УДК 551.46.077:629.584

РАЗРАБОТКА И ТЕСТИРОВАНИЕ СИСТЕМ УПРАВЛЕНИЯ АНПА НА БАЗЕ ПРОГРАММНОЙ ПЛАТФОРМЫ RCE

А.И. Боровик

Федеральное государственное бюджетное учреждение науки
Институт проблем морских технологий ДВО РАН[1]

Использование программной платформы RCE при создании системы управления (СУ) автономного необитаемого подводного аппарата (АНПА) позволяет значительно упростить и ускорить процесс написания исходного кода компонентов [1, 2]. Для эффективной организации работы группы специалистов разработан алгоритм создания и тестирования СУ, основанный на спиральной модели процесса разработки программного обеспечения и использовании средств эмуляции и тестирования, встроенных в платформу RCE. В рамках алгоритма выбраны технические средства автоматизации процесса сборки, создания программной документации, непрерывного анализа качества кода и контроля версий. Разработанный алгоритм позволяет добиться распараллеливания работы программистов и тестировщиков, снижения вероятности возникновения ошибок и как следствие – увеличения скорости написания кода и упрощения присоединения новых специалистов к процессу.

ВВЕДЕНИЕ

Система управления (СУ) представляет собой совокупность программных средств, предназначенных для управления работой оборудования робота с целью выполнения поставленных перед ним задач. Для облегчения процесса проектирования, создания и эксплуатации систем управления часто используется специальное промежуточное программное обеспечение, называемое программной платформой. В ИПМТ была разработана и в настоящее время используется и дорабатывается универсальная компонентно-ориентированная событийная программная платформа RCE (Robot Components Engine) [1, 2].

Зачастую организации процесса разработки уделяют недостаточно внимания, а между тем, как показывают современные исследования в области инженерии программного обеспечения, она имеет огромную значимость при создании сложных компьютерных программ, к которым относятся в том числе и системы управления роботами [3–5]. Грамотная организация процесса позволяет значительно сократить материальные и временные затраты на разработку программного обеспечения, добиться эффективной работы команды программистов, повысить качество конечного продукта и его пригодность для использования в последующих проектах.

В данной статье предлагается алгоритм разработки и тестирования систем управления АНПА на базе программной платформы RCE, который позволяет добиться распараллеливания работы программистов и тестировщиков, снижения вероятности возникновения ошибок и как следствие – увеличения скорости написания кода и упрощения присоединения новых специалистов к процессу.

Состав команды и роли участников разработки системы управления

Сложные программные комплексы создаются коллективным трудом группы специалистов. Перед началом разработки такой системы необходимо четко определиться с ролью каждого участника, его обязанностями и уровнем ответственности. Алгоритм разработки, предлагаемый программной платформой RCE, определяет 6 ролей для участников процесса разработки и тестирования системы управления АНПА:

- архитектор системы управления;
- разработчик драйвера устройства;
- разработчик программы-алгоритма;
- разработчик программы пользовательского интерфейса;
- тестировщик;
- оператор (оператор-тестировщик).

Архитектор системы управления отвечает за функционирование конечного программного комплекса и является руководителем проекта. Он обеспечивает взаимодействие главного конструктора

[1] 690091, Владивосток, ул. Суханова, 5а. Тел.: 8(423) 2215545, доб. 445. E-mail: alexey@borovik.me

робота и представителей заказчика с отделом разработки ПО, определяет вид конечной системы и общий принцип ее функционирования. Архитектор утверждает компонентный состав системы управления и используемые интерфейсы, определяет общий дизайн системы (распределение компонентов СУ по процессам RCE и компьютерам робота). Архитектор должен полностью представлять себе процесс функционирования программной части робота на уровне интерфейсов взаимодействия и программных компонентов, а также все возможности и технические особенности программной платформы RCE. Основными требованиями к архитектору СУ являются наличие высшего образования и умение руководить группой специалистов. Умение программировать на языке C/C++ для него необязательно (но желательно). Желателен опыт работы в другой роли, например в роли разработчика драйвера или алгоритма поведения.

Разработчик драйвера устройства должен понимать принципы функционирования конкретного устройства, способы получения от него нужной информации и представления ее в виде, регламентированном интерфейсом платформы. В случае использования промышленного устройства, способы взаимодействия с которым, как правило, документированы, разработчиком драйвера может быть программист без специального образования по профилю работы устройства. В случае использования устройства собственного производства наилучшим является вариант, когда разработкой драйвера занимается тот же человек (или команда), который занимался разработкой самого устройства. Для этого специалиста основным является требование к знанию языка программирования C++ на уровне, достаточном для взаимодействия с платформой RCE.

Разработчик программы-алгоритма должен представлять суть процесса, описываемого конкретным алгоритмом. К примеру, суть работы алгоритма движения подводного аппарата заключается в регулировании упоров движительной системы для достижения указанных скоростей по осям. Разработчик алгоритма должен представлять поведение аппарата в воде и влияние мощности каждого двигателя на результирующий вектор движения. В то же время для описания алгоритма обхода препятствий (который является высокоуровневым по отношению к алгоритму движения) разработчику нужно представлять процесс движения в виде математической модели и решать ее соответствующими методами. Таким образом, рекомендация к образованию вытекает из специфики составляемого алгоритма. Для решения наиболее частых задач робототехники потребуется высшее образование по специальностям «математика» или «физика». Требования к знанию языка C++ для этого специалиста такие же, как и для разработчика драйвера.

*Разработчик программ пользовательского интерфейс*а отвечает за создание программ (или языка команд, в случае командного интерфейса), понятных для конечного пользователя системы. Для создания программ с графическим интерфейсом потребуется знание соответствующих технологий – Qt, VxWidgets и т.п. Знание языка программирования C++ на высоком уровне для такого специалиста обязательно.

Тестировщик (специалист отдела контроля качества) отвечает за контроль качества конечного программного продукта. Тестировщик составляет модульные и интеграционные тесты для проверки корректности работы компонентов и поддерживает в актуальном состоянии базу известных проблем и ошибок в ПО. Тестировщик должен знать язык программирования C++ на достаточном для взаимодействия с платформой уровне, уметь пользоваться специальными техническими средствами для организации процесса тестирования – как сторонними, так и встроенными в RCE.

Оператор робота – конечный пользователь робототехнической системы, управляющий механизмом для получения требуемого результата. Оператор должен уметь привести систему управления в рабочее состояние, управлять работой робота, следить и оценивать получаемые результаты посредством программ пользовательского интерфейса. Специфических требований к его образованию нет. В состав группы разработчиков обязательно должен входить *оператор-тестировщик*, который будет выполнять комплексное тестирование всего аппаратно-программного комплекса АНПА с точки зрения его потребительских свойств.

При условии успешного внедрения платформы RCE в робототехнический процесс на конкретном предприятии требования к составу группы разработчиков проекта значительно упростятся. Вместе с платформой может поставляться библиотека драйверов и алгоритмов, содержащая наиболее часто использующиеся компоненты. В этом случае разработка программного обеспечения робота сведется к компоновке нужной системы из существующих модулей (задаче архитектора системы) и написанию отдельных алгоритмов и драйверов, специфичных для конкретного АНПА.

Алгоритм разработки систем управления на базе RCE

Под моделью процесса разработки программного обеспечения понимается его формализованное представление [3, 5]. В современной литературе описаны следующие модели: каскадная (водопадная) модель [5, 6], V-модель [7], итеративная модель [8] и спиральная модель [9]. В настоящее время спиральная модель применяется для создания больших, сложных и дорогих программных комплексов, в том числе робототехнических [5, 10]. Именно спиральная модель выбрана в качестве базы для алгоритма создания СУ на базе RCE.

Предложенный алгоритм разработки СУ состоит из начального шага (переход на который с более поздних шагов невозможен) и 6 шагов, выполняющихся в цикле. После завершения каждой итерации цикла создается *прототип* или, на более поздних итерациях, *версия* программного обеспечения СУ. Алгоритм может быть наглядно представлен в виде схемы, изображенной на рис. 1. Опишем последовательность действий, закладываемую алгоритмом.

Начальный шаг. На данном шаге архитектор определяет модель архитектуры системы управления и выделяет уровни абстракции команд [2]. Данный шаг определят базовый вид разрабатываемой системы, поэтому возврат к нему с последующих шагов невозможен.

Утверждение компонентов. На данном шаге в ходе первой итерации цикла архитектор распределяет функции системы управления между отдельными компонентами, принимает решение об использовании уже существующих программ или написании новых, а также назначает ответственных за их разработку или доработку программистов. Программисты определяют входные и выходные данные компонентов, а также обрабатываемые ими события. На последующих итерациях, с учетом накопленного в процессе разработки опыта, список компонентов, а также параметры их работы могут быть изменены программистами по согласованию с архитектором системы. Все принятые решения фиксируются документально. При необходимости изменения состава коллектива разработчиков все

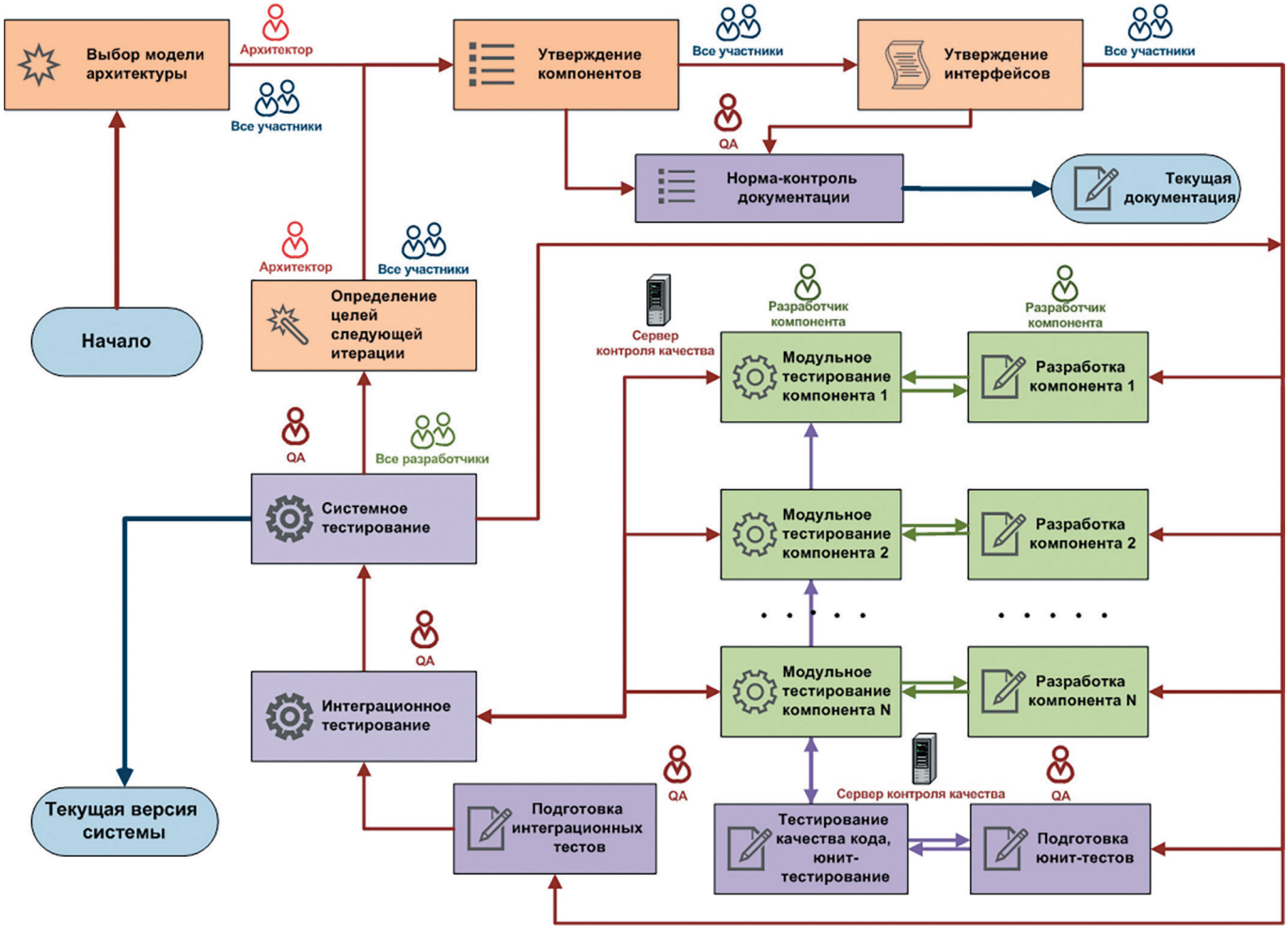

Рис. 1. Алгоритм разработки систем управления на базе RCE

подобные решения выполняются на этом шаге.

Результатом данного шага является документация системы управления, описывающая версию системы, которую планируется получить по итогам текущей итерации цикла разработки. Данная документация оформляется согласно применимым в конкретном проекте правилам (обычно по ГОСТ 19, 34, РД-50 [11]. Получаемая на данном шаге документация не является финальной, она отражает лишь текущее представление разработчиков о системе. В ходе последующих итераций цикла разработки документация может быть изменена (на этом же шаге) с учетом изменений, планируемых в системе. Отметим, что высокоуровневая документация СУ, получаемая на данном шаге, описывает лишь общие свойства компонентов (входные и выходные данные, события, пользовательский интерфейс). Низкоуровневая документация генерируется на основе исходного кода и обновляется каждый день в автоматическом режиме.

Утверждение интерфейсов. На первой итерации цикла архитектор определяет применяемые в системе универсальные интерфейсы (подробно классификация интерфейсов RCE дана в [1]). Он принимает решение об использовании существующих интерфейсов без изменения, их доработке или разработке новых, а также назначает ответственных за данные работы программистов. Для каждого интерфейса ответственные разработчики определяют список входящих в него сообщений, состав каждого сообщения и диапазоны изменения значений отдельных параметров. На каждой итерации цикла все изменения в интерфейсах согласуются между разработчиками комплекса. Согласованные интерфейсы компилируются в виде библиотек, которые могут быть подключены к эмулирующему комплексу RCE. Для каждого интерфейса также определяются основные параметры тестирования, которое будет применяться ко всем компонентам, использующим этот интерфейс.

Согласованные интерфейсы описываются в виде документов, которые включаются в текущую документацию проекта. Составлением этих документов занимается отдел контроля качества. Результатом данного шага являются интерфейсы, описанные в виде заголовочных файлов платформы RCE, интерфейсных библиотек и протоколов информационного сопряжения (обычно описанных согласно ГОСТ 19.xxx).

Параллельная разработка и модульное тестирование. В ходе данного шага группа программистов параллельно и независимо друг от друга разрабатывает компоненты системы управления. В это же время отдел тестирования разрабатывает набор модульных тестов, предназначенных для тестирования отдельных компонентов. Все участники процесса используют в своей работе утвержденные на предыдущем шаге интерфейсы. Важно, что тесты разрабатываются без учета знаний о внутренней специфике работы компонента, исключительно на основе утвержденных ранее правил его взаимодействия с внешними компонентами и утвержденной документации.

Модульное тестирование выполняется параллельно процессу разработки в автоматическом режиме. Ночью сервер контроля качества выполняет компиляцию измененных в течение рабочего дня компонентов и тестирует их с использованием написанных программ-тестов. Помимо проверки работоспособности в ходе модульного тестирования выполняется также тестирование производительности компонентов и тестирование качества кода с использованием специальных технических средств. Результаты тестирования оформляются сервером в виде отчета, доступного для просмотра программистам. Для составления модульных тестов используются возможности эмулирующего комплекса RCE.

Схема рабочего дня программиста на этом шаге процесса изображена на рис. 2.

Основные этапы работы:

- получение отчета от системы автоматического тестирования и контроля качества кода;
- исправление ошибок;
- модульное тестирование изменений (выполнение тестов, которые вызвали ошибку в ходе автоматического тестирования);
- описание и документирование новых функций;

Рис. 2. Схема рабочего дня программиста

- внедрение новых функций (утвержденных на этапе утверждения интерфейсов);
- загрузка кода на сервер.

На данном шаге первой (стартовой) итерации цикла программисты разрабатывают прототипы компонентов, которые могут не выполнять реальных вычислений, но должны строго подчиняться требованиям интерфейсов и документации в части входных и выходных данных. На последующих итерациях осуществляются внедрение описанного функционала и доработка компонентов с учетом новых требований.

Интеграционное тестирование. В ходе данного шага тестировщики проверяют взаимодействие всех компонентов системы при выполнении тестовых миссий. Тестирование осуществляется с использованием эмулирующего комплекса программной платформы. Тестовые миссии составляются заранее специалистами отдела контроля качества и должны покрывать основные сценарии возможного использования разрабатываемого АНПА. Найденные в ходе выполнения тестирования ошибки должны заноситься в систему управления проектами и исправляться программистами.

Переход на данный шаг с шага параллельной разработки означает завершение процесса внедрения новых функций в исходный код компонентов. Разработчики компонентов к этому времени должны полностью закончить реализацию закрепленных в документации возможностей и сосредоточиться на взаимодействии с отделом тестирования для устранения ошибок. Результатом данного шага является прототип или версия системы управления, без ошибок работающая на эмуляторе платформы при выполнении разработанного набора тестов.

Системное тестирование. В ходе данного шага текущая версия системы управления тестируется в стендовых или натурных условиях на реальном прототипе аппаратной платформы (или уже готовом АНПА) по составленной заранее специалистами отдела качества программе испытаний. Системное тестирование выполняется оператором-тестировщиком с использованием актуальной пользовательской документации и программ с графическим интерфейсом. В ходе данного тестирования проверяется работа программ на реальном оборудовании в реальных или приближенных к реальным условиям. Кроме того, проверяются актуальность и понятность документации, удобство пользовательского интерфейса и т.п. К данным работам могут привлекаться представители заказчика для оценки текущего состояния работ, выработки предложений и замечаний. Системное тестирование на поздних итерациях цикла разработки может быть объединено с предварительными или межведомственными испытаниями программно-аппаратного комплекса АНПА.

Результатом проведенного системного тестирования является текущая версия (или, на ранних стадиях разработки, прототип) системы управления с описанием присущих системе неразрешенных проблем и недостатков.

Определение цели следующей итерации и времени ее завершения. На данном шаге архитектор системы, по возможности и необходимости привлекая представителей заказчика, определяет основные цели следующей итерации процесса, планирует шаги и время их завершения. Цели работ должны определяться с учетом проблем, выявленных и не решенных в ходе системного тестирования.

Количество итераций предложенного алгоритма разработки не регламентировано, однако для соответствия работ ГОСТ 34.601-90 список итераций должен включать, как минимум, следующие.

Начальная (стартовая) итерация, результатом которой является нулевой прототип системы управления, не выполняющий реальной работы, но структурно соответствующий желаемому результату.

Эскизное проектирование, результатом которого является эскизный проект (прототип) системы управления.

Техническое проектирование, результатом которого является технический прототип СУ.

Рабочее проектирование, результатом которого является первая рабочая версия системы управления.

Финальная доработка, в ходе которой в проекте СУ устраняются проблемы, выявленные на стадии предварительных испытаний робота.

Финальная доработка не является последней итерацией, поскольку во время эксплуатации робота заказчиком могут быть выявлены новые проблемы или поставлены новые задачи. В этом случае будет определен список требований к новой версии СУ и процесс разработки продолжится.

Рассмотрим основные технические средства, которые используются для организации описанного алгоритма разработки.

Технические средства алгоритма

Контроль версий. В рамках задачи контроля версий решаются следующие вопросы:

- поддержание актуальной версии системы у каждого из участвующих в разработке специалистов;
- просмотр и отмена любых сделанных изменений вплоть до

самого начального состояния программного комплекса;

- поддержание журнала изменений, в котором фиксируются авторы изменений, а также причины, по которым изменения были внесены;
- создание промежуточных версий программного обеспечения (соответствующих итерациям цикла разработки) и ответвлений (при использовании одного и того же кода в разных проектах).

Для решения описанного комплекса задач существует специальной класс программного обеспечения – системы управления версиями (Version Control System, VCS) [12]. Среди наиболее известных можно выделить CVS, SVN, Git. Подробный обзор этих программ выходит за рамки данной статьи. Для организации контроля версий при разработке СУ на RCE используется система SVN.

Создание программной документации. Программная документация системы управления может быть разделена по трем уровням:

1) документация по использованию всего программно-аппаратного комплекса АНПА (эксплуатационные документы);

2) документация по работе отдельных компонентов (спецификации, протоколы информационного сопряжения, и т.п.);

3) документация для разработчиков компонентов системы управления и программ пользовательского интерфейса по функциям и классам.

Документы первого и второго уровней составляются на этапах утверждения компонентов и интерфейсов и должны поддерживаться в актуальном состоянии на протяжении всего времени жизни проекта. Залогом поддержания документации в актуальном состоянии является ее использование для составления тестов, что обеспечивает постоянную проверку соответствия программ их эксплуатационным документам. Данные документы предоставляются заказчику, поэтому их вид жестко регламентирован.

Документация для разработчиков компонентов (3-й уровень) обычно не входит в обязательный перечень документов, поэтому часто не составляется и не поддерживается. Подобный подход приводит к возникновению различных ошибок при интеграции новых компонентов в систему, сложностям в обучении новых участников процесса разработки и как следствие – замедлению работы. Установка конкретных правил составления подобных документов поможет избежать различных ошибок на начальном этапе разработки и значительно упростить доработку программного комплекса в дальнейшем.

Одним из наиболее простых способов по созданию подобной документации является написание самодокументированного кода. Описание принципов работы функций и программных примитивов в таком случае приводится в самом коде, следуя определенным несложным правилам оформления. Как правило, функции и классы в самодокументированном коде предваряются специальными комментариями, описывающими принципы их работы, входные и выходные параметры, родительские отношения с другими классами и т.п. Исходные файлы такого кода анализируются специальной программой – генератором документации, которая составляет на их основе пригодные для чтения файлы справочной системы в автоматическом режиме. Одним из наиболее продвинутых средств по документированию исходного кода в настоящее время является DOxygen [13], поддерживающий большое количество языков программирования, в том числе и язык C++. В проектах на RCE рекомендуется использовать синтаксис именно данного генератора.

Автоматизация сборки ПО СУ заключается в автоматизации выполнения многих связанных с разработкой рутинных действий, таких как: компиляция исходного кода в бинарный код, компоновка объектных модулей в исполняемые модули, выполнение тестов, создание сопроводительной документации (из файлов исходного кода с помощью генератора документации) и прочих [10].

Платформа RCE поддерживает следующие типы автоматизации:

- автоматизация по запросу (on-demand automation) – выполнение всех назначенных действий системой по команде пользователя;
- автоматизация по плану (scheduled automation) – автоматизация, выполняемая в автоматическом режиме по заранее определенному плану; сборка ПО осуществляется на основе файлов с исходным кодом в среде контроля версий один раз в сутки – ночью (чтобы не мешать программистам и подготовить финальный вариант системы и результат тестов к их приходу на работу).

Для выполнения автоматизации по запросу в платформе RCE используются скрипты кроссплатформенной системы автоматизации CMake [14]. От других популярных систем автоматизации (Make, Autotools, Scons) CMake отличается продвинутым скриптовым языком, специально предназначенным для решения задач автоматизации, а также большим количеством поддерживаемых интегрированных средств разработки (IDE) в качестве целевых для генерации файлов проектов. Для автоматизации по плану используются скрипты CMake и скрипты для

ОС Linux системного планировщика задач cron, установленных на сервере контроля качества.

Автоматизация по плану составляет основу практики «непрерывной интеграции», которая подразумевает создание законченных версий программного продукта на постоянной основе. В любой момент времени доступна последняя версия системы управления (наряду с технической документацией), которая может быть при необходимости проверена непосредственно на АНПА.

Проверка качества кода. В последнее время появились системы, осуществляющие непрерывный анализ качества кода. Данные системы в автоматическом режиме осуществляют поиск ошибок и потенциально опасных мест в коде на основе его статического анализа, без компиляции и составления тестов [15]. Кроме того, подобные системы проверяют код на соответствие стандартам программирования, выдерживание общего стиля и т.п. Как показало использование подобных систем на практике [16, 17], они действительно позволяют находить ошибки, которые проходят фильтр модульных тестов по тем или иным причинам, и значительно снизить необходимость последующего рефакторинга кода.

Описание и подробный разбор подобных систем выходят за рамки данной статьи. В рамках предложенного алгоритма разработки используется система PVS-Studio, интегрированная в платформу SonarQube. Запуск процедуры анализа автоматизирован и выполняется в рамках ночной автоматизации по плану.

Управление проектом. Для эффективного использования описанных выше средств необходимо наличие сервера управления проектами – специальной программы, которая позволяет отслеживать найденные в ходе автоматизированного тестирования ошибки, вести учет временных затрат, планировать внедрение новых функций и назначать ответственных специалистов, осуществлять доступ к автоматической документации проекта, предоставлять средства общения для разработчиков и тестировщиков и т.д. В настоящее время существует большое количество самых разнообразных систем управления проектами, как платных, так и бесплатных. В рамках предлагаемого алгоритма разработки СУ используется бесплатная система Redmine.

Средства тестирования RCE. Для выполнения тестов в платформе RCE предусмотрена специальная программа-эмулятор. Рассмотрим ее основные возможности.

На этапе модульного тестирования драйвера эмулятор позволяет симулировать данные от устройства, которое на момент разработки программы может быть еще не готово. Драйвер использует функции платформы для получения данных от устройства посредством серийного порта. Эмулятор позволяет настроить платформу так, чтобы эти функции возвращали данные не с конкретного порта системы, а из заданного бинарного файла. При этом эмулятор позволяет настроить скорость, с которой строчки данных определенной длины, считанные из файла, будут поступать в систему.

Для модульного и интеграционного тестирования всех компонентов СУ используется специальный механизм эмулятора, позволяющий генерировать любые сообщения интерфейса. В качестве источника исходных данных для сообщения может быть выбран один из двух компонентов:

- генератор, который позволяет задавать характер (линейная функция, степенная функция, показательная функция, синусоида, произвольные значения), диапазон и скорости изменения дробных или целочисленных параметров сообщения;
- проигрыватель «логов», который позволяет использовать сообщения из записанных ранее журналов работы системы («логов»).

Планируется создание 3-мерного эмулятора, который также сможет выступать источником симулированных сообщений.

Для облегчения системного тестирования эмулятор позволяет изменять скорость течения времени в компонентах (это возможно благодаря тому, что все компоненты используют средства RCE для получения текущего времени). Благодаря этому есть возможность за несколько минут получить результат многочасовой работы системы.

Преимущества предложенного алгоритма

К основным преимуществам предложенного алгоритма разработки систем управления относятся:

1) постоянный контроль качества конечной системы;

2) наличие промежуточных версий программного обеспечения;

3) возможность параллельной и независимой работы программистов и их равномерная загрузка;

4) наличие зафиксированных классификаций всех сущностей;

5) четко определенные технические средства организации процесса разработки;

6) использование продвинутых средств тестирования и отладки;

7) установленная парадигма написания кода (компонентно-ориентированная система с событийно-ориентированными компонентами);

8) закрепленные требования к умениям всех участников разработки;

9) использование единого языка программирования;

10) поддержание документации системы в актуальном состоянии;

11) возможность организации процесса разработки согласно требованиям ГОСТ 34.

Описанные свойства предложенного алгоритма позволяют эффективно организовать процесс разработки, снизить количество ошибок в конечном продукте, повысить скорость написания кода, а также с минимальными затратами изменять состав коллектива разработчиков системы. Применение данного алгоритма на практике позволит получить более эффективный и надежный программный продукт.

ЛИТЕРАТУРА

1. Боровик А.И., Наумов Л.А. Компонентно-ориентированная программная платформа для автономных мобильных роботов // Известия ЮФУ [Таганрог]. 2013. № 3. С. 39–47.

2. Боровик А.И., Наумов Л.А. Компонентно-ориентированная система управления АНПА ММТ-2012 // Известия ЮФУ [Таганрог]. 2014. № 3. С. 102–112.

3. Липаев В.В. Программная инженерия. Методологические основы: учебник. М.: Теис, 2006. 608 с.

4. The Guide to the Software Engineering Body of Knowledge. SWEBOK, IEEE Computer Society Professional Practices Committee, 2004.

5. Архипенков С. Лекции по управлению программными проектами [Электронный ресурс]. – URL: http://www.arkhipenkov.ru/resources/sw_project_management.pdf (дата обращения: 21.08.2017).

6. Royce D.W.W. Managing the Development of large Software Systems // IEEE Wescon. 1970. August. P. 1–9.

7. Forsberg K., Mooz H., Cotterman H. Visualizing Project Management: Models and Frameworks for Mastering Complex Systems. Hoboken, New Jersey: John Wiley & Sons, Inc., 2005. Vol. 3. 480 p.

8. Larman C., Basili V.R. Iterative and incremental development: A brief history // Computer. 2003. Vol. 36, No 6. P. 47–56.

9. Boehm B.W. A Spiral Model of Software Development and Enhancement // Computer. 1988. Vol. 21, No 5. P. 61–72.

10. Дюваль П.М., Матиас С.М., Гловер Э. Непрерывная интеграция: улучшение качества программного обеспечения и снижение риска: пер. с англ. М.: ООО «И.Д. Вильямс», 2008. 240 с.

11. RuGost – разработка документации по ГОСТ 34, 19, РД-50 [Электронный ресурс]. – URL: http://www.rugost.com (дата обращения: 21.08.2017).

12. Pilato C.M., Collins-Sussman B., Fitzpatrick B.W. Version Control with Subversion [Электронный ресурс]. O'Reilly Media, 2008. Vol. 2. 432 p. – URL: http://svnbook.red-bean.com/en/1.7 svn-book.pdf (дата обращения: 21.08.2017).

13. Heesch D. Doxygen [Электронный ресурс]. – URL: http://www.stack.nl/~dimitri/doxygen/ (дата обращения: 21.08.2017).

14. Martin K., Hoffman B. Mastering CMake: A Cross-Platform Build System. Kitware, Inc., 2008. Vol. 4. 385 p.

15. Карпов А. Статический анализ как часть процесса разработки Unreal Engine [Электронный ресурс]. – URL: https://habrahabr.ru/company/pvs-studio/blog/331724/ (дата обращения: 21.08.2017).

16. Карпов А. 27000 ошибок в операционной системе Tizen [Электронный ресурс]. – URL: https://www.viva64.com/ru/b/0519/ (дата обращения: 21.08.2017).

17. Размыслов С. Серьёзные ошибки в коде CryEngine V [Электронный ресурс]. – URL: https://habrahabr.ru/company/pvs-studio/blog/325600/ (дата обращения: 21.08.2017).