Максим Степаненко



Технический долг: как стартап убивает себя своим кодом

Что такое tech debt, как его измерять и когда платить

Максим Степаненко · 2026

Технический долг: как стартап убивает себя своим кодом

Что такое tech debt, как его измерять и когда платить

Максим Степаненко

Innova Publishing · 2026

Страница книги, свежая версия и другие гайды серии: guides.mmix.biz/tech-debt

Помощь специалистов по теме «IT-консалтинг: аудит и план»: Marketing MIX — mmix.ua

ГЛАВА 1

Что такое технический долг и откуда он берётся

Технический долг — это метафора предложенная Ward Cunningham в 1992 году. Быстрое решение сейчас = взять кредит у будущего. Это не обязательно плохо — иногда стратегически правильно. Проблема начинается когда долг накапливается и начинает тормозить скорость разработки, причиняя «процентные платежи» в виде медленной delivery и большого числа багов.

Откуда берётся tech debt: Time pressure (MVP rush): «Нам нужно запустить через 2 недели». Решения принимаются быстро, тесты не пишутся, архитектура не продумана. Это нормально и часто правильно на pre-product stage. Growing team: разные разработчики имеют разные стандарты. Без code review culture — каждый пишет по-своему. Аккумулируется inconsistency. Changing requirements: архитектура спроектирована для одного use case. Requirements меняются, добавляются хаки и workarounds вместо redesign. Knowledge silos: «только Вася знает как работает модуль авторизации». Вася увольняется. Никто не понимает код. Outsourced development: внешние контракторы оптимизируют completion, не maintainability. Прощай архитектура.

Виды tech debt по Cunningham: Reckless deliberate: «Некогда думать об архитектуре — запускаемся». Prudent deliberate: «Мы знаем что это не идеально, но сейчас нужно запустить — исправим позже». Reckless inadvertent: «Мы не знали что делаем это неправильно». Prudent inadvertent: «Теперь понимаем как надо было сделать». Reckless deliberate — самый токсичный. Prudent deliberate — нормальная бизнес-практика при правильном управлении.

Когда tech debt убивает стартапы: стартап растёт. Нанимает больше разработчиков. Velocity падает (не растёт!) при каждом

новом найме. Каждый новый feature требует трогать 10 мест в коде. Один баг fix создаёт три новых. CEO: «Мы наняли ещё 5 разработчиков, почему мы деплоим медленнее?» Это advanced stage tech debt poisoning. Признаки: cycle time feature от идеи до продакшена > 4 недель. Более 30% времени команды — bug fixes и incidents. Разработчики боятся трогать определённые части кода. Onboarding нового разработчика занимает 2+ месяца.

ГЛАВА 2

Как измерять tech debt объективно

«У нас много tech debt» — субъективное ощущение. CEO не купит его без данных. Вот как измерить объективно.

DORA Metrics (DevOps Research and Assessment): четыре метрики которые коррелируют с business performance: Deployment Frequency: как часто деплоите в production? Elite: multiple times per day. Low: once per month or less. Lead Time for Changes: от commit до production. Elite: < 1 час. Low: 1–6 месяцев. Change Failure Rate: % deployments вызывающих incident. Elite: 0–15%. Low: 46–60%. Time to Restore Service: как быстро восстанавливаетесь после incident. Elite: < 1 час. Low: 1–6 месяцев. Высокий tech debt = плохие DORA metrics. Это objective доказательство для CEO.

Cycle Time: от «start» (developer picks up ticket) до «done» (feature in production). Измеряйте в Jira, Linear, или вашем project management tool. Тренд важнее абсолютного числа. Если cycle time растёт при той же команде — tech debt accumulates.

Defect Rate: % bugs reported per feature shipped. Если растёт — quality падает. Code Coverage: % кода покрытого automated tests. Не golden metric (70% coverage может быть бесполезным если тесты плохие). Но полное отсутствие тестов = risk indicator.

Developer Experience Survey: раз в квартал спрашивайте команду: «По шкале 1–10 насколько легко вам было вносить изменения в последние 2 недели?» «Что больше всего мешало вашей работе?» «Есть ли части кода которых вы боитесь трогать?» Качественные данные + количественные = полная картина.

SonarQube и CodeClimate: инструменты статического анализа кода. Считают: code duplication (DRY нарушения).

Complexity (cyclomatic complexity). Code smells (паттерны ассоциированные с проблемами). Technical Debt в часах (оценка сколько нужно чтобы исправить). SonarQube: free self-hosted. CodeClimate: $50+/мес для teams.

ГЛАВА 3

Prioritization: когда платить долг, когда игнорировать

Не весь tech debt стоит возвращать. Главный вопрос: какой долг тормозит growth больше всего?

Tech Debt ROI framework: для каждого долга оцените: Impact на velocity (1–5): насколько этот долг замедляет delivery? Impact на reliability (1–5): насколько часто этот долг вызывает баги/incidents? Effort to fix (1–5, где 5 = много усилий). ROI score = (Velocity + Reliability) / Effort. Приоритизируйте по убыванию ROI score.

Категории tech debt по срочности: P0 — Fix Now: вызывает production incidents. Security vulnerabilities. Блокирует ключевые customer requests. P1 — Fix This Sprint: существенно замедляет development. Часто обсуждается в standups как friction. P2 — Schedule for Next Quarter: снижает code readability. Будет тормозить будущие features. P3 — Track but Don't Fix Yet: code smell без практического impact. Может быть никогда не тронут.

«Boy Scout Rule» (постепенное улучшение): каждый раз когда трогаете кусок кода — оставьте его немного лучше чем нашли. Не рефакторьте всё. Просто: добавьте тест для функции которую изменили. Улучшите название переменной которую запутали. Удалите dead code который встретили. Это organic debt reduction без специальных sprint'ов.

20% time: выделите 20% developer time на tech debt каждый sprint. Это не «nice to have» — это investment в future velocity. Frame it for CEO: «Если мы не инвестируем 20% сейчас, в следующем году мы будем делать то же самое за 2x дольше». Debt Sprints: раз в квартал — full sprint только на tech debt. Подходит если debt накопился критически. Минус: developers теряют context с product.

Никогда не делайте: полный rewrite «с нуля». Big Bang Rewrite — почти всегда провальная стратегия. Занимает в 3–5x больше чем ожидается. Продукт стоит пока rewrite идёт. «Новый» код накапливает собственный долг. Альтернатива: Strangler Fig pattern. Постепенная замена по частям.

ГЛАВА 4

Как говорить о tech debt с CEO и инвесторами

Технический долг невидим для нетехнических стейкхолдеров. Задача CTO/Tech Lead — сделать его видимым через business language.

Переводите технические метрики в бизнес-метрики: Вместо «у нас высокий cyclomatic complexity» → «каждый новый feature требует в 2x больше времени из-за сложности кода». Вместо «нет unit tests» → «мы обнаруживаем 40% багов в production вместо development, что обходится нам в N дней восстановления». Вместо «tech debt» → «engineering debt» или «infrastructure investment». Слова имеют значение.

Business case для tech debt investment: ROI расчёт: текущая velocity: 5 features/мес. При текущем росте долга: через 12 месяцев velocity упадёт до 2 features/мес. Стоимость разработчиков: $50,000/мес. 50% production on fixing bugs vs new features: это $25,000/мес «потерь». Инвестиция в 2 месяца debt reduction: $100,000. Результат: velocity возвращается к 5+ features/мес. ROI: окупается за 4 месяца.

Визуализация tech debt: burn-up chart с velocity trend. Показывает падение velocity с течением времени. Bug rate over time. Incident frequency. Эти графики понятны любому CEO.

Pre-funding technical due diligence: investors и acquirers делают technical due diligence. Red flags: monolith без документации. Нет тестов. Нет CI/CD. Security vulnerabilities (найдут через SAST tools). Key-person dependency (код читает только один человек). Подготовьтесь заранее: базовый CI/CD. Документация ключевых архитектурных решений. Secret scanning в репозитории. Dependency audit (outdated libraries с known CVEs).

Культура качества: tech debt — симптом, не причина. Причина: культура которая не valuing code quality. Как создать культуру качества: code reviews обязательны (даже если slow). Definition of Done включает тесты. «Broken window» policy: не оставляете broken code. Blameless postmortems после incidents. Celebrate refactoring наравне с new features.

ГЛАВА 5

Инструменты и практики для управления долгом

Правильные инструменты делают управление tech debt системным а не реактивным.

CI/CD как основа: Continuous Integration: автоматический build и тесты при каждом PR. GitHub Actions, GitLab CI, CircleCI — бесплатны для small teams. Continuous Delivery: автоматический deploy в staging после merge. Continuous Deployment: автоматический deploy в production. Без CI/CD: manual, slow, error-prone. С CI/CD: быстро, надёжно, устраняет человеческий фактор из deployment.

Автоматизированные тесты: Unit tests: тестируют отдельные функции. Быстрые. Integration tests: тестируют взаимодействие компонентов. E2E tests: тестируют user flows в браузере (Playwright, Cypress). Не гонитесь за 100% coverage. Начните с: tests для business-critical flows. Tests для часто ломающихся мест. Tests для каждого нового bug (regression tests).

Мониторинг и observability: Datadog, New Relic, или self-hosted Grafana+Prometheus. Основные метрики: application performance (p95, p99 latency). Error rate. Database query performance. Infrastructure health. Sentry или Bugsnag: автоматический error tracking. Каждый unhandled exception → ticket в вашем tracker. PagerDuty или OpsGenie: incident management и on-call rotation.

Documentation: Architecture Decision Records (ADR): короткие документы о ключевых архитектурных решениях. Почему мы выбрали PostgreSQL а не MongoDB? Почему microservices а не monolith? Это бесценно через 12 месяцев. README.md в каждом репозитории: как запустить locally. Как запустить тесты. Как делать deploy. Onboarding guide для новых разработчиков.

Dependency management: Dependabot (GitHub) или Renovate — автоматически создают PR для обновления зависимостей. OWASP Dependency Check — проверяет known CVEs в зависимостях. Устаревшие зависимости = tech debt + security risk. Автоматизируйте их обновление.

Что дальше

Спасибо, что прочитали «Технический долг: как стартап убивает себя своим кодом».

Обновления этой книги, файлы EPUB и PDF и все гайды серии — на странице книги: guides.mmix.biz/tech-debt

Если нужна помощь специалистов по теме «IT-консалтинг: аудит и план»: Marketing MIX — mmix.ua.

Максим Степаненко, Innova Publishing.