

THE LOGS OF THE BHAGAVAD GITA

Domain. Dharma. Samsara.



Tapio Niemelä

THE LOGS OF THE BHAGAVAD GITA

Tapio Niemelä

This book is available at https://leanpub.com/logs_of_gita

This version was published on 2026-09-29



This is a [Leanpub](#) book. Leanpub empowers authors and publishers with the Lean Publishing process. [Lean Publishing](#) is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

© 2026 Tapio Niemelä

Contents

THE LOGS OF THE BHAGAVAD GITA	i
Before the First Review Comment	ii
CHAPTER 1: Sanjaya's Vision and the Commit History of Kurukshetra	1
CHAPTER 2: Krishna's School of Architecture and the Immortal Invariants	6
CHAPTER 3: Karma Yoga and the Orchestration of Deeds	14
CHAPTER 4: The Oldest Commit and the Generations of Knowledge	15
CHAPTER 5: Sanyasa Yoga, or the Trap of the Great Rewrite	16
CHAPTER 6: Dhyana Yoga and the Art of Deep Work	17
CHAPTER 7: Jnana-Vijnana Yoga, or the Synthesis of Abstraction and Runtime Reality	18
CHAPTER 8: Akshara Brahma Yoga, or the Rehydration of the Aggregate	19
CHAPTER 9: Raja-Vidya Raja-Guhya Yoga, or the Royal Architec- tural Secret	20
CHAPTER 10: Vibhuti Yoga, or the System's Mighty Manifestations and Entropy	21
CHAPTER 11: Vishvarupa Darsana Yoga, or the Vision of the Cosmic System and All Its Dependencies	22

CHAPTER 12: Bhakti Yoga, or the Art of Loving the Codebase	23
CHAPTER 13: Kshetra-Kshetrajna Vibhaga Yoga, or Distinguishing the Codebase from the One Who Understands It	24
CHAPTER 14: Gunatraya-Vibhaga Yoga, or the Three Qualities and the Laws of Organisational Karma	25
CHAPTER 15: Purushottama Yoga, or the Yoga of the Supreme Architect and the Eternal Dependency Tree	26
CHAPTER 16: Daivasura-Sampad-Vibhaga Yoga, or Distinguishing Divine and Demonic Development Practices	27
CHAPTER 17: Shraddhatraya-Vibhaga Yoga, or the Threefold Faith and Motives in Coding	28
CHAPTER 18: Moksha-Sannyasa Yoga, or Final Liberation and Ar- chitectural Enlightenment	29
About the Book	30

THE LOGS OF THE BHAGAVAD GITA

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/logs_of_gita.

Domain • Dharma • Samsara

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/logs_of_gita.

Before the First Review Comment

Every software system eventually becomes a philosophical problem.

At first, it is merely a repository. There are tickets, classes, database tables, endpoints, deadlines, and people trying to be helpful. The requirements appear finite. The architecture diagram fits on one slide. Someone says the system is almost complete.

Then time begins its work.

A temporary decision becomes a convention. A convenient getter becomes an API. A database representation becomes the domain model. A workaround survives the problem it was created to solve. New developers copy what they find, because existing code is the most persuasive documentation a system possesses.

Eventually, nobody can explain whether the architecture reflects the business or whether the business has simply learned to speak in the language of its architecture.

At that point, the repository begins asking questions:

- What does this value actually mean?
- Who owns this decision?
- Why must this operation cross three services?
- Is this complexity essential to the domain, or merely accidental complexity accumulated by the implementation?
- Does the test protect a business rule—or only the behaviour of an old mistake?

And the most dangerous question of all:

If the pipeline is green, why does this change still feel wrong?

The Logs of the Bhagavad Gita begins at precisely that moment.

Arjuna has not written the disputed code. He has been asked to review it. The Merge Request adds two getters to an immutable Money Value Object so that a mapper can construct an external DTO.

The change is small. The reasoning is sensible. The tests pass.

Arjuna nevertheless sees a possible future growing inside it: external code extracts amount and currency, performs calculations, copies rounding rules, forgets currency checks, and reconstructs Money after the important decisions have already been made elsewhere.

He cannot prove that this future will occur. He cannot honestly claim that getters are inherently evil. He also cannot approve the change merely to remove the Merge Request from everyone's path.

He stands between action and inaction, certainty and doubt, collegiality and responsibility.

In other words, he stands on Kurukshetra.

The original Bhagavad Gita is a dialogue spoken at the beginning of a war. Arjuna sees teachers, relatives, friends, and respected elders on both sides of the battlefield. Unable to reconcile his duty with the consequences of acting, he lowers his bow. Krishna does not give him a coding standard. He asks Arjuna to understand action, knowledge, duty, consequence, attachment, perception, and the nature of reality itself.

This book commits the questionable act of translating that crisis into software architecture.

- **Dharma** becomes the responsibility proper to a developer, component, Aggregate Root, Application Service, projection, or Saga.
- **Karma** becomes the chain of consequences created by every technical decision. A commit produces its intended fruit—but also unfruits: dependencies, conventions, assumptions, and future constraints that were never mentioned in the Jira ticket.

- **Samsara** becomes the cycle through which today's elegant greenfield system becomes tomorrow's legacy platform and is reborn as another greenfield system carrying the same misunderstood domain assumptions.
- **Knowledge yoga** becomes knowledge crunching.
- **Meditation** becomes deep work.
- **The three gunas** become qualities visible in code, decisions, and engineering culture.
- **Krishna's cosmic form** becomes the complete production system revealed through observability and distributed tracing: every service, event, database, timeout, retry, historical compromise, and hidden dependency visible at once.

Liberation is not perfect architecture. It is freedom from the need to make architecture immortal.

The analogies are playful, but the underlying argument is serious.

Domain-Driven Design is not primarily a collection of tactical patterns. Aggregate Roots, Entities, Value Objects, Repositories, and Domain Events matter, but they are not the soul of DDD. The soul is the iterative attempt to understand a business reality well enough to express it honestly.

That process has no final specification.

Knowledge crunching produces a hypothesis. A model gives the hypothesis form. Tests make its assumptions executable. Code reveals where the model resists change. Refactoring writes new understanding back into the system. Then a difficult example appears, and the cycle begins again.

Artificial intelligence can accelerate every stage of this work. It can also interrupt the learning cycle by implementing the first plausible interpretation with such speed and confidence that nobody pauses to ask whether the interpretation was true.

Architecture therefore matters more, not less.

Clear language gives both human and artificial minds concepts with which to reason. Boundaries reduce the amount of irrelevant context they must hold. Tests provide executable claims about behaviour. A well-factored model makes the important part of the problem visible by pushing accidental complexity aside.

Yet none of these can replace judgment.

A test cannot tell us whether we tested the right rule. A type system cannot guarantee that our concepts are honest. An Aggregate Root cannot protect an invariant nobody has discovered. An AI agent cannot ask the domain expert a question the team believes has already been answered.

Someone must still notice the pain in the code and ask what it is trying to say.

That person is not an architectural police officer. They are a code whisperer.

They do not begin by asking:

Who wrote this?

They ask:

What earlier decision produced this pain?

And before approving the next change:

What consequences—what unfruits—might this decision leave to those who come after us?

This book is not a translation, summary, or scholarly commentary on the Bhagavad Gita. It is an affectionate and deliberately absurd adaptation written from inside the lived reality of software development. Sanskrit concepts are used as bridges for thought, not as claims of religious authority.

The jokes belong to software engineering. Any wisdom that survives them belongs to a much older tradition.

You do not need to know the Bhagavad Gita to read this book.

You do not need to know Domain-Driven Design either.

You need only to have encountered a technically successful solution that somehow made the system less truthful.

The battle begins with two getters.

The pipeline is green.

Arjuna lowers his bow. 🏹

CHAPTER 1: Sanjaya's Vision and the Commit History of Kurukshetra

Architectural Sutra: *When a developer opens a diff, he also opens the past. Architecture begins where certainty ends and responsibility remains.*

The blind owner asks for a status report

Dhritarashtra was blind. He had never looked at the codebase himself, and he had no access to the performance metrics. He was the Product Owner, seated in the executive suite, and he wanted to know one thing only: whether the features sworn to in the sprints were in production.

He turned to Sanjaya. Sanjaya was the team's architecture whisperer and the keeper of the CI/CD pipeline, and to him had been granted *divya-drishti* – the divine faculty of sight, a real-time observability tool reaching into every microservice log and every pull request review.

Dhritarashtra:

“Sanjaya! When my team (the Kauravas) and the Pandavas each gathered in the Kurukshetra branch to hold their reviews, what became of the code-base?”

Sanjaya reports from the battlefield

Sanjaya:

“O King, Duryodhana – commander of the Just Ship It legion – surveyed the architectural field of the Pandavas. He saw their DDD model well-structured, their Value Objects guarded, their database migrations trustworthy. He took fright and hurried at once to his old teacher Drona.”

Duryodhana looked upon the Pandavas' branch and sought comfort in belittling their data model.

Duryodhana:

“Behold, master, how vast their domain model is! They have absolute invariants there, strict Aggregate Roots and immutable classes. But look upon our own army! Here we have swift getters and setters, public fields, global state and rapid hotfixes. Our forces are boundless; their model is rigid!”

Duryodhana ordered the war horns sounded: let every man merge his own branch without architectural review! Through the air rang the blast of the trumpets: *Just Ship It!*

The pipelines turned yellow, and warnings filled the build logs.

On the opposing side, in the Pandava chariot, sat Arjuna and his charioteer Krishna. Their chariot ran on no ordinary platform; it was built upon a perfectly tested asynchronous event core.

Arjuna asked Krishna to drive the chariot between the two branches.

Arjuna between the two armies**Arjuna:**

“Krishna, drive my development environment here, into the middle. I wish to see those who have entered into this battle. I wish to see the change that I am to approve or reject.”

Krishna drove the chariot into the very middle of the Kurukshetra commit history – between the feature branch and main, to the place where a locally sensible change and the long-term integrity of the domain model meet.

Arjuna looked to both sides, and his heart broke.

On the one side – in the *Just Ship It* ranks – stood commits of his own. There were the classes he had written himself five years ago in a hurry, on the principle of *“we’ll fix this later.”* There stood Drona, the old teaching codebase from which he had learned his first programming languages. There stood Bhishma, the old infrastructure that no one dared

dismantle. There stood his teammates, with whom he had drunk coffee and celebrated successful releases.

And on the other side stood the DDD ideals he had pledged himself to defend.

Arjuna:

“Krishna! When I see my own colleagues and my own history in code set against one another on that field, my hands begin to tremble. My mouth goes dry. My keyboard slips from my lap.”

The paralysis of the mind (Arjuna Vishada Yoga)

Arjuna looked at the merge request. The change was small: the internal structure of the Money Value Object was to be exposed through getter methods, so that a mapper could build an external DTO.

```
public final class Money {  
  
    private final BigDecimal amount;  
    private final Currency currency;  
  
    public BigDecimal getAmount() {  
        return amount;  
    }  
  
    public Currency getCurrency() {  
        return currency;  
    }  
}
```

The justification was impeccable:

We only need the amount and the currency in the mapper. The change does not alter behaviour.

The pipeline was green. Every test passed. Money was still immutable, and not one existing use case was broken.

That was precisely why Arjuna feared the change.

Arjuna:

“Krishna, how could I reject this? They truly do need the amount and the currency at the boundary. The fields already exist, and reading them alters no object. My colleagues are doing nothing obviously wrong.

But if I approve this, Money will little by little cease to be a concept that performs a calculation. Its internal structure will spread into mappers, services and conditionals. Everyone will pull the amount out, do something to it, and try to assemble money back together again.

If I defend encapsulation, I look like a dogmatist. If I approve the getters without arguing the boundary, I help build with my own hands the anaemic model I have sworn to fight.

What good is a green build, if every green test confirms a model that says less and less about the domain?”

Arjuna did not fear two methods. He feared the world that would begin to be built upon them.

He did not want to press Approve. He did not want to press Reject.

Sanjaya:

“Having spoken thus, Arjuna cast down his bow – his own IDE window – and closed it in the middle of the battlefield. He sat down on the floor of the chariot, shut the terminal and clutched his head in his hands, broken with sorrow.”



Figure 1. Arjuna's Dilemma

CHAPTER 2: Krishna's School of Architecture and the Immortal Invariants

Architectural Sutra: *Code changes its language, shape and syntax, but the truth it carries does not. An invariant is immortal only for as long as someone remembers what it means.*

Arjuna collapses at the keyboard

Sanjaya:

“Beholding Arjuna, who sat with tears in his eyes before a closed laptop, Krishna – the supreme keeper of application architecture – looked upon him gently but firmly, and spoke these words.”

Krishna:

“Arjuna! Whence comes this weakness in the middle of the busiest sprint? This is unworthy of an architect, and it will not carry you to production. Rise up, close the complaints channel in Slack, and set to work!”

Arjuna:

*“Krishna, how could I reject this change? The mapper truly does need a representation of money. The getters change no state, and no existing test breaks. But how could I accept that the rest of the system begins to use the internal structure of the *Money* object as its own programming model?”*

Krishna looked at the diff and asked:

*“Does the mapper need the internal structure of the *Money* object – or does it need a representation from the *Money* object?”*

The immortality of the invariant (Sankhya Yoga)

Krishna smiled lightly. He did not declare every getter to be adharmā, nor did he offer one universal interface. He moved the conversation to the responsibility of the concept, which Arjuna had not yet found the words for.

Krishna:

“You grieve for that which is not worthy of grief, though you speak words of wisdom. The wise architect grieves neither for what is deleted, nor for what is created.

The business truth does not begin when a developer gives it a class name, nor does it cease when that class is removed.

As a person casts off worn garments and puts on new ones, so the Domain casts off obsolete implementations and assumes new forms.

No weapon can sever it, nor can production fire consume it. Water cannot dissolve it, nor can the winds of technological fashion dry it away.

So it is, Arjuna.

Frameworks arise and pass away. Classes are born, renamed and deleted. Databases are migrated, interfaces replaced and entire architectures reduced to ashes.

Yet the truth of the business is not destroyed with its representation. It existed before the first line of code was written, and it remains when the last repository has been forgotten.

Code is only the body through which the Domain acts for a time. Do not mistake the mortality of the implementation for the mortality of the truth it seeks to express.”

The DTO needed a representation of Money, but the representation did not need to dictate the behaviour of the concept.

Money could protect its own calculations and at the same time offer the boundary an explicit representation:

```
public final class Money {  
  
    private final BigDecimal amount;  
    private final Currency currency;  
  
    public Money subtract(Money other) {  
        requireSameCurrency(other);  
        return new Money(amount.subtract(other.amount), currency);  
    }  
  
    public MoneyRepresentation representation() {  
        return new MoneyRepresentation(amount, currency);  
    }  
}
```

This was not the one correct API. Sometimes record `Money(BigDecimal amount, Currency currency)` is a perfectly honest model, and sometimes an infrastructure adapter may read the representation it needs for persistence.

What was decisive was not the syntax of the getter, but **who made the business decisions about money.**

```
var discountedAmount = money.getAmount()  
    .subtract(discount.getAmount());  
  
return new Money(discountedAmount, money.getCurrency());
```

When such calculation spreads outward, Money is nominally a Value Object but in practice two primitives once more. The outside code becomes responsible for the arithmetic, for keeping the currency intact, for rounding, for the validity of the result and for assembling the new object.

A Value Object may offer an external representation. It need not surrender its internal structure as the programming model of the rest of the system.

Krishna looked at the getter methods, at the DTO and at the concept behind them. They were not the same thing.

Krishna:

“That which is mere implementation – accidental complexity – has no lasting existence. That which is a genuine business invariant – essential complexity – does not cease merely because its representation changes.

If Money turns from a class into a record, or its DTO is replaced by another representation, do you imagine that the bond between an amount and its currency has died?

That bond cannot be deleted by refactoring, nor protected by the `private` keyword alone. It survives in the system only while its decisions remain faithful to it.

The Domain cannot be destroyed by code. But it may be concealed beneath the code until no developer remembers that it was ever there.

Therefore, when you look upon a Merge Request, do not ask first whether the syntax is elegant or whether the pipeline is green.

Ask this:

Does this change protect the Domain, or does it slowly teach the system to forget what the Domain is?

To keep asking this question is your dharma.“

Nishkama Karma – act without attachment to results

Arjuna looked at Krishna in bewilderment. If code grew old in any case and everything became legacy, why trouble to defend a single invariant in review?

Krishna answered with the most famous teaching of the Gita:

Krishna:

“You have a right to the work alone – in this case, to an honest review – never to its fruits: an eternal monument, a perfect architecture or praise from the steering group.

Do not imagine yourself the cause of the fruits of your work. Nor, because those fruits are not yours to command, become attached to inaction.

Inaction does not release you from responsibility, Arjuna. What you leave undone also bears consequences.

Never do the work merely to get a ticket closed. Do your work steadily, free of attachment to approval or rejection. This evenness of mind is called refactoring.“

Code without attachment to the fruits:

Goal: A closed Jira ticket, praise ---> Binds you to fear and stress

Goal: The honesty of the model TODAY ---> Nishkama Karma (freedom to act)

The chain of attachment

Arjuna:

“Krishna, how can attachment to the result corrupt a developer’s judgement? Is not the desire to complete the work a virtue?”

Krishna:

“When a developer dwells continually upon the fruit – the closed ticket, the successful deployment, the praise of others – attachment to that fruit arises.

From attachment comes desire: ‘This must be merged now.’ When that desire is obstructed, anger arises.

From anger comes delusion. The reviewer appears to be an enemy, the failing test an obstacle, and the Domain itself an unnecessary delay.

From delusion comes the loss of memory: the developer forgets why the rule exists, what the users asked for and which truth the model was meant to preserve.

When memory is lost, judgement is destroyed. And when judgement is destroyed, the developer brings ruin upon the very system he intended to improve.

He who does his work in fear of a rejected PR, or in expectation of praise or bonus points, is the slave of his results. But he who concentrates on understanding the business model in this very moment attains a serene mind – though the codebase storms around him.“

Sthitaprajna – the steady architect

Arjuna wiped away his tears and asked something very practical:

Arjuna:

“How does one recognise the architect or senior developer whose mind is steady (Sthitaprajna)? How does he speak in code review? How does he react when a P1 production crisis strikes at 16:55 on a Friday?”

Krishna:

“He whom production alerts do not paralyse and quick wins do not blind, he is steady of mind.

As the tortoise draws its limbs into its shell, so the steady developer withdraws his attention from the panic in Slack, from empty framework hype and from arguments on social media. He does not hate legacy code, nor does he worship the newest fashionable language.

While others toss upon the sea of requirements like a raging ocean, the steady architect remains calm. New requirements flow into his codebase every day, but he neither swells with dogmatism nor crumbles under haste. He attains peace.“

The outcome of Chapter II

A new perspective begins to take shape for Arjuna:

1. The implementation is mortal, but the **Domain truth** it seeks to express does not die merely because its representation changes.
2. Deleting or changing old code is not architectural murder, so long as the business **invariant** beneath it is preserved and clarified.
3. The work must be done well **here and now**, regardless of whether the code becomes legacy next year.
4. The developer's freedom lies in not binding his identity to the fruits of his work: to hype, monuments, approval or the perfection of a refactoring.

Arjuna quietly took hold of his Gandiva and opened the diff again. His hands had stopped trembling. He placed them upon the keyboard.

He typed a comment into the review box. Then he read it over:

```
// comment-draft-1.md
```

```
These getters violate the encapsulation of the Money Value Object.
```

```
The bond between amount and currency is a business invariant, and invariants do not disappear when the syntax changes. Expose its internal structure, and calculations will migrate outside the object, currency checks will be duplicated, and the model will slowly say less and less about the Domain.
```

```
Please refactor this: remove the getters and give the mapper an explicit representation of Money.
```

It was correct. Every word of it was correct.

Arjuna looked at the comment for a long moment, but did not press *Comment*. It was, he suspected, less a question than a verdict.

Still uncertain whether his silence was wisdom or avoidance, he turned to Krishna.

“Your teaching is clear, Krishna. But if wisdom and steadiness are more important than hasty action, why do you nevertheless command me to enter this difficult battle and to take a position on this Merge Request?”

CHAPTER 3: Karma Yoga and the Orchestration of Deeds

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/logs_of_gita.

CHAPTER 4: The Oldest Commit and the Generations of Knowledge

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/logs_of_gita.

CHAPTER 5: Sanyasa Yoga, or the Trap of the Great Rewrite

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/logs_of_gita.

CHAPTER 6: Dhyana Yoga and the Art of Deep Work

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/logs_of_gita.

CHAPTER 7: Jnana-Vijnana Yoga, or the Synthesis of Abstraction and Runtime Reality

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/logs_of_gita.

CHAPTER 8: Akshara Brahma Yoga, or the Rehydration of the Aggregate

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/logs_of_gita.

CHAPTER 9: Raja-Vidya Raja-Guhya Yoga, or the Royal Architectural Secret

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/logs_of_gita.

CHAPTER 10: Vibhuti Yoga, or the System's Mighty Manifestations and Entropy

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/logs_of_gita.

CHAPTER 11: Vishvarupa Darsana Yoga, or the Vision of the Cosmic System and All Its Dependencies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/logs_of_gita.

CHAPTER 12: Bhakti Yoga, or the Art of Loving the Codebase

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/logs_of_gita.

CHAPTER 13: Kshetra-Kshetrajna Vibhaga Yoga, or Distinguishing the Codebase from the One Who Understands It

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/logs_of_gita.

CHAPTER 14: Gunatraya-Vibhaga Yoga, or the Three Qualities and the Laws of Organisational Karma

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/logs_of_gita.

CHAPTER 15: Purushottama Yoga, or the Yoga of the Supreme Architect and the Eternal Dependency Tree

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/logs_of_gita.

CHAPTER 16:

Daivasura-Sampad-Vibhaga Yoga, or Distinguishing Divine and Demonic Development Practices

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/logs_of_gita.

CHAPTER 17:

Shraddhatraya-Vibhaga Yoga, or the Threefold Faith and Motives in Coding

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/logs_of_gita.

CHAPTER 18: Moksha-Sannyasa Yoga, or Final Liberation and Architectural Enlightenment

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/logs_of_gita.

About the Book

A Merge Request adds two harmless getters to an immutable Money Value Object. The pipeline is green. The mapper needs the data. Nobody has done anything obviously wrong. Arjuna cannot approve it.

Standing between the DDD Alliance and the Just Ship It Legion, he sees what the tests cannot: a locally reasonable change may teach the entire system to dismantle a domain concept into primitives.

Fortunately, his reviewer is Krishna.

Across eighteen chapters, *The Logs of the Bhagavad Gita* reimagines the Gita as a dialogue about software architecture, Domain-Driven Design, technical debt, knowledge crunching, Event Sourcing, refactoring, entropy—and the moral terror of reviewing code written by people you respect.

Here:

- **karma** becomes the delayed consequences of every commit
- **dharmā** becomes the responsibility proper to each component and developer
- **samsara** becomes the endless cycle of greenfield, legacy, and rewrite
- **the three gunas** become qualities of code and engineering culture
- **the cosmic form** becomes the complete production dependency graph
- **liberation** begins when a developer can finally delete an abstraction without taking it personally

This is not a translation of the Bhagavad Gita. It is an affectionate, technically serious, and occasionally irresponsible encounter between an ancient philosophical dialogue and the modern software project.

*No framework will attain enlightenment.
Your Aggregate Root might.*