранить ZAP сессию.

ZAP сессия будет по умолчанию заданного имени и адреса, записанное на диск в базе данных HSQLDB.

Если вы не хотите, чтобы сеанс, то при выходе из ZAP, эти сессии файлы будут удалены. (ZAP означает, что следующий запуск не будет видеть перед операцией / протоколов испытаний)

Если вы решили сохранить сеанс, информация сеанса будет сохранена в локальную базу данных, поэтому в следующий раз вы можете продолжать использовать предыдущую операцию.

Задание

Выявление уязвимостей веб-страниц к угрозам в соответствии с этой опцией.

Контрольные вопросы

1. Проверка безопасности программного обеспечения?
2. Что такое Zed Attack Proxy (ZAP)?

Практическая работа №14

Тема: Подготовка проектов для внедрения с помощью инструмента Inno Setup Compiler

Цель работы: Формироват навики подготовка проектов для внедрения с помощью инструмента Inno Setup Compiler

Теоритическая часть

В этой статье пойдет речь про Inno Setup — удобный инструмент для упаковки приложений для OS Windows.

Почему Inno Setup? "Сегодня Inno Setup конкурирует и даже превосходит многие коммерческие инсталляторы в наборе функций и стабильности", — говорится на официальном сайте разработчика. И действительно, этот инструмент довольно простой для освоения и мощный по своим возможностям.

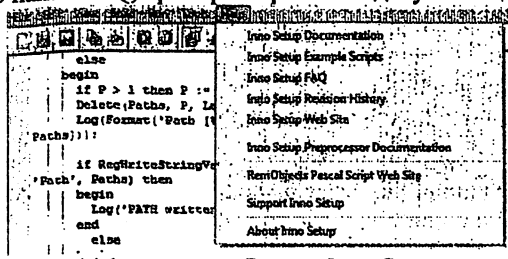
На данный момент разработчики Inno Setup в качестве ключевых возможностей заявляют:

- поддержку всех версий Windows, начиная с 2000;
- поддержку как 64-битной, так и 32-битной установки;
- встроенный обработчик скриптов Pascal для более гибкой настройки;
- возможность взять приложение и все файлы, которые нужны для его работы, и упаковать их в один компактный .exe файл;
- Inno Setup поддерживает несколько режимов сжатия данных. При необходимости встроенное сжатие можно отключать и использовать сторонние архиваторы;
- позволяет при установке добавлять ключи реестра, запускать дополнительные приложения во время, до и после установки;
- возможность удалять ключи реестра и все следы активности программы, гибко настраивать деинсталлятор.

Практическая часть

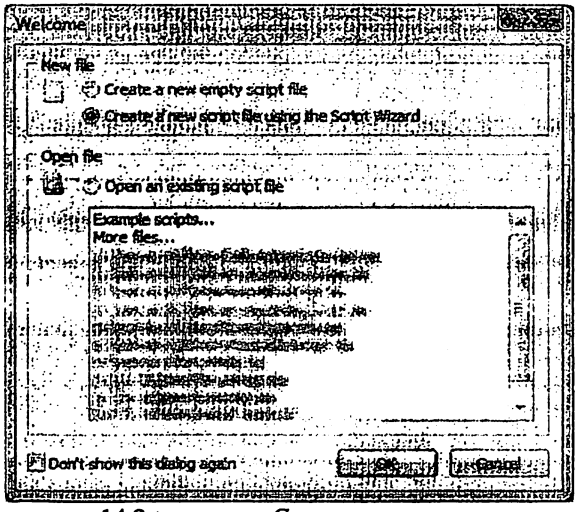
Скачать бесплатно стабильную версию Inno Setup Compiler можно с официального сайта. Там же можно найти FAQ и документацию на английском языке и сторонние библиотеки, упрощающие процесс создания Inno Setup Script (.iss).

В классическом приложении Inno Setup Compiler есть раздел Help, в котором можно найти FAQ, примеры скриптов и документацию.



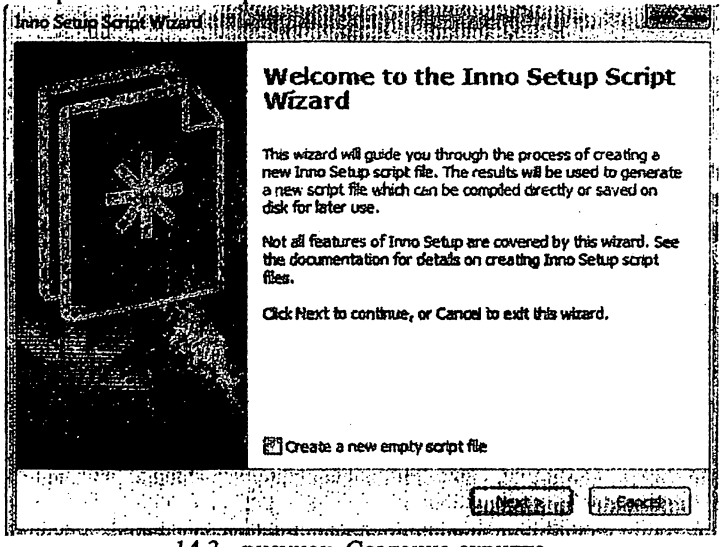
14.1 – рисунок. Запуск Inno Setup.

При первом запуске программы пользователя встречает диалог, который предлагает посмотреть примеры, создать свой первый Inno Setup Script (.iss) с помощью мастера создания скриптов или открыть уже существующий .iss скрипт.



14.2 - рисунок. Создание скрипта.

Для создания установочного файла можно воспользоваться мастером создания скриптов Inno Setup.



14.3 - рисунок. Создание скрипта.

Мастер предложит заполнить:

- сведения о приложении,
- предпочтительные пути установки,
- файлы, из которых будет состоять конечный дистрибутив,
- нужны ли ярлыки, и если да, то где,
- лицензионное соглашение,
- документы с инструкциями,
- имя и место для итогового дистрибутива,
- пароль, если он требуется,
- выбрать языки, которые должен поддерживать инсталлятор.

После того, как мастер соберет все необходимые сведения, он предложит скомпилировать полученный скрипт. В папке, которая была выбрана для итогового дистрибутива, будет лежать готовый к установке файл.

Расширенная настройка

Если есть необходимость кастомизировать интерфейс инсталлятора, поработать с реестром или настроить действия в процессе удаления программы, то скрипт инсталлятора придется доработать.

Скрипт Inno Setup разделен на секции, каждая из которых отвечает за свой функционал. При этом бывает так, что для одной и той же цели могут подойти разные секции. Порядок выполнения секций строго определен и от расположения секций в скрипте не зависит.

Все секции скрипта, за исключением [Setup], [Messages], [CustomMessages], [LangOptions] и [Code], содержат строки, разделенные на параметры. Каждый параметр состоит из имени, за которым следует двоеточие, а затем значение. Если не указано иное, параметры являются необязательными, поскольку они принимают значение по умолчанию, если их не указать. Несколько параметров в строке разделяются точкой с запятой и могут быть перечислены в любом порядке.

```
Compression=lzma
SolidCompression=yes

[Languages]
Name: "english"; MessagesFile: "compiler:Default.isl"

[Tasks]
Name: "desktopicon"; Description: "{cm:CreateDesktopIcon}";
GroupDescription: "{cm:AdditionalIcons}"; Flags: unchecked

[Files]
Source: "C:\Program Files (x86)\Inno Setup 5\Examples\MyProg.exe";
 DestDir: "{app}"; Flags: ignoreversion
; NOTE: Don't use "Flags: ignoreversion" on any shared system files

[Icons]
Name: "{commonprograms}\{#MyAppName}"; Filename: "{app}\{#
MyAppExeName}"
Name: "{commondesktop}\{#MyAppName}"; Filename: "{app}\{#
MyAppExeName}"; Tasks: desktopicon

[Run]
Filename: "{app}\{#MyAppExeName}"; Description: "
{cm:LaunchProgram,{#StringChange(MyAppName, '&', '&&')}}"; Flags:
nowait postinstall skipifsilent
```

Пример скрипта Inno Setup

В скрипте поддерживаются комментарии, для это нужно начать строку с точки с запятой (“;”).

Описание и примеры секций скрипта

В самом начале скрипта, до всех секций, определяются необходимые константы:

```
#define   MyAppName      "Example"
#define   MyAppVersion   "0.0.1"
#define   MyAppPublisher "example developer"
#define   URL            "http://www.example.com"
#define   MyAppURL       "example.exe"
```

В данном случае MyAppName — это имя приложения, MyAppVersion — версия приложения, MyAppPublisher — разработчик. Эти константы будут доступны в любой части скрипта и при необходимости что-то изменить, достаточно сделать это один раз в одном месте.

[Setup]

В данном разделе содержатся глобальные параметры, используемые установщиком и деинсталлятором.

```
AppId={{9E594ED6-EBDC-4D98-990E-F3F6B9AB9ACE}
AppName={#MyAppName}
AppVersion={#MyAppVersion}
AppPublisher={#MyAppPublisher}
AppPublisherURL={#MyAppURL}
AppSupportURL={#MyAppURL}
```

Задание

Создадите установочный файл используя программа Inno Setup.

Контрольные вопросы

1. Основной функция программа Inno Setup?
2. Какие проблемы решают программа Inno Setup?

СОДЕРЖАНИЕ

Методическое пособие к практическим работам
по предмету "Протоколы безопасное связи"
для студентов бакалавриата направления образования
5330300 – "Информационная безопасность"

Рассмотрено на заседании
кафедры "Криптология" и рекомендовано
научно-методический совету факультета
202__ год ____ _____
протокол № ____

Рассмотрено на заседании
факультета "КБ" и рекомендовано
научно-методический совету университета
202__ год ____ _____
протокол № ____

Рассмотрено и рекомендовано к печати
на заседании учебно-методического совета
ТУИТ им.Мухаммада ал-Хоразмий
202__ год ____ _____
протокол № ____

Составители: D.Sc. Д.Я. Иргашева
PhD. З.Т. Худойкулов
О.О. Турсунов
И.С. Олимов

Рецензенты: к.т.н. О.П. Ахмедова
PhD. О.Ё. Зокиров

Главный редактор: _____

Формат 60x84 1/16. Печ.лист 6.
Заказ № 96. Тираж 15.
Отпечатано в «Редакционно издательском»
отделе при ТУИТ.
Ташкент ул. Амир Темур, 108.