УДК 004.053
https://doi.org/10.36906/AP-2022/38

Казаченко Т.А., Мелехин М.И.
Литовка А.Д., Батыркаев А.Г.
Аитов А.И.
ORCID: 0000-0003-4941-9104, канд. физ.-мат. наук
Пермский государственный аграрно-технологический университет
им. академика Д.Н. Прянишникова
г. Пермь, Россия

АРХИТЕКТУРНЫЙ ТЕХНИЧЕСКИЙ ДОЛГ В AGILE-ПРОЕКТАХ

Аннотация. Вопрос образования архитектурного технического долга и управления им особенно актуален в промышленных разработках программных продуктов с использованием популярных гибких технологий. Архитектурный технический долг наиболее тяжелый в обслуживании, серьезно влияет на качество создаваемого кода и требует значительных затрат времени и усилий на изменение. Знание причин образования такого технического долга даст возможность сформулировать принципы, соблюдение которых позволит им управлять.

Ключевые слова: гибкие технологии; архитектурное решение; архитектурный технический долг.

Kazachenko T.A., Melekhin M.I.
Litovka A.D., Batyrkaev A.G.
Aitov A.I.
ORCID: 0000-0003-4941-9104, Ph.D.
Perm State Agro-Technological University named after Academician D.N. Pryanishnikov
Perm, Russia

ARCHITECTURAL TECHNICAL DEBT IN AGILE PROJECTS

Abstract. The issue of the formation of architectural technical debt and its management is especially relevant in the industrial development of software products using popular flexible technologies. Architectural technical debt is the hardest to maintain, seriously affects the quality of the generated code and requires a significant amount of time and effort to change. Knowing the reasons for the formation of such a technical debt will make it possible to formulate principles that will allow them to manage.

Keywords: agile technologies; architectural design decisions; architectural technical debt.

Согласно отчёту ScrumTrek (https://clck.ru/dVnaf) в России все чаще используют гибкие технологии в разработках программных продуктов. Как отмечает эта консалтинговая

компания, главным привлекательным эффектом от внедрения Agile является высокая скорость поставки продукта бизнес-заказчику. Для гибкого подхода характерно инкрементальное проектирование, то есть архитектура наращивается в рамках каждого спринта итеративным и экспериментальным способом. Такой подход позволяет постоянно уточнять архитектуру разработки с изменением знаний команды разработчиков о ней. Поэтому в agile-проектах в отличие от waterfall-проектов не существует предварительной архитектуры разрабатываемого ПО. Об этом же говорят практикующие разработчики: «Нельзя прийти и создать сразу идеальный проект. Каждый день мы будем что-то узнавать о предметной области, и эти новые знания нужно сразу внедрять. Вначале это будут костыли, из которых копится технический долг, – и это нормально. Ведь мы не можем, получая новую информацию, каждый раз переписывать продукт заново. Бизнес нас не поймёт» Алексей Некрасов, лидер направления Python в МТС, программный директор направления Python в Skillbox (https://clck.ru/dVnXy). Таким образом, некоторые архитектурные решения (АР), принятые во время разработки программного обеспечения, влекут за собой технический долг, который может быть как преднамеренным, так и случайным.

Особенностью решений архитектурного проектирования является то, что они оказывают наибольшее влияние на качество программной системы, и после реализации их трудно изменить, это хорошо осознается и практиками, и исследователями. Именно поэтому сначала в рамках спринта архитекторы определяют архитектурное решение, не требующего технического обслуживания, соответствующее функциональным и нефункциональным требованиям заказчика. Такое поддерживаемое архитектурное решение может служить целью последующего рефакторинга альтернативных решений и работы с техническим долгом.

Если заранее известно, что новые требования к разрабатываемому ПО в последующих итерациях должны основываться на принятом в настоящее время архитектурном решении, если есть намерение повторно использовать разработанные программные компоненты в других продуктах, то целесообразно найти способ реализовать найденное поддерживаемое архитектурное решение, не влекущее технических долгов. В этом случае разработка стабильной и поддерживаемой, а не хрупкой и сложной в обслуживании архитектуры продукта может облегчить выполнение этих требований.

Однако, в силу ряда условий, присущих гибким технологиям разработки, о которых скажем ниже, как правило, не удаётся реализовать поддерживаемое решение, не влекущее технический долг. Поэтому параллельно с таким решением формируются альтернативные, в которых технический долг уже преднамеренно заложен. Архитектурные решения, влекущие за собой преднамеренное возникновение долгов, представляют собой особый тип АР с заранее известными рисками и отрицательными последствиями для ремонтопригодности и возможности развития программной системы. Работая над этими решениями, инженеры-программисты стремятся достичь в первую очередь двух основных целей:

• *выполнение видимых требований заказчика*, т. е. решение, влекущее за собой возникновение долгов, должно соответствовать всем функциональным требованиям заказчика и атрибутам качества во время выполнения;

• *локализации возникшей технической задолженности*, т. е. архитектурное решение, вызывающее техническую задолженность, должно быть заключено в ограниченное количество компонентов. Это упростит его реинжиниринг до поддерживаемого решения в будущем выпуске.

Для выработки таких альтернативных решений делаются одноразовые прототипы MVP (Minimum Viable Product) пользовательских историй, позволяющие при минимальных затратах синхронизировать разработку с исследованиями реакции на продукт целевой аудитории. Окончательный выбор альтернативного АР делается на основе подтверждённых гипотез пользовательских историй, включённых в работу, по принципу «решить в последний ответственный момент», когда техническая неопределённость разработки существенно снижена. Наиболее часто встречающимися ограничениями при разработке архитектурных решений, влекущих технический долг, являются несколько факторов:

• *строгие временные ограничения*. Например, фиксированная дата доставки продукта на рынок до выпуска аналогичной разработки конкурентами;

• *доступность ресурсов*. Например, если одна из команд занята разработкой других функций и не может участвовать в разработке поддерживаемого архитектурного решения, то предпочтительным будет решение, влекущее за собой возникновение долгов;

• *лояльность к продуктам компании*. Инженеры-программисты могут быть вынуждены выбирать в качестве инструментов разработки продукты своей компании, даже если выбранные продукты не могут быть оптимальным решением проблемы. Это заставляет инженеров-программистов адаптировать и настраивать эти продукты, чтобы преодолеть их ограничения, что в свою очередь вызывает создание обходных путей и генерирует технический долг.

Согласование предварительно выбранного альтернативного архитектурного решения с соответствующими заинтересованными сторонами является одной из сложных задач по обслуживанию технического долга, которую приходится решать лицам, принимающим решения в проекте. Решение, связанное с возникновением долгов, может повлиять на скорость разработки в следующих итерациях из-за выплаты процентов по техническому долгу и стать причиной ошибок в будущих выпусках, которые повлияют на способность к масштабированию и внешнее качество продукта. Менеджеры без соответствующего технического опыта с трудом согласуют запланированные работы команды по обслуживанию технического долга, поскольку не видят долгосрочные затраты, связанные с ним. Поэтому инженеры-программисты, представляя план по перепроектированию решения, связанного с возникновением долгов, в поддерживаемое решение, определенное в начале процесса проектирования, обязаны сделать как можно более ясными для менеджеров риски принятия архитектурного решения, приносящего технический долг. В идеале люди, контролирующие бюджет проекта, должны хорошо разбираться в технических вопросах, тогда они будут знать, на какой риск они пойдут. Особенно понятными для бизнеса будут риски, которые могут привести к его остановке, – падающие сайты, системы поддержки клиентов, т. е. риски, связанные с разработкой front-end компонентов, которые делают бизнес «видимым» для

клиентов, но в то же время подвержены наибольшим динамическим изменениям, поскольку содержат больше всего неопределённости. Именно в этих продуктах технический долг накапливается быстрее всего.

Наиболее трудно определить технический долг, полученный в результате архитектурных решений, случайно приводящих к такому долгу. В некоторых исследованиях указываются наиболее частые причины, влекущие появление такого долга [1]:

• *неглубокий архитектурный анализ из-за гибких и экспериментальных подходов.* Хотя гибкий подход эффективен для разработки и продажи продуктов, но анализ рынка, проектирование и разработка продукта ведутся параллельно и в короткий период времени. Поэтому увеличивается риск пропуска архитектурно-значимых требований и, как следствие, введение архитектурных решений, связанных с долгами, в следующих итерациях. В качестве неожиданных архитектурно-значимых требований могут быть не только функциональные требования (например, добавление новой функции к продукту), но и атрибуты качества, которые не имели достаточного приоритета при первоначальном проектировании архитектуры программного обеспечения. Например, неожиданное требование повторного использования компонента, изначально разработанного для использования одной командой, а затем необходимость использования того же компонент другими командами, которые также участвуют в разработке продукта, но уже в другом месте. Это технический долг, связанный с необходимостью поделиться компонентом с кем-то ещё;

• *отсутствие архитектурных навыков*. Для agile-проектов характерны небольшие продуктовые группы, состоящее из 7–20 человек, в которых разделение между архитектурой и программированием очень размыто. Каждый программист – это своего рода архитектор, независимо от того, хочет он этого или нет. Кроме того, официальный архитектор – это, как правило, разработчик-практик, у которого проектные навыки могут быть существенно ниже, чем навыки разработчика. Поэтому недостаточный уровень архитектурных навыков может быть причиной создания неоптимальных АР и, как следствие, возникновение непреднамеренного технического долга;

• *отсутствие подходящих альтернативных технологий*. Из-за отсутствия подходящих альтернатив при выборе технологического инструментария (например, библиотеки или фреймворка) инженеры-программисты иногда вынуждены выбирать решение с потенциальными недостатками, которые могут поставить под угрозу ремонтопригодность системы при будущих изменениях и нести технический долг;

• *отсутствие поддержки со стороны поставщиков технологий*. Поставщики технологий могут прекратить развитие и поддержку устаревших технологических решений. Использование таких устаревших инструментов при разработке ПО может создавать непреднамеренный технологический долг, который трудно преодолеть. Для предотвращения подобного архитектурного технического долга необходимо отслеживать приоритеты вендоров в отношении поставляемых инструментов;

• *отсутствие документации с обоснованием проектного решения.* В agile-проектах нередки изменения в составе группы разработки. При этом фиче-команды сосредоточены на

создании нового функционала и мало времени уделяют документированию обоснования проектных решений. Поэтому новые члены команды не имеют возможности понять причины существующих АР в полной мере. Впоследствии это приводит к внесению изменений, которые непреднамеренно отклоняются от намеченной цели архитектуры и затрудняют реализацию, например, масштабирования разработки, поскольку архитектурная целостность является ключевым свойством системы для её будущих изменений.

Из вышесказанного следует, для создания в agile-проектах программного продукта, имеющего потенциал развития, а также для эффективного управления архитектурным техническим долгом целесообразно соблюдать ряд принципов:

• придерживаться инкрементального принципа создания архитектуры разрабатываемого ПО, выращивая её от спринта к спринту, стремясь сделать её не всеохватывающей, а лёгкой для рефакторинга, широко используя механизмы обеспечения гибкости: интерфейсы, полиморфизм, шаблоны проектирования, структуры внедрения зависимостей, указатели на функции и многое другое (https://clck.ru/dVnZM);

• подбирать в качестве архитектора программиста-эксперта, имеющего высокую квалификацию проектирования, способного выбрать сбалансированное архитектурное решение на каждом этапе создания кода;

• выполнять регулярное ревью кода для ранней идентификации неявного технического долга, планировать рефакторинг и выплату технического долга, разбивая его на малые задачи, чётко обозначенные целями и сроками. Это позволит не сильно замедлить скорость поставки продукта, сохранить мотивацию команды на исправление написанного кода, а также снижать объем технического долга или хотя бы держать его на фиксированном уровне;

• при согласовании с бизнесом затрат на рефакторинг и работ по техническому долгу говорить на его языке: объяснять финансовые риски и приводить примеры из жизни. Поскольку бизнес имеет потенциал прибыли от развития продукта гораздо больший, чем затраты на обслуживание технического долга, то для бизнеса борьба с техническим долгом – это не убытки, а отложенная ценность, способствующая повышению выгоды;

• в каждом спринте после реализации функционала регулярно проводить воркшопы по SAD (System Architecture Documentation) для создания адаптивной документации по системной архитектуре, что позволит участникам команды лучше понять разработанную архитектуру и находить решения для её улучшения.

В agile –проекте, как, впрочем, и в других проектах, основными принципом, позволяющими, удерживать технический долг на низком уровне, является работа на опережение, не позволяющая проблеме разрастаться.

Литература

1. Soliman M., Avgeriou P., Li Y. Architectural design decisions that incur technical debt – An industrial case study // Information and Software Technology. 2021. Vol. 139, 106669. https://doi.org/10.1016/j.infsof.2021.106669