Валидация мышления программиста

По мотивам классического рационализма

Сергей Минюров, Москва, 2019

Я сомневаюсь, значит мыслю; я мыслю, значит существую

Рене Декарт

Программист это инженер, создающий технические решения для построения информационных систем. Информационная система и программные решения принципиально отличаются от традиционного, материального мира, с которыми до этого человек взаимодействовал:

1. Отсутствуют физические ограничения: при этом виртуальность процесса и результата через моделирование позволяют достигать принципиально новых результатов, создавать новый мир.
2. Мышление и его непосредственный продукт (идеи, документация, программный код) становятся основной частью социальных и производственных процессов.

Мышление как произвольный творческий процесс балансируется проверкой: внутренней логической и внешней практической. Критичность и дисциплина мышления являются необходимым условием его продуктивности: способность к самопроверке можно представить как некоторое зеркало, позволяющее посмотреть на процессы и результаты мышления со стороны, определять и применять критерии для управления собственной деятельностью и достижения целей.

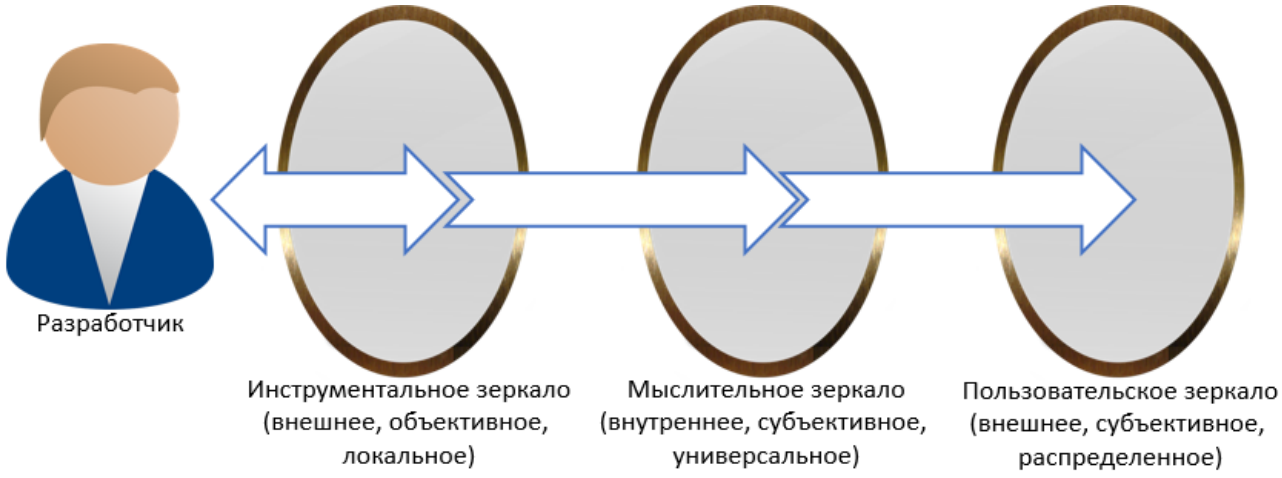

Рисунок 1. Критичное мышление как многоуровневое зеркало

Программист как писатель или конструктор создает произвольную лингвистическую смысловую систему, используя языки программирования. Творческий произвол и отсутствие материальных ограничений компенсируется формализмом языка программирования и буквальным исполнением программого кода. Это задает начальный критерий его деятельности: способность написать текст (программный код), «понятный» (компилируемый в бинарный код) для компьютера.

Листинг 1. Пример формального синтаксиса на языке программирования

```
namespace Philosophy
{
    public abstract class Idea
    {
        public Person Author { get; }
        public DateTime CreatedAt { get; }
        public SyntaxTree Expression { get; }
        public LogicalTree Value { get; }
        public Dictionary<SynthesisMethod, Idea> Predecessors { get; }
        public Dictionary<SynthesisMethod, Idea> Followers { get; }
        public abstract LogicalTree Inference(Question question, ProblemContext context);
    }
}
```

Программисты разработали инструменты для своей работы: среды разработки (например, Eclipse, Visual Studio), методики и средства для проектирования и тестирования решений. Благодаря этому разработчик имеет быструю и «богатую» обратную связь, формирующую для его мышления инструментальное зеркало.

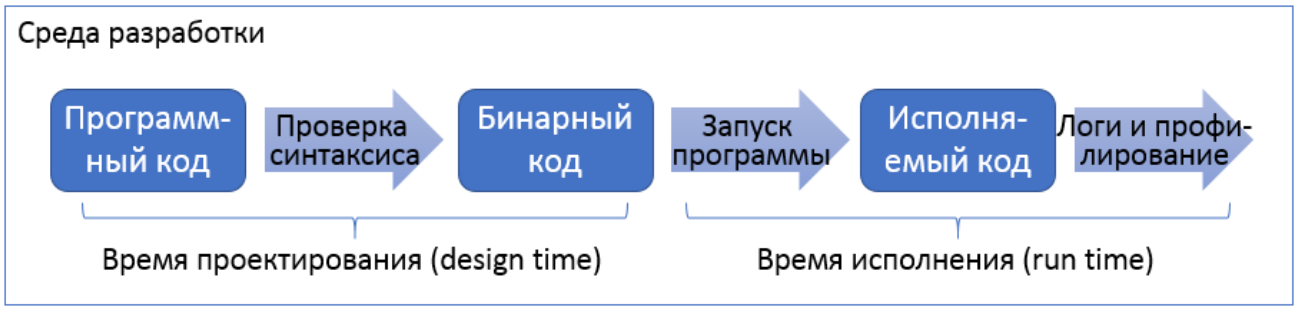

Рисунок 2. Рабочий процесс программиста в среде разработки

Программирование, также как логика и математика, имеет фундаментальную проблему собственного осмысления: формальная правильность выражений не определяет содержательность и ценность информации, которая в них выражается. Более того, интерпретация значения выражений в формальных системах является принципиально сложной задачей. Для программиста это выражается в том, что написание кода является более простой задачей, чем чтение и анализ готового кода именно из-за семантического разрыва между текстом и интенцией (зачем этот код был написан, почему он написан именно таким образом).

> Поэтому важной компетенцией для программиста является его предметный опыт. Например, если проект связан с медициной, то непосредственное общение с врачами как с экспертами, изучение медицинских знаний для программиста будут очень полезным опытом, позволяющим разрабатывать более функциональные и простые решения.
>
> При этом постепенно формируется мыслительное зеркало как метауровень мышления, при накоплении опыта и формирования логических моделей. Этот уровень позволяет управлять сложностью бесконечного числа возможностей и вариантов решений.

Для мышления программиста важным является непосредственное общение с пользователями его программ, понимание их опыта и точки зрения. В определенном смысле обратная связь от пользователей является конечным, определяющим критерием, хотя и имеющим субъективное содержание.

Коммуникации программиста

Верно определяйте слова, и вы освободите мир от половины недоразумений

Рене Декарт

Разработка программного обеспечения это сложный социально-производственный процесс, в котором участвуют множество людей, и не только в самой ИТ-отрасли. Ближний круг профессионального общения программиста состоит из других программистов и тестировщиков, аналитиков и архитекторов, руководителя проекта и пользователей.

Программист может делать проект полностью самостоятельно, в этом случае он берет все роли на себя. Сравнивая командные и индивидуальные проекты, оказывается, что коммуникации и взаимодействие в команде могут очень сильно усложнять рабочий процесс.

> Таким образом, команда из двух программистов не является в два раза более производительной, чем один программист. Если команда большая, то это может приводить к парадоксу: когда один опытный программист может сделать лучше и быстрее, чем целая команда, именно из-за издержек взаимодействия.
>
> Отсюда идеал управления командой является стремление к достижению уровня производительности индивидуального программирования: это является критерием работы руководителя, аналитика и архитектора, прежде всего по качеству постановки задач.

С другой стороны, если члены команды нашли общий язык и адаптировали рабочий процесс под свои сильные стороны каждого из участников, то командная работа дает более глубокое понимание задач и позволяет находить более эффективные решения, чем на индивидуальном проекте. Создание такого «коллективного разума» является внутренней стратегической задачей руководителя проекта.

Рисунок 3. Коммуникации с точки зрения программиста

Как мы выше говорили, коммуникации и взаимодействие в команде и с пользователями являются критическими факторами для успешности проекта и производительности работы. Обычная практика основана на документах, которые предназначены, прежде всего, для фиксации ответственности и взаимных обязательств. Использование их как носителей знаний, даже в электронном формате и онлайн, имеют принципиальные ограничения. Соответственно, проектные знания фиксируются в

самых разных формах: электронная почта, мессенджеры, устное обсуждение - и оказываются фрагментированными и неактуализируемыми.

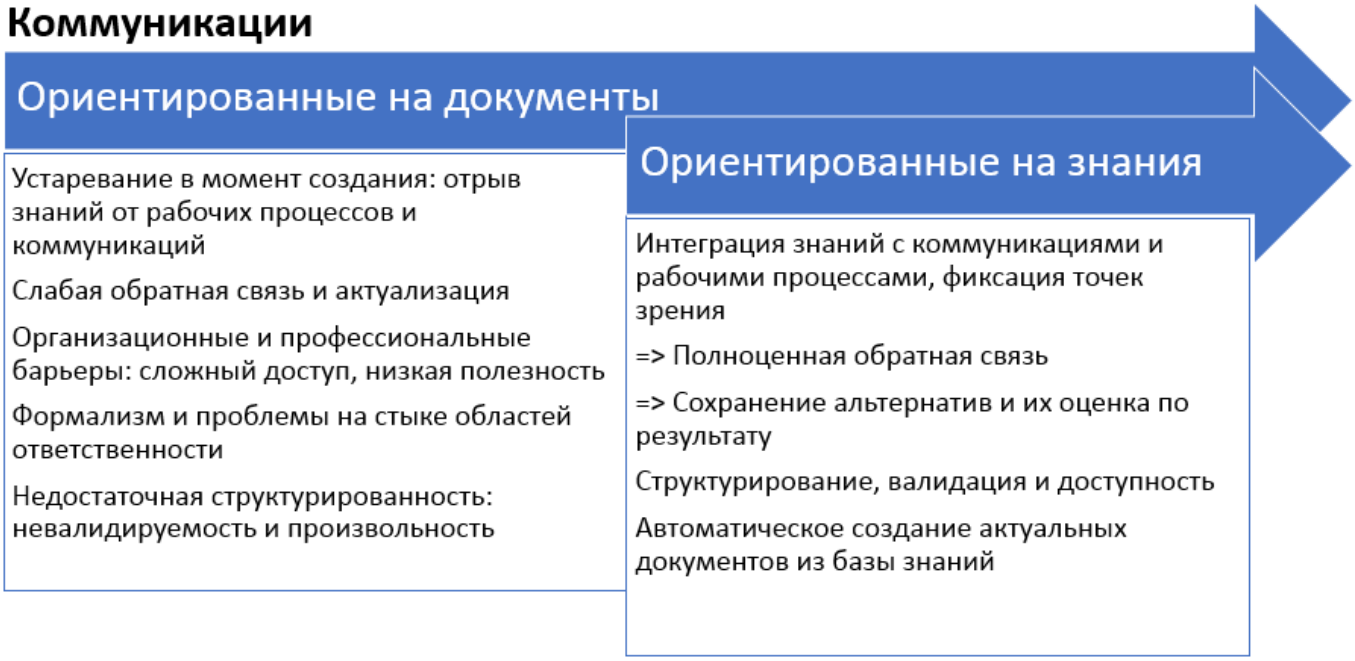

Рисунок 4. Рабочие коммуникации

В проектной документации фиксируются прежде всего задачи и доступ к ресурсам. Описание архитектурных и технических идей и решений документируеттся минимильно для уменьшения трудоемкости и из-за проблем с ручной актуализацией. В небольших командах это компенсируется непосредственным общением (обсуждение вариантов решений, просмотр кода, парное программирование) и формированием практик.

На больших и сложных проектах фрагментация и устаревание знаний становится принципиальной проблемой, приводящей к многократному увеличению трудоемкости и серьезному снижению качества решений («так получилось»).

Уровень решения	Пространство решений	Целевая функция
Деятельность		Существование
Процесс		Продуктивность
Исполнитель		Развитие
Роль		Качество
Функция		Надежность
Способ		Экономичность

Рисунок 5. Уровни принятия решений

Для совместной работы важным критерием является достижение консенсуса и взаимного понимания потребностей каждой роли на проекте. Также бывают серьезные проблемы из-за формального разделения функций и ответственности: в таком случае программист становится техническим исполнителем, не знающим и не интересующимся предметной областью проекта. Это сужает уровень понимания и принятия решений до уровня функций и способа их реализации. Такая пассивная роль

программиста на проекте не может полностью компенсироваться аналитиком и архитектором. Это приводит к значительному падению производительности разработки и качества решений.

Для меня возможность исследовать разные предметные области является одним из важных преимуществ профессии программиста. И личный опыт подтверждает ценность предметных знаний для проектирования архитектуры и реализации программного решения.

В зависимости от специфики и сложности проекта, уровни зрелости команды и соответствия уровней компетенций ее участников можно использовать различные модели коммуникаций, например:

1. Хирургическая бригада[1] – если имеется большой разброс в уровнях компетенций, то важно организовать работу команды вокруг опытных специалистов и делегировать им полномочия по принятию решений в соответствующих областях ответственности.
2. Конвейер (на основе управления знаниями и самоорганизации) – если команда зрелая и мотивированная, уровень компетенций выровненный, то важно сделать рабочий процесс открытым (в т.ч. для пользователей) и постоянно адаптируемым к возможностям специалистов и проектным задачам.

В зрелой профессиональной команде нет иерархий и жестких функциональных разделений, поскольку каждый из участников имеет возможность внести свой вклад в общий результат. Программист может обладать полезным опытом для аналитика или архитектора. И наоборот, открытый рабочий процесс, общение с аналитиком и архитектором позволяет программисту не ограничивать свое мышление только технической стороной решения.

Среды деятельности как уровни валидации

Один, глядя в лужу, видит в ней грязь, а другой — отражающиеся в ней звёзды

Иммануил Кант

Мышление программиста обладает только частичной способностью к самопроверке: логическая правильность не гарантирует содержательную (адекватную реальности и деятельности) правильность в наших рассуждениях. Поэтому необходим консенсус с другими программистами и пользователями, что программный код действительно выполняет поставленные задачи соответствующим образом.

В общении с коллегами (другими программистами, аналитиком или архитектором) важным критерием является совместное понимание предметной области, требований и логики решения. Но это также является недостаточным и субъективным условием.

Важным фактором является совместная работа с тестировщиком. На больших и сложных проектах это является необходимым условием для получения работающего решения. Программист также занимается тестированием для ускорения разработки в рамках проекта в целом: чем раньше ошибка обнаружена, тем легче ее исправить.

> Тестировщик совмещает компетенции программиста и аналитика и считается более квалифицированным специалистом в сравнении с программистом. Тестировщик также занимается программированием для автоматизации тестирования, что может быть более трудоемкой и сложной работой, чем обычное программирование.

[1] Описана в книге Фредерика Брукса по управлению проектами «Мифический человеко-месяц».

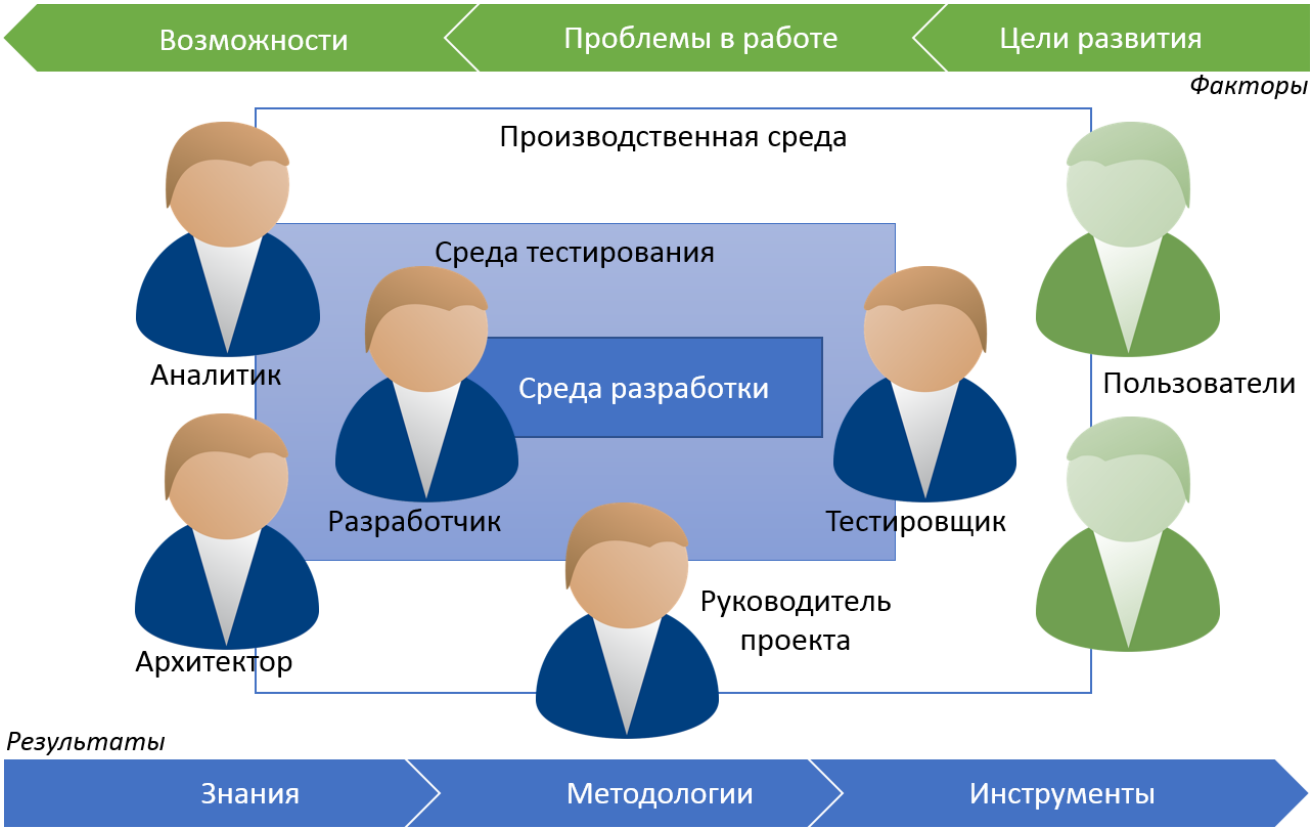

Рисунок 6. Среды деятельности, факторы и результаты

В абстрактном отношении пользователь задает конечный критерий полезности решения и правильности его работы. В практическом плане каждый пользователь имеет собственную точку зрения и требуется значительная аналитическая работа для получения полезной обратной связи.

> На проекте я стараюсь найти мотивированных и компетентных пользователей, установить с ними доверительные отношения и обеспечить эффективное решение их задач. Для этого можно поработать на рабочих местах пользователей, увидеть задачи и проблемы их глазами. В этом случае обратная связь получается простой, качественной и сфокусированной на целях проекта.

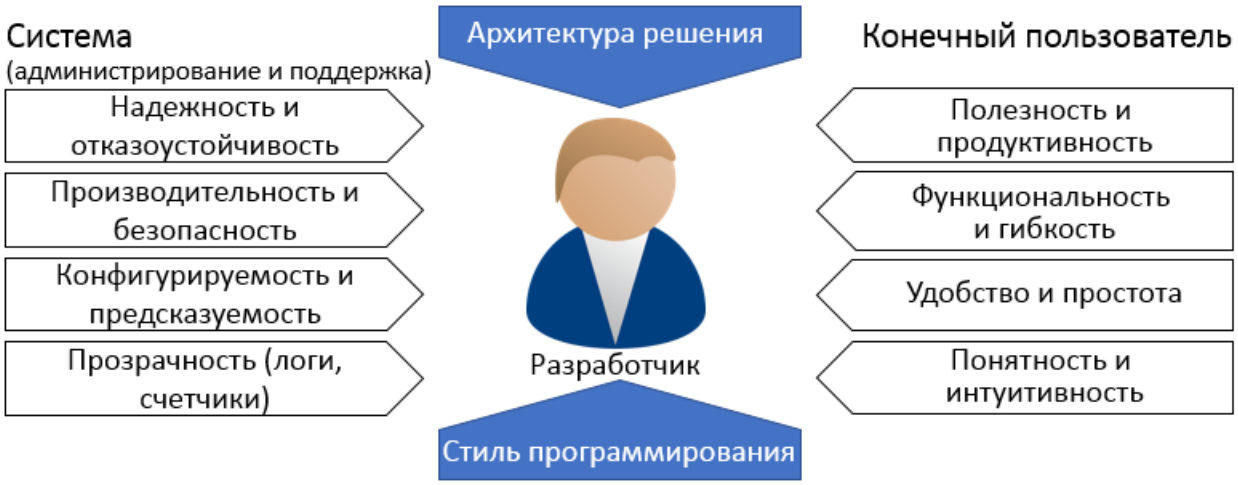

Рисунок 7. Контекст мышления программиста: внутренние и внешние факторы

Программист имеет два контекста мышления:

- Разработка – основной рабочий контекст, важный для концентрации программиста непосредственно в процессе поиска и реализации технических решений.
- Эксплуатация – расширенный рабочий контекст, важный для успешного внедрения и использования программы. Этот контекст необходим на этапах проектирования и внедрения.

Не существует универсального набора требований и решений, подходящего под любой проект. В профессиональном сообществе постоянно разрабатываются шаблоны решений и практики, необходимым компонентом которых является условия и ограничения их применения.

Мышление программиста

Человек не станет господином природы, пока он не стал господином самого себя.

Георг Вильгельм Фридрих Гегель

Мышление остается для нас загадкой, о которой с давних времен имеется множество различных представлений и теорий. В дальнейшем мы будем использовать логические метафоры, не претендуя на научную теорию о мышлении.

Вначале нам нужно идентифицировать предмет. При различении мы выделяем его свойства и состав, находим отличия для схожих предметов. Отождествление выделяет общие и существенные (с определенной точки зрения) характеристики для различных предметов.

Рисунок 8. Мыслительные операции

Свертывание упрощает сложные мыслительные конструкции (преобразует в символы) для использования результатов в дальнейшем процессе мышления, обеспечивает передачу опыта. Развертывание обеспечивает глубинное понимание нового (решение проблемы) на основе прошлого опыта мышления, является способом интеграции мышления в практическую деятельность.

Виды программирования и различия в мышлении:

- Системное программирование – решение задач по управлению аппаратными ресурсами (компьютер, сетевые устройства, автоматизированное оборудование), разработка операционной системы, драйверов устройств: мышление близкое к математическому.
- Инструментальное программирование – создание инструментов для других программистов, в т.ч. и для системного: среда разработки и т.д. На прикладном проекте эффективность его работы заключается в обобщении и решении класса задач, разработки инфраструктуры (базовых компонентов), используемой затем другими программистами. Важной компетенцией является системно-логическое мышление. Для прикладных задач требуется глубокое знание предметной области проекта.

- Прикладное программирование – решение задач для конечных пользователей: автоматизация функций и процессов. Имеется большое множество специализаций на разных языках и платформах: интеграция, безопасность, базы данных, сервисы, пользовательский интерфейс и т.д. Мышление близкое к лингвистическому и логическому.

В зависимости от индивидуальных склонностей разные программисты могут иметь в большей степени техническое или предметное мышление. В команде важно иметь программистов обоего типа и обеспечить их эффективное взаимодействие.

> Важным моментом является ограниченность нашего мышления по обработки информации. Если опытный программист имеет достаточную техническую и предметную компенетцию, то из-за данной перегрузки он не сможет одновременно продуктивно решать прикладные, инструментальные или системные задачи: нужно договориться в команде об эффективном разделении видов задач.

Программист постоянно пишет, читает и анализирует программный код, который написан им самим, либо другим программистом. При этом ему нужно в собственном мышлении формировать модели двух уровней:

- Статическая логическая модель – позволяет управлять сложностью самого программного кода, понимать логику его написания и организации.
- Динамическая исполняемая модель – позволяет понимать поведение программы во время ее исполнения, корректировать и развивать статическую логиченскую модель, которая ее порождает.

Рисунок 9. Мыслительный цикл программиста

Понимание предметной логики («Что» и «Зачем») является важным для разработки технического решения и выбора оптимального варианта («Как» и «Почему»). На основании мыслительных операций и мыслительного цикла формируется две практики проектирования:

1. Декомпозиция (превращение сложности в простоту) – после определения границ решения (идентификация) определяются компоненты (различение), определяются интерфейсы для их взаимодействия (свертывание), базовые абстракции и варианты реализации (отождествление). При тестировании проверяется правильность работы программных функций и процессов (развертывание).
2. Моделирование (превращение знания в инструмент мышления) – синтез предметной и программной логики является областью инженерного творчества: разделение логики на объекты и уровни, соответствующие предметной области, оказывается великолепной эвристикой для поиска простых и надежных решений.

Валидация мышления

Прекрасно то, что нравится независимо от смысла
Иммануил Кант

Первичная проверка собственного мышления выполняется программистом в рамках своей профессиональной компетенции: способность создавать логически целостный и синтаксически правильный программный код.

> Принципиальным преимуществом является возможность «материализовать» свою идею в программном коде и посмотреть на нее со стороны. Также очень важным, как мы говорили в начале, является наличие инструментального зеркала, создающие короткие петли обратной связи. Это, в свою очередь, создает возможности для ускоренного развития мышления и углубления познания техники программирования и предметной области.

В силу принципиальной ограниченности мышления и семантической неопределенности формальной логики в языке программирования синтаксическая и функциональная валидация не является достаточной, и нам требуется также внешнее подтверждение правильности и полезности программного решения.

> Опытный программист оценивает разрабатываемое решение по выразительности, стройности и целостности логики. Это является его внутренним, интуитивным критерием истины.

Рисунок 10. Уровни валидации мышления программиста

При внешней валидации можно сказать, что программы проходят естественный отбор, который также не является полностью детерменированным. Время жизни программы и ее популярность (как и научных идей) определяется заранее неизвестным количеством факторов. Идеи из программы могут повторно использоваться в других решениях, становиться шаблонами решений и профессиональными практиками, если они демонстрируют новые подходы или результаты.

> Сотрудничество в команде при разработке решения и с пользователями при внедрении основано на взаимной ответственности. С юридической точки зрения компания-поставщик традиционно снимает с себя ответственность за последствия использования ее продукта. Программа как открытая система всегда зависит от внешних факторов. Технически невозможно реализовать полную диагностику и компенсацию любых нештатных ситуаций.
>
> В плане мотивации профессиональная ответственность разработчика перед своими коллегами или пользователями является критическим фактором качества и продуктивности в его работе.

Таким образом, программирование как инженерная деятельность имеет социальную реальность как конечный и окончательный механизм валидации ее полезности и ценности.