Труды МАИ. 2022. № 123
Trudy MAI, 2022, no. 123

Научная статья
УДК 004.053
DOI:10.34759/trd-2022-123-21

ПРОБЛЕМЫ СОПРОВОЖДЕНИЯ АППАРАТНО-ПРОГРАММНЫХ КОМПЛЕКСОВ

Владимир Николаевич Лукин[1], Юрий Борисович Чечиков[2], ✉
Виталий Евгеньевич Секретарев[3], Алла Леонидовна Дзюбенко[4],
Наталья Фаридовна Алтухова[5]
[1,2,3]Московский авиационный институт (национальный исследовательский университет), Москва, Россия
[1]Московский государственный психолого-педагогический университет,
Москва, Россия
[4,5]Финансовый университет при правительстве РФ,
Москва, Россия
[1]lukinvn@list.ru
[2]yourych@mail.ru✉
[3]sinoptik.mnpk@gmail.com
[4]al_dz@list.ru
[5]nfaltuhova@fa.ru

Аннотация. В данной статье рассматриваются проблемы, связанные со сложностью обслуживания аппаратных и программных систем в условиях нехватки времени, расширения комплекса и обновления элементной базы. Целью данной работы

является снижение сложности разработки и сопровождения программного обеспечения комплекса, повышение надежности программирования. Задачей исследования является обоснование подхода к построению программно-аппаратных систем в парадигме объектно-ориентированного программирования, а также его применение к построению систем различного уровня сложности. В данной статье анализируются внешние и внутренние проблемы, влияющие на продолжительность жизненного цикла аппаратных и программных систем для различных предметных областей. Подходом к решению проблем является технология, основанная на объектно-ориентированной парадигме, включающей понятия интерфейсов, абстрактных базовых классов, вертикального и веерного наследования, виртуальных функций, полиморфизма. Технологии объектно-ориентированного программирования обеспечивают одновременное функционирование существующих и вновь появившихся устройств без переписывания системы. Предложен вариант адаптивной ОС RT для подключения бортового оборудования, который значительно сокращает время разработки серийных и отдельных изделий и повышает их надежность. Приведен пример возможного применения предложенной методологии в области автоматизации лабораторных исследований в медицине и показано как при большом разнообразии парка автоанализаторов общие архитектурные решения позволяют разработать базовую модель лаборатории с возможностью детализации для каждого прибора. Новизной является использование объектно-ориентированного подхода при разработке операционной системы реального времени (RTOS), что сокращает время и трудоемкость разработки серийных и отдельных продуктов.

Ключевые слова: аппаратно-программный комплекс, операционная система, объектно-ориентированное программирование, адаптивность, аппаратный интерфейс передачи данных, полиморфизм

Для цитирования: Лукин В.Н., Чечиков Ю.Б., Секретарев В.Е., Дзюбенко А.Л., Алтухова Н.Ф. Проблемы сопровождения аппаратно-программных комплексов // Труды МАИ. 2022. №123. DOI:10.34759/trd-2022-123-21

PROBLEMS OF MAINTENANCE OF SOFTWARE AND HARDWARE SYSTEMS

Vladimir N. Lukin[1], Yury B. Chechikov[2]✉, Vitaly E. Sekretarev[3], Alla L. Dzyubenko[4], Natalya F. Altukhova[5]

[1,2,3]Moscow Aviation Institute (National Research University),
Moscow, Russia

[1]Moscow State Psychological and Pedagogical University,
Moscow, Russia

[4.5]Financial University under the Government of the Russian Federation,
Moscow, Russia

[1]lukinvn@list.ru

[2]yourych@mail.ru✉

[3]sinoptik.mnpk@gmail.com

[4]al_dz@list.ru

[5]nfaltuhova@fa.ru

Abstract. This article discusses the problems associated with the complexity of maintenance of hardware and software systems in conditions of lack of time, expansion of the complex and updating of the element base. The purpose of this work is to reduce the complexity of the development and maintenance of the software complex, increase the reliability of programming. The objective of the research is to substantiate the approach to the construction of software and hardware systems in the paradigm of object-oriented programming, as well as its application to the construction of systems of various levels of complexity. This article analyzes external and internal problems affecting the life cycle of hardware and software systems for various subject areas. An approach to solving problems is a technology based on an object-oriented paradigm that includes the concepts of interfaces, abstract base classes, vertical and fan inheritance, virtual functions, polymorphism. Object-oriented programming technologies ensure the simultaneous functioning of existing and newly appeared devices without rewriting the system. A variant of the RT adaptive OS for connecting on-board equipment is proposed, which significantly reduces the development time of serial and individual products and increases their reliability. An example of the possible application of the proposed methodology in the field of automation of laboratory research in medicine is given and it is shown how, with a wide variety of autoanalyzer fleet, common architectural solutions make it possible to develop a basic laboratory model with the possibility of detailing for each device. The novelty is the use of an object-oriented approach in the development of a real-time operating system (RTOS), which reduces the time and complexity of developing serial and individual products.

***Keywords*:** hardware-software complex, operating system, object-oriented programming, adaptability, hardware data interface, polymorphism

For citation: Lukin V.N., Chechikov Yu.B., Sekretarev V.E., Dzyubenko A.L., Altukhova N.F. Problems of maintenance of software and hardware systems. *Trudy MAI*, 2022, no. 123. DOI: 10.34759/trd-2022-123-21

Широкое распространение так называемых «умных» вещей – естественный результат снижения стоимости, уменьшения габаритов и повышения качества микроэлектронных устройств. С другой стороны, разработка на их базе аппаратно-программных комплексов (АПК) требует значительных затрат средств и времени. В этих условиях естественное следствие – это продление срока использования АПК. Однако появляются новые устройства, новые области их использования, и то, что было актуальным, быстро устаревает.

При этом программное обеспечение остаётся содержательно таким же, и его, казалось бы, можно перенести в новую аппаратную среду. Но проблема в том, что на нижнем уровне взаимодействие программ и устройств меняется, и для корректного управления требуется модифицировать программы. Этот процесс относится к сопровождению, трудоёмкость которого, а, значит, и длительность жизненного цикла АПК, во многом зависит от адаптивности программного обеспечения. В свою очередь, адаптивность (способность к модификации) зависит от качества программного кода и его сложности. Исследования показывают, что затраты на сопровождение, включающее развитие программного комплекса и исправление обнаруженных ошибок, могут составлять до половины его стоимости [1, 2].

Таким образом, учитывая постоянную модернизацию АПК, необходимо найти технологии для быстрой и надёжной разработки и поддержки систем на их базе.

Самая трудоемкая и ответственная часть при внесении изменений в аппаратно-программные комплексы – это программное обеспечение. Даже мощная аппаратная часть не в состоянии реализовать свой потенциал при низком качестве программного обеспечения [3, 4].

Как известно [5, 6, 7, 10, 11, 13-15], в основе успеха лежит качество проектирования и квалификация разработчиков. Однако при использовании инструментальных сред разработчика есть проблемы, которые существенно влияют на качество разработки.

Анализ проблемы

Возникающие проблемы могут быть как внешними, так и внутренними.

Под внешними будем понимать объективные трудности, связанные с подключением нового устройства. Для его подключения программист должен разобраться в логике нового устройства, написать внутренние программы для микроконтроллера (микропроцессорного устройства, предназначенного для обмена информации с системой), согласовать обновленную логику его работы с правилами работы в системе.

Чтобы запрограммировать микроконтроллер, используют специализированные интегрированные среды разработки (Integrated Development Environment – IDE), поставляемые фирмами-разработчиками, такими как, CubeMX или Keil µVision. Если он той же фирмы, что и предыдущий, скорее всего, инструментальные средства разработки будут теми же, что и раньше. В противном случае для использования

новой среды разработки потребуется осваивать её и быть готовым к неприятным неожиданностям.

Разобраться в новой документации и среде разработки не так просто, и для этого квалификация прикладного инженера должна быть не ниже, чем у разработчика среды, для ее достижения необходимо потратить несколько лет. Даже простая, казалось бы, настройка среды требует специальных знаний, чтобы все заработало в штатном режиме. Подобная настройка выполняется один раз, и использовать предыдущий опыт практически невозможно, так как после обновления среды меняются настройки и пользовательский интерфейс и настраивать все приходится заново.

Обычно советуют воспользоваться автоматической генерацией программного кода (IDE) [12]. Но полученный при этом текст тяжёл для восприятия: структура программы непонятна, тем более что связь её частей, назначение функций и переменных неочевидно.

Таким образом, быстро модифицировать текст программы невозможно: изменения хотя бы в одной функции могут привести к непредсказуемой реакции программы. Ещё хуже, если изменения сделаны с ошибкой: может появиться множество наведённых ошибок, на первый взгляд, не связанных с этой.

К сожалению, ошибки в управляющих комплексах могут стоить очень дорого. Так, из-за программной ошибки рухнуло два Боинга-737 Max: один 29 октября 2018 г., погибло 189 человек, другой 10 марта 2019, погибло 157 человек. И программисты реально начинают бояться вносить изменения в программу и отказывают клиентам в

ее модификации. Так что, если однажды задача была решена успешно, программный код уже не модифицируется, и система в этом состоянии доживает свой срок.

Внутренние проблемы – это следствие удобства сопровождения АПК. Оно, во многом, зависит от качества проектирования системы [1, 3, 4]: выбора архитектуры, подбора инструментальных средств, разработки внутренних стандартов работы в коллективе. Если проектирование велось наспех или его выполняли разработчики невысокой квалификации, архитектура системы станет, скорее всего, необоснованно сложной, связи между элементами будут запутанны и неочевидны, и внести изменения в требуемые сроки станет практически невозможно. Поэтому следует заранее планировать возможные направления развития системы, хотя иногда сделать это весьма непросто.

Что касается программирования, то для удобства сопровождения следует обеспечить максимальное удобство поиска и исправления ошибок, в противном случае естественная потребность развивать и масштабировать уже работающую систему будет связана с неприемлемыми рисками. Именно поэтому квалификация сопровождающего программиста должна быть выше квалификации разработчика [1, 2, 8]: изменения приходится вносить в работающий комплекс за короткие сроки.

Таким образом, при разработке новой системы надо стремиться не к тому, чтобы сдать ее как можно быстрее, а к тому, чтобы обеспечить ее удобное сопровождение на протяжении всего жизненного цикла.

Решение

Одним из подходов к решению данной проблемы может стать технология, которая обычно не используется для разработки АПК, а именно объектно-

ориентированный подход (ООП) с его понятиями интерфейсов, абстрактных базовых классов, вертикального и веерного наследования (один предок и несколько прямых потомков), виртуальных функций, полиморфизма.

Аппаратные интерфейсы передачи данных, такие как UART, SPI, USB; MIL-STD1553B, имеют свою логику работы, но все они могут быть приведены к одному абстрактному базовому классу, в котором объявлены только переменные и основные функции приема и передачи. Конкретная их реализация при веерном наследовании для различных классов одного уровня будет своя.

За счет наследования (обобщения-специализации) гарантируется общий подход к алгоритмам работы, стандартизация имен переменных и функций всех родственных объектов иерархии. Тем самым повышается надежность программы и понятность её текста при значительном снижении стоимости сопровождения.

Добавление нового устройства в систему упрощается за счёт виртуальных функций (полиморфизма). Для этого в иерархию классов к суперклассу устройства добавляется подкласс, в котором реализованы методы, отражающие только особенности нового устройства, а общее для всех устройств поведение наследуется от суперкласса. Особенности поведения каждого устройства за счёт полиморфизма обеспечивают различные реализации при едином интерфейсе. В результате получается открытая, удобная к масштабированию, система.

Итак, технологии ООП обеспечивают одновременное функционирование существующих и вновь появившихся устройств без переписывания системы.

Рассмотрим проекты с малой детализацией, например, системы сбора аналоговой информации на различных микроконтроллерах: 1986VE4Y, 1967VK01T,

TMS320F28335. Задачи, в принципе, одинаковы: надо принять данные с АЦП, оцифровать их, отправить внешней задаче и синхронизировать к ним доступ. Однако для каждого микроконтроллера программный код свой, и программисту приходится держать в голове его версии. Это при том, что алгоритм один и тот же, различия лишь в настройках регистров АЦП для каждого микроконтроллера [16, 17].

Естественное решение – создание базовой иерархии, на которой реализованы общие задачи (прием/оцифровка/передача). Конкретизация работы с АЦП для каждого микроконтроллера производится на самом нижнем уровне. Она связана, в частности, с тем, что в разных средах разработчика свои правила распределения памяти по определенным адресам, одинаковые по смыслу регистры могут называться по-разному и иметь свой порядок следования. Обработчики прерываний тоже различные, хотя суть их одна.

Более высокий уровень иерархии для нашего примера – это комплексная система управления (КСУ) с блоком датчиков, вычислительной частью и исполнительными механизмами. В качестве входной информации КСУ получает первичную информацию, но не аналоговую, а уже оцифрованную. Затем полученные данные обрабатываются в соответствии с правилами управления и передаются исполнительным устройствам. Мы видим, что та же схема приема, обработки, передачи информации на более высоком уровне абстракции реализована по-другому.

И, наконец, следующий уровень – это большие системы, например, полностью собранный борт самолета. На финальном этапе собираются приборы и комплексы от разных предприятий, подготовленные разными специалистами с различными подходами. У каждого разработчика по отдельности все работает, но на комплексе

что-то идёт не так. Разобраться, что не работает и у кого, крайне сложно, и этот процесс требует слишком много времени. Основные проблемы, как показывает практика, возникают при приеме и передаче информации между устройствами. Оперативно привести в рабочее состояние программное обеспечение сложно, и гарантировать надёжность комплекса невозможно.

Проблема заключается в том, что для задач подобного класса традиционно используется парадигма процедурно-ориентированного программирования. Она хороша тем, что позволяет достаточно хорошо прогнозировать время отклика. Это играет большую роль в системах реального времени. В её рамках для каждого микроконтроллера создаются свои библиотеки функций, пусть сильно совпадающие, но имеющие в глубине вызовов обращения к конкретному функционалу. И здесь начинаются трудности: выделить в отдельную библиотеку общую часть не представляется возможным. С ростом количества микроконтроллеров значительно растет объем программного кода, который надо помнить и хорошо в нем разбираться. Для программиста трудоемкость задачи резко возрастает, а разработка существенно дорожает.

Рациональным, а, возможно, и оптимальным выходом могла бы стать группировка устройств с концептуально общим поведением, но различной его реализацией на низком уровне, что соответствует объектно-ориентированной парадигме (ООП), изначально предназначенной для задач моделирования.

Она, естественным образом, приводит к использованию общих технологических решений, оставляя технический уровень для специалистов более низкой квалификации. Тем самым, при заданной надёжности изделия, уменьшается

время и стоимость его разработки, что, в свою очередь, даёт возможность ускоренной модификации жизненно важных устройств. В нашем случае, когда критическим параметром становится время адаптации при высокой надёжности, его применение выглядит вполне уместным.

Таким образом, в качестве технологического подхода для разработки надёжных высокоуровневых иерархических систем как нельзя лучше подходит методика ООП. Она изначально ориентирована на возможность различной реализации концептуально схожих процессов или на изменения уже разработанных в ходе сопровождения (модификации). Действительно, на уровне базового класса все единообразно: во всех системах данные нужно собрать, обработать и передать. Однако реализация низших уровней иерархии будет иметь свою конкретику: вычисления, интерфейсы ввода/вывода данных, исполнительные механизмы. Эти действия относительно легко и надёжно реализуются на уровне подклассов.

Для практической реализации этого подхода разработана Адаптивная ОС РВ, которая имеет базовый уровень в виде автономной адаптивной системы ввода/вывода. Она работает с объектами верхнего уровня иерархии, реализующими прием/запись/передачу данных, а для конкретного устройства функционал вызывается с нижнего уровня за счет механизма виртуальных функций и динамического полиморфизма. Таким образом учитывается единство структуры алгоритмов и данных.

Экспериментальные разработки показали, что предложенное решение подходит под все задачи, связанные с управлением аппаратно-программными комплексами.

Рассмотрим подход к решению наших задач в парадигме объектно-ориентированного программирования. Для работы с SPI (Serial Peripheral Interface) нужно создать базовый класс для SPI. На данном уровне не имеет значения какой вычислитель будет использоваться. Этот базовый класс копируется в те места, где требуется работа с SPI: различные микроконтроллеры, ядра микропроцессоров, но без конкретики. Конкретная настройка режимов работы (Master или Slave) реализуется в классах нижнего уровня. Все SPI имеют в базовом классе виртуальные функции инициализации (Init()), чтения FIFO (ReadFIFO()) и записи в FIFO (WriteFIFO()), которые реализуются в потомках. Вызов конкретной функции производится динамически из базового класса в процессе работы. Таким образом, при единой структуре появляется видовое многообразие частных аппаратно-программных решений.

Приведём ещё пример. При автоматизации работы медицинской биохимической лаборатории нужно учитывать парк автоанализаторов, методы сбора и обработки поступающей информации. В Клинической больнице, где авторы разрабатывали программное обеспечение для большой биохимической лаборатории (производительность 1,3 млн. исследований в год, лабораторий такого масштаба в мире было 3-4), анализаторы закупались у разных фирм, они имели абсолютно разный тип взаимодействия с ЭВМ: Technicon аналоговый, LKB, Beckman – цифровой, OLLI – комплекс со встроенной внутренней обработкой данных и т.п. Их подключение требовало значительных затрат времени как на изучение документации, так и на проведение работ. При этом принцип подключения один и тот же, разница только в деталях. Если рассмотреть проведение одного и того же исследования, мы увидим,

что в разных лабораториях оно различно из-за наличия необходимых химических реактивов и измерительных приборов. В этом случае объектно-ориентированный подход был бы весьма кстати, но он тогда был новым, применять его в медицине было рискованно: от результата анализа может зависть жизнь пациента.

На основе ООП можно было бы разработать базовую модель с возможностью детализации для каждого прибора. Конечно, программы нижнего уровня надо разрабатывать, но архитектурные решения были бы общие. Поэтому, в частности, в мире и было мало таких лабораторий: не хватало высококвалифицированных программистов.

Объединение нескольких биохимических лабораторий требует введения внутренних стандартов на их разработку: нужна единая структура, единый формат обмена информацией между производственными участками лаборатории и самими лабораториями. В рамках процедурного подхода снизить трудоемкость комплексной разработки весьма непросто: без единого фундамента разработки при добавлении новой лаборатории программный код значительно разрастается.

Если рассмотреть ситуацию с позиций ООП, то можно выделить единый сценарий работы: получить список заказов для проведения того или иного исследования, организовать очередь в автоанализаторе и получить результаты измерений, сформировать итоговый бланк-ответ, который отсылается лечащему врачу. Имея общий для всех лабораторий базовый класс, можно значительно удешевить разработку и повысить технологичность сопровождения.

Теперь пример полиморфизма родственных систем. Авторами совместно с Институтом механики МГУ в своё время была разработана небольшая мобильная

лаборатория на тех же принципах, но с иными критериями оптимизации. Требовалась лёгкая дешёвая система, но надёжная и работающая корректно. Толчком было землетрясение в Спитаке, туда следовало направить помощь. Пришлось переписывать много кода, система получилась, но было потеряно слишком много времени. Тем не менее, она была рекомендована для массового внедрения в больницах и поликлиниках Москвы. И во время внедрения стало ясно, что для таких задач ООП – находка. Но из-за финансовых проблем в медицине массовое внедрение не состоялось.

Результаты

1. В рамках предложенного подхода разработана адаптивная ОС РВ с автономной системой ввода/вывода, которая позволяет создавать оптические и цифровые бортовые вычислительные системы, системы организации беспроводной передачи данных (по радиоканалам, Wi-Fi, GPS, Bluetooth, лазерной передачи данных), обладающие свойствами быстрой адаптации к конкретным условиям, удобством эксплуатации на протяжении жизненного цикла [9, 18, 19, 20].

2. Снижение трудоёмкости сопровождения АПК позволяет ускорить их модернизацию, что даёт возможность своевременно вводить в действие новые изделия. При этом обеспечивается более высокий уровень надежности программного обеспечения, снижается стоимость внесения изменений, облегчается многократное использование имеющихся наработок. Кроме того, благодаря преемственности разработок повышается надежность всего семейства изделий.

3. На основе единой структуры становятся возможными решения, которые учитывают специфику области функционирования будущих систем и обеспечивают

видовое многообразие аппаратно-программных решений соединения с оборудованием или частями системы. Использование Адаптивной ОС РВ снижает сроки и трудоемкость разработки серийных и индивидуальных изделий. Область применения Адаптивной ОС РВ инвариантна к предметным областям: промышленность, оборона, медицина, образование, коммуникационные сервисы.

4. Данная разработка допускает работу как с импортными, так и с отечественными микросхемами, что отвечает тенденции повсеместного импортозамещения в оборонной отрасли и не только в ней. В случае перехода на отечественную элементную базу к существующей иерархии классов, отражающих специфику устройств, добавится фрагмент, реализующий только особенности нового элемента, что гораздо проще, чем создавать и отлаживать весь программный комплекс заново.

Заключение

Новизна разработки заключается в применении объектно-ориентированного подхода в разработке операционной системы реального времени (ОС РВ), что рассматривалось как решение, связанное с риском потери производительности при работе в режиме жёсткого реального времени. Однако расчёты и эксперименты показали применимость её к аппаратному программному обеспечению в авиационных системах. Существенный выигрыш такого решения – снижение стартовых затрат на разработку новых систем, а главное – возможность значительного продления их жизненного цикла при модификации бортового оборудования. Как показывает опыт, при переходе к объектно-ориентированной

парадигме объем программного кода гарантированно сокращается как минимум вдвое, при значительном повышении качества программного текста.

Как следствие, впервые предложена методика разработки прототипа системы, настроенного на работу не с отдельным микроконтроллером, а с семейством микроконтроллеров и центральных процессоров, основанных на конкретном ядре (Cotrex M4F, Cotrex M3, Stm32H7xx b и т. д.).

Для создания систем различных уровней сложности реализован подход к построению фундамента, на основе которого можно разрабатывать прикладные системы со своим функционалом. Разработчик стартует не с нуля, а с уровня, обеспечивающего гарантированный бесперебойный обмен информацией между устройствами различных типов. Поэтому инженер может приступить к своим непосредственным задачам, связанным с созданием системы обмена информацией, минуя этап освоения нового микроконтроллера.

Помимо прочего, при построении системы на основе созданного прототипа снижаются требования к квалификации разработчика и сопровождающего программиста, что в настоящее время, в условиях повсеместного дефицита высококвалифицированных специалистов, служит немаловажным фактором.

Практика разработки систем, работающих в режиме жёсткого реального времени, показала, что арсенал программных средств разработчика и сопровождающего программиста полезно расширить за счёт методов объектно-ориентированного программирования, которые традиционно в этой области не применяются.

Список источников

1. Гласс Р., Нуазо Р. Сопровождение программного обеспечения / Пер. с англ. – М.: Мир, 1983. – 156 с.

2. Гласс Р. Факты и заблуждения профессионального программирования / Пер. с англ. – СПб.: Символ-плюс, 2007. – 240 с.

3. Лукин В.Н. Дзюбенко А.Л., Чечиков Ю.Б. Подходы к разработке пользовательского интерфейса // Программирование. 2020. № 5. С. 16–24. DOI: 10.31857/S0132347420050052

4. Lukin V.N., Dzyubenko A.L., Chechikov. Yu.B. Approaches to User Interface Development // Programming and Computer Software. 2020, vol. 46, no. 5, pp. 316–324. DOI: 10.1134/S0361768820050059

5. Вигерс К., Битти Д. Разработка требований к программному обеспечению / Пер. с англ. — М.: Изд-во «Русская редакция»; СПб.: БХВ-Петербург, 2020. – 736 с.

6. Информационная технология. Сопровождение программных средств. ГОСТ Р ИСО/МЭК 14764–2002, 2003.07.01

7. Процессы жизненного цикла программных средств. ГОСТ Р ИСО/МЭК 12207–2010, 2012.03.01

8. Емелин И.В., Лукин В. Н., Эльчиян Р.А. О жизненном цикле системы реального времени // Прикладная информатика. 1986. №1(10). С. 135–143.

9. Чечиков Ю.Б., Секретарев В.Е., Лукин В.Н., Дзюбенко А.Л. и др. Адаптивная система обмена данными // Научно-техническая информация. Серия 2: информационные процессы и системы. 2022. № 2. С 25–28.

10. Леффингуэлл Д., Уидриг Д. Принципы работы с требованиями к программному обеспечению. Унифицированный подход. / Пер. с англ. - М.: Издательский дом «Вильямс», 2002. - 448 с.

11. GOST 28195-89. Software quality assessment. General provisions. Moscow, House of Standards, 1989, 38 p.

12. Code Composer Studio (CCS) Integrated Development Environment (IDE) — CCSTUDIO — TI Tool Folder. URL: http://www.ti.com/tool/ccstudio

13. Борзов Д.Б., Чернышев А.А., Сизов А.С., Соколова Ю.В. Методика и алгоритм построения вычислительной сети на основе беспроводного протокола // Труды МАИ. 2021. № 121. URL: https://trudymai.ru/published.php?ID=162667. DOI: 10.34759/trd-2021-121-20

14. Оценка качества программных средств. Общие положения. ГОСТ 28195-89. - М.: Изд-во стандартов, 1989. - 38 с.

15. Оценка программ. Протокол доступа к сетевому ресурсу: http://www.structur.h1.ru/ocenka.htm

16. Гуревич О.С., Кессельман О.Г., Трофимов А.С., Чернышов В.И. Современные беспроводные технологии на авиационном борту // Труды МАИ. 2017. № 94. URL: https://trudymai.ru/published.php?ID=81143

17. IAR Embedded Workbench Kickstart — IAR-KICKSTART — TI Software Folder. URL: http://www.ti.com/tool/iar-kickstart

18. Цилькер Б.Я., Орлов С.А. Организация ЭВМ и систем. – СПб.: 2004. - 668 с.

19. Бородин В.В., Петраков А.М., Шевцов В.А. Анализ эффективности передачи данных в сети связи группировки беспилотных летательных аппаратов // Труды МАИ. 2015. № 81. URL: http://trudymai.ru/published.php?ID=57894

20. Кучерявый А.А. Бортовые информационные системы. - Ульяновск: УлГТУ, 2004. - 504 с.

References

1. Glass R., Nuazo R. *Soprovozhdenie programmnogo obespecheniya* (Software maintenance guidebook), Moscow, Mir, 1983, 156 p.

2. Glass R. *Fakty i zabluzhdeniya professional'nogo programmirovaniya* (Facts and misconceptions of professional programming), Saint Petersburg, Simvol-plyus, 2007, 240 p.

3. Lukin V.N. Dzyubenko A.L., Chechikov Yu.B. *Programmirovanie*, 2020, no. 5, pp. 16–24. DOI: 10.31857/S0132347420050052

4. Lukin V.N., Dzyubenko A.L., Chechikov. Yu.B. Approaches to User Interface Development, *Programming and Computer Software*, 2020, vol. 46, no. 5, pp. 316–324. DOI: 10.1134/S0361768820050059

5. Vigers K., Bitti D. *Razrabotka trebovanii k programmnomu obespecheniyu* (Development of software requirements), Moscow, Izd-vo «Russkaya redaktsiya»; Saint Prtersburg, BKhV-Peterburg, 2020, 736 p.

6. *Informatsionnaya tekhnologiya. Soprovozhdenie programmnykh sredstv. GOST R ISO/MEK 14764–2002* (Information technology. Software maintenance. GOST R ISO/IEC 14764-2002), 2003.07.01

7. *Protsessy zhiznennogo tsikla programmnykh sredstv. GOST R ISO/MEK 12207–2010* (Software lifecycle processes. GOST R ISO/IEC 12207-2010), 2012.03.01

8. Emelin I.V., Lukin V.N., El'chiyan R.A. *Prikladnaya informatika*, 1986, no. 1(10), pp. 135–143.

9. Chechikov Yu.B., Sekretarev V.E., Lukin V.N., Dzyubenko A.L. et al. *Nauchno-tekhnicheskaya informatsiya. Seriya 2: informatsionnye protsessy i sistemy*, 2022, no. 2, pp 25–28.

10. Leffinguell D., Uidrig D. *Printsipy raboty s trebovaniyami k programmnomu obespecheniyu. Unifitsirovannyi podkhod* (Principles of working with software requirements. A unified approach), Moscow, Izdatel'skii dom «Vil'yams», 2002, 448 p.

11. *GOST 28195-89. Software quality assessment. General provisions.* Moscow, House of Standards, 1989, 38 p.

12. *Code Composer Studio (CCS) Integrated Development Environment (IDE) — CCSTUDIO — TI Tool Folder.* URL: http://www.ti.com/tool/ccstudio

13. Borzov D.B., Chernyshev A.A., Sizov A.S., Sokolova Yu.V. *Trudy MAI,* 2021, no. 121. URL: https://trudymai.ru/eng/published.php?ID=162667. DOI: 10.34759/trd-2021-121-20

14. *Otsenka kachestva programmnykh sredstv. Obshchie polozheniya. GOST 28195-89* (Software quality assessment. General provisions. GOST 28195-89), Moscow, Izd-vo standartov, 1989, 38 p.

15. *Otsenka programm. Protokol dostupa k setevomu resursu:* http://www.structur.h1.ru/ocenka.htm

16. Gurevich O.S., Kessel'man O.G., Trofimov A.S., Chernyshov V.I. *Trudy MAI*, 2017, no. 94. URL: https://trudymai.ru/eng/published.php?ID=81143

17. *IAR Embedded Workbench Kickstart — IAR-KICKSTART — TI Software Folder*. URL: http://www.ti.com/tool/iar-kickstart

18. Tsil'ker B.Ya., Orlov S.A. *Organizatsiya EVM i system* (Organization of computers and systems), Saint Petersburg, 2004, 668 p.

19. Borodin V.V., Petrakov A.M., Shevtsov V.A. *Trudy MAI*, 2015, no. 81. URL: http://trudymai.ru/eng/published.php?ID=57894

20. Kucheryavyi A.A. *Bortovye informatsionnye sistemy* (On-board information systems), Ul'yanovsk, UlGTU, 2004, 504 p.

Статья поступила в редакцию 25.03.2022; одобрена после рецензирования 06.04.2022; принята к публикации 20.04.2022.
The article was submitted on 25.03.2022; approved after reviewing on 06.04.2022; accepted for publication on 20.04.2022.