PLCopen®

for efficiency in automation

Учебная документация PLCopen

Руководство по использованию метрик качества ПО

версия 1.0, официальный релиз

Дата публикации: 07.11.2023

Переводчик: Е. Кислов
Версия перевода: 1.0
Дата публикации: 16.08.2024

Оглавление

Список авторов и история версий

Данный документ является официальным документом организации **PLCopen**:

Руководство по использованию метрик качества ПО

Он представляет собой результат деятельности рабочей группы и включает вклад каждого из участников:

Участник	**Компания**
Eva-Maria Neumann	Technical University of Munich
Dr.-Ing. Juliane Fischer	Technical University of Munich
Univ.-Prof. Dr.-Ing. Birgit Vogel-Heuser	Technical University of Munich
Bernhard Reiter	CODESYS Group
Sebastian Diehm	Schneider Electric Automation GmbH
Christian Keupp	Schneider Electric Automation GmbH
Mohua Ghosh	Schneider Electric
Denis Chalon	Schneider Electric
Antonio Giusto	Tetra Pak
Chris Freiberg	BHP Group
Bugra M. Yildiz	Software Improvement Group
Michael Stiller	Fraunhofer IKS
Bill Lydon	PLCopen
Eelco van der Wal	PLCopen

История версий

Версия	**Дата**	**Описание**
0.1	30.11.2022	Первая версия, подготовленная Eva-Maria Neumann
0.2	22.12.2022	Дополнения по результатам видеоконференции 22 декабря и многочисленных отзывов по терминологии и описанию сценариев использования метрик
0.3	23.03.2023	Подготовлены текстовые описания процесса разработки программного обеспечения и сценариев использования метрик. Сформирован черновик для получения рекомендаций по структуре документа
0.4	11.04.2023	Выпуск обновлённого шаблона для раздела рекомендаций
0.5	11.05.2023	Дополнения по результатам видеоконференции 11 мая
0.6	25.05.2023	Дополнения по терминологии, сценариям использования и рекомендациям
0.7	21.06.2023	Первая полная (с точки зрения текста) версия черновика документа
0.8	20.07.2023	Дополнения по результатам обратной связи
0.99	06.10.2023	Вариант для внутреннего рецензирования рабочей группой для подготовки версии 1.0
--	02.11.2023	Финальная версия, подготовленная рабочей группой. Не была выпущена
1.0	07.11.2023	Официальный релиз

1. Вступление. Цель создания документа

Компании, работающие в сфере производства промышленного оборудования, сталкиваются с постоянно растущим конкурентным давлением, которое приводит к необходимости проектировать, разрабатывать и внедрять высококачественное программное обеспечение для систем управления. Поскольку ПО приобретает всё большее значение как ключевой элемент функциональности производственных комплексов, многократное повторное использование высококачественного программного обеспечения систем управления является решающим фактором в обеспечении эффективной разработки технический решений этих компаний и, как следствие, обеспечении долгосрочной конкурентоспособности. Поэтому управление качеством ПО систем управления становится всё более важным – как для ПО, разработанного внутри компании (например, для выполнения требований клиентов), так и для «внешнего» ПО, разрабатываемого субподрядчиками. Более того, непрерывное управление качеством может способствовать росту «зрелости» команд разработчиков, потому что повышает осведомлённость о качестве ПО и тому, как на него влияют их проектные решения. Кроме того, процесс оценки качества для каждой версии ПО способен помочь оперативно выявить и «обратить вспять» ухудшение качества, которое, например, может возникнуть из-за увеличения сложности приложения ПЛК в процессе его доработки. Также это потенциально может снизить долгосрочные издержки, возникающие из-за увеличения затрат, необходимых для сопровождения ПО и связанных с его высокой сложностью [1].

В области информационных технологий метрики качества ПО стали подходящим средством объективного измерения и сравнения показателей программного обеспечения. Способы оценки характеристик ПО систем управления с помощью метрик уже существуют и используются в машиностроении и производстве продукции, и многие поставщики платформ автоматизации поддерживают эти подходы с помощью включения инструментов статического анализа кода в свои среды программирования. Однако на практике многие разработчики ПО систем управления по-прежнему неохотно включают расчёт этих показателей в свои процессы разработки – например, из-за отсутствия знаний о том, как использовать их в своей повседневной работе. Надо сказать, что до сих пор оценка качества во многом основывается на опыте разработчиков ПО. Таким образом, количественные показатели качества обладают огромным потенциалом для поддержки разработчиков в принятии решений, основанных на их опыте, путём предоставления объективной информации о характеристиках кода. Более того, для менеджеров метрики ПО могут служить количественными индикаторами, позволяющими оправдать более высокие затраты на обеспечение качества ПО на ранней стадии проекта или выступать в качестве основы для сравнительного анализа ПО, предоставляемого различными поставщиками.

В данном руководстве рассматриваются вопросы внедрения системы оценки качества (основанной на метриках) управляющего ПО, разрабатываемого для промышленных логических контроллеров (ПЛК), в повседневные рабочие процессы, что поможет различным заинтересованным сторонам, связанным с разработкой ПО в компаниях, занимающихся созданием систем управления в отраслях машиностроения и производства продукции. Документ описывает существующие подходы, основанные на исследованиях и инструментах, предоставляемых поставщиками платформ для автоматизации, и рассматривает их применение в различных сценариях использования с учётом специфичных для разных компаний ограничений.

Таким образом, с его помощью внедрение системы оценки качества ПО в повседневные рабочие процессы может быть достигнуто с минимальными усилиями и, в то же время, с максимальной выгодой.

2. Как использовать этот документ

Это руководство предназначено для специалистов в области машиностроения и производства продукции, которые ищут «точку входа» для внедрения метрик качества ПО в свои рабочие процессы. Основное внимание уделяется программному обеспечению систем управления, разрабатываемому на языках стандарта МЭК 61131-3 и выполняемому на промышленных контроллерах (ПЛК), с акцентом на ограничения, существующие в упомянутых выше отраслях. Предполагается, что некоторые аспекты документа могут быть адаптированы и к другим областям применения ПЛК – например, системам автоматизации зданий и системам автоматизации, используемым в промышленном транспорте.

2.1. Моменты, которые нужно учитывать при использовании данного документа

При использовании данного документа, пожалуйста, обратите внимание на следующие моменты:

- основной тематикой данного руководства является расчёт количественных показателей определённого набора атрибутов качества ПО: *удобства сопровождения*, *возможности многократного (повторного) использования*, *тестируемости*, *эффективности* и *надёжности*. Доказано, что эти атрибуты особенно актуальны для программного обеспечения ПЛК в задачах машиностроения и производства продукции. Однако руководство не утверждает, что фокусировка внимания только на этих пяти атрибутах гарантирует всестороннюю оптимизацию качества программного обеспечения;

- целью данного руководства является определение метрик, которые присутствуют в используемых в промышленности инструментах статического анализа кода, тем самым облегчая применение приведённых рекомендаций. В документе рассматриваются следующие инструменты: CODESYS Static Analysis, Schneider Electric EcoStruxure Machine Code Analysis, Schneider Electric – Control Engineering Verification (ранее известный как Itris PLC Checker) и Software Improvement Group Sigrid;

- в данном руководстве рассмотрены типичные сценарии из рабочих процессов создания промышленного программного обеспечения в области машиностроения и производства продукции, в которых применение метрик качества ПО приносит пользу. Документ не претендует на перечисление всех возможных сценариев использования метрик – рассмотрены лишь те, которые являются наиболее актуальными и популярными в промышленности;

- в данном руководстве приведены рекомендации по количественной оценке показателей качества ПО с использованием метрик, доступных в рамках рассматриваемых в документе инструментов статического анализа. Они (рекомендации) предназначены для ознакомления разработчиков программного обеспечения с методами управления качеством. Руководство не пытается охватить всю существующую специфику всех предметных областей, а предоставляет разработчикам наиболее важную информацию и советы по улучшению качества ПО;

- расчёт метрик приносит пользу только тогда, когда его результаты корректно интерпретируются использующими эту информацию людьми. По этой причине количественные значения показателей качества рекомендуется использовать как средство выявления изменений (возможно, непредвиденных) в ПО, а не как повод сразу приниматься за оптимизацию. Целесообразно автоматизировать расчёт показателей и анализировать результаты, чтобы выявлять «аномалии» и своевременно информировать о них все заинтересованные стороны (например, разработчиков, менеджеров по качеству, менеджеров проектов и т. д.).

2.2. Структура документа

В пункте 3 приведены основы теории оценки качества ПО и перечень терминов и определений, использованных в тексте данного документа.

В пункте 4 описан типовой процесс разработки промышленного ПО в области машиностроения и производства продукции, включая конкретные пользовательские сценарии, в рамках которых может быть полезен расчёт метрик качества.

Пункт 5 является ключевой частью документа и содержит конкретные рекомендации по выбору метрик для количественной оценки различных атрибутов качества ПО.

В пункте 6 приводятся примеры расчёта метрик для конкретных примеров кода.

В приложении 1 приведено сопоставление доступных метрик и атрибутов качества ПО.

3. Оценка качества ПО с помощью метрик – термины и определения

В данном пункте приводится теоретическая информация об оценке качества ПО.

В п. 3.1 приводится определения качества ПО и перечисляются его атрибуты.

В п. 3.2 объясняется, как метрики способствуют объективному измерению этих атрибутов.

В п. 3.3 приводятся термины и определения, используемые в тексте данного документа.

3.1. Качество ПО и его атрибуты

Модели качества ПО определяют характеристики качества программного обеспечения, а также взаимосвязи между ними [2]. Согласно стандарту IEEE 1061, атрибуты качества представляют собой «характеристики программного обеспечения или общий термин, обозначающий факторы, подфакторы качества или значения показателей качества». В IT-отрасли за последние десятилетия было разработано множество моделей качества программных продуктов, таких как модель качества Боэма [3] или модель Дроми [4]. Среди наиболее известных моделей качества стоит назвать модель качества программного обеспечения МакКолла [5] и стандарт ISO 25010 [6], которые будут описаны ниже.

МакКолл и его коллеги определяют качество ПО как «общий термин, применимый к любой особенности, характеристике (индивидуальной или общей) или отличительному атрибуту, который указывает на степень совершенства или определяет фундаментальную природу какой-либо сущности». На основе обзора литературы показатели качества были распределены по трём основным категориям (стадиям): внедрение, эксплуатация и доработка (см. рис. 1).

Рис. 1. Модель качества МакКола [5]

Стандарт ISO/IEC 25010 определяет качество ПО как «степень удовлетворения системой заявленных и подразумеваемых потребностей различных заинтересованных сторон, которая позволяет, таким образом, оценить её достоинства». Стандарт описывает 8 характеристик качества, имеющих различные подхарактеристики (см. рис. 2).

Функциональная пригодность	**Уровень производительности**	**Совместимость**	**Удобство использования**
Функциональная полнота Функциональная корректность Функциональная целесообразность	Временные характеристики Использование ресурсов Потенциальные возможности	Сосуществование Интероперабельность	Определимость пригодности Изучаемость Управляемость Защищенность от ошибки пользователя Эстетика пользовательского интерфейса Доступность
Надёжность	**Защищенность**	**Сопровождаемость**	**Переносимость**
Завершённость Готовность Отказоустойчивость Восстанавливаемость	Конфиденциальность Целостность Неподдельность Отслеживаемость Подлинность	Модульность Возможность многократного использования Анализируемость Модифицируемость Тестируемость	Адаптируемость Устанавливаемость Взаимозаменяемость

Рис. 2. Модель качества, описанная в стандарте ISO 25010 [6]

Исследования и анализ кодовой базы, проведённые в различных компаниях машиностроительной отрасли, показали, что следующие характеристики качества:

надёжность, эффективность, сопровождаемость, возможность многократного[1] (повторного) использования, тестируемость

имеют важное значение в контексте развития технологических трендов, оставаясь при этом актуальными в течение десятилетий [7]. Сравнение определений, используемых МакКоллом и стандартом ISO 25010, приведённое в таблице 1 (см. ниже), показывает, что они в существенной степени совпадают и дополняют друг друга. В данном руководстве используется комбинация определений из обоих источников.

[1] Данный термин использован в российском переводе стандарта (ГОСТ Р ИСО/МЭК 25010—2015). Далее в тексте документа будет использоваться более привычный для российской технической литературы вариант «повторное использование кода» (прим. переводчика)

Табл. 1. Сравнение определений некоторых характеристик качества по МакКоллу и ISO 25010

	Определение МакКолла [5]	Определение ISO 25010 [6]
Надёжность	Степень, в которой программа соответствует своим **спецификациям** и выполняет **задачи пользователя** (подхарактеристика стадии **эксплуатации продукта**)	Степень выполнения системой, продуктом или компонентом определённых функций при указанных условиях в течение установленного периода времени. *Подхарактеристика*: **завершённость** – степень соответствия системы, продукта или компонента при **нормальной работе** требованиям **надёжности**
Уровень производительности	Объём **вычислительных ресурсов** и кода, необходимый программе для выполнения своих функций (подхарактеристика стадии **эксплуатации продукта**)	Производительность относительно суммы **использованных ресурсов** при определённых условиях. *Подхарактеристика*: **временные характеристики** – степень соответствия требованиям по времени отклика, времени обработки и показателям пропускной способности продукта или системы
Сопровождаемость	Усилия, необходимые для **обнаружения** и **исправления ошибки** в используемой программе (подхарактеристика стадии **доработки продукта**)	**Результативность** и **эффективность**, с которыми продукт или система могут быть **модифицированы** предполагаемыми специалистами по обслуживанию
Возможность многократного использования	Степень, в которой программа **может использоваться в других приложениях** — в зависимости от объёма функций, которые она выполняет	**Возможность многократного использования**: степень, в которой актив может быть использован в нескольких системах или в создании других активов. **Модульность**: степень представления системы или компьютерной программы в виде отдельных блоков таким образом, чтобы изменение одного компонента оказывало **минимальное воздействие на другие компоненты**
Тестируемость	**Усилия**, необходимые для проведения **тестирования программы**, которое подтвердит, что она выполняет свою предполагаемую функцию	Степень простоты эффективного и рационального определения для системы, продукта или компонента критериев тестирования, а также простоты выполнения тестирования с целью определения соответствия этим критериям (подхарактеристика **сопровождаемости**)

3.2. Метрики качества ПО

Метрики доказали свою пригодность для количественной оценки характеристик качества на основе числовых и статистических характеристик программного обеспечения. Стандарт IEEE 1061 [8] определяет метрику как «функцию, входными данными которой является информация о ПО, а выходными – числовое значение, которое можно интерпретировать как степень проявления у программного обеспечения заданной характеристики, влияющей на его качество».

Метрики могут применяться на различных уровнях детализации архитектуры ПО. Набор метрик, разработанный Кемерером и Чидамбером [9] фокусируется на ПО, разработанном в объектно-ориентированном стиле с использованием структурного подхода к проектированию. Это подразумевает, что процесс расчёта метрик фокусируется не только на исходном коде отдельных программных модулей (**POU**; см. определение в п. 3.3), но и на их структурных взаимосвязях между собой. С другой стороны, некоторые метрики фокусируются на внутренних параметрах конкретных POU – например, их сложности (см., например, цикломатическую сложность МакКейба [10] или метрики сложности Холстеда [11]). В целом, метрики позволяют произвести количественную оценку определённой характеристики качества (например, тестируемости), которая может быть рассчитана на основе определённой комбинации параметров ПО.

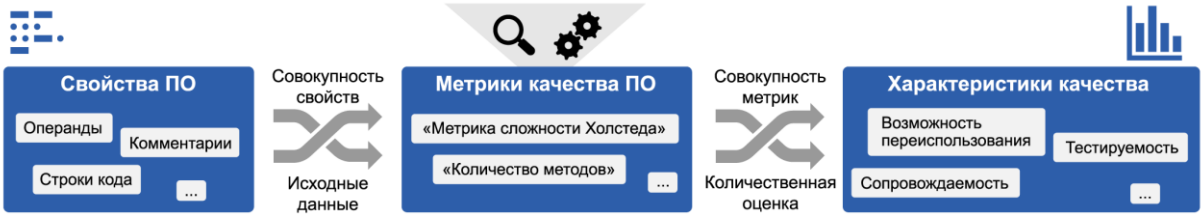


Рис. 3. Взаимосвязь между свойствами ПО, характеристиками качества и метриками качества [12]

В рамках данного руководства метрики сгруппированы по перечисленным ниже категориям. Список категорий был сформирован путём изучения метрик, доступных в коммерческих инструментах статического анализа и описанных в тематической профессиональной литературе.

- **Метрики, связанные с размером** – например, число строк кода POU или размер его данных в байтах;
- **Переменные и интерфейсы POU** – метрики, связанные с областью объявления POU;
- **Встроенная документация** – метрики, связанные с комментариями в области объявления и области реализации POU;
- **Потоки данных** – метрики, связанные с вызовами объектов и потоками информации, передаваемыми в POU и получаемыми от них;
- **Сложность ПО** – метрики, связанные со сложностью программного обеспечения;
- **МЭК ООП-элементы** – метрики, связанные с элементами объектно-ориентированного программирования, описанными в стандарте МЭК 61131-3 (интерфейсами, методами, свойствами, наследованием);

- **Элементы, специфичные для конкретного языка** – метрики, связанные с элементами, специфичными для конкретных языков стандарта МЭК 61131-3 (например, FBD и SFC);
- **Индикаторы возможности повторного использования** – метрики, которые дают представление о возможности повторного использования отдельных POU в рамках проектов или библиотек.

3.3. Термины и определения

В этом разделе в алфавитном порядке приведены термины и определения, использованные в данном руководстве. Формулировки взяты из стандартов МЭК 61131-3 [13] и ISO/IEC/IEEE 24765 [14], документации известных производителей ПЛК и общих практик разработки программного обеспечения.

Приёмочные испытания (Acceptance Tests) – испытания системы или единицы оборудования после её установки, обычно проводимые заказчиком на его территории при участии поставщика для проверки соблюдения требований, перечисленных в договоре поставки.

Действие (Action) – логическая переменная или набор подлежащих выполнению операций вместе со связанной с ними управляющей конструкцией.

Гибкая методология разработки ПО (Agile software development) – методология создания ПО, основанная на идее итеративной разработки, в рамках которой формирование требований и их реализация выполняются за счёт сотрудничества самоорганизующихся кросс-функциональных команд. Позволяет командам быстрее создавать «ценности для бизнеса» (т.е. выпускать новые версии продуктов) с высоким уровнем качества в рамках предсказуемого производственного процесса и оперативно реагировать на изменения, происходящие на рынке.

Прикладное программное обеспечение (Application software) – файл проекта, специфичный для конкретной системы (или производственной установки), и исполняемый на одном или нескольких ПЛК. Обычно содержит реализацию алгоритмов работы системы и человеко-машинный интерфейс (HMI). *Синонимы*: проект ПЛК, приложение ПЛК.

Вызов (Call) – языковая конструкция, вызывающая выполнение функции, функционального блока или метода.

Класс (Class) – программный компонент (POU), состоящий из:

- определения структуры данных;
- набора методов, выполняемых над этой структурой.

Класс является реализацией; он содержит конкретную структуру и конкретный набор методов. Типом для класса является *интерфейс*.

Связность (Cohesion) – степень, в которой часть кодовой базы образует логически единую атомарную единицу (модуль, класс, функцию). Зависит от количества связей внутри кодовой единицы.

Конфигурация (Configuration) – элемент языка, соответствующий системе (в широком смысле) программируемого контроллера (см. рис. 4).

CI/CD

- *CI – Continuous Integration* (Непрерывная интеграция): практика разработки ПО, которая заключается в постоянном слиянии изменений, выполняемых различными разработчиками, в общую основную ветвь разработки;
- *CD – Continuous Delivery/Development* (Непрерывная доставка/развёртывание):
  - *Непрерывная доставка*: изменения, внесённые разработчиком в приложение, автоматически проверяются на наличие ошибок и загружаются в репозиторий, откуда их затем можно перенести в рабочее окружение;
  - *Непрерывное развёртывание*: автоматический перенос изменений, внесённых разработчиком, из репозитория в рабочее окружение, где их могут использовать клиенты или другие заинтересованные стороны.

Сцепление (Coupling) – степень взаимозависимости между различными модулями, компонентами или классами в программной системе. Определяет, в какой мере внесение изменений в одном модуле требуют изменений в других модулях.

Объявление (Declaration) – механизм для определения элемента языка программирования.

Глобальные переменные (Global variables) – переменные с глобальной областью действия (их можно использовать в любом POU проекта).

Программная модель МЭК 61131-3 (IEC 61131-3 Software model) – основные элементы языка стандарта МЭК 61131-3 и их взаимосвязи, показанные на рис. 4. К этим элементам относятся программы, функциональные блоки, классы, функции и элементы конфигурации – ресурсы, задачи, глобальные переменные, пути доступа к объектам и механизмы инициализации параметров ПЛК, специфичные для конкретной среды разработки.

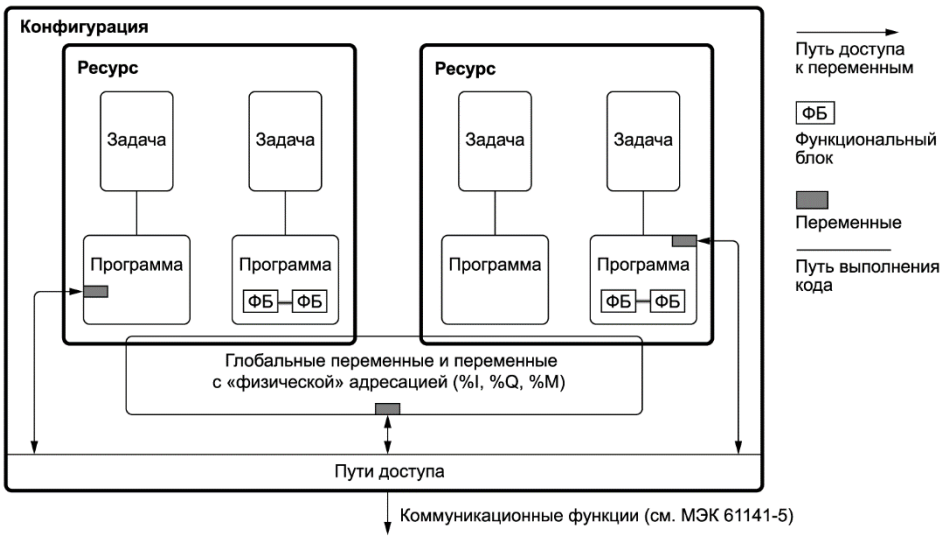


Рис. 4. Программная модель, описанная в стандарте МЭК 61131-3 [13]

Поток данных (Information flow) – механизм передачи данных между различными POU. Может быть прямым (через передачу/получение значений при вызове POU), так и косвенным (через чтение/запись глобальных переменных).

Наследование (Inheritance) – создание нового класса, функционального блока или интерфейса на основе существующего класса, функционального блока или интерфейса соответственно.

Входная переменная (Input variable) – переменная, используемая для передачи значения программному компоненту (POU), который не является классом.

Экземпляр (Instance) – отдельная именованная копия структуры данных, связанная с функциональным блоком, классом или программой и сохраняющая свои значения между вызовами данного объекта.

Создание экземпляра (Instantiation) – создание экземпляра.

Интерфейс (Interface) – элемент языка программирования, относящийся к парадигме объектно-ориентированного программирования и представляющий собой набор прототипов методов.

Техническое сопровождение (Maintenance activity) – деятельность по устранению проблем (корректирующее сопровождение), внедрению новых функций (полное сопровождение), внесению улучшений без изменения поведения (профилактическое сопровождение) или модификации программного обеспечения для адаптации к изменившимся условиям (адаптивное сопровождение).

Метод (Method) – элемент языка программирования, похожий на функцию, который создаётся в рамках функционального блока или класса, и в пределах которого обеспечен неявный доступ к переменным экземпляра функционального блока или класса.

Пример: функциональный блок управления котлом может иметь методы **Fill** и **HeatUp**, каждый из которых реализует соответствующую функциональность.

Выходная переменная (Output variable) – переменная, используемая для возврата значения из программного компонента (POU), который не является классом.

Проект ПЛК (PLC project) – совокупность прикладного программного обеспечения и используемых им библиотек (см. рис. 4).

Синоним: приложение ПЛК, программа ПЛК (не путать с POU типа «программа»).

POU (Program Organization Unit) – программный компонент. Исходный код проекта ПЛК состоит из POU. К POU относятся функции, функциональные блоки, классы и программы (в терминах стандарта МЭК 61131-3).

Ресурс (Resource) – элемент языка программирования, соответствующий «функции обработки сигналов» и её «человеко-машинному интерфейсу» и «функциям интерфейса датчиков и исполнительных механизмов», в случае наличия таковых (см. рис. 4).

Библиотека (Software library) – набор стандартизированного, нацеленного на повторное использование кода, который обычно охватывает конкретную предметную область или функциональность, и включает в себя:

- интерфейсы, их методы и свойства;
- пользовательские типы данных (перечисления, структуры, псевдонимы, объединения);
- глобальные переменные, константы, списки параметров;
- списки текстов, пулы изображений, экраны визуализации, элементы визуализации;
- дополнительные файлы (например, файлы документации).

Добавление библиотеки в проект ПЛК позволяет использовать её модули таким же образом, как и те функциональные блоки и переменные, которые созданы и объявлены непосредственно в проекте.

Шаблон проекта (Software template) – предварительно разработанный и доступный для редактирования файл проекта ПЛК, включающий модули общего назначения, которые могут быть эффективно и быстро адаптированы разработчиками к требованиям конкретной системы или производственной установки. Шаблон может быть открыт отдельно и часто служит «отправной точкой» для разработки проектов.

Программный фреймворк (Software framework) – платформа, помогающая разработчику при разработке ПО (проектов ПЛК и библиотек). Обычно представляет собой набор инструментов, но может также включать библиотеки, примеры кода и, например, руководства по программированию. Таким образом, рекомендации из данного документа предназначены для интеграции в используемый читателем фреймворк.

Задача (Task) – элемент контроля выполнения, обеспечивающий циклическое или ациклическое (по команде) выполнение группы связанных между собой программных компонентов (см. рис. 4).

Юнит-тестирование (Unit test) – тестирование отдельных программных модулей (автоматизированное или ручное) разработчиком или независимым тестировщиком для обеспечения отсутствия ошибок (логических или программных).

Каскадная модель разработки ПО (Waterfall procedure model) – линейная, процедурно-неитеративная модель разработки программного обеспечения, состоящая из последовательно выполняемых фаз. Завершение предыдущей фазы проекта является необходимым условием для начала следующей.

4. Процесс разработки ПО

4.1. Типовой процесс разработки ПО для ПЛК

На рисунке ниже приведён пример процесса разработки ПО для ПЛК, который будет использоваться для иллюстрации различных сценариев, в рамках которых расчёт метрик качества может быть полезен. Эта польза заключается в поддержке субъективного опыта разработчиков по оценке сложности ПО с помощью количественных показателей характеристик качества, рассматриваемых в данном руководстве (см. п. 3.1).

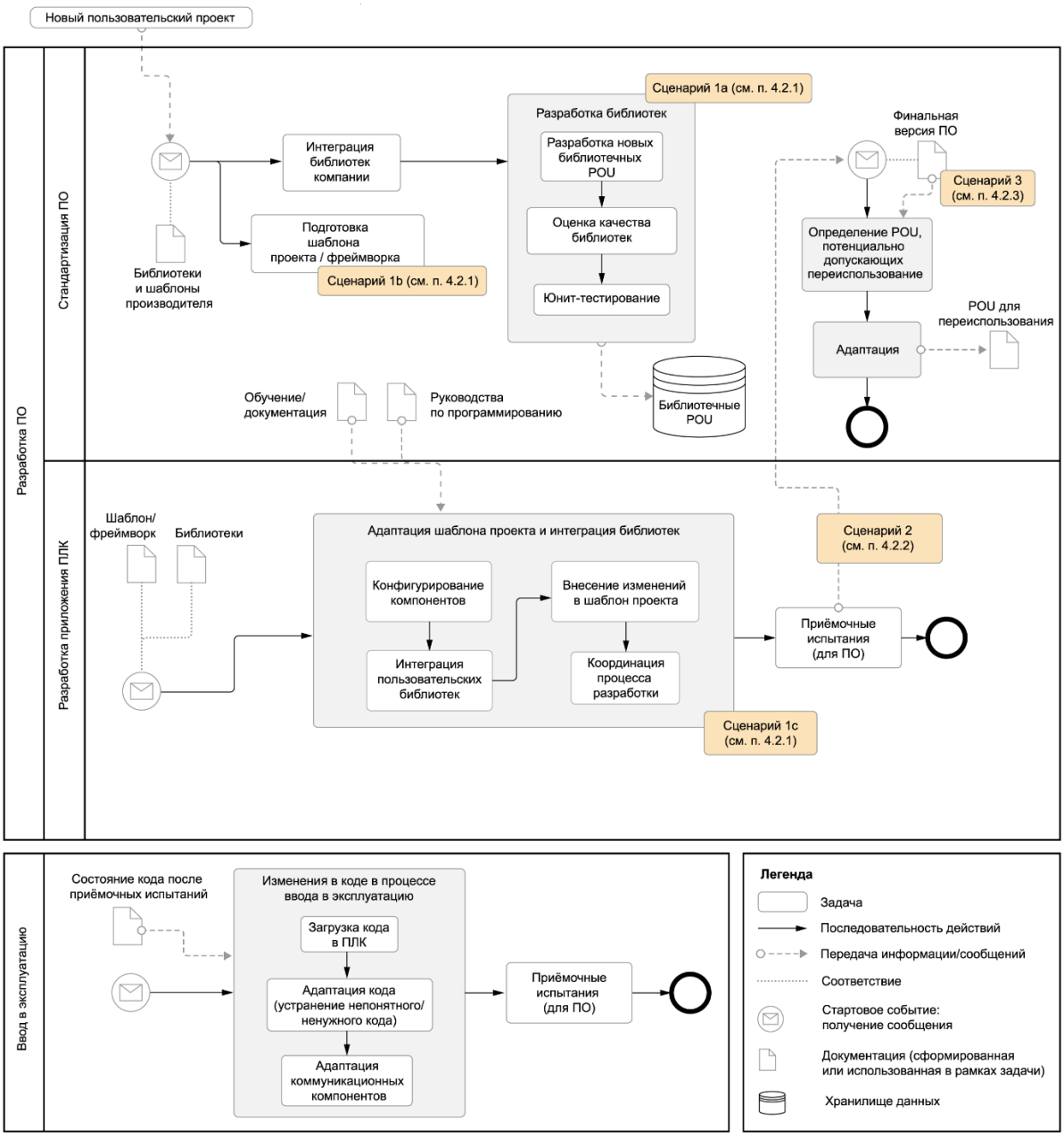


Рис. 5. Обзор типового процесса разработки промышленного ПО, который в схожей форме наблюдался в различных компаниях в отраслях машиностроения и производства продукции

Рабочий процесс состоит из двух основных частей: собственно разработка ПО (включая его стандартизацию и разработку приложения ПЛК) и ввод в эксплуатацию. Важно отметить, что представленный на рисунке выше пример рабочего процесса используется лишь для демонстрации преимуществ, которые может принести расчёт метрик качества в ситуациях, встречающихся в реальной жизни. Он не претендует на охват всех аспектов разработки ПО, которые можно встретить в промышленной отрасли. Даже в том случае, если разработка ПО в конкретной компании отклоняется от рассматриваемого базового рабочего процесса (например – не используются библиотеки), метрики могут быть полезны в сценариях, схожих с теми, что рассмотрены в данном разделе.

Процесс стандартизации ПО (см. верхний сегмент на рис. 5) включает стандартизацию повторно используемых программных компонентов (POU), которые обычно сгруппированы по библиотекам и могут использоваться в приложениях ПЛК, создаваемых для разных систем и установок. Соответственно, такие POU должны быть «заслуживающими доверия» и гарантировать надёжное функционирование в различных приложениях с разными граничными условиями; обычно это обеспечивается всесторонним модульным тестированием. Компании, которые занимаются выпуском серийной продукции, часто используют стандартизированные шаблоны, входящие в состав применяемого ими фреймворка, которые могут обеспечивать интерфейс для инфраструктуры приложения – например, для обработки исключений или обмена данными с HMI. Шаблоны могут использоваться в качестве основы или примера на стадии разработки приложения ПЛК.

В процессе разработки приложения ПЛК происходит интеграция в проект библиотек. В случае использования шаблонов – эти шаблоны адаптируются к функциональным требованиям разрабатываемой системы или производственной установки. Функциональность и надёжность ПО впоследствии проверяется на приёмочных испытаниях.

В области машиностроения и, особенно, производства продукции программное обеспечение часто дорабатывается во время ввода в эксплуатацию – например, если функциональность системы должна быть исправлена или изменена в сжатые сроки (или даже уже в процессе эксплуатации). Возможность внесения изменений в приложение «на горячую» (без остановки процесса управления) в рассматриваемых отраслях является крайне востребованной функцией ПЛК (по наблюдениям разработчиков статического анализатора **Schneider Electric Control Engineering – Verification** – она требуется примерно 25% клиентов). Поэтому данный этап включен в базовый рабочий процесс (см. рис. 5), чтобы учитывать эти потенциально возможные изменения ПО при использовании оценки качества, основанной на метриках.

4.2. Сценарии использования метрик качества в рамках процесса разработки ПО

Предупреждение о рассматриваемых в документе сценариях

При использовании рекомендаций из документа обратите внимание на следующие моменты:

- общая рекомендация по использованию метрик: при внесении каких-либо изменений в ПО всегда оценивайте, как они влияют на аспекты качества;
- пожалуйста, учитывайте особенности вашего рабочего процесса, системы и других сущностей при адаптации к ним рекомендаций из данного документа. Применимость конкретных сценариев зависит от заинтересованных сторон и нюансов процесса разработки.

В следующих подпунктах приведён обзор типичных сценариев использования метрик качества ПО. Эти сценарии основаны на реальных ситуациях, возникающих в компаниях из отрасли машиностроения и производства продукции, и демонстрируют пользу количественной оценки характеристик качества программного обеспечения.

Описание каждого сценария содержит ряд пунктов, включающих в себя всю необходимую информацию:

- **Заинтересованные стороны**: роли (должности), задействованные в конкретном сценарии, которые могут извлечь пользу из расчёта метрик качества;
- **Описание**: краткое описание сценария;
- **Этап рабочего процесса**: конкретная точка рабочего процесса, в которой могут быть рассчитаны метрики;
- **Возможные вариации и ограничения**: примеры возможных изменений и альтернативных вариантов в рамках сценария использования, а также предполагаемые ограничения;
- **Затронутые характеристики качества**: перечисление характеристик качества, для которых в данном сценарии производится расчёт метрик;
- **Полученная выгода**: предпосылки использования метрик в рамках данного сценария и получаемая с их помощью выгода.

Существуют три основных сценария интеграции процедуры расчёта метрик качества в процесс разработки промышленного ПО. В зависимости от специфических для вашей компании условий, необходимо в рамках проекта по разработке выбрать из них один конкретный, который лучше всего подходит для ваших целей и потребностей:

- непрерывная оценка качества на протяжении всего цикла разработки ПО;
- оценка качества существующего ПО в определённых точках рабочего процесса (например, сравнение метрик, рассчитанных до и после внесения изменений в приложение ПЛК);
- оценка качества ПО, разрабатываемого «с нуля».

4.2.1. Сценарий 1: непрерывная оценка качества POU в процессе разработки ПО

В идеальном случае оценка качества POU непрерывно выполняется в течение всего его жизненного цикла и производится каждый раз при внесении в код каких-либо изменений. Хотя это обычная практика для разработки на языках высокого уровня (например, Java или C#) – она не получила стандартизации и широкого распространения в сфере разработки приложений для ПЛК. Тем не менее, непрерывная оценка качества ПО и оперативное выявление отклонений метрик качества от целевых показателей является весьма полезным, так как позволяет в долгосрочной перспективе избежать затратного по времени и ресурсам рефакторинга кода. Далее данный сценарий и его возможные вариации описываются с учётом вышеупомянутых фактов.

Заинтересованные стороны: разработчики библиотек, разработчики приложений, менеджеры проектов, клиенты;

Описание: после каждого значительного изменения (например, исправления ошибки, добавления новой функции и т. д.) выполняется оценка качества перед выпуском новой версии (или варианта) POU для обеспечения непрерывного мониторинга качества;

Этап рабочего процесса: любой этап рабочего процесса, на котором ПО может измениться в течение его жизненного цикла, например:

- a) после изменений в библиотеках POU (например, при исправлении багов, найденных при юнит-тестировании, или добавлении нового функционала);
- b) при подготовке шаблонов и фреймворка проекта (например, их адаптация к требованиям разработчиков приложений и исправление ошибок, найденных при юнит-тестировании и приёмочных испытаниях);
- c) после изменений шаблонов или фреймворка, связанных с их адаптацией к специфике конкретного проекта.

Возможные вариации и ограничения:

- *CI/CD с системой контроля версий*: автоматическая непрерывная оценка качества ПО после каждого изменения перед отправкой изменений в ветку main (в случае использования системы контроля версий, основанной на Git). Если результат оценки не удовлетворяет целевым показателям – то выявленные проблемы должны быть исправлены или прокомментированы;
- *Гибкая методология разработки ПО (Agile)*: периодический расчёт метрик и сравнение их значений с аналогичными значениями на начало спринта. Эффективность оценки может быть повышена путём автоматизации этого процесса. В рамках Agile оценка качества может быть частью критериев готовности (Definition-of-Done);
- *Каскадная модель разработки ПО (Waterfall)*: интеграция регулярных расчётов метрик в рабочий процесс на этапе, который предшествует вводу в эксплуатацию.

Затронутые характеристики качества: сопровождаемость, возможность повторного использования, тестируемость, уровень эффективности, надёжность;

Полученная выгода: отклонения от целевых показателей качества обнаруживаются на ранней стадии проекта и могут быть вовремя исправлены. Это позволяет в долгосрочной перспективе избежать затрат на исправление ошибок и дорогостоящего сервисного сопровождения, а также избежать дорогостоящего рефакторинга ПО на поздних стадиях проекта по разработке и выполнять его оптимизацию «на лету».

4.2.2. Сценарий 2: оценка качества кода до и после ввода системы в эксплуатацию

В процессе ввода системы в эксплуатацию на реальном объекте, программное обеспечение часто подвергается всё продолжающимся и продолжающимся изменениям. Из-за дефицита времени эти изменения иногда вносятся на скорую руку техническими специалистами, не имеющими продвинутых навыков программирования. Метрики качества ПО в этом случае могут, например, помочь определить POU, код которых чаще всего менялся на этапе пусконаладки. Это, в свою очередь, может быть симптомом сложности кода (по крайней мере, для инженеров-наладчиков) и быть стимулом для его рефакторинга.

Заинтересованные стороны: инженеры, запускающие систему в эксплуатацию, разработчики приложений ПЛК, менеджеры;

Описание: после ввода системы в эксплуатацию оцениваются изменения в характеристиках качества, связанные с переписыванием фрагментов проекта ПЛК в процессе пусконаладки. Это позволяет определить POU, код которых чаще всего менялся в этот отрезок времени.

Этап рабочего процесса: после приёмочных испытаний;

Возможные вариации и ограничения:

- инженеры, занимающиеся запуском системы в эксплуатацию, имеют доступ ко всему исходному коду проекта ПЛК – и оценка качества выполняется для всего кода;
- инженеры, занимающиеся запуском системы в эксплуатацию, имеют доступ лишь к некоторым фрагментам исходного кода проекта ПЛК – и оценка качества выполняется только для этих фрагментов.

Затронутые характеристики качества: сопровождаемость, возможность повторного использования, уровень эффективности, надёжность;

Полученная выгода: метрики качества могут использоваться для определения фрагментов ПО, которые наиболее сильно изменились в процессе ввода системы в эксплуатацию. Это может являться признаком того, что данные фрагменты было трудно понять инженерам-наладчикам или же они мешали завершению пусконаладки – поэтому следует провести их анализ и, возможно, доработку.

Метрики могут автоматически идентифицировать те части ПО, которые обычно не должны редактироваться во время ввода системы в эксплуатацию и поэтому должны быть «защищены» от непреднамеренных изменений.

4.2.3. Сценарий 3: аудит объекта / установки после завершения проекта

В активно развивающейся и постоянно изменяющейся отрасли промышленной автоматизации важность высококачественного ПО продолжает расти. Вот почему компании всё больше заинтересованы в целенаправленном и эффективном использовании потенциала оптимизации программного обеспечения. Одним из возможных подходов является аудит объекта или установки после завершения проекта.

Метрики качества могут использоваться для объективной оценки основных характеристик ПО, не требуя при этом глубоких познаний о его исходном коде. Поэтому данный сценарий особенно интересен с точки зрения менеджмента – в тех ситуациях, когда решения об оценке качества принимаются путём сравнения затрат на эту процедуру и потенциальной выгоды от неё.

Заинтересованные стороны: менеджеры, сотрудники отдела качества (если такие имеются);

Описание: выбранные метрики рассчитываются для всех программ ПЛК с целью их классификации по размеру и сложности, а также для выявления потенциального технического долга и расстановки приоритетов по задачам, связанным с будущими доработками. В этом сценарии планирование оценки качества осуществляется не самими разработчиками, а менеджерами; его итоги позволяют им принять правильные бизнес-решения.

Этап рабочего процесса: после приёмочных испытаний или завершения проекта;

Возможные вариации и ограничения:

- для объекта: сопоставление метрик для различных ПЛК в пределах одного объекта (например, фабрики) с целью сравнения эффективности различных установок/производственных линий в пределах системы;
- для установок (но применимо и к объекту): сравнение метрик для различных типов установок.

Затронутые характеристики качества: сопровождаемость, возможность повторного использования, тестируемость, уровень эффективности, надёжность;

Полученная выгода: расчёт количественных показателей, на основании которых можно будет обсуждать приоритеты задач, связанных с рефакторингом. Эффективность оптимизации в различных проектах может быть выражена в виде числовых оценок. Недостатки качества (в т. ч. повторяющиеся) могут быть автоматически определены в разных проектах, что приведёт к выявлению «скрытых» проблем, влияющих на качество ПО, и укажет на возможные меры оптимизации.

4.2.4. Другие сценарии, в рамках которых расчёт метрик качества может быть полезен

- проверка качества стороннего кода (разработанного другими компаниями);
- для выбора поставщика ПО: руководство по применению метрик качества может быть включено в контракт на разработку;
- определение функций ПО, которые можно улучшить путём рефакторинга существующей кодовой базы;
- проверка качества во время приёмочных испытаний.

5. Рекомендации по использованию метрик при разработке ПО для ПЛК

В IT-отрасли существует набор инструментов (например, SonarQube от Sonar и Sigrid от Software Improvement Group), рекомендаций и руководств (например, Common Weakness Enumeration) по улучшению качества программного обеспечения. В области систем промышленной автоматизации такие инструменты и руководства разрабатываются поставщиками платформ (например, Schneider Electric и CODESYS Group), а также независимыми организациями – такими, как PLCopen. Все эти инструменты, руководства и рекомендации имеют общую черту – они призваны помочь разработчикам ПО в правильном расчёте метрик и интерпретации результатов, и оперируют определёнными категориями.

В рамках данного руководства для единообразного оформления рекомендаций они тоже распределены по категориям. Набор этих категорий сформирован на основе консенсуса и опирается на известные инструменты и руководства; он должен быть понятен для сторон, заинтересованных в улучшении качества ПО.

- **Пример «некачественного» кода**: негативный пример исходного кода, не обеспечивающего определённую характеристику качества, и пояснение, как можно сделать этот факт явным с помощью рассматриваемых метрик;
- **Пример «качественного» кода**: положительный пример исходного кода, обеспечивающего определённую характеристику качества, и пояснение, как можно сделать этот факт явным с помощью рассматриваемых метрик;
  Примечание: фрагменты «качественного» и «некачественного» кода взяты из учебного, «идеализированного» примера управления водонагревателем, который не призван показать весь возможный спектр функционала ПО для ПЛК, применяемого в промышленности. Для анализа кода использовался инструмент CODESYS Static Analysis.
- **Реализация в инструментах статического анализа**: примеры метрик, доступных в платформах промышленной автоматизации и инструментах статического анализа, которые могут использоваться для количественного измерения определённого фактора качества. В рекомендациях данного документа рассматриваются платформы и инструменты, перечисленные ниже. В будущих версиях руководства список будет расширен.
  - CODESYS – Static Analysis (обозначение: CODESYS Static Analysis);
  - Schneider Electric – EcoStruxure – Machine Advisor Code Analysis (обозначение: SE – EcoStruxure MACA);
  - Schneider Electric – EcoStruxure Control Engineering – Verification (обозначение: SE – EcoStruxure CE-V);
  - Software Improvement Group – Sigrid (обозначение: SIG Sigrid Maintainability and Architecture Quality – AQ).
- **Возможные меры по улучшению качества**: действия, которые можно выполнить, или аспекты, которые следует учитывать для повышения степени соответствия кода рассматриваемой характеристике качества путём корректировки влияющих на неё факторов.

Поскольку основное внимание в данном руководстве уделяется оценке качества ПО для ПЛК на основе метрик, то добавлены дополнительные категории, поясняющие детали применения рекомендаций:

- **Сценарии и шаги рабочего процесса**: сценарии использования, в рамках которых измерение данной характеристики качества принесёт пользу (см. п. 4);
- **Потенциальные конфликты**: характеристики качества, которые могут ухудшиться при оптимизации данной характеристики качества;
- **Ссылки на другие руководства**: ссылки на другие руководства PLCopen, к которым можно обратиться для получения более детальной информации по рассматриваемым аспектам.

Изложенные ниже рекомендации сгруппированы по целевым характеристикам качества ПО:

- сопровождаемость (п. 5.1);
- возможность повторного использования (п. 5.2);
- тестируемость (п. 5.3);
- уровень эффективности (п. 5.4);
- надёжность (п. 5.5).

Для определения метрик, подходящих для количественной оценки соответствующих характеристик, было проведено два семинара с привлечением международных экспертов в области разработки программного обеспечения для промышленных ПЛК. Эксперты рассмотрели точки зрения производителей платформ автоматизации («поставщиков» метрик) и производителей систем автоматизации, разработанных с помощью этих платформ («пользователей» метрик). В ходе семинаров метрики, доступные в статических анализаторах этих платформ и описанные в научной литературе, были сопоставлены с характеристиками качества ПО (подробнее см. в Приложении 1).

Рассматриваемые метрики сгруппированы по уровню квалификации, необходимому для их корректного расчёта и интерпретации результатов (см. п. 3.2):

Начальный уровень в использовании метрик:

- метрики, связанные с размером;
- элементы, специфичные для конкретного языка;
- сложность ПО;
- встроенная документация.

Продвинутый уровень в использовании метрик:

- переменные и интерфейсы POU;
- потоки данных;
- индикаторы возможности повторного использования.

Экспертный уровень в использовании метрик:

- МЭК ООП-элементы.

Сравнение используемых статических анализаторов приведено в Приложении 1. В следующих пунктах приведены рекомендации по измерению пяти упомянутых выше целевых характеристик качества, сформулированные в ходе экспертных семинаров.

5.1. Метрики, связанные с сопровождаемостью

Сценарии и шаги рабочего процесса (см. п. 4.2):

- **Сценарий 1**: непрерывная оценка качества POU в процессе разработки ПО.
  Мотивация: проверить, влияет ли конкретное изменение на сопровождаемость ПО и наблюдать, как изменяется степень сопровождаемости с течением времени;
- **Сценарий 2**: оценка качества кода до и после ввода системы в эксплуатацию.
  Мотивация: проверить, повлияли ли изменения, произведённые в ПО на этапе пусконаладки, на его сопровождаемость и требуется ли ревизия изменённых фрагментов кода.
- **Сценарий 3**: аудит объекта / установки после завершения проекта.
  Мотивация: сравнить сопровождаемость ПО данного проекта с другими проектами, чтобы выявить его преимущества и недостатки, а также определить возможные точки оптимизации в проектных решениях.

Метрики, связанные с размером
Размер POU ↑ **Высокий показатель**: 👎 ↓ **Низкий показатель**: 👍 **Причина**: для понимания принципа работы POU большого размера требуется значительное время, что затрудняет его сопровождение. **Реализация в инструментах статического анализа**: • CODESYS Static Analysis: Lines Of Code • SE – EcoStruxure MACA: Number of instructions • SE – EcoStruxure CE-V: NOS – Number Of Statement • SIG Sigrid Maintainability: Unit Size
Количество действий ↑ **Высокий показатель**: 👍 ↓ **Низкий показатель**: 👎 **Причина**: инкапсуляция фрагментов кода в небольшие программные модули (например, действия) облегчает сопровождение ПО. **Реализация в инструментах статического анализа**: • SE – EcoStruxure MACA: Number Of Actions • SE – EcoStruxure CE-V: plcobjecttype counter • SIG Sigrid Maintainability: Unit Size

⃠ **Потенциальные конфликты**: ↓ *уровень эффективности* может снизиться из-за увеличения количества вызовов модулей.
Возможные меры по улучшению качества: • обязательно используйте принцип единой ответственности: выделяйте код, не связанный с основной задачей POU, в другие POU или, если он нужен для вспомогательной функциональности, в действия и методы вашего POU; • для структурирования кода POU выделяйте его отдельные, функционально связанные фрагменты в действия и методы; • избегайте дублирования фрагментов кода путём использования «наследуемых» функциональных блоков, а также структур и массивов.

Метрики, связанные с элементами, специфичными для конкретного языка
Количество шагов, переходов и ветвей в SFC ↑ **Высокий показатель**: 👎 ↓ **Низкий показатель**: 👍 **Причина**: следует избегать создания очень больших POU на языке SFC – особенно со значительным числом ветвей, из-за которых происходит «распухание» схемы в ширину; это затрудняет отслеживание изменений, происходящих при редактировании таких POU. **Реализация в инструментах статического анализа**: • CODESYS Static Analysis: Number of SFC branches, Number of SFC steps • SE – EcoStruxure MACA: Number Of Transitions • SE – EcoStruxure CE-V: nbofbranches, g7height, g7width
Количество цепей в FBD ↑ **Высокий показатель**: 👎 ↓ **Низкий показатель**: 👍 **Причина**: большое количество цепей (networks) в POU, написанном на языке FBD, может помешать быстро понять его функционал, тем самым затрудняя сопровождение. Этот эффект усиливается, если отдельные цепи сложны сами по себе. **Реализация в инструментах статического анализа**: • SE – EcoStruxure MACA: Number Of FBD Networks, Halstead Complexity for FBD
Возможные меры по улучшению качества: • если POU на SFC/FBD невозможно охватить взглядом (требуется увеличение масштаба или прокрутка окна редактора), то следует рассмотреть вариант распределения его функциональности по нескольким отдельным POU; • если отдельные цепи FBD сложны для понимания, то попробуйте распределить их функциональность по нескольким отдельным цепям; • используйте в FBD функциональные блоки с высоким уровнем абстракции, решающие конкретные задачи из предметной области (например, управление насосом, обработка аварий и т. д.), а не простые «базовые» блоки (триггеры, счётчики, таймеры); это приводит к уменьшению количества и размеров цепей.

Метрики, связанные переменными и интерфейсами POU
Использование глобальных переменных ↑ **Высокий показатель**: 👎 ↓ **Низкий показатель**: 👍 **Причина**: использование глобальных переменных затрудняет понимание того, как происходит передача данных между различными POU; такие потоки информации сложно отследить, и они могут приводить к нежелательным зависимостям. **Реализация в инструментах статического анализа**: • CODESYS Static Analysis: Used different global variables • SE – EcoStruxure MACA: Number of GVL Usages • SE – EcoStruxure CE-V: extvarref
Количество переменных в интерфейсе POU ↑ **Высокий показатель**: 👎 ↓ **Низкий показатель**: 👍 **Причина**: большое количество входов-выходов в одном POU приводит к его сильной зависимости от других объектов (формирующих данные, передаваемые в этот POU, и использующих данные, получаемые от него), что может привести к нежелательным перекрёстным эффектам (т. е. редактирование одного POU потребует внесения изменений в другие POU, связанные с ним). **Реализация в инструментах статического анализа**: • CODESYS Static Analysis: Number of input/output variables • SE – EcoStruxure CE-V: inputcount, outputcount, nbofparam • SIG Sigrid Maintainability: Unit Interfacing
Возможные меры по улучшению качества: • проверьте, является ли использование глобальных переменных действительно необходимым и могут ли быть нужные для POU данные получены им «напрямую», через входные переменные; • если по каким-то причинам отказаться от использования глобальных переменных нельзя, то проверьте, что они сгруппированы логичным образом (например, функционально связанные переменные выделены в структуры); • изучите POU с аномально большим числом входов и выходов – возможно, их функциональность может быть распределена по отдельным POU или методам / действиям; • если различные переменные из области входов/выходов обрабатываются совместно в разных фрагментах кода – то выделите их в структуру; • проектируйте интерфейс (набор входов и выходов) POU таким образом, чтобы получать и передавать через него все нужные данные, избегая использования глобальных переменных.

Метрики, связанные со встроенной документацией

Комментарии в исходном коде

↑ **Высокий показатель**: 👍
↓ **Низкий показатель**: 👎

Причина: документирование кода с помощью комментариев поможет понять его функциональность разработчикам, которые будут заниматься его сопровождением.

Реализация в инструментах статического анализа:

- CODESYS Static Analysis: Percentage of comment
- SE – EcoStruxure MACA: Source Code Commented Ratio, Commented Variables Ratio, Number Of Multiline Comments, Number Of Header Comment Lines
- SE – EcoStruxure CE-V: percentage of comment, result of verification tool

Возможные меры по улучшению качества: уровень документированности кода можно увеличить, добавив дополнительные комментарии. Метрики могут подсказать, должны ли эти комментарии быть связаны с областью кода или с область объявления переменных.

Примечание: большое число комментариев не всегда является признаком качества. Важно писать содержательные комментарии, а не просто дословно описывать каждую строку кода. В крайней случае, можно создать отдельный POU, который будет содержать только комментарии и замети к проекту, и исключить его из компиляции.

Метрики, связанные с МЭК ООП-элементами

Глубина наследования

↑ **Высокий показатель**: 👎
↓ **Низкий показатель**: 👍

Причина: высокий уровень глубины наследования (EXTENDS) может затруднять понимание и сопровождение кода из-за проблем с отслеживаемостью зависимостей. Особенно длинные цепочки наследования (с глубиной > 10) становится крайне сложно редактировать, если ошибки возникают в «родительском» элементе и распространяются через наследование.

Реализация в инструментах статического анализа:

- CODESYS Static Analysis: DIT – Depth of Inheritance Tree, NOC – Number Of Children
- SE – EcoStruxure MACA: Extended By, Extends

⦸ **Потенциальные конфликты**:
↓ *уровень тестируемости и возможности повторного использования* может снизиться при уменьшении глубины наследования и потери преимуществ от наследования структур данных и функциональности POU.

Связность

↑ **Высокий показатель**: 👍
↓ **Низкий показатель**: 👎

Причина: низкий уровень связности указывает на то, что POU не соответствует принципу единой ответственности – т. е. пытается выполнять несколько отдельных задач. Сопровождение таких POU представляет сложность, потому что требуется понять все его функции вместо одной.

Реализация в инструментах статического анализа:

- CODESYS Static Analysis: LCOM – Lack of cohesion in methods (inverted logic as in a high lack of cohesion is disadvantageous)
- SIG Sigrid AQ: Component Cohesion

Сцепление между POU и окружающими объектами

↑ **Высокий показатель**: 👎
↓ **Низкий показатель**: 👍

Причина: высокая степень сцепления между POU и окружающими его объектами противоречит концепции модульности и, таким образом, затрудняет сопровождение.

Реализация в инструментах статического анализа:

- CODESYS Static Analysis: RFC – Response For Class, CBO – Coupling Between Objects
- SIG Sigrid Maintainability: Module coupling

Инкапсуляция данных и функционала в свойствах и методах

↑ **Высокий показатель**: 👍
↓ **Низкий показатель**: 👎

Причина: выделение монолитного кода в небольшие программные модули (например, в свойства и методы) облегчает сопровождение ПО.

Реализация в инструментах статического анализа:

- SE – EcoStruxure MACA: Number Of Properties, Number Of Methods

⦸ **Потенциальные конфликты**:
↓ *уровень эффективности* может снизиться в случае увеличения числа вызываемых POU (в т. ч. вложенных вызовов, когда один POU вызывает другой).

Возможные меры по улучшению качества:

- в случае использования в проекте наследования проясните отношения между POU (классами) в комментариях к коду (если это требуется);
- слишком высокая степень сцепления или отсутствие связности могут быть показателями того, что распределение функциональности между модулями сделано не лучшим образом. Следует проверить, можно ли провести рефакторинг соответствующих методов или перераспределить их функциональность;
- в конкретных ситуациях может быть выгоднее использовать композицию вместо наследования;
- принцип инверсии зависимостей: модули высокого уровня абстракции не

должны зависеть от низкоуровневых модулей. Оба типа модулей должны зависеть от интерфейсов. Это облегчает повторное использование, сопровождение и тестирование.

Метрики, связанные со сложностью ПО

Сложность и объём исходного кода

↑ **Высокий показатель**: 👎
↓ **Низкий показатель**: 👍

Причина: высокий уровень сложности ПО затрудняет отслеживание потенциальных перекрёстных эффектов при внесении изменений – например, в цикл (на текстовом языке) или цепь (на графическом языке) с высоким уровнем вложенности.
***Примечание*:** следует различать сложность прикладного приложения ПЛК и библиотечного кода; для используемого (но не редактируемого) библиотечного кода допускается высокий уровень сложности, поскольку она не затрагивает разработчика, использующего библиотечные POU (пока их интерфейсы остаются достаточно простыми).

Реализация в инструментах статического анализа:

- CODESYS Static Analysis: Halstead (D/HV/HL), Complexity (McCabe)
- SE – EcoStruxure MACA: Halstead Complexity, Cyclomatic Complexity
- SE – EcoStruxure CE-V: length, volume, difficulty, vg
- SIG Sigrid Maintainability: Unit size, unit complexity (McCabe)

Возможные меры по улучшению качества:

- проверьте, можно ли реализовать данный функционал более понятным образом, используя меньше циклов и вложенных цепей;
- проверьте, можно ли сократить число используемых операторов – например, путем рефакторинга слишком больших POU и перераспределения их функциональности.

Метрики, связанные с потоками данных

Поток данных от POU к окружающим его объектам

↑ **Высокий показатель**: 👎
↓ **Низкий показатель**: 👍

Причина: если окружающим объектам требуется получать много данных от POU, то это увеличивает риск перекрёстных эффектов при внесении изменений в код.

Реализация в инструментах статического анализа:

- SE – EcoStruxure MACA: Fan Out

Возможные меры по улучшению качества:

- проверьте, можно ли сократить число POU, вызываемых данным POU – например, путём перераспределения функциональности POU между другими POU, методами и действиями;
- для уменьшения «выходной нагрузки» POU создавайте дополнительные POU в качестве «прослойки» – это уменьшит в проекте число POU с низким уровнем связности и высоким уровнем сцепления;
- упрощайте структуры данных.

Метрики, связанные с индикаторами возможности повторного использования

Глубина вызовов

↑ **Высокий показатель**: 👎
↓ **Низкий показатель**: 👍

Причина: изменения на «нижнем уровне» длинной вложенной цепочки вызовов могут оказывать перекрёстные эффекты на POU, использующие их функциональность. Чем больше уровень вложенности вызовов, тем выше риск таких перекрёстных эффектов.

Реализация в инструментах статического анализа:

- SE – EcoStruxure CE-V: calldepthmin, calldepthmax

Дублирование кода

↑ **Высокий показатель**: 👎
↓ **Низкий показатель**: 👍

Причина: при написании кода методом «копипасты» (копирования и вставки) увеличиваются усилия, затрачиваемые на исправление ошибок и доработку функционала. Это связано с тем, что одно изменение потребуется вносить в нескольких фрагментах проекта.

Реализация в инструментах статического анализа:

- CODESYS Static Analysis: Duplication ratio
- SIG Sigrid Maintainability: Duplication

⦸ **Потенциальные конфликты**:
↓ увеличение количества зависимостей в проекте из-за появления фрагментов, в которых код вызывается «по ссылке (например, через вызов функции или механизмы объектно-ориентированного программирования)

Возможные меры по улучшению качества:

- если проект ПЛК или его фрагменты характеризуются высоким уровнем глубины вызовов, то следует проанализировать, можно ли его уменьшить путём объединения нескольких POU в один;
- применяйте механизмы, позволяющие добиться повторного использования кода – например, создавайте функции и используйте возможности ООП (наследование и т. д.).

***Отказ от ответственности*:** приведённые ниже фрагменты «качественного» и «некачественного» кода взяты из учебного, «идеализированного» примера управления водонагревателем, который не призван показать весь возможный спектр функционала ПО для ПЛК, применяемого в промышленности. Эти фрагменты не связаны друг с другом – то есть пример «качественного» кода не является оптимизированной версией примера «некачественного» кода. Так как исходный проект разработан в среде CODESYS, то для анализа кода использовался инструмент CODESYS Static Analysis.

Пример «некачественного» кода: в категориях «Метрики, связанные с размером кода» и «Метрики, связанные со сложностью ПО» можно заметить «выбросы» – высокие показатели сложности Холстеда и МакКейба (автора концепции [цикломатической сложности]), а также одно из самых высоких значений метрики «Количество операторов», что побуждает исследовать возможные пути оптимизации кода для повышения удобства его сопровождения (см. рис. 6).

Program unit	NOS	McCabe	D (Halstead)
FB_BasicPIDCtrl (FB)	53	14	26,1
FB_ErrorHandler (FB)	19	14	37,9
FC_ErrorCode (FUN)	8	8	2,8
FB_AutomaticMode (FB)	17	7	12,4
F_AlignValues (FUN)	5	5	12,3
FB_Motor.CyclicAction	6	4	5,6

Рис. 6. Таблица метрик из проекта CODESYS, отсортированная по убыванию цикломатической сложности

Анализ POU выявляет причину высокой цикломатической сложности: множество вложенных операторов FOR и IF, которые затрудняют сопровождение кода.

```
//monitor all register Device by the IBehaviour Modell
FOR uiLoop := 0 TO GVL_Const.MAX_Observer DO
    IF aritfObserver[uiLoop] <> 0 THEN
        _uiNumberOfDevices := _uiNumberOfDevices + 1;
        aritfObserver[uiLoop].GetModelState(    xCommit:= _stInfoState[uiLoop].xCommit,
                                    xDone=> _stInfoState[uiLoop].xDone,
                                    xBusy=> _stInfoState[uiLoop].xBusy,
                                    xError=> _stInfoState[uiLoop].xError,
                                    xAborted=> _stInfoState[uiLoop].xAborted,
                                    iErrorID=> _stInfoState[uiLoop].iErrorID,
                                    eState=> _stInfoState[uiLoop].eState);
        _stInfoState[uiLoop].strName := aritfObserver[uiLoop].GetName();
        IF NOT _stErrorState[uiLoop] AND _stInfoState[uiLoop].xError THEN
            _stErrorState[uiLoop] := _stInfoState[uiLoop].xError;
            //Build Error String
            FOR uiHelp := 10 TO 1 BY -1 DO
                IF uiHelp > 1 THEN
                    astrActive[uiHelp] := astrActive [uiHelp-1];
                ELSIF uiHelp = 1 THEN
                    astrActive[uiHelp] := CONCAT(CONCAT(CONCA
                END_IF
            END_FOR
        ELSIF  _stErrorState[uiLoop] AND NOT _stInfoState[uiL
            _stErrorState[uiLoop] := _stInfoState[uiLoop].xEr
            //Build Error String
            FOR uiHelp := 10 TO 1 BY -1 DO
                IF uiHelp > 1 THEN
                    astrActive[uiHelp] := astrActive [uiHelp-
                ELSIF uiHelp = 1 THEN
                    astrActive[uiHelp] := CONCAT(CONCAT(CONCA
                END_IF
            END_FOR
        END_IF
    END_IF
END_FOR
```

```
                ELSIF uiHelp = 1 THEN
                    astrActive[uiHelp]
                END_IF
            END_FOR
        END_IF
    END_IF
END_FOR
```

Рис. 7. Фрагмент кода, содержащий множество вложенных операторов управления

Пример «качественного кода» кода: перераспределение функциональности POU (в данном случае – путём выделения кода в методы) позволяет уменьшить их значение цикломатической сложности.

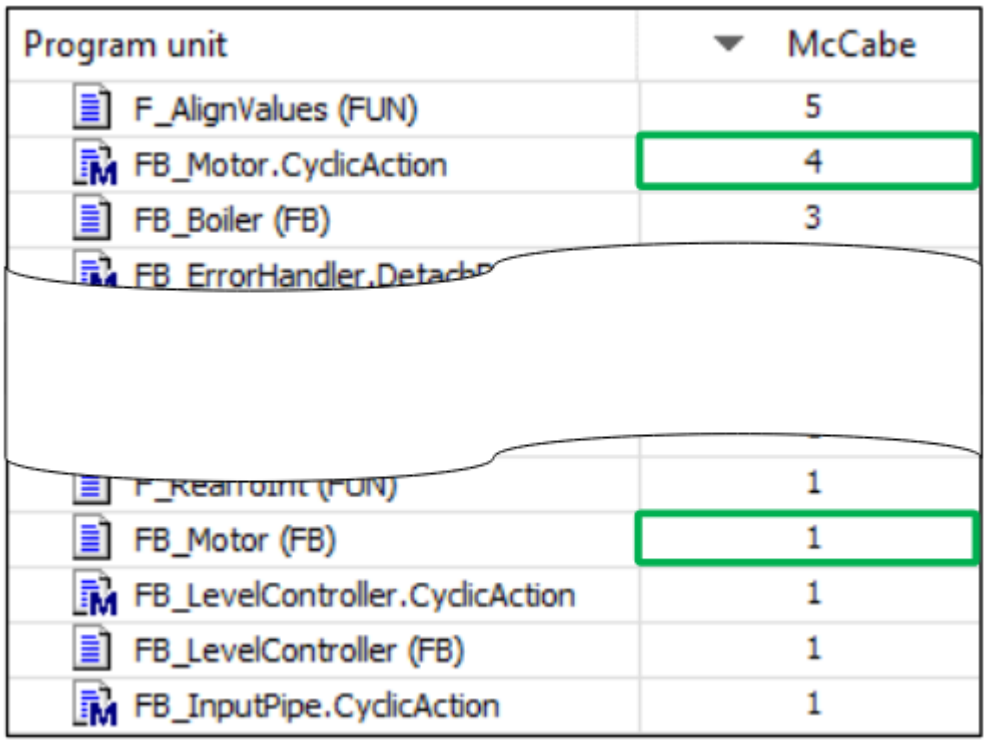


Рис. 8. Фрагмент таблицы метрик, демонстрирующий снижение цикломатической сложности после оптимизации POU

Анализ категории «Метрики, связанные с МЭК ООП-элементами» показывает, что низкая глубина наследования (Depth-Of-Inheritance; DIT) и разумное число дочерних объектов (Number-of-Children; NOC) облегчают сопровождение проекта.

Ссылки на другие руководства: Обучающие материалы PLCopen. Руководство по применению объектно-ориентированного подхода, версия 1.0 (PLCopen Guidelines for usage of Object-Oriented Programming, V1.0).

5.2. Метрики, связанные с возможностью повторного использования

Сценарии и шаги рабочего процесса (см. п. 4.2):

- **Сценарий 1**: непрерывная оценка качества POU в процессе разработки ПО.
  Мотивация: проверить, влияет ли конкретное изменение на возможность повторного использования ПО и наблюдать, как изменяется степень этой возможности с течением времени;
- **Сценарий 2**: оценка качества кода до и после ввода системы в эксплуатацию.
  Мотивация: проверить, повлияли ли изменения, произведённые в ПО на этапе пусконаладки, на возможность его повторного использования и требуется ли ревизия изменённых фрагментов кода. Если эти изменения вносятся методом «копипасты» (копирования и вставки), то их последствия можно сгладить путём стандартизации POU, рассчитанных на повторное использование;
- **Сценарий 3**: аудит объекта / установки после завершения проекта.
  Мотивация: сравнить количество повторно используемых POU данного проекта с другими проектами, чтобы выявить его преимущества и недостатки, а также определить возможные точки оптимизации в проектных решениях – связанные, например, с эффективностью процесса разработки.

Метрики, связанные с размером
Размер POU ↑ **Высокий показатель**: 👎 ↓ **Низкий показатель**: 👍 **Причина**: POU небольшого размера (малой гранулярности) в большинстве случаев являются более «гибкими», так как обычно реализуют очень конкретную и «ёмкую» функциональность и, таким образом, могут быть повторно использованы в различных проектах. С другой стороны, большой размер POU может быть индикатором «всеобъемлющей» функциональности, жёстко связанной с конкретной установкой/системой, что затрудняет повторное использование. **Реализация в инструментах статического анализа**: • CODESYS Static Analysis: Lines Of Code • SE – EcoStruxure MACA: Number of instructions • SE – EcoStruxure CE-V: NOS – Number Of Statement • SIG Sigrid Maintainability: Unit Size
Возможные меры по улучшению качества: • обязательно используйте принцип единой ответственности: выделяйте код, не связанный с основной задачей POU, в другие POU или, если он нужен для вспомогательной функциональности, в действия и методы вашего POU; • для структурирования кода POU выделяйте его отдельные, функционально связанные фрагменты в действия и методы; • избегайте дублирования фрагментов кода путём использования «наследуемых» функциональных блоков, а также структур и массивов.

Метрики, связанные с элементами, специфичными для конкретного языка
Количество шагов, переходов и ветвей в SFC ↑ **Высокий показатель**: 👎 ↓ **Низкий показатель**: 👍 **Причина**: см. «Метрики, связанные с размером». **Реализация в инструментах статического анализа**: • CODESYS Static Analysis: Number of SFC branches, Number of SFC steps • SE – EcoStruxure MACA: Number Of Transitions • SE – EcoStruxure CE-V: nbofbranches, g7height, g7width
Количество цепей в FBD ↑ **Высокий показатель**: 👎 ↓ **Низкий показатель**: 👍 **Причина**: см. «Метрики, связанные с размером». **Реализация в инструментах статического анализа**: • SE – EcoStruxure MACA: Number Of FBD Networks, Halstead Complexity for FBD
Возможные меры по улучшению качества: • если POU на SFC/FBD невозможно охватить взглядом (требуется увеличение масштаба или прокрутка окна редактора), то следует рассмотреть вариант распределения его функциональности по нескольким отдельным POU; • если отдельные цепи FBD сложны для понимания, то попробуйте распределить их функциональность по нескольким отдельным цепям; • используйте в FBD функциональные блоки с высоким уровнем абстракции, решающие конкретные задачи из предметной области (например, управление насосом, обработка аварий и т. д.), а не простые «базовые» блоки (триггеры, счётчики, таймеры); это приводит к уменьшению количества и размеров цепей.

Метрики, связанные переменными и интерфейсами POU
Использование глобальных переменных ↑ **Высокий показатель**: 👎 ↓ **Низкий показатель**: 👍 **Причина**: использование глобальных переменных приводит к нежелательным зависимостям POU от этих переменных, которые не могут быть разрешены через вызовы других POU, что затрудняет создание чётко определённых интерфейсов – а это является важным предварительным условием для повторного использования кода. **Реализация в инструментах статического анализа**: • CODESYS Static Analysis: Used different global variables • SE – EcoStruxure MACA: Number of GVL Usages • SE – EcoStruxure CE-V: extvarref

Прямой доступ к аппаратным ресурсам

↑ **Высокий показатель**: 👎
↓ **Низкий показатель**: 👍

Причина: прямой доступ к аппаратным ресурсам контроллера может быть показателем того, что POU адаптирован к конкретной платформе – а это может затруднить его использование в других приложениях.

Реализация в инструментах статического анализа:

- CODESYS Static Analysis: Number of direct address accesses(I/Os)

Возможные меры по улучшению качества:

- проверьте, является ли использование глобальных переменных действительно необходимым и могут ли быть нужные для POU данные получены им «напрямую», через входные переменные;
- если по каким-то причинам отказаться от использования глобальных переменных нельзя, то проверьте, что они сгруппированы логичным образом (например, функционально связанные переменные выделены в структуры);
- изучите POU с аномально большим числом входов и выходов – возможно, их функциональность может быть распределена по отдельным POU или методам / действиям;
- если различные переменные из области входов/выходов обрабатываются совместно в разных фрагментах кода – то выделите их в структуру;
- проектируйте интерфейс (набор входов и выходов) POU таким образом, чтобы получать и передавать через него все нужные данные, избегая использования глобальных переменных.

Метрики, связанные с МЭК ООП-элементами

Использование наследования

↑ **Высокий показатель**: 👍
↓ **Низкий показатель**: -

Причина: использование наследования для расширения функциональности существующих POU является признаком того, что они могут быть легко адаптированы к другим приложениям.

Реализация в инструментах статического анализа:

- CODESYS Static Analysis: DIT – Depth of Inheritance Tree, NOC – Number Of Children
- SE – EcoStruxure MACA: Extended By, Extends

⦸ **Потенциальные конфликты**:
↓ *уровень сопровождаемости* может снизиться, если связи между родительскими и дочерними POU сложно отследить.

Связность

↑ **Высокий показатель**: 👍
↓ **Низкий показатель**: 👎

Причина: высокий уровень связности в POU указывает на то, что каждый из них решает одну конкретную задачу, и это облегчает их повторное использование.

Реализация в инструментах статического анализа:

- CODESYS Static Analysis: LCOM – Lack of cohesion in methods (inverted logic as in a high lack of cohesion is disadvantageous)
- SIG Sigrid AQ: Component Cohesion

Сцепление между POU и окружающими объектами

↑ **Высокий показатель**: 👎
↓ **Низкий показатель**: 👍

Причина: высокая степень сцепления между POU и окружающими объектами указывает на зависимость от контекста, в котором он применяется, что затрудняет его повторное использование.

Реализация в инструментах статического анализа:

- CODESYS Static Analysis: RFC – Response For Class, CBO – Coupling Between Objects
- SIG Sigrid Maintainability: Module coupling

Реализация интерфейсов

↑ **Высокий показатель**: 👍
↓ **Низкий показатель**: -

Причина: чётко определённые интерфейсы являются одним из условий повторного использования POU. Использование объектов типа «интерфейс», описанных в объектном-ориентированном расширении стандарта МЭК 61131-3, считается ценным инструментом для адаптации функционала POU к различным приложениям с сохранением его интерфейса (набора методов и свойств), что облегчает повторное использование.

Реализация в инструментах статического анализа:

- SE – EcoStruxure MACA: Implemented By, Implements

⦸ **Потенциальные конфликты**:
↓ проведённые исследования [15] показали, что *уровень эффективности ПО* и производительность системы исполнения (рантайма) контроллера могут снизиться при вызове свойств и методов через объекты типа «интерфейс».

Инкапсуляция данных и функционала в свойствах и методах

↑ **Высокий показатель**: 👍
↓ **Низкий показатель**: 👎

Причина: выделение монолитного кода в небольшие программные модули повышает гибкость POU и возможность его повторного использования.

Реализация в инструментах статического анализа:

- SE – EcoStruxure MACA: Number Of Properties, Number Of Methods

⦸ **Потенциальные конфликты**:
↓ *уровень эффективности* может снизиться в случае увеличения числа вызываемых POU (в т. ч. вложенных вызовов, когда один POU вызывает другой).

Возможные меры по улучшению качества:

- в случае использования в проекте наследования проясните отношения между POU (классами) в комментариях к коду (если это требуется);
- слишком высокая степень сцепления или отсутствие связности могут быть показателями того, что распределение функциональности между модулями сделано не лучшим образом. Следует проверить, можно ли провести рефакторинг соответствующих методов или перераспределить их функциональность;
- минимальная глубина наследования способствует повторному использованию POU. Ограничьте глубину наследования разумно выбранным значением, чтобы избежать сложного и непредсказуемого поведения; здесь необходим баланс;
- принцип инверсии зависимостей: модули высокого уровня абстракции не должны зависеть от низкоуровневых модулей. Оба типа модулей должны зависеть от интерфейсов. Это облегчает повторное использование, сопровождение и тестирование.

Метрики, связанные со сложностью ПО

Сложность и объём исходного кода

↑ **Высокий показатель**: 👎
↓ **Низкий показатель**: 👍

Причина: высокий уровень сложности POU может быть признаком «монолитности» приложения – то есть основная часть его функционала сосредоточена в этом POU, что затрудняет повторное использование.

Реализация в инструментах статического анализа:

- CODESYS Static Analysis: Halstead (D/HV/HL), Complexity (McCabe)
- SE – EcoStruxure MACA: Halstead Complexity, Cyclomatic Complexity
- SE – EcoStruxure CE-V: length, volume, difficulty, vg
- SIG Sigrid Maintainability: Unit size, unit complexity (McCabe)

Возможные меры по улучшению качества:

- проверьте, можно ли реализовать данный функционал более понятным образом, используя меньше циклов и вложенных цепей;
- проверьте, можно ли сократить число используемых операторов – например, путем рефакторинга слишком больших POU и перераспределения их функциональности;
- разбивайте код на небольшие функции, чтобы уменьшить его сложность и повысить возможность повторного использования;
- инкапсуляция и абстракция могут помочь повысить возможность повторного использования.

Метрики, связанные с потоками данных

«Прямой» обмен данными через вызовы POU

↑ **Высокий показатель**: 👍
↓ **Низкий показатель**: 👎

Причина: «прямой» обмен данными через вызовы POU в сочетании со сведённым к нулю использованием глобальных переменных часто является индикатором продуманных интерфейсов POU, облегчающих их повторное использование.

Реализация в инструментах статического анализа:

- CODESYS Static Analysis: Number of calls
- SE – EcoStruxure MACA: Call In, Call Out
- SE – EcoStruxure CE-V: calledcount, callproc

⦸ **Потенциальные конфликты**:
↓ проведённые исследования [15] показали, что *уровень эффективности ПО* может снизиться при многочисленных вызовах POU.

Поток данных от POU к окружающим его объектам

↑ **Высокий показатель**: 👎
↓ **Низкий показатель**: 👍

Причина: существенный поток данных между POU и окружающими объектами делает эти объекты зависимыми от POU, что затрудняет повторное использование кода.

Реализация в инструментах статического анализа:

- SE – EcoStruxure MACA: Fan Out

Возможные меры по улучшению качества:

- проверьте, можно ли сократить число POU, вызываемых данным POU – например, путём перераспределения функциональности данного POU между другими POU, методами и действиями;
- для уменьшения «выходной нагрузки» POU создавайте дополнительные POU в качестве «прослойки» – это уменьшит в проекте число POU с низким уровнем связности и высоким уровнем сцепления;
- упрощайте структуры данных.

Метрики, связанные с индикаторами возможности повторного использования

Глубина вызовов

↑ **Высокий показатель**: 👎
↓ **Низкий показатель**: 👍

Причина: высокий уровень вложенности вызовов может указывать на то, что ПО не является монолитным, а имеет иерархическую структуру, т. е. его функциональность инкапсулирована и подготовлена для повторного использования.

Реализация в инструментах статического анализа:

- SE – EcoStruxure CE-V: calldepthmin, calldepthmax

Дублирование кода

↑ **Высокий показатель**: 👎
↓ **Низкий показатель**: 👍

Причина: при написании кода методом «копипасты» (копирования и вставки) увеличиваются усилия, затрачиваемые на исправление ошибок и доработку функционала. Это связано с тем, что одно изменение потребуется вносить в нескольких фрагментах проекта.

Реализация в инструментах статического анализа:

- CODESYS Static Analysis: Duplication ratio
- SIG Sigrid Maintainability: Duplication

⦸ **Потенциальные конфликты**:
↓ увеличение количества зависимостей в проекте из-за появления фрагментов, в которых повторно вызывается один и тот же код (например, через вызов функции или механизмы объектно-ориентированного программирования)

Возможные меры по улучшению качества:

- если проект ПЛК или его фрагменты характеризуются высоким уровнем глубины вызовов, то следует проанализировать, можно ли его уменьшить путём объединения нескольких POU в один;
- применяйте механизмы, позволяющие добиться повторного использования кода – например, создавайте функции и используйте возможности ООП (наследование и т. д.).

***Отказ от ответственности*:** приведённые ниже фрагменты «качественного» и «некачественного» кода взяты из учебного, «идеализированного» примера управления водонагревателем, который не призван показать весь возможный спектр функционала ПО для ПЛК, применяемого в промышленности. Эти фрагменты не связаны друг с другом – то есть пример «качественного» кода не является оптимизированной версией примера «некачественного» кода. Так как исходный проект разработан в среде CODESYS, то для анализа кода использовался инструмент CODESYS Static Analysis.

Пример «некачественного» кода: с первого взгляда неясно, что в методе, приведённом на скриншоте, используется 5 глобальных переменных (и это было бы неочевидно даже при взгляде на его область объявления). Эта скрытая зависимость от списка глобальных переменных (GVL) может вызвать проблемы с повторным использованием метода **FB_Motor.CyclicAction**, затрудняя создание чётко определенного интерфейса блока **FB_Motor**, который мог бы использоваться другими POU.

```
1   IF (enMode=0) THEN
2       iOutputMotor := F_RealToInt(rValveState,0.0,
3                                   (rMaxFeedwaterPressure * rMaxValveState),
4                                   0, 10000);
5   ELSE
6       IF ((counter MOD 25) = 0) THEN
7           iMotor:= REAL_TO_INT((rValveState - rValveStateBefore)*1000.0);
8           rValveStateBefore :=rValveState;
9       END_IF;
10      iOutputMotor  := iMotor;
11  END_IF;
```

legend: global variable


```
1   IF (enMode=0) THEN
2       iOutputMotor := F_RealToInt(rValveState,0.0,
3                                   (rMaxFeedwaterPressure * rMaxValveState),
4                                   0, 10000);
5   ELSE
6       IF ((counter MOD 25) = 0) THEN
7           iMotor:= REAL_TO_INT((rValveState - rValveStateBefore)*1000.0);
8           rValveStateBefore :=rValveState;
9       END_IF;
10      iOutputMotor  := iMotor;
11  END_IF;
```

legend: global variable

Рис. 10. Фрагмент кода с использованием глобальных переменных

Program unit	Globals
FC_ErrorCode (FUN)	7
FB_Motor.CyclicAction	5
P_Initialisation (PRG)	4
FB_InputParameter (FB)	4
FB_AutomaticMode (FB)	3

Рис. 11. Таблица метрик из проекта CODESYS, отсортированная по убыванию числа глобальных переменных

Пример «качественного» кода: благодаря «прямой» передаче данных в POU через входные переменные при его вызове (а не через глобальные переменные) – зависимости между POU чётко определены и наглядно видны. Как уже упоминалось ранее – такой чётко определённый интерфейс POU является ключевым требованием для обеспечения возможности его повторного использования. Он может значительно сократить число скрытых зависимостей – и, соответственно, уменьшить число ошибок при повторном использовании этого конкретного POU.

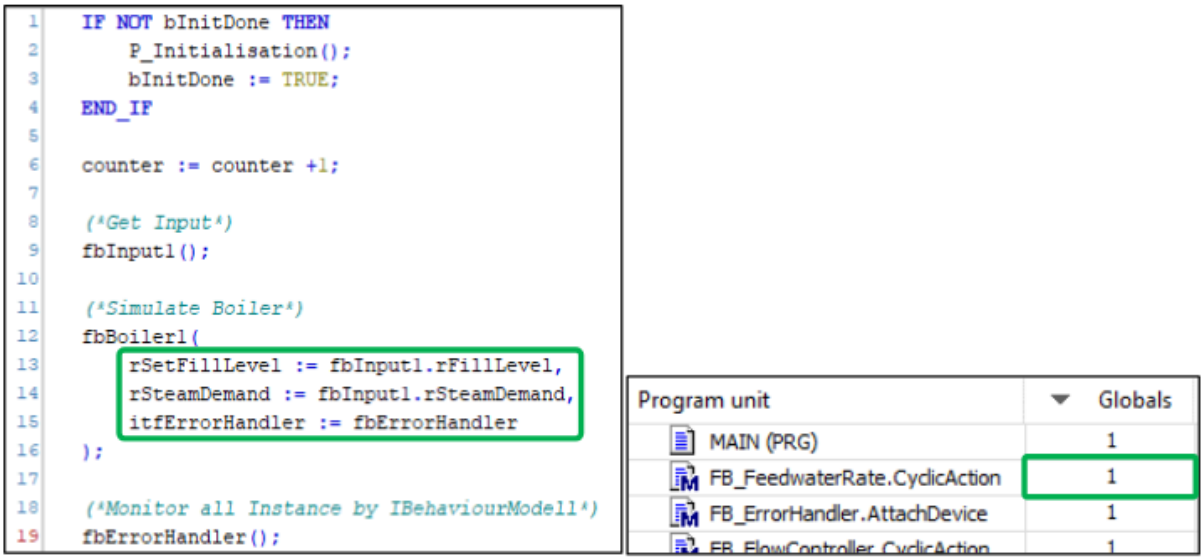


Рис. 12. Фрагмент кода POU и соответствующее ему значение метрики, подтверждающее отсутствие глобальных переменных

Ссылки на другие руководства: Обучающие материалы PLCopen. Руководство по применению объектно-ориентированного подхода, версия 1.0 (PLCopen Guidelines for usage of Object-Oriented Programming, V1.0).

5.3. Метрики, связанные с тестируемостью

Сценарии и шаги рабочего процесса (см. п. 4.2):

- **Сценарий 1**: непрерывная оценка качества POU в процессе разработки ПО.
  Мотивация: проверить, влияет ли конкретное изменение на тестируемость ПО и наблюдать, как изменяется степень тестируемости с течением времени;
- **Сценарий 3**: аудит объекта / установки после завершения проекта.
  Мотивация: сравнить степень тестируемости данного проекта с другими проектами, чтобы выявить его преимущества и недостатки, а также определить возможные точки оптимизации в проектных решениях – связанные, например, с эффективностью процесса тестирования.

Метрики, связанные с размером
Размер POU ↑ **Высокий показатель**: 👎 ↓ **Низкий показатель**: 👍 **Причина**: POU большого размера часто включают в себя сложные потоки управления, тестирование которых требует значительных ресурсов. **Реализация в инструментах статического анализа**: • CODESYS Static Analysis: Lines Of Code • SE – EcoStruxure MACA: Number of instructions • SE – EcoStruxure CE-V: NOS – Number Of Statement • SIG Sigrid Maintainability: Unit Size
Количество действий ↑ **Высокий показатель**: 👍 ↓ **Низкий показатель**: 👎 **Причина**: наличие значительного количества действий может указывать на то, что функциональность приложения инкапсулирована в небольшие программные модули, которые легко можно протестировать. **Реализация в инструментах статического анализа**: • SE – EcoStruxure MACA: Number Of Actions • SE – EcoStruxure CE-V: plcobjecttype counter • SIG Sigrid Maintainability: Unit Size ⦸ **Потенциальные конфликты**: ↓ *уровень эффективности* может снизиться из-за увеличения количества вызовов модулей.

Возможные меры по улучшению качества:

- обязательно используйте принцип единой ответственности: выделяйте код, не связанный с основной задачей POU, в другие POU или, если он нужен для вспомогательной функциональности, в действия и методы вашего POU;
- для структурирования кода POU выделяйте его отдельные, функционально связанные фрагменты в действия и методы;
- избегайте дублирования фрагментов кода путём использования «наследуемых» функциональных блоков, а также структур и массивов.

Метрики, связанные с элементами, специфичными для конкретного языка

Количество шагов, переходов и ветвей в SFC

↑ **Высокий показатель**: 👎
↓ **Низкий показатель**: 👍

Причина: см. «Метрики, связанные с размером».

Реализация в инструментах статического анализа:

- CODESYS Static Analysis: Number of SFC branches, Number of SFC steps
- SE – EcoStruxure MACA: Number Of Transitions
- SE – EcoStruxure CE-V: nbofbranches, g7height, g7width

Количество цепей в FBD

↑ **Высокий показатель**: 👎
↓ **Низкий показатель**: 👍

Причина: см. «Метрики, связанные с размером».

Реализация в инструментах статического анализа:

- SE – EcoStruxure MACA: Number Of FBD Networks, Halstead Complexity for FBD

Возможные меры по улучшению качества:

- если POU на SFC/FBD невозможно охватить взглядом (требуется увеличение масштаба или прокрутка окна редактора), то следует рассмотреть вариант распределения его функциональности по нескольким отдельным POU;
- если отдельные цепи FBD сложны для понимания, то попробуйте распределить их функциональность по нескольким отдельным цепям;
- используйте в FBD функциональные блоки с высоким уровнем абстракции, решающие конкретные задачи из предметной области (например, управление насосом, обработка аварий и т. д.), а не простые «базовые» блоки (триггеры, счётчики, таймеры); это приводит к уменьшению количества и размеров цепей.

Метрики, связанные переменными и интерфейсами POU
Прямой доступ к аппаратным ресурсам ↑ **Высокий показатель**: 👎 ↓ **Низкий показатель**: 👍 **Причина**: если POU требует прямого доступа к аппаратным ресурсам контроллера для выполнения своих функций (например, считывания данных с датчиков или отправки команд исполнительным механизмам) – это приводит к дополнительным зависимостям, которые необходимо учитывать во время тестирования, и увеличивает усилия, которые требуются для организации эмуляции работы оборудования или использования реального оборудования. **Реализация в инструментах статического анализа**: • CODESYS Static Analysis: Number of direct address accesses(I/Os)
Количество переменных в интерфейсе POU ↑ **Высокий показатель**: 👎 ↓ **Низкий показатель**: 👍 **Причина**: большое количество переменных интерфейса (особенно входов) POU приводит к его сильной зависимости от других объектов (формирующих данные, передаваемые в этот POU, и использующих данные, получаемые от него), в результате чего для проверки корректности работы POU требуется сформировать множество различных наборов входных данных. **Реализация в инструментах статического анализа**: • CODESYS Static Analysis: Number of input/output variables • SE – EcoStruxure CE-V: inputcount, outputcount, nbofparam • SIG Sigrid Maintainability: Unit Interfacing
Возможные меры по улучшению качества: • проверьте, является ли использование глобальных переменных действительно необходимым и могут ли быть нужные для POU данные получены им «напрямую», через входные переменные; • если различные переменные из области входов/выходов обрабатываются совместно в разных фрагментах кода – то выделите их в структуру.

Метрики, связанные с МЭК ООП-элементами

Использование наследования

↑ **Высокий показатель**: 👍
↓ **Низкий показатель**: -

Причина: наследование позволяет повторно использовать функциональность базового класса в различных приложениях, и это сокращает затраты на тестирование, так как тестирование функциональности базового класса достаточно провести один раз, а не для всех наследуемых от него ФБ.

Реализация в инструментах статического анализа:

- CODESYS Static Analysis: DIT – Depth of Inheritance Tree, NOC – Number Of Children
- SE – EcoStruxure MACA: Extended By, Extends

🚫 **Потенциальные конфликты**:
↓ *уровень сопровождаемости* может снизиться, если связи между родительскими и дочерними POU сложно отследить.

Связность

↑ **Высокий показатель**: 👍
↓ **Низкий показатель**: 👎

Причина: высокий уровень связности обычно является признаком того, что POU выполняет одну конкретную задачу (а не несколько различных). Это упрощает тестирование, потому что для проверки корректности работы такого POU часто достаточного всего одного тестового сценария.

Реализация в инструментах статического анализа:

- CODESYS Static Analysis: LCOM – Lack of cohesion in methods (inverted logic as in a high lack of cohesion is disadvantageous)
- SIG Sigrid AQ: Component Cohesion

Сцепление между POU и окружающими объектами

↑ **Высокий показатель**: 👎
↓ **Низкий показатель**: 👍

Причина: в отличие от высокого уровня связности, высокая степень сцепления POU с окружающими объектами приводит к большому количеству зависимостей, которые нужно учитывать при разработке тестовых сценариев (см. выше «Количество переменных в интерфейсе POU»).

Реализация в инструментах статического анализа:

- CODESYS Static Analysis: RFC – Response For Class, CBO – Coupling Between Objects
- SIG Sigrid Maintainability: Module coupling

Реализация интерфейсов

↑ **Высокий показатель**: 👍
↓ **Низкий показатель**: -

Причина: при разумном использовании интерфейсы обеспечивают распределение функциональности ПО между отдельными модулями, что облегчает тестирование.

Реализация в инструментах статического анализа:

- SE – EcoStruxure MACA: Implemented By, Implements

🚫 **Потенциальные конфликты**:
↓ проведённые исследования [15] показали, что *уровень эффективности ПО* и производительность системы исполнения (рантайма) контроллера могут снизиться при вызове свойств и методов через объекты типа «интерфейс».

Инкапсуляция данных и функционала в свойствах и методах

↑ **Высокий показатель**: 👍
↓ **Низкий показатель**: 👎

Причина: выделение монолитного кода в небольшие программные модули сокращает число тестовых сценариев, необходимых для проверки конкретного POU.

Реализация в инструментах статического анализа:

- SE – EcoStruxure MACA: Number Of Properties, Number Of Methods

🚫 **Потенциальные конфликты**:
↓ *уровень эффективности* может снизиться в случае увеличения числа вызываемых POU (в т. ч. вложенных вызовов, когда один POU вызывает другой).

Возможные меры по улучшению качества:

- в случае использования в проекте наследования проясните отношения между POU (классами) в комментариях к коду (если это требуется);
- слишком высокая степень сцепления или отсутствие связности могут быть показателями того, что распределение функциональности между модулями сделано не лучшим образом. Следует проверить, можно ли провести рефакторинг соответствующих методов или перераспределить их функциональность;
- принцип инверсии зависимостей: модули высокого уровня абстракции не должны зависеть от низкоуровневых модулей. Оба типа модулей должны зависеть от интерфейсов. Это облегчает повторное использование, сопровождение и тестирование.

Метрики, связанные со сложностью ПО

Сложность и объём исходного кода

↑ **Высокий показатель**: 👎
↓ **Низкий показатель**: 👍

Причина: высокий уровень сложности POU может быть признаком значительного количества потоков управления (в т. ч. вложенных), что требует написания большого числа комплексных тестовых сценариев для проверки всех возможных случаев.

Реализация в инструментах статического анализа:

- CODESYS Static Analysis: Halstead (D/HV/HL), Complexity (McCabe)
- SE – EcoStruxure MACA: Halstead Complexity, Cyclomatic Complexity
- SE – EcoStruxure CE-V: length, volume, difficulty, vg
- SIG Sigrid Maintainability: Unit size, unit complexity (McCabe)

Возможные меры по улучшению качества:

- проверьте, можно ли реализовать данный функционал более понятным образом, используя меньше циклов и вложенных цепей;
- проверьте, можно ли сократить число используемых операторов – например, путем рефакторинга слишком больших POU и перераспределения их функциональности.

Метрики, связанные с потоками данных

«Прямой» обмен данными через вызовы POU

↑ **Высокий показатель**: 👍
↓ **Низкий показатель**: 👎

Причина: «прямой» обмен данными через вызовы POU в сочетании со сведённым к нулю использованием глобальных переменных облегчает тестирование, потому что снижает риск наличия неявных или скрытых зависимостей, которые могут не учитываться при проведении тестов.

Реализация в инструментах статического анализа:

- SE – EcoStruxure MACA: Call Out
- SE – EcoStruxure CE-V: callproc

⦸ **Потенциальные конфликты**:
↓*Уровень эффективности ПО* и производительность системы исполнения (рантайма) контроллера могут снизиться при многочисленных вызовах POU.

Число операций чтений / записи переменных

↑ **Высокий показатель**:
↓ **Низкий показатель**:

Причина: значительное количество чтения и записи переменных указывает на существенное число зависимостей, которые необходимо учитывать при тестировании.

Реализация в инструментах статического анализа:

- SE – EcoStruxure MACA: Number Of Writes, Number Of Reads
- SE – EcoStruxure CE-V: callproc

Поток данных от POU к окружающим его объектам

↑ **Высокий показатель**:
↓ **Низкий показатель**:

Причина: существенный поток данных между POU и окружающими объектами указывает на значительное число зависимостей, которые необходимо учитывать при тестировании.

Реализация в инструментах статического анализа:

- SE – EcoStruxure MACA: Fan Out

Возможные меры по улучшению качества:

- проверьте, можно ли сократить число POU, вызываемых данным POU – например, путём перераспределения функциональности POU между другими POU, методами и действиями;
- для уменьшения «выходной нагрузки» POU создавайте дополнительные POU в качестве «прослойки» – это уменьшит в проекте число POU с низким уровнем связности и высоким уровнем сцепления;
- упрощайте структуры данных.

Метрики, связанные с индикаторами возможности повторного использования

Глубина вызовов

↑ **Высокий показатель**:
↓ **Низкий показатель**:

Причина: высокий уровень вложенности вызовов может указывать на то, что тестируемый ПО является частью длинной цепочки вызовов, т. е. внесение в него изменений может привести к перекрёстным эффектам.

Реализация в инструментах статического анализа:

- SE – EcoStruxure CE-V: calldepthmin, calldepthmax

Дублирование кода

↑ **Высокий показатель**: 👎
↓ **Низкий показатель**: 👍

Причина: при написании кода методом «копипасты» (копирования и вставки) увеличиваются усилия, затрачиваемые на тестирование и внесение изменений. Это связано с тем, что один и тот же тест потребуется проводить для нескольких фрагментов проекта (такая же ситуация и с внесением изменений).

Реализация в инструментах статического анализа:

- CODESYS Static Analysis: Duplication ratio
- SIG Sigrid Maintainability: Duplication

⃠ **Потенциальные конфликты**:
↓ увеличение количества зависимостей в проекте из-за появления фрагментов, в которых повторно вызывается один и тот же код (например, через вызов функции или механизмы объектно-ориентированного программирования)

Возможные меры по улучшению качества:

- если проект ПЛК или его фрагменты характеризуются высоким уровнем глубины вызовов, то следует проанализировать, можно ли его уменьшить путём объединения нескольких POU в один;
- применяйте механизмы, позволяющие добиться повторного использования кода – например, создавайте функции и используйте возможности ООП (наследование и т. д.).

***Отказ от ответственности*:** приведённые ниже фрагменты «качественного» и «некачественного» кода взяты из учебного, «идеализированного» примера управления водонагревателем, который не призван показать весь возможный спектр функционала ПО для ПЛК, применяемого в промышленности. Эти фрагменты не связаны друг с другом – то есть пример «качественного» кода не является оптимизированной версией примера «некачественного» кода. Так как исходный проект разработан в среде CODESYS, то для анализа кода использовался инструмент CODESYS Static Analysis.

Пример «некачественного» кода: большое количество входных переменных (8) ФБ ПИД-регулятора **FB_BasicPIDCtrl** требует значительных усилий для разработки тестов, охватывающих различные комбинации их значений. Эта ситуация усугубляется высоким уровнем вложенности управляющих конструкций (на что указывает значение метрики цикломатической сложности МакКейба).

Program unit	Inputs	McCabe
FB_BasicPIDCtrl (FB)	8	14
FB_ErrorHandler (FB)	0	14
FC_ErrorCode (FUN)	1	8
FB_AutomaticMode (FB)	0	7
F_AlignValues (FUN)	3	5
FB_Motor.CyclicAction	1	4
FB_DRAND (FB)	1	3

Рис. 13. Таблица метрик из проекта CODESYS, отсортированная по убыванию цикломатической сложности МакКейба

Пример «качественного» кода: минимальная (или вообще нулевая) зависимость POU от окружающих его объектов облегчает и ускоряет тестирование. В данном конкретном случае этот эффект для ФБ **FB_DeviceBasic** был достигнут за счёт использования интерфейсов, что позволило создать небольшие отдельные тесты для каждого POU, реализующего данный интерфейс.

Program unit	Inputs	McCabe
FB_DeviceBasic (FB)	0	1
FB_ErrorHandler (FB)	0	14
P_Initialisation (PRG)	0	1
FB_InputParameter (FB)	0	3
MAIN (PRG)	0	2

```
1  {attribute 'reflection'}
2  FUNCTION_BLOCK FB_DeviceBasic IMPLEMENTS IBasicElement, CBML.IActionProvider, CBML.IActionController
3
4  VAR_OUTPUT
```

Рис. 14. Таблица метрик из CODESYS и с фрагмент кода ФБ **FB_DeviceBasic**, демонстрирующий использование интерфейсов

5.4. Метрики, связанные с оценкой эффективности и производительности

Сценарии и шаги рабочего процесса (см. п. 4.2):

- **Сценарий 1**: непрерывная оценка качества POU в процессе разработки ПО.
  Мотивация: проверить, влияет ли конкретное изменение на эффективность ПО и наблюдать, как изменяется степень эффективности с течением времени;
- **Сценарий 2**: оценка качества кода до и после ввода системы в эксплуатацию.
  Мотивация: проверить, повлияли ли изменения, произведённые в ПО на этапе пусконаладки, на его эффективность и требуется ли ревизия изменённых фрагментов кода;
- **Сценарий 3**: аудит объекта / установки после завершения проекта.
  Мотивация: сравнить эффективность ПО данного проекта с другими проектами, чтобы выявить его преимущества и недостатки, а также определить возможные точки оптимизации в проектных решениях – связанные, например, с производительностью используемых устройств системы автоматизации.

Метрики, связанные с размером
Размер POU ↑ **Высокий показатель**: 👎 ↓ **Низкий показатель**: 👍 **Причина**: проведённые исследования показали, что если POU имеет большой размер (т. е. занимает много памяти), то присвоение переменных его входам при вызове негативно влияет на производительность ПО. **Реализация в инструментах статического анализа**: • CODESYS Static Analysis: Code size, Stack size • SE – EcoStruxure MACA: Memory Size (Data) SE – EcoStruxure CE-V: Memory size
Количество действий ↑ **Высокий показатель**: 👎 ↓ **Низкий показатель**: 👍 **Причина**: большое количество действий обычно приводит к значительному числу их вызовов, что может оказать негативное влияние на производительность POU, который вызывает эти действия. **Реализация в инструментах статического анализа**: • SE – EcoStruxure MACA: Number Of Actions • SE – EcoStruxure CE-V: plcobjecttype counter • SIG Sigrid Maintainability: Unit Size ⦸ **Потенциальные конфликты**: ↓ *сопровождаемость* и *тестируемость* могут ухудшиться, потому что уменьшение числа действий снизит степень инкапсуляции функциональности ПО.

Возможные меры по улучшению качества:

- для структурирования кода POU выделяйте его отдельные, функционально связанные фрагменты в действия и методы. Однако для критичных к времени выполнения фрагментов ПО избегайте неоправданно длинных цепочек вызовов.

Метрики, связанные переменными и интерфейсами POU

Использование глобальных переменных

↑ **Высокий показатель**: 👎
↓ **Низкий показатель**: 👍

Причина: использование глобальных переменных может снизить эффективность ПО в случае их значительного количества и интенсивного доступа к ним.

Реализация в инструментах статического анализа:

- CODESYS Static Analysis: Used different global variables
- SE – EcoStruxure MACA: Number of GVL Usages
- SE – EcoStruxure CE-V: extvarref

Количество переменных в интерфейсе POU

↑ **Высокий показатель**: 👎
↓ **Низкий показатель**: 👍

Причина: большое количество локальных переменных POU может указывать на интенсивную передачу данных внутри него, что снижает эффективность его работы.

Реализация в инструментах статического анализа:

- CODESYS Static Analysis: Number of input/output variables
- SE – EcoStruxure MACA: Number of Variables

Возможные меры по улучшению качества:

- проверьте, является ли использование глобальных переменных действительно необходимым и могут ли быть нужные для POU данные получены им «напрямую», через входные переменные;
- если по каким-то причинам отказаться от использования глобальных переменных нельзя, то проверьте, что они сгруппированы логичным образом (например, функционально связанные переменные выделены в структуры);
- изучите POU с аномально большим числом входов и выходов – возможно, их функциональность может быть распределена по отдельным POU или методам / действиям;
- если различные переменные из области входов/выходов обрабатываются совместно в разных фрагментах кода – то выделите их в структуру;
- проектируйте интерфейс (набор входов и выходов) POU таким образом, чтобы получать и передавать через него все нужные данные, избегая использования глобальных переменных.

Метрики, связанные с МЭК ООП-элементами
Реализация интерфейсов ↑ **Высокий показатель**: 👎 ↓ **Низкий показатель**: 👍 **Причина:** анализ производительности промышленного ПО показывает [15], что его уровень эффективности может снизиться при вызове свойств и методов через объекты типа «интерфейс». **Реализация в инструментах статического анализа**: • SE – EcoStruxure MACA: Implements ⦸ **Потенциальные конфликты**: ↓ *возможность повторного использования* и *тестируемость* снижаются при отказе от использования интерфейсов и наследования.
Инкапсуляция данных и функционала в свойствах и методах ↑ **Высокий показатель**: 👎 ↓ **Низкий показатель**: 👍 **Причина:** выделение монолитного кода в небольшие программные модули выгодно с точки зрения модульности ПО, но может привести к снижению его производительности за счёт увеличения количества вызовов. **Реализация в инструментах статического анализа**: • SE – EcoStruxure MACA: Number Of Properties, Number Of Methods ⦸ **Потенциальные конфликты**: ↓ *сопровождаемость*, *возможность повторного использования* и *тестируемость* снижаются в случае отсутствия инкапсуляции функциональности ПО в небольшие программные модули.
Возможные меры по улучшению качества: • слишком высокая степень сцепления или отсутствие связности могут быть показателями того, что распределение функциональности между модулями сделано не лучшим образом. Следует проверить, можно ли провести рефакторинг соответствующих методов или перераспределить их функциональность.

Метрики, связанные с потоками данных
Передача данных между POU путём их вызовов ↑ **Высокий показатель**: 👎 ↓ **Низкий показатель**: 👍 **Причина**: большое количество вызовов POU может снизить производительность приложения контроллера – т. е. увеличить время его выполнения. **Реализация в инструментах статического анализа**: • SE – EcoStruxure MACA: Call Out • SE – EcoStruxure CE-V: callproc ⦸ **Потенциальные конфликты**: ↓ *возможность повторного использования* и *тестируемость* снижаются в случае отсутствия инкапсуляции функциональности ПО в небольшие программные модули и использовании вместо этого «монолитных» POU.
Поток данных от POU к окружающим его объектам ↑ **Высокий показатель**: 👎 ↓ **Низкий показатель**: 👍 **Причина**: если окружающим объектам требуется получать много данных от POU, то это увеличивает риск перекрёстных эффектов при внесении изменений в код. **Реализация в инструментах статического анализа**: • SE – EcoStruxure MACA: Fan Out
Возможные меры по улучшению качества: • проверьте, можно ли сократить число исходящих из POU потоков данных – например, путём перераспределения функциональности POU между другими POU, методами и действиями; • упрощайте структуры данных.

***Отказ от ответственности*:** приведённые ниже фрагменты «качественного» и «некачественного» кода взяты из учебного, «идеализированного» примера управления водонагревателем, который не призван показать весь возможный спектр функционала ПО для ПЛК, применяемого в промышленности. Эти фрагменты не связаны друг с другом – то есть пример «качественного» кода не является оптимизированной версией примера «некачественного» кода. Так как исходный проект разработан в среде CODESYS, то для анализа кода использовался инструмент CODESYS Static Analysis.

Примеры «некачественного» и «качественного» кода: невозможно однозначно определить пороговые значения метрик POU, при превышении которых производительность приложения контроллера существенно снизится. Однако интенсивная передача данных внутри POU, вызванная большим количеством локальных переменных, может оказать негативное влияние на время выполнения приложения; например, ФБ **FB_AutomaticMode** существенно меньше влияет на производительность, чем ФБ **FB_LevelController**.

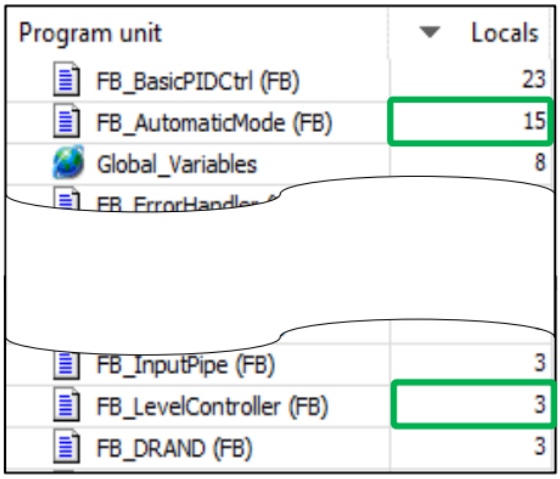


Рис. 15. Таблица метрик из проекта CODESYS, отсортированная по убыванию числа локальных переменных POU

5.5. Метрики, связанные с надёжностью

Сценарии и шаги рабочего процесса (см. п. 4.2):

- **Сценарий 1**: непрерывная оценка качества POU в процессе разработки ПО.
  Мотивация: проверить, влияет ли конкретное изменение на надёжность ПО и наблюдать, как изменяется степень надёжности с течением времени;
- **Сценарий 2**: оценка качества кода до и после ввода системы в эксплуатацию.
  Мотивация: проверить, повлияли ли изменения, произведённые в ПО на этапе пусконаладки, на его надёжность и требуется ли ревизия изменённых фрагментов кода;
- **Сценарий 3**: аудит объекта / установки после завершения проекта.
  Мотивация: сравнить надёжность ПО данного проекта с другими проектами, чтобы выявить его преимущества и недостатки, а также определить возможные точки оптимизации в проектных решениях – связанные, например, с планированием сценариев тестирования для глобальных/критических изменений, которые могут повлиять на надёжность функционирования ПО.

Метрики, связанные с элементами, специфичными для конкретного языка
Количество переходов в SFC ↑ **Высокий показатель**: 👎 ↓ **Низкий показатель**: 👍 **Причина**: большое количество ветвлений в SFC-схеме может сделать код запутанным и затруднить внесение изменений; появляется вероятность того, что не будет учтено взаимное влияние элементов схемы друг на друга, и это приведёт к снижению надёжности ПО. **Реализация в инструментах статического анализа**: • CODESYS Static Analysis: Number of SFC branches, Number of SFC steps • SE – EcoStruxure MACA: Number Of Transitions • SE – EcoStruxure CE-V: nbofbranches, g7height, g7width
Количество цепей в FBD ↑ **Высокий показатель**: 👎 ↓ **Низкий показатель**: 👍 **Причина**: большое количество цепей (networks) в POU, написанном на языке FBD, указывает на то, что он выполняет несколько функций, и это мешает оценить надёжность его работы. **Реализация в инструментах статического анализа**: • SE – EcoStruxure MACA: Number Of FBD Networks, Halstead Complexity for FBD

Возможные меры по улучшению качества:

- если POU на SFC/FBD невозможно охватить взглядом (требуется увеличение масштаба или прокрутка окна редактора), то следует рассмотреть вариант распределения его функциональности по нескольким отдельным POU;
- если отдельные цепи FBD сложны для понимания, то попробуйте распределить их функциональность по нескольким отдельным цепям;
- используйте в FBD функциональные блоки с высоким уровнем абстракции, решающие конкретные задачи из предметной области (например, управление насосом, обработка аварий и т. д.), а не простые «базовые» блоки (триггеры, счётчики, таймеры); это приводит к уменьшению количества и размеров цепей.

Метрики, связанные переменными и интерфейсами POU

Использование глобальных переменных

↑ **Высокий показатель**: 👎
↓ **Низкий показатель**: 👍

Причина: изменения в POU, выполняющем запись в глобальные переменные, может существенно повлиять на другие POU, считывающие значения этих переменных. Кроме того, передача данных через глобальные переменные затрудняет поиск ошибок в ПО.

Реализация в инструментах статического анализа:

- CODESYS Static Analysis: Used different global variables
- SE – EcoStruxure MACA: Number of GVL Usages
- SE – EcoStruxure CE-V: extvarref

Возможные меры по улучшению качества:

- проверьте, является ли использование глобальных переменных действительно необходимым и могут ли быть нужные для POU данные получены им «напрямую», через входные переменные;
- если по каким-то причинам отказаться от использования глобальных переменных нельзя, то проверьте, что они сгруппированы логичным образом (например, функционально связанные переменные выделены в структуры);
- изучите POU с аномально большим числом входов и выходов – возможно, их функциональность может быть распределена по отдельным POU или методам / действиям;
- если различные переменные из области входов/выходов обрабатываются совместно в разных фрагментах кода – то выделите их в структуру;
- проектируйте интерфейс (набор входов и выходов) POU таким образом, чтобы получать и передавать через него все нужные данные, избегая использования глобальных переменных.

Метрики, связанные с МЭК ООП-элементами
Сцепление между POU и окружающими объектами ↑ **Высокий показатель**: 👎 ↓ **Низкий показатель**: 👍 **Причина**: большое количество зависимостей программного модуля от окружающих объектов может привести к перекрёстным эффектам при внесении в него изменений и снизить надёжность ПО. **Реализация в инструментах статического анализа**: • CODESYS Static Analysis: RFC – Response For Class, CBO – Coupling Between Objects • SIG Sigrid Maintainability: Module coupling
Возможные меры по улучшению качества: • слишком высокая степень сцепления или отсутствие связности могут быть показателями того, что распределение функциональности между модулями сделано не лучшим образом. Следует проверить, можно ли провести рефакторинг соответствующих методов или перераспределить их функциональность.

Метрики, связанные со сложностью ПО
Сложность и объём исходного кода ↑ **Высокий показатель**: 👎 ↓ **Низкий показатель**: 👍 **Причина**: чем выше сложность ПО, тем больше риск допустить ошибку при внесении изменений – и, соответственно, тем ниже надёжность системы. **Реализация в инструментах статического анализа**: • CODESYS Static Analysis: Halstead (D/HV/HL), Complexity (McCabe) • SE – EcoStruxure MACA: Halstead Complexity, Cyclomatic Complexity • SE – EcoStruxure CE-V: length, volume, difficulty, vg • SIG Sigrid Maintainability: Unit size, unit complexity (McCabe)
Возможные меры по улучшению качества: • проверьте, можно ли реализовать данный функционал более понятным образом, используя меньше циклов и вложенных цепей; • проверьте, можно ли сократить число используемых операторов – например, путем рефакторинга слишком больших POU и перераспределения их функциональности.

Метрики, связанные с потоками данных

«Прямой» обмен данными через вызовы POU

↑ **Высокий показатель**: 👍
↓ **Низкий показатель**: -

Причина: значительное количество вызовов POU указывает на то, что его функциональность используется множеством других POU. Это увеличивает вероятность обнаружения (и исправления) ошибок в данном POU, что повышает надёжность ПО.

Реализация в инструментах статического анализа:

- CODESYS Static Analysis: Number of calls
- SE – EcoStruxure MACA: Call In, Call Out
- SE – EcoStruxure CE-V: calledcount, callproc

Метрики, связанные с индикаторами возможности повторного использования

Количество использованных библиотек

↑ **Высокий показатель**: 👍
↓ **Низкий показатель**: 👎

Причина: использование библиотечных элементов, которые обычно тщательно протестированы и рассчитаны на применение в различных приложениях, повышает надёжность ПО.

Реализация в инструментах статического анализа:

- SE – EcoStruxure MACA: Number Of Library References

Глубина вызовов

↑ **Высокий показатель**: 👎
↓ **Низкий показатель**: 👍

Причина: если в POU, расположенном в конце длинной цепочки вложенных вызовов, вносятся изменения, то существует вероятность, что потенциально совершённые ошибки повлияют и на вызывающие его POU, что ухудшает надёжность ПО.

Реализация в инструментах статического анализа:

- SE – EcoStruxure CE-V: calldepthmin, calldepthmax

Дублирование кода

↑ **Высокий показатель**: 👎
↓ **Низкий показатель**: 👍

Причина: при написании кода методом «копипасты» (копирования и вставки) допущенные ошибки «расползаются» по всему приложению. Устранение этих ошибок для восстановления надёжности ПО займёт много времени, потому что потребуется вносить исправления в несколько фрагментов проекта.

Реализация в инструментах статического анализа:

- CODESYS Static Analysis: Duplication ratio
- SIG Sigrid Maintainability: Duplication

⦸ **Потенциальные конфликты**:
↓ увеличение количества зависимостей в проекте из-за появления фрагментов, в которых повторно вызывается один и тот же код (например, через вызов функции или механизмы объектно-ориентированного программирования)

Возможные меры по улучшению качества:

- если проект ПЛК или его фрагменты характеризуются высоким уровнем глубины вызовов, то следует проанализировать, можно ли его уменьшить путём объединения нескольких POU в один.

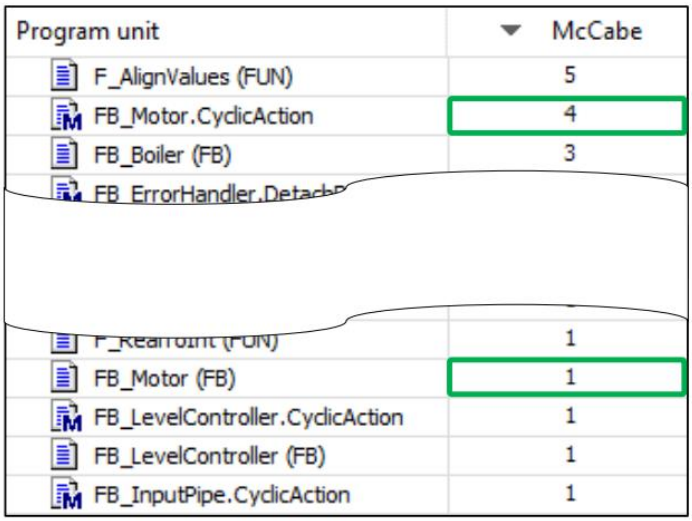
***Отказ от ответственности*:** приведённые ниже фрагменты «качественного» и «некачественного» кода взяты из учебного, «идеализированного» примера управления водонагревателем, который не призван показать весь возможный спектр функционала ПО для ПЛК, применяемого в промышленности. Эти фрагменты не связаны друг с другом – то есть пример «качественного» кода не является оптимизированной версией примера «некачественного» кода. Так как исходный проект разработан в среде CODESYS, то для анализа кода использовался инструмент CODESYS Static Analysis.

Пример «некачественного» кода: как и в аналогичном разделе п. 5.1, в некоторых POU можно заметить «выбросы» – высокие показатели сложности Холстеда и МакКейба (автора концепции цикломатической сложности). Вполне вероятно, что они приведут к снижению надёжности и, соответственно, являются потенциальными точками для оптимизации – например, путём инкапсуляции кода.

Program unit	McCabe	D (Halstead)
FB_BasicPIDCtrl (FB)	14	26,1
FB_ErrorHandler (FB)	14	37,9
FC_ErrorCode (FUN)	8	2,8
FB_AutomaticMode (FB)	7	12,4
F_AlignValues (FUN)	5	12,3
FB_Motor.CyclicAction	4	5,6
FB_PRAND (FB)	3	6,5

Рис. 16. Таблица метрик из проекта CODESYS, отсортированная по убыванию цикломатической сложности

Пример «качественного» кода: см. информацию из аналогичного раздела п. 5.2. Выделение функциональности проекта в небольшие и простые POU может повысить надёжность ПО.



Program unit	McCabe
F_AlignValues (FUN)	5
FB_Motor.CyclicAction	4
FB_Boiler (FB)	3
FB_ErrorHandler.DetachP	
F_Rearrunt (FUN)	1
FB_Motor (FB)	1
FB_LevelController.CyclicAction	1
FB_LevelController (FB)	1
FB_InputPipe.CyclicAction	1

Рис. 17. Фрагмент таблицы метрик, демонстрирующий снижение цикломатической сложности после оптимизации POU

6. Пороговые значения метрик и их использование в масштабных приложениях для ПЛК

В IT-отрасли определены пороговые значения, которые помогают ориентироваться при интерпретации результатов расчёта метрик (см. таблицу 2). Анализ и сравнение этих значений, используемых в литературе и инструментах статического анализа, показывают, что даже для языков программирования высокого уровня (C++, C#, Java и т. д.) нет общего консенсуса относительно их диапазонов. Даже для одной и той же метрики разные авторы и разработчики статических анализаторов приводят существенно отличающиеся пороговые значения (например, для цикломатической сложности – в оригинальной статье МакКейба [10] указано значение 10, в Squore Vector – 15, а в статье Tarcísio et al [16] – всего 2). Тем не менее, такие значения можно использовать как приблизительные рекомендации при оценке сложности и определения «аномалий» в метриках ПО для ПЛК, программируемых на языках стандарта МЭК 61131-3 – особенно на ST, который своим синтаксисом похож на языки высокого уровня.

Объектно-ориентированные расширения стандарта МЭК 61131-3, присутствующие в некоторых современных платформах промышленной автоматизации (например, CODESYS и основанных на нём IDE), концептуально позволяют разрабатывать ПО для ПЛК, которое ни в чём не уступает ПО, написанному на языках высокого уровня. Поэтому в контексте приложений ПЛК, для разработки которых использовались современные «best practices» из мира IT (например, ООП), подразумевается, что приведённые в таблице ниже пороговые значения могут быть использованы в качестве надёжных ориентиров. Тем не менее, следует отметить, что между ПО для ПЛК и ПО, разрабатываемого на языках высокого уровня, часто существуют значительные отличия, которые могут ограничить использование этих пороговых значений – в частности, для приложений ПЛК, написанных на графических языках или в процедурном стиле. Переоценка пороговых значений для конкретных приложений должна проводиться эвристически и основываться на логически обоснованных тезисах, сформулированных с помощью причинно-следственного анализа – например, определения порогового значения сложности для конкретного POU, при превышении которого он считается неприспособленным для сопровождения.

Один из таких подходов, связывающих значение метрики с её воздействием на ПО, описан в [17] – он представляет собой технологически-независимый способ оценки метрик на основе эталонных показателей. В [1] приводятся результаты повторных исследований, показывающие, что существует корреляция между временем, потраченным на решение проблем с ПО (исправление ошибок, внедрение улучшений, выпуск патчей и т. д.) и значениями метрик сопровождаемости: чем выше уровень сопровождаемости – тем быстрее получается вносить изменения в ПО.

Табл. 2. Составной список предлагаемых пороговых значений метрик для языков программирования высокого уровня, сформированный в результате анализа статических анализаторов и результатов опубликованных исследований

- **LOC** – Lines of Code (количество строк кода);
- **DIT** – Depth of Inheritance Tree (глубина наследования);
- **CBO** – Coupling Between Objects (сцепление между объектами);
- **NOC** – Number Of Children (количество наследников);
- **LCOM** – Lack of Cohesion Of Methods (отсутствие связанности методов);
- **RFC** – Response For Class (количество методов, вызываемых классом).

Источник информации	**Метрика (связанная с одной из метрик п. 5)**	**Предлагаемое пороговое значение**
Squore Vector (from an example backend project in php) [18]	Цикломатическая сложность Кол-во параметров метода Кол-во комментариев в «заголовке» модуля Кол-во выполняемых операторов	≤ 15 ≤ 5 > 0 ≤ 50
McCabe (intro of complexity metric) [10, 19]	Цикломатическая сложность	≤ 10
embold Metric Thresholds (Component Level) [20]	Кол-во строк кода Степень комментирования кода LCOM Цикломатическая сложность	≤ 1000 > 30 ≤ 0.77 50
PHP Mess Detector [21]	Цикломатическая сложность Кол-во строк кода (в классе) Кол-во методов	≤ 10 ≤ 1000 ≤ 25
Holzmann (NASAs 10 rules) [22]	LOC	≤ 60
Shatnawi Raed (analysis of 11 open-source java projects) [23]	DIT CBO	≤ 3 ≤ 17
Tarcísio et al. (analysis of Qualitas Corpus: a curated collection of software systems in java) [16]	DIT NOC LCOM Цикломатическая сложность Кол-во параметров	≤ 2 ≤ 1 ≤ 0.167 ≤ 2 ≤ 2
Rosenberg et al. (expert audit by NASAs Software Assurance. Technology Center in C++/Java) [24]	DIT CBO RFC	2 – 5 ≤ 5 ≤ 50
Herbold, Grabowski, Waack (collection of research results) [25]	Цикломатическая сложность для C Цикломатическая сложность для C++ Цикломатическая сложность для C# CBO для Java RFC для Java	≤ 24 ≤ 10 ≤ 10 ≤ 5 ≤ 100
Alves et al. (technology-agnostic benchmark-based metric calculation) [17]. New thresholds cf. [27] *Обратите внимание, что разработчикам не обязательно добиваться того, чтобы каждый POU находился строго в зоне «низкого риска». Рассматриваемый в исследовании подход предназначен для оценки кодовой базы в целом и позволяет определить связь между значениями метрик и реальными целевыми показателями системы автоматизации*	Размер модуля (кол-во строк кода) Сложность модуля (цикломатическая сложность) Сцепление между модулями (зависимость POU от входных данных, получаемых из других POU)	низкий риск: 15 средний риск: 16-30 высокий риск: 31-60 оч. высокий риск: 60+ низкий риск: 1-6 средний риск: 6-10 высокий риск: 11-25 оч. высокий риск: 26+ низкий риск: 0-10 средний риск: 11-20 высокий риск: 21-50 оч. высокий риск: 51+

При разработке приложений ПЛК используются различные языки программирования стандарта МЭК 61131-3 для создания POU разной функциональности (от выполнения простых вычислений до управления сложными технологическими установками), что затрудняет определение каких-то конкретных пороговых значений метрик – поэтому в данном руководстве они не указываются, чтобы не давать простор для неправильных интерпретаций. Но чтобы сориентировать читателя, как анализировать значения метрик в рамках ПО для ПЛК, далее приводятся некоторые примеры, в которых рассматриваются метрики из п. 5, значения которых были рассчитаны для фрагментов двух реальных промышленных проектов, признанных репрезентативным для широкого спектра систем автоматизации. Расчёт метрик производился с помощью анализатора кода **SE – EcoStruxure Machine Advisor**, интегрированного в среду разработки **SE – EcoStruxure Machine Expert**.

Документирование кода: хорошо структурированный промышленный пример: в результате статического анализа разработчик получает исчерпывающий обзор о том, какой процент переменных в списках глобальных переменных сопровождается комментариями. Эта ценная информация может использоваться для утверждения списка переменных проекта или подтверждения соблюдения внутренних правил оформления кода в компании.

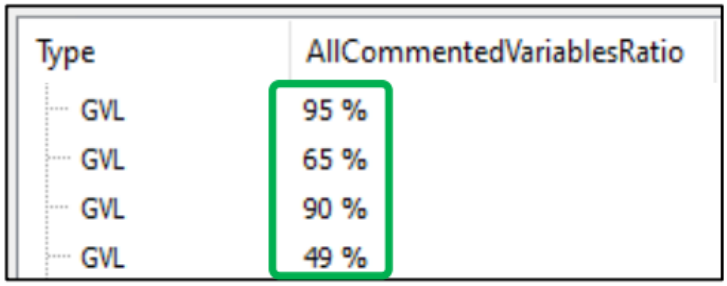


Рис. 18. Таблица метрик из ПО **EcoStruxure Machine Expert**, демонстрирующая хорошую степень документированности списков глобальных переменных

Документирование кода: валидация документации: хотя значение метрики, казалось бы, указывает на исчерпывающее документирование этого POU, вполне очевидно, что оно искажено из-за закомментированного «мёртвого кода», который не имеет отношения к документации. Поэтому следует проводить валидацию документации. Метрики, связанные с документацией, особенно сильно подвержены как преднамеренным, так и непреднамеренным искажениям, но при должном внимании и усердии такие проблемы можно отследить и решить.

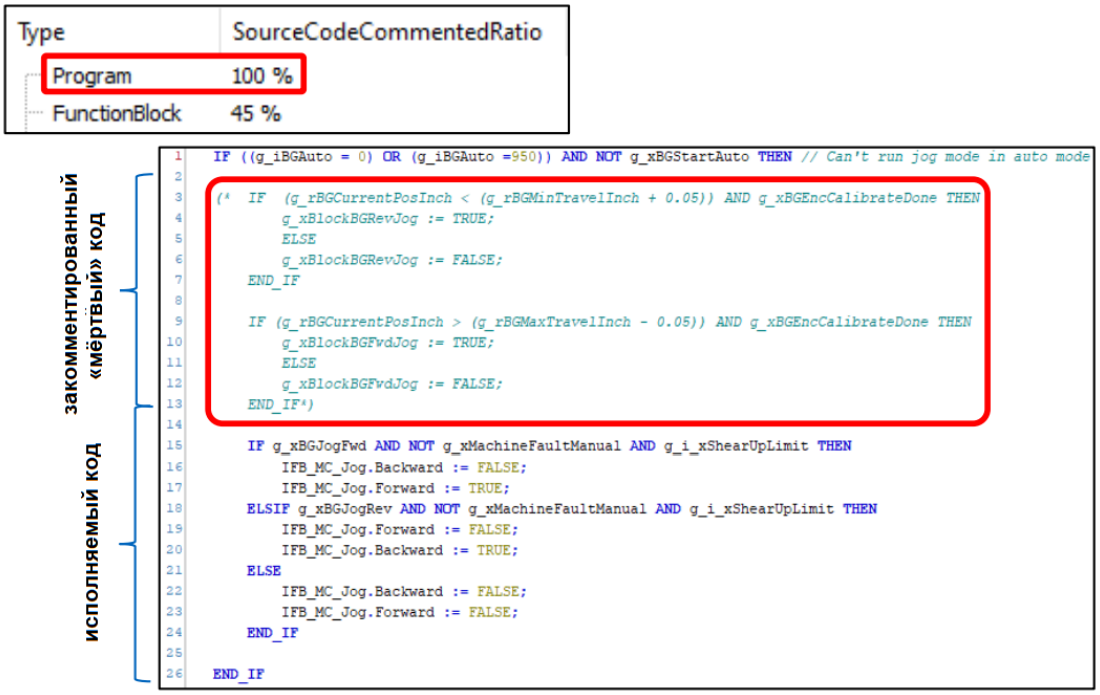


Рис. 19. Фрагмент POU, включающий в себя типичную ошибку – закомментированный «мёртвый» код – и соответствующее искажение в таблице метрик

Сложность ПО: пример очень сложного промышленного приложения: представленная здесь таблица метрик взята из реального промышленного приложения, которое используется в качестве негативного примера с большим потенциалом для оптимизации – особенно в плане сопровождаемости. Тем не менее, этот пример подчёркивает разброс числовых значений метрик – особенно по сравнению с идеализированными, «дидактическими» примерами, использованными в п. 5. Кроме того, он демонстрирует невозможность выработки универсальных рекомендаций по допустимому диапазону значений метрик, потому что они должны оцениваться отдельно для каждого конкретного случая экспертами по соответствующей предметной области.

Type	CyclomaticComplexity
FunctionBlock	133
Action	71
FunctionBlock	68
FunctionBlock	62
Action	59
Action	52
Action	52
FunctionBlock	50
Action	46
FunctionBlock	43
Action	37
Action	37
Action	37
Action	37

Type	HalsteadDifficulty
Action	92.81
Action	72.75
Action	71.35
Action	71.35
Action	71.35
Action	70.77
FunctionBlock	69.47
Function	65.07
FunctionBlock	64.13
Function	62.49
Function	60
Action	56.57
Function	54.07
Action	53.21
Action	53.21

Рис. 20. Таблица метрик из ПО **EcoStruxure Machine Expert**, демонстрирующая разброс числовых значений метрик

Поток данных: сравнение промышленных приложений: этот пример развивает идею о том, что нецелесообразно сравнивать значения метрик различных проектов и пытаться уложить их в какой-то универсальный диапазон. Вместо этого разумнее изучить «выбросы» и общие закономерности, а сравнение значений метрик проводить только для похожих проектов или проектов, связанных с одной и той же предметной областью.

Type	NumOfReads	NumOfWrites
Action	242	333
Action	242	333
Action	242	333
Action	242	334
Action	124	159
Action	122	158
Action	97	99
Action	83	73

Type	NumOfReads	NumOfWrites
Program	47	32
Program	34	18
Program	27	19
FunctionBlock	24	25
Program	23	22
Program	21	16
Program	20	15
FunctionBlock	17	13

Рис. 21. Таблицы метрик из двух различных промышленных приложений, демонстрирующие изменчивость их значений от проекта к проекту

Список использованной литературы

[1] J. Wijnmaalen, C. Chen, D. Bijlsma, and A. M. Oprescu, “The Relation between Software Maintainability and Issue Resolution Time: A Replication Study,” in SaTToSE, 2019.
[2] H. Zhu, Q. Zhang, and Y. Zhang, “HASARD: A Model-Based Method for Quality Analysis of Software Architecture,” in Relating System Quality and Software Architecture: Elsevier, 2014, pp. 123–156.
[3] B. W. Boehm, J. R. Brown, and M. Lipow, “Quantitative evaluation of software quality,” in International Conference on Software Engineering, 1976.
[4] R. G. Dromey, “A model for software product quality,” IEEE Transactions on Software Engineering, vol. 21, no. 2, pp. 146–162, 1995.
[5] J. A. McCall, P. K. Richards, and G. F. Walters, “Factors in Software Quality. Volume I. Concepts and Definitions of Software Quality,” 1977.
[6] Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models, 25010, ISO/IEC, 2011.
[7] B. Vogel-Heuser, A. Fay, I. Schaefer, and M. Tichy, “Evolution of software in automated production systems: Challenges and research directions,” Journal of Systems and Software, vol. 110, pp. 54–84, 2015, doi: 10.1016/j.jss.2015.08.026.
[8] IEEE Standard for a Software Quality Metrics Methodology, 1061, IEEE, Piscataway, NJ, USA, 1998.
[9] S. R. Chidamber and C. F. Kemerer, “A metrics suite for object oriented design,” IEEE Transactions on Software Engineering, vol. 20, no. 6, pp. 476–493, 1994.
[10] T. J. McCabe, “A Complexity Measure,” IEEE Transactions on Software Engineering, SE-2, no. 4, pp. 308–320, 1976, doi: 10.1109/tse.1976.233837.
[11] M. H. Halstead, Elements of software science. New York, NY: North-Holland, 1977.
[12] E. M. Neumann et al., “Metric-based Identification of Target Conflicts in the Development of Industrial Automation Software Libraries,” in IEEE International Conference on Industrial Engineering and Engineering Management (IEEM), Kuala Lumpur, Malaysia, 2022, pp. 1493–1499.
[13] Programmable controllers - Part 3: Programming languages (IEC 61131-3:2013); German version EN 61131-3:2013, 61131-3, International Electrotechnical Commission (IEC), 2014.
[14] Systems and software engineering - Vocabulary, 24765, ISO/IEC/IEEE, Piscataway, NJ, USA, 2010.
[15] E.-M. Neumann et al., “Identifying Runtime Issues in Object-Oriented IEC 61131-3-Compliant Control Software using Metrics,” in Annual Conference of the IEEE Industrial Electronics Society (IECON), Singapore, Singapore, 2020, pp. 259–266.
[16] Tarcísio G. S. Filó, “A Catalogue of Thresholds for Object-Oriented Software Metrics,” pp. 48–55, 2015. [Online]. Available: http://personales.upv.es/thinkmind/dl/conferences/softeng/softeng_2015/softeng_2015_3_10_55070.pdf
[17] T. L. Alves, J. P. Correia, and J. Visser, “Benchmark-Based Aggregation of Metrics to Ratings,” in Joint Conf. of the 21st Int. Workshop on Software Measurement and the 6th Int. Conf. on Software Process and Product Measurement, Nara, Japan, 2011, pp. 20–29.
[18] Vector Informatik GmbH, Squore: Analytics for Projects Monitoring. [Online]. Available: https://www.vector.com/int/en/products/products-a-z/software/squore/ (accessed: Jul. 18 2023).
[19] A. H. Watson and T. J. McCabe, “Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric,” 1996.
[20] embold, Metric Thresholds. [Online]. Available: https://docs.embold.io/de/metric-thresholds/ (accessed: Jul. 18 2023).
[21] PHP, Mess Detector - Code Size Rules. [Online]. Available: https://phpmd.org/rules/codesize.html
[22] G. J. Holzmann, “The Power of Ten - Rules for Developing Safety Critical Code,” 2006.
[23] R. Shatnawi, “Deriving metrics thresholds using log transformation,” Journal of Software: Evolution and Process, vol. 27, no. 2, pp. 95–113, 2015.
[24] L. Roseberg, R. Stapko, and A. Gallo, “Risk-based Object Oriented Testing,” 1999. [Online]. Available: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=0cadbbdc244f38e4dfbae022b5e5e3fdc8249dd0
[25] S. Herbold, J. Grabowski, and S. Waack, “Calculation and optimization of thresholds for sets of software metrics,” Empirical Software Engineering, vol. 16, no. 6, pp. 812–841, 2011.
[26] T. L. Alves, C. Ypma, and J. Visser, “Deriving metric thresholds from benchmark data,” in IEEE International Conference on Software Maintenance (ICSM), Timișoara, Romania, 2010.
[27] SIG/TÜViT, Evaluation Criteria Trusted Product Maintainability: Guidance for producers (V15.0). [Online]. Available: https://www.softwareimprovementgroup.com/software-analysis/

Приложение 1. Таблица соответствия доступных метрик и характеристик качества ПО

Легенда

++	существенная прямая корреляция между значением метрики и степенью характеристики качества
+	умеренная прямая корреляция между значением метрики и степенью характеристики качества
O	отсутствие корреляции между значением метрики и степенью характеристики качества
-	умеренная обратная корреляция между значением метрики и степенью характеристики качества
--	существенная обратная корреляция между значением метрики и степенью характеристики качества

Метрики, связанные с размером

CODESYS Group		**Schneider Electric – EcoStruxure Machine Advisor Code Analysis**		**Schneider Electric – Control Engineering – Verification (ранее известный как Itris PLC Checker)**		**Software Improvement Group Sigrid**		**Надёжность**	**Уровень производительности**	**Сопровождаемость**	**Возможность повторного использования**	**Тестируемость**
Метрика	Описание	Метрика	Описание	Метрика	Описание	Метрика	Описание					
NOS	Кол-во операторов			Number of instructions	Кол-во инструкций (условных операторов, операторов цикла, операторов *break*) в данном POU			O	O	-	-	-
		Lines Of Code (LOC)	Кол-во строк исходного кода программы			Unit size	Кол-во строк кода конкретного POU (функции, ФБ, цепи и т. д. – в зависимости от языка программирования. Для графических языков каждый элемент в цепи добавляет +1 к значению метрики)	O	O	-	-	-

Code Size, Variable Size	Кол-во байт	Memory Size (Data)	Объём выделенной и обрабатываемой памяти для каждого экземпляра комплексного типа (информация о типе и его переменных)	Memory size	Кол-во бит			O	-	O	O	O
		Number Of Actions	Кол-во действий, привязанных к программе или функциональному блоку	plcobjecttype counter	Для каждого типа ПЛК подсчитывается кол-во объектов, найденных в приложении (функциональных блоков, функций, программ, подпрограмм и т. д.)			O	-	+	O	+

Метрики, связанные с элементами, специфичными для конкретного языка

CODESYS Group		**Schneider Electric – EcoStruxure Machine Advisor Code Analysis**		**Schneider Electric – Control Engineering – Verification (ранее известный как Itris PLC Checker)**		**Software Improvement Group Sigrid**		**Надёжность**	**Уровень производительности**	**Сопровождаемость**	**Возможность повторного использования**	**Тестируемость**
Метрика	Описание	Метрика	Описание	Метрика	Описание	Метрика	Описание					
SFC branches	Кол-во ветвей в POU на языке SFC			nbofbranches	Кол-во ветвей в данной SFC-подпрограмме			--	O	--	-	--
SFC steps	Кол-во шагов в POU на языке SFC							-	O	-	-	-
		Number Of FBD Networks	Кол-во цепей в POU на языке FBD (программе, ФБ, функции, методе или свойстве)					-	O	-	-	-
		Halstead Complexity for FBD	Метрика сложности Холстеда. Рассчитывается на основании общего кол-ва операторов/операндов и кол-ва уникальных операторов/операндов					-	O	-	-	-
		Number Of Transitions	Кол-во переходов в POU на языке SFC					-	O	-	-	-
				g7height	Сумма шагов и переходов данной SFC-подпрограммы без учёта расходящихся ветвей («высота»)			-	O	-	-	-
				g7width	Максимальное кол-во «одноуровневых» (расходящихся или параллельных) ветвей данной SFC-подпрограммы («ширина»)			-	O	-	-	-

Метрики, связанные переменными и интерфейсами POU

CODESYS Group		**Schneider Electric – EcoStruxure Machine Advisor Code Analysis**		**Schneider Electric – Control Engineering – Verification (ранее известный как Itris PLC Checker)**		**Software Improvement Group Sigrid**		**Надёжность**	**Уровень производительности**	**Сопровождаемость**	**Возможность повторного использования**	**Тестируемость**
Метрика	Описание	Метрика	Описание	Метрика	Описание	Метрика	Описание					
Global	Кол-во уникальных глобальных переменных	Number Of GVL Usages	Кол-во глобальных переменных, используемых данным POU (программой, ФБ, функцией, методом и т. д.) для чтения или записи	extvarref	Кол-во обращений в коде POU к внешним объектам (т. е. не к его параметрам и локальным переменным)			-	-	--	--	O
I/Os	Кол-во операций «прямого доступа» ко входам и выходам							O	O	O	-	-
Local	Кол-во локальных переменных	Number Of Variables	Кол-во переменных в области объявления объекта (программы, ФБ, функции, метода, свойства, перехода, списка глобальных переменных и т. д.)					O	-	O	O	O
Inputs	Кол-во входных переменных			inputcount	Кол-во входных параметров данного POU			O	O	-	O	-
Outputs	Кол-во выходных переменных			outputcount	Кол-во выходных параметров данного POU			O	O	-	O	-
				nbofparam	Общее кол-во параметров (входных, выходных и входов-выходов) данного POU	Unit interfacing	Общее кол-во параметров (входных, выходных и входов-выходов) данного POU	O	O	-	O	-

Метрики, связанные со встроенной документацией

CODESYS Group		**Schneider Electric – EcoStruxure Machine Advisor Code Analysis**		**Schneider Electric – Control Engineering – Verification (ранее известный как Itris PLC Checker)**		**Software Improvement Group Sigrid**		**Надёжность**	**Уровень производительности**	**Сопровождаемость**	**Возможность повторного использования**	**Тестируемость**
Метрика	Описание	Метрика	Описание	Метрика	Описание	Метрика	Описание					
Comments	Процент кода, снабжённый комментариями	Source Code Comment Ratio	Процентное соотношение между строками комментариев и строками исходного кода в области кода POU	percentage of comment	Процент кода, снабжённый комментариями – глобальная оценка на уровне всего приложения			O	O	++	O	O
		Commented Variables (All) Ratio	Процентное соотношение между прокомментированными и не прокомментированными переменными POU (касается всех переменных)	result of Verification tool				O	O	++	O	O
		Commented Variables (In + Out + Global) Ratio	Процентное соотношение между прокомментированными и не прокомментированными переменными, объявленными в области VAR_GLOBAL, VAR_INPUT, VAR_OUTPUT и VAR_IN_OUT					O	O	++	O	O
		Number Of Multiline Comments	Кол-во многострочных (занимающих более одной строки) комментариев в данном объекте					O	O	++	O	O
		Number Of Header Comment Lines	Кол-во строк комментариев в «заголовке» области объявления объекта (т.е. – над строкой с ключевым словом PROGRAM, FUNCTION BLOCK и т. д.)					O	O	++	O	O

Метрики, связанные с МЭК ООП-элементами

CODESYS Group		**Schneider Electric – EcoStruxure Machine Advisor Code Analysis**		**Schneider Electric – Control Engineering – Verification (ранее известный как Itris PLC Checker)**		**Software Improvement Group Sigrid**		**Надёжность**	**Уровень производительности**	**Сопровождаемость**	**Возможность повторного использования**	**Тестируемость**
Метрика	Описание	Метрика	Описание	Метрика	Описание	Метрика	Описание					
DIT	Глубина наследования							O	O	-	++	+
NOC	Кол-во дочерних объектов	Extended By	Кол-во наследников данного ФБ или интерфейса					O	O	-	++	+
		Extends	Кол-во ФБ и интерфейсов, имеющих наследников					O	O	-	++	+
RFC								O	O	--	-	--
CBO	Уровень сцепления между объектами					Component coupling	Уровень зависимости архитектурных компонентов друг от друга и от других компонентов, входящих в состав системы	-	O	-	--	--
LCOM	Уровень **отсутствия** связности в методах					Component cohesion (обратно пропорциональна метрике LCOM, используемой в CODESYS)	Уровень инкапсуляции бизнес-логики в архитектурных компонентах системы	O	O	-	-	-
		Implemented By	Кол-во ФБ, реализующих данный интерфейс					O	O	O	+	+

		Implements	Кол-во интерфейсов, реализуемых данным ФБ					O	-	O	+	+
		Number Of Methods	Кол-во методов программы или ФБ					O	-	+	+	+
		Number Of Properties	Кол-во свойств программы или ФБ					O	-	+	+	+

Метрики, связанные со сложностью ПО

CODESYS Group		**Schneider Electric – EcoStruxure Machine Advisor Code Analysis**		**Schneider Electric – Control Engineering – Verification (ранее известный как Itris PLC Checker)**		**Software Improvement Group Sigrid**		**Надёжность**	**Уровень производительности**	**Сопровождаемость**	**Возможность повторного использования**	**Тестируемость**
Метрика	Описание	Метрика	Описание	Метрика	Описание	Метрика	Описание					
HL (Halstead)	«Длина» POU по метрике Холстеда			length				-	O	-	-	-
HV (Halstead)	«Объём» POU по метрике Холстеда			volume				-	O	-	-	-
D (Halstead)	Уровень сложности метрики Холстеда	Halstead Complexity (для ST и FBD)	Метрика сложности Холстеда. Рассчитывается на основании общего кол-ва операторов/операндов и кол-ва уникальных операторов/операндов	difficulty				-	O	--	-	--
McCabe	Уровень цикломатической сложности МакКейба	Cyclomatic Complexity	Уровень цикломатической сложности МакКейба, определённый путём подсчёта кол-ва линейно независимых путей потока управления в исходном коде	vg		Unit complexity	Уровень цикломатической сложности МакКейба конкретного объекта (функции, ФБ, цепи и т. д. – в зависимости от языка программирования. Для графических языков элемент с несколькими исходящими связями считается точкой ветвления потока управления)	-	O	--	-	--

Метрики, связанные с потоком данных

CODESYS Group		**Schneider Electric – EcoStruxure Machine Advisor Code Analysis**		**Schneider Electric – Control Engineering – Verification (ранее известный как Itris PLC Checker)**		**Software Improvement Group Sigrid**		**Надёжность**	**Уровень производительности**	**Сопровождаемость**	**Возможность повторного использования**	**Тестируемость**
Метрика	Описание	Метрика	Описание	Метрика	Описание	Метрика	Описание					
Calls	Кол-во вызовов	Call In	Кол-во вызовов данного POU другими POU	calledcount	Кол-во вызовов данного POU по ссылке			+	O	O	++	O
		Call Out	Кол-во вызовов других POU в данном POU	callproc	Кол-во вызовов других POU в данном POU (как пользовательских, так и системных)			O	-	O	+	+
		Number Of Writes	Кол-во операций записи переменных в данном POU	maxmemwrite	Максимальное количество (в случае структуры с различными полями) операций записи, применяемых к данной ячейке памяти			O	O	O	O	-
		Number Of Reads	Кол-во операций чтения переменных в данном POU	minmemwrite	Минимальное количество (в случае структуры с различными полями) операций записи, применяемых к данной ячейке памяти			O	O	O	O	-
		Fan Out	Кол-во POU, вызываемых в данном POU					O	O	--	-	--

Метрики, связанные с индикаторами возможности повторного использования

CODESYS Group		Schneider Electric – EcoStruxure Machine Advisor Code Analysis		Schneider Electric – Control Engineering – Verification (ранее известный как Itris PLC Checker)		Software Improvement Group Sigrid		Надёжность	Уровень производительности	Сопровождаемость	Возможность повторного использования	Тестируемость
Метрика	Описание	Метрика	Описание	Метрика	Описание	Метрика	Описание					
		Number Of Library References	Кол-во библиотек, используемых в проекте					+	O	O	O	O
				calldepthmin	Минимальная глубина стека вызовов			-	O	-	-	-
				calldepthmax	Максимальная глубина стека вызовов (примечание: метрика не вычисляется в случае рекурсивного вызова)			-	O	-	-	-
Clone ratio	Процент «скопипащенного» кода в проекте					Duplication	Коэффициент дублирования (и/или избыточности) кода в кодовой базе, вызванного «копипастой»	-	O	--	-	-