УДК 004.413
DOI: 10.17587/prin.13.311-321

**Д. О. Змеев**, ассистент, denis.zmeev@accounts.tsu.ru,
**О. А. Змеев**, д-р физ.-мат. наук, проф. кафедры, ozmeyev@gmail.com,
**Л. С. Иванова**, ассистент, lidiya.ivanova@persona.tsu.ru,
Томский государственный университет

# Практика работы с антипаттернами для Essence Practice Library

*Представлено расширение для библиотеки практик языка Essence в виде практики работы с антипаттернами. Для представления антипаттернов в системе предложены субальфа Antipattern, ее состояния и контрольные точки. Для фиксации данных об антипаттерне предложены рабочий продукт Antipattern Report и его уровни детализации. Для проведения анализа архитектуры системы предложена активность Inspect Architecture. Активность Fix Architecture представляет действия по устранению недостатков архитектуры. Анализ кода представлен в виде активности Review the Code, а исправление найденных в ходе анализа недостатков — в виде активности Refactor the Code. Проанализировано влияние субальфы Antipattern на состояние альфы Software System. Даны рекомендации о проведении предложенных активностей.*

***Ключевые слова:*** *антипаттерн, код-ревью, недостаток кода, практика, разработка, рефакторинг, управление проектом, Essence, Practice Library, SEMAT*

## Введение

Отрасль разработки программного обеспечения (ПО) является одной из самых быстроразвивающихся, массовый переход на удаленную работу в условиях пандемии ускорил это развитие [1]. За последние годы был предложен ряд методологий по управлению проектами в сфере разработки ПО. Однако статистика, представленная Standish Group в новом отчете CHAOS Report [2], показывает, что только около 31 % проектов завершаются успешно, 50 % вызывают трудности, а 19 % проектов являются провальными, причем данные значения существенно не изменились с 2010 г. [2—4]. Поскольку статистика успешности проектов не улучшается, необходимо рассмотреть различные негативные факторы, отрицательно влияющие на успешность проектов. Такими факторами могут быть риски управления, меняющиеся требования, а также различные проблемы, возникающие в процессе разработки ПО — баги (ошибки) и антипаттерны [5] — неудачные архитектурные решения часто возникающих задач проектирования и/или реализации.

Инициатива SEMAT (*Software Engineering Method And Theory*) [6] является новой попыткой систематизировать имеющиеся методологии и практики программной инженерии, создав единое Ядро и язык описания Essence [7], позволяющий описать новые практики понятным практически любому человеку способом. Создание такого языка по сути является созданием метамодели для описания ранее придуманных и новых методологий. Авторы языка Essence опубликовали Библиотеку Практик (*Practice Library*) [8], в которую вошли такие известные методологии (в терминах Essence они носят название "методы"), как Унифицированный процесс [9] и гибкая методология разработки Agile [10].

Тема расширения языка Essence и Библиотеки Практик уже была затронута в литературе [11—14]. Например, авторы работы [12] предприняли попытку описать подход TOGAF (*The Open Group Architecture Framework* [15]) в терминах Essence. Авторы инициативы SEMAT продолжают дорабатывать Библиотеку, например, работа [11] расширяет Библиотеку практикой применения вариантов использования. Интерес вызывает работа [16], авторы которой предложили систему измерений для альфы Software System (Программная Система) для оценки качества продукта и получения данных о здоровье и прогрессе проекта. В работе [13] авторы предлагают усилить стандарт с помощью унификации используемой терминологии.

Авторы работ [12, 14] отмечают необходимость расширения Ядра Essence и Библиотеки Практик, кроме того, автор книги [17] отмечает необходимость рассмотрения косвенных убытков и причин их возникновения (которыми являются и анти-

паттерны) в рамках результатов работы инициативы SEMAT.

Поскольку Библиотека Практик по-прежнему крайне ограничена, а большинство работ предлагают лишь расширения существующих элементов, возникает необходимость добавления новых методов и практик. Работа с антипаттернами и другими недостатками кода в рамках процессов по разработке ПО может быть представлена в виде практики, которая может использоваться как отдельно, так и в составе новых или уже описанных в Библиотеке методов. Поскольку антипаттерны оказывают на систему отрицательное воздействие, препятствуют ее развитию и могут стать косвенной причиной гибели проекта, разработка данной практики представляет не только практический, но и исследовательский интерес.

Целью работы, результаты которой представлены в статье, является разработка практики работы с антипаттернами в терминах языка Essence. Актуальность темы состоит в необходимости расширения Библиотеки Практик в связи с высоким негативным влиянием антипаттернов на проект. Научная новизна выполненного авторами исследования определяется отсутствием в Библиотеке Практик каких-либо упоминаний антипаттернов и недостатков кода.

При разработке практики использовался опыт авторов книг по работе с антипаттернами и недостатками кода [5, 18, 19].

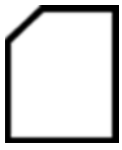
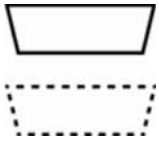
## 1. Элементы языка Essence

Ядро Essence состоит из практик — способов организации специализированных работ в рамках процесса разработки программного продукта. Практики, использованные командой для проекта, объединяются в методы. В табл. 1 приведены краткий обзор элементов языка и их графическая нотация. Описание практики работы с антипаттернами с использованием данных элементов представлено в разд. 3.

В Ядро Essence также входят конкретные Альфы, например, Software System — система, состоящая из ПО, аппаратного обеспечения и данных. Полный перечень альф Ядра представлен на рис. 1.

*Таблица 1*

**Основные элементы языка Essence**

| Пиктограмма | Название | Определение |
|---|---|---|
|  | Альфа (*Alpha*) | Изначально — акроним от английского Abstract Level Progress Health: семантически фиксирует некоторую бизнес-область метода или практики, состояние которой необходимо отслеживать в процессе реализации проекта |
|  | Состояние Альфы (*Alpha State*) | Сущность, фиксирующая прогресс изменений Alpha (альфы) в ходе реализации метода или практики. Состоит из набора истинных/ложных утверждений, называемых Checkpoint |
|  | Пространство Активностей (*Activity Space*) | На высоком абстрактном уровне описывает виды деятельности, необходимые в процессе реализации метода или практики (с точки зрения объектно-ориентированного программирования в некотором смысле являются абстрактными классами или интерфейсами) |
|  | Активность (*Activity*) | Моделирует детальные и конкретные действия, которые необходимо совершить в процессе выполнения проекта, чтобы добиться прогресса, предписываемого практикой (с точки зрения объектно-ориентированного программирования в некотором смысле являются аналогами объектов или классов, реализующих интерфейсы) |
|  | Рабочий продукт (*Work Products*) | Моделирует сущности, необходимые для описания результатов, которые создаются в ходе выполнения активностей или задач в тех или иных практиках |
|  | Уровень детализации (*Level of detail*) | Сущность, фиксирующая прогресс, достигнутый рабочим продуктом. Обычно выделяются два подвида: обязательный — со сплошной рамкой — и необязательный — со штриховой рамкой. Фактически выполняют роль, аналогичную состояниям Alpha |

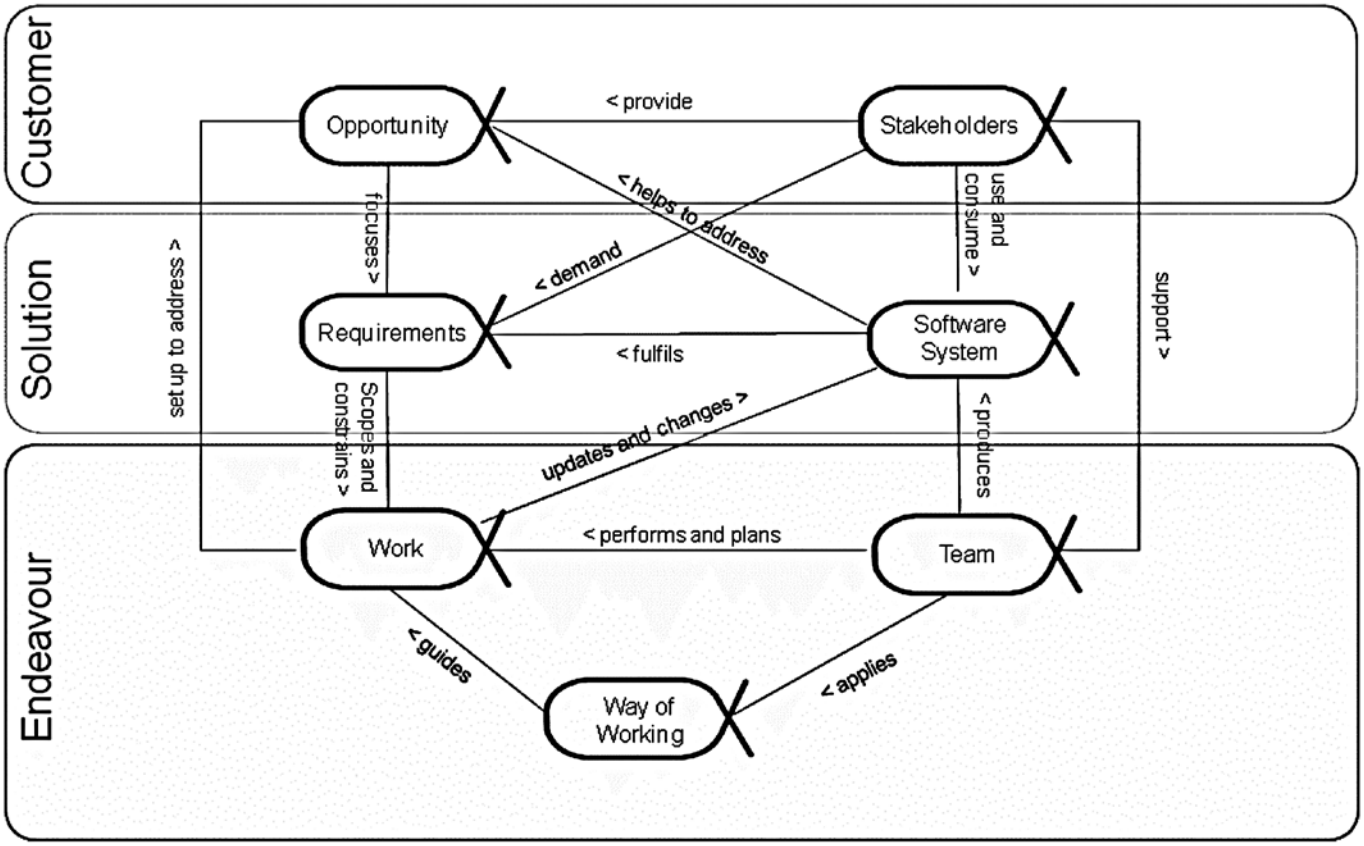



**Рис. 1. Альфы Ядра Essence [3]**

С точки зрения работы с антипаттернами наибольший интерес среди активностей Ядра представляет активность Implement the System — создать систему, внедрив, протестировав и интегрировав один или несколько ее элементов; включает в себя исправление ошибок и модульное тестирование.

Опираясь на Ядро Essence и Библиотеку Практик, авторами данной статьи была составлена собственная практика работы с антипаттернами в рамках процесса разработки программного продукта.

## 2. Работа с антипаттернами

В рамках данной статьи термином антипаттерн объединены различные понятия, обозначающие недостатки, возникшие на этапах проектирования и/или реализации программного продукта (antipattern [5], code smell [18]). Примерами антипаттернов являются как "классические" Божественный объект [5], Стрельба дробью [18] и Полтергейст [5], так и недостатки исходного кода, которые рассмотрены в посвященной рефакторингу литературе: Длинный метод [5], Условная сложность [5] и др. Следует уточнить, что ошибки ("баги" и другие примеры некорректного поведения ПО) в контексте данной работы не рассматриваются.

В процессе работы над программным продуктом антипаттерны могут быть обнаружены как во время различных обзоров исходного кода и документации (аудит, код-ревью), так и при добавлении нового или изменении существующего функционала. Следует отметить, что тема автоматизации процесса поиска антипаттернов актуальна [20], поскольку процесс поиска таких недостатков является достаточно трудоемким, особенно, если речь идет о крупных программных продуктах.

Найденный антипаттерн становится частью так называемого технического долга [21]. В дальнейшем, если затраты на устранение недостатка не превышают возможные финансовые потери от его присутствия в системе, найденный антипаттерн устраняется с помощью процедуры рефакторинга. Следует также учитывать, что, по мнению авторов книги [18], "важным предварительным условием проведения рефакторинга является наличие надежных тестов".

## 3. Практика работы с антипаттернами

Практика работы с антипаттернами представляет собой рекомендации по работе с различными недостатками проектирования и реализации, которые могут быть найдены в диаграмме или в ис-

ходном коде программной системы. В нее вошли субальфа Антипаттерн, рабочий продукт Отчет об Антипаттерне и активности Проверить архитектуру, Исправить архитектуру, Проверить код, Провести рефакторинг кода.

### 3.1. Субальфа Антипаттерн

Для представления антипаттерна и работы с ним в рамках процесса разработки программного продукта авторами настоящей статьи предлагается субальфа (альфа второго уровня) Антипаттерн (*Antipattern*) — распространенное неэффективное решение часто возникающей задачи проектирования и/или реализации. Антипаттерн не является причиной некорректной работы приложения, однако может быть причиной будущих проблем модификации и повторного использования. Далее в работе при использовании термина Антипаттерн в смысле субальфы термин будет употреблен с заглавной буквы, в контексте антипаттерна какой-то системы — с прописной.

Антипаттерн является субальфой, представленной в Ядре альфы Software System (рис. 2), он препятствует ее развитию, поскольку наличие в Software System антипаттернов может отрицательно сказаться на ее способности к прогрессу. Степень отрицательного влияния оценить достаточно сложно.

На протяжении всего жизненного цикла в ПО могут возникать антипаттерны — неудачные решения, отрицательно влияющие на повторное использование и расширяемость системы, а также удобочитаемость кода. Такие недостатки не обнаруживаются в результате тестирования и являются техническим долгом, который нужно устранять.

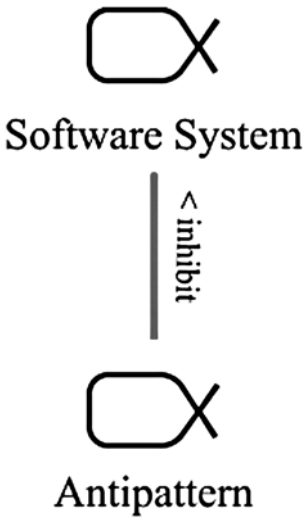



**Рис. 2. Субальфа Антипаттерн**

Антипаттерны могут препятствовать развитию Software System, в которой они присутствуют. Они могут быть обнаружены и идентифицированы во всех состояниях Software System, кроме Retired, поскольку в данном состоянии работа с разрабатываемой системой полностью прекращается. В состоянии Architecture Selected антипаттерны могут быть обнаружены в документации к проекту (диаграммах). В состояниях Demonstrable — Operational антипаттерны могут быть обнаружены в документации и исходном коде системы. Своевременное устранение антипаттернов увеличивает скорость разработки и исправления ошибок, т. е. способствует более быстрому переходу в следующее состояние.

Состояния субальфы Антипаттерн и соответствующие им контрольные точки представлены в табл. 2 (поскольку большинство опубликованных в настоящий момент описаний практик существуют только на английском языке, в тексте данной работы приведены русские описания, но в расширенной английской аннотации приведен английский вариант).

Как показано в табл. 2, Антипатерн проходит через состояния подозревается, идентифицирован, метод рефракторинга определен, исправлен, закрыт. Если детально описывать жизненный цикл Антипатерна, складывается описанная ниже последовательность.

Сначала, в результате ревью, исправления бага, разработки нового функционала может возникнуть подозрение на антипаттерн. Область, в которой обнаружено подозрение на антипаттерн, помечается для анализа. В результате анализа антипаттерн идентифицируется: определяются его границы, тип, зависимости. Определяется степень влияния антипаттерна на систему. В случае, если влияние незначительно (устранение антипаттерна дороже, чем отрицательный урон от его наличия в системе), работу по устранению можно прекратить. В зависимости от типа антипаттерна определяется способ его устранения. Затем проводится процедура рефакторинга, антипаттерн устраняется. С помощью тестирования и ревью определяется, что антипаттерн устранен, новых недостатков не возникло, следовательно, антипаттерн можно считать закрытым.

Состояние субальфы Антипаттерн не зависит от состояния Software System. Если поддержка и расширение функционала системы не планируется — антипаттерны могут остаться в состоянии подозревается или идентифицирован. Аналогично для отдельных элементов системы: если не планируется их поддержка, расширение или повторное использование — найденные в данных областях антипаттерны можно не устранять.

*Таблица 2*

**Состояния субальфы Антипаттерн**

| Состояние | Описание | Контрольные точки |
|---|---|---|
| Обнаружен | Выделена часть системы, которая содержит антипаттерн | Обнаружены признаки некачественного проектирования или реализации системы.<br>Определены границы области низкого качества в системе.<br>Определены рабочие продукты, связанные с низкокачественной частью системы |
| Идентифицирован | Определены местоположение антипаттерна в системе и его тип | Определены элементы системы, входящие в антипаттерн.<br>Определен тип антипаттерна.<br>Выявлены зависимости системы от элементов, составляющих антипаттерн.<br>Проведена оценка влияния антипаттерна на систему |
| Метод рефакторинга определен | Проведен выбор наиболее подходящего метода исправления (рефакторинга) антипаттерна | Определены возможные варианты рефакторинга.<br>Проведена оценка стоимости выявленных вариантов рефакторинга.<br>Выбран наиболее подходящий вариант рефакторинга |
| Исправлен | Антипаттерн удален из системы | Элементы системы, входящие в антипаттерн, полностью покрыты тестами.<br>Проведен рефакторинг антипаттерна.<br>Тесты подтверждают, что работоспособность системы была сохранена.<br>Обновлены Рабочие продукты, связанные с измененной частью системы |
| Закрыт | Отсутствие антипаттерна в системе подтверждено | Были проведены тесты, обзоры или другие соответствующие действия, чтобы гарантировать, что антипаттерн исправлен или не является фактическим дефектом или недостатком.<br>Завершены мероприятия по управлению антипаттерном.<br>Технический долг уменьшился |

### 3.2. Рабочий продукт Отчет об Антипаттерне

Для работы с информацией об Антипаттерне в рамках проекта предлагается рабочий продукт (*Work Product*) Отчет об Антипаттерне (*Antipattern Report*). Отношения между рабочим продуктом и альфами представлены на рис. 3.

Уровни детализации данного артефакта представлены в табл. 3. Отчет об Антипаттерне описывает местонахождение, тип антипаттерна, а также содержит информацию о работах, проведенных по его устранению. При использовании различных программных продуктов для управления проектом Отчет об Антипаттерне может быть представлен в виде задачи (*Task, Issue*), записи в Wiki или Readme-файлах проекта.

В качестве "частей системы" могут выступать: класс, совокупность связанных классов, слой, выделяемый в системе, и другие элементы, являющиеся составными элементами антипаттерна.

### 3.3. Активности

Для формализации процедур по изменению состояния субальфы Антипаттерн предлагаются следующие Активности, представленные на рис. 4.

Рассмотрим отдельные активности, представленные на рис. 4.

1. *Inspect Architecture (Проверить архитектуру)* — проверить архитектуру системы на наличие антипаттернов (рис. 5).

Данную активность следует проводить для всех частей системы.

Проверить архитектуру необходимо для того, чтобы:

- проверить качество проектирования системы;

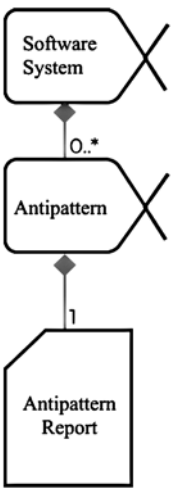



**Рис. 3. Альфы и рабочий продукт Antipattern Report (Отчет об Антипаттерне)**

*Таблица 3*

**Уровни детализации рабочего продукта Отчет об Антипаттерне**

| Уровень детализации | Описание | Контрольные точки |
|---|---|---|
| Обнаружено наличие антипаттерна | В системе обнаружен недостаток | Определены части системы, входящие в антипаттерн |
| Определены характеристики антипаттерна | Обнаруженный в системе недостаток локализован, описан и проанализирован | Определен тип антипаттерна |
| Определено влияние антипаттерна | Проведена оценка влияния антипаттерна и принято решение о целесообразности его устранения | Проведена оценка стоимости исправления антипаттерна.<br>Проведена оценка возможных потерь, связанных с наличием антипаттерна.<br>Антипаттерн признан обязательным к исправлению |
| Антипаттерн устранен (необязателен) | Недостаток, найденный в системе, полностью устранен, все связанные работы успешно завершены | Проведен рефакторинг.<br>Тестирование показало корректность работы частей системы, входящих в антипаттерн.<br>Деятельность, связанная с устранением антипаттерна, завершена |

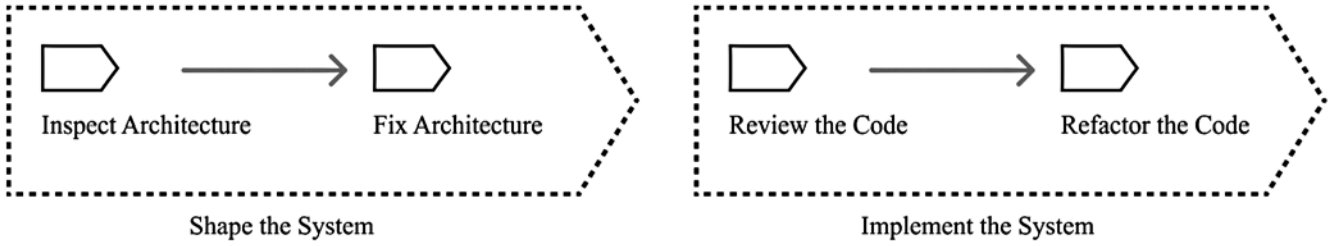



**Рис. 4. Активности по работе с субальфой Антипаттерн**

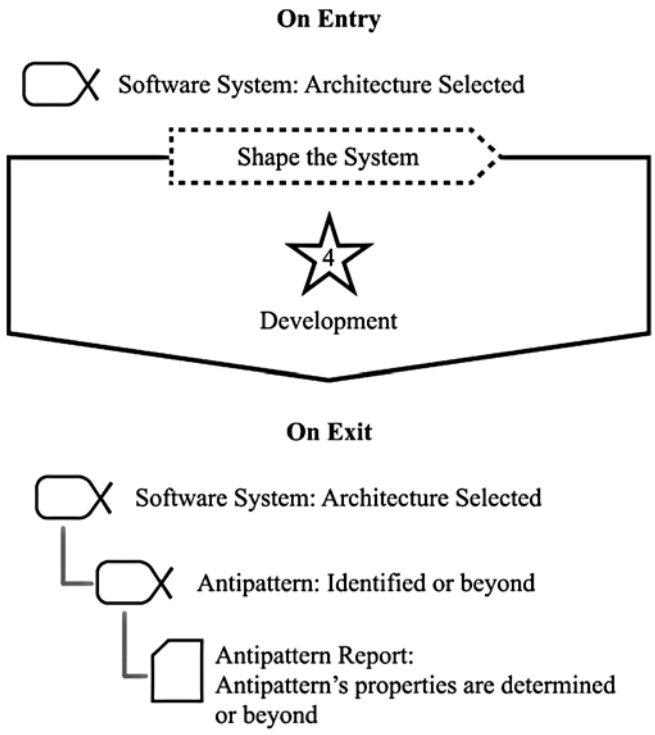



**Рис. 5. Активность Inspect Architecture**

- проверить отсутствие антипаттернов;
- проверить отсутствие багов;
- увеличить осведомленность команды о проекте.

Критерии завершения активности: система проверена полностью. Проверяющие члены команды согласны, что найденные проблемные вопросы устранены, либо признаны незначительными.

Описываемая в данной работе практика может быть использована в составе методов, например, при включении практики антипаттернов в метод *Essential Unified Process*, активность *Inspect Architecture* будет взаимодействовать с субальфами Компонент (*Component*) и Архитектура (*Architecture*) (рис. 6). В результате активности также могут быть выявлены баги (расхождения с требованиями к системе).

2. *Fix Architecture (Исправить архитектуру)* — устранить антипаттерны в архитектуре системы (рис. 7).

Данная активность позволяет устранить существующий антипаттерн, а также улучшить качество архитектуры системы, что положительно скажется на повторной используемости и расширяемости.

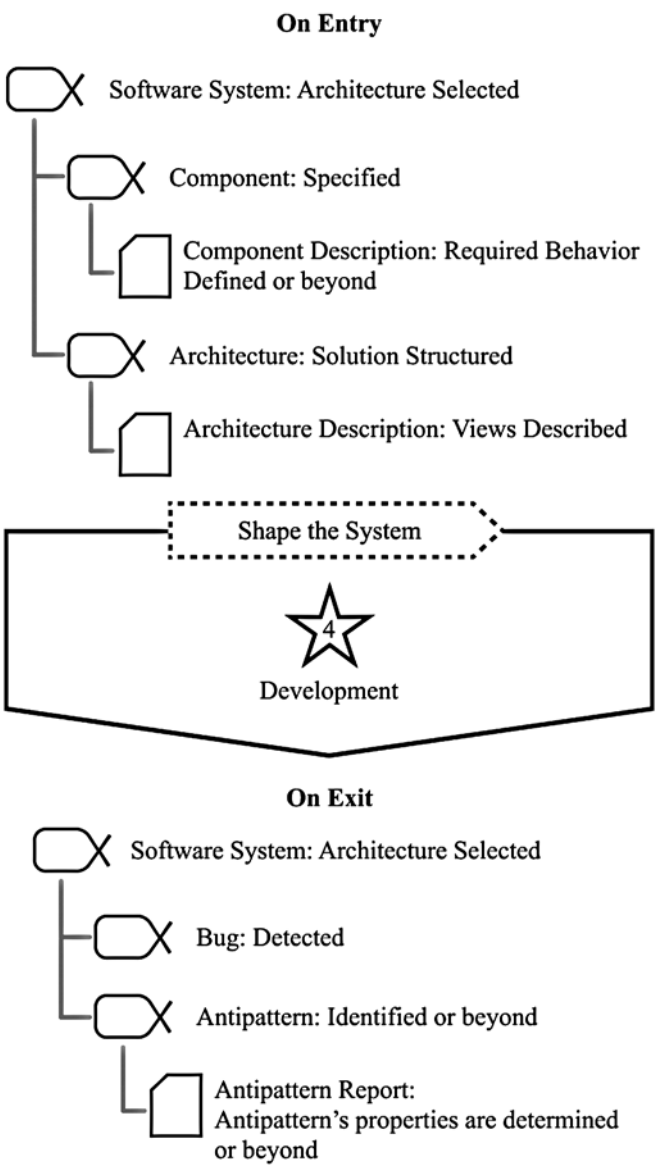



**Рис. 6. Активность Inspect Architecture в методе Essential Unified Process**

Критерий завершения данной активности — Антипаттерн, найденный ранее, устранен. Проверяющие члены команды согласны, что найденные проблемные вопросы устранены, либо признаны незначительными.

В составе метода *Essential Unified Process* данная активность может использовать и изменять рабочие продукты *Component Description* и *Architecture Description*. В результате ее проведения может измениться состояние субальфы Баг на Fixed (ошибка может быть исправлена).

3. *Review the Code (Проверить код)* — провести код-ревью исходного кода системы (рис. 8).

Код-ревью необходимо, чтобы:

- проверить качество исходного кода;
- проверить отсутствие антипаттернов;

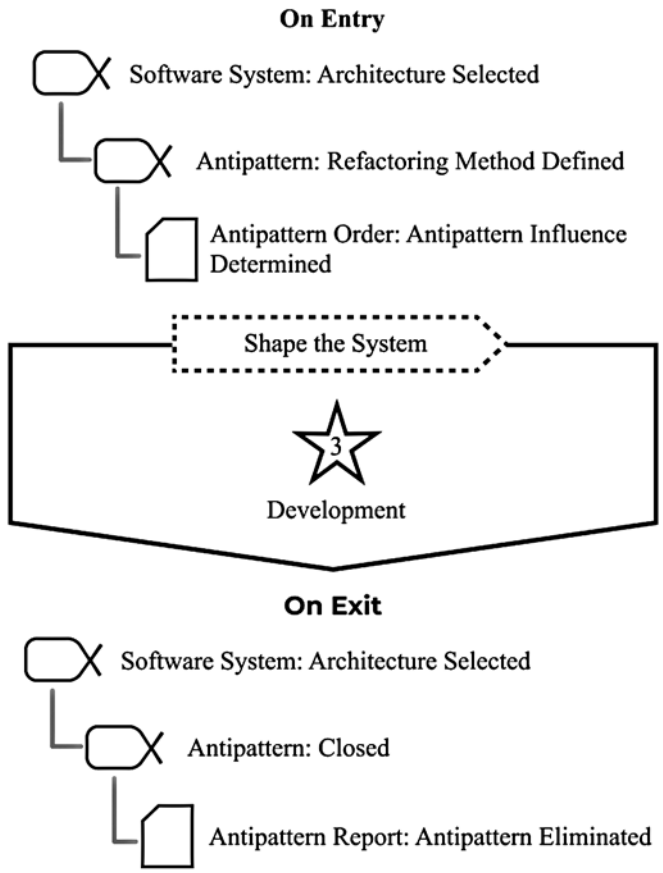



**Рис. 7. Активность Fix Architecture**

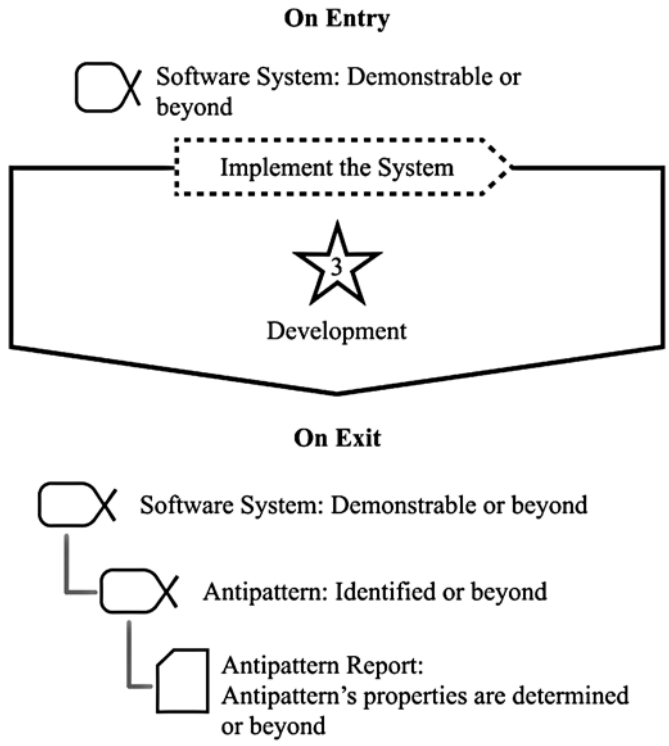



**Рис. 8. Активность Review the Code**

- проверить отсутствие багов, обнаруживаемых в исходном коде;
- увеличить осведомленность команды о проекте.

Данную активность следует проводить для всех компонентов системы.

Критерии завершения активности: система проверена полностью. Проверяющие члены команды согласны, что найденные проблемные вопросы устранены, либо признаны незначительными.

Чаще всего методика проведения код-ревью определяется правилами, установленными в конкретной команде разработчиков. В литературе [22, 23] предлагаются перечисленные далее варианты проведения процедуры код-ревью.

- Проверка кода (*Code Inspections*) [23]. Проверки кода проводятся после завершения проектирования участка программной системы, после завершения реализации участка системы и после создания модульных тестов для участка системы. Данный подход подробно описан в статье [23].
- Современный метод (*Modern Code Review*) [22]. Для автоматизации процесса могут быть использованы различные инструменты, такие как Gerrit [24], CodeFlow [25], а также встроенные средства GitHub [26], GitLab [27]. Один или несколько ревьюеров проверяют изменения кодовой базы, проведенные членом команды перед добавлением изменений в репозиторий. Процесс можно разделить на несколько итераций, которые продолжаются, пока не будут устранены все значимые из найденных недостатков и все обнаруженные баги. Подробное описание варианта реализации подхода в компании Google можно найти в работе [22].

4. *Refactor the Code (Провести рефакторинг кода)* — провести рефакторинг исходного кода системы, не изменяя ее поведение (рис. 9).

Проведение рефакторинга необходимо, чтобы:

- устранить технический долг;
- улучшить читаемость кода;
- улучшить расширяемость системы;
- улучшить способность кода к повторному использованию.

Данная активность позволяет устранить существующий антипаттерн, а также улучшить качество исходного кода компонента, что положительно скажется на повторной используемости, читаемости, расширяемости исходного кода. В процессе рефакторинга могут быть найдены и устранены ошибки системы.

Критерий завершения данной активности — антипаттерн устранен без нарушения работоспособности системы: недостаток, найденный ранее,

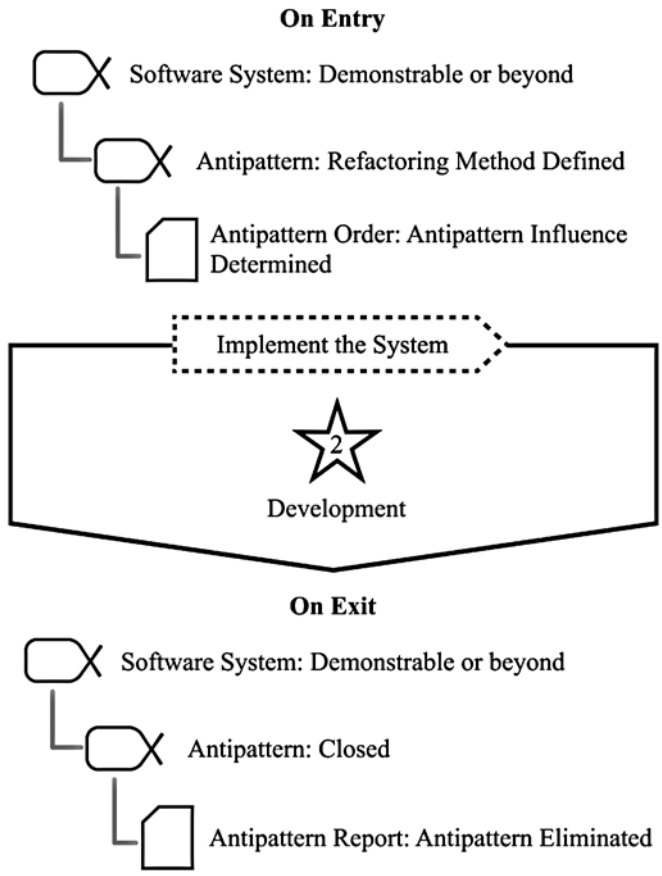



**Рис. 9. Активность Refactor the Code**

устранен. Тесты, покрывающие данный фрагмент системы, успешно завершаются. Внешнее поведение системы не изменилось. Качество кода данного фрагмента системы улучшилось.

Применяемые способы рефакторинга зависят от вида найденного недостатка кода. Различные способы рефакторинга подробно описаны в литературе [18, 19]. Согласно классической процедуре проведения рефакторинга [18], для проверки успешности проведения рефакторинга необходимо наличие автоматических тестов. Для работы с тестами в Ядре Essence присутствует пространство активностей Протестировать Систему (*Test the System*).

## Заключение

Предложенная в настоящей статье практика позволяет формализовать процедуру по работе с антипаттернами при разработке ПО. Возможна комбинация предложенного подхода с другими практиками.

Предложенная субальфа Антипаттерн является одним из примеров субальфы, которая может оказывать отрицательное влияние на развитие проекта. Рабочий продукт, подтверждающий на-

личие в Software System неудачных решений — Отчет об Антипаттерне, — является свидетельством того, что в будущем могут появиться различные проблемные вопросы, связанные с расширением и развитием разрабатываемой системы. Это обстоятельство может стать основанием для переработки существующего исходного кода или переосмысления принятых архитектурных решений.

Результаты работы планируется использовать в расширении системы поддержки принятия решений [28], поскольку сам факт существования антипаттернов в системе может являться флагом того, что менеджер ошибается в своих предположениях относительно текущего состояния проекта.

**Список литературы**


1. **Digital** Economy Report 2021. Cross-border data flows and development: For whom the data flow. URL: https://unctad.org/system/files/official-document/der2021_en.pdf
2. **Portman H.** Review Standish Group — CHAOS 2020: Beyond Infinity. URL: https://hennyportman.wordpress.com/2021/01/06/review-standish-group-chaos-2020-beyond-infinity/
3. **Wojewoda S., Hastie S.** Standish Group 2015 Chaos Report — Q&A with Jennifer Lynch. URL: https://www.infoq.com/articles/standish-chaos-2015/
4. **The Standish** Group — CHAOS Report. Project Smart. URL: https://www.projectsmart.co.uk/white-papers/chaos-report.pdf
5. **Brown W. J., Malveau R. C., McCormic III H. W., Mowbray T. J.** AntiPatterns: Refactoring Software, Architectures. and Projects in Crisis. New York: John Wiley & Sons, 1998.
6. **Пак Дж., Якобсон И., Майбург Б., Джонсон П.** SEMAT вчера, сегодня и завтра: перспективы промышленного использования // Программная инженерия. 2014. № 11. С. 6—16.
7. **Jacobson I., Ng P.-W., McMahon P. E., Spence I., Lidman S.** The Essence of Software Engineering: Applying the SEMAT Kernel. Addison-Wesley, 2013. 224 p.
8. **Practice** Library. URL: https://practicelibrary.ivarjacobson.com/start
9. **Scott K.** The Unified Process Explained, 1st edition. Addison-Wesley Professional, 2001. 208 p.
10. **Beck K., Beedle M., van Bennekum A.** et al. Agile-манифест разработки программного обеспечения. URL: https://agilemanifesto.org/iso/ru/manifesto.html
11. **Jacobson I., Spence I., Bittner K.** USE-CASE 2.0: The Guide to Succeeding with Use Case. Ivar Jacobson International, 2011. 55 p.
12. **Munera D., Villa G. F.** A brief TOGAF description using SEMAT Essence Kernel. URL: https://www.researchgate.net/publication/335855005_A_brief_TOGAF_description_using_SEMAT_Essence_Kernel
13. **Zapata-Jaramillo C., Henao-Roqueme A.** A proposal for improving the Essence standard by using terminology unification // Ingenieria. 2021. Vol. 26, No. 2. P. 213—232. DOI: 10.14483/23448393.16428.
14. **Simonette M., Spina E.** Software & Systems Engineering Interplay and the SEMAT Kernel // The Voice of the Systems - The Journal Of The Israeli Systems Engineers. 2018. Issue 22. P. 6—20.
15. **The TOGAF** Standard, Version 9.2 Overview. URL: https://www.opengroup.org/togaf
16. **Perdomo Charry W., Zapata C.** Software quality measures and their relationship with the states of the software system alpha // Ingeniare. 2021. Vol. 29. P. 346—363. DOI: 10.4067/S0718-33052021000200346.
17. **Jones C.** Software Development Patterns and Antipatterns. CRC Press, 2022. 512 p. DOI: 10.1201/9781003193128.
18. **Фаулер М., Бек К., Брант Д.** и др. Рефакторинг: улучшение проекта существующего кода. СПб: Альфа-книга, 2017. 448 с.
19. **Кириевски Д.** Рефакторинг с использованием шаблонов. М.: Вильямс, 2016. 400 с.
20. **Змеев О. А., Иванова Л. С.** Поиск артефактов проектирования. Обзор подходов // Вестник Томского государственного университета. Управление, вычислительная техника и информатика. 2015. № 2 (31). С. 81—90.
21. **Cunningham W.** The WyCash Portfolio Management System // Addendum to the proceedings on Object oriented programming systems, languages, and applications. 1992. P. 29—30.
22. **Sadowski C., Söderberg E., Church L., Sipko M., Bacchelli A.** Modern code review: a case study at google // Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP'18). Association for Computing Machinery, New York, NY, USA, 2018. P. 181—190. DOI: 10.1145/3183519.3183525.
23. **Fagan M.** Design and code inspections to reduce errors in program development // IBM Systems Journal. 1976. Vol. 15, No. 3. P. 182—211.
24. **Gerrit** Code Review. URL: https://www.gerritcodereview.com
25. **CodeFlow**. URL: https://www.getcodeflow.com
26. **GitHub.** URL: https://github.com
27. **GitLab.** URL: https://about.gitlab.com
28. **Змеев Д. О., Иванова Л. С., Рафикова Р. Р.** О представлении прогресса проекта по разработке программного обеспечения в форме динамической байесовской сети // Информационные технологии и математическое моделирование (ИТММ-2020). Материалы XIX Международной конференции имени А. Ф. Терпугова. Томск, 2021. С. 291—297.


# Antipattern Practice for Essence Practice Library


**D. O. Zmeev,** denis.zmeev@accounts.tsu.ru, **O. A. Zmeev,** ozmeyev@gmail.com,
**L. S. Ivanova**, lidiya.ivanova@persona.tsu.ru,
Tomsk State University, Tomsk, 634050, Russian Federation

*Corresponding author:*

**Lidiya S. Ivanova,** Assistant, Tomsk State University, 634050, Tomsk, Russian Federation
E-mail: lidiya.ivanova@persona.tsu.ru



*Received on April 03, 2022*
*Accepted on May 26, 2022*

*The Essence graphical representation language allows to describe various project management practices in software development. At the moment, the Practice Library describes the most popular development methodologies, but work with various risks, such as code smells or antipatterns, which may be cause of future problems, is not represented.*

*This article presents an extension for the Practice Library of the Essence language in the form of a practice for working with antipatterns. To represent antipatterns in the system, the Antipattern subalpha, its states and checkpoints are proposed. Antipattern's states and checkpoints:*

*1. Detected:*
- *Signs of poor-quality design or implementation of the Software System have been found.*
- *The boundaries of the low-quality area in the Software System have been defined.*
- *Work Products associated with the inferior part of the Software System have been defined.*

*2. Identified:*
- *The Software System Elements included in the Antipattern have been determined.*
- *The type of antipattern has been determined.*
- *Dependencies of the Software System on the antipattern elements have been determined.*
- *The impact of the antipattern on the Software System has been estimated.*

*3. Refactoring Method Defined:*
- *Possible options for refactoring have been identified.*
- *The cost of the refactoring options has been assessed.*
- *The most suitable refactoring method has been selected.*

*4. Fixed:*
- *The Software System elements included in the antipattern are completely covered by tests.*
- *The antipattern has been refactored.*
- *Tests confirm that the Software System remains operational.*
- *The Work Products related to the changed part of the Software System have been updated.*

*5. Closed:*
- *Tests, reviews or other appropriate activities have been undertaken to ensure that the antipattern has been corrected or shown not to actually be a fault or flaw.*
- *The antipattern management has been finalized.*
- *Technical debt has decreased.*

*To record data about an antipattern, the work product Antipattern Report and its levels of details with checkpoints are proposed. Levels of details and checkpoints are:*

*1. Antipattern detected:*
- *The components of the Software System included in the antipattern are defined.*

*2. Antipattern's properties are determined:*
- *An antipattern type has been defined.*

*3. Antipattern influence determined:*
- *The cost of antipattern fixing was estimated.*
- *An assessment of possible losses associated with the presence of an antipattern was carried out.*
- *The antipattern is recognized as mandatory for elimination.*

*4. Antipattern eliminated — optional:*
- *Refactoring completed.*
- *Testing showed the correct work of the components included in the antipattern.*
- *Antipattern elimination activity completed.*

*To analyze the architecture of the system, the Inspect Architecture activity is proposed. The Fix Architecture activity represents actions to fix architecture flaws. Code analysis is presented as a Review the Code activity, and correction of deficiencies found during the analysis is presented as a Refactor the Code activity. The influence of subalpha on the state of the Software System alpha is analyzed. Recommendations were given on the proposed activities. Information about activities is presented in the form of diagrams in the Essence language.*

*The proposed practice allows to record information about the found flaws in the code, process them correctly and avoid problems with the project in the future. It is an example of working with entities that negatively affect the progress of the project.*

**Keywords:** *Antipattern, Code Review, Code Smell, Development, Essence, Practice, Practice Library, Project Management, Refactoring, SEMAT*

*For citation:*

**Zmeev D. O., Zmeev O. A., Ivanova L. S.** Antipattern Practice for Essence Practice Library, *Programmnaya Ingeneria,* 2022, vol. 13, no. 7, pp. 311—321.

DOI: 10.17587/prin.13.311-321

## References

1. **Digital** Economy Report 2021. Cross-border data flows and development: For whom the data flow, available at: https://unctad.org/system/files/official-document/der2021_en.pdf
2. **Portman H.** Review Standish Group — CHAOS 2020: Beyond Infinity, available at: https://hennyportman.wordpress.com/2021/01/06/review-standish-group-chaos-2020-beyond-infinity/
3. **Wojewoda S., Hastie S.** Standish Group 2015 Chaos Report — Q&A with Jennifer Lynch, available at: https://www.infoq.com/articles/standish-chaos-2015/
4. **The Standish** Group — CHAOS Report, available at: https://www.projectsmart.co.uk/white-papers/chaos-report.pdf
5. **Brown W. J., Malveau R. C., McCormic III H. W., Mowbray T. J.** *AntiPatterns: Refactoring Software, Architectures and Projects in Crisis*, New York, John Wiley & Sons, 1998.
6. **Park J. S., Jacobson I., Myburgh B., Johnson P.** SEMAT Yesterday, Today and Tomorrow. An Industrial Perspective, *Programmnaya Ingeneriya*, 2014, no. 11, pp. 6—16 (in Russian).
7. **Jacobson I., Ng P.-W., McMahon P. E., Spence I., Lidman S.** *The Essence of Software Engineering: Applying the SEMAT Kernel,* Addison-Wesley, 2013, 224 p.
8. **Practice** Library, available at: https://practicelibrary.ivarjacobson.com/start
9. **Scott K.** *The Unified Process Explained*, 1st edition, Addison-Wesley Professional, 2001, 208 p.
10. **Beck K., Beedle M., van Bennekum A.** et al. Manifesto for Agile Software Development, available at: https://agilemanifesto.org
11. **Jacobson I., Spence I., Bittner K.** USE-CASE 2.0: The Guide to Succeeding with Use Case. Ivar Jacobson International, 2011. 55 p.
12. **Munera D., Villa G. F.** A brief TOGAF description using SEMAT Essence Kernel. available at: https://www.researchgate.net/publication/335855005_A_brief_TOGAF_description_using_SEMAT_Essence_Kernel
13. **Zapata-Jaramillo C., Henao-Roqueme A.** A proposal for improving the Essence standard by using terminology unification, *Ingenieria,* 2021, vol. 26, no. 2, pp. 213—232. DOI: 10.14483/23448393.16428.
14. **Simonette M., Spina E.** Software & Systems Engineering Interplay and the SEMAT Kernel, *The Voice of the Systems — The Journal Of The Israeli Systems Engineers*, 2018, is. 22, pp. 6—20.
15. **The TOGAF** Standard, Version 9.2 Overview, available at: https://www.opengroup.org/togaf
16. **Perdomo Charry W., Zapata C.** Software quality measures and their relationship with the states of the software system alpha, *Ingeniare,* 2021, vol. 29. pp. 346—363. DOI: 10.4067/S0718-33052021000200346.
17. **Jones C.** Software Development Patterns and Antipatterns. CRC Press, 2022. DOI: 10.1201/9781003193128.
18. **Fowler M., Beck K., Brant J., Opdyke W., Roberts D.** *Refactoring: Improving the Design of Existing Code*, Boston: Addison-Wesley Professional, 1999.
19. **Kerievsky J.** *Refactoring to Patterns,* Boston, Addison-Wesley Professional, 2004, 367 p.
20. **Zmeev O. A., Ivanova L. S.** Design artifacts detection. Review of the approaches, *Vestnik Tomskogo gosudarstvennogo universiteta. Upravlenie, vychislitel'naya tekhnika i informatika,* 2015, no. 2 (31), pp. 81—90 (in Russian).
21. **Cunningham W.** The WyCash Portfolio Management System, *Addendum to the proceedings on Object oriented programming systems, languages, and applications,* 1992, pp. 29—30.
22. **Sadowski C., Söderberg E., Church L., Sipko M., Bacchelli A.** Modern code review: a case study at google, *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP '18).* Association for Computing Machinery, New York, NY, USA, 2018, pp. 181—190. DOI: 10.1145/3183519.3183525.
23. **Fagan M.** Design and code inspections to reduce errors in program development, *IBM Systems Journal*, 1976, vol. 15, no. 3, pp. 182—211.
24. **Gerrit** Code Review, available at: https://www.gerritcodereview.com
25. **CodeFlow,** available at: https://www.getcodeflow.com
26. **GitHub,** available at: https://github.com
27. **GitLab,** available at: https://about.gitlab.com
28. **Zmeev D. O., Ivanova L. S., Rafikova R. R.** Presenting the progress of a software development project in the form of a dynamic Bayesian network, *Informacionnye tekhnologii i matematicheskoe modelirovanie (ITMM-2020). Materialy XIX Mezhdunarodnoj konferencii imeni A. F. Terpugova,* Tomsk, 2021, pp. 291—297 (in Russian).


## ИНФОРМАЦИЯ