ИНФОРМАЦИОННАЯ СИСТЕМА ДИАГНОСТИКИ ПРОЦЕССОВ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ НА ОСНОВЕ МЕТРИК DORA И SPACE С АРХИТЕКТУРНОЙ СЕГРЕГАЦИЕЙ ВИДИМОСТИ ДАННЫХ

С. Н. Яцук[1,а], **С. Г. Еловой**[1,2,б]

[1] *Сургутский государственный университет, г. Сургут, Российская Федерация*

[2] *Сургутский филиал федерального государственного бюджетного учреждения «Федеральный научный центр Научно-исследовательский институт системных исследований Национального исследовательского центра «Курчатовский институт», г. Сургут, Российская Федерация*

[а] *ORCID: https://orcid.org/0009-0002-8257-1132,* ✍ *yatsuk_sn@edu.surgu.ru*

[б] *ORCID: https://orcid.org/0000-0002-1860-9303, s.elovoy@yandex.ru*

Аннотация: рассмотрена задача автоматизированной диагностики процессов разработки программного обеспечения на основе данных систем контроля версий. Показано, что существующие инструменты инженерной аналитики преимущественно реализуют сценарий управленческого контроля производительности сотрудников, что в силу закона Гудхарта приводит к деградации собираемых показателей и сопротивлению команд. Предложена архитектура информационной системы, в которой защита приватности участников и противодействие закону Гудхарта являются не дополнительной функцией, а условием реализации программного интерфейса: ролевая сегрегация видимости данных реализована на уровне REST API по принципу защиты в глубину и возвращает отказ доступа на уровне сервиса, а не маскирует данные в пользовательском интерфейсе. В качестве методологической основы использованы исследовательская модель DORA в актуальной пятиметричной редакции и фреймворк SPACE как принцип многомерности диагностики. Описаны состав измеряемых метрик и алгоритмы их расчета на основе инкрементально собираемых данных GitLab: время прохождения изменений, частота развертываний, доля неудачных изменений, декомпозированное время жизни запроса на слияние, распределение размеров запросов на слияние и показатель устойчивости команды Bus Factor по модулям кодовой базы. Реализован минимально жизнеспособный продукт системы с тремя ролевыми дашбордами, парной визуализацией показателей скорости и качества, прозрачностью алгоритмов расчета, экспортом отчетов и развертыванием средствами docker-compose, пригодным для сценариев локализации обрабатываемых данных. Результаты применимы для диагностики процессов разработки в командах, использующих GitLab.

Ключевые слова: диагностика процессов разработки программного обеспечения, метрики DORA, фреймворк SPACE, закон Гудхарта, анализ данных Git-репозиториев, система контроля версий, ролевое разграничение доступа, парная визуализация показателей, приватность данных, bus factor, REST API.

Благодарности: работа выполнена в рамках государственного задания НИЦ «Курчатовский институт» — НИИСИ по теме № FNEF-2024-0001 «Создание и реализация доверенных систем искусственного интеллекта, основанных на новых математических и алгоритмических методах, моделях быстрых вычислений, реализуемых на отечественных вычислительных системах» (1023032100070-3-1.2.1).

Для цитирования: Яцук С. Н., Еловой С. Г. Информационная система диагностики процессов разработки программного обеспечения на основе метрик DORA и SPACE с архитектурной сегрегацией видимости данных. *Успехи кибернетики.* 2026;7(3):50–55.

Поступила в редакцию: 20.07.2026. *В окончательном варианте: 21.09.2026.*

INFORMATION SYSTEM FOR SOFTWARE DEVELOPMENT PROCESS ANALYSIS USING DORA AND SPACE METRICS

S. N. Yatsuk[1,a], **S. G. Elovoi**[1,2,b]

[1] *Surgut State University, Surgut, Russian Federation*

[2] *Surgut Branch of Scientific Research Institute for System Analysis of the National Research Centre "Kurchatov Institute", Surgut, Russian Federation*

[a] ORCID: https://orcid.org/0009-0002-8257-1132, ✍ yatsuk_sn@edu.surgu.ru
[b] ORCID: https://orcid.org/0000-0002-1860-9303, s.elovoy@yandex.ru

Abstract: we developed an information system for automated diagnostics of software development processes based on version-control data. We analyzed limitations of existing engineering analytics tools, which often focus on managerial monitoring of individual performance and may create incentives to optimize measured indicators rather than underlying processes, consistent with Goodhart's law. We designed an architecture in which privacy protection and resistance to such metric-driven behavior are treated as requirements of the application programming interface rather than as user-interface features. Role-based access to data is enforced at the REST API level according to the defense-in-depth principle, with unauthorized requests rejected at the service layer rather than hidden only in the user interface. The methodological framework combines the five-metric DORA model with the SPACE framework for multidimensional assessment. We defined algorithms for computing lead time for changes, deployment frequency, change failure rate, decomposed merge-request cycle time, merge-request size distribution, and the Bus Factor for code modules from incrementally collected GitLab data. We implemented a minimum viable product with three role-based dashboards, paired visualization of speed and quality metrics, transparent calculation algorithms, report export, and Docker Compose deployment suitable for data-localization requirements. The system can be used to diagnose software development processes in teams working with GitLab.

Keywords: software development diagnostics, DORA metrics, SPACE framework, Goodhart's law, Git repository data analysis, version management system, role-based access control, paired metric visualization, data privacy, bus factor, REST API.

Acknowledgements: this study is a part of the government contract No. FNEF-2024-0001 Development and Implementation of Trusted AI Systems Using New Mathematical and Algorithmic Methods; Fast Computing Models on Domestic Hardware awarded to the Kurchatov Institute (1023032100070-3-1.2.1).

Cite this article: Yatsuk S. N., Elovoi S. G. Information System for Software Development Process Analysis Using DORA and SPACE Metrics. *Russian Journal of Cybernetics.* 2026;7(3):50–55.

Original article submitted: 20.07.2026. *Revision submitted: 21.09.2026.*

Введение

Сложность программных проектов растет, команды работают распределенно, а требования к скорости поставки повышаются. Управлять инженерным процессом по субъективным оценкам уже недостаточно: организация, которая не видит картины своей разработки, медленнее реагирует на изменения и накапливает скрытые риски: перегрузку участников, деградацию кода, концентрацию знаний на отдельных разработчиках.

Ответом стали модели измерения процессов разработки. Программа DORA выявила системные метрики, которые коррелируют с производительностью команд и их операционной устойчивостью [1]. Актуальная редакция включает пять показателей в группах пропускной способности и нестабильности. Фреймворк SPACE дополняет DORA принципом многомерности: он требует учитывать сразу несколько граней продуктивности и предостерегает от метрик активности в отрыве от метрик результата [2].

Коммерческие платформы позиционируются как средства управленческого контроля производительности сотрудников, что противоречит закону Гудхарта: показатель, который стал целью оценки, перестает быть достоверной мерой [3]. Такие метрики искажаются в ущерб работе и вызывают сопротивление команд. Решения с открытым кодом ограничиваются статистикой коммитов и не дают ценности уровня DORA. Системы к тому же ориентированы на зарубежные облачные хостинги, тогда как многие отечественные организации работают в GitLab и требуют локализации данных. Ранее мы предложили платформу GITALPHA для оценки эффективности разработки [4]; настоящая работа развивает подход и смещает акцент с оценки сотрудников на диагностику процессов.

Цель работы — спроектировать и реализовать информационную систему диагностики процессов разработки ПО, которая инкрементально собирает данные из репозиториев GitLab, рассчитывает метрики DORA и SPACE, визуализирует и выгружает их при ролевом разграничении

доступа. Объект исследования — процессы разработки в распределенных командах; предмет — методы и средства их автоматизированной диагностики. Новизна состоит в архитектуре, где защита приватности участников и противодействие закону Гудхарта выступают условием реализации REST API.

Методологическая основа диагностики

Методологическую основу составляют три положения. Первое — модель DORA в пятиметричной редакции. Группа Throughput включает время прохождения изменений, частоту развертываний и время восстановления после неудачного развертывания; группа Instability — долю неудачных изменений и долю переделок. Ключевой вывод DORA: скорость и стабильность поставки не исключают друг друга и команды-лидеры показывают высокие значения по всем метрикам [1]. Это обосновывает парное представление показателей.

Второе — фреймворк SPACE, который рассматривает продуктивность как многомерную величину. Пять измерений модели — Satisfaction, Performance, Activity, Communication, Efficiency — задают рамку, где ни одна грань не должна доминировать. Метрики активности собираются проще прочих и потому опасны в отрыве от остальных граней [2].

Третье — закон Гудхарта как ограничитель. В формулировке М. Стратерн: когда мера становится целью, она перестает быть хорошей мерой [3]. Метрика, которая превращена в инструмент индивидуальной оценки, деформирует поведение: счетчик коммитов ведет к инфляции мелких коммитов, а премия за число строк наказывает за рефакторинг. Противодействовать этому нужно на уровне архитектуры, а не рекомендаций.

Состав измеряемых метрик и алгоритмы расчета

Реализованы шесть метрик трех уровней: поток поставки, код-ревью и устойчивость команды. Все они трактуются как характеристики процесса, а не сотрудника, и считаются на уровне команды; исключение — личный кабинет разработчика.

Метрики потока поставки взяты из DORA. Время прохождения изменений — интервал между первым коммитом по задаче и успешным развертыванием изменения:

$$= t_{deploy} - t_{commit}, \tag{1}$$

где t_{deploy} — момент развертывания, t_{commit} — момент первого коммита. Развертывания определяются по тегам репозитория; чтобы результат был устойчив к выбросам, отображаются медиана и 90-й перцентиль. Частота развертываний — число успешных развертываний за единицу времени:

$$DF = N_{deploy}/T, \tag{2}$$

где N_{deploy}— число успешных развертываний за период, T — длительность периода. Доля неудачных изменений — парная метрика — отношение числа развертываний с откатом или экстренным исправлением к общему их числу:

$$CFR = \frac{N_{fail}}{N_{deploy}} \cdot 100\%, \tag{3}$$

где N_{fail} — число развертываний, которые помечены как неудачные по меткам вошедших в релиз запросов на слияние.

Метрики код-ревью извлекаются из данных GitLab и не требуют интеграции со средствами непрерывной интеграции. Код-ревью чаще всего становится зоной деградации процесса и одновременно распространяет знания и контролирует качество [5, 6]. Время жизни запроса на слияние разбивается на три фазы и полный цикл:

$$\Delta_1 = t_{fr} - t_{open}, \tag{4}$$

$$\Delta_2 = t_{ap} - t_{fr}, \tag{5}$$

$$\Delta_3 = t_{mg} - t_{ap}, \tag{6}$$

$$\Delta = t_{mg} - t_{open}, \tag{7}$$

где t_{open} — момент открытия запроса, t_{fr} — первый комментарий ревьюера, t_{ap} — согласование, t_{mg} — слияние. Декомпозиция позволяет найти узкое место процесса; черновые запросы из выборки исключаются. Размеры запросов распределяются по бакетам числа измененных строк {≤50, 51–200, 201–400, 401–800, >800}: крупные запросы просматриваются формально, что обесценивает проверку [7]. Доля возвратов и откатов служит парным индикатором качества ревью [8].

Метрики устойчивости выявляют риски, которые не видны в текущей работе. Реализован Bus Factor по модулям: для каждого модуля, который задается шаблоном пути к файлам, подсчитывается число активных контрибьюторов за 90 суток:

$$BF\,(m) = A_m \vee, \tag{8}$$

где A_m — множество активных контрибьюторов модуля m за окно наблюдения. Модуль с единственным контрибьютором считается риском концентрации знаний [9, 10].

Соответствие метрик граням SPACE приведено в таблице 1, доминирующей грани нет. Грань Satisfaction требует опросов и остается вне автоматического сбора.

Таблица 1

Соответствие метрик системы граням модели SPACE

Грань SPACE	**Метрики системы**	**Способ покрытия**
S – Satisfaction	–	Вне автоматического сбора (опросы)
P – Performance	Change Failure Rate; Reopen / Revert Rate	Качество результата работы
A – Activity	MR Size; Deployment Frequency	Объем действий в паре с качеством
C – Communication	Review Coverage; Knowledge Sharing Index; Bus Factor	Паттерны взаимодействия и обмена знаниями
E – Efficiency / Flow	Lead Time; Cycle Time MR; Failed Deployment Recovery Time	Эффективность сквозного процесса

Архитектура сегрегации видимости данных

Центральное проектное решение — архитектурная сегрегация видимости данных, которая делает доступ руководства к индивидуальным показателям участников технически невозможным. Это не настройка прав, а встроенное свойство программного интерфейса.

Ролевая модель построена на двух независимых осях. Глобальная роль пользователя определяет класс доступных операций, роль в команде — отношение пользователя к конкретной команде. Пользователь может иметь глобальную роль лида, но не быть лидом конкретной команды, и тогда он не получит ее данных.

Разграничение построено по принципу защиты в глубину и состоит из двух ступеней. Первая — промежуточное ПО уровня маршрута, которое фильтрует глобальную роль: командные агрегаты доступны участникам команды, лиду и руководителю отдела; метрики код-ревью — только лиду; метрики DORA и Bus Factor — лиду и руководителю отдела. Вторая — проверка области видимости на уровне сервиса: она устанавливает принадлежность пользователя к команде и отказывает в доступе к данным вне его зоны видимости.

Матрица допустимых сочетаний роли и метрики вынесена в единый источник истины: сервис формирует общий набор метрик команды и обнуляет недоступные роли показатели, поэтому клиент отличает отсутствие доступа от отсутствия данных; тот же механизм действует при экспорте.

Сегрегация образует три уровня видимости (таблица 2). Разработчик видит показатели своей работы и обезличенный командный ориентир; тимлид — командные агрегаты, но не значения участников; руководитель отдела — кросс-командные метрики DORA.

Архитектуру дополняют два принципа интерфейса. Парная визуализация означает, что метрика скорости всегда показывается вместе с метрикой качества: частота развертываний —

Таблица 2

Ролевая модель видимости данных

Роль	**Доступно**	**Недоступно**
Разработчик	Полные индивидуальные метрики; командный baseline; история своих показателей	Индивидуальные метрики других участников
Тимлид	Командные агрегаты; Cycle Time MR с декомпозицией; MR Size; Bus Factor; сигналы аномалий без раскрытия значений	Конкретные индивидуальные значения участников
Руководитель отдела	Кросс-командные метрики DORA; сравнительная динамика команд	Любые индивидуальные данные; данные глубже уровня команды

рядом с долей неудачных изменений, время жизни запроса — рядом с распределением размеров. Это мешает оптимизировать одну ось в ущерб другой. Прозрачность алгоритмов означает, что формула расчета и источники данных каждой метрики доступны прямо в интерфейсе, что снижает барьер доверия команд.

Сбор и обработка данных, реализация системы

Система состоит из серверного и клиентского приложений, которые взаимодействуют по REST. Сервер написан на Node.js со сквозной статической типизацией; хранилищем служит PostgreSQL: предметная модель сильно связна, а расчет времени прохождения изменений требует оконного сопоставления запросов на слияние с развертываниями. Клиент — одностраничное веб-приложение.

Сбор данных организован как периодический инкрементальный процесс (рис. 1). Авторизация в GitLab выполняется по персональному токену: он шифруется алгоритмом AES-256-GCM и расшифровывается только в момент обращения к платформе. Планировщик обходит подключенные проекты и для каждого выгружает коммиты, запросы на слияние с их ревью и теги релизов, после чего связывает развертывания с вошедшими в них запросами. Вставки идемпотентны: повторный запуск не порождает дубликатов, а прерванный процесс продолжается с закладки. Рассчитанные значения материализуются в снапшоты, что разгружает дашборды.

Рис. 1. *Конвейер инкрементального сбора и обработки данных системы*

Расчетная часть построена по шаблону «стратегия»: введены абстрактный калькулятор метрики и набор конкретных калькуляторов, которые реализованы как чистые функции над выборкой из базы данных и потому просто тестируются. Модуль экспорта формирует отчеты в формате CSV и переиспользует те же функции расчета, что и интерфейс. Сервер, СУБД и клиент разворачиваются средствами docker-compose, что позволяет ввести систему в эксплуатацию в инфраструктуре заказчика.

Результаты и обсуждение

Результат работы — реализованный минимально жизнеспособный продукт. Он обеспечивает подключение к инстансу GitLab; инкрементальный сбор данных о коммитах, запросах на

слияние, ревью и релизах; расчет шести метрик; три ролевых дашборда; ролевое разграничение доступа на уровне REST API; экспорт отчетов в формате CSV; развертывание средствами docker-compose.

От коммерческих платформ система отличается архитектурным характером защиты приватности: ролевая сегрегация видимости реализована на уровне программного интерфейса, а не в настройках. Это снимает целый класс угроз: команды доверяют системе добровольно лишь тогда, когда ее нельзя применить как инструмент индивидуального контроля. В нашей предыдущей работе [4] платформа GITALPHA оценивала эффективность разработки; здесь предметом измерения выступает процесс, а не сотрудник [11].

Границы продукта заданы сознательно. Метрики Failed Deployment Recovery Time и Deployment Rework Rate требуют интеграции с системами мониторинга и отнесены к направлениям развития; показатели Workload Distribution и Knowledge Sharing Index, а также сигналы аномалий рассматриваются как расширение. Поддержку других хостингов даст абстрактный коннектор источников данных.

Заключение

Предложена и реализована информационная система диагностики процессов разработки ПО по данным Git-репозиториев. Методологическую основу составили модель DORA в пятиметричной редакции, фреймворк SPACE и закон Гудхарта. Основной результат — архитектура, где защита приватности участников и противодействие закону Гудхарта выступают не дополнительной функцией, а условием реализации программного интерфейса: ролевая сегрегация видимости обеспечивается на уровне REST API и дополняется парной визуализацией скорости и качества. Система применима в командах, которые используют GitLab, в том числе при локальном развертывании. Развитие связано с расширением состава метрик и поддержкой других источников данных.

ЛИТЕРАТУРА

1. Forsgren N., Kersten M. DevOps Metrics. *Communications of the ACM.* 2018;61(4):44–48. DOI: 10.1145/3159169.
2. Forsgren N., Storey M.-A., Maddila C., Zimmermann T., Houck B., Butler J. The SPACE of Developer Productivity: There's More to It than You Think. *ACM Queue.* 2021;19(1):20–48. DOI: 10.1145/3454122.3454124.
3. Strathern M. 'Improving Ratings': Audit in the British University System. *European Review.* 1997;5(3):305–321.
4. Яцук С. Н., Еловой С. Г. Аналитическая платформа для оценки эффективности разработки GITALPHA. *Труды НИИСИ РАН.* 2026;16(1):49–52.
5. Bacchelli A., Bird C. Expectations, Outcomes, and Challenges of Modern Code Review. *Proc. 35th Int. Conf. on Software Engineering (ICSE).* 2013:712–721. DOI: 10.1109/ICSE.2013.6606617.
6. McIntosh S., Kamei Y., Adams B., Hassan A. E. An Empirical Study of the Impact of Modern Code Review Practices on Software Quality. *Empirical Software Engineering.* 2016;21(5):2146–2189.
7. Gousios G., Pinzger M., van Deursen A. An Exploratory Study of the Pull-Based Software Development Model. *Proc. 36th Int. Conf. on Software Engineering (ICSE 2014).* 2014:345–355.
8. Beller M., Bacchelli A., Zaidman A., Juergens E. Modern Code Reviews in Open-Source Projects: Which Problems Do They Fix? *Proc. 11th Working Conf. on Mining Software Repositories (MSR).* 2014:202–211.
9. Avelino G., Passos L., Hora A., Valente M. T. A Novel Approach for Estimating Truck Factors. *Proc. 24th IEEE Int. Conf. on Program Comprehension (ICPC).* 2016:1–10.
10. Jabrayilzade E., Evtikhiev M., Tüzün E., Kovalenko V. Bus Factor in Practice. *Proc. 44th Int. Conf. on Software Engineering: Software Engineering in Practice (ICSE-SEIP).* 2022:97–106.
11. Rigby P. C., Bird C. Convergent Contemporary Software Peer Review Practices. *Proc. 2013 9th Joint Meeting on Foundations of Software Engineering (ESEC/FSE).* 2013:202–212.