

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение  
высшего образования

**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ  
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**

УТВЕРЖДАЮ

Зав.кафедрой,

к. ф.-м. н.

\_\_\_\_\_ М. В. Огнева

## **ОТЧЕТ О ПРАКТИКЕ**

студента 2 курса 273 группы КНиИТ

Пронина Антона Алексеевича

вид практики: производственная распределенная (научно-исследовательская  
работа)

кафедра: Кафедра информатики и программирования

курс: 2

семестр: 3

продолжительность: 18 нед., с 01.09.2023 г. по 15.01.2024 г.

Руководитель практики от университета,

старший преподаватель

\_\_\_\_\_

Е. Е. Лапшева

Тема практики: «Разработка системы программирования “КуМир роботы”»

## СОДЕРЖАНИЕ

ВВЕДЕНИЕ .....	4
1 Предметная область. Подходы, понятия, средства .....	6
1.1 Среда исполнения и язык программирования КуМир .....	6
1.2 Архитектура системы программирования КуМир .....	7
1.3 Обновление зависимостей проекта .....	12
2 Аппаратно-программный комплекс Arduino .....	17
2.1 Робототехника в школе .....	17
2.2 Анализ инструкций и плат Arduino .....	18
2.3 Программные средства взаимодействия с платами Arduino .....	20
2.4 Система программирования “КуМир роботы” .....	24
ЗАКЛЮЧЕНИЕ .....	28
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ .....	30
Приложение А Исходный код dockerfile, унифицирующий настройки сре- ды запуска системы тестирования .....	34
Приложение Б Исходный код скрипта, производящего тестирование транс- лятора .....	36

## ВВЕДЕНИЕ

Одним из наиболее интересных вопросов, требующих особого внимания в обучении информатике, является вопрос системы обучения программированию. Это связано с тем, что профессия специалистов в области информатики и информационных технологий в какой-то мере начинается со школы. Одним из прямых приложений программирования является робототехника.

Ввиду растущего интереса в сфере образования к обучению робототехнике в школе и широкой распространенности и глубокой степени интеграции современной образовательной системы с этим языком, появилась идея о разработке транслятора с языка КуМир в язык C++. Актуальность идеи заключается в снижении входного порога в область робототехники как со стороны ученика, предоставляя возможность, используя полученные навыки программирования в системе КуМир, заниматься разработкой роботов, так и со стороны преподавателя, уменьшая затраты на приобретение программных и аппаратных средств разработки. В качестве аппаратной платформы данной задачи был выбран электронный конструктор и удобная платформа быстрой разработки электронных устройств для новичков и профессионалов — платы Arduino, ввиду низкой стоимости устройств, периферийных модулей, простоты разработки аппаратных устройств на базе этих плат, высокой модульности систем и их высокой распространенности среди робототехников.

Транслятор с языка КуМир в язык C++ стал отправной точкой процесса создания комплекса аппаратно-программных и методических средств изучения робототехники в школе с использованием языка программирования КуМир. Важным аспектом комплекса является доступность и удобство изучения робототехники. Для повышения доступности изучения, было принято решение разработать отдельный клиент системы программирования КуМир “КуМир Ро-

боты”, направленный на изучение робототехники на основе плат Arduino.

Цель работы — разработка системы программирования роботов на основе плат Arduino на базе исходных кодов системы программирования КуМир. Для выполнения поставленной цели, требуется выполнить следующие задачи:

- проанализировать исходные коды системы программирования КуМир;
- проанализировать историю развития технологий, используемых для разработки системы программирования;
- доработка и устранение ошибок в исходном коде системы программирования КуМир;
- проанализировать системы программирования плат Arduino и их аналоги;
- разработать отдельный клиент для разработки роботов;
- обновить используемые библиотеки и фреймворки для разработки среды программирования;
- разработать плагин для взаимодействия с платами Arduino.

## **1 Предметная область. Подходы, понятия, средства**

### **1.1 Среда исполнения и язык программирования КуМир**

КуМир (Комплект Учебных МИРов) — система программирования, предназначенная для поддержки начальных курсов информатики и программирования в средней и высшей школе с открытым исходным кодом [1]. Спектр возможностей применения данной системы программирования ограничен, ряд стандартных команд позволяет разрабатывать небольшие приложения с целью изучения основ программирования на процедурных языках.

На данный момент язык КуМир — это язык, с которого хорошо начать, чтобы освоить основы алгоритмического подхода и процедурный стиль программирования. Система КуМир [2] в современном ее состоянии (с подсистемой ПиктоМир) состоит из расширяемого набора исполнителей (или роботов), набора систем программирования и вспомогательных утилит и программ. Расширяемый набор исполнителей (роботов) представляет собой отдельные самостоятельные программы и (или) электронно-механические устройства, имеющие собственное пультовое управление (интерфейс), а также возможность локального или сетевого управления из выполняющей системы. Набор систем программирования:

- система программирования КуМир на школьном алгоритмическом языке, состоящая из редактора-компилятора алгоритмического языка с многооконным интерфейсом, интегрированная с выполняющей системой. Система не является полноценным интерпретатором или компилятором школьного алгоритмического языка. Зачастую, программа на школьном алгоритмическом языке компилируется в некий промежуточный код (подобный подход был использован в языке Java), который затем интерпретируется и выполняется с автоматической генерацией точек останова по событи-

ям и шагам (при пошаговом выполнении). В системе программирования КуМир также отсутствует отладчик в его классическом понимании. Это имеет свои причины, так как сильно упрощает процесс освоения и написания системы программирования КуМир, а также отладку программ. Все изменения используемых в школьной программе величин автоматически визуализируются на полях программы. Визуализируются и результаты логических операций. Кроме того, к системе можно подключать любого исполнителя (робота) из набора и программно управлять им;

- система бестекстового, пиктографического программирования ПиктоМир. В ней учащийся собирает программу из команд исполнителя (робота). Управляющие конструкции языка представлены определенной параметрической организацией алгоритмов. Созданная учащимися программа при выполнении передает команды робота и получает его состояние;
- система программирования на производственном языке программирования C++, Python и т.п [3];

## **1.2 Архитектура системы программирования КуМир**

Помимо инструментов разработки, среда обладает рядом функциональных возможностей — просмотр документации, создание и выполнение практикумов, работа с окном в свернутом режиме и пр. Стартовое окно системы программирования КуМир представлено на рисунке 1.

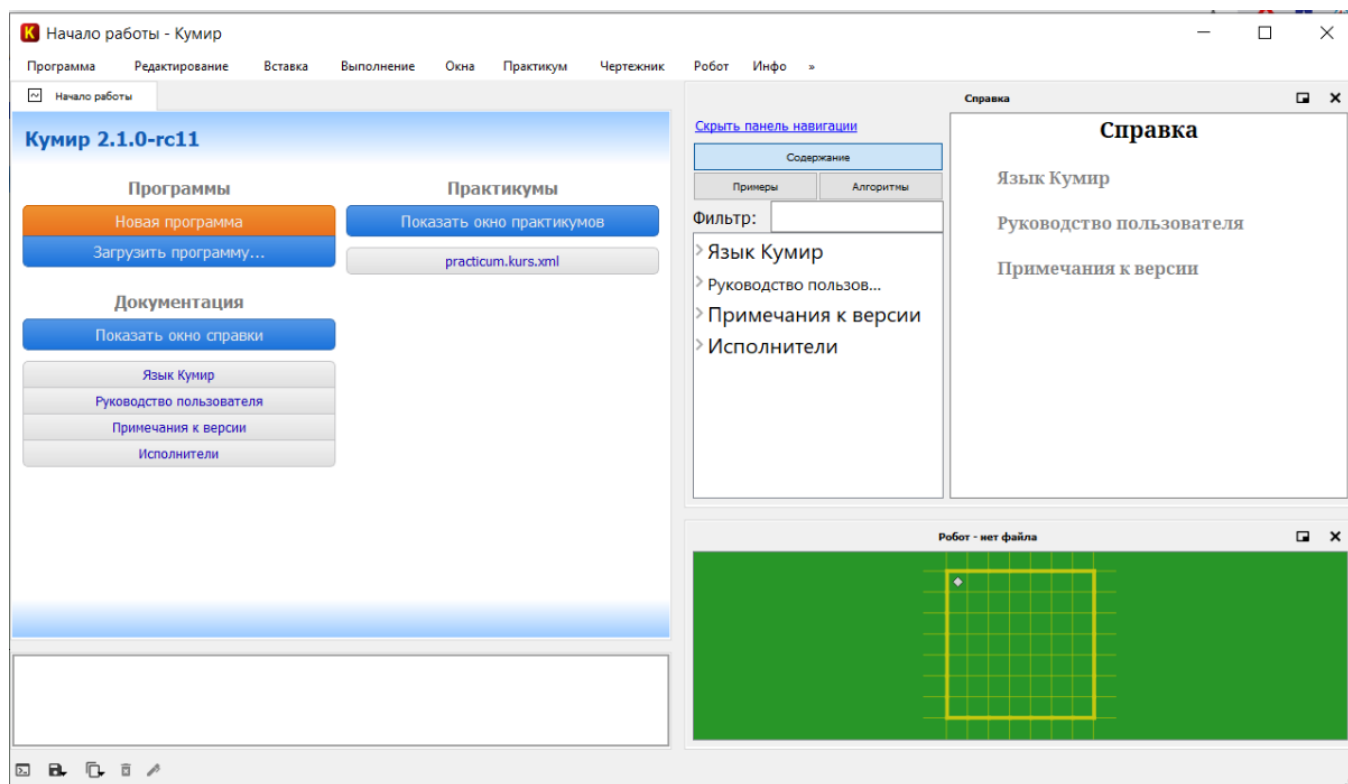


Рисунок 1 — Внешний вид среды программирования КуМир

С точки зрения кода, система программирования КуМир представляет собой монолитное приложение. Исходные коды разделены на 2 основные части:

- библиотеки;
- плагины.

Существует ряд клиентов среды программирования КуМир — для учителя, для ученика старших классов, стандартный клиент. Помимо клиентов, существуют отдельные приложения для компиляции и исполнения кода на языке программирования КуМир. Описанные выше приложения создаются при помощи разделяемого программного кода и вспомогательных инструментов.

К вспомогательным инструментам стоит отнести и ряд функций, упрощающих работу с системой сборки CMake [4]. В исходных кодах можно найти функции упрощающие поиск библиотек для фреймворка QT, определения и установки файлов ресурсов при сборке, а также функции для сборки артефактов приложения(клиентов) из исходных кодов.



Для создания нового артефакта - клиента или приложения командной строки используется CMake-функция *kumir2\_add\_launcher*. По созданному конфигурационному файлу CmakeLists.txt с описанием нового клиента по аналогии с имеющимися, во время сборки приложения, программа-сборщик обнаружит файл конфигурации и с его помощью создаст клиент. Пример конфигурационного файла CmakeLists.txt приведен в листинге 1:

```
1 project(kumir2-robots)
2 cmake_minimum_required(VERSION 3.0)
3
4 find_package(Kumir2 REQUIRED)
5
6 kumir2_add_launcher(
7     NAME            kumir2-robots
8     SPLASHSCREEN     "splashscreen-classic.png"
9     WINDOW_ICON      "window-icon-classic.png"
10    APP_ICON_NAME     "kumir2-classic"
11    X_ICONS_DIR        "../.../app_icons/linux/hicolor"
12    WIN_ICONS_DIR      "../.../app_icons/win32"
13    X_NAME             "Kumir Robots Edition"
14    X_NAME_ru          "Кумир для роботов"
15    X_CATEGORIES       "Education,X-KDE-Edu-Misc"
16    CONFIGURATION
17    "CourseManager,Editor,ActorArduino,
18    ArduinoCodeGenerator\ preload=Files\),
19    KumirAnalizer\ preload=Files\),*CodeGenerator,
20    KumirCodeRun(nobreakpoints),!CoreGUI\ (nosessions\)" )
```

Листинг 1 — Пример содержимого файла CMakeLists.txt для нового клиента системы программирования КуМир

Функция *kumir2\_add\_launcher* позволяет декларативно настроить результат

сборки — указать его название, иконку, а также ряд зависимостей для сборки приложения.

Клиент среды программирования состоит из плагинов и исполнителей. Плагины ссылаются на библиотеки КуМир-а, предоставляя конкретные реализации на основе контрактов [5], описываемых моделями библиотек.

Среди инструментов разработчика существуют скрипты для кодогенерации оснастки исполнителей, развертывания приложения в разных операционных системах и генерации CMake-скриптов.

Библиотеки представляют собой набор базовых сущностей, реализующих определенную часть функционала. В подавляющем большинстве, классы, описывающие область знания не содержат в себе логики работы с данной областью и представляют лишь анемичные модели [6], [7] для хранения состояния. Для работы с моделями используются генераторы или фабрики. Среди библиотек существует собственная реализация AST-дерева [8]. Библиотека содержит модели, описывающие выражения, типы, модули, алгоритмы, переменные и лексемы. Программный код, использующий данные модели представляет собой плагины для кодогенерации.

Кроме AST-деревьев, при помощи библиотек описаны:

- модели уведомлений об ошибках;
- модели для работы с XML-документацией;
- модели для работы с плагинами (базовый функционал, использующийся для создания конкретных реализаций);
- модели для работы с кодировками;
- промежуточные модели для работы с LLVM.

Описанные выше библиотеки используются для создания плагинов — основных блоков приложений. Система плагинов позволяет гибко конфигурировать и изменять набор пользовательских функций. Плагин может инкапсулировать

как функции по трансляции или компиляции языка КуМир в другой язык, так и графическую оболочку приложения, отдельные части его интерфейса.

Список функций, разделенных при помощи плагинов приведен ниже:

- графический интерфейс;
- режим учителя;
- статический анализ программного кода;
- генерация программного кода с языка программирования КуМир в платформу-ориентированный код;
- компиляция программного кода на языке программирования КуМир;
- запуск кода на языке программирования КуМир для выбранной архитектуры процессора и операционной системы;
- генерация программного кода с языка программирования КуМир в инструкции LLVM;
- поддержка языка программирования Python 3.

В ходе ВКР бакалавра был разработан плагин для генерации программного кода с языка программирования КуМир в язык программирования C++, а также исполнитель “Ардуино” [9], предоставляющий набор базовых операций для работы с платами Arduino. Эти доработки являются ключевыми составляющими для разработки отдельного клиента “КуМир Роботы”, предназначенного для программирования роботов на базе плат Arduino.

Анализ архитектуры проекта и имеющихся инструментов позволил определить необходимый функционал для разработки нового приложения. Первым шагом на пути разработки стал анализ зависимостей приложения и обновление основной зависимости — фреймворка QT.

### 1.3 Обновление зависимостей проекта

Обновление зависимостей проекта является важной частью его поддержки и сопровождения. Вот несколько причин, почему это необходимо:

1. исправление ошибок и уязвимостей: Одна из основных причин обновления зависимостей — исправление ошибок и уязвимостей в используемых библиотеках. Разработчики постоянно работают над улучшением своего программного обеспечения и выпускают исправления проблем, которые были обнаружены. Если вы не обновляете зависимости, ваш проект может содержать уязвимости, которые могут быть использованы злоумышленниками;
2. улучшение производительности и функционала: Обновление зависимостей может также привести к улучшению производительности вашего проекта и добавлению новых функций. Разработчики могут оптимизировать код, исправлять узкие места и добавлять новые возможности в новых версиях библиотек. Если вы не обновляете зависимости, вы можете упустить эти улучшения и новые возможности;
3. совместимость с другими зависимостями: Когда вы обновляете одну зависимость, это может потребовать обновления других зависимостей, чтобы сохранить совместимость между ними. Когда вы используете устаревшие версии библиотек, вы можете столкнуться с проблемами совместимости, которые могут привести к ошибкам или неожиданному поведению вашего проекта;
4. поддержка сообщества и документация: Обновление зависимостей помогает поддерживать активное сообщество вокруг вашего проекта. Разработчики библиотек постоянно работают над улучшением своего программного обеспечения и предоставляют обновления с новыми возможностями или исправлениями. Если вы не обновляете зависимости, вы можете

упустить эти улучшения и использовать менее поддерживаемые версии библиотек.

В целом, обновление зависимостей — важный элемент поддержки проекта. Оно помогает сохранить безопасность, улучшить производительность, добавить новые функции и поддерживать совместимость со сторонними библиотеками [10].

Система программирования КуМир имеет ряд зависимостей — библиотек и фреймворков, используемых для разработки. Основными являются:

- фреймворк QT;
- система сборки CMake;
- интерпретатор языка программирования Python;
- библиотека Zlib;
- библиотека boost;
- библиотека LLVM.

Фаза активной разработки среды и языка программирования КуМир была прекращена в 2014 году. Судя по состоянию репозитория с исходными кодами [11], последние изменения, вносимые в код лишь исправляют небольшие недочеты и недоработки. На данный момент, развитие языка программирования КуМир приостановлено, изменения, в корне меняющие функционал системы программирования не планируются [12]. Большинство исходного кода написано с использованием фреймворка QT, четвертой версии. Поддержка данной версии была прекращена в 2015 году и признана устаревшей [13].

Установка данной версии фреймворка проблематична ввиду отсутствия пакетированных образов. Альтернативой поиску необходимой зависимости является сборка пакета из исходных кодов. Оба рассмотренных выше варианта позволят безопасно разрабатывать приложение, с минимальным риском возникновения ошибок в результате сборки. Однако, использование неактуальных

версий зависимостей предполагает наличие потенциальных уязвимостей, проблем с развертыванием приложений на устройствах потребителей, а также ряд других уязвимостей связанных с прекращением поддержки [14].

Для разработки отдельного клиента с новым функционалом для программирования роботов на базе исходных кодов системы программирования КуМир, потребовалось произвести миграцию исходных кодов на поддерживаемую версию фреймворка *QT* – *QT5.15*. Среди основных сложностей при миграции стоит выделить следующие:

- изменение программного интерфейса фреймворка;
- несовместимость части библиотек с новой версией;
- несовместимость с системой сборки.

После изменения версии используемого фреймворка на более современную, в первую очередь пришлось решать проблемы с системой сборки. В новой версии фреймворка разработчики приняли решение об изменении подхода к именованию основных строительных блоков и изменению пакета, содержащего данный функционал.

Далее возник ряд предупреждений об использовании устаревших инструкций, например, в версии фреймворка отказались от использования отдельных перечислений, решив использовать одно общее перечисление [15], описывающее широкий спектр свойств. Иными словами, вместо *QT :: Color :: Blue*, теперь стоит использовать *QT :: Blue*. Подобные недочеты составили 70% всех изменений, внесенных при обновлении зависимости проекта.

Серьезной проблемой стало обновление модуля для работы с пользовательской документацией, отображенной на рисунке 2.

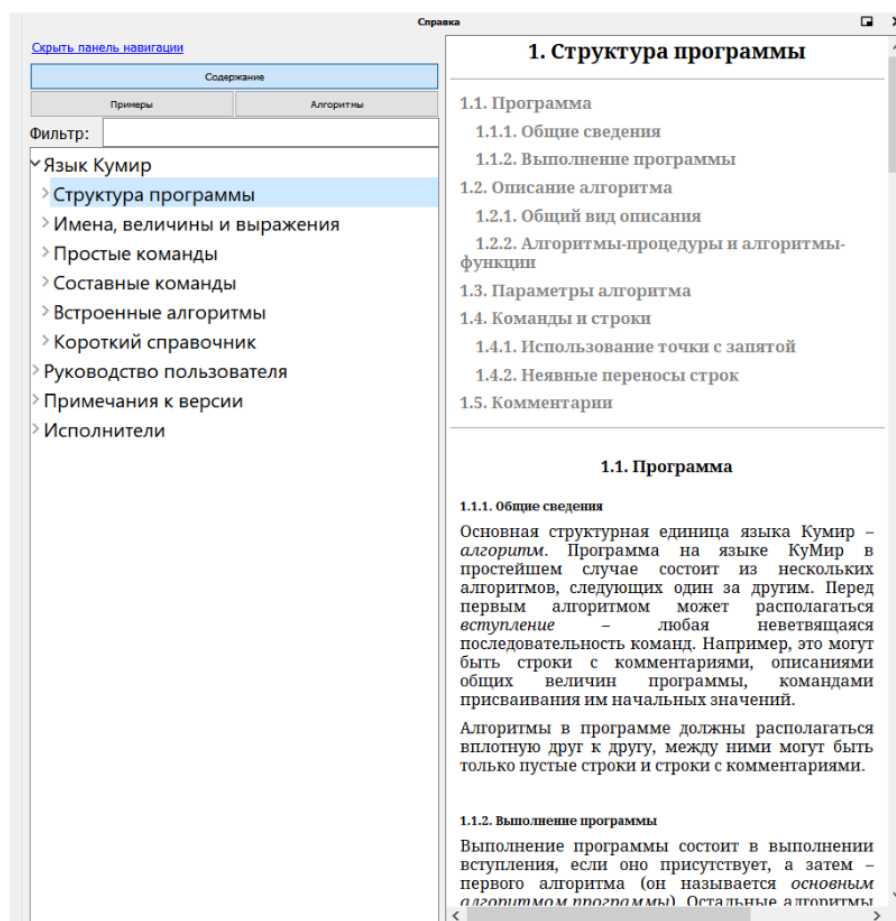


Рисунок 2 — Внешний вид плагина для работы с документацией

Файлы с документацией представляют собой xml-файлы. Основная сущность, используемая для разбора этих файлов была помечена устаревшей в новой версии фреймворка. Ввиду кроссплатформенности приложения, код парсинга xml-документов различался для различных операционных систем. Последней сложностью, возникшей при миграции стало изменение кода обработки пользовательского ввода с клавиатуры. При изучении функционала системы, собранной из исходных кодов, периодически возникала ошибка, приводящая к закрытию приложения. Профилирование приложения не позволило определить причину ошибки. Проблемным местом оказался модуль обработки пользовательского ввода с клавиатуры. Код модуля использовал библиотеку X11 [16].

В результате обновления миграции приложения удалось повысить доступность приложения для современных систем, повысить его надежность, а также

упростить дальнейшую разработку решив проблему совместимости версий зависимостей проекта.

Миграция приложения на новую версию фреймворка стала отправной точкой в процессе разработки отдельного клиента для программирования роботов. Была создана отдельная ветка в репозитории, где была произведена миграция и началась разработка приложения “КуМир Роботы”. Помимо создания конфигурации и определения необходимых зависимостей для приложения, был разработан плагин для работы с платами Arduino. Перед разработкой данного расширения был произведен анализ существующих средств аппаратной обработки плат Arduino.



## **2 Аппаратно-программный комплекс Arduino**

### **2.1 Робототехника в школе**

Arduino — комплекс аппаратно-программных средств с открытым исходным кодом, обладающий низким порогом вхождения как со стороны программирования, так и со стороны электроники. Электронные платы способны считывать и генерировать входные и выходные сигналы. Для передачи последовательности команд управления плате используется язык программирования Arduino и соответствующая среда исполнения. Многолетняя история развития платформы Arduino включает в себя тысячи проектов — от небольших устройств до крупных научных инструментов. Сообщество разработчиков данной платформы состоит из программистов различного уровня — от любителей до профессионалов, собравших и систематизировавших свой опыт взаимодействия с Arduino для помощи в вопросах разработки проектов, полезный как новичкам, так и профессионалам.

Использование платформы Arduino в педагогической деятельности открывает новые возможности для студента и школьника. Проекты, реализуемые в учебном процессе в образовательных учреждениях, могут выполняться и развиваться дома.

В последние годы во многих школах активно организуется основная и дополнительная образовательная деятельность учащихся, связанная с освоением элементов робототехники [17]. Использование платформы Arduino для образовательных учреждений позволяет получить возможность развить навыки программирования на практике, а также освоить азы схемотехники, дает возможность обучающимся освоить основные приемы разработки аппаратной и программной части автономных автоматизированных комплексов. Исследователи вопроса актуальности изучения робототехники на базе платформы Arduino

в образовательном процессе отмечают повышение креативности учащихся [18], активное формирование и оттачивание профессиональных навыков и умений, а также развиваются и осваиваются компетенции естественно-научной направленности, связанные с физико-техническими дисциплинами [19].

## 2.2 Анализ инструкций и плат Arduino

Семейство плат Arduino включает в себя несколько моделей, которые отличаются по характеристикам, возможностям и цене:

- **Arduino Uno:** Это самая популярная и распространенная модель платы Arduino. Она оснащена микроконтроллером ATmega328P и имеет несколько цифровых и аналоговых входов/выходов. На плате есть разъемы для подключения сенсоров, дисплеев, моторов и других компонентов. Arduino Uno имеет простой интерфейс, что делает его идеальным для начинающих. Платы полностью совместимы с наборами плат расширения, позволяющими увеличить спектр возможностей применения платы — от добавления ethernet до портативного зарядного устройства с аккумулятором;
- **Arduino Mega:** Этот микроконтроллер имеет более мощный процессор ATmega2560, обладает большим количеством входов/выходов и памяти. Arduino Mega обычно используется для более сложных проектов, требующих большего количества подключенных компонентов;
- **Arduino Leonardo/Arduino Micro:** Эти модели оснащены микроконтроллерами ATmega32U4 и обладают возможностью эмуляции устройств USB, таких, как клавиатура или мышь. Arduino Leonardo/Arduino Micro обычно используются для создания интерактивных устройств, таких как игровые контроллеры;
- **Arduino Nano:** Эта небольшая и компактная модель имеет те же характеристики, что и Arduino Uno, но в более удобном форм-факторе. Arduino

Nano обычно используется в проектах, где необходимо сохранить место на плате. Пригодятся, если вам нужно собрать очень компактное устройство;

- **Arduino Due:** Эта модель оснащена микроконтроллером ARM Cortex-M3, который позволяет разработчикам выполнять сложные вычисления и работать с более высокой производительностью. Arduino Due часто используется для создания проектов, требующих большой вычислительной мощности.

Кроме вышеперечисленных моделей, существуют и другие варианты, такие как Arduino Yun, Arduino Nano Every, Arduino MKR и другие. Каждая из них имеет свою спецификацию и предназначение, что позволяет выбрать подходящую модель в зависимости от конкретных потребностей и требований вашего проекта.

Семейства плат отличаются друг от друга рядом технических характеристик, а именно:

- **микроконтроллер:** Каждая плата Arduino основана на определенном микроконтроллере, таком как ATmega328P, ATmega2560, ATmega32U4 и других. Разные микроконтроллеры имеют разные характеристики, такие как частота работы, объем памяти и количество входов/выходов;
- **форм фактор:** Существует несколько различных форм-факторов плат Arduino, таких, как Arduino Uno, Arduino Mega, Arduino Nano и другие. Каждая плата имеет свои уникальные размеры и разъемы для подключения различных модулей и дополнительных компонентов;
- **функциональность:** В разных семействах плат Arduino могут быть различные функциональные возможности. Например, некоторые платы могут иметь встроенный WiFi или Bluetooth модуль, возможность работы с сенсорами или поддержку расширений для специфических задач;
- **напряжение питания:** Некоторые платы могут работать от батарейного питания или низкого напряжения, в то время как другие могут требовать

более высокого напряжения;

- цена: Разные платы Arduino имеют разные цены в зависимости от их характеристик и функциональности. Некоторые платы могут быть более дорогими из-за дополнительных функций или встроенных модулей.

ля процессоров с различной архитектурой процесс прошивки будет отличаться — для каждой архитектуры имеется своя программа, осуществляющая прошивку платы. Несмотря на это, программирования плат Arduino осуществляется при помощи скетчей — программ на специализированном языке программирования Arduino. Язык является подмножеством языка C++ и содержит набор базовых, кроссплатформенных функций.

В качестве целевой платформы были выбраны платы Arduino Uno ввиду широкой распространенности в сообществе, низкой стоимости и простоты использования.

### **2.3 Программные средства взаимодействия с платами Arduino**

При работе с платами Arduino вопросы, касающиеся работы с платами решаются при помощи Arduino IDE. Интерфейс приложения показан на рисунке 3.

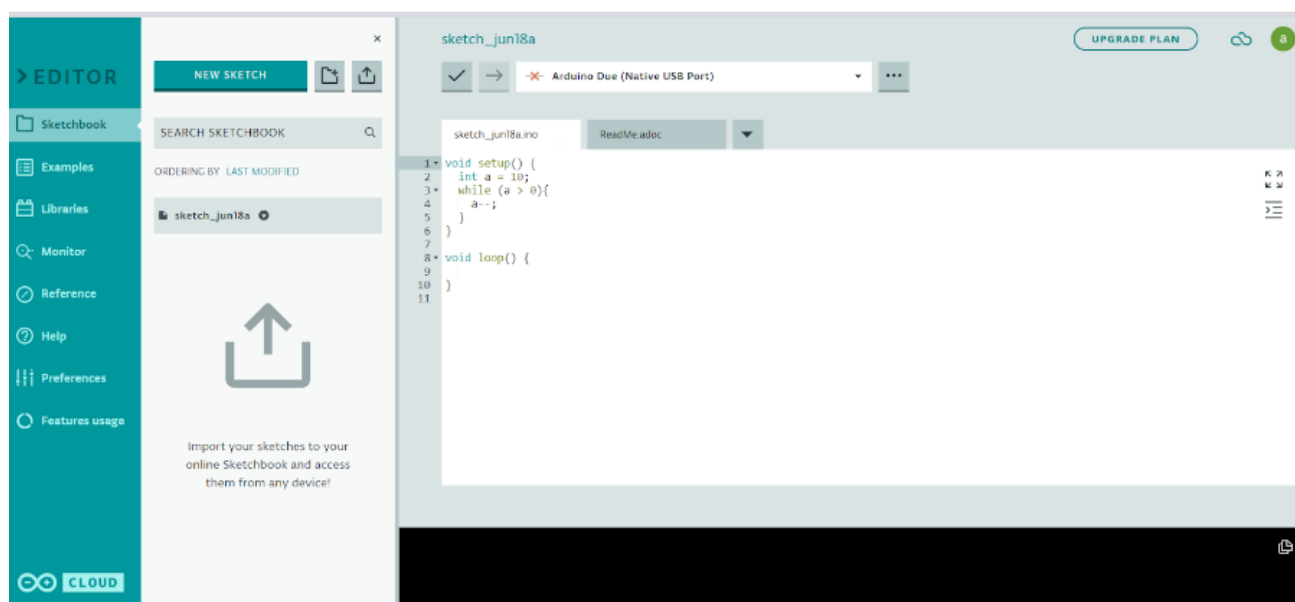


Рисунок 3 — Интерфейс программы Arduino IDE

Для передачи последовательности команд управления плате используется язык программирования Arduino и соответствующая среда исполнения. Программное обеспечение может быть легко использовано на любой операционной системе [20].

На текущий момент существует две версии среды программирования Arduino IDE. Среда программирования позволяет автоматически обнаруживать подключенные к устройству платы, производить их прошивку, а также обрабатывать поступающие с подключенной платы сигналы в реальном времени. Приложение портировано на все операционные системы, программный код для работы с платами имеет общую реализацию, учитывающую различия операционных систем, имеющие значительное влияние на работу с платами.

Исходный код системы программирования Arduino IDE как первой [21], так и второй [22] версии открыт для просмотра и редактирования. Для анализа исходных кодов было решено выбрать более современную версию приложения и рассмотреть алгоритмы взаимодействия приложения с платами.

Вторая версия приложения разработана при помощи языка программиро-

вания Typescript [23] и библиотеки react-js [24]. Приложение может работать как в браузере, так и локально на устройстве пользователя.

При авторизации в приложении у пользователя появляется возможность установить расширение для браузера, как показано на рисунке 4 и начать работу.

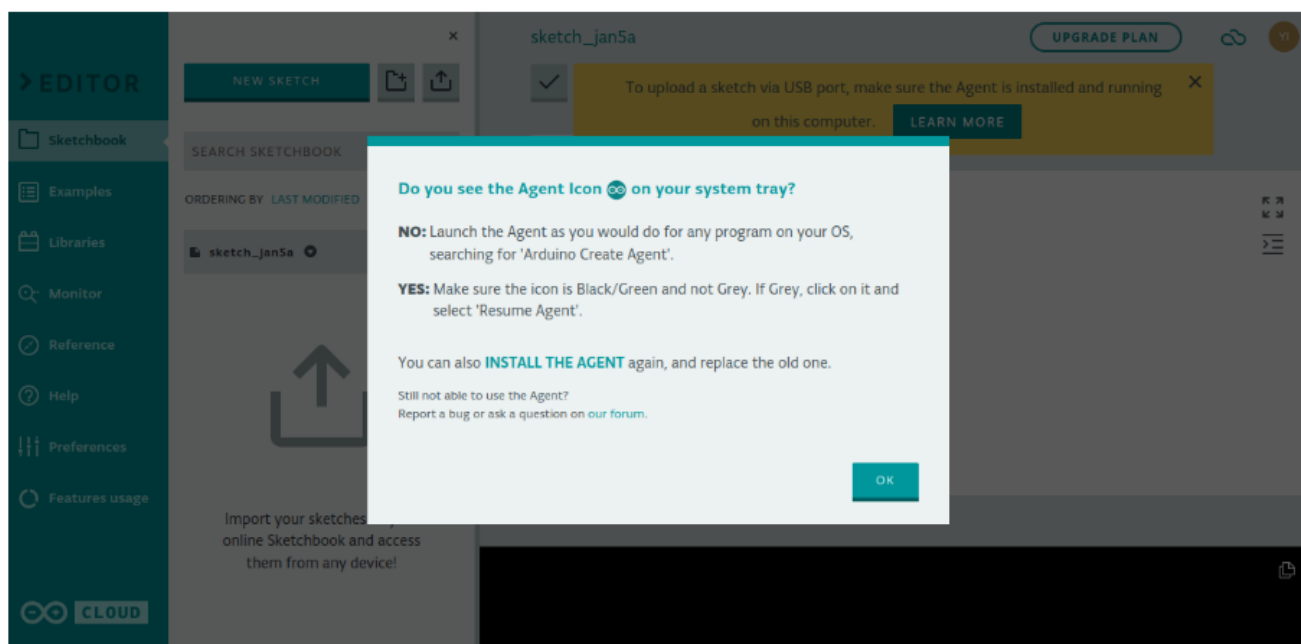


Рисунок 4 — Предложение установить дополнительное ПО для старта работы с Arduino

В репозитории среды программирования можно найти информацию об использовании приложения Arduino-cli [25], [26] для взаимодействия с платами. Arduino-cli — это инструмент командной строки для управления платами Arduino без использования среды разработки Arduino IDE. Он позволяет компилировать, загружать и управлять скетчами Arduino с помощью командной строки, что делает его удобным инструментом для автоматизации процесса разработки и интеграции Arduino в различные среды разработки и рабочие процессы. Arduino-cli поддерживает широкий спектр плат Arduino и различных плат расширения, а также предоставляет возможность установки дополнительных библиотек и платформ.

Приложение обладает набором удобных команд при помощи которых можно организовать сообщение и передачу данных. Среди доступных функций требовалось внедрить в собственный клиент следующие:

- компиляция скетча;
- поиск плат;
- прошивка выбранной платы;
- поиск плат в реальном времени.

Приложение предоставляет два способа для интеграции. Первый предполагает использование приложения по прямому назначению (в качестве приложения командной строки). Общение между потребителем и приложением осуществляется при помощи вызова команд с определенным набором аргументов. `Arduino-cli` может работать в фоновом режиме — имеется возможность запуска приложения как системную службу или демона в зависимости от операционной системы.

Второй способ обмена информации между приложением и потребителями является открытие `gprs`-канала [27], [28]. Часть сообщений передается через дуплексный канал связи. Примером такого взаимодействия является опрос подключенных к устройству пользователя плат `Arduino`. Остальные сообщения передаются по симплексному каналу связи.

В результате анализа было решено разработать плагин для интеграции приложения `Arduino-cli` как приложения командной строки и предоставления описанных выше возможностей пользователям. С точки зрения имеющейся архитектуры была добавлена библиотека, содержащая контракты для встраивания приложений командной строки, а также был разработан плагин, отвечающий за поиск, соединение и работу с `Arduino-cli`.

## 2.4 Система программирования “КуМир роботы”

По окончании анализа исходных кодов среды и языка программирования КуМир, была составлена схема доработки исходного кода проекта для реализации отдельного клиента среды программирования для изучения робототехники. Схема дорабатываемых библиотек в разрезе архитектуры проекта приведена на рисунке 5. Схема добавленных плагинов в разрезе архитектуры проекта приведена на рисунке 6.



Рисунок 5 — Разработанная библиотека(выделена рамкой)

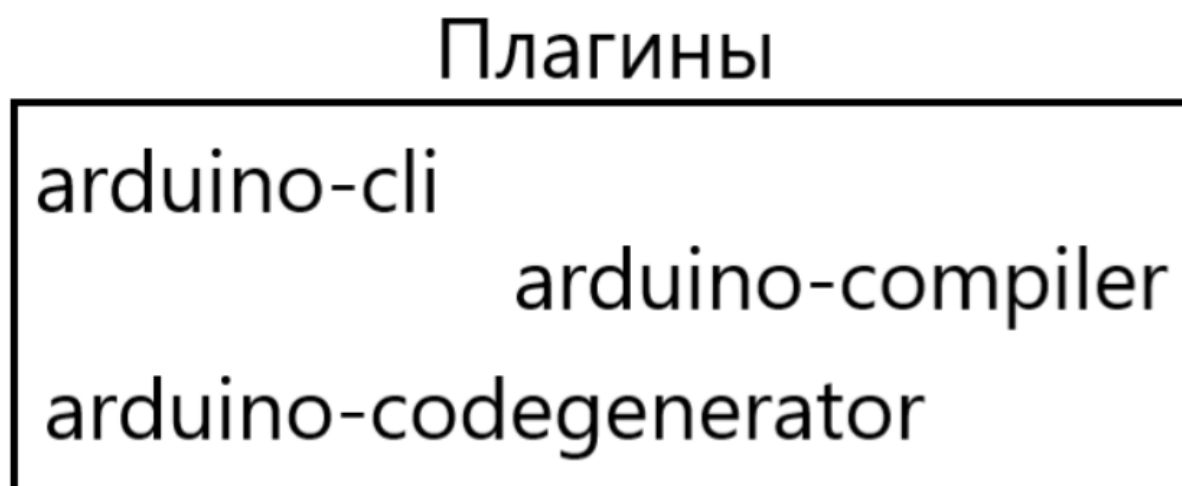


Рисунок 6 — Список разработанных плагинов для работы с платами Arduino

При создании файла конфигурации нового приложения было решено не включать в него стандартный набор исполнителей - Робот, Кузнечик и пр. При



работе в данном клиенте единственным доступным исполнителем является исполнитель “Ардуино”, предоставляющий следующие функции:

- функции аналогового и дискретного ввода/вывода сигналов платы;
- функции для соединения ПК с платами;
- набор констант для работы с портами платы.

Плагины `arduino-compiler` и `arduino-codegenerator` работают по аналогии с уже существующими плагинами для КуМира. Использование этих плагинов при разработке нового приложения не составляет сложностей. Плагин `Arduino-cli` накладывает ряд дополнительных ограничений на пользователя, связанных с установкой дополнительного ПО, а именно самой программы `Arduino-cli`.

В связи с добавлением внешней зависимости для приложения плагин для работы с платами `Arduino` содержит виджет проверки состояния доступности программы `Arduino-cli`. Внешний вид виджета представлен на рисунке 7.

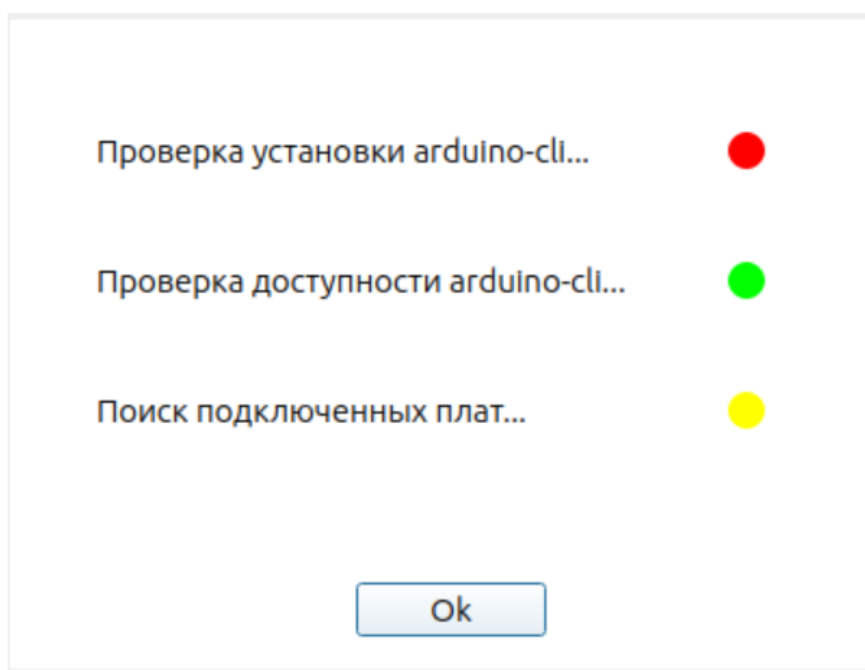


Рисунок 7 — Внешний вид виджета для возможности разрабатывать платы  
Arduino

Виджет открывается при запуске приложения. Проверка возможности работы включает в себя 3 этапа:

- проверка установки `arduino-cli` — поиск установленного на устройстве пользователя приложения;
- проверка доступности `arduino-cli` — проверка наличия прав и корректности установки приложения;
- поиск подключенных плат — опрос портов устройства пользователя на предмет наличия подключенных плат.

Первые 2 пункта проверки являются обязательными и должны быть успешно пройдены для старта работы. Результат работы виджета сохраняется в настройках приложения. Проверка возможности работы может быть произведена не только при запуске клиента “КуМир Роботы“. Кнопка с соответствующим названием находится в меню “Настройки“.

В случае успешного прохождения проверок функционал по работе с платами доступен. Иначе часть возможностей становится недоступной. При попытке воспользоваться недоступной частью функционала, система уведомляет пользователя о невозможности действия. Виджет работает по принципу конечного автомата. Ряд состояний, определяющих работу виджета приведен ниже:

1. проверка установки;
2. проверка доступности;
3. `arduino-cli` не доступна;
4. поиск плат;
5. платы не найдены;
6. одна или несколько плат найдены;
7. проверки успешно пройдены.

В случае завершения работы виджета в состояниях 1-3, пользователю предлагается перейти по ссылке на официальный сайт приложения и установить его. Для проверки корректности установки `Arduino-cli` виджет пытается обратиться по 1 из стандартных путей установки приложения в системе. В случае установ-

ки приложения по пути, отличающемуся от стандартного, пользователь может добавить путь в файловой системе к установленному приложению.

Для отслеживания списка подключенных плат используется команда *boardlist* с флагом *-w*. Список подключенных плат доступен во вкладке “Платы“. При наведении пользователь видит выпадающий список с набором подключенных к устройству плат. Для различия плат с одинаковыми именами используется параметр *FQBN*. Представляет собой строку, которая содержит информацию о модели платы, архитектуре микроконтроллера и версии платформы Arduino. *FQBN* используется в Arduino IDE и других инструментах для определения параметров и компиляции программного кода для конкретной платы Arduino. Примером *FQBN* может быть *"arduino : avr : uno"*, где *"arduino"* указывает на производителя, *"avr"* — на архитектуру микроконтроллера и *"uno"* — на модель платы.

Разработанная система программирования значительно упрощает процесс изучения робототехники. Предыдущая итерация проекта представляла собой консольное приложение-транслятор для генерации скетча для платы Arduino по программному коду на языке программирования КуМир. Отдельный клиент предоставляет возможность интерактивного изучения робототехники используя один интерфейс как для отслеживания состояния плат, так и для работы с платами напрямую — их прошивки, компиляции и настройки скетча.

## ЗАКЛЮЧЕНИЕ

В ходе работы были изучены материалы по системе программирования КуМир, аппаратно-программной платформе Arduino и фреймворку QT. Были решены следующие задачи:

- проанализированы исходные коды системы программирования КуМир;
- проанализирована история развития технологий, используемых для разработки системы программирования;
- были произведены доработки и устранение ошибок в исходном коде системы программирования КуМир;
- проанализировано семейство плат Arduino и их аналоги;
- разработан отдельный клиент для разработки роботов;
- обновлены используемые библиотеки и фреймворки для разработки среды программирования;
- разработан плагин для взаимодействия с платами Arduino.

Разработка приложения “КуМир роботы“ прошла начальную стадию развития — произошло логическое и физическое разделение исходного кода от основной массы, был разработан ряд отдельных плагинов, формулирующих на-  
полнение приложения. В дальнейшем, планируется ряд доработок для повышения удобства пользования приложением. Список возможных доработок приведен ниже:

- профилирование приложения с целью поиска и устранения утечек памяти;
- создание виджета, демонстрирующего устройство подключенной платы;
- создание плагина-шаблонизатора плат для выбора .h-файла, содержащего набор предзаданных портов и констант;

- динамическое изменение статического анализатора при помощи внедренного шаблона платы;
- создание библиотеки шаблонов, позволяющей пользователям делиться шаблонами роботов;
- интеграция с API github для повышения доступности библиотек шаблонов — добавление возможности подключения в клиент репозитория с шаблонами;
- рефакторинг компилятора языка программирования КуМир — отказ от использования кода фреймворка QT.

По окончании реализации и тестирования среды программирования “КуМир роботы“ для разработки роботов, планируется спроектировать и разработать одного или нескольких роботов на базе аппаратного комплекса Arduino для опробования разработки в школах.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1 Официальный сайт КуМир [Электронный ресурс]. — 2022. — URL: <https://www.niisi.ru/kumir/>. Загл. с экр. Яз. рус.
- 2 Леонов, А. Г. Методика преподавания основ алгоритмизации на базе системы «КуМир» [Электронный ресурс]. — 2009. — URL: [https://inf.1sept.ru/view\\_article.php?ID=200901701](https://inf.1sept.ru/view_article.php?ID=200901701). Загл. с экр. Яз. рус.
- 3 Кушниренко, А. Г. докл. ПиктоМир: пропедевтика алгоритмического языка (опыт обучения программированию старших дошкольников) // Большой московский семинар по методике раннего обуч. информатике. — М.: ИТО-РОИ, 2012.
- 4 Манаев Р.Г. ТЕХНОЛОГИЯ ВНЕДРЕНИЯ НЕПРЕРЫВНОЙ ИНТЕГРАЦИИ В КРУПНЫХ ВЫСОКОНАГРУЖЕННЫХ СИСТЕМАХ С МИНИМИЗАЦИЕЙ ОШИБОК И ВРЕМЕННЫХ ПОТЕРЬ СО СТОРОНЫ РАЗРАБОТЧИКОВ // Инновации и инвестиции. 2020. №12.
- 5 Маклафлин Б., Поллайс Г., Уэст Д. Объектно-ориентированный анализ и проектирование. - СПб.: Питер, 2013. - 608с.: ил.
- 6 Фаулер, Мартин. Шаблоны корпоративных приложений. : Пер. с англ. - М. : ООО “И.Д. Вильямс”, 2016. - 544с.: ил. - Парал. тит. англ.
- 7 Эванс, Эрик. Предметно-ориентированное проектирование (DDD): структуризация сложных программных систем.: Пер. с англ. - СПб.: ООО “Диалектика”, 2020 - 448с.: ил. - Парал. тит. англ.
- 8 Ахо, Альфред В., лам, Моника С., Сети, Рави, Ульман, Джеффри Д. Компиляторы: принципы, технологии и инструментарий. 2-е изд.: Пер. с англ. - М.: ООО “И.Д. Вильямс”, 2018 - 1184с.: ил. - Парал. тит. англ.

- 9 Пронин А.А., Синельников Е.А. Модули в языке программирования КуМир 2.0 // Информационные технологии в образовании: сборник / редакция: С. Г. Григорьев [и др.]. – Саратов: Саратовский университет [издание], 2022. – Вып. 5: материалы XIV Всероссийской научно-практической конференции «Информационные технологии в образовании» (ИТО-Саратов-2022), Саратов, 28-29 октября 2022 г. – 290 с. : ил. (9,19Мб). – URL: <https://sgu.ru/node/197426>. – Режим доступа: Свободный. Продолжающиеся издания СГУ на сайте [www.sgu.ru](http://www.sgu.ru). [207-211]
- 10 Смирнов Максим докл. Модернизация унаследованных приложений // конференция ArchDays 2023
- 11 Исходный код среды исполнения КуМир [Электронный ресурс]. — URL: <https://github.com/a-a-maly/kumir2> Загл. с экр. Яз. рус
- 12 Кушниренко, А. Г. Опыт интеграции цифровой образовательной среды КуМир в платформу Мирера // Объединенная конференция "СПО: от обучения до разработки": материалы конференции / Под ред. В. Л. Чёрный. — МАКС Пресс, 2022. — С. 24–30.
- 13 Список версий фреймворка QT [Электронный ресурс]. — URL: [https://wiki.qt.io/Portal:Quick\\_Access](https://wiki.qt.io/Portal:Quick_Access)
- 14 Вареница Виталий Викторович, Марков Алексей Сергеевич, Савченко Владислав Вадимович, Цирлов Валентин Леонидович ПРАКТИЧЕСКИЕ АСПЕКТЫ ВЫЯВЛЕНИЯ УЯЗВИМОСТЕЙ ПРИ ПРОВЕДЕНИИ СЕРТИФИКАЦИОННЫХ ИСПЫТАНИЙ ПРОГРАММНЫХ СРЕДСТВ ЗАЩИТЫ ИНФОРМАЦИИ // Вопросы кибербезопасности. 2021. №5 (45).
- 15 Амини Камран. Экстремальный Си. Параллелизм, ООП и продвинутые возможности. — СПб.: Питер, 2021. — 752 с.: ил. — (Серия «Для профессионалов»).

- 16 Бобков В.А., Черкашин А.С. Обработка и визуализация пространственных данных на гибридном вычислительном кластере // Прикладная информатика. 2014. №4 (52).
- 17 Шабалин, К. В. Формирование креативных способностей школьников при выполнении проектов на базе платформы Arduino / К. В. Шабалин // Педагогическое образование в России. — 2022. — No 2. — С. 135–140.
- 18 Глазов Сергей Юрьевич, Сергеев Алексей Николаевич, Усольцев Вадим Леонидович ВОЗМОЖНОСТИ ПРИМЕНЕНИЯ ПЛАТФОРМЫ ARDUINO В УЧЕБНОМ ПРОЦЕССЕ ПЕДАГОГИЧЕСКОГО ВУЗА И ОБЩЕОБРАЗОВАТЕЛЬНЫХ ШКОЛ // Известия ВГПУ. 2021. №10 (163).
- 19 Серёгин, М. С. Использование платформы Arduino в образовательной деятельности / М. С. Серёгин // Инновационная наука. — 2019. — No 6. — С. 62–64.
- 20 Серёгин, М. С. Использование платформы Arduino в образовательной деятельности / М. С. Серёгин // Инновационная наука. — 2019. — No 6. — С. 62–64.
- 21 Исходный код приложения Arduino IDE первой версии [Электронный ресурс]. — URL: <https://github.com/arduino/Arduino>
- 22 Исходный код приложения Arduino IDE современной версии [Электронный ресурс]. — URL: <https://github.com/arduino/arduino-ide>
- 23 Борис Черный. Профессиональный TypeScript. Разработка масштабируемых JavaScript-приложений. - СПб.: Питер, 2021. - 352 с.: ил. - (Серия “Бестселлеры O’Reily”)
- 24 Мардан Азат. React быстро. Веб-приложения на React, JSX, Redux и GraphQL. — СПб.: Питер, 2019. — 560 с.: ил. — (Серия «Библиотека про-



граммиста»).

- 25 Исходный код приложения Arduino CLI [Электронный ресурс]. — URL: <https://github.com/arduino/arduino-cli>
- 26 Документация к приложению Arduino CLI [Электронный ресурс]. — URL: <https://arduino.github.io/arduino-cli/0.35/>
- 27 Официальный сайт протокола grpc [Электронный ресурс]. — URL: <https://grpc.io/>
- 28 Индрасири Касун, Курупу Данеш. gRPC: запуск и эксплуатация облачных приложений. Go и Java для Docker и Kubernetes. - СПб.: Питер, 2021. - 224с.: ил. - (Серия “Бестселлеры O’Reilly”)

## ПРИЛОЖЕНИЕ А

### Исходный код `dockerfile`, унифицирующий настройки среды запуска системы тестирования

```
1 FROM ubuntu
2
3 ARG path_to_tests_folder="./Tests"
4
5 LABEL EMAIL=gorka19800@gmail.com
6
7 #install all the necessary libs and apps
8 RUN apt-get update
9 RUN apt-get dist-upgrade -y
10 RUN echo "8"| apt-get install -y qttools5-dev-tools
11 RUN apt-get install -y git python3 cmake qtbase5-dev g++ libqt5svg5-dev
12   libqt5x11extras5-dev qtscript5-dev libboost-system-dev zlib1g zlib1g-dev
13 #setup git
14 RUN git config --global user.email "gorka19800@gmail.com"
15 RUN git config --global user.name "Test suit"
16 #clone repo and prepare for building kumir-to-arduino translator
17 RUN mkdir /home/Sources
18 WORKDIR /home/Sources/
19 RUN git clone https://github.com/CaMoCBaJL/kumir2
20 WORKDIR kumir2/
21 RUN git pull
22 RUN git checkout translator_tests
23 RUN git merge -s ours --no-edit origin/ArduinoFixes
24 RUN mkdir build
25 WORKDIR build/
26 #build translator
27 RUN cmake -DUSE_QT=5 -DCMAKE_BUILD_TYPE=Release ..
28 RUN make -j 18
29 #start tests
```

```
30 WORKDIR ../kumir_tests/
31 RUN touch test_results.log
32 RUN python3 test_script.py -d -o ./test_results.log
33 -tr ../build/bin/kumir2-arduino -t \${path_to_tests_folder}
```

## ПРИЛОЖЕНИЕ Б

### Исходный код скрипта, производящего тестирование транслятора

```
1  import os
2  import sys
3  import subprocess
4
5  #constants
6  class CONSOLE_BG_COLORS:
7      HEADER = '\033[95m'
8      OKBLUE = '\033[94m'
9      OKCYAN = '\033[96m'
10     OKGREEN = '\033[92m'
11     WARNING = '\033[93m'
12     FAIL = '\033[91m'
13     ENDC = '\033[0m'
14     BOLD = '\033[1m'
15     UNDERLINE = '\033[4m'
16
17  class TEST_RESULT_STATE:
18     COMPLETED = 1
19     MISSING_EXPECTATION = 2
20     COMPILER_ERROR_HAPPEND = 3
21     FAILED = -1
22     NONE = 0
23
24  COMPILER_ERROR_LABEL = "ERROR!"
25
26  EXPECTATION_FILE_EXTENTION = ".exp"
27  SOURCE_FILE_EXTENTION = ".kum"
28
29  CONTROL_CHARACTERS = ["\n", "\r", "\t", " "]
30  BYTES_TO_READ_COUNT = 1024
```

```

31 CHAR_THRESHOLD = 0.3
32 TEXT_CHARACTERS = ''.join(
33     [chr(code) for code in range(32, 127)] +
34     list('\b\f\n\r\t')
35 )
36 BINARY_CHAR_EXAMPLE = '\x00'
37
38 TESTS_FOLDER_NAME = "Tests"
39
40 ARGS = {"help": ["-h", "--help"],
41         "translator": ["-tr", "--translator"],
42         "output": ["-o", "--output"],
43         "duplicate": ["-d", "--duplicate"],
44         "skip-successfull": ["-ss", "--skip-successfull"],
45         "skip-failed": ["-sf", "--skip-failed"],
46         "skip-without-expectation": ["-swe", "--skip-without-expectation"],
47         "skip-with-compiler-error": ["-sce", "--skip-with-compiler-error"],
48         "brief": ["-b", "--brief"],
49         "path-to-tests": ["-t", "path-to-tests"]
50     }
51
52 ERRORS = {
53     "wrong input": "Wrong args input - you should type path to file after -c or
54     -o flags!",
55     "file not exist": "File doesn't exist!",
56     "wrong file extention": "File extention for translator is not correct! It should
57     be a bin-file.",
58     "no path to tests": "Wrong args input - you should type path to tests folder
59     after -t flag!"
60 }
61
62 #data structure to store test results
63 class TestResult:

```

```

64
65     __text_color = CONSOLE_BG_COLORS.OKGREEN
66     __header_text = ""
67     SKIPPED_TEST_TYPES = [TEST_RESULT_STATE.NONE]
68
69     def __init__(self, name: str, source: str, expectation: str, result: str):
70         self.name = name
71         self.source_file_name = source
72         self.expectation_file_name = expectation
73         self.resultFileName = result
74         self.state = TEST_RESULT_STATE.NONE
75
76     def __str__(self):
77         self.__setup_output()
78         additional_test_data = f'''
79         Test group expectations: {self.expectation_file_name}
80         Test group results: {self.resultFileName}.
81         To get more detail info about comparison results, print:
82         vimdiff {self.source_file_name} {self.expectation_file_name}
83             {self.resultFileName}
84         or vimdiff {self.expectation_file_name} {self.resultFileName}
85         {CONSOLE_BG_COLORS.ENDC}
86         '''
87         return f'''{self.__text_color}
88         Test {self.name}.
89         {self.__header_text}
90         Sources for the test: {self.source_file_name}
91         {additional_test_data if self.state is TEST_RESULT_STATE.COMPLETED
92          or self.state is TEST_RESULT_STATE.FAILED else ""}
93         '''
94
95     def __setup_output(self):
96         if self.state == TEST_RESULT_STATE.COMPLETED:

```

```

97         self.__text_color = CONSOLE_BG_COLORS.OKGREEN
98         self.__header_text = "Congratulations! Test completed successfully!"
99     elif self.state == TEST_RESULT_STATE.MISSING_EXPECTATION:
100         self.__text_color = CONSOLE_BG_COLORS.OKCYAN
101         self.__header_text = "Oh! Test didn't complete: no expectation"
102     elif self.state == TEST_RESULT_STATE.COMPILER_ERROR_HAPPEND:
103         self.__text_color = CONSOLE_BG_COLORS.WARNING
104         self.__header_text = "Oh! Test didn't complete: compiler error"
105     elif self.state == TEST_RESULT_STATE.FAILED:
106         self.__text_color = CONSOLE_BG_COLORS.FAIL
107         self.__header_text = "Sorry, but test failed..."
108
109     class TestSection:
110
111         def __init__(self, section_name) -> None:
112             self.name = section_name
113             self.test_results = []
114
115         def __str__(self) -> str:
116             columns, _ = os.get_terminal_size()
117             if (len(self.test_results) > 0):
118                 return f"""
119                 {CONSOLE_BG_COLORS.WARNING + "-" *columns + CONSOLE_BG_COLORS.ENDC}
120                 Test section {self.name} starts here:
121                 {os.linesep.join(list(map(lambda test_result: str(test_result),
122                 self.test_results))))}
123                 End of {self.name} section tests
124                 {CONSOLE_BG_COLORS.WARNING + "-" *columns + CONSOLE_BG_COLORS.ENDC}
125                 """
126
127             return ''
128
129     #functions

```

```

130 def remove_control_characters(data_array):
131     result = []
132     for i in data_array:
133         for cc in CONTROL_CHARACTERS:
134             i = i.replace(cc, "")
135
136         if i:
137             result.append(i)
138
139     return result
140
141 def get_file_data(filename):
142     if not os.path.exists(filename):
143         return ''
144     file = open(filename, "r")
145     result = file.readlines()
146     file.close()
147     return result
148
149 def has_errors(text):
150     return COMPILER_ERROR_LABEL in text
151
152 def process_sources(source_filename, path_to_translator):
153     path, _ = os.path.splitext(source_filename)
154     result_filename = path + ".kumir.c"
155
156     #call kumir2-arduino with params:
157     --out="path_to_cwd/results/test_name.kumir.c" -s ./test_name.kum
158     popen = subprocess.Popen([path_to_translator,
159                               f'--out={result_filename}',
160                               '-s',
161                               source_filename],
162                               stdout=subprocess.PIPE)

```



```

163     popen.wait()
164     popen.stdout.read()
165
166     return result_filename
167
168 def compare_data(expected_data, processed_data) -> TEST_RESULT_STATE:
169     result_without_kumir_ref = remove_control_characters(processed_data[2:])
170     expected_data = remove_control_characters(expected_data)
171
172     if (has_errors(result_without_kumir_ref)):
173         return TEST_RESULT_STATE.COMPILER_ERROR_HAPPEND
174
175     for i in range(len(result_without_kumir_ref)):
176         if result_without_kumir_ref[i] != expected_data[i]:
177             return TEST_RESULT_STATE.FAILED
178
179     return TEST_RESULT_STATE.COMPLETED
180
181 def get_test_result_type_counters(test_sections):
182     counters = {
183         TEST_RESULT_STATE.COMPLETED: 0,
184         TEST_RESULT_STATE.FAILED: 0,
185         TEST_RESULT_STATE.COMPILER_ERROR_HAPPEND: 0,
186         TEST_RESULT_STATE.MISSING_EXPECTATION: 0,
187     }
188
189     for test_section in test_sections:
190         for test_result in test_section.test_results:
191             counters[test_result.state] += 1
192
193
194     return counters
195

```

```

196 #log comparison results to file
197 def log_test_results(logs_filename, data_to_log: TestSection, log_to_console):
198     result_type_counters = get_test_result_type_counters(data_to_log)
199     completed_tests_count, failed_tests_count,
200     compiler_error_happend_tests_count,
201     missing_expectations_tests_count = \
202     [result_type_counters[k] for k in result_type_counters]
203
204     log_data = []
205     for test_section in data_to_log:
206         test_section.test_results = list(filter
207         (lambda test_result: test_result.state not in
208         TestResult.SKIPPED_TEST_TYPES, test_section.test_results))
209         log_data.append(test_section)
210
211     log_data_strings = list(map(lambda x: str(x), log_data))
212     log_data_strings.insert(0, f'''
213     There were found: {len(log_data)} tests.
214     {CONSOLE_BG_COLORS.OKGREEN} Completed:
215     {completed_tests_count} {CONSOLE_BG_COLORS.ENDC}
216     {CONSOLE_BG_COLORS.FAIL} Failed:
217     {failed_tests_count} {CONSOLE_BG_COLORS.ENDC}
218     {CONSOLE_BG_COLORS.WARNING} With compiler error happend:
219     {compiler_error_happend_tests_count} {CONSOLE_BG_COLORS.ENDC}
220     {CONSOLE_BG_COLORS.OKCYAN} Missed expectation file:
221     {missing_expectations_tests_count} {CONSOLE_BG_COLORS.ENDC}''')
222
223     if not os.path.exists(logs_filename):
224         open(logs_filename, "a").close()
225
226     log_file = open(logs_filename, "a")
227     log_file.writelines(log_data_strings)
228     log_file.close()

```

```

229
230     if log_to_console:
231         print(f"{os.linesep}".join(log_data_strings))
232
233 def is_binary_file(filename):
234     file_stream = open(filename, 'rb')
235     file_content = file_stream.read(BYTES_TO_READ_COUNT)
236     file_stream.close()
237
238     if not len(file_content):
239         #file is empty, nothing to read
240         return False
241
242     if ord(BINARY_CHAR_EXAMPLE) in file_content:
243         #file contains binary symbols
244         return True
245
246     binary_chars = file_content.translate(TEXT_CHARACTERS)
247     return float(len(binary_chars)) / len(file_content) > CHAR_THRESHOLD
248
249 def validate_file_name(filename: str, is_log_file: bool):
250     if not os.path.exists(filename):
251         print(filename + ERRORS.get("file not exist"))
252         sys.exit(2)
253     if not os.path.isfile(filename):
254         print(ERRORS.get("wrong input"))
255         sys.exit(2)
256     if not is_log_file and not is_binary_file(filename):
257         print(ERRORS.get("wrong file extention"))
258         sys.exit(2)
259
260 def show_help():
261     print(""" kumir2-arduino tester.

```

Description:

Approach of this app is to debug the work of kumir2 to arduino translator. It uses compiled translator's instance, pre-built locally on PC.

To start the work you should input path to compiler and path to logs file.

Flags:

`[-h] [--help]` - show help.

`[-tr] [--translator] ["the path to pre-built kumir2 to arduino translator instance"]` - show app what translator instance to use.

`[-t] [--path-to-tests] ["the path to tests folder"]` = show app the folder with test files.

`[-o] [--output] ["the path to log file"]` - show app where to store test logs.

`[-d] [--duplicate]` - duplicate output to console.

`[-ss] [--skip-successfull]` - skip open log info about successfully completed tests.

`[-sf] [--skip-failed]` - skip open log info about failed tests.

`[-swe] [--skip-without-expectation]` - skip open log info about tests for which the file with expectations was not found.

`[-sce] [--skip-with-compiler-error]` - skip open log info about tests ended with compiler error.

```

295     [-b] [--brief] - skip open log info about all tests.
296     """)
297
298 def process_args():
299     result = ["", "", False, ""]
300     if ARGS["help"][0] in sys.argv or ARGS["help"][1]
301     in sys.argv:
302         show_help()
303         sys.exit(2)
304
305     if ARGS["skip-successfull"][0] in sys.argv or ARGS["
306     skip-successfull"][1] in sys.argv:
307         TResult.SKIPPED_TEST_TYPES.append(
308             TEST_RESULT_STATE.COMPLETED)
309
310     if ARGS["skip-without-expectation"][0] in sys.argv
311     or ARGS["skip-without-expectation"][1] in sys.argv:
312         TResult.SKIPPED_TEST_TYPES.append(TEST_RESULT_STATE
313             .MISSING_EXPECTATION)
314
315     if ARGS["skip-with-compiler-error"][0] in sys.argv or
316     ARGS["skip-with-compiler-error"][1] in sys.argv:
317         TResult.SKIPPED_TEST_TYPES.append(TEST_RESULT
318             _STATE.COMPILER_ERROR_HAPPEND)
319
320     if ARGS["skip-failed"][0] in sys.argv or ARGS[
321     "skip-failed"][1] in sys.argv:
322         TResult.SKIPPED_TEST_TYPES.append(
323             TEST_RESULT_STATE.FAILED)
324
325     if ARGS["brief"][0] in sys.argv or ARGS["brief"][1] in sys.argv:
326         TResult.SKIPPED_TEST_TYPES = [
327             TEST_RESULT_STATE.COMPILER_ERROR_HAPPEND,

```

```

328         TEST_RESULT_STATE.COMPLETED,
329         TEST_RESULT_STATE.FAILED,
330         TEST_RESULT_STATE.MISSING_EXPECTATION
331     ]
332
333     args = sys.argv[1:]
334     for i in range(1, len(args)):
335         if args[i - 1] in ARGS["translator"] or args[i - 1]
336             in ARGS["output"] or args[i - 1] in ARGS
337             ["path-to-tests"]:
338             if (os.path.isfile(args[i])):
339                 validate_file_name(args[i], False if args[i - 1]
340                     in ARGS["translator"] else True)
341
342             if args[i - 1] in ARGS["translator"]:
343                 result[0] = os.path.abspath(args[i])
344             elif args[i - 1] in ARGS["output"]:
345                 result[1] = os.path.abspath(args[i])
346             elif args[i - 1] in ARGS["path-to-tests"]:
347                 result[3] = os.path.abspath(args[i])
348             elif args[i-1] in ARGS["duplicate"]:
349                 result[2] = True
350
351     return result
352
353 def get_files_with_absolute_paths(folder_name):
354     path_to_folder = os.path.join(os.getcwd(), folder_name)
355     files =
356     list(map(lambda x: os.path.join(path_to_folder, x),
357         os.listdir(path=path_to_folder)))
358     files.sort()
359
360     return files

```

```

361
362 def get_source_and_expectation(dir_files):
363     sources = []
364     expectations = []
365     for file in dir_files:
366         if (os.path.isfile(file)):
367             ext = os.path.splitext(file)[1]
368             if (ext == EXPECTATION_FILE_EXTENTION):
369                 expectations.append(file)
370             elif (ext == SOURCE_FILE_EXTENTION):
371                 sources.append(file)
372
373     return [sources, expectations]
374
375 def get_folder_contents_full_paths(path_to_folder):
376     return list(
377         map(
378             lambda x: os.path.join(os.sep, path_to_folder, x),
379             os.listdir(path_to_folder)
380         )
381     )
382
383 def calculate_test_sections(path_to_tests_folder):
384     result = []
385     test_folder_paths = get_folder_contents_full
386     _paths(path_to_tests_folder)
387     for test_folder_path in test_folder_paths:
388         if os.path.isfile(test_folder_path):
389             continue
390
391         result.append(TestSection(test_folder_path.
392             split(os.sep)[-1]))
393     test_paths = get_folder_contents_full_paths

```

```

394         (test_folder_path)
395     for test_dir_path in test_paths:
396         source_files, expectation_files =
397         get_source_and_expectation(get_folder_contents_full_paths
398         (test_dir_path))
399         if os.path.isfile(test_dir_path) or len
400         (expectation_files) > 1 or len(source_files)
401         < 1:
402             continue
403
404         result[-1].test_results.append(TestMethod(
405             test_dir_path.split(os.sep)[-1],
406             source_files[0],
407             "", ""
408         ))
409     )
410
411     if len(expectation_files) == 1:
412         result[-1].test_results[-1]
413         .expectation_file_name = expectation_files[0]
414         result[-1].test_results[-1].resultFileName
415         = process_sources(source_files[0],
416             args_data[0])
417         result_data = get_file_data(result[-1]
418             .test_results[-1].resultFileName)
419         expected_data =
420         get_file_data(result[-1].test_results[-1].expectation_file_name)
421         result[-1].test_results[-1].state
422         = compare_data(expected_data, result_data)
423     else:
424         result[-1].test_results[-1].state =
425         TEST_RESULT_STATE.MISSING_EXPECTATION
426

```



```
427
428     return result
429
430 if __name__=="__main__":
431     args_data = process_args()
432
433     if not args_data[3]:
434         print("Didn't find any test to execute. Shutting down.")
435         sys.exit()
436
437     test_results = calculate_test_sections(args_data[3])
438
439     log_test_results(args_data[1], test_results, args_data[2])
```