УДК 004

Берестнев Петр Дмитриевич магистрант кафедры информатики и вычислительной техники, Самарский государственных технический университет, г. Самара

АРХИТЕКТУРА КАК КОНФИГУРАЦИЯ: КОНЦЕПТУАЛЬНАЯ МОДЕЛЬ И ГРАНИЦЫ ПРИМЕНИМОСТИ ПОДХОДА

Аннотация

В статье рассматривается подход «архитектура как конфигурация», развивающий практики архитектурного описания программных систем в условиях роста сложности и распространения генеративных инструментов разработки. Предложена концептуальная модель, в которой архитектурные артефакты трактуются как машиночитаемая конфигурация с явными соответствиями между уровнями абстракции. Раскрывается сквозной контур «описание — проверка — генерация — верификация», объединяющий этапы проектирования и реализации. Показано, что ключевой эффект подхода состоит в снижении неоднозначности интерпретации и повышении воспроизводимости инженерных решений при условии формальных правил связности и проверяемости. Сформулированы условия, при которых подход дает наибольший эффект, и направления его развития в контексте применения больших языковых моделей.

Annotation

The article discusses the "architecture as configuration" approach, which develops practices for architectural description of software systems in the context of increasing complexity and the spread of generative development tools. A conceptual model is proposed, in which architectural artifacts are interpreted as a machine-readable configuration with explicit mappings between abstraction levels. The article reveals the end-to-end "description-validation-generation-verification" loop, which combines the stages of design and implementation. It is shown that the key effect of

this approach is to reduce ambiguity in interpretation and increase the reproducibility of engineering solutions, provided that there are formal rules for consistency and verifiability. The conditions under which the approach is most effective and the directions for its development in the context of using large language models have been formulated.

Ключевые слова: архитектура программных систем, архитектурная конфигурация, прослеживаемость, формализация, генерация кода, большие языковые модели, валидация архитектуры, многоуровневое описание

Keywords: software system architecture, architectural configuration, traceability, formalization, code generation, large language models, architectural validation, multi-level description

Введение

Современная инженерия программного обеспечения характеризуется противоречием между темпом разработки и растущей сложностью целевых систем. С одной стороны, организации ожидают ускорения вывода функциональности на рынок; с другой стороны, архитектурные решения становятся многослойными и чувствительными к несогласованности между доменной логикой, интерфейсами, данными и инфраструктурой. На практике это проявляется в разрыве между архитектурными артефактами и кодовой реализацией: даже при наличии диаграмм и текстовых спецификаций команды по-разному трактуют исходные решения, а часть знаний остается неявной.

Стандарты и методы документирования архитектуры указывают, что архитектурное описание должно задаваться как система представлений и точек зрения, связанных между собой соответствиями [6]. При этом в индустриальных процессах часто доминируют либо коммуникационные артефакты, либо локально формальные артефакты, но не целостная модель, которая одновременно понятна участникам и пригодна для инструментальной обработки [4], [10].

На этом фоне возникает концепт «архитектура как конфигурация»: архитектурное описание трактуется не как статический документ, а как структурированный, версионируемый и проверяемый объект, используемый для автоматизированных преобразований в производные артефакты (контракты, шаблоны реализации, тестовые каркасы, инфраструктурные конфигурации). Актуальность такого перехода усиливается распространением генеративных ИИ-инструментов, которые без формализованного контекста ускоряют локальную разработку, но не гарантируют межуровневую согласованность.

Цель статьи — предложить концептуальную модель подхода «архитектура как конфигурация», определить его архитектурные и организационные предпосылки, а также очертить границы применимости в реальных проектах.

Литературный обзор

В классических работах по программной архитектуре подчеркивается необходимость разделения архитектурных представлений и согласования интересов разных стейкхолдеров [2], [6]. Подход views-and-beyond рассматривает архитектурную документацию как совокупность связанных представлений, где отсутствие соответствий между ними приводит к потере целостности и повышению риска ошибочных решений [4].

Многоуровневые визуальные подходы обеспечивают практичную декомпозицию «от контекста к компонентам» и улучшают коммуникацию внутри команды [3]. Однако в большинстве случаев эти артефакты остаются слабо формализованными для машинной проверки и генерации. В то же время стандартизованные контрактные форматы задают строгую структуру API и служат опорой для инструментального контура (генерация, тестирование, документация), но сами по себе не охватывают весь архитектурный контекст [10].

Подходы DDD и связанные практики решают задачу выделения доменных границ и языка предметной области, что критически важно для устойчивой архитектуры [5], [12]. При этом перенос доменных решений в технические представления часто выполняется вручную, и именно здесь возникает значительная доля неоднозначности.

Идея дисциплины ограничений и управляемости зависимостей в архитектуре раскрыта в работах по чистой архитектуре и фундаментальным инженерным подходам [7], [8]. Эти источники подтверждают тезис, что рост качества архитектурных решений достигается не увеличением свободы проектирования, а введением устойчивых правил композиции и границ ответственности.

Таким образом, в литературе представлены компоненты будущего решения: многоуровневость, доменная декомпозиция, контрактность, инструментальная обработка. Недостаточно проработанной остается интеграция этих компонентов в единый объект «архитектура как конфигурация» с формальным контуром проверяемости.

Материалы и методы

Исследование выполнено как концептуально-методологическое и включает три группы методов.

1. Нормативно-аналитический анализ источников: стандарты архитектурного описания, методические работы по документированию архитектуры, практики контрактного проектирования, DDD-подходы [1], [2], [4], [6], [10].

2. Сравнительный анализ инженерных подходов по набору критериев: уровень формализации, поддержка прослеживаемости, пригодность к автоматизированным преобразованиям, стоимость сопровождения.

3. Концептуальное моделирование целевого контура «архитектура как конфигурация», включая описание ролей, артефактов, переходов между уровнями и контрольных точек валидации.

Для сопоставления подходов использованы следующие критерии:

- тип исходного контекста (текстовый, визуальный, контрактный, конфигурационный);
- степень машинной интерпретируемости;
- устойчивость к неоднозначности;
- пригодность к сквозной генерации;
- требования к зрелости процессов.

Сравнительная таблица подходов

Подход	**Машинная интерпретируемость**	**Прослеживаемость между уровнями**	**Пригодность к генерации**	**Риск неоднозначности**
Разрозненные тексты и ad-hoc	Низкая	Низкая	Низкая	Высокий
Многоуровневые визуальные модели	Средняя	Средняя	Средняя	Средний
Контрактный подход	Высокая локально	Средняя	Высокая локально	Средний
Архитектура как конфигурация	Высокая	Высокая	Высокая сквозная	Низкий

Таблица 1. Сопоставление подходов по критериям формализации, прослеживаемости и генерации.

Результаты

1. Концептуальная модель «архитектура как конфигурация»

Предложенная модель включает четыре взаимосвязанных слоя:

1. Смысловой слой: доменные термины, границы контекстов, ключевые сценарии.

2. Структурный слой: системный контекст, контейнеры, компоненты и их зависимости.

3. Контрактный слой: интерфейсы, схемы данных, ограничения совместимости.

4. Исполнимый слой: шаблоны/каркасы кода, инфраструктурные артефакты, тестовые заготовки.

Связи между слоями задаются как явные соответствия, которые позволяют:

- проверять полноту и непротиворечивость модели;
- локализовать влияние изменений;
- выполнять управляемые преобразования в производные артефакты.

2. Сквозной контур жизненного цикла

Выделен контур, в котором архитектурная конфигурация выступает центральным объектом:

уточнение домена, структуризация, формализация контрактов, автоматические проверки, генерация артефактов, обратная верификация и обновление конфигурации.

В этом контуре генерация не рассматривается как финальный «магический» этап, а как итеративное преобразование, ограниченное проверяемой моделью.

3. Условия результативности подхода

Показано, что наибольший практический эффект достигается при сочетании трех условий:

- наличие формальных критериев согласованности;
- модульность конфигурации по ограниченным контекстам;
- организационная дисциплина актуализации архитектурного объекта при изменениях системы.

Обсуждение

Предложенная модель подтверждает, что «архитектура как конфигурация» является не заменой архитектурного мышления, а механизмом его операционализации. По сути, подход переводит архитектурные решения в форму, пригодную для повторяемого применения и проверки.

С теоретической точки зрения это соответствует развитию архитектурного описания от коммуникационных артефактов к инженерным объектам с заданной семантикой [4], [6]. С практической точки зрения подход снижает транзакционные издержки в распределенных командах: часть согласований переносится из устной коммуникации в проверяемую модель.

Одновременно выявлены ограничения:

1. Риск ложной формализации: формально заполненные артефакты могут не отражать фактические решения команды.

2. Стоимость сопровождения: без автоматизации проверок и процедур обновления модель быстро устаревает.

3. Порог внедрения: необходима зрелость процессов и единая политика архитектурной дисциплины.

4. Инструментальная неоднородность: интеграция разнородных инструментов (модели, контракты, генераторы, CI) требует дополнительных инженерных решений.

В контексте LLM подход выглядит особенно значимым. Без конфигурационного архитектурного ядра LLM-инструменты в основном ускоряют локальную разработку. При наличии конфигурации и правил проверяемости они могут использоваться как исполнители в управляемом контуре преобразований.

Заключение

В статье предложена и обоснована концептуальная модель подхода «архитектура как конфигурация». Показано, что переход к конфигурационному представлению архитектуры позволяет объединить многоуровневое описание, контрактный подход и инструментальные проверки в единый воспроизводимый инженерный контур.

Основные результаты:

1. Сформулирована структура модели и роль межуровневых соответствий.

2. Выделен сквозной контур жизненного цикла архитектурной конфигурации.

3. Определены условия эффективности и границы применимости подхода.

4. Показано место подхода в контексте использования LLM-инструментов в инженерии ПО.

Перспективы дальнейших исследований связаны с формализацией метрик качества архитектурной конфигурации, эмпирической оценкой эффекта на реальных проектах и разработкой типовых профилей внедрения для организаций с различной зрелостью процессов.

Список литературы

1. Ambler S. W. Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process. Wiley, 2002.

2. Bass L., Clements P., Kazman R. Software Architecture in Practice. 4th ed. Addison-Wesley, 2021.

3. Brown S. The C4 model for visualising software architecture. URL: https://c4model.com (дата обращения: [указать]).

4. Clements P., Bachmann F., Bass L. et al. Documenting Software Architectures: Views and Beyond. 2nd ed. Addison-Wesley, 2010.

5. Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison-Wesley, 2003.

6. ISO/IEC/IEEE 42010:2011. Systems and software engineering — Architecture description.

7. Martin R. C. Clean Architecture: A Craftsman’s Guide to Software Structure and Design. PrenticeHall, 2017.

8. Richards M., Ford N. Fundamentals of Software Architecture: An Engineering Approach. O’Reilly, 2020.

9. OMG. Unified Modeling Language (UML), Version 2.5.1. OMG formal/2017-12-05.

10. OpenAPI Initiative. OpenAPI Specification, Version 3.1.0. URL: https://spec.openapis.org/oas/v3.1.0 (дата обращения: [указать]).

11. Patton J., Economy P. User Story Mapping: Discover the Whole Story, Build the Right Product. O'Reilly, 2014.

12. Vernon V. Implementing Domain-Driven Design. Addison-Wesley, 2013.

References

1. Ambler S. W. Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process. Wiley, 2002.

2. Bass L., Clements P., Kazman R. Software Architecture in Practice. 4th ed. Addison-Wesley, 2021.

3. Brown S. The C4 model for visualising software architecture. URL: https://c4model.com (дата обращения: [указать]).

4. Clements P., Bachmann F., Bass L. et al. Documenting Software Architectures: Views and Beyond. 2nd ed. Addison-Wesley, 2010.

5. Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison-Wesley, 2003.

6. ISO/IEC/IEEE 42010:2011. Systems and software engineering — Architecture description.

7. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. PrenticeHall, 2017.

8. Richards M., Ford N. Fundamentals of Software Architecture: An Engineering Approach. O'Reilly, 2020.

9. OMG. Unified Modeling Language (UML), Version 2.5.1. OMG formal/2017-12-05.

10. OpenAPI Initiative. OpenAPI Specification, Version 3.1.0. URL: https://spec.openapis.org/oas/v3.1.0 (дата обращения: [указать]).

11. Patton J., Economy P. User Story Mapping: Discover the Whole Story, Build the Right Product. O'Reilly, 2014.

12. Vernon V. Implementing Domain-Driven Design. Addison-Wesley, 2013.