МИНОБРНАУКИ РОССИИ

РГУ нефти и газа (НИУ) имени И.М. Губкина

|  |  |
| --- | --- |
| Факультет | **Автоматики и вычислительной техники** |
| Кафедра | **Автоматизированных систем управления** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Оценка комиссии: |  | | | Рейтинг: |  |
| Подписи членов комиссии: | | | | | |
|  | |  | Волков Д.А. | | |
| (подпись) | |  | (фамилия, имя, отчество) | | |
|  | |  | Мухина А.Г. | | |
| (подпись) | |  | (фамилия, имя, отчество) | | |
|  | | | | | |
| (дата) | | | | | |
|  | |  |  | | |

**КУРСОВАЯ РАБОТА**

|  |  |
| --- | --- |
| по дисциплине | Мониторинг надежности в АСДУ |
|  | |

|  |  |
| --- | --- |
| на тему | Разработка программы для оценки надежности АСДУ |
|  | |
|  | |

|  |  |  |  |
| --- | --- | --- | --- |
| «К ЗАЩИТЕ» |  | ВЫПОЛНИЛ: |  |
|  |  | Студент группы | **АСМ-19-04** |
|  |  |  | (номер группы) |
| Ст. преп. Волков Денис Андреевич |  | Харитонов Виталий Игоревич | |
| (должность, ученая степень; фамилия, и.о.) |  | (фамилия, имя, отчество) | |
|  |  |  | |
| (подпись) |  | (подпись) | |
|  |  |  | |
| (дата) |  | (дата) | |

|  |  |  |  |
| --- | --- | --- | --- |
|  | Москва, 20 | 20 |  |

МИНОБРНАУКИ РОССИИ

РГУ нефти и газа (НИУ) имени И.М. Губкина

|  |  |
| --- | --- |
| Факультет | **Автоматики и вычислительной техники** |
| Кафедра | **Автоматизированных систем управления** |

**ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ**

|  |  |
| --- | --- |
| по дисциплине | Мониторинг надежности в АСДУ |

|  |  |
| --- | --- |
| на тему | Разработка программы для оценки надежности АСДУ |

|  |  |  |  |
| --- | --- | --- | --- |
| ДАНО студенту | **Харитонову Виталию Игоревичу** | группы | **АСМ-19-04** |
|  | (фамилия, имя, отчество в дательном падеже) |  | (номер группы) |

**Содержание работы:**

|  |  |
| --- | --- |
|  | Разработка модуля оценки надежности диспетчера |
|  | Разработка модуля оценки и прогнозирования надежности технологического оборудования |
|  | Разработка модуля оценки надежности ПО |
|  | Разработка структуры базы данных |
|  | Интеграция разработанных модулей в единое информационное пространство |

**Исходные данные для выполнения работы:**

|  |  |
| --- | --- |
|  |  |

**Рекомендуемая литература:**

|  |  |
| --- | --- |
|  | Шишмарёв, В. Ю. Надежность технических систем : учебник для вузов / В. Ю. Шишмарёв. — 2-е изд., испр. и доп. — Москва : Издательство Юрайт, 2020. — 289 с. — (Высшее образование). |
|  | Яхьяев Н.Я., Кораблин А.В. Основы теории надежности. Учебник для вузов. — М.: Академия, 2014. — 208 с. |

**Графическая часть:**

|  |  |
| --- | --- |
|  | ER - диаграмма базы данных |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Руководитель: |  |  | Старший преподаватель |  |  |  | Волков Д. А. |
|  | (уч.степень) |  | (должность) |  | (подпись) |  | (фамилия, имя, отчество) |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Задание принял к исполнению: | студент |  |  |  | Харитонов В. И. |
|  | |  | (подпись) |  | (фамилия, имя, отчество) |

**Оглавление**

[Введение 4](#_Toc60043886)

[Основная часть 6](#_Toc60043887)

[Описание предметной области 6](#_Toc60043888)

[Оценка надежности безошибочного выполнения оператором поставленных ему задач 6](#_Toc60043889)

[Оценка надежности технологического оборудования 7](#_Toc60043890)

[Расчет надежности ПО. Модель Миллса 10](#_Toc60043891)

[Разработка модуля оценки надежности диспетчера 11](#_Toc60043892)

[Разработка модуля оценки и прогнозирования надежности технологического оборудования 14](#_Toc60043893)

[Разработка модуля оценки надежности ПО 16](#_Toc60043894)

[Интеграция разработанных модулей в единое информационное пространство 17](#_Toc60043895)

[Разработка структуры базы данных 19](#_Toc60043896)

[Таблица Probabilities 19](#_Toc60043897)

[Таблица Elements 19](#_Toc60043898)

[Таблица Reliability 19](#_Toc60043899)

[Заключение 20](#_Toc60043900)

[Литература 21](#_Toc60043901)

[Приложение 22](#_Toc60043902)

# Введение

Точность работы диспетчера – это степень соответствия выполнения им определенных функций предписанному алгоритму. Она зависит от степени сложности выполняемых операций, условий и режима работы, состояния нервной системы диспетчера, его индивидуальных способностей и других факторов.

Несвоевременное исполнение тех или иных операций диспетчером снижает эффективность функционирования системы в целом, что может квалифицироваться как допущение ошибок.

Как показывают практика и исследования, технологическое оборудование в процессе эксплуатации постепенно теряет свои начальные характеристики, что приводит к снижению качествен­ных показателей технологического процесса. При этом восстановление работоспособности, как правило, связано с большими за­тратами времени и средств.

Каждый разработчик стремится сделать свою программу максимально надежной, эффективной и безотказной. К сожалению, полностью исключить факт сбоя или ошибки в настоящий момент не представляется возможным, хотя каждый день в этом направлении ведется работа.

Но, представляется возможным рассчитать вероятность возникновения ошибок в программе вследствие тестирования ее на различных математических моделях надежности. Для того чтобы знать, на сколько надежна программа, ее необходимо протестировать.

В соответствии с поставленной целью были сформулированы следующие задачи:

* Разработать модуль оценки надежности диспетчера.
* Разработать модуль оценки и прогнозирования надежности технологического оборудования.
* Разработать модуль оценки надежности ПО.
* Разработать структуру базы данных.
* Интегрировать разработанные модули в единое информационное пространство.

Объектами исследования являются методы оценки надежности работы диспетчеров, технологического оборудования и ПО.

Предметом исследования является программа на платформе PyCharm на языке Python.

# Основная часть

## **Описание предметной области**

### **Оценка надежности безошибочного выполнения оператором поставленных ему задач**

Надежность работы диспетчера определяется как вероятность успешного выполнения им поставленной задачи на заданном этапе функционирования системы.



где:

𝑃Т – вероятность безотказной работы технических и программных средств;

𝑃ОП –вероятность безошибочного выполнения операций диспетчером;

𝑃ИСП – вероятность исправления диспетчером допущенной ошибки;

Вероятность безотказной работы и вероятность исправления ошибки могут быть заранее рассчитаны, либо формироваться на основе результатов тестирования диспетчерского персонала.

Для типовых, часто повторяющихся операций в качестве показателя безошибочности может использоваться интенсивность ошибок:

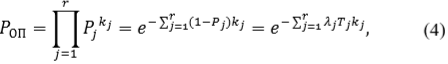


где:

𝑃𝑗 – вероятность безошибочного выполнения операций 𝑗-го типа; 𝜆𝑗 – интенсивность ошибок 𝑗-го вида;

𝑁𝑗, 𝑛𝑗 – общее число выполненных операций 𝑗-го вида и допущенное при этом число ошибок;

𝑇𝑗 – среднее время выполнения операций 𝑗-го вида. Тогда вероятность безошибочного выполнения операций диспетчером может быть вычислена следующим образом:



где:   
𝑘𝑗 – число выполненных операций 𝑗-го вида;

𝑟 – число различных типов операций (𝑗 = 1,2,…𝑟).

Для некоторых операций диспетчер может исправить допущенную ошибку, таким образом показатель компенсации ошибки – это вероятность исправления диспетчером допущенной ошибки, т.е. способность самоконтроля диспетчером своих действий. Рассчитывается по следующей формуле:



где:

𝑃К – вероятность выдачи сигнала системой контроля;

𝑃ОБ – вероятность обнаружения диспетчером сигнала контроля;

𝑃И – вероятность исправления ошибочных действий при повторном выполнении операции.

### **Оценка надежности технологического оборудования**

Задача работы – разработка программного модуля оценки надежности технологического оборудования. Пользователю предоставляется возможность в диалоговом режиме возможность построить для анализируемой системы диаграмму надежности. Пользователь может проследить во времени изменение надежности всех элементов системы и, задавая требуемый минимальный уровень надежности системы в целом, оценить момент времени снижения до критического уровня надежности, в котором уже следует произвести замену соответствующего элемента.

Обозначение исходных данных:

Звено i – номер подсистемы диаграммы надежности;

j – номер элемента в i-й подсистеме;

Pt – требуемый уровень надежности системы;

Обозначение оцениваемых данных:

Tij – момент замены j-ого элемента в i-й подсистеме;

Pi – граничный уровень надежности i-й подсистемы;

Диаграмма надежности строится посредством выбора для i-й подсистемы возможной конфигурации элементов:

* один элемент;
* два параллельно соединенных элемента;
* параллельно соединенные две группы элементов, первая состоящая из двух последовательных элементов, а вторая из одного элемента;
* параллельно соединенные две группы элементов, каждая из которых состоит из двух последовательных элементов.

Имея заданное пользователем значение требуемого уровня надежности и, располагая числом подсистем, определяется требуемый уровень надежности для каждой подсистемы. Требуемый уровень надежности i-й подсистеме будет:



где: PTP – требуемый уровень надежности системы; n – общее количество подсистем.

Вычислив требуемый уровень надежности для каждой подсистемы, рассчитываем необходимый уровень надежности для каждого элемента. В зависимости от конфигурации элементов в группе надежность определяется по одной из нижеперечисленных формул соответственно:

для конфигурации а):



для конфигурации б):

из уравнения для определения надежности параллельно соединенных двух элементов  , находим:



для конфигурации в):

в этом случае надежность третьего элемента будет определяться по выше приведенной формуле:



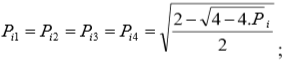
а для первого и второго надежность будет:



так как они являются параллельно соединенные.

для конфигурации г):

в этом случае надежность элементов будет определятся по формуле:



Зная требуемую надежность каждого элемента и время безотказной работы находим моменты замены.

Для систем, в которых происходит интенсивное использование элементов и требуемый уровень надежности достаточно велик, можно предположить, что процесс изменения надежности развивается по экспоненциальному закону распределения:



где: λ= 1/время безотказной работы; t-время,

требуемый момент замены определяется по формуле:



Если показатель - “Наработка на отказ”, то переход к определению λ осуществляется по формуле:

,

где Т – наработка на отказ.

### **Расчет надежности ПО. Модель Миллса**

Использование этой модели предполагает необходимость перед началом тестирования искусственно "засорять" программу, т.е. вносить в нее некоторое количество известных ошибок. Ошибки вносятся случайным образом и фиксируются в протоколе искусственных ошибок. Специалист, проводящий тестирование, не знает ни количества, ни характера внесенных ошибок до момента оценки показателей надежности по модели Миллса. Предполагается, что все ошибки (как естественные, так и искусственно внесенные) имеют равную вероятность быть найденными в процессе тестирования.

Тестируя программу в течение некоторого времени, собирают статистику об ошибках. В момент оценки надежности по протоколу искусственных ошибок все ошибки делятся на собственные и искусственные. Соотношение, называемое формулой Миллса, N=S\*n/V дает возможность оценить первоначальное число ошибок в программе N. Здесь S - количество искусственно внесенных ошибок; n - число найденных собственных ошибок; V - число обнаруженных к моменту оценки искусственных ошибок.

## **Разработка модуля оценки надежности диспетчера**

Разработан программный модуль с графическим интерфейсом, позволяющий вычислить вероятность безошибочного выполнения операций оператором.

Модуль обладает следующими возможностями:

* загрузка исходных данных из файла (xls) и выгрузка обратно;
* отображение частот ошибок, загруженных из файла (таблица) с возможностью редактирования (добавление, удаление, изменение);
* задание показателей, необходимых для расчета (𝑇𝑗,𝑁𝑗,𝑃К…).

На сегодняшний день существует множество языков программирования, каждый из которых имеет свои особенности. Но хочется выделить Python, как популярную универсальную среду разработки программного кода с тридцатилетней историей.

Python, как и любой другой язык программирования, имеет свои отличительные особенности. Итак, можно выделить следующие:

* Кроссплатформенность. Python – это интерпретируемый язык, его интерпретаторы существуют для многих платформ. Поэтому с запуском его на любой ОС не должно возникнуть проблем.
* С Python доступно огромное количество сервисов, сред разработки и фреймворков. Легко можно найти подходящий продукт для работы.
* Возможность подключить библиотеки, написанные на С. Это позволяет повысить эффективность, улучшить быстродействие.
* Наличие самых разных источников информации о Python. Не составит труда найти ответ на любой возникший вопрос, так существует много бесплатной литературы, обучающих видео-пособий, готовых исходников и шаблонов для работы в открытом доступе.

Python легко конкурирует с другими языками программирования, так как имеет множество достоинств. Во-первых, это понятный и простой язык программирования. Особенно подходит он для новичков. Можно создать интересные приложения, и при этом не придется сидеть неделями, изучая сложный синтаксис.

Динамическая типизация – это одно из главных достоинств языка Python. Для новичков это возможность упростить написание кода и избежать множества фатальных ошибок и багов в работе. Также в Python нет операторных скобок, с расставлением которых зачастую возникают сложности.

По скорости выполнения программ, когда это касается крупных полномасштабных проектов, Python, конечно же, не лидер. Здесь минусом является и автоматическое управление памятью, и полная динамическая типизация. Python уступает значительно таким языкам как Java, С, С++, но и в то же время с легкостью дает фору JavaScript, Ruby, PHP [3]. Подключение библиотек, написанных на С и возможность предварительной компиляции кода в байт-код – все это позволяет улучшить быстродействие [3].

Также хочется отметить, что Python – это язык программирования, востребованный сегодня и с большим потенциалом в будущее. Сегодня рынок труда нуждается в квалифицированных специалистах со знаниями Python.

PyCharm - это интегрированная среда разработки для Python, которая имеет полный комплект средств, необходимых для эффективного программирования на Python.

PyCharm имеет удобный редактор кода со всеми полезными функциями: подсветкой синтаксиса, автоматическим форматированием, дополнением и отступами. PyCharm позволяет проверять версии интерпретатора языка на совместимость, а также использовать шаблоны кода.

Тем, кто часто использует документацию, будет удобно смотреть ее прямо в окне редактора (для элементов) либо в браузере (для внешней документации).

PyCharm позволяет быстро производить рефакторинг кода, а также использовать удобный графический отладчик.

Утилита поддерживает все свежие версии Django, а также IronPython, Jython, Cython, PyPy wxPython, PyQt, PyGTK и многие другие инструменты.

В PyCharm можно проводить интегрированное Unit тестирование, использовать интерактивные консоли для Python, Django, SSH, отладчика и баз данных.

PyCharm имеет большую коллекцию плагинов, и его можно использовать в связке с разными трекерами вроде JIRA, Youtrack, Lighthouse, Redmine, Trac и так далее.

PyCharm кросс-платформенная среда разработки: можно использовать на Linux, Windows и Mac OS.

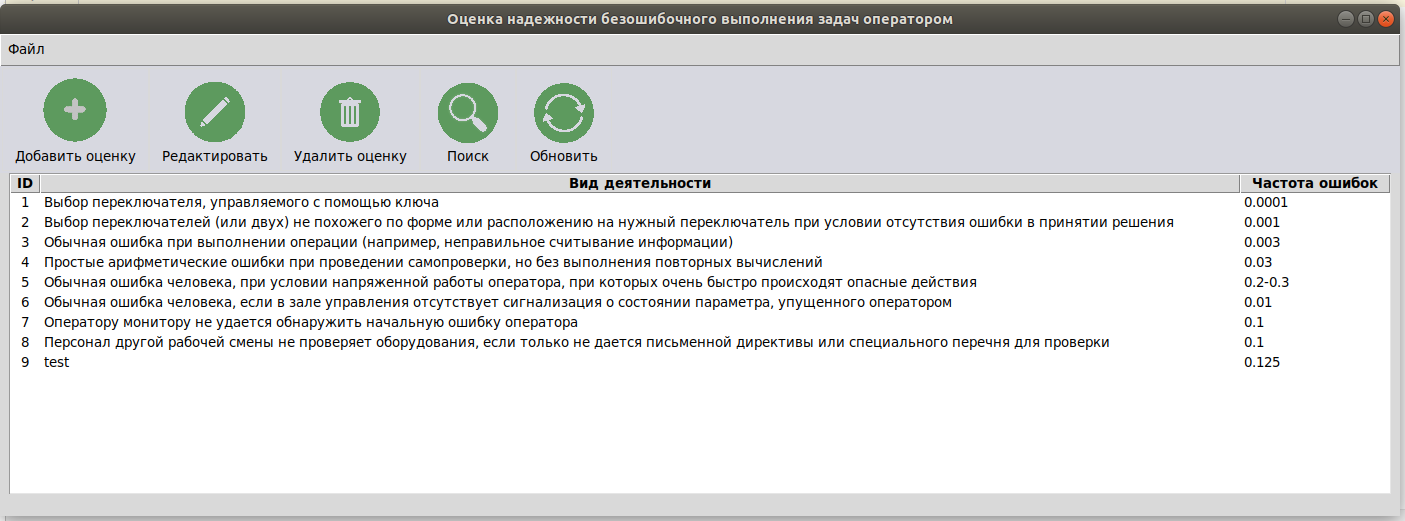
****

Рисунок 1. Окно программы по оценке безошибочного выполнения задач оператором

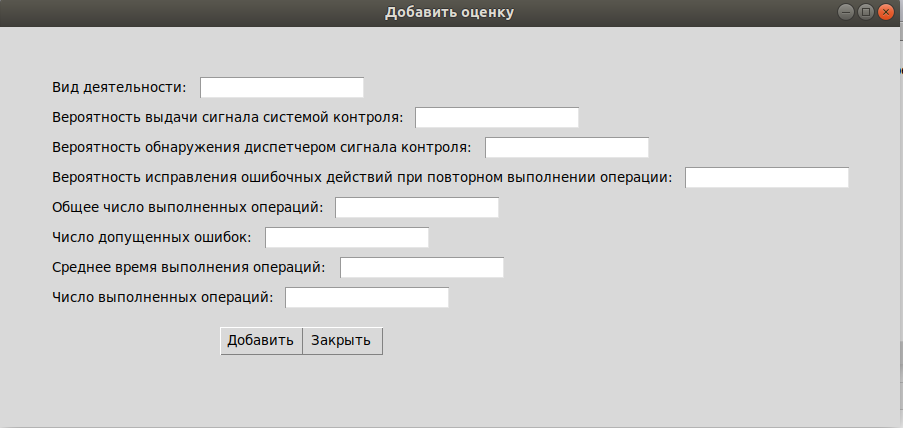
****

Рисунок 2. Форма для заполнения данных по оценке безошибочного выполнения задач оператором

## **Разработка модуля оценки и прогнозирования надежности технологического оборудования**

Разработана программа, позволяющая создать план замены технологического оборудования.

Реализована возможность сохранения в БД и выбора из БД структуры собранной схемы (отдельные элементы, показатели), а также сохранение рассчитанных планов замены оборудования. При этом реализовано отдельное область, позволяющая просмотреть сохраненные элементы.

В качестве [GUI фреймворк](https://python-scripts.com/category/gui)а (graphical user interface) используется Tkinter. У Tkinter есть несколько преимуществ. Он кроссплатформенный, поэтому один и тот же код можно использовать на Windows, macOS и Linux [1].

Визуальные элементы отображаются через собственные элементы текущей операционной системы, поэтому приложения, созданные с помощью Tkinter, выглядят так, как будто они принадлежат той платформе, на которой они работают.

Хотя Tkinter является популярным GUI фреймворком на Python, у него есть свои недостатки. Один из них заключается в том, что графические интерфейсы, созданные с использованием Tkinter, выглядят устаревшими. Если вам нужен современный, броский интерфейс, то Tkinter может оказаться не совсем тем, для этого есть [PyQt5](https://python-scripts.com/pyqt5) который развивается сильнее в данном плане.

Тем не менее, в плане использования, Tkinter является относительно легким по сравнению с другими библиотеками. Это отличный выбор для создания GUI приложений в Python, особенно если современный облик не в приоритете для программы, а большую роль играет функциональность и кроссплатформенная скорость.

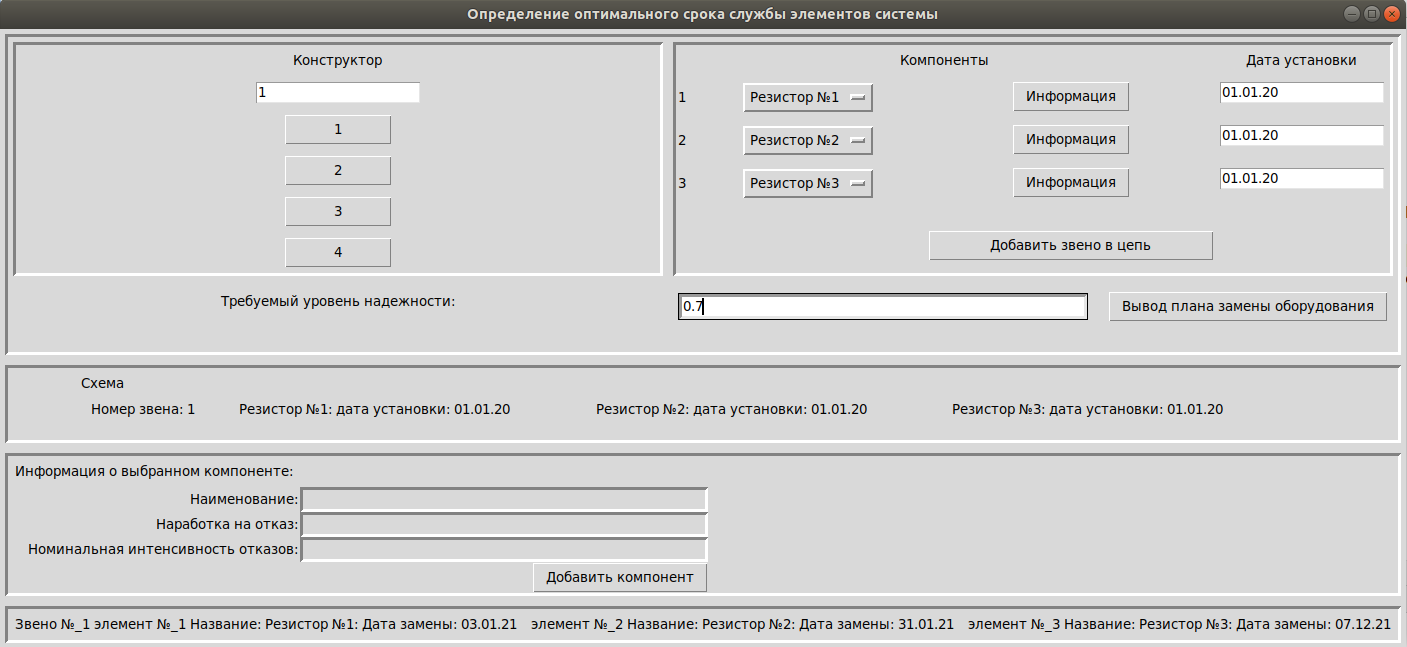
****

Рисунок 3. Окно программы по определению оптимального срока службы элементов системы

## **Разработка модуля оценки надежности ПО**

Разработан модуль оценки надежности ПО.

Реализована возможность сохранения в БД SQLite и выбора из БД исходных данных для расчета надежности ПО, а также результатов расчета.

SQLite – это встраиваемая библиотека, в которой реализовано многое из стандарта SQL 92. Её притязанием на известность является как собственно сам движок базы, так и её интерфейс (точнее его движок) в пределах одной библиотеки, а также возможность хранить все данные в одном файле. Я отношу позицию функциональности SQLite где-то между MySQL и PostgreSQL. Однако, на практике, SQLite не редко оказывается в 2-3 раза (и даже больше) быстрее. Такое возможно благодаря высокоупорядоченной внутренней архитектуре и устранению необходимости в соединениях типа «сервер-клиент» и «клиент-сервер».

Всё это, собранное в один пакет, лишь немногим больше по размеру клиентской части библиотеки MySQL, является впечатляющим достижением для полноценной базы данных. Используя высоко эффективную инфраструктуру, SQLite может работать в крошечном объёме выделяемой для неё памяти, гораздо меньшем, чем в любых других системах БД [2].

Python имеет встроенную поддержку SQLite базы данных, для этого не нужно ничего дополнительно устанавливать, достаточно в скрипте указать импорт стандартной библиотеки.

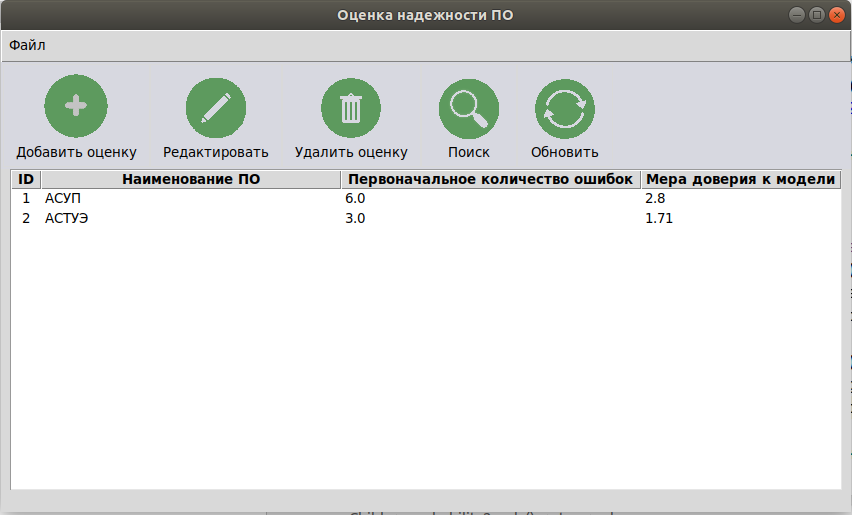
****

Рисунок 4. Окно программы по оценке надежности ПО

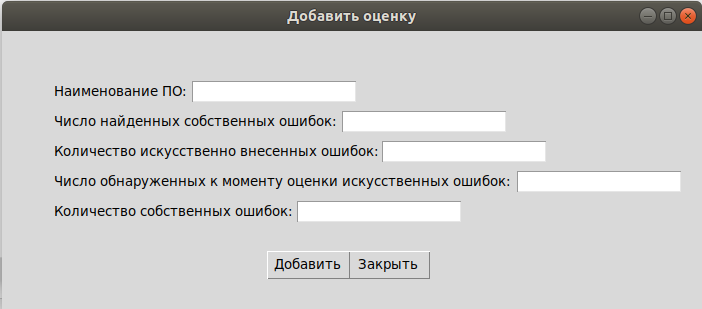
****

Рисунок 5. Форма для заполнения данных по оценке надежности ПО

## **Интеграция разработанных модулей в единое информационное пространство**

Разработана программа, позволяющая запустить любой из разработанных модулей.

В общую директорию были помещены файлы всех модулей и создан дополнительный файл для запуска окна выбора программы.

В самом начале выполнения программы появляется форма, где пользователю предлагается заполнить соответствующие поля необходимыми для расчета данными.

Программа считывает данные, после успешно пройденной проверки на корректность и полноту, программа начинает производить расчет по формулам.

После этого результаты выводятся в специально отведенные окна.

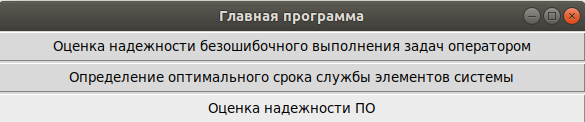
****

Рисунок 6. Окно объединенной программы

# Разработка структуры базы данных

Для сохранения результатов используется экспорт данных в базу SQLite.

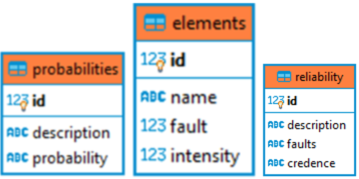


Рисунок 7. ER - диаграмма базы данных

Структура базы данных состоит из трех таблиц, по одной для каждого модуля.

## **Таблица Probabilities**

Включает в себя три поля:

* id – номер записи;
* description – поле для названия вида деятельности;
* probability – поле для записи рассчитанной вероятности.

## **Таблица Elements**

Включает в себя четыре поля:

* id – номер записи;
* name – поле для названия технологического оборудования;
* fault – поле для значения наработки на отказ;
* intensity – поле для номинальной интенсивности отказов.

## **Таблица Reliability**

Включает в себя четыре поля:

* id – номер записи;
* description – поле для наименования ПО;
* faults – поле для записи первоначального количества ошибок;
* credence – поле для записи рассчитанной меры доверия к модели.

# Заключение

Язык программирования Python на основе PyCharm способен реализовать все необходимые средства для расчета надежности программ.

В ходе выполнения работы были разработаны:

* модуль оценки надежности диспетчера, позволяющий пользователю на основе введенной информации (показатели, операции и т.д.) рассчитать вероятность безошибочного выполнения операций диспетчером;
* модуль оценки и прогнозирования надежности технологического оборудования, позволяющий создать план замены технологического оборудования;
* модуль оценки надежности ПО;

Также был проведен анализ предметной области, выявлены требования к разрабатываемой программе, было спроектировано и реализовано приложение.

# Литература

1. Васильев, А.Н. Python на примерах. Практический курс по программированию / А.Н. Васильев - М: Наука и техника, 2016 - 432 c.
2. Златопольский, Д.М. Основы программирования на языке Python / Д.М. Златопольский. - М.: ДМК Пресс, 2017. - 277 c.
3. Мэтиз Э. Изучаем Python. Программирование игр, визуализация данных, веб-приложения / Мэтиз Эрик. - М.: Питер, 2018. - 760 c
4. Шишмарёв, В.Ю. Надежность технических систем: учебник для вузов / В. Ю. Шишма рёв. — 2-е изд., испр. и доп. — Москва : Издательство Юрайт, 2020. — 289 с. — (Высшее образование).
5. Яхьяев Н.Я., Кораблин А.В. Основы теории надежности. Учебник для вузов. — М.: Академия, 2014. — 208 с.

# Приложение

В данном разделе представлен листинг основных функций расчета каждого модуля и листинг подключения к базе данных.

Функция расчета вероятности безошибочного выполнения операций оператором:

**def** probability\_culc(self):  
 **try**:  
 pisp = float(self.entry\_probability\_1.get()) \* float(self.entry\_probability\_2.get()) \* float(self.entry\_probability\_3.get())  
 pop = math.exp(-(float(self.entry\_2.get()) / float(self.entry\_1.get()) \* float(self.entry\_3.get())) \* float(self.entry\_3.get()) \* float(self.entry\_4.get()))  
 **return** pop + (1 - pop) \* pisp  
 **except** ValueError:  
 tk.Label(text=**"Ошибка. Ввод некорректен."**)

Функция расчета оптимального срока службы элементов системы:

**def** solver():  
 **global** plan  
 plan = shema\_data  
 data = {}  
 number = len(shema\_data.keys())  
 p = float(ent\_reliability.get())  
 p\_i = p\*\*(1/number)  
 **for** link **in** shema\_data:  
 t = len(shema\_data[link])  
 **if** t == 1:  
 p\_i\_j = p\_i  
 result = - float(math.log(p\_i\_j) \* float(shema\_data[link][1][**'fault'**]))  
 plan[link][1][**'date'**] += datetime.timedelta(days=int(result))  
 **elif** t == 2:  
 p\_i\_j = (2 - (4-4\*p\_i)\*\*(1/2))/2  
 result\_1 = - float(math.log(p\_i\_j) \* float(shema\_data[link][1][**'fault'**]))  
 result\_2 = - float(math.log(p\_i\_j) \* float(shema\_data[link][2][**'fault'**]))  
 plan[link][1][**'date'**] += datetime.timedelta(days=int(result\_1))  
 plan[link][2][**'date'**] += datetime.timedelta(days=int(result\_2))  
 **elif** t == 3:  
 p\_i\_j\_3 = (2 - (4-4\*p\_i)\*\*(1/2))/2  
 p\_i\_j = ((2 - (4-4\*p\_i)\*\*(1/2))/2)\*\*(1/2)  
 result\_1 = - float(math.log(p\_i\_j) \* float(shema\_data[link][1][**'fault'**]))  
 result\_2 = - float(math.log(p\_i\_j) \* float(shema\_data[link][2][**'fault'**]))  
 result\_3 = - float(math.log(p\_i\_j\_3) \* float(shema\_data[link][3][**'fault'**]))  
 plan[link][1][**'date'**] += datetime.timedelta(days=int(result\_1))  
 plan[link][2][**'date'**] += datetime.timedelta(days=int(result\_2))  
 plan[link][3][**'date'**] += datetime.timedelta(days=int(result\_3))  
 **elif** t == 4:  
 p\_i\_j = ((2 - (4-4\*p\_i)\*\*(1/2))/2)\*\*(1/2)  
 result\_1 = - float(math.log(p\_i\_j) \* float(shema\_data[link][1][**'fault'**]))  
 result\_2 = - float(math.log(p\_i\_j) \* float(shema\_data[link][2][**'fault'**]))  
 result\_3 = - float(math.log(p\_i\_j) \* float(shema\_data[link][3][**'fault'**]))  
 result\_4 = - float(math.log(p\_i\_j) \* float(shema\_data[link][4][**'fault'**]))  
 plan[link][1][**'date'**] += datetime.timedelta(days=int(result\_1))  
 plan[link][2][**'date'**] += datetime.timedelta(days=int(result\_2))  
 plan[link][3][**'date'**] += datetime.timedelta(days=int(result\_3))  
 plan[link][4][**'date'**] += datetime.timedelta(days=int(result\_4))  
 **for** label **in** frm\_4.grid\_slaves():  
 **if** int(label.grid\_info()[**"row"**]) >= 0:  
 label.destroy()  
 **for** link **in** plan:  
 **for** element **in** plan[link]:  
 **if** element == 1:  
 lbl = tk.Label(master=frm\_4, text=**f"Звено №\_{**link**} элемент №\_{**element**} Название: {**plan[link][element][**'name'**]**}: Дата замены: {**plan[link][element][**'date'**].strftime(**'%d.%m.%y'**)**}"**)  
 lbl.grid(row=link, column=element, padx=5, pady=5, sticky=**'n'**)  
 **else**:  
 lbl = tk.Label(master=frm\_4, text=**f"элемент №\_{**element**} Название: {**plan[link][element][**'name'**]**}: Дата замены: {**plan[link][element][**'date'**].strftime(**'%d.%m.%y'**)**}"**)  
 lbl.grid(row=link, column=element, padx=5, pady=5, sticky=**'n'**)

Функция расчета первоначального количества ошибок:

**def** faults\_culc(self):  
 **try**:  
 numerator = float(self.entry\_1.get()) \* float(self.entry\_2.get())  
 denominator = float(self.entry\_3.get())  
 **return** round(numerator / denominator, 2)  
 **except** ValueError:  
 tk.Label(text=**"Ошибка. Ввод некорректен."**)

Функция расчета меры доверия к модели:

**def** credence\_culc(self):  
 **try**:  
 **if** float(self.entry\_1.get()) \* float(self.entry\_2.get()) == float(self.entry\_3.get()):  
 numerator = float(self.entry\_2.get())  
 denominator = float(self.entry\_2.get()) + float(self.entry\_4.get()) + 1  
 **return** round(numerator / denominator, 2)  
 **else**:  
 numerator = float(self.entry\_2.get()) / (float(self.entry\_3.get()) - 1)  
 denominator = (float(self.entry\_2.get()) + float(self.entry\_4.get()) + 1) / (float(self.entry\_3.get()) + float(self.entry\_4.get()))  
 **return** round(numerator / denominator, 2)  
 **except** ValueError:  
 tk.Label(text=**"Ошибка. Ввод некорректен."**)

Функция добавления записи в базу данных:

**def** add():  
 sql = **f"INSERT INTO elements (name, fault, intensity) VALUES ('{**entry\_1.get()**}', {**entry\_2.get()**}, {**entry\_3.get()**})"** conn = sqlite3.connect(**'database.db'**)  
 data = conn.execute(sql)  
 conn.commit()  
 conn.close()  
 window.quit()

Функция обновления сохраненной информации:

**def** update\_record(self, description, faults, credence):  
 self.db.c.execute(**'''UPDATE reliability SET description=?, faults=?, credence=? WHERE ID=?'''**,  
 (description, faults, credence, self.tree.set(self.tree.selection()[0], **'#1'**)))  
 self.db.conn.commit()  
 self.view\_records()

Функция чтения данных из базы:

**def** view\_records(self):  
 self.db.c.execute(**'''SELECT \* FROM reliability'''**)  
 [self.tree.delete(i) **for** i **in** self.tree.get\_children()]  
 [self.tree.insert(**''**, **'end'**, values=row) **for** row **in** self.db.c.fetchall()]

Функция удаления элемента из базы данных:

**def** delete\_records(self):  
 **for** selection\_item **in** self.tree.selection():  
 self.db.c.execute(**'''DELETE FROM reliability WHERE id=?'''**, (self.tree.set(selection\_item, **'#1'**),))  
 self.db.conn.commit()  
 self.view\_records()

Функция поиска элемента в базе данных:

**def** search\_records(self, description):  
 description = (**'%'** + description + **'%'**,)  
 self.db.c.execute(**'''SELECT \* FROM reliability WHERE description LIKE ?'''**, description)  
 [self.tree.delete(i) **for** i **in** self.tree.get\_children()]  
 [self.tree.insert(**''**, **'end'**, values=row) **for** row **in** self.db.c.fetchall()]

Функция добавления звена на схему надежности:

**def** add\_link():  
 link = number\_of\_link.get()  
 number\_of\_element\_1 = **None** name\_of\_element\_1 = **None** date\_of\_element\_1 = **None** number\_of\_element\_2 = **None** name\_of\_element\_2 = **None** date\_of\_element\_2 = **None** number\_of\_element\_3 = **None** name\_of\_element\_3 = **None** date\_of\_element\_3 = **None** number\_of\_element\_4 = **None** name\_of\_element\_4 = **None** date\_of\_element\_4 = **None  
 for** label **in** frm\_com.grid\_slaves():  
 **if** int(label.grid\_info()[**"row"**]) > 0 **and** int(label.grid\_info()[**"row"**]) < 5:  
 **if** int(label.grid\_info()[**"row"**]) == 1 **and** int(label.grid\_info()[**"column"**]) == 0:  
 number\_of\_element\_1 = int(label[**'text'**])  
 **if** int(label.grid\_info()[**"row"**]) == 1 **and** int(label.grid\_info()[**"column"**]) == 1:  
 name\_of\_element\_1 = label[**'text'**]  
 fault\_1 = h(name\_of\_element\_1)[2]  
 intensity\_1 = h(name\_of\_element\_1)[3]  
 **if** int(label.grid\_info()[**"row"**]) == 1 **and** int(label.grid\_info()[**"column"**]) == 3:  
 date\_of\_element\_1 = datetime.datetime.strptime(label.get(), **'%d.%m.%y'**)  
 **if** int(label.grid\_info()[**"row"**]) == 2 **and** int(label.grid\_info()[**"column"**]) == 0:  
 number\_of\_element\_2 = int(label[**'text'**])  
 **if** int(label.grid\_info()[**"row"**]) == 2 **and** int(label.grid\_info()[**"column"**]) == 1:  
 name\_of\_element\_2 = label[**'text'**]  
 fault\_2 = h(name\_of\_element\_2)[2]  
 intensity\_2 = h(name\_of\_element\_2)[3]  
 **if** int(label.grid\_info()[**"row"**]) == 2 **and** int(label.grid\_info()[**"column"**]) == 3:  
 date\_of\_element\_2 = datetime.datetime.strptime(label.get(), **'%d.%m.%y'**)  
 **if** int(label.grid\_info()[**"row"**]) == 3 **and** int(label.grid\_info()[**"column"**]) == 0:  
 number\_of\_element\_3 = int(label[**'text'**])  
 **if** int(label.grid\_info()[**"row"**]) == 3 **and** int(label.grid\_info()[**"column"**]) == 1:  
 name\_of\_element\_3 = label[**'text'**]  
 fault\_3 = h(name\_of\_element\_3)[2]  
 intensity\_3 = h(name\_of\_element\_3)[3]  
 **if** int(label.grid\_info()[**"row"**]) == 3 **and** int(label.grid\_info()[**"column"**]) == 3:  
 date\_of\_element\_3 = datetime.datetime.strptime(label.get(), **'%d.%m.%y'**)  
 **if** int(label.grid\_info()[**"row"**]) == 4 **and** int(label.grid\_info()[**"column"**]) == 0:  
 number\_of\_element\_4 = int(label[**'text'**])  
 **if** int(label.grid\_info()[**"row"**]) == 4 **and** int(label.grid\_info()[**"column"**]) == 1:  
 name\_of\_element\_4 = label[**'text'**]  
 fault\_4 = h(name\_of\_element\_4)[2]  
 intensity\_4 = h(name\_of\_element\_4)[3]  
 **if** int(label.grid\_info()[**"row"**]) == 4 **and** int(label.grid\_info()[**"column"**]) == 3:  
 date\_of\_element\_4 = datetime.datetime.strptime(label.get(), **'%d.%m.%y'**)  
 **if** number\_of\_element\_4:  
 shema\_data[int(link)] = {  
 number\_of\_element\_1: {**'name'**: name\_of\_element\_1, **'fault'**: fault\_1, **'intensity'**: intensity\_1, **'date'**: date\_of\_element\_1},  
 number\_of\_element\_2: {**'name'**: name\_of\_element\_2, **'fault'**: fault\_2, **'intensity'**: intensity\_2, **'date'**: date\_of\_element\_2},  
 number\_of\_element\_3: {**'name'**: name\_of\_element\_3, **'fault'**: fault\_3, **'intensity'**: intensity\_3, **'date'**: date\_of\_element\_3},  
 number\_of\_element\_4: {**'name'**: name\_of\_element\_4, **'fault'**: fault\_4, **'intensity'**: intensity\_4, **'date'**: date\_of\_element\_4}  
 }  
 start = int(link)  
 **for** label **in** frm\_2.grid\_slaves():  
 **if** int(label.grid\_info()[**"row"**]) == start:  
 label.destroy()  
 lbl\_number = tk.Label(master=frm\_2, text=**f'Номер звена: {**start**}'**)  
 lbl\_number.grid(row=start, column=0, sticky=**'e'**)  
 lbl\_name\_1 = tk.Label(master=frm\_2, text=**f"{**name\_of\_element\_1**}: дата установки: {**date\_of\_element\_1.strftime(**'%d.%m.%y'**)**}"**)  
 lbl\_name\_1.grid(row=start, column=1, sticky=**'n'**)  
 lbl\_name\_2 = tk.Label(master=frm\_2, text=**f"{**name\_of\_element\_2**}: дата установки: {**date\_of\_element\_2.strftime(**'%d.%m.%y'**)**}"**)  
 lbl\_name\_2.grid(row=start, column=2, sticky=**'n'**)  
 lbl\_name\_3 = tk.Label(master=frm\_2, text=**f"{**name\_of\_element\_3**}: дата установки: {**date\_of\_element\_3.strftime(**'%d.%m.%y'**)**}"**)  
 lbl\_name\_3.grid(row=start, column=3, sticky=**'n'**)  
 lbl\_name\_4 = tk.Label(master=frm\_2, text=**f"{**name\_of\_element\_4**}: дата установки: {**date\_of\_element\_4.strftime(**'%d.%m.%y'**)**}"**)  
 lbl\_name\_4.grid(row=start, column=4, sticky=**'n'**)  
 **elif** number\_of\_element\_3:  
 shema\_data[int(link)] = {  
 number\_of\_element\_1: {**'name'**: name\_of\_element\_1, **'fault'**: fault\_1, **'intensity'**: intensity\_1, **'date'**: date\_of\_element\_1},  
 number\_of\_element\_2: {**'name'**: name\_of\_element\_2, **'fault'**: fault\_2, **'intensity'**: intensity\_2, **'date'**: date\_of\_element\_2},  
 number\_of\_element\_3: {**'name'**: name\_of\_element\_3, **'fault'**: fault\_3, **'intensity'**: intensity\_3, **'date'**: date\_of\_element\_3}  
 }  
 start = int(link)  
 **for** label **in** frm\_2.grid\_slaves():  
 **if** int(label.grid\_info()[**"row"**]) == start:  
 label.destroy()  
 lbl\_number = tk.Label(master=frm\_2, text=**f'Номер звена: {**start**}'**)  
 lbl\_number.grid(row=start, column=0, sticky=**'e'**)  
 lbl\_name\_1 = tk.Label(master=frm\_2, text=**f"{**name\_of\_element\_1**}: дата установки: {**date\_of\_element\_1.strftime(**'%d.%m.%y'**)**}"**)  
 lbl\_name\_1.grid(row=start, column=1, sticky=**'n'**)  
 lbl\_name\_2 = tk.Label(master=frm\_2, text=**f"{**name\_of\_element\_2**}: дата установки: {**date\_of\_element\_2.strftime(**'%d.%m.%y'**)**}"**)  
 lbl\_name\_2.grid(row=start, column=2, sticky=**'n'**)  
 lbl\_name\_3 = tk.Label(master=frm\_2, text=**f"{**name\_of\_element\_3**}: дата установки: {**date\_of\_element\_3.strftime(**'%d.%m.%y'**)**}"**)  
 lbl\_name\_3.grid(row=start, column=3, sticky=**'n'**)  
 **elif** number\_of\_element\_2:  
 shema\_data[int(link)] = {  
 number\_of\_element\_1: {**'name'**: name\_of\_element\_1, **'fault'**: fault\_1, **'intensity'**: intensity\_1, **'date'**: date\_of\_element\_1},  
 number\_of\_element\_2: {**'name'**: name\_of\_element\_2, **'fault'**: fault\_2, **'intensity'**: intensity\_2, **'date'**: date\_of\_element\_2}  
 }  
 start = int(link)  
 **for** label **in** frm\_2.grid\_slaves():  
 **if** int(label.grid\_info()[**"row"**]) == start:  
 label.destroy()  
 lbl\_number = tk.Label(master=frm\_2, text=**f'Номер звена: {**start**}'**)  
 lbl\_number.grid(row=start, column=0, sticky=**'e'**)  
 lbl\_name\_1 = tk.Label(master=frm\_2, text=**f"{**name\_of\_element\_1**}: дата установки: {**date\_of\_element\_1.strftime(**'%d.%m.%y'**)**}"**)  
 lbl\_name\_1.grid(row=start, column=1, sticky=**'n'**)  
 lbl\_name\_2 = tk.Label(master=frm\_2, text=**f"{**name\_of\_element\_2**}: дата установки: {**date\_of\_element\_2.strftime(**'%d.%m.%y'**)**}"**)  
 lbl\_name\_2.grid(row=start, column=2, sticky=**'n'**)  
 **elif** number\_of\_element\_1:  
 shema\_data[int(link)] = {  
 number\_of\_element\_1: {**'name'**: name\_of\_element\_1, **'fault'**: fault\_1, **'intensity'**: intensity\_1, **'date'**: date\_of\_element\_1}  
 }  
 start = int(link)  
 **for** label **in** frm\_2.grid\_slaves():  
 **if** int(label.grid\_info()[**"row"**]) == start:  
 label.destroy()  
 lbl\_number = tk.Label(master=frm\_2, text=**f'Номер звена: {**start**}'**)  
 lbl\_number.grid(row=start, column=0, sticky=**'e'**)  
 lbl\_name\_1 = tk.Label(master=frm\_2, text=**f"{**name\_of\_element\_1**}: дата установки: {**date\_of\_element\_1.strftime(**'%d.%m.%y'**)**}"**)  
 lbl\_name\_1.grid(row=start, column=1, sticky=**'n'**)