ТЕХНИЧЕСКИЙ ДОЛГ В ПРОГРАММНЫХ СИСТЕМАХ: ПРИЧИНЫ НАКОПЛЕНИЯ, ПОСЛЕДСТВИЯ И ПОДХОДЫ К УПРАВЛЕНИЮ

Захаров А.С.

Захаров Андрей Сергеевич – магистрант,
кафедра бизнес-информатики и менеджмента,
Санкт-Петербургский государственный университет аэрокосмического приборостроения,
г. Санкт-Петербург

Аннотация: *в статье рассматривается технический долг как фактор, влияющий на развитие и сопровождение программных систем. Анализируются основные причины его накопления, возможные последствия для качества программного обеспечения и скорости разработки. Систематизируются подходы к выявлению, оценке и управлению техническим долгом с учётом особенностей программного продукта и ограниченности ресурсов команды.*
Ключевые слова: *технический долг, программное обеспечение, разработка программных систем, качество кода, программная архитектура, рефакторинг, сопровождение программного обеспечения.*

TECHNICAL DEBT IN SOFTWARE SYSTEMS: CAUSES OF ACCUMULATION, CONSEQUENCES AND MANAGEMENT APPROACHES

Zakharov A.S.

Zakharov Andrey Sergeevich – Master's student,
DEPARTMENT OF BUSINESS INFORMATICS AND MANAGEMENT,
SAINT PETERSBURG STATE UNIVERSITY OF AEROSPACE INSTRUMENTATION,
SAINT PETERSBURG

Abstract: *The article examines technical debt as a factor affecting the development and maintenance of software systems. The main causes of technical debt accumulation and its possible consequences for software quality and development efficiency are analyzed. Approaches to identifying, assessing and managing technical debt are systematized, taking into account the characteristics of software products and the limited resources of development teams.*
Keywords: *technical debt, software engineering, software development, code quality, software architecture, refactoring, software maintenance.*

УДК 004.4

Разработка программного обеспечения представляет собой непрерывный процесс принятия технических и организационных решений. При создании и развитии цифровых продуктов необходимо одновременно учитывать требования к функциональности, сроки реализации, доступные ресурсы и качество программной системы. В практической деятельности эти требования могут вступать в противоречие. Ускорение выпуска новой функции нередко предполагает использование временных решений, упрощение внутренней структуры программы или перенос отдельных работ по улучшению качества на более поздний срок. Подобные действия могут обеспечивать краткосрочные преимущества, однако способны создавать дополнительные затраты при дальнейшем развитии продукта.

Для описания таких последствий в программной инженерии используется понятие технического долга. Оно отражает ситуацию, при которой ранее принятое техническое решение облегчает выполнение текущей задачи, но создаёт необходимость дополнительных усилий в будущем. В работе Ф. Крухтена, Р. Норда и И. Озкая технический долг рассматривается как понятие, объединяющее особенности программной архитектуры, качества реализации и управленческих решений, влияющих на дальнейшее развитие программного продукта [1, с. 18–21]. При этом наличие технического долга не обязательно означает наличие непосредственной ошибки в работе программы.

Цель настоящей статьи заключается в систематизации причин возникновения технического долга, рассмотрении его последствий для разработки и сопровождения программных систем, а также в определении подходов к управлению им. Исследование носит теоретико-аналитический характер и основано на сравнении научных публикаций, посвящённых классификации технического долга, его накоплению, оценке и приоритизации работ по устранению. Особое внимание уделяется разграничению технических компромиссов, которые могут быть экономически обоснованными, и решений, создающих неоправданные долгосрочные затраты.

Сущность технического долга можно пояснить на примере развития информационной системы организации. Предположим, что разработчикам необходимо в ограниченные сроки создать функцию

формирования отчётов. Для ускорения работы они могут разместить логику обработки данных непосредственно в пользовательском модуле, не выделяя её в самостоятельный компонент. На первоначальном этапе такое решение позволяет быстрее реализовать необходимую функциональность. Однако при появлении дополнительных типов отчётов изменение программы становится сложнее, поскольку правила обработки данных оказываются тесно связаны с интерфейсом. В дальнейшем для расширения функциональности может потребоваться переработка ранее созданной структуры.

Приведённый пример показывает принципиальное отличие технического долга от обычных затрат на развитие программного продукта. Не всякое изменение существующего кода является погашением долга. Если программа перерабатывается в связи с появлением новых требований, дополнительные усилия могут быть естественной частью её развития. О техническом долге целесообразно говорить в тех случаях, когда ранее принятое решение создаёт устранимые ограничения, увеличивающие стоимость последующих изменений по сравнению с более устойчивым техническим решением.

В исследовании Э. Том, А. Аурум и Р. Видгена подчёркивается многогранность данного понятия. Авторы рассматривают технический долг не только как следствие недостаточного качества программного кода, но и как результат взаимодействия технических, организационных и управленческих факторов [2, с. 1498–1516]. Такой подход представляется обоснованным, поскольку причины накопления долга могут возникать задолго до непосредственного написания программы и сохраняться на протяжении всего жизненного цикла продукта.

Одной из наиболее распространённых причин технического долга являются ограничения сроков разработки. При необходимости быстро представить работоспособную версию продукта команда может отказаться от отдельных улучшений архитектуры, сократить объём автоматизированных проверок или использовать упрощённые способы взаимодействия компонентов. Подобные решения не всегда являются ошибочными. Например, при создании прототипа бывает рационально проверить востребованность основной функции до вложения значительных ресурсов в сложную инфраструктуру. Проблема возникает тогда, когда временное решение сохраняется в постоянно развивающемся продукте без пересмотра его ограничений.

Другой причиной выступает недостаточная определённость требований. На ранних этапах проекта разработчики могут не располагать полной информацией о будущей нагрузке, количестве пользователей или направлениях расширения функциональности. Архитектура, соответствовавшая первоначальным условиям, со временем может перестать удовлетворять новым требованиям. При этом изменение внешних обстоятельств само по себе не свидетельствует о некачественной разработке. Технический долг возникает тогда, когда накопленные ограничения продолжают сохраняться и заметно затрудняют дальнейшие изменения.

Существенное значение имеют и особенности организации разработки. Недостаточное документирование решений, отсутствие единых требований к качеству кода, нерегулярное тестирование и слабая координация между участниками проекта могут способствовать накоплению технического долга. Например, если несколько разработчиков реализуют сходную функциональность различными способами, в программе постепенно возникают повторяющиеся компоненты. В дальнейшем изменение общего правила требует согласованной переработки нескольких участков кода, что увеличивает трудоёмкость сопровождения.

В систематическом исследовании З. Ли, П. Авгериу и П. Ляна технический долг классифицируется по различным областям программной системы, включая требования, архитектуру, проектирование, код, тестирование, документацию и инфраструктуру [3, с. 193–220]. Данная классификация позволяет рассматривать проблему шире, чем нарушение отдельных правил программирования. Различные виды долга отличаются характером последствий и требуют неодинаковых способов управления.

Долг программного кода связан с особенностями непосредственной реализации функций. Его проявлениями могут быть избыточное дублирование, чрезмерно сложные методы, недостаточно понятные названия и высокая связанность отдельных классов. Такие недостатки затрудняют чтение и изменение программы. В отдельных случаях они повышают вероятность возникновения ошибок при внесении новой функциональности. Однако оценивать качество кода необходимо с учётом назначения системы: формальное нарушение определённого правила не всегда создаёт значимые практические последствия.

Архитектурный технический долг затрагивает более общую организацию программной системы. Например, отсутствие чётких границ между функциональными модулями может приводить к тому, что изменение одной части программы требует переработки нескольких других. Подобные зависимости способны ограничивать масштабирование, обновление технологий и независимую разработку отдельных компонентов. Устранение архитектурного долга часто требует больших усилий, чем исправление локальных недостатков кода, поскольку предполагает изменение структуры взаимодействия элементов системы.

В исследовании Т. Бескер, А. Мартини и Я. Босха архитектурный технический долг рассматривается как самостоятельная проблема развития программных систем. Авторы показывают значение его выявления и понимания взаимосвязей между причинами возникновения и последствиями для сопровождения продукта [4, с. 1–16]. Это позволяет сделать вывод о необходимости учитывать архитектурные решения не только на этапе первоначального проектирования, но и в процессе дальнейшего развития программного обеспечения.

Отдельными разновидностями являются долг тестирования и документации. Первый возникает, когда существующие проверки недостаточно полно отражают функциональность системы, а изменения невозможно уверенно подтверждать автоматизированными тестами. Второй связан с отсутствием, неполнотой или устареванием сведений о структуре и работе программы. Например, при отсутствии документации по взаимодействию сервисов новый разработчик может затрачивать значительное время на восстановление логики системы. Подобные проблемы не всегда заметны пользователю, но непосредственно влияют на эффективность сопровождения.

Последствия технического долга могут проявляться постепенно. На начальном этапе отдельное упрощение не оказывает существенного влияния на процесс разработки. Однако при накоплении взаимосвязанных ограничений каждое последующее изменение требует дополнительных проверок, согласований и исправлений. В результате команда начинает расходовать больше времени на поддержку существующей функциональности, а внедрение новых возможностей становится менее предсказуемым.

Для объяснения подобной ситуации используется экономическая аналогия с обычным долгом. Первоначально принятое упрощённое решение можно рассматривать как обязательство, устранение которого требует определённых трудовых затрат. Дополнительные усилия, возникающие при последующих изменениях из-за сохранения этого решения, сопоставимы с процентами по долгу. Однако данная аналогия имеет ограничения: технический долг не является денежным обязательством, а его последствия часто нельзя заранее оценить с высокой точностью.

Кроме того, не всякий технический долг обязательно приводит к непрерывному увеличению затрат. Если определённый компонент редко изменяется и стабильно выполняет свои функции, его недостатки могут длительное время не создавать значительных проблем. Напротив, ограничения в часто изменяемой части системы способны существенно замедлять разработку. Поэтому при оценке долга необходимо учитывать не только техническое состояние программного компонента, но и частоту его изменения, значимость для продукта и вероятность возникновения дополнительных расходов.

Важным последствием накопления технического долга является снижение предсказуемости сроков выполнения задач. Разработчик может рассчитывать на относительно небольшое изменение функциональности, но в процессе работы обнаружить многочисленные зависимости и необходимость исправления ранее созданных решений. В результате фактическая трудоёмкость оказывается выше первоначальной оценки. Подобные ситуации затрудняют планирование выпуска новых версий и могут приводить к перераспределению ресурсов с развития продукта на устранение технических ограничений.

Существенное значение имеет и кадровый аспект. При высокой сложности программного кода новым сотрудникам требуется больше времени на освоение системы. Опытные разработчики могут обладать знаниями о её неочевидных зависимостях, которые недостаточно отражены в документации. При изменении состава команды подобная информация частично утрачивается. Следовательно, технический долг способен повышать зависимость проекта от отдельных работников и усложнять передачу профессиональных знаний.

Управление техническим долгом должно начинаться с его выявления. В рамках этого процесса команда определяет решения, которые способны создавать дополнительные затраты при дальнейшем развитии системы. Для обнаружения проблем могут использоваться анализ программного кода, результаты тестирования, архитектурные проверки и обсуждение сложностей, возникающих при выполнении задач. При этом автоматизированные инструменты позволяют обнаруживать лишь часть потенциальных недостатков, поэтому их результаты необходимо дополнять экспертной оценкой разработчиков.

Следующим этапом является документирование выявленных ограничений. Для каждого значимого случая целесообразно фиксировать затронутый компонент, характер проблемы, возможные последствия и предполагаемый способ устранения. Подобные записи могут включаться в общий перечень задач разработки. Это позволяет сохранять информацию о технических компромиссах и возвращаться к их пересмотру при планировании новых версий продукта. Особенно важно указывать причины сознательно принятого упрощённого решения, чтобы в дальнейшем команда могла понять первоначальные обстоятельства выбора.

Однако простое составление списка проблем не обеспечивает эффективного управления. При ограниченности ресурсов невозможно одновременно устранять все недостатки программной системы. Поэтому необходима приоритизация технического долга. В систематическом обзоре В. Ленардуцци и соавторов рассматриваются различные способы определения очередности работ по устранению долга с

учётом технических и экономических факторов. Авторы отмечают отсутствие единого универсального подхода к приоритизации, применимого ко всем программным системам [5].

Практически обоснованным представляется сопоставление нескольких характеристик: предполагаемой стоимости устранения проблемы, вероятности её влияния на будущие изменения, значимости затронутого компонента и возможных последствий сохранения текущего решения. Например, переработка часто изменяемого модуля, от которого зависит несколько основных функций системы, может иметь больший приоритет, чем исправление изолированного участка кода, который редко используется. При этом подобная оценка должна пересматриваться по мере изменения требований к продукту.

Основным техническим способом сокращения долга программного кода является рефакторинг – изменение внутренней структуры программы без намеренного изменения её внешнего поведения. Он может включать упрощение сложных методов, устранение повторяющихся фрагментов, разделение ответственности между компонентами и улучшение их взаимодействия. Однако рефакторинг должен сопровождаться проверкой работоспособности системы. Если существующая функциональность недостаточно покрыта тестами, масштабная переработка может создавать дополнительные риски.

Для предотвращения накопления новых проблем важны регулярная проверка программного кода и соблюдение согласованных требований к его качеству. Обсуждение изменений между разработчиками позволяет выявлять нежелательные зависимости, сложные конструкции и недостаточно обоснованные решения до их включения в основную кодовую базу. Автоматизированное тестирование и непрерывная интеграция дополнительно помогают обнаруживать нарушения работоспособности при внесении изменений. Вместе с тем подобные инструменты не отменяют необходимости оценки архитектуры и содержательных требований.

Важную роль играет организация планирования. Если все доступные ресурсы систематически направляются исключительно на создание новой функциональности, задачи по улучшению внутренней структуры могут откладываться неопределённо долго. Более устойчивым представляется подход, при котором работы по снижению технического долга рассматриваются как обычная часть развития продукта. Их объём определяется текущими рисками, состоянием системы и ожидаемыми направлениями изменений, а не необходимостью полностью устранить все недостатки.

При этом не следует стремиться к абсолютному отсутствию технического долга. Сложность программного обеспечения постоянно изменяется, а требования пользователей и доступные технологии обновляются. Избыточное совершенствование компонентов, которые не создают значимых ограничений, может приводить к неэффективному расходованию ресурсов. Поэтому управленческая задача состоит в поддержании приемлемого уровня технического долга, при котором сохранение отдельных компромиссов не препятствует достижению целей проекта.

Показателем результативности управления может выступать изменение трудоёмкости повторяющихся задач, частоты ошибок после обновлений и времени, необходимого для внедрения новой функциональности. Дополнительно можно учитывать количество выявленных проблем и долю наиболее значимых из них, устранённых в установленный период. Однако эти показатели должны анализироваться совместно: уменьшение числа замечаний автоматизированного анализатора само по себе не доказывает улучшения архитектуры или ускорения разработки.

Проведённый анализ позволяет заключить, что технический долг является закономерным явлением в развитии программных систем. Его возникновение может быть связано как с сознательными решениями, направленными на ускорение разработки, так и с недостатками организации работы, изменением требований или ограниченностью исходной информации. Основные отрицательные последствия проявляются в увеличении сложности сопровождения, снижении предсказуемости изменений и повышении затрат на развитие продукта.

Таким образом, эффективное управление техническим долгом предполагает его своевременное выявление, документирование, оценку возможных последствий и выборочное устранение наиболее значимых проблем. Наиболее обоснованным является не стремление к идеальному техническому состоянию программы любой ценой, а поддержание баланса между текущей скоростью разработки и долгосрочной устойчивостью программной системы. Последовательное применение такого подхода позволяет сохранять возможности дальнейшего развития продукта, ограничивать накопление трудоёмких зависимостей и более рационально использовать ресурсы команды.

Список литературы / References

1. *Kruchten P., Nord R. L., Ozkaya I.* Technical Debt: From Metaphor to Theory and Practice // IEEE Software. 2012. Vol. 29, № 6. P. 18–21. DOI: 10.1109/MS.2012.167. URL: https://doi.org/10.1109/MS.2012.167 (дата обращения: 22.11.2026).

2. *Tom E., Aurum A., Vidgen R.* An exploration of technical debt // Journal of Systems and Software. 2013. Vol. 86, № 6. P. 1498–1516. DOI: 10.1016/j.jss.2012.12.052. URL: https://doi.org/10.1016/j.jss.2012.12.052 (дата обращения: 22.11.2026).
3. *Li Z., Avgeriou P., Liang P.* A systematic mapping study on technical debt and its management // Journal of Systems and Software. 2015. Vol. 101. P. 193–220. DOI: 10.1016/j.jss.2014.12.027. URL: https://doi.org/10.1016/j.jss.2014.12.027 (дата обращения: 22.11.2026).
4. *Besker T., Martini A., Bosch J.* Managing architectural technical debt: A unified model and systematic literature review // Journal of Systems and Software. 2018. Vol. 135. P. 1–16. DOI: 10.1016/j.jss.2017.09.025. URL: https://doi.org/10.1016/j.jss.2017.09.025 (дата обращения: 22.11.2026).
5. *Lenarduzzi V., Besker T., Taibi D., Martini A., Arcelli Fontana F.* A systematic literature review on Technical Debt prioritization: Strategies, processes, factors, and tools // Journal of Systems and Software. 2021. Vol. 171. Art. 110827. DOI: 10.1016/j.jss.2020.110827. URL: https://doi.org/10.1016/j.jss.2020.110827 (дата обращения: 22.11.2026).