

СОВМЕСТНОЕ ИСПОЛЬЗОВАНИЕ ШАБЛОНОВ АРХИТЕКТУРЫ И ОБОСНОВАНИЯ БЕЗОПАСНОСТИ В РАЗРАБОТКЕ АВТОМОБИЛЬНЫХ СИСТЕМ ПО СТАНДАРТАМ ФУНКЦИОНАЛЬНОЙ БЕЗОПАСНОСТИ

Королев А.С., Тихомиров М.А., Кировский О.М., Кодабашян Л.С.

Российский технологический университет – МИРЭА, Москва, Россия

korolev@mirea.ru, maxim.tikhomirov@safetyconsult.tech, kirovskij@mirea.ru, kodabashyan@mirea.ru

Аннотация. В статье рассмотрены подходы к совместному использованию шаблонов архитектуры и обоснования безопасности в разработке автомобильных систем. Исследуются процессы разработки, относящиеся к «гибким» моделям жизненного цикла преимущественно программных систем, когда при неполных требованиях создаются первые версии программного обеспечения, которые валидируются с заказчиком и дорабатываются в итеративной манере. Архитектура аппаратной части при этом слабо связана с архитектурой программного обеспечения, благодаря использованию универсальных вычислительных платформ.

Ключевые слова: высокоавтоматизированные транспортные средства, архитектура, функциональная безопасность, обоснование безопасности, модели-ориентированная системная инженерия, жизненный цикл.

Введение

Современные автомобили имеют в своем составе большое количество систем, которые можно считать преимущественно программными системами, т.е. системами, функциональность которых значительным образом определяется программным обеспечением (ПО). Некоторые исследователи называют процесс, когда ПО оказывает ведущее влияние на поведение систем, процессом софтверизации (softwarization) [1, 2]. С точки зрения системной инженерии (СИ) это означает, что при разработке таких систем функциональные требования в первую очередь предъявляются к ПО. В качестве аппаратной части таких систем чаще всего используются универсальные вычислительные платформы, в отличие от «классического» подхода, при котором аппаратная часть разрабатывается для конкретного электронного блока управления (ЭБУ) параллельно с программной. Это позволяет снизить зависимость между разработкой аппаратной и программной частей. Если же имеющиеся универсальные платформы не поддерживают функциональные требования, назначенные на систему, дополнительные требования к аппаратной части определяются на более позднем этапе, когда архитектура ПО уже создана, а большинство функциональных требований к ПО зафиксированы.

В начале XXI века приобрели большую популярность «гибкие» методы разработки ПО, и в 2001 году был опубликован Манифест гибкой разработки программного обеспечения [3]. Один из его принципов заключается в том, что «работающий продукт важнее исчерпывающей документации». Это меняет подход к организации разработки ПО. Если раньше важной задачей разработки были сбор и документирование требований, а затем – документирование того, что удалось реализовать все требования, принятые в разработку, то теперь предварительная документация требований отброшена, а место верификации заняла проводимая в итеративном режиме валидация с участием заказчика.

«Рабочий продукт», получаемый в результате применения гибких методологий, удовлетворяет потребностям заказчика. Однако, поскольку пропущен этап сбора полной спецификации требований, продукт может обладать поведением, неизвестным ни заказчику, ни разработчику – поведением, которое проявляется в определенных случаях, не выявленных во время валидации. Именно поэтому тенденции перехода к гибким методам разработки в первое время не сильно затронули автомобильную промышленность. Инциденты, которые могут произойти вследствие нерегламентированного поведения автомобильных систем, заставляли уделять больше внимания не времени разработки ПО, а обеспечению его безопасности, что подразумевало трудоемкую работу с требованиями [4].

В 2011 году было опубликовано первое издание стандарта функциональной безопасности ИСО 26262 [5], в котором определены лучшие практики обеспечения функциональной безопасности автомобильных систем. Часть 6 стандарта посвящена разработке безопасного программного обеспечения. Стандарт ничего не говорит о методах разработки, которые должны применяться. Однако следует отметить, что в основе стандарта лежит V-модель жизненного цикла (ЖЦ), подразумевающая сбор полной спецификации требований и последующие несколько стадий верификации перед непосредственно валидацией.

В 2018 году было опубликовано второе издание стандарта ИСО 26262. Несмотря на прошедшие 10 лет, в основе стандарта лежит все та же V-модель ЖЦ. Согласно этому стандарту, после завершения разработки инженеры должны доказать, что созданные системы не только выполняют свои функции,

но и гарантируют безопасность. Таким образом, отказ от сбора требований (в частности, требований безопасности) и верификации в пользу валидации с заказчиком невозможен [6].

Не следует думать, что автомобильная отрасль осталась цитаделью «старого порядка» и методы гибкой разработки там совершенно не применяются. Все ведущие автомобильные компании применяют методы гибкой разработки, учитывая требования стандартов безопасности [7, 8]. Таким образом, сбор требований безопасности и их верификация стали, как и разработка, итеративными. Теперь, когда преимущественно программная система получает дополнительную функциональность, команда разработчиков ПО должна работать над двумя типами задач одновременно: задачами разработки ПО и аналитическими задачами, которые раньше решались до начала разработки ПО (например, задачи по анализу безопасности и управлению требованиями). В этих условиях прямое и полное применение процессов, описанных в стандартах безопасности, проблематично. При этом применение гибких подходов в контексте, где безопасность критически важна, до сих пор представляет собой проблему, которая не до конца решена на практике [9].

Несмотря на эти противоречия, в настоящее время большинство сертификаций автомобильного ПО подразумевают проверку соответствия процесса разработки требованиям стандартов безопасности. Поскольку, как говорилось выше, прямое применение стандартизированных процессов затруднено, разработчикам процессов и сотрудникам органов по сертификации приходится отступать от требований стандартов и приходиться к соглашению о том, что допустимо, а что нет. Растет роль человеческого фактора.

Сейчас исследователями продвигается сертификация, основанная на полуформальном (формализованном) обосновании безопасности ([10, 11, 12, 13]). Переход к такому типу сертификации дал бы разработчикам автомобильных систем свободу в выборе процессов разработки, позволив использовать более экономные и быстрые гибкие методы, без снижения значимости требований к функциональной безопасности.

1. Постановка проблемы

Рассмотрим проект, выполняемый четырьмя командами: командой системной архитектуры (СА), командой функциональной безопасности (ФБ) и двумя командами по разработке компонентов (РК1 и РК2). На рисунке 1 (а, б, в, г) представлены четыре возможные проектные ситуации. Верхний ряд представляет «классический» процесс с V-моделью ЖЦ: сначала создается архитектура, затем она проверяется командой ФБ, а затем требования передаются командам компонентов. Нижний ряд показывает разработку по гибкой методологии, в котором анализ безопасности и разработка компонентов выполняются параллельно, чтобы обеспечить создание ценности в течение одной итерации. В левом столбце представлены случаи, когда архитектура верна с первого раза, а в правом – есть одна итерация по устранению недостатков.

Красные прямоугольники в правом столбце иллюстрируют потерянный потенциал команд разработки, если архитектура не прошла верификацию командой ФБ. Хотя рисунок 1 и не имеет масштаба, т.е. строгого соответствия единиц измерения на рисунке (мм.) единицам измерения времени (час.) в реальной деятельности, но можно утверждать, что процесс разработки по гибкой методологии более эффективен в случае создания правильной архитектуры с первого раза, в противном же случае несет большие риски. Цель данной статьи – показать процесс, который минимизирует риск неудачи при разработке по гибкой методологии с минимальными накладными расходами.

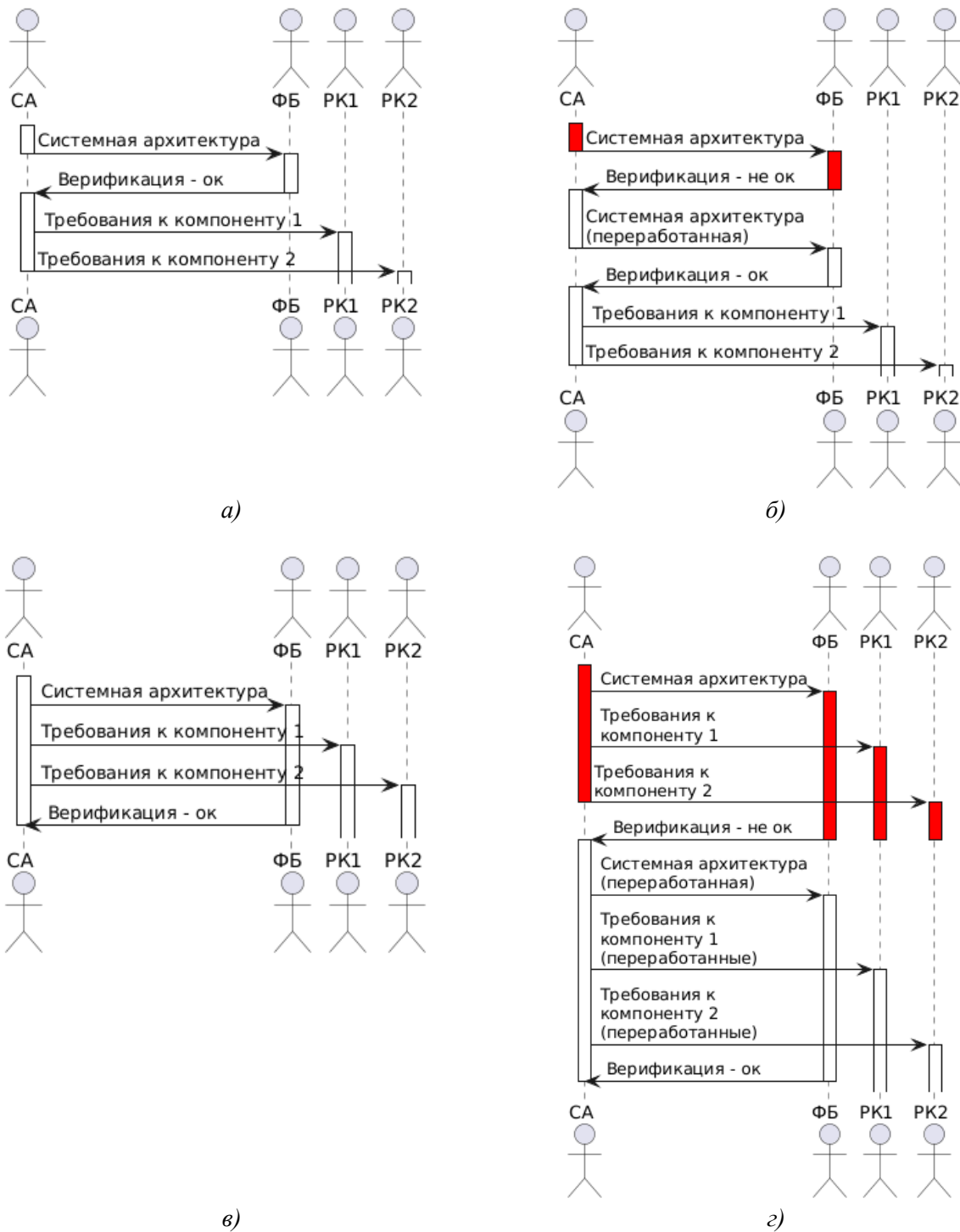


Рис. 1. (а, б, в, г). Рабочие процессы системного проектирования: а) – последовательная методология разработки (V-модель ЖЦ), архитектура удовлетворяет требованиям безопасности; б) – последовательная методология разработки (V-модель ЖЦ), архитектура не удовлетворяет требованиям безопасности; в) – гибкая методология разработки, архитектура удовлетворяет требованиям безопасности; г) – гибкая методология разработки, архитектура не удовлетворяет требованиям безопасности

2. Шаблоны

2.1. Шаблоны архитектуры ПО

Шаблон архитектуры ПО – это общее решение часто встречающейся проблемы в определенном контексте, которое можно использовать повторно [14]. Шаблоны содержат абстракции (например, абстрактный компонент, или абстрактное, т.е. не привязанное к конкретной системе, утверждение). На основе шаблона можно создать экземпляр, то есть заменить абстракции, используемые в шаблонах, определенными элементами (функциями, целями безопасности и т. д.), относящимися к разрабатываемому продукту. Перед созданием экземпляра некоторые шаблоны должны быть расширены в соответствии с правилами, заложенными в описание шаблона. Шаблоны архитектуры также могут содержать требования, предъявляемые к элементам.

Наиболее известные архитектурные шаблоны, используемые в контексте безопасности, показаны на рисунке 2 (а, б, в). Это, соответственно, резервирование, голосование n-из-m и восстановление предыдущего значения.



Рис. 2. (а, б, в). Архитектурные шаблоны: а) – резервирование, б) – голосование n-из-m, в) – восстановление прежнего значения

2.2. Шаблоны обоснования безопасности

Обоснование безопасности – это структурированный набор аргументов, подкрепленный свидетельствами, который обеспечивает убедительное, понятное и обоснованное доказательство того, что система безопасна для данного применения в определенном контексте [15]. Мы будем использовать нотацию, структурированную по целям (англ. Goal-Structured Notation, GSN) [16], созданную Safety-Critical Systems Club (SCSC) для графического представления обоснований безопасности. Диаграмма GSN состоит из элементов и взаимосвязей, описание которых представлено в таблице 1.

Таблица 1. Элементы нотации, структурированной по целям (GSN)

Буквенный код	Графическое изображение	Название	Определение
G		Цель	Утверждение, формирующее часть доказательства
C		Контекст	Артефакт, описывающий контекст: утверждение или ссылка на источник информации
J		Разъяснение	Объяснение правомерности принятого решения
St		Стратегия	Логическая связь между целью и поддерживающими ее целями или решениями
Sn		Свидетельство, или решение	Ссылка на артефакт, содержащий доказательство достижения цели
-		Поддерживается	Отношение «поддерживается» между элементами
-		В контексте	Отношение между целью/стратегией и предположением/разъяснением
-		-	Элемент шаблона требует создание экземпляра
-		-	Элемент шаблона требует расширение
-		-	Элемент шаблона требует создание экземпляра и расширение
-		-	В результате создания экземпляра может получиться от 0 до n элементов

3. Координированное использование шаблонов (КИШ)

3.1. Координация шаблонов

Цель данной статьи – показать процесс, который минимизирует риск неудачи при разработке по гибкой методологии с минимальными накладными расходами. Для достижения этой цели мы создаем каталог, в котором каждый шаблон архитектуры связан с одним шаблоном обоснования безопасности. В процессе создания архитектуры мы выбираем пары шаблонов, расширяем их и создаем экземпляры. Затем нужно определить, подходит ли нам полученное техническое решение. Для этого мы проверяем его на соответствие инвариантам. Если полученные экземпляры не нарушают инвариантов, то можно обновить архитектуру системы и обоснование безопасности, добавив в них соответствующие экземпляры как части. Инварианты – это ограничения, накладываемые на архитектуру и обоснование безопасности внешними факторами или требованиями более высокого порядка, например, концепцией функциональной безопасности.

С точки зрения порядка разработки автомобильных систем, описанная здесь процедура применяется на этапе создания архитектуры системы. Согласно стандарту ИСО 26262, концепция функциональной безопасности разрабатывается на предшествующей фазе ЖЦ, а значит, данные концепции ФБ будут доступны на этапе создания архитектуры. Кроме того, источником инвариантов могут быть

ограничения, связанные с предприятием, например, невозможность разрабатывать ПО с уровнем полноты безопасности системы (УПБТС, англ. Automotive Safety Integrity Level, ASIL) выше В.

Алгоритм, предложенный в данной работе, показан на рисунке 3.

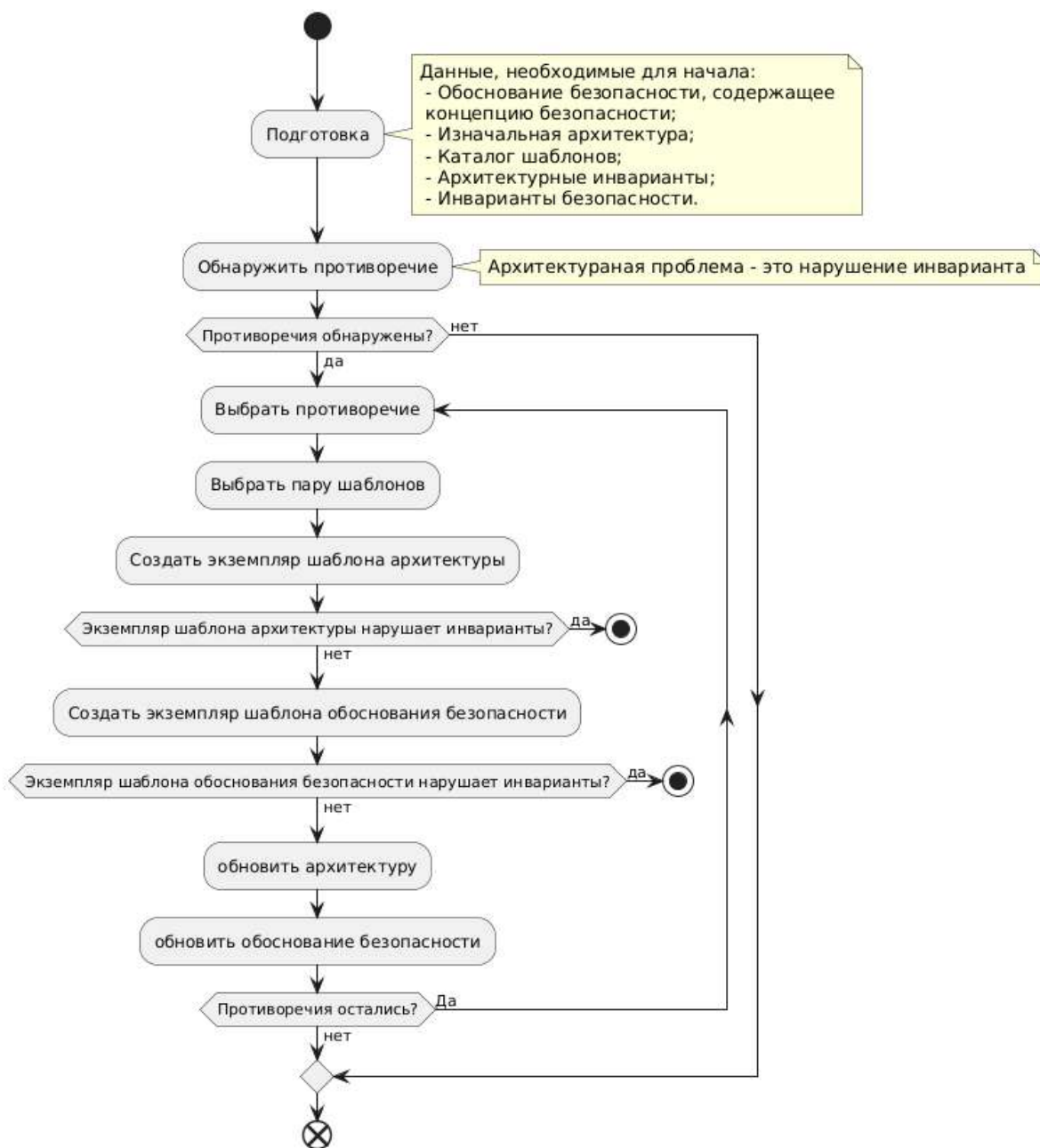


Рис. 3. Алгоритм координированного использования шаблонов (КИШ)

Сначала выявляются архитектурные проблемы, то есть инварианты, которым не удовлетворяет исходная архитектура. Если их нет, мы можем использовать предварительную архитектуру для дальнейшей декомпозиции требований, и нет необходимости применять алгоритм КИШ. В противном случае мы выявляем архитектурные проблемы, которым не удовлетворяет текущая архитектура. Для решения проблемы мы применяем пару шаблонов. В каталоге пары шаблонов могут быть связаны с проблемами, которые они решают, например, пара шаблонов «резервирование» может быть связана с проблемой «слишком высокий УПБТС компонентов ПО». Затем мы создаем экземпляры обоих шаблонов из пары. Если ни один из полученных экземпляров не противоречит инвариантам, мы обновляем архитектуру и обоснование безопасности с помощью полученных экземпляров шаблонов. В противном случае эта пара не подходит, и нужно выбрать другую.

Алгоритм выполняется до тех пор, пока не останется никаких архитектурных проблем.

4. Пример применения КИШ

4.1. Пример и предварительная структура: Система LKAS

Система удержания в полосе (англ. Lane Keeping Assist System, LKAS) – это система ADAS, которая устанавливается на многие серийные автомобили. Очень простой вариант архитектуры LKAS представлен на рисунке 4.



Рис. 4. Архитектура LKAS (предварительный вариант)

Цель безопасности, определенная на предыдущем этапе анализа, – «LKAS должна предотвращать непреднамеренное руление (ASIL D)». Безопасное состояние – «отсутствие непреднамеренных изменений угла поворота руля на выходе LKAS». Пример безопасности в формате GSN показан на рисунке 5.



Рис. 5. Обоснование безопасности LKAS в GSN (предварительный вариант)

4.2. Применение метода КИШ

Инвариант. Мы предполагаем, что команда может писать ПО только в соответствии с уровнями ASIL A или ASIL B, не выше. Архитектура, соответствующая представленному на рисунке 5 обоснованию безопасности, нарушает этот инвариант.

Выбор пар шаблонов. Теперь нам нужно выбрать пары шаблонов. Наша цель – уменьшить ASIL на компонентах ПО, разрабатываемых командой. Сначала мы применим пару шаблонов «резервирование». Шаблон безопасности для этой пары представлен на рисунке 6, результат создания его экземпляра на рисунке 7.

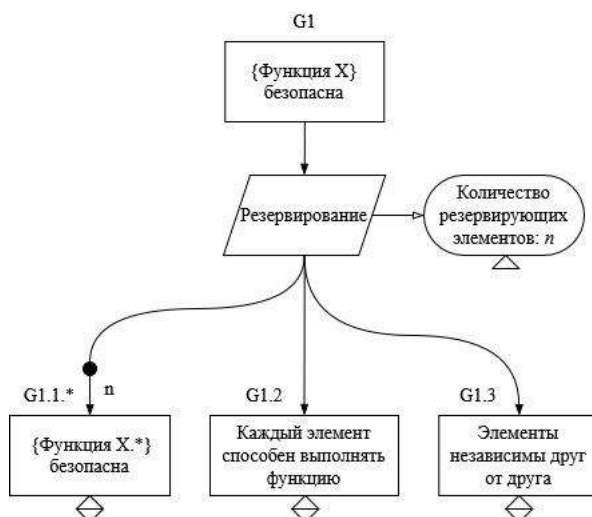


Рис. 6. Шаблон обоснования безопасности «резервирование»

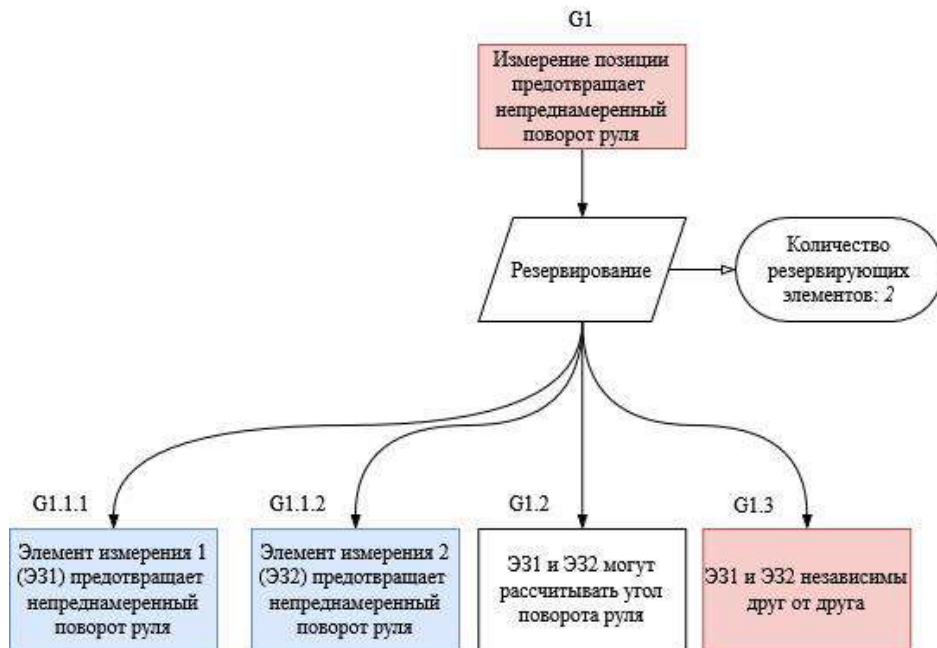


Рис. 7. Экземпляр шаблона обоснования безопасности «резервирование»

Однако применение одной пары шаблонов не решает проблему второго утверждения из рис. 5. Применим шаблон «восстановление предыдущего значения», для которого шаблон обоснования безопасности показан на рисунке 8, а экземпляр шаблона на рисунке 9. Окончательный вариант архитектуры системы представлен на рисунке 10. Эта архитектура связана с обоснованием безопасности (рисунок 11), которое гарантированно выполняет требования безопасности и инвариант «ни один компонент ПО не имеет ASIL выше В».

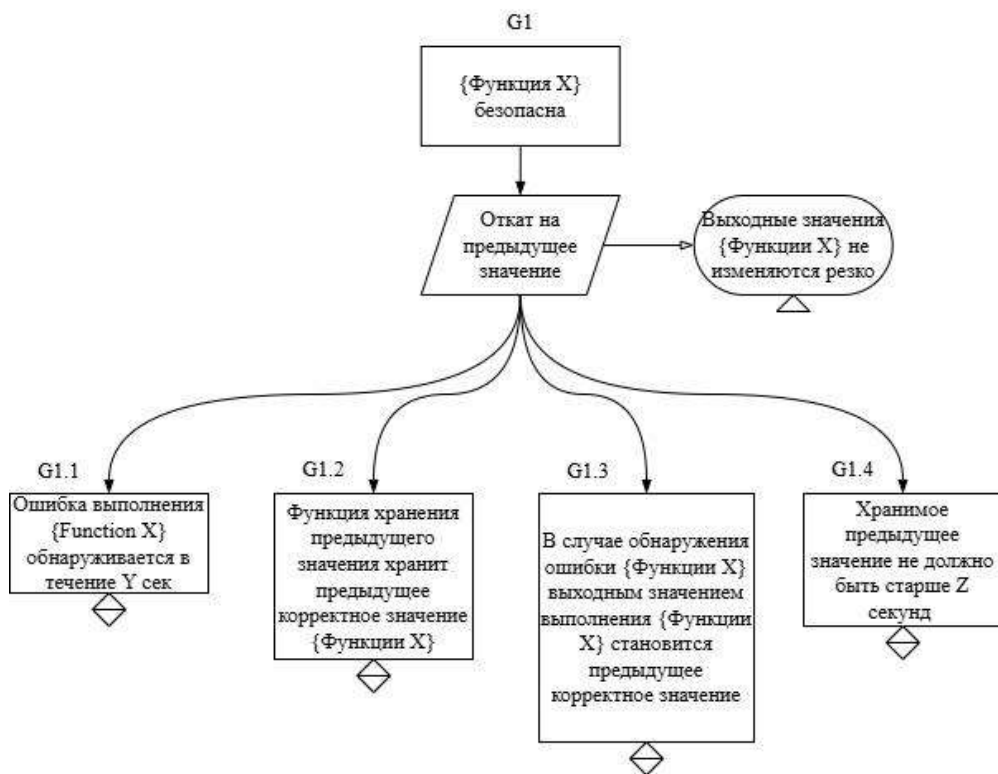


Рис. 8. Шаблон обоснования безопасности «восстановление предыдущего значения»

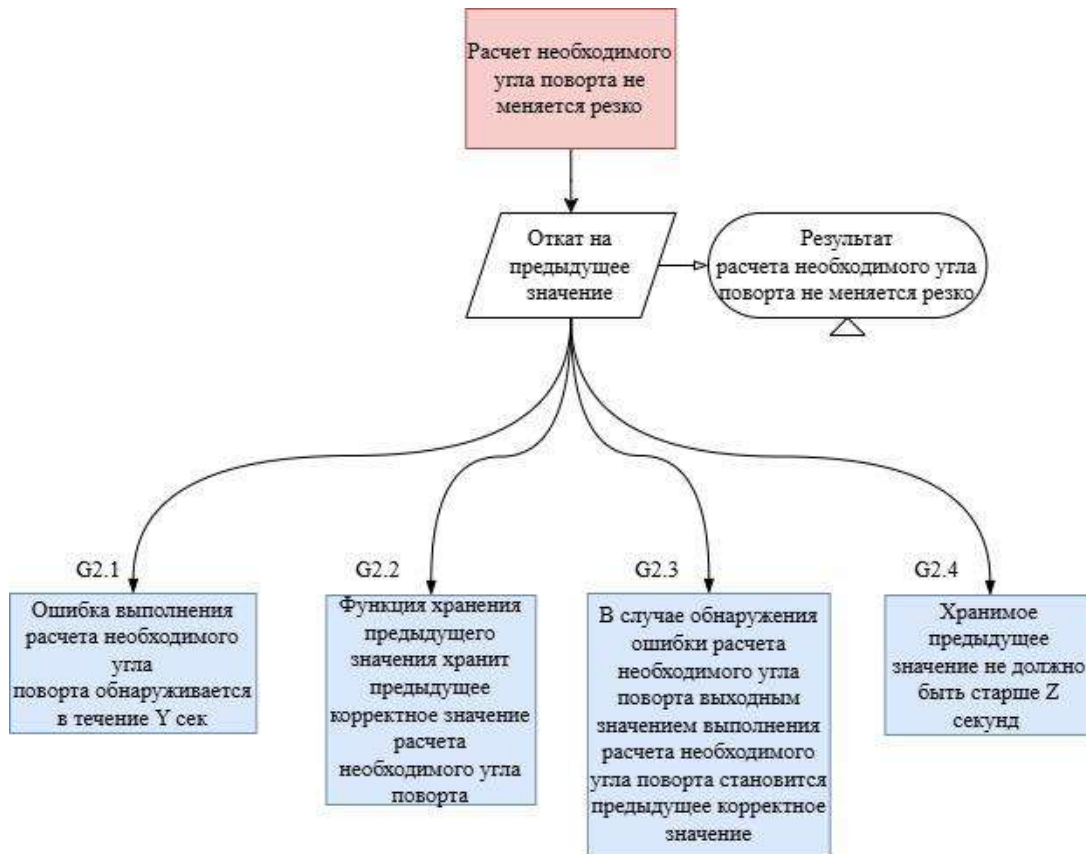


Рис. 9. Экземпляр шаблона обоснования безопасности «восстановление предыдущего значения»

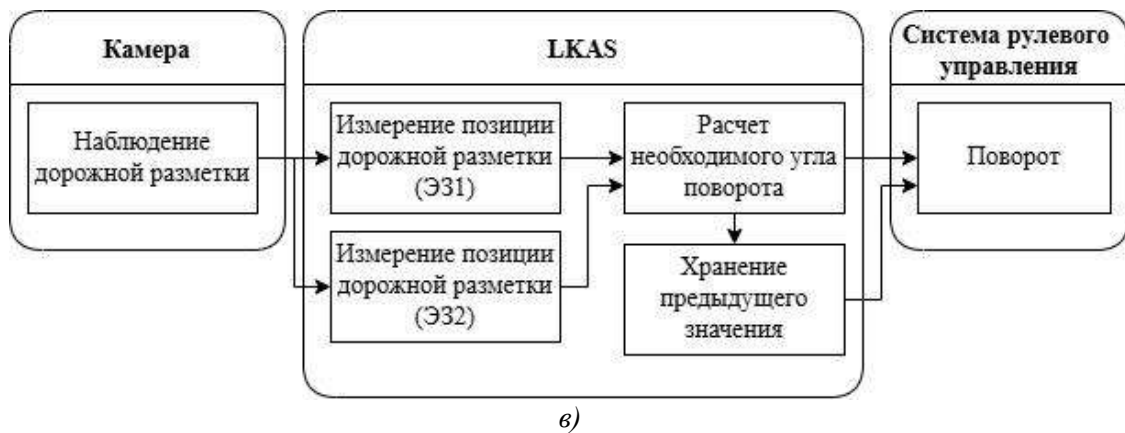


Рис. 10. Архитектура системы LKAS (окончательный вариант)

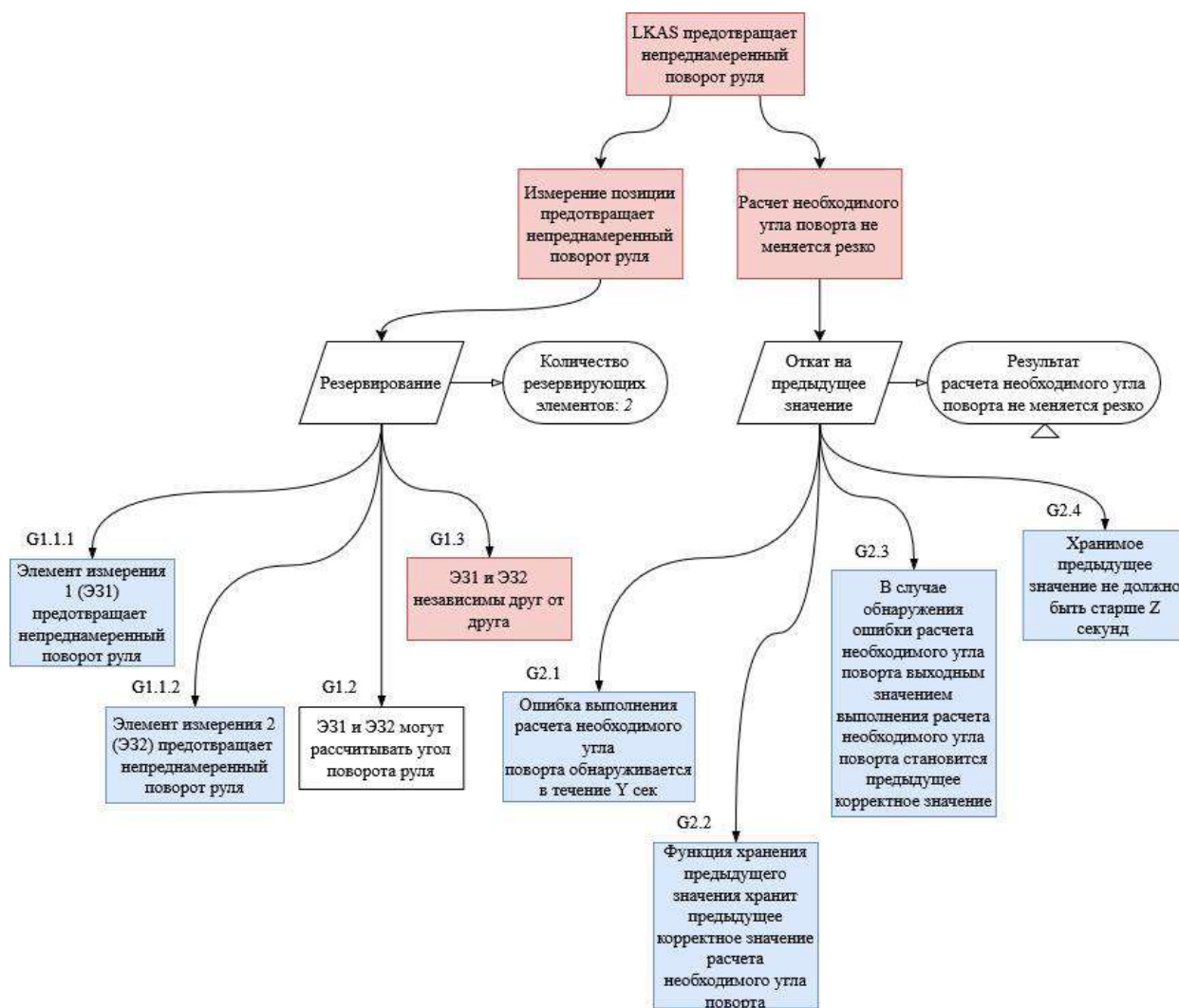


Рис. 11. Окончательный вариант обоснования безопасности

5. Заключение

В данной статье представлен метод проектирования архитектуры систем с учетом требований безопасности с помощью координированного использования шаблонов (КИШ). Он демонстрирует подход, при котором использование архитектурных шаблонов и шаблонов обоснования безопасности позволяет получать архитектуру и обоснование безопасности системы параллельно, без промежуточного шага в виде анализа безопасности. Это снижает риски ошибок и потенциально позволит сэкономить усилия и снизить риск неудачи при «гибкой» разработке автомобильных систем.

Следующими направлениями работы, которые мы хотим рассмотреть в ходе развития метода, являются создание и публикация содержательного каталога пар шаблонов с соответствующей систематизацией, а также применение метода для решения других задач.

Литература

1. Журавлева Е.Ю. Софтверизация общества: истоки и перспективы // Социологические исследования. –2019. – № 4. – С. 109–117.
2. Haerberle M. et al. Softwarization of Automotive E/E Architectures: A Software-Defined Networking Approach // IEEE Vehicular Networking Conference (VNC), New York, NY, USA, 2020.
3. Kent Beck et al. Manifesto for Agile Software Development. – 2001. <https://agilemanifesto.org/> (Дата обращения 06.04.2025).
4. Wolf M. Embedded Software in Crisis// Computer, 2016 – Vol. 39, № 1. – P. 88–90.
5. ISO 26262 – Road Vehicles. Functional Safety, 2018.
6. Koopman P. Practical Experience Report: Automotive Safety Practices vs. Accepted Principles // Computer Safety, Reliability, and Security. SAFECOMP, Vasteras, 2018.

7. *Brian Katumba, Eric Knauss*. Agile Development in Automotive Software Development: Challenges and Opportunities // 15th International Conference Proceedings, PROFES 2014, Helsinki, Finland, December 10–12, 2014.
8. *Philipp Diebold, Udo Mayer*. On the Usage and Benefits of Agile Methods & Practices – A Case Study at Bosch Chassis Systems Control // International Conference on Agile Software Development, 2017.
9. *Heeager L.T. and Nielsen P.A.* A conceptual model of agile software development in a safety-critical context: A systematic literature review // Information and Software Technology. – 2018. – Vol. 103. – P. 22–39.
10. *Yu H., Lin C. and Kim B.* Automotive Software Certification: Current Status and Challenges // SAE Int. J. Passeng. Cars – Electron. Electr. Syst. – 2016. – Vol. 9, № 1. P. 74–80.
11. *Leveson N.* The Use of Safety Cases in Certification and Regulation, MIT, 2011.
12. *Alexander R., Kelly T., Kurd Z. and McDermind J.* Safety Cases for Advanced Control Software: Safety Case Patterns. – Department of Computer Science, University of York, York, 2007.
13. *Kirovskii O., Korolev A.* Joint Assessment of Automotive Systems Safety and Security Using Bayesian Networks // Proceedings of the Eighth International Scientific Conference Intelligent Information Technologies for Industry (IITI'24), Volume 1, – Springer, 2024. – P. 454–465.
14. *Taylor R.N., Medvidovic N. and Dashofy E.M.* Software Architecture. Foundations, Theory, and Practice, John Wiley & Sons, 2010.
15. UK DEFSTAN 00-56 Safety Management Requirements for Defence Systems, 2007.
16. Safety-Critical Systems Club, Goal Structured Notation, May 2021. [Online]. Available: <https://scsc.uk/gsn>. (Дата обращения 06.04.2025).