# Описание архитектуры систем в виде программного кода и автоматизированный контроль архитектуры

**Герман Хохлов**

Январь 2021

## Введение

В различных областях деятельности встречается задача контроля систем различного класса (информационных, производственных, инженерных, систем безопасности и т.п.) на соответствие заранее заданным требованиям. В данной статье будут рассматриваться требования, предъявляемые не к отдельно взятой системе, а определенному классу систем. Такие наборы требований, выраженные в виде технических документов, определяющих термины, правила, требования к порядку выполнения определенных действий, будем называть стандартами [1].

Количество систем, требующих контроля на соответствие требованиям стандартов в организациях, постоянно увеличивается. Одновременно увеличивается функциональность и сложность этих систем, что приводит к повышению трудоемкости контроля на всех этапах жизненного цикла – выбор, создание, эксплуатация. Необходимость контроля на соответствие систем стандартам и проектной документации является существенной фактором, влияющим на трудозатраты, сроки, бюджеты и риски проектов.

Для оптимизации и повышения эффективности контрольных процедур активно используются средства автоматизации. Однако, как правило, эти методики контроля сильно зависят от технологической реализации объектов контроля, методик сбора данных, а также от предметной области: разработка программного обеспечения, технологическая инфраструктура, информационная безопасность, управленческий контроль и т.п.

В данной статье предлагается универсальная методика контроля архитектуры систем на соответствие требованиям стандартов. Данная методика не зависит от технологической реализации систем и предметной области.

Для обеспечения унификации архитектура системы, а также применяемые к ней стандарты формулируются в виде кода на комбинации языков программирования - языка разметки (markup language), декларативных и алгоритмических.

В качестве базового примера будет рассматриваться задача контроля информационных систем (ИС) на соответствие архитектурным стандартам и стандартам информационной безопасности.

## Архитектура системы в виде программного кода

По мере распространения облачных вычислений в информационных технологиях получила развитие методология управления технологической инфраструктурой Infrastructure as Code (IaC) [2]. Данная методология позволяет описывать и использовать информационные ресурсы в виде программного кода. Широко распространились соответствующие инструменты управления и контроля “на основе кода” - Ansible, Chef и другие. Подход Infrastructure as Code позволил значительно повысить эффективность управления информационными ресурсами, надежность эксплуатации технической инфраструктурой, а также позволяет объективно оценивать качество инфраструктурных сервисов.

Идеи распространения подхода управления «на основе кода» на прикладное программное обеспечение также активно обсуждаются Интернет.

В данной статье предлагается расширить и систематизировать идею управления с помощью программного кода на «системы» в обобщенном смысле этого термина [3].

В настоящее время преобладает подход к описанию архитектуры систем [4] – на основе текстовых описаний или графических диаграмм (IDEF0). Подобное представление архитектуры систем удобно для обсуждения или анализа специалистами, однако оно плохо подходит для автоматизированной обработки.

Для целей автоматизированной обработки предлагается описывать архитектуру систем в виде машино-читаемого кода. Для сохранения возможности ручного анализа данный код должен сохранить и свойство человеко-читаемости.

Примерами архитектурно-значимых свойств в ИС могут быть:

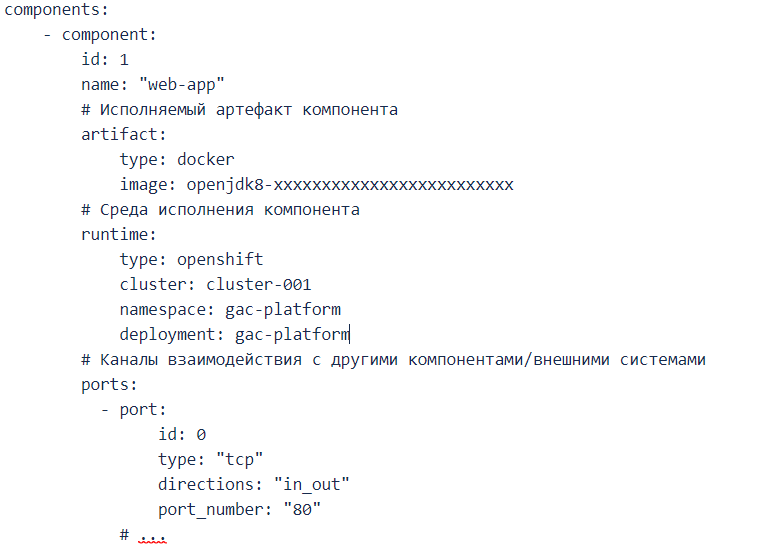
* Компонентный состав ИС
* Интеграционные потоки ИС
* Внутренние межкомпонентные взаимодействия
* Другие архитектурно-значимые свойства.

Для описания структуры (морфологии) систем хорошо подходят языки разметки (markup), например язык YAML [5]. Язык YAML уже широко используется для описании структуры приложений, размещаемых в облачных средах (cloud infrastructure). YAML хорошо подходит для описания структуры систем в силу того, что он отвечает следующим требованиям:

* «машино-читаемость» кода – возможность программной обработки артефактов
* «человеко-читаемость» кода - понятный человеку – простота разработки / сопровождения / контроля
* расширяемый язык
* язык поддерживает структуры данных
* переносимый между различными программными платформами.

На основе языка YAML может быть разработан доменный язык (DSL - Domen Specific Language) для описания архитектуры систем конкретного класса. Важно отметить, что семантика ключевых слов DSL может зависеть от контролируемой предметной области, но не зависит от конкретных объектов контроля в рамках этой предметной области, что обеспечивает применимость единого DSL к широкому классу систем.

**Пример описания структуры информационных систем на языке YAML приведен на Рисунке 1.**



***Рисунок 1. Пример архитектуры системы в виде кода на языке YAML***

Как видно из Рис.1, архитектура системы в виде программного кода может использоваться как для ручного анализа специалистами предметных областей, так и для автоматизированной обработки.

Для описания архитектурно-значимых свойств систем, выходящих за рамки описания структуры, языка разметки может оказаться недостаточно. Для этих целей могут использоваться декларативные и алгоритмические языки. Соответствующий пример описания архитектурных свойств на декларативном языке Rego приведен ниже.

Отметим, что архитектура системы может быть проектируемая (спроектированная) и фактическая. Как следует из названия, проектируемая архитектура разрабатывается специалистами в рамках прямого процесса проектирования. После чего (или в процессе) она может быть формализована в виде программного кода, как это описано выше. Различия между спроектированной и фактической архитектурами Системы могут быть довольно серьезными и должны быть явно проанализированы, описаны и согласованы.

Фактическая архитектура может быть восстановлена на основе фактически функционирующей системы методами реверсивной инженерии. Будем называть такую восстановленную фактическую архитектуру системы **Реверсивной архитектурой**.

Методы реверсивной инженерии достаточно разнообразны, но зависят от предметной области и технологий разработки системы. Например, для информационных систем это: статический анализ кода, анализ сетевого трафика, мониторинг производительности, анализ инцидентов, искусственный интеллект и др. [6].

Пример реверсивной архитектуры, сформированной на основе кода ИС, приведен на Рисунке 1.

В результате процессов прямого и обратного (реверсивного) проектирования могут быть сформированы два типа артефактов, выраженных в виде программного кода, которые могут использоваться для автоматизированного контроля. Общая схема автоматизированного контроля архитектуры системы на предмет соответствия стандартам приведена в следующем разделе.



***Рисунок 2. Пример реверсивной архитектуры системы***

## Общая схема контроля архитектуры систем на основе кода

Будем называть контрольные требования, выраженные в виде машино-читаемого программного кода **Смарт-стандартами**.

Смарт-стандарты могут разрабатываться на основе обычных артефактов – стандарты, проектная документация и т.п.

Пример Смарт-стандарта приведен на Рисунке 3.



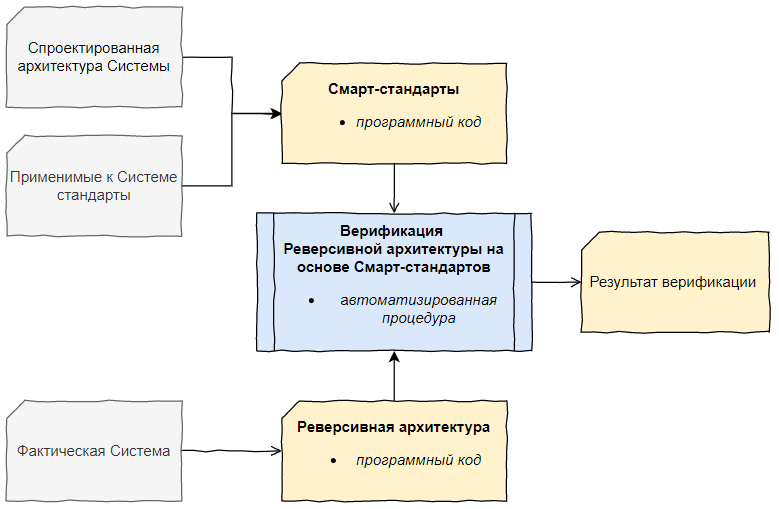
***Рисунок 3. Пример Смарт-стандарта на языке YAML***

Как видно из Рисунка 3, для разработки данного примера смарт-стандарта также был использован язык YAML. Для о писания правил использовался декларативный язык Rego.

Помимо формализации собственно требований, смарт-стандарты на основе программного кода могут:

* Взаимодействовать через программные интерфейсы с контролируемыми объектами, вызываться в автоматизированных процессах
* Перехватывать управление и инициировать новые автоматизированные процессы, в т.ч. на основе встроенных в смарт-стандарты контрактов на основе кода (смарт-контрактов)
* Служить “воротами качества” (quality gates) для контролируемых объектов разработки ПО / производственных процессов / процессов эксплуатации, т.п.
* Обеспечивать контроль соответствия контролируемых объектов политикам и стандартам.

На Рисунке 3 приведена схема контроля архитектуры Системы на основе программных артефактов Смарт-стандарт и Реверсивная архитектура.



***Рисунок 3. Схема контроля на основе программных артефактов***

Отметим, что артефакт Смарт-стандарт является чисто логическим и – он не зависит от технологической реализации Системы, технологии сбора и обогащения данных, специфики требований стандарта и т.п.

По мере развития методологии контроля на основе кода, может формироваться общая библиотека смарт-стандартов, которая может использоваться для широкого класса Систем, не зависимо от технологий их реализации, языков и т.п.

## Формирование артефактов на основе кода

Как отмечалось выше, артефакты «Смарт-стандарт» и «Реверсивная архитектура» не содержат зависимости от технологической реализации объекта управления или специфики требований политик управления и могут разрабатываться и развиваться не зависимо от объектов контроля.

Смарт-стандарт должен формироваться на основе требований стандартов путем переноса значимых для контроля требований в машино-читаемый вид. Как правило эта процедура требует участия специалиста.

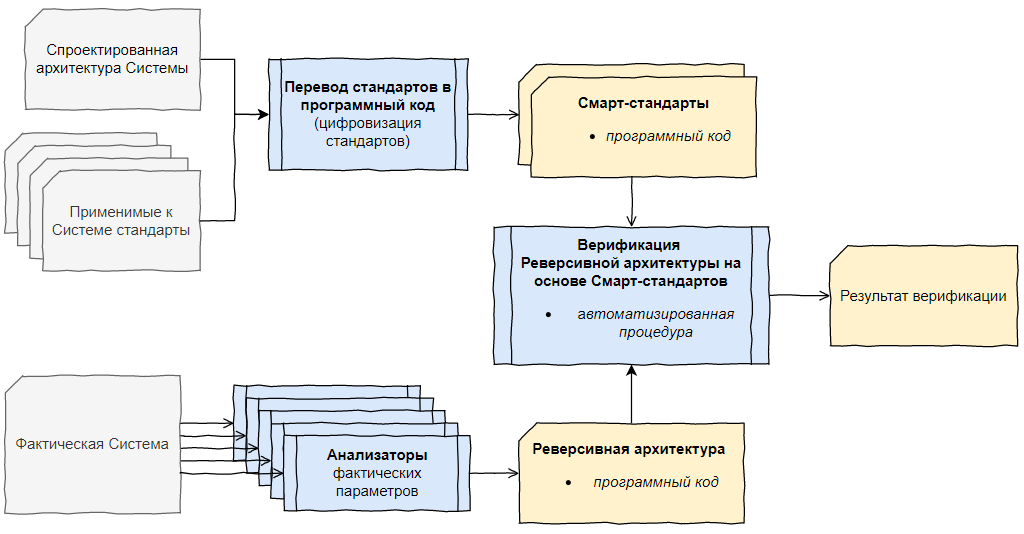
Для формирования Реверсивной архитектуры методами реверсивной инженерии могут использоваться дополнительные компоненты, которые будем называть **Анализаторами** архитектурных свойств. Анализаторы существенно зависят от технологии реализации системы, а также могут существенно зависеть от специфики контролируемого требования.

Анализаторы архитектуры ИС могут разрабатываться в виде специализированного ПО, например:

* Инструменты статического анализа кода информационных систем
* Инструменты мониторинга сетевого трафика, производительности, инцидентов, …
* Инструменты мониторинга производственных систем
* Анализаторы на основе искусственного интеллекта и машинного обучения.

По опыту применения методологии также может использоваться обогащение Реверсивной архитектуры экспертными знаниями специалистов, работающих с контролируемым объектом управления – архитекторов, администраторов, экспертов безопасности, т.п.

Общая схема контроля архитектуры системы на основе кода представлена на Рисунке 4.



***Рисунок 4. Общая схема*** *методологии контроля* ***на основе кода***

## Реализация методологии контроля архитектуры систем на основе кода

Для проверки описанной методологии был реализован прототип расширяемой программной платформы Governance as Code (GaC). Функциональная диаграмма платформы приведена на Рисунке 5. Как видно данная диаграмма полностью соответствует обобщенной схеме, изображенной на Рисунке 4.

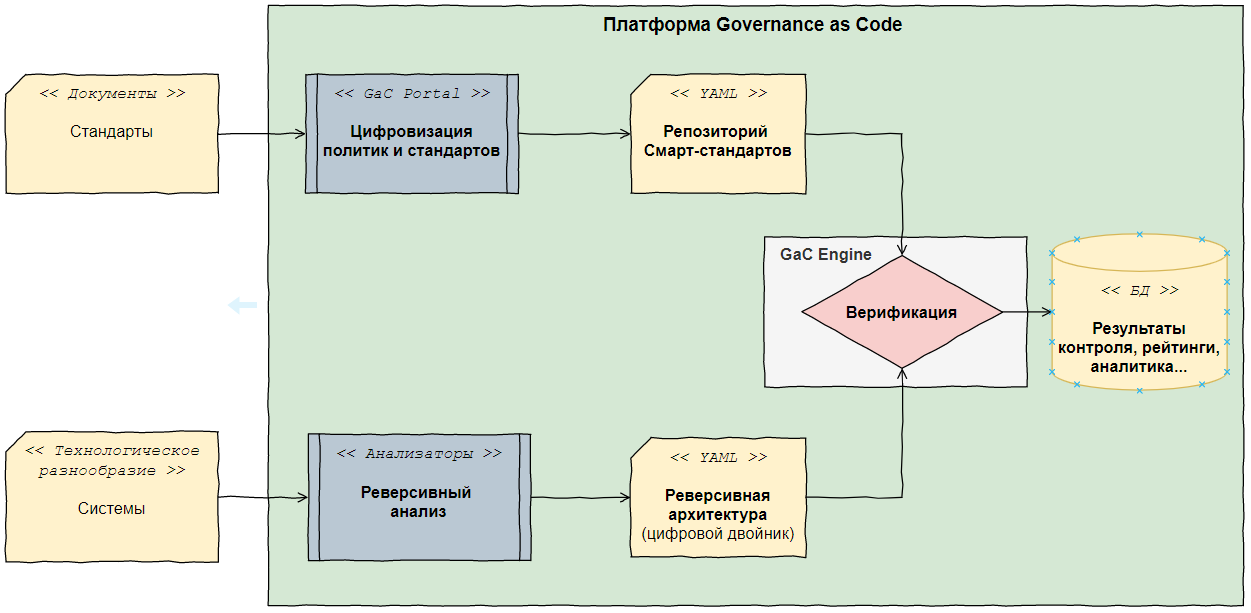
Выбранная для целей контроля предметная область – контроль архитектуры создаваемых информационных систем на соответствие требованиям стандартов – архитектурных, информационной безопасности и т.д.

В качестве Анализаторов использовались:

* Анализатор на основе статического анализа кода
* Экспертный анализ объекта управления.

Для целей верификации реверсивной архитектуры на основе смарт-стандартов был реализован специализированный модуль на основе программного компонента с открытым кодом Open Policy Agent (OPA) [6].

Для целей сквозной автоматизации процесса контроля разработки ИС, платформа Governance as Code была встроена в конвейер разработки программного обеспечения (DevOps-конвейер).



***Рисунок 5. Функциональная диаграмма платформы Governance as Code***

## Оценка эффекта от внедрения методологии

Предлагаемая методика контроля систем может позволить достигнуть следующих целей:

1. Минимизация ручных операций при проведении приемки и приемо-сдаточных испытаний. Снижение трудоемкости контрольных процедур при разработке/приемке систем.
2. Снижение зависимости проектов от «человеческого фактора» при создании/контроле/приемке систем - за счет четкой формализации требований. Объективная оценка результатов на основе строгих, заранее заданных критериев.
3. Снижение зависимости от квалификации персонала (как на этапах создания систем, так и на этапе контроля/приемки).
4. Повышение качества создаваемых систем.

Как следствие, перечисленные выше факторы обеспечивают:

1. Снижение общих рисков проектов
2. Снижение сроков проектов
3. Снижение общей стоимости проектов.

Приведем некоторые оценки по снижению общей стоимости проекта создания системы. Затраты на и обеспечение качества в организациях составляют от 2% до 20% и более от объема продаж (оборота) в машиностроении [7]. В индустрии разработки программного обеспечения затраты на контроль качества существенно выше: 15-30%. Затраты на контроль составляют 25% от общих затрат на качество [7], что с среднем составляет 2-6% от общего объема продаж (оборота). Сквозная автоматизация контрольных процедур может сократить не менее половины затрат на контроль. Таким образом, можно консервативно оценить эффект от внедрения методологии контроля на основе кода как 1-3% от общего объема продаж (оборота).

Отметим также экономический эффект для контролирующей стороны. В данном случае затраты на сам контроль сводятся к анализу результатов автоматизированного анализа. Можно ожидать, что затраты уменьшатся не менее чем в 10 раз.

Сроки проведения контрольных процедур за счет сквозной автоматизации могут быть сокращены достаточно существенно – не менее чем в 10 раз.

Отдельно хочется отметить эффект, получаемый за счет возможности проведения автоматизированных контрольных процедур на ранних стадиях разработки систем, когда стоимость исправления проблем наиболее низкая.

## Выводы

В статье предлагается универсальный подход к контролю систем на соответствие стандартам в автоматизированном виде. Методика позволяет расширить рамки применения идеи контроля «на основе кода» за пределы технологической инфраструктуры, а также за пределы информационных технологий.

Разработка стандартов на основе кода может вестись независимо от технологий, языков, процессов разработки, т.п. Требования стандартов могут быть переведены в цифровую форму – в формат программного кода, и предоставлять в виде библиотек автоматизированного контроля.

Перевод стандартов в формат программного кода позволит повысить эффективность контрольных процедур, снизить сроки, бюджеты, и риски проектов, а также понизить влияние человеческого фактора, включая требования к персоналу (квалификация персонала, количество высококвалифицированных специалистов и т.д.) Возможность встраивания библиотек контроля на ранних стадиях производственного цикла («сдвиг влево»), позволит дополнительно снизить общие трудозатраты по проекту.

## Ссылки

1. European Committee for Standardization. <https://www.cen.eu/work/endev/whatisen/pages/default.aspx>
2. Wittig, Andreas; Wittig, Michael (2016). Amazon Web Services in Action. Manning Press.
3. Guide to the Systems Engineering Body of Knowledge (SEBoK). <https://www.sebokwiki.org/wiki/Introduction_to_Systems_Engineering>
4. И. Б. Родионов, «Теория систем и системный анализ». <https://victor-safronov.ru/systems-analysis/lectures/rodionov.html>
5. Eilam, Eldad (2005). *Reversing: secrets of reverse engineering*. John Wiley & Sons. [ISBN](https://en.wikipedia.org/wiki/ISBN_(identifier)) [978-0-7645-7481-8](https://en.wikipedia.org/wiki/Special:BookSources/978-0-7645-7481-8).
6. YAML: YAML Ain't Markup Language, <https://yaml.org/>
7. Анализ затрат на качество. Материалы проекта[ISO 9000](http://www.iso9000.ru/)**.** <https://www.cfin.ru/management/iso9000/iso9000_cost.shtml>