Министерство образования Республики Беларусь
Учреждение образования «Витебский государственный университет имени П.М. Машерова»
Кафедра информатики и информационных технологий

# ТЕСТИРОВАНИЕ, ВЕРИФИКАЦИЯ И АТТЕСТАЦИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

***Методические рекомендации***



*Витебск*
*ВГУ имени П.М. Машерова*
*2020*

УДК 004.415.2:378.147(075.8)
ББК 32.972.11р30я73
Т36

Печатается по решению научно-методического совета учреждения образования «Витебский государственный университет имени П.М. Машерова». Протокол № 3 от 30.12.2019.

Составитель: старший преподаватель кафедры информатики и информационных технологий ВГУ имени П.М. Машерова **В.В. Шедько**

Р е ц е н з е н т :
заведующий кафедрой прикладного и системного программирования
ВГУ имени П.М. Машерова,
кандидат физико-математических наук *С.А. Ермоченко*

**Т36 Тестирование, верификация и аттестация программного обеспечения :** методические рекомендации / сост. В.В. Шедько. – Витебск : ВГУ имени П.М. Машерова, 2020. – 31 с.

Методические рекомендации содержат материал по предмету «Верификация и аттестация программного обеспечения», вопросы и индивидуальные задания для лабораторных занятий и самостоятельной работы, краткие теоретические сведения для успешного их выполнения.

Учебное издание предназначено для студентов второй ступени дневной и заочной форм обучения по дисциплине «Верификация и аттестация программного обеспечения» специальностей «Информатика и технологии программирования» и «Управление информационными ресурсами», а также может быть использовано обучающимися первой ступени по дисциплине «Тестирование и оценка качества программного обеспечения» специальности «Прикладная информатика (по направлениям)».

**УДК 004.415.2:378.147(075.8)**
**ББК 32.972.11р30я73**

**СОДЕРЖАНИЕ**

# ЛАБОРАТОРНАЯ РАБОТА № 1
# СТАНДАРТИЗАЦИЯ ОЦЕНКИ КАЧЕСТВА ПО

***Цель и задачи работы: Получить навыки практического использования стандартов для оценки качества программных продуктов.***

### *Теоретический материал*

Обеспечение качества при разработке программного обеспечения предполагает определение или выбор стандартов, применяемых либо к процессу разработки ПО, либо к готовому продукту. Эти стандарты могут быть частью процессов производства ПО.

В процессе обеспечения качества применяются два вида стандартов:

1. Стандарты на продукцию. Применимы к уже готовым программным продуктам. Они включают стандарты на сопроводительную документацию, например структуру документа, описывающего системные требования, а также такие стандарты, как, например, стандарт заголовка комментариев в определении класса объекта, стандарты написания программного кода, определяющие способ использования языка программирования.

2. Стандарты на процесс создания ПО. Определяют ход самого процесса создания программного продукта, например разработку спецификации, процессы проектирования и аттестации. Кроме того, они могут описывать документацию, создаваемую в ходе выполнения этих процессов.

Между этими стандартами существует очень сильная взаимосвязь. Стандарты на продукцию применимы к результату процесса разработки ПО, а стандарты на процесс в большинстве случаев подразумевают выполнение определенных действий, направленных на получение товара, соответствующего стандартам на продукцию.

Стандарты аккумулируют все лучшее из практической деятельности по созданию ПО, собирают знания и опыт, имеющие значение для организации-разработчика. Практические знания приобретаются путем долгого поиска и ошибок. Привнесение этого опыта в определенный стандарт помогает избежать повторения прошлых ошибок. Стандарты предоставляют необходимую основу для реализации процесса обеспечения качества. Имея в наличии стандарты, для обеспечения качества достаточно контролировать, чтобы они выполнялись в процессе создания ПО. Стандарты незаменимы, когда работа переходит от одного сотрудника к другому. В этом случае деятельность всех специалистов в организации подчиняется единому нормативу. Следовательно, требуется меньше затрат на изучение сотрудником новой работы.

Создание стандартов по разработке ПО – процесс долгий и утомительный. Группы обеспечения качества, которые занимаются составлением стандартов, обычно основывают нормативы организации на общих национальных и международных стандартах. Используя их в качестве отправного пункта,

группа обеспечения качества разрабатывает свой "справочник" по стандартам. В нем содержатся стандарты, отражающие специфику деятельности данной организации. Иногда специалисты по разработке ПО относятся к стандартам как к бюрократическому наследию, неприменимому к разработке ПО. Особенно это проявляется при выполнении такой утомительной и скучной процедуры, как заполнение всевозможных форм и регистрация работ.

Менеджеры по качеству, отвечающие за разработку стандартов, должны быть достаточно подготовленными и вовлекать самих программистов в разработку стандартов. Они должны ясно понимать, с какой целью разрабатывается стандарт, и четко следовать установленным правилам и нормативам. Важно, чтобы документ с описанием стандарта включал не только изложение самого норматива качества, но и объяснение необходимости именно этого норматива. Регулярно просматривать и обновлять стандарты, чтобы идти в ногу с быстро развивающимися технологиями. Как только стандарт разработан, его помещают в справочник организации по стандартам, где его холят и лелеют, меняя с большой неохотой. Справочник по стандартам – вещь для организации необходимая, однако он должен развиваться по мере развития новых технологий. Обеспечить поддержку стандартов программными средствами везде, где только можно.

Стандарты на процессы разработки ПО также могут вызвать ряд проблем, если ставят перед командой разработчиков практически неосуществимые задачи. Такие стандарты дают руководящие советы по выполнению работы, при этом менеджеры проектов могут интерпретировать их каждый по-своему. Нет смысла указывать определенное направление работы, если оно неприменимо к данному проекту или к самой команде разработчиков. Поэтому менеджер проекта должен иметь право изменять стандарты процесса создания ПО в соответствии со специфическими условиями именно данного проекта. Здесь следует оговориться, что это утверждение не относится к стандартам на качество готовой продукции и на процесс сопровождения программной системы, которые могут быть изменены только после глубокого изучения данного вопроса.

Менеджер проекта и менеджер по управлению качеством могут легко избежать подводных камней, связанных с "неподходящими" стандартами, путем тщательной разработки плана мероприятий по обеспечению качества. Именно они должны решить, какие стандарты из справочника можно использовать без изменений, какие из них подлежат изменению, а какие следует исключить. Иногда возникает необходимость в разработке нового стандарта, что может быть вызвано условиями выполнения определенного проекта. Например, требуется установить стандарт для формальной спецификации, если прежде в проектах он не использовался. Такие стандарты должны разрабатываться в процессе выполнения проекта.

Необходимость стандартов на документацию в программном проекте становится очевидной, если не существует никакого другого реального

способа отображения процесса разработки ПО. Стандартные документы имеют четкую последовательную структуру, вид и качество, а значит, их легко читать и воспринимать.

Существует три типа стандартов на документацию.

1. Стандарты на процесс создания документации. Определяют способ создания технической документации.

2. Стандарты на документ. Определяют структуру и внешний вид документов.

3. Стандарты на обмен документами. Гарантируют совместимость всех электронных версий документов.

***Задание***

1) Ознакомитесь с теоретическим материалом и ответьте на контрольные вопросы.

2) Изучите структуру оценки качества ПО в международном стандарте *ISO 9126:1991,* характеристики и субхарактеристики качества.

3) Составьте перечень требований для качественной оценки информационно – обучающей программы по стандарту *ISO 9126:1991.*

4) Для перечня требований из п.3 введите нормализацию для количественной оценки показателей качества для информационно – обучающей программы, обоснуйте свой выбор.

5) Объединив результаты п.3 и 4, составьте спецификацию качества для информационно – обучающей программы.

***Задания для самостоятельной работы***

1. Дополните спецификацию из п.5 общими требованиями к ПО, которые присутствуют в стандартах, но нет в *ISO 9126:199.*
2. Найдите и ознакомьтесь со стандартами на процессы создания ПО и оценки качества готового ПО в РБ.
3. Оцените качество конкретно взятого программного продукта (например, сайта) с точки зрения стандартов качества готового ПО в РБ

***Контрольные вопросы***

1. Для чего нужна стандартизация оценки качества ПО.
2. Назовите стандарты регламентирующие оценку качества ПО.
3. Перечислите основные характеристики качества ПО в стандарте *ISO 9126:1991*.
4. Укажите основные субхарактеристики для оценки надежности программного обеспечения в стандарте *ISO 9126:1991*.
5. Что такое нормализация требований качества ПО, и для чего она нужна?
6. Укажите особенности использования характеристик надежности в спецификациях конкретного программного продукта.
7. Расскажите о современных технологиях повышения качества программного обеспечения.

# ЛАБОРАТОРНАЯ РАБОТА № 2
# МОДЕЛИ КАЧЕСТВА И НАДЕЖНОСТИ ПО

***Цель и задачи работы: Получить навыки практического использования численной оценки надежности ПО и сложных программных комплексов.***

### *Теоретический материал*

*Надежность программного обеспечения* – это свойство сохранять заданные характеристики при определенных условиях эксплуатации. Надежность ПО определяется его безотказностью и восстанавливаемостью. *Безотказность* ПО – его свойство сохранять работоспособность в процессе обработки информации, которую можно определить вероятностью работы без отказов при определенных условиях внешней среды в течение заданного периода наблюдения.

Отказ программы – это недопустимое отклонение характеристик процесса функционирования программы от требуемых. Определенные условия внешней среды – это совокупность входных данных и состояния вычислительной системы. Заданный период наблюдения обычно соответствует необходимому числу прогонов программы для решения задачи.

Безотказность ПО можно охарактеризовать также *средним временем между двумя отказами* в процессе выполнения программы $T$ (при условии, что сбой аппаратных средств отсутствует). С точки зрения надежности принципиальное отличие программных средств от аппаратных состоит в том, что программы не изнашиваются и не подвержены физическому старению в процессе работы. Безотказность ПО определяется его корректностью, а значит целиком зависит от наличия в нем ошибок, внесенных на этапе создания и хранения.

*Восстанавливаемость* программы может быть оценена сравнительной продолжительностью устранения ошибки в программе и восстановления ее работоспособности. Восстановление после отказа может заключаться в корректировке текста программы, исправлений данных, внесения изменений в организацию вычислительного процесса. Восстанавливаемость зависит от сложности структуры комплекса программ, от алгоритмического языка, от качества документации и т.д. Можно также говорить об *устойчивости* ПО, понимая под этим способность ограничивать последствия собственных ошибок и противостоять неблагоприятным условиям внешней среды.

В зависимости от степени серьезности последствий ошибок в программе, отклонения выполнения программой заданных функций можно разделить следующим образом: полное прекращение выполнения функций на длительное или неопределенное время или кратковременное прекращение хода вычислительного процесса.

Эргономичность – в изначальном смысле это эффективность инструмента производства или системы в эргономике. Эргономичность как характеристика программного продукта обозначает степень, с которой программа позволяет минимизировать усилия пользователя по подготовке исходных данных, обработке данных и оценке полученных результатов. Чем меньше движений совершает пользователь мышью, чем меньше информации вводит он с клавиатуры и чем быстрее он находит требуемую информацию – тем выше степень эргономичности.

Модели надежности ПО позволяют исследовать закономерности появления ошибок ПО, а также прогнозировать надежность эксплуатации ПО. По различным признакам модели делятся на: статические и динамические, дискретные и непрерывные, эмпирические и аналитические.

Простая интуитивная модель предполагает проведение тестирования двумя группами программистов (или двумя программистами в зависимости от величины программы) независимо друг от друга, использующими независимые тестовые наборы. В процессе тестирования каждая из групп фиксирует все найденные ею ошибки. При оценке числа оставшихся в программе ошибок результаты тестирования обеих групп собираются и сравниваются.

Аналитические модели строятся в предположении, что появление ошибок является случайным событием и имеет вероятностный характер. Модель надежности программ с дискретным увеличением времени наработки на отказ строится на гипотезе, что устранение ошибки приводит к увеличению времени наработки на отказ на некоторую случайную величину. Экспоненциальная модель надежности ПО основана на предположении об экспоненциальном характере изменения числа ошибок во времени. В этой модели прогнозируется надежность программы на основе данных, полученных во время тестирования.

Для определения надежности больших программных комплексов используются *марковские модели*. В марковском процессе выбор следующего модуля зависит только от модуля, выполняемого в данный момент и не зависит от предыстории. Структуру управления программой по марковской модели можно представить в виде направленного графа.

***Задание***

1) Ознакомитесь с общими теоретическими положениями и ответьте на контрольные вопросы. См. список рекомендуемой литературы или Методические рекомендации к выполнению лабораторных работ, расположенные в ИС разделе.

2) Подготовить краткую характеристику модели надежности, согласно варианту.

| Вариант | Модель | Вариант | Модель |
|---|---|---|---|
| 1. | Нельсона | 7. | Джелинского – Моранды |
| 2. | Коркорэна | 8. | LaPadula |
| 3. | Липова | 9. | Шумана |
| 4. | Миллса | 10. | переходных вероятностей |
| 5. | Муса | 11. | Эмпирические |
| 6. | Шика – Волвертона | | |

3) Реализуйте в виде программного продукта (информационно – обучающей программы) выполнение задания из п.2. - среду реализации выберите самостоятельно.

4) Проведите необходимые организационные мероприятия и реализуйте вычисление надежности, используя простую интуитивную модель, для программы из пункта 3.

***Контрольные вопросы***

1. Перечислите основные характеристики надежности ПО.
2. Укажите основные классы скрытых ошибок программного обеспечения.
3. Существует ли модель, позволяющая точно определить количество обнаруженных ошибок ПО на определенном промежутке времени?
4. Укажите особенности использования модели надежности с дискретным увеличением времени наработки на отказ и экспоненциальной модели.

# ЛАБОРАТОРНАЯ РАБОТА № 3
# МЕТОДЫ И СПОСОБЫ ПОВЫШЕНИЯ НАДЕЖНОСТИ И КАЧЕСТВА ПО

***Цель и задачи работы: Изучить пути повышения качества и надежности ПО и сложных программных комплексов.***

### *Теоретический материал*

Опыт создания и применения сложных информационных систем (ИС) в последние десятилетия выявил множество ситуаций, при которых сбои и отказы их функционирования были обусловлены дефектами комплексов программ, что приводило к большому экономическому ущербу. Вследствие ошибок в программах автоматического управления погибло несколько отечественных, американских и французских спутников, происходили отказы и катастрофы в сложных административных, банковских и технологических информационных системах.

Обеспечение надежности должно реализовываться специалистами в жизненном цикле программных средств **на основе использования** современной методологии, технологического инструментария, стандартов и нормативных документов.

Для обеспечения надежности программных средств необходимы **разработка и применение эффективных методов и средств**, предупреждающих и выявляющих дефекты, а также удостоверяющих надежность программ и оперативно защищающих функционирование ПС при их проявлениях. Для систематической, координированной борьбы с угрозами надежности **должны проводиться исследования** конкретных факторов, влияющих на качество функционирования и безопасность применения программ со стороны реально существующих и потенциально возможных дефектов в создаваемых комплексах программ. В каждом проекте **должен целенаправленно разрабатываться скоординированный комплекс** методов и средств обеспечения заданной надежности функционирования ПС при реально достижимом снижении уровня дефектов и ошибок разработки. Учет факторов, влияющих на затраты ресурсов при создании конкретного ПС, должен позволять рационализировать их использование и добиваться заданной надежности функционирования ПС при минимальных или допустимых затратах.

Для обеспечения надежности программных средств в конкретных проектах **должны быть организованы и стимулированы разработка, освоение и применение современных автоматизированных технологий и инструментальных средств**, обеспечивающих *предупреждение или исключение* большинства видов дефектов и ошибок при создании и модификации ПС и их компонентов.

**Сложность** – это одна из главных причин ненадежности программного обеспечения. Сложность почти не поддается ни точному определению, ни измерению. Однако можно сказать, что мерой сложности объекта является количество интеллектуальных усилий, необходимых для понимания этого объекта.

В борьбе со сложностью программного обеспечения можно привлечь две концепции из общей теории систем. Первая – *независимость.* В соответствии с этой концепцией для минимизации сложности необходимо максимально усилить независимость компонентов системы. По существу это означает такое разбиение системы, чтобы высокочастотная динамика ее была заключена в единых компонентах, а межкомпонентные взаимодействия представляли лишь низкочастотную динамику системы.

Вторая концепция – *иерархическая структура.* Каждый уровень представляет собой совокупность структурных отношений между элементами нижних уровней. Иерархия позволяет проектировать, описывать и понимать сложные системы. К этим двум концепциям сокращения сложности (независимость и иерархическая структура) можно добавить третью: *проявление связей* всюду, где они возникают. Основная проблема многих больших программных систем – огромное количество независимых побочных эффектов, создаваемых компонентами системы. Из–за этих побочных эффектов систему невозможно понять.

Одним из эффективных путей повышения надежности ПС является *стандартизация технологических процессов и объектов* проектирования, разработки и сопровождения программ. В стандартах жизненного цикла ПС обобщаются опыт и результаты исследований множества специалистов и рекомендуются наиболее эффективные современные методы и процессы. В результате таких обобщений отрабатываются технологические процессы и приемы разработки, а также методическая база для их автоматизации.

Классификация процессов совершенствования производства программного обеспечения по модели SEI больше подходит для процесса разработки больших и длительно эксплуатируемых систем, создаваемых большими компаниями-разработчиками. Для малых и средних компаний-разработчиков данная модель подходит не в полной мере.

Вместо того чтобы разбивать процесс совершенствования производства на уровни и строить между ними нестойкие взаимосвязи, рациональнее, по моему мнению, применить обобщенную классификацию процессов совершенствования производства, которая подходит большинству организаций и программных проектов.

Можно выделить несколько общих типов процессов совершенствования:

1. Неформальный процесс. Не имеет четко выраженной модели совершенствования производства. Его с успехом может использовать отдельная команда разработчиков. Неформальность процесса никоим образом не исключает такие формальные действия, как управление конфигура-

цией; однако при этом сами действия и их взаимосвязи не предопределены заранее.

2. Управляемый процесс. Имеет подготовленную модель, которая управляет процессом совершенствования. Модель определяет действия, их график и взаимосвязи между ними.

3. Методически обоснованный процесс. Подразумевается, что введены в действие определенные методы (например, систематически применяются методы объектно-ориентированного проектирования). Для процессов этого типа будут полезными CASE-средства поддержки проектирования и анализа процессов.

4. Процесс непосредственного совершенствования. Имеет четко поставленную цель совершенствования технологического процесса, для чего существует отдельная строка в бюджете организации и определены нормы и процедуры внедрения нововведений. Частью такого процесса является количественный анализ процесса совершенствования.

***Задание***

1) Изучите теоретический материал о повышении надежности ПО. Определите факторы в простой интуитивной модели, с помощью которых можно повысить надежность ПО. Проанализируйте факторы не входящие в данную модель, с помощью которых можно повысить надежность ПО, и с помощью которых можно повысить качество программы.

2) Составьте план для повышения надежности и качества ПО, из лабораторной работы №2, и проанализируйте – как его реализация скажется на изменении надежности в рамках простой интуитивной модели.

3) Проанализируйте какие современные ИТ повышения надежности и качества ПО, использовались, а какие нет при выполнении задания из п. 2

***Контрольные вопросы***

1. Перечислите основные характеристики влияющие на надежность ПО в простой интуитивной модели.
2. Перечислите способы и методы повышения качества и надежности программного обеспечения.
3. Укажите особенности использования современных ИТ для повышения надежности ПО.
4. Опишите способы оценки надежности больших программных комплексов.
5. Расскажите о современных технологиях повышения надежности программного обеспечения.

# ЛАБОРАТОРНАЯ РАБОТА № 4
# ВЕРИФИКАЦИЯ ПО

***Цель и задачи работы: Получить навыки практической верификации программных продуктов.***

### *Теоретический материал*

Верификация – установление истинности научных утверждений посредством их опытной проверки.

Верификация используется в различных сферах – в производстве товаров и услуг, в медицине, в интернете. Верификация модели дает возможность создать качественный прототип будущего изделия. В IT-технологиях верификация применяется для подтверждения личности пользователя при работе в Сети или использовании платежных систем. С ее помощью выявляют подделки, бракованные изделия, корректируют медицинские диагнозы, регистрируются в онлайн-сервисах и соцсетях.

**Верификация программного обеспечения** – аналогична верификации любого произведенного продукта. После разработки ПО оно тестируется на соответствие техзаданию. Продукт должен содержать все составные части, требуемые заказчиком. Насколько при этом продукт соответствует целям заказчика – вопрос не верификации, а валидации. Если программный калькулятор выглядит как калькулятор, запускается как калькулятор, соответствует требованиям заказчика к калькулятору, то он верифицирован. А вот если при этом созданное ПО не выполняет расчеты, нужные заказчику, или содержит ошибки в таких расчетах – значит, калькулятор не валидизирован.

**Верификация сайта** – подтверждение, что интернет-ресурс создан с серьезными намерениями, а не как «двойник» для совершения мошенничества или других столь же малопривлекательных целей. Лишь реальный сайт будет учитываться поисковыми системами. Также верификация требуется, если есть подозрение, что ресурс украден у его владельца. К примеру, Яндекс в качестве доказательств требует мета-теги, DNS и файл HTML.

### *Задание*

1) Ознакомитесь с теоретическим материалом и ответьте на контрольные вопросы.

2) Проведите верификацию программы, созданной Вами в лабораторной работе № 2, согласно спецификации созданной Вами в лабораторной работе № 1.

3) Обменяйтесь с партнером программами, созданными в лабораторной работе № 2, и проведите верификацию программы партнера, со-

гласно спецификации созданной Вами в лабораторной работе №1. Сравните результаты и сделайте выводы.

4) Обменяйтесь с партнером спецификациями, созданными в лабораторной работе № 1, и проведите верификацию Вашей программы и программы партнера, согласно спецификации партнера. Сравните результаты и сделайте выводы.

5) *(Задание для самостоятельной работы) Проведите сравнительный анализ результатов из п.*2-4* и сделайте выводы.

***Контрольные вопросы***

1. Для чего нужна верификация ПО.
2. Перечислите виды верификации ПО.
3. В чем отличия верификации от тестирования.
4. Что общего и чем различаются верификация и валидация.
5. Укажите основные компоненты верификации ПО.
6. Расскажите о современных технологиях повышения качества программного обеспечения.

# ЛАБОРАТОРНАЯ РАБОТА № 5
# ОСНОВНЫЕ ПОНЯТИЯ И ПРИНЦИПЫ ОРГАНИЗАЦИИ ТЕСТИРОВАНИЯ

***Цель и задачи работы: Получить навыки организации практического тестирования ПО.***

### *Теоретический материал*

Целью и содержанием отладки являются: поиск, локализация и устранение ошибок в программе.

Наличие ошибки в программах может проявляться по разному:

– программа не завершается или ее обработка прерывается до выдачи результата;

– программа завершается, но результат выдается неверный;

– выдается неверный результат спустя некоторое время после эксплуатации программы;

В первых двух случаях, когда ошибка явно существует, продолжается отладка. В третьем случае ошибки наиболее неприятные и могут привести к серьезным последствиям, так как обнаруживают себя тогда, когда программа уже сдана в эксплуатацию. Ошибки такого рода возникают в больших программных системах и имеют место в тех логических ветвях программы, которые не были проверены в процессе отладки и редко активизируются при реализации программы.

Использование структурных методов и передовых технологий разработки программ, включающих правильную организацию отладки, значительно ускоряет процесс отладки и облегчает нахождение ошибок.

Условно ошибки можно разделить на синтаксические и логические.

Синтаксические ошибки состоят в нарушении формальных правил написания программы и появляются в результате недостаточного знания пользователем языка программирования, а также невнимательности при технической подготовке программы к обработке в машине. К синтаксическим ошибкам можно отнести неправильную запись ключевого слова, отсутствие описания массива, пропуск скобки в арифметическом выражении либо инструкции, задающей формат, и т.д.

Логические ошибки подразделяются на ошибки алгоритма и семантические ошибки. Ошибки алгоритма возникают при несоответствии алгоритма поставленной задаче. Это прежде всего ошибки спецификации, неверная запись расчетной формулы, расходимость итерационного процесса и т.д. Ошибки семантические являются следствием неправильного понимания программистом смысла инструкций языка программирования или недостаточного знания математического обеспечения. Отладка состоит из 3-х взаимосвязанных действий:

– контроль правильности программы;

– локализация ошибок, обнаруженных в процессе контроля;

– исправление ошибок.

Перечисленные действия могут многократно повторяться.

Контроль программы - важнейший этап отладки, его цель – обнаружение ошибок. Методика отладки отражает последовательность применения различных методов контроля и состоит из следующих фаз:

1) визуальный контроль текста программы;

2) синтаксический контроль;

3) контроль ограничений структурного программирования;

4) статический семантический контроль;

5) тестирование программы на специально подбираемых тестах.

Первая и четвертая фазы относятся к ручному контролю, вторая и пятая – к автоматическому, а третья – к ручному контролю, если специальных инструментальных средств нет, и к автоматическому – в противном случае. Более 50% ошибок выявляется без применения компьютера.

Первые четыре фазы контроля являются контролем текста на конкретном языке программирования. Тестирование является контролем результатов, который базируется на спецификации задачи и логике алгоритма ее решения.

Визуальный контроль осуществляется путем просмотра текста алгоритма или программы с целью определения неправдоподобных или сомнительных конструкций. Этот контроль проводится по перечню шаблонных конструкций и ситуаций, которые следует проверять в программе:

1) Обращение к данным.

2) Описание данных.

3) Вычисления.

4) Операции сравнения.

5) Передачи управления.

6) Межмодульный интерфейс.

7) Инструкция ввода-вывода.

Визуальный контроль существенно сокращает время отладки и уменьшает стоимость отладки, так как помогает выявить и исправить значительную часть ошибок без выхода на машину. Часть этих ошибок предупреждается при анализе аномалий на этапе спецификации.

Задачей синтаксического контроля является проверка текста программы на соответствие формальному описанию синтаксиса языка программирования. Синтаксический контроль производится с помощью системных средств отладки. Результатом работы системных средств могут быть информационные и диагностические сообщения, а также дампы (печать состояния памяти).

Задачей семантического контроля является проверка правильности применения конструкций языка программирования и выявление в тексте программы конструкций, не формализованных в синтаксисе языка. Стати-

ческий семантический контроль состоит в исследовании синтаксически правильной программы, основанном на анализе управляющих а информационных связей и выявлении в программе конструкций, сознательное использование которых маловероятно. Такой контроль обычно выявляет ошибки следующих видов:

– недостижимая инструкция, т.е. инструкция, к которой не ведет ни один путь в программе;

– неправильный порядок инструкций ввода-вывода;

– неинициированная переменная, т.е. переменная, которой не было присвоено значение хотя бы на одном пути;

– наличие переменных, которые были описаны, но не используются ни в одной инструкции;

– отсутствие изменения переменных, которые определяют условие завершения цикла.

Статический семантический контроль может быть совмещен с визуальным контролем, если нет специальных инструментальных средств для его автоматизации.

Наиболее эффективным методом тестирования является детерминированное тестирование, при котором известны и контролируются каждая комбинация исходных данных и соответствующие ей результаты исполнения программы.

Детерминированное тестирование основывается на двух подходах: структурное тестирование (СТ) и функциональное тестирование (ФТ).

*Метод трассировки при визуальном и компьютерном способах отладки.*

Цель метода - локализация ошибки, т.е. обнаружение точного места в программе, где находится источник ошибки. Суть этого метода состоит в пошаговом выполнении всех действий, которые предписаны программой.

Трассировка может являться способом визуального контроля и выполняться без помощи компьютера, а также может выполняться с помощью компьютера.

***Задание***

1. Ознакомиться с теоретическим материалом по теме.
2. Определите номер Вашего варианта по формуле:

№ варианта = 1+ остаток от деления на 17(Целая часть( Число*Месяц*ГодРождения / № в журнале) );

3. Составить спецификацию для задачи задания 2.
4. Написать программу для решения задачи задания 2, согласно спецификации.
5. Составить план тестирования и отладки для программы из задания 4.
6. Детализировать и реализовать план отладки из задания 5.
7. Оформить документацию по проделанной работе.

# ЛАБОРАТОРНАЯ РАБОТА № 6
# ОРГАНИЗАЦИЯ ТЕСТИРОВАНИЯ

***Цель и задачи работы: Получить навыки различных видов ручного тестирования ПО.***

***Теоретический материал***

*Тестирование программ*

Процесс тестирования состоит из этапов:

1. Проектирование тестов.2. Исполнение тестов.3. Анализ полученных результатов.

На первом этапе выбирается подмножество множества тестов, которое сможет найти наибольшее количество ошибок за наименьший промежуток времени. На этапе исполнения тестов проводят, запуск тестов и отлавливают ошибки в тестируемом программном продукте.

Функциональные тесты составляются на уровне спецификации, до решения задачи. Будущий алгоритм рассматривается как «черный ящик» - функция с неизвестной (или не рассматриваемой) структурой, преобразующая входы в выходы. Суть функциональных тестов: каким бы способом ни решалась задача, при заданных входных значениях должны получиться соответствующие выходные значения.

Структурные тесты составляются для проверки логики решения, или логики работы уже готового алгоритма. Логика определяется последовательностью операций, их условным выполнением или повторением (т. е. композицией базовых конструкций). Совокупность структурных тестов должна обеспечить проверку каждой из таких конструкций. Чаще всего совокупность тщательно составленных функциональных тестов покрывает множество структурных тестов.

Наиболее рационально - сначала разрабатывать функциональные тесты, а затем – структурные.

Функциональное тестирование (тестирование «черного ящика») выявляет следующие категории ошибок:

· некорректность или отсутствие функций;
· ошибки интерфейса;
· ошибки в структурах данных;
· ошибки машинных характеристик (нехватка памяти и др.);
· ошибки инициализации и завершения.

Технология тестирования ориентирована на сокращение необходимого количества тестовых вариантов и выявление классов ошибок, а не отдельных ошибок. Разбиение на классы эквивалентности самый популярный способ достижения вышеизложенных целей. Его суть заключается в

разделении области входных данных программы на классы эквивалентности и разработке для каждого класса одного тестового варианта.

Класс эквивалентности – набор данных с общими свойствами, в силу чего при обработке любого набора данных этого класса задействуется один и тот же набор операторов. Классы эквивалентности определяются по спецификации программы. Тесты строятся в соответствии с классами эквивалентности, а именно: выбирается вариант исходных данных некоторого класса и определяются соответствующие выходные данные.

Самыми общими классами эквивалентности являются классы допустимых и недопустимых (аномальных) исходных данных. Описание класса строится как комбинация условий, описывающих каждое входное данное.

Условия допустимости или недопустимости данных задают возможные значения данных и могут описывать:

· некоторое конкретное значение; определяется один допустимый и два недопустимых класса эквивалентности: заданное значение, множество значений меньше заданного, множество значений больше заданного;

· диапазон значений; определяется один допустимый и два недопустимых класса эквивалентности: множество значений в границах диапазона; множество значений, выходящих за левую границу диапазона; множество значений, выходящих за правую границу диапазона;

· множество конкретных значений; определяется один допустимый и один недопустимый класс эквивалентности: заданное множество и множество значений, в него не входящих.

Такие классы можно описать языком логики, например, языком исчисления предикатов. Описания более сложных условий и соответствующих классов могут быть построены на основании приведенных выше условий.

Метод анализа граничных значений для построения тестов дополняет предыдущий и предполагает анализ значений, лежащих на границе допустимых и недопустимых данных. Построение таких тестов часто диктуется интуицией.

Основные правила построения тестов:

· если условие правильности данных задает диапазон, то строятся тесты для левой и правой границы диапазона; для значений чуть левее левой и чуть правее правой границы;

· если условие правильности данных задает дискретное множество значений, то строятся тесты для минимального и максимального значений; для значений чуть меньше минимума и чуть больше максимума;

· если используются структуры данных с переменными границами (массивы), то строятся тесты для минимального и максимального значения границ.

Взаимосвязь классов эквивалентности и соответствующих им действий описывается формально в виде графа (диаграмм причин-следствий)

на основе автоматного подхода. Граф преобразуется в таблицу решений, столбцы которой в свою очередь преобразуются в тестовые варианты.

*Структурное тестирование* (тестирование программ как "белого ящика") предполагает детальное изучение текста (логики) программы и построение (подбор) таких входных наборов данных, которые позволили бы при многократном выполнении программы на ЭВМ обеспечить выполнение максимально возможного количества маршрутов, логических ветвлений, циклов и т.д.

*Функциональное тестирование* (тестирование программ как "черного ящика") полностью абстрагируется от логики программы, предполагается, что программа - "черный ящик", а тестовые наборы выбираются на основании анализа входных функциональных спецификаций.

Подмножество всех возможных тестов, которое имеет наивысшую вероятность обнаружения большинства ошибок, называется **эффективным.** Чтобы разработать эффективный тестовый набор, необходимо знать ряд методов его построения и придерживаться определенных правил и рекомендаций. В соответствии с методом детерминированного тестирования при структурном тестировании ориентируются на построение тестовых наборов по принципу "белого ящика", а при функциональном тестировании - по принципу "черного ящика".

При построении тестовых наборов данных по принципу "белого ящика" руководствуются следующими критериями:

**Покрытие операторов**. Этот критерий предполагает выбор такого тестового набора данных, который вызывает выполнение каждого оператора в программе хотя бы один раз.

**Покрытие узлов ветвления** (покрытие решений). Этот критерий предполагает разработку такого количества тестов, чтобы в каждом узле ветвления был обеспечен переход по веткам "истина" и "ложь" хотя бы один раз.

**Покрытие условий**. Если узел ветвления содержит более одного условия, тогда нужно разработать число тестов, достаточное для того, чтобы возможные результаты каждого условия в решении выполнялись, по крайней мере один раз.

**Комбинаторное покрытие условий**. Этот критерий требует создания такого числа тестов, чтобы все возможные комбинации результатов условия в каждом решении по крайней мере один раз.

При построении тестов по стратегии "черного ящика" программа рассматривается как "черный ящик", а исходной информацией для тестовых наборов служат ее спецификации. К стратегии "черного ящика" относятся методы:

**Метод эквивалентного разбиения**. Построение тестов методов эквивалентного разбиения осуществляется в 2 этапа: 1) выделение классов эквивалентности; 2) построение тестов. Класс эквивалентности множество

входных значений, каждое из которых имеет одинаковую вероятность обнаружения конкретного типа ошибки.

**Анализ граничных значений**. Этот метод предполагает исследование ситуаций, возникающих на границах и вблизи границ эквивалентных разбиений.

**Метод функциональных диаграмм.** Метод заключается в преобразовании входной спецификации программы в функциональную диаграмму (диаграмму причинно-следственных связей) с помощью простейших булевских отношений, построения таблицы решений, которая является основой для написания эффективных тестовых наборов данных.

***Задание***

1. Ознакомиться с теоретическим материалом по теме.
2. Для программы решения задачи из лабораторной работы № 5 составить планы структурного и функционального тестирования.
3. Детализировать и реализовать планы из задания 2.
4. За документировать результаты задания 3.
5. Проанализировать наборы тестов из задания 3.

***Задания для самостоятельной работы***

1) Изучите теоретический материал о видах тестирования, которые не использовались в лабораторной работе.
2) Составьте краткий реферат(презентацию) об одном из нефункциональных видов тестирования.

Реализуйте тестирование рассмотренное в задании 2 на задаче из лабораторной работы № 5.

# ЛАБОРАТОРНАЯ РАБОТА № 7
# АВТОМАТИЗАЦИЯ ТЕСТИРОВАНИЯ

***Цель и задачи работы: Получить навыки автоматизации тестирования ПО, использования средств автоматизации.***

*Теоретический материал*

**Автоматизированное тестирование программного обеспечения** (**SoftwareAutomationTesting**) – это процесс верификации программного обеспечения, при котором основные функции и шаги теста, такие как запуск, инициализация, выполнение, анализ и выдача результата, выполняются автоматически при помощи инструментов для автоматизированного тестирования.

**Инструмент для автоматизированного тестирования** (**AutomationTestTool**) – это программное обеспечение, посредством которого специалист по автоматизированному тестированию осуществляет создание, отладку, выполнение и анализ результатов прогона тест скриптов.

**Тест Скрипт** (**TestScript**) – это набор инструкций, для автоматической проверки определенной части программного обеспечения.

**Тестовый набор** (**TestSuite**) – это комбинация тест скриптов, для проверки определенной части программного обеспечения, объединенной общей функциональностью или целями, преследуемыми запуском данного набора.

**Тесты для запуска** (**TestRun**) – это комбинация тест скриптов или тестовых наборов для последующего совместного запуска (последовательного или параллельного, в зависимости от преследуемых целей и возможностей инструмента для автоматизированного тестирования).

**Автоматизированное функциональное тестирование ПО** (**FunctionalAutomationTesting**) – это процесс верификации **функциональных требований** и особенностей тестируемого приложения, посредством инструментов для автоматизированного тестирования. (см. также Функциональное тестирование)

С автоматизацией тестирования, как и со многими другими узконаправленными IT - дисциплинами, связано много неверных представлений. Для того, чтобы избежать неэффективного применения автоматизации, следует обходить ее недостатки и максимально использовать преимущества. Далее мы перечислим и дадим небольшое описание для основных нюансов автоматизации и дадим ответ на основной вопрос данной статьи – когда автоматизацию все таки стоит применять.

**Преимущества автоматизации тестирования:**

- Повторяемость – все написанные тесты всегда будут выполняться однообразно, то есть исключен «человеческий фактор». Тестировщик не пропустит тест по неосторожности и ничего не напутает в результатах.

• Быстрое выполнение – автоматизированному скрипту не нужно сверяться с инструкциями и документациями, это сильно экономит время выполнения.

• Меньшие затраты на поддержку – когда автоматические скрипты уже написаны, на их поддержку и анализ результатов требуется, как правило, меньшее время чем на проведение того же объема тестирования вручную.

• Отчеты – автоматически рассылаемые и сохраняемые отчеты о результатах тестирования.

• Выполнение без вмешательства – во время выполнения тестов инженер-тестировщик может заниматься другими полезными делами, или тесты могут выполняться в нерабочее время (этот метод предпочтительнее, так как нагрузка на локальные сети ночью снижена).

**Недостатки автоматизации тестирования :**

• Повторяемость – все написанные тесты всегда будут выполняться однообразно. Это одновременно является и недостатком, так как тестировщик, выполняя тест вручную, может обратить внимание на некоторые детали и, проведя несколько дополнительных операций, найти дефект. Скрипт этого сделать не может.

• Затраты на поддержку – несмотря на то, что в случае автоматизированных тестов они меньше, чем затраты на ручное тестирование того же функционала – они все же есть. Чем чаще изменяется приложение, тем они выше.

• Большие затраты на разработку – разработка автоматизированных тестов это сложный процесс, так как фактически идет разработка приложения, которое тестирует другое приложение. В сложных автоматизированных тестах также есть фреймворки, утилиты, библиотеки и прочее. Естественно, все это нужно тестировать и отлаживать, а это требует времени.

• Стоимость инструмента для автоматизации – в случае если используется лицензионное ПО, его стоимость может быть достаточно высока. Свободно распространяемые инструменты как правило отличаются более скромным функционалом и меньшим удобством работы.

• Пропуск мелких ошибок - автоматический скрипт может пропускать мелкие ошибки на проверку которых он не запрограммирован. Это могут быть неточности в позиционировании окон, ошибки в надписях, которые не проверяются, ошибки контроля и форм с которыми не осуществляется взаимодействие во время выполнения скрипта.

***Задание***

1) Ознакомитесь с теоретическим материалом по теме.

2) Изучите программные средства, с помощью которых будет осуществляться автоматизация тестирования.

3) Выберите объект тестирования. Рекомендуется, если вы владеете навыками программирования, в качестве объекта тестирования выбрать решение задачи из лабораторной работы № 5.

4) Составьте план тестирования выбранного объекта.

5) Разработайте набор тест-кейсов и тест-сьют для тестируемого приложения.

6) Реализуйте автоматизацию тестирования, задокументируйте результаты, оформите отчет.

***Задания для самостоятельной работы***

1) Изучите теоретический материал о видах автоматизации тестирования.

2) Составьте краткий реферат(презентацию) о видах и особенностях автоматизации тестирования ПО для различных прикладных областей.

3) Реализуйте автоматизацию тестирования объекта из лабораторной работы №7 другим видом автоматизации тестирования, отличным от реализованного в лабораторной работе № 7.

***Контрольные вопросы***

1. Для чего нужна автоматизация тестирования ПО.
2. Перечислите виды автоматизации тестирования ПО.
3. В чем проблемы организации автоматизации тестирования.
4. Преимущества автоматизации тестирования ПО.
5. Недостатки автоматизации тестирования ПО.
6. Укажите основные этапы организации автоматизации тестирования ПО.
7. Расскажите о современных технологиях автоматизации тестирования программного обеспечения.

# МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ К ЛАБОРАТОРНОЙ РАБОТЕ № 7 «АВТОМАТИЗАЦИЯ ТЕСТИРОВАНИЯ»

## Автоматизация тестирования специальным ПО

Автоматизация тестирования может осуществляться с использованием специального программного обеспечения –в качестве примера можно воспользоваться Selenium IDE— это инструмент, используемый для разработки тестовых сценариев. Он представляет собой простое в использовании дополнение к браузеру Firefox и является наиболее эффективным способом разработки тестовых сценариев.

Запустите Firefox и скачайте и установите IDE с веб-сайта SeleniumHQ**:http://docs.seleniumhq.org/download/**

Перезапустите Firefox. После перезапуска Selenium IDE появится в меню “Инструменты”.

Используя, документацию с сайта и справочную систему Selenium IDE изучите синтаксис языка команд Selenium, возможности IDE и ее основные компоненты:

Панель меню

Панель инструментов (Toolbar)

Панель тестового сценария

Дополнение среди прочего содержит контекстное меню, позволяющее пользователю сначала выбрать любой элемент интерфейса на отображаемой браузером в данный момент странице, а затем выбрать команду из списка команд Selenium с параметрами, предустановленными в соответствии с выбранным элементом. Это не только экономит время, но и дает замечательную возможность для изучения языка команд Selenium.

## Автоматизация тестирования возможностями среды программирования

Возможности среды программирования для автоматизации тестирования программного обеспечения рассмотрим на примере языка python. Bpython имеется тип данных - исключения (exceptions) для того, чтобы сообщать программисту об ошибках. Иерархию встроенных в python исключений можно посмотреть в Приложении 1.

Для обработки исключений используется конструкция **try – except,** пример её применения в автоматизации тестирования:

```
f = open('1.txt')
ints = []
```

```
try:
forlinein f:
ints.append(int(line))
exceptValueError:
  print('Ошибка: Не числовой тип.')
exceptException:
  print('Ошибка: проверьтекод. ')
else:
print('Выполнено без ошибок.')
finally:
f.close()
  print('Файлзакрыт.')
```

В Python встроен модуль unittest, который поддерживает автоматизацию тестов, использование общего кода для настройки и завершения тестов, объединение тестов в группы, позволяет отделять тесты от фреймворка для вывода информации.

Для автоматизации тестов, unittest поддерживает концепции:

- **Испытательный стенд** (testfixture) - выполняется подготовка для выполнения тестов и действия для очистки после выполнения тестов, может включать: запуск серверного процесса, создание временных баз данных, и т.п..
- **Тестовый случай** (testcase) - минимальный блок тестирования - проверяет ответы для разных наборов данных. Модуль unittest содержит базовый класс TestCase, который используется для создания новых тестовых случаев.
- **Набор тестов** (testsuite) - несколько тестовых случаев, наборов тестов или и того и другого - используется для объединения тестов, которые должны быть выполнены вместе.
- **Исполнитель тестов** (testrunner) - компонент управляющий выполнением тестов и предоставляющий пользователю результат, может использовать графический или текстовый интерфейс или возвращать специальное значение, сообщающее о результатах выполнения тестов.

Модуль unittest предоставляет набор инструментов для написания и запуска тестов.

Пример скриптатестирования методов строк:

```
importunittest
classTestStringMethods(unittest.TestCase):
deftest_upper(self):
self.assertEqual('foo'.upper(), 'FOO')
deftest_isupper(self):
```

```
self.assertTrue('FOO'.isupper())
self.assertFalse('Foo'.isupper())
deftest_split(self):
  s = 'helloworld'
self.assertEqual(s.split(), ['hello', 'world'])
  # Проверим, что s.split не работает, если разделитель - не строка
withself.assertRaises(TypeError):
s.split(2)
if __name__ == '__main__':
unittest.main()
```

Подробнее с возможностями автоматизации тестирования в языке python можно познакомиться в документации к языку или в справочной литературе, например:

**https://pythonworld.ru/moduli/modul-unittest.html**

## ПРИЛОЖЕНИЕ 1

# ИЕРАРХИЯ ВСТРОЕННЫХ В PYTHON ИСКЛЮЧЕНИЙ

**BaseException** – базовое исключение, от которого берут начало все остальные.

- **SystemExit** – исключение, порождаемое функцией sys.exit при выходе из программы.
- **KeyboardInterrupt** – порождается при прерывании программы пользователем (обычно сочетанием клавиш Ctrl+C).
- **GeneratorExit** – порождается при вызове метода close объекта generator.
- **Exception** – а вот тут уже заканчиваются полностью системные исключения (которые лучше не трогать) и начинаются обыкновенные, с которыми можно работать.
  - **StopIteration** – порождается встроенной функцией next, если в итераторе больше нет элементов.
  - **ArithmeticError** – арифметическая ошибка.
    - **FloatingPointError** – порождается при неудачном выполнении операции с плавающей запятой. На практике встречается нечасто.
    - **OverflowError** – возникает, когда результат арифметической операции слишком велик для представления. Не появляется при обычной работе с целыми числами (так как python поддерживает длинные числа), но может возникать в некоторых других случаях.
    - **ZeroDivisionError** – деление на ноль.
  - **AssertionError** – выражение в функции assert ложно.
  - **AttributeError** – объект не имеет данного атрибута (значения или метода).
  - **BufferError** – операция, связанная с буфером, не может быть выполнена.
  - **EOFError** – функция наткнулась на конец файла и не смогла прочитать то, что хотела.
  - **ImportError** – не удалось импортирование модуля или его атрибута.
  - **LookupError** – некорректный индекс или ключ.
    - **IndexError** – индекс не входит в диапазон элементов.

        - **KeyError** – несуществующий ключ (в словаре, множестве или другом объекте).
    - **MemoryError** – недостаточно памяти.
    - **NameError** – не найдено переменной с таким именем.
        - **UnboundLocalError** – сделана ссылка на локальную переменную в функции, но переменная не определена ранее.
    - **OSError** – ошибка, связанная с системой.
        - **BlockingIOError**
        - **ChildProcessError** – неудача при операции с дочерним процессом.
        - **ConnectionError** – базовый класс для исключений, связанных с подключениями.
            - **BrokenPipeError**
            - **ConnectionAbortedError**
            - **ConnectionRefusedError**
            - **ConnectionResetError**
        - **FileExistsError** – попытка создания файла или директории, которая уже существует.
        - **FileNotFoundError** – файл или директория не существует.
        - **InterruptedError** – системный вызов прерван входящим сигналом.
        - **IsADirectoryError** – ожидался файл, но это директория.
        - **NotADirectoryError** – ожидалась директория, но это файл.
        - **PermissionError** – не хватает прав доступа.
        - **ProcessLookupError** – указанного процесса не существует.
        - **TimeoutError** – закончилось время ожидания.
    - **ReferenceError** – попытка доступа к атрибуту со слабой ссылкой.
    - **RuntimeError** – возникает, когда исключение не попадает ни под одну из других категорий.
    - **NotImplementedError** – возникает, когда абстрактные методы класса требуют переопределения в дочерних классах.
    - **SyntaxError** – синтаксическая ошибка.
        - **IndentationError** – неправильные отступы.
            - **TabError** – смешивание в отступах табуляции и пробелов.

- **SystemError** – внутренняя ошибка.
- **TypeError** – операция применена к объекту несоответствующего типа.
- **ValueError** – функция получает аргумент правильного типа, но некорректного значения.
- **UnicodeError** – ошибка, связанная с кодированием / раскодированием unicode в строках.
    - **UnicodeEncodeError** – исключение, связанное с кодированием unicode.
    - **UnicodeDecodeError** – исключение, связанное с декодированием unicode.
    - **UnicodeTranslateError** – исключение, связанное с переводом unicode.
- **Warning** – предупреждение.

Учебное издание

# ТЕСТИРОВАНИЕ, ВЕРИФИКАЦИЯ И АТТЕСТАЦИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Методические рекомендации

Составитель

**ШЕДЬКО** Василий Викторович

Технический редактор *Г.В. Разбоева*

Компьютерный дизайн *Е.А. Барышева*

Подписано в печать 2020. Формат 60x84 $^{1}/_{16}$. Бумага офсетная.

Усл. печ. л. 1,81. Уч.-изд. л. 1,34. Тираж экз. Заказ .

Издатель и полиграфическое исполнение – учреждение образования «Витебский государственный университет имени П.М. Машерова».

Свидетельство о государственной регистрации в качестве издателя, изготовителя, распространителя печатных изданий № 1/255 от 31.03.2014.

Отпечатано на ризографе учреждения образования «Витебский государственный университет имени П.М. Машерова». 210038, г. Витебск, Московский проспект, 33.