# Разработка ПО управления системой неразрушающего контроля

Сатановский А.Д.

СПбГУ, Кафедра системного программирования

**Научный руководитель:** Луцив Д.В.
**Рецензент:** Попович И.В.

17 мая 2025 г.

# План презентации

1. Введение в неразрушающий контроль: определение, цели и области применения
2. Цель и задачи
3. Требования, архитектура, алгоритмы, модули системы
4. Результаты и дальнейшая работа

# Введение в неразрушающий контроль: определение, цели и области применения

**Неразрушающий контроль** – это метод контроля надёжности основных рабочих свойств и параметров объекта или его отдельных элементов без необходимости вывода объекта из эксплуатации или его демонтажа.

**Целью** использования неразрушающего контроля в промышленности является надёжное выявление опасных дефектов изделий

**Область применения:** промышленность

Одним из перспективных направлений является радиографический метод неразрушающего контроля, который активно развивается на базе НИПК Электрон

# Цель и задачи

**Цель работы** – разработка и реализация архитектуры программного обеспечения для системы управления неразрушающим контролем, включающей модули позиционирования, безопасности и пульта управления.

**Основные задачи:**

1. Разработка архитектурных решений: системы позиционирования, системы безопасности, пульта управления
2. Выбор протоколов взаимодействия между компонентами системы (клиент-сервер-контроллер)
3. Подбор библиотек для разработки
4. Реализация архитектуры и бизнес-логики
5. Тестирование (ручное и автоматизированное с использованием Gitlab CI/CD)
6. Создание пакетов для развертывания

# Требования

**Функциональные требования**

- Автоматический запуск модуля при включении питания с индикацией готовности
- Прием, обработка и ответ на команды от клиентских приложений
- Унифицированный формат сообщений об ошибках
- Регулярная рассылка статуса системы подключенным клиентам, включая:
  - Состояние контроллеров
  - Текущие координаты
  - Статус выполнения команд

# Требования

**Нефункциональные требования**

- **Отказоустойчивость**: Каждый модуль работает на отдельном Raspberry Pi
- **Надежность**: Ведение детального журнала событий
- **Гибкость**: Настройка системы через конфигурационные файлы
- **Стандартизация**: Взаимодействие по TCP с использованием формата Gcode Marlin
- **Очередность обработки**: Запросы от нескольких клиентов обрабатываются последовательно
- **Автозапуск**: Управление запуском через supervisor

# Разработка и архитектура

**Выбор протоколов взаимодействия**

- Клиент - Сервер TCP + Gcode Marlin (Ethernet)
- Сервер - Контроллер Modbus RTU (Serial Port RS485)

Пример Gcode команды для системы позиционирования на уровне TCP:

- **G1 F100 X50.5 Y-25.3 Z22.4**

На уровне Modbus RTU транслируется в пакеты:

- Установка скорости 100 мм/с для всех осей
- Отправка команд относительного перемещения:
  - Ось X: +50.5 мм
  - Ось Y: -25.3 мм
  - Ось Z: +22.4 мм

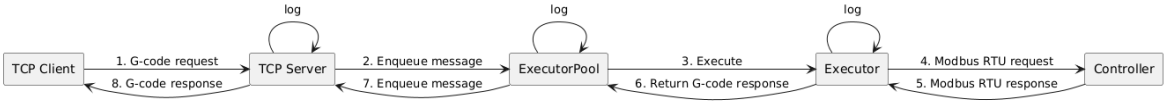
# Разработка и архитектура

Raspberry Pi4 + Ubuntu 22.04. Разработка ведется на C++

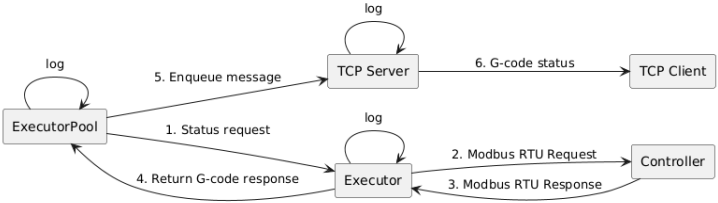
**Используемые библиотеки**:

- nlohmann/json – для файлов конфигурации
- gpr – парсер gcode
- DFHack/clsocket – sockets
- qt-logger – логгирование
- googletest unit-тестирование

# Диаграмма выполнения команды

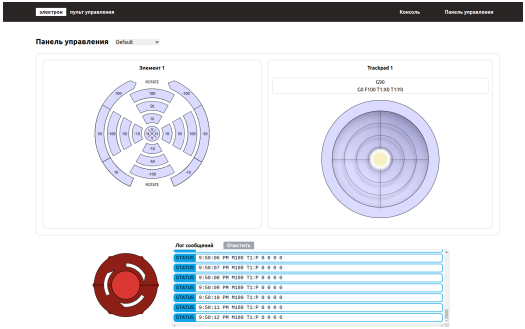


# Диаграмма получения статуса



# Пульт управления

(a) **Аппаратный пульт управления системой позиционирования**




(b) **Программный пульт управления**

# Алгоритмы

**Проблема:** особенности аппаратуры порождают различные неточности

1. Удержание точки интереса
2. Контроль рабочих пределов
3. Повышение точности позиционирования

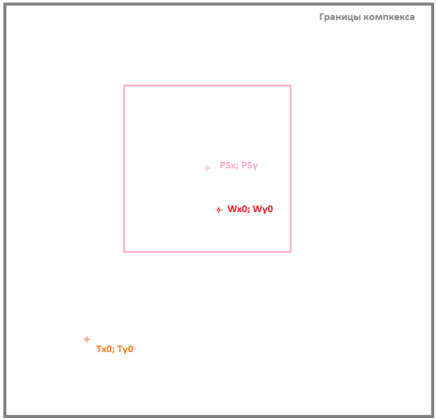
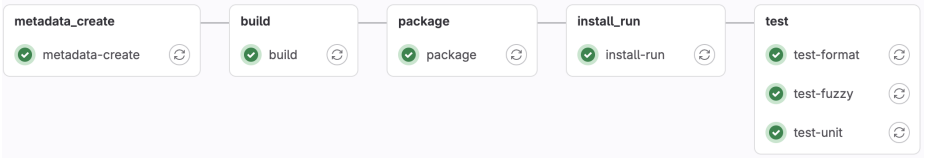



Рис.: **Системы координат комплекса AXI.**

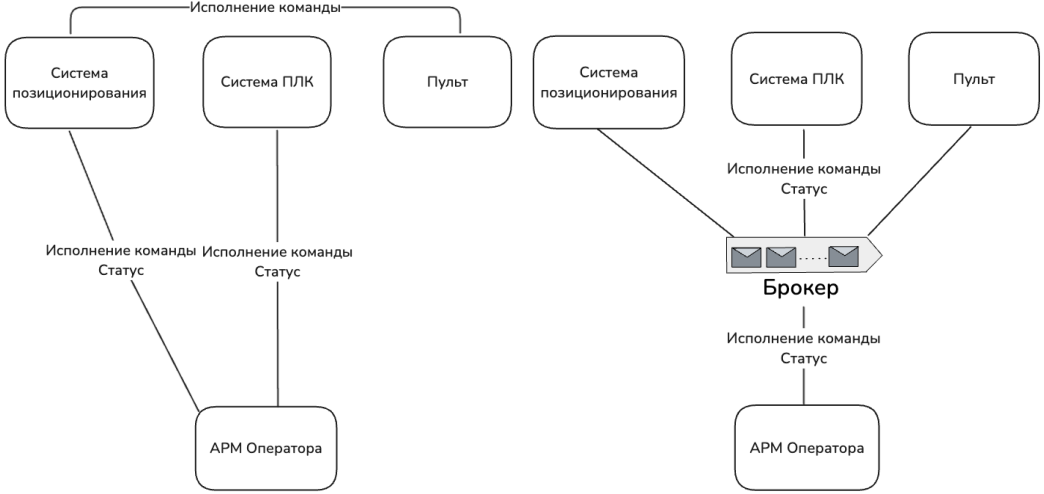
# GitLab CI/CD




(a) **Пайплайн сборки, пакетирования, тестирования**

- Создание версии пакета
- Сборка
- Пакетирование
- Проверка установки и запуска
- Тестирование на формат сообщений
- Нагрузочное тестирование
- Unit-тестирование

# В процессе




(a) **Старая архитектура** (b) **Новая архитектура**

# Дальнейшая работа

- **Модернизация архитектуры:** внедрение брокера сообщений, создание библиотеки общего кода
- **Повышение качества кода:** тестирование с санитайзерами, увеличение покрытия unit-тестами, профилирование производительности
- **Инфраструктурные улучшения:** автоматическое развертывание на стендах, организация хранилища пакетов, улучшение CI/CD пайплайнов

# Результаты

- **Реализована система позиционирования:** основные функции управления перемещением, регулярное тестирование на рабочих стендах, более 150 Unit тестов
- **Реализованы и протестированы компоненты:** сервер пульта управления (интегрирован с системой позиционирования), сервер системы безопасности
- **Внедрены алгоритмы управления:** удержание точки интереса, контроль рабочих пределов, повышение точности позиционирования
- **Разработан веб-пульт для тестирования:** отправка команд управления, получение и отображение статуса системы, переключение между модулями по IP-адресу
- **Проведена оптимизация архитектуры:** выделен базовый класс для работы с TCP-соединениями
- **Настроена CI/CD система:** автоматическая сборка и тестирование, генерация установочных пакетов, поддержка разных архитектур (x86_64, arm64)