技術的負債は開発者体験を悪化させる

@mtx2s

2022-12-21 技術的負債の返済から改善する開発者体験 - Techmee vol.5

自己紹介

松本 成幸（まつもと しげゆき） / @mtx2s

- コンシューマー向けソフトウェアプロダクトを開発する組織のエンジニアリングマネージャー

- はてなブログやnote, Twitterで細々と発信しています
  - https://mtx2s.hatenablog.com/
  - https://note.com/mtx2s
  - https://twitter.com/mtx2s

『技術的負債は開発者体験を悪化させる』

アジェンダ

- 2つの体験の似た特徴
- 振る舞いの価値、構造の価値
- 無謀な開発による悪循環
- 無謀な負債、慎重な負債
- 2つの体験の決定的な違い

アジェンダ

- 2つの体験の似た特徴
- 振る舞いの価値、構造の価値
- 無謀な開発による悪循環
- 無謀な負債、慎重な負債
- 2つの体験の決定的な違い

ソフトウェアプロダクトにまつわる「体験」とは？

○○体験

○○ Experience

ソフトウェアプロダクトにまつわる「体験」とは？

ユーザー体験

UX: User eXperience

開発者体験

DX: Developer eXperience

よく似た特徴を持っている

2つの体験によく似た特徴とは？

もし、**優れた体験**を得られなければどうなるのか？

優れた体験を得られなければこうなる

ユーザー体験

UX: User eXperience

利用する気が失せた
ユーザーは離反する

開発者体験

DX: Developer eXperience

開発する気が失せた
開発者は離反する
退職

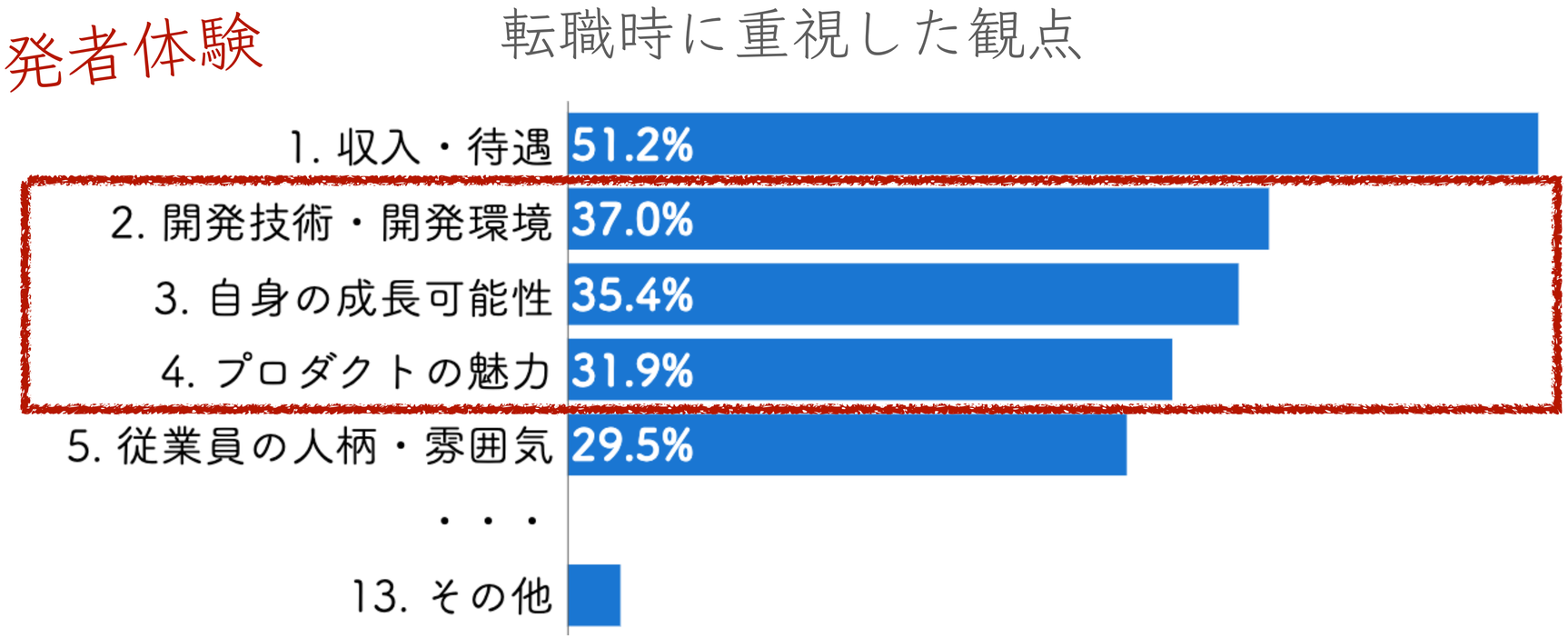
技術的負債は開発者体験の阻害要因

“「レガシーコード」という言葉を聞いて、皆さんは何を思い浮かべるでしょうか。仮に私と同じような立場であれば、変更が必要なものの、本当に理解することができない、構造のわかりにくい、複雑に絡み合ったコードを思い出すことでしょう。簡単なはずの機能追加をしようとして徹夜したことや、士気を喪失してしまったこと、チーム全員がコードにうんざりしてどうでもよくなってしまった感覚、（中略）などを思い出すでしょう。そのコードを改善しようと考えることすら嫌な気持ちになるかもしれません。そんな手間をかけるのは無駄に思えるからです。”

『レガシーコード改善ガイド』 はじめに

優れた開発者体験を期待できそうな企業を求めて転職

優れた開発者体験
を期待？



※下記ページに掲載された図をもとに作成
ITエンジニアの転職先決定要因に関する調査結果発表、groovesとHERPが実施 | CodeZine
https://codezine.jp/article/detail/14890

技術的負債から逃れられない

どこに行っても**大抵、技術的負債を抱えている**

真正面から向き合うしかない！

アジェンダ

- 2つの体験の似た特徴
- 振る舞いの価値、構造の価値
- 無謀な開発による悪循環
- 無謀な負債、慎重な負債
- 2つの体験の決定的な違い

ソフトウェアの変更が何を変えるのか？

	要件追加	バグ修正	リファクタリング	最適化
構造	変化する	変化する	変化する	
機能	変化する	変化する		
リソース利用				変化する

『レガシーコード改善ガイド』 第1章 ソフトウェアの変更

要件追加やバグ修正の場合

	要件追加	バグ修正	リファクタリング	最適化
構造	変化する	変化する	変化する	
機能	変化する	変化する		
リソース利用				変化する

「構造」と「機能」の両方が変化する

リファクタリングの場合

	要件追加	バグ修正	リファクタリング	最適化
構造	変化する	変化する	変化する	
機能	変化する	変化する		
リソース利用				変化する

「構造」だけが変化する

ここに技術的負債が蓄積されている

ソフトウェアプロダクトの2つの価値

“ソフトウェアには2種類の価値がある。「振る舞いの価値」と「構造の価値」だ。そして、後者のほうが価値が大きい。なぜなら、それがソフトウェアをソフトにする価値だからだ。”

変更しやすくする（変更容易性）

小なり

振る舞いの価値 ＜ 構造の価値

『Clean Architecture』 第15章 アーキテクチャとは？

振る舞いの変更にばかり注力している

技術的負債が増え続ける開発現場では……

	要件追加	バグ修正	リファクタリング	最適化
構造	変化する	変化する	変化する	
機能	変化する	変化する		
リソース利用				変化する

ここに注力

だから構造に技術的負債が蓄積されていく

技術的負債が増え続ける開発現場では……

	要件追加	バグ修正	リファクタリング	最適化
構造	変化する	変化する	変化する	
機能	変化する	変化する		
リソース利用				変化する

技術的負債が
蓄積されていく

蓄積された技術的負債の返済は進まない

技術的負債が増え続ける開発現場では……

	要件追加	バグ修正	リファクタリング	最適化
構造	変化する	変化する	変化する	
機能	変化する	変化する		
リソース利用				変化する

返済が進まない

真逆のことが起きている

技術的負債が増え続ける開発現場では、

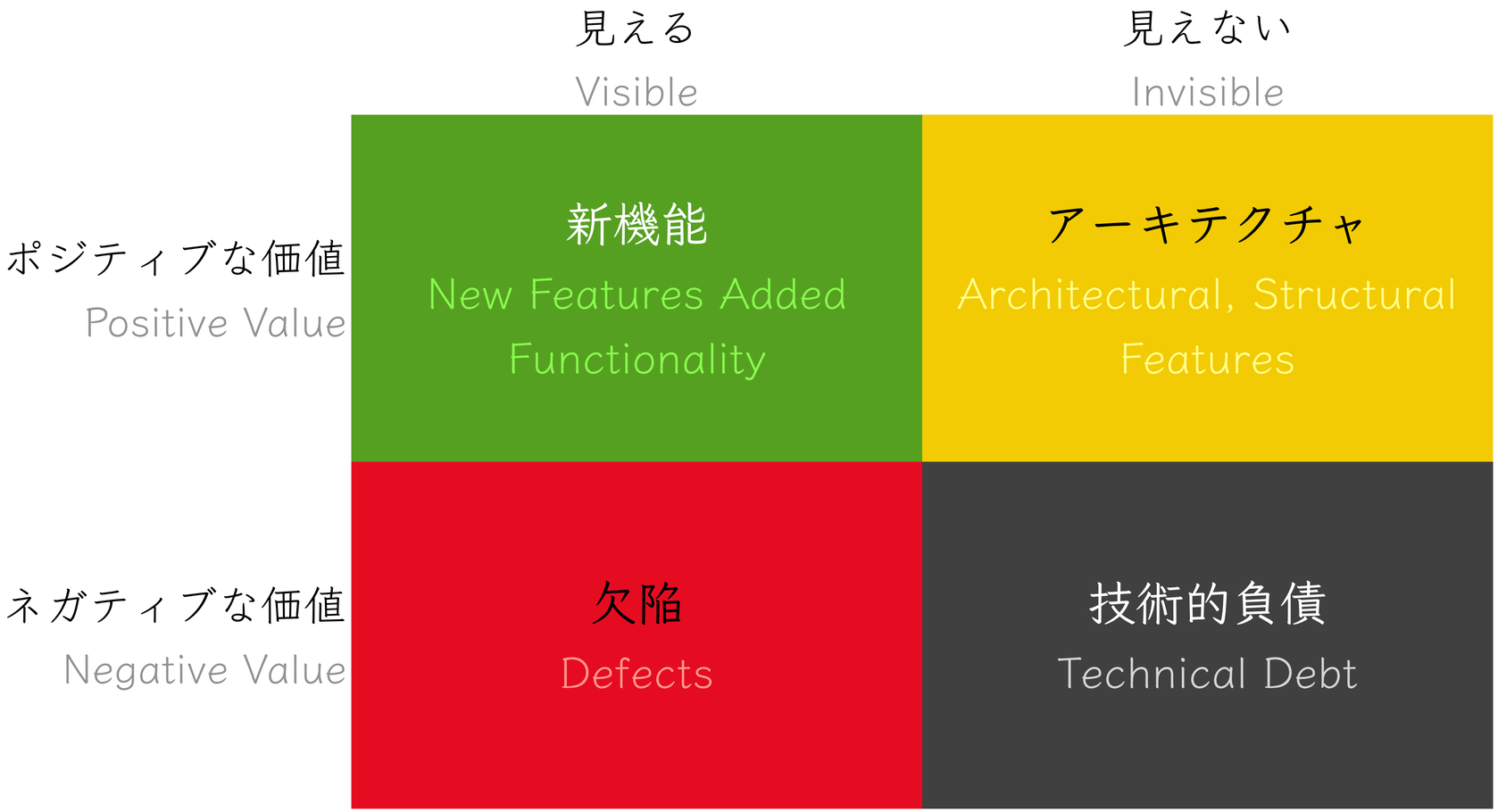
振る舞いの変更にばかり注力し、構造の価値が後まわしになる

技術的負債の蓄積によって**ソフトウェアがソフトでなくなっていく**

振る舞いの価値 > 構造の価値
大なり

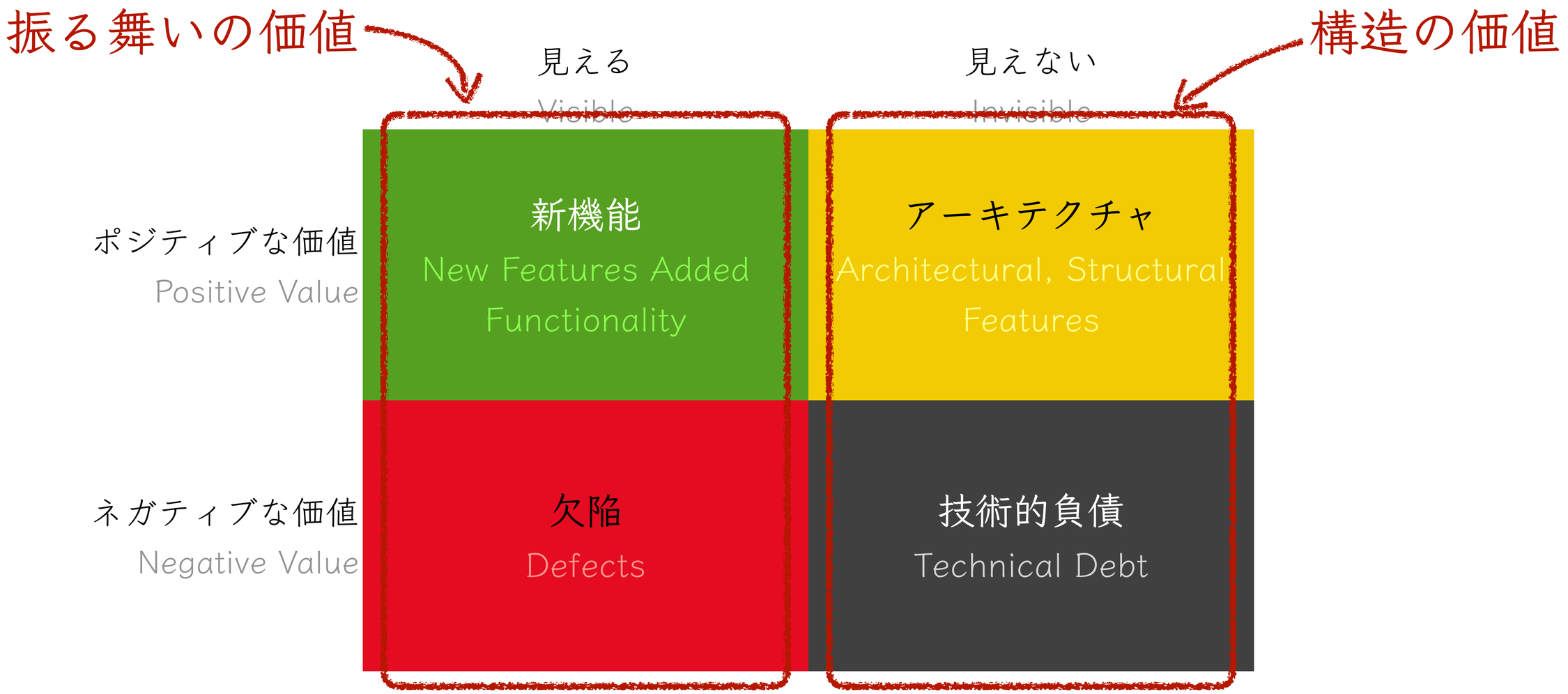
逆？なんでこうなる？

4色に塗り分けられたバックログ



※下記ページに掲載された図をもとに作成
The (missing) value of software architecture | Philippe Kruchten
https://philippe.kruchten.com/2013/12/11/the-missing-value-of-software-architecture/

振る舞いの価値は見える、構造の価値は見えない



※下記ページに掲載された図をもとに作成

The (missing) value of software architecture | Philippe Kruchten

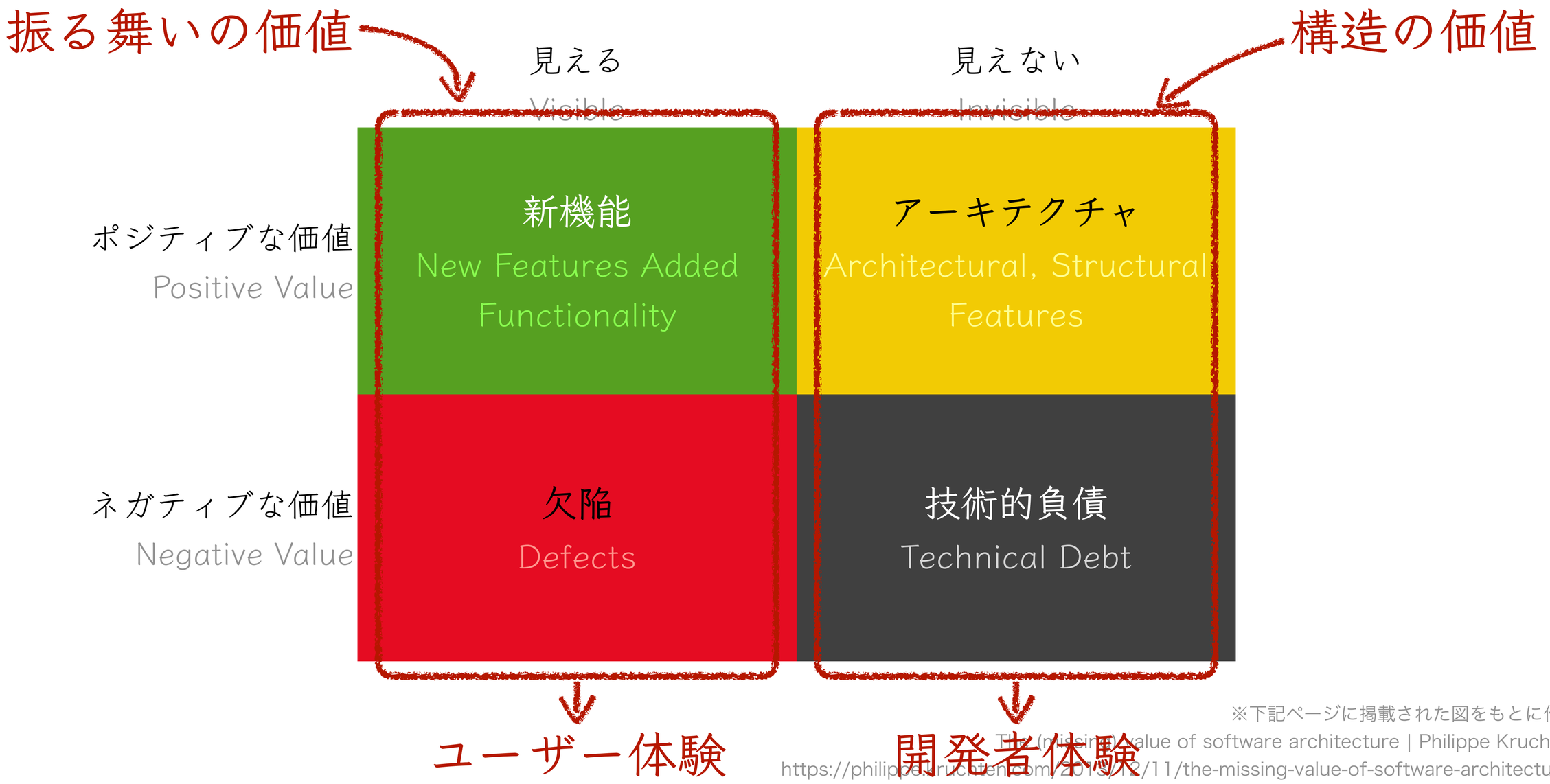
https://philippe.kruchten.com/2013/12/11/the-missing-value-of-software-architecture/

誰に見える/見えないのか？

- 「振る舞いの価値」はユーザーが体験できるもの
- 「構造の価値」はユーザーが体験できないもの

- 「構造の価値」は開発者が体験できるもの

2つの価値と2つの体験の関係



※下記ページに掲載された図をもとに作成

The (missing) value of software architecture | Philippe Kruchten
https://philippekruchten.com/2015/12/11/the-missing-value-of-software-architecture/

開発チームの時間全てを「見える価値」に投下

技術的負債が増え続ける開発現場では……

ステークホルダーの関心は、ユーザー体験に影響する「見える価値」、つまり**「振る舞い」に対してのみ向けられる**ので、開発チームの限りある時間を全てその実現に費やそうとする
そうして**短期的にはユーザー体験が向上**する

技術的負債が増え続ける開発現場では……

一方で構造には技術的負債が蓄積され続け、ソフトウェアがソフトでなくなっていき、**開発者体験は悪化**を続ける
振る舞いの変更は困難になり、**ユーザー体験の向上も停滞**しだす

「短期的」の先

このような**無謀な開発**には**持続性がない**

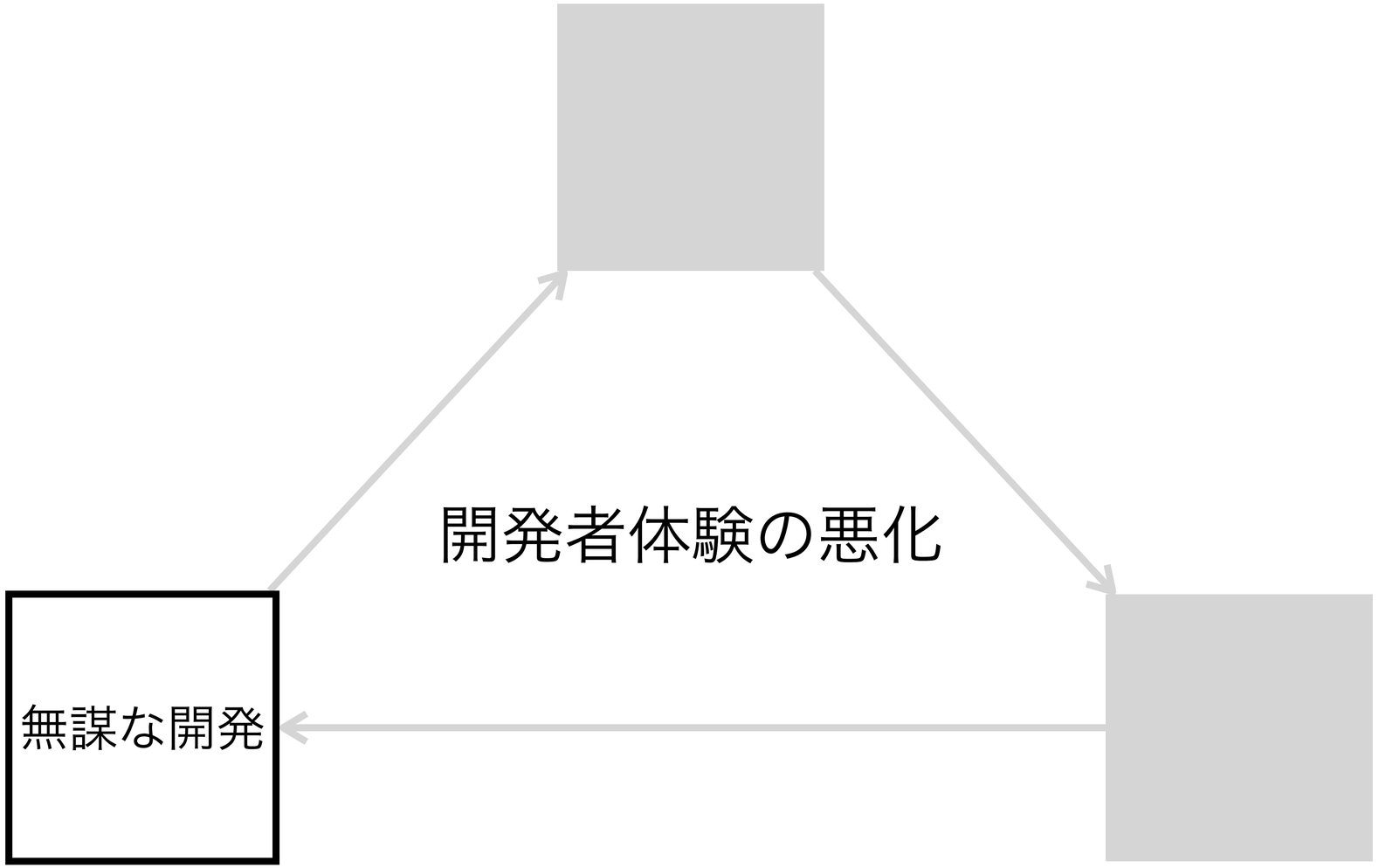
アジェンダ

- 2つの体験の似た特徴
- 振る舞いの価値、構造の価値
- 無謀な開発による悪循環
- 無謀な負債、慎重な負債
- 2つの体験の決定的な違い

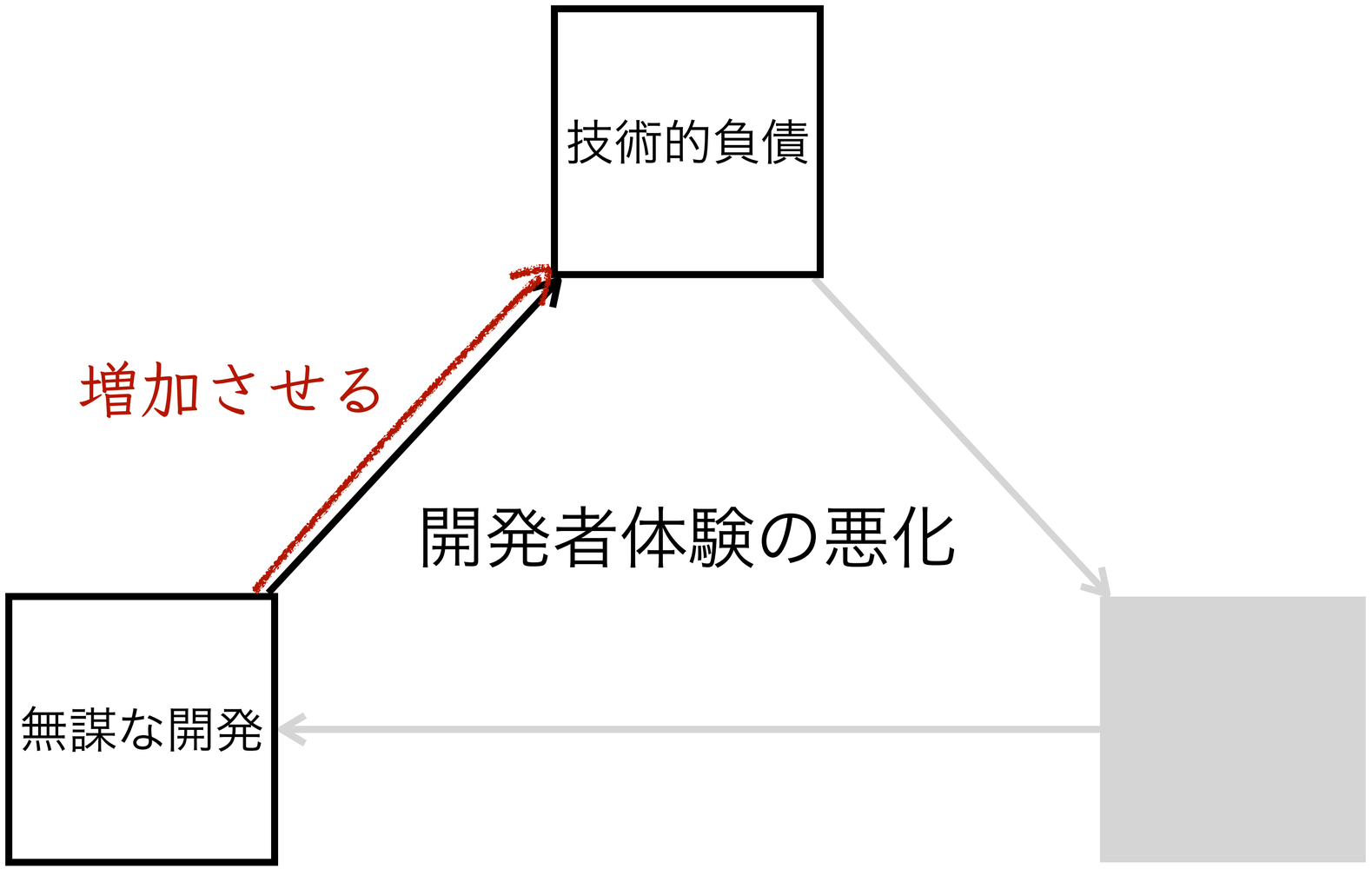
無謀な開発による悪循環

無謀な開発は、開発者体験悪化の**悪循環**を生む

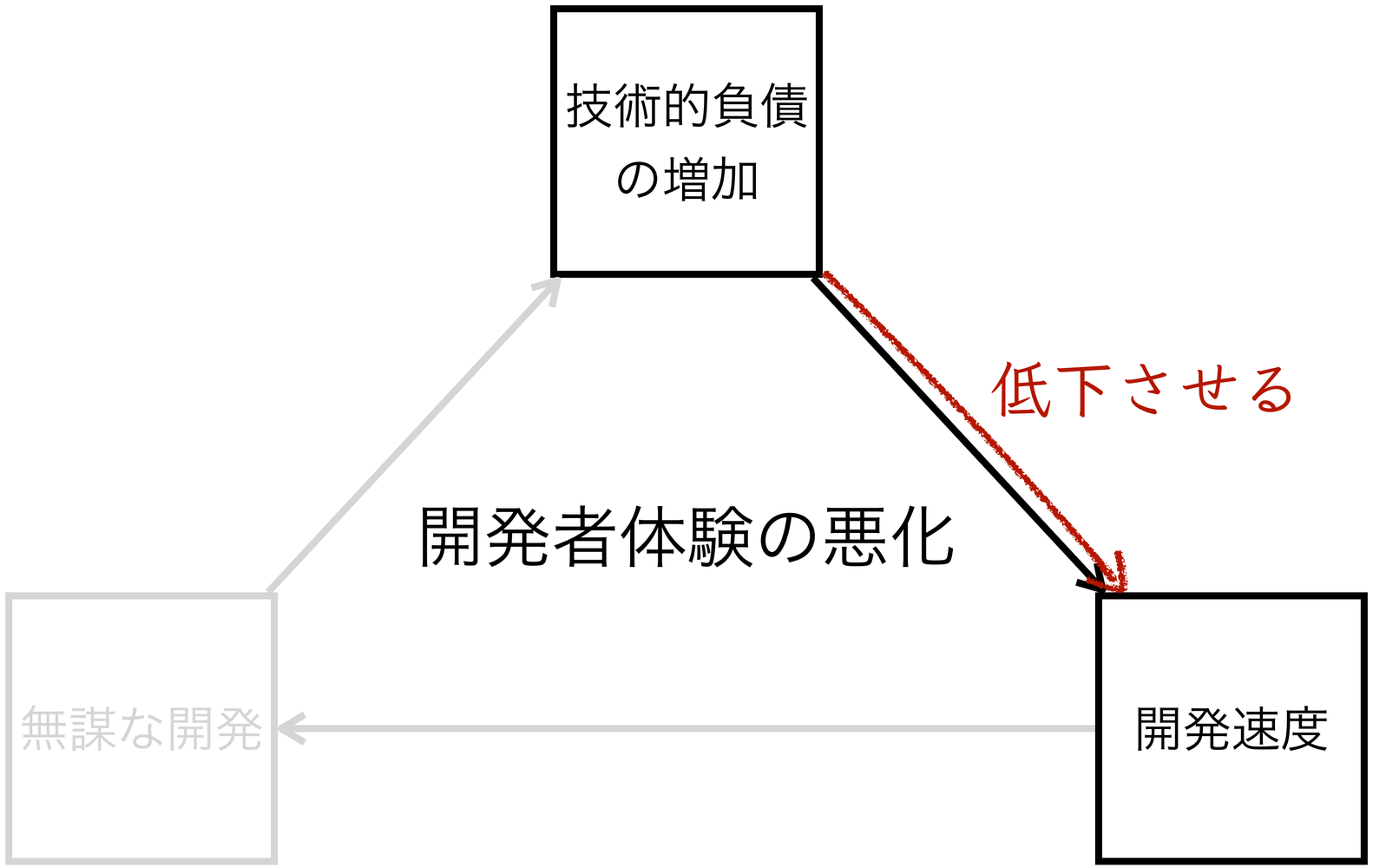
時間へのプレッシャーなどから無謀な開発が行われる



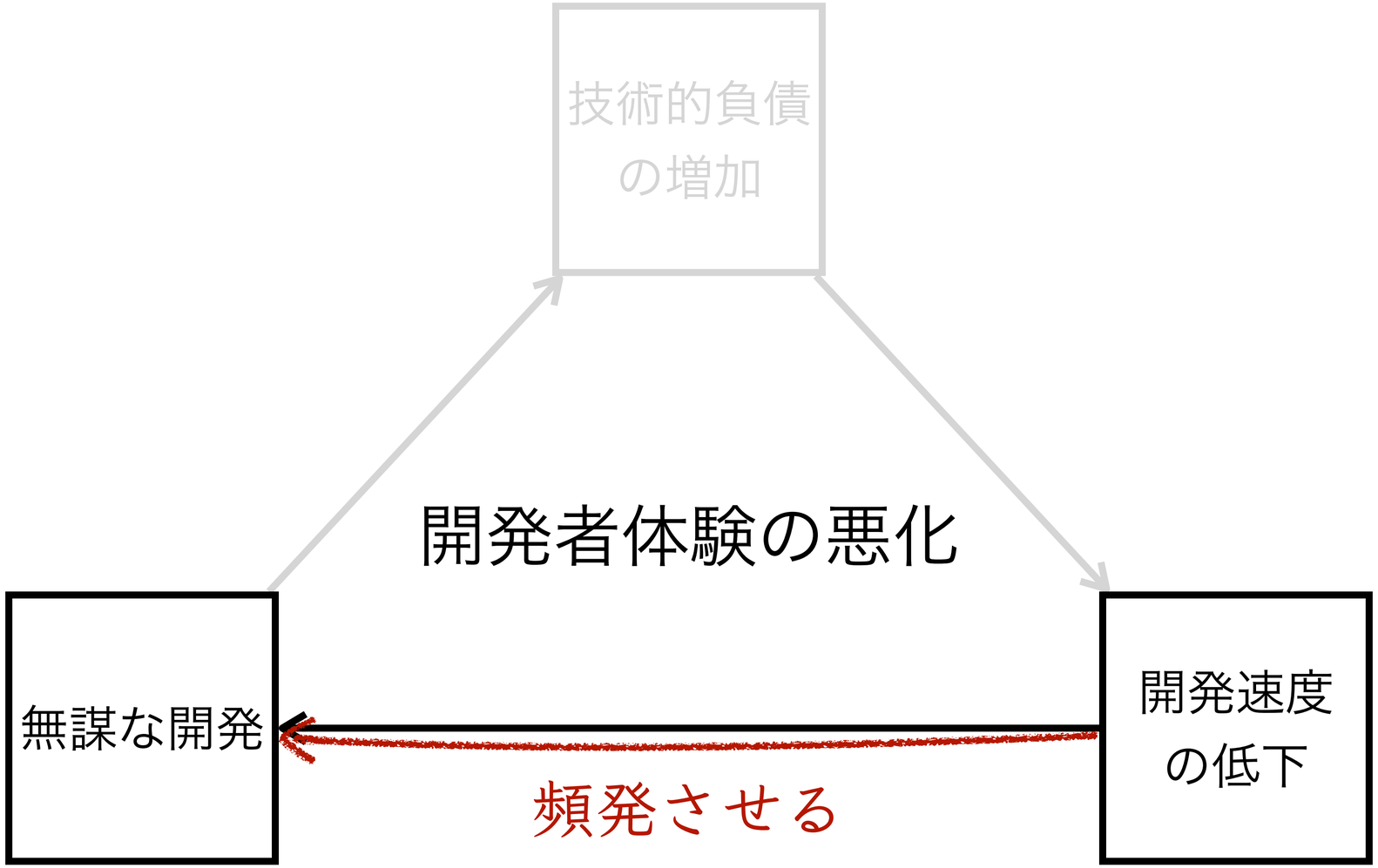
無謀な開発が技術的負債を増加させる



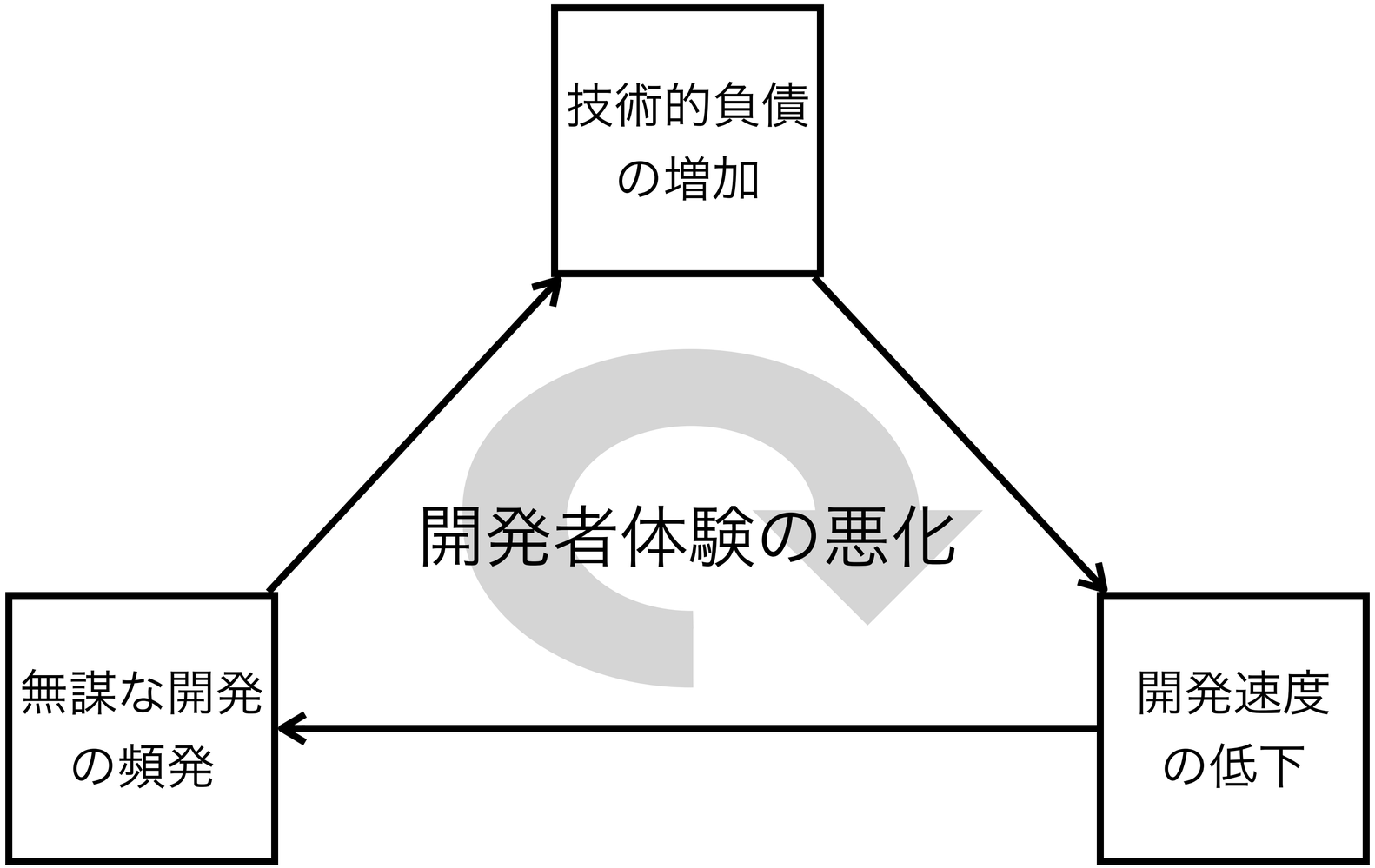
技術的負債の増加が開発速度を低下させる



開発速度の低下が無謀な開発を頻発させる



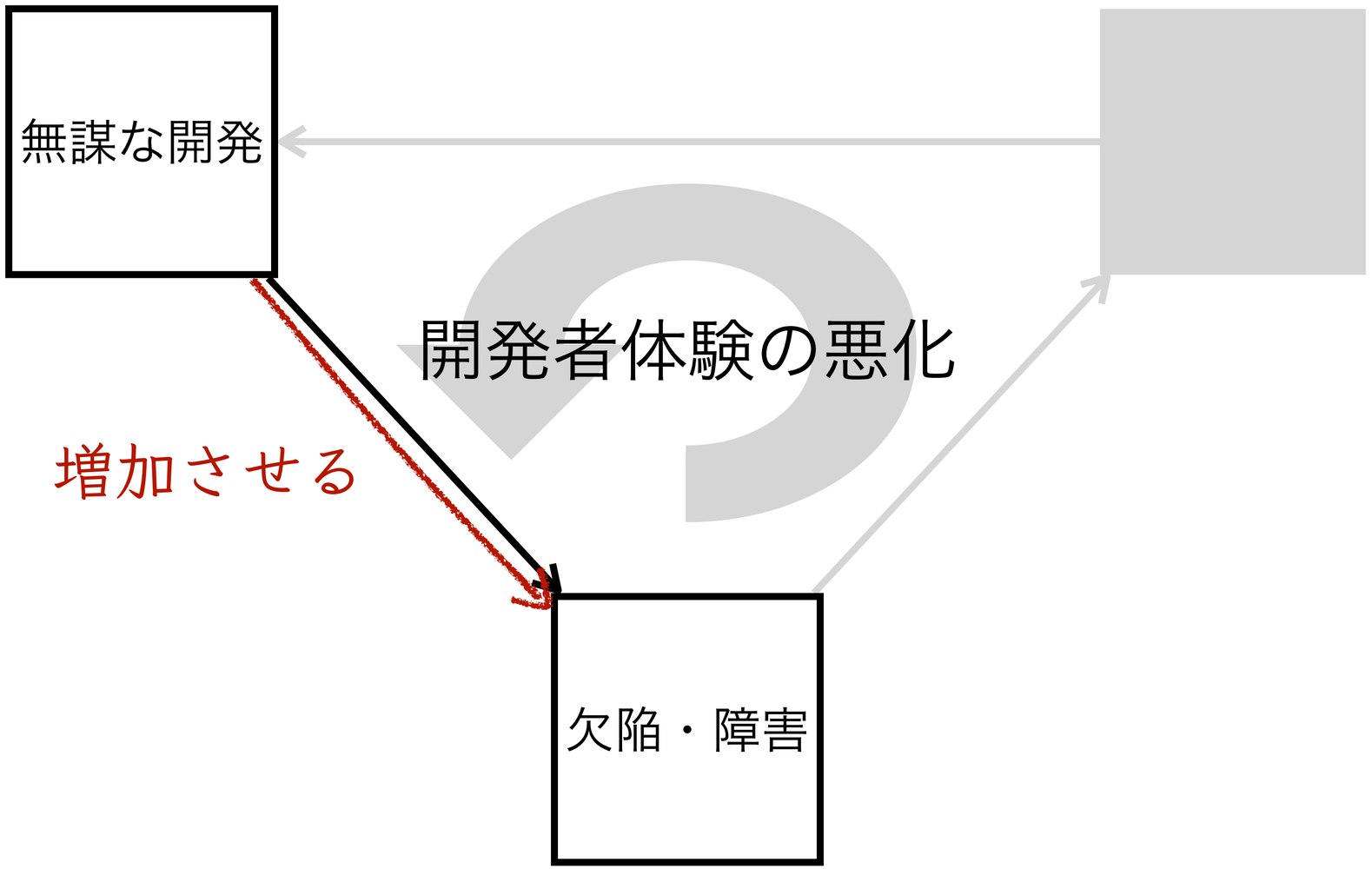
無謀な開発による開発者体験悪化の悪循環の完成



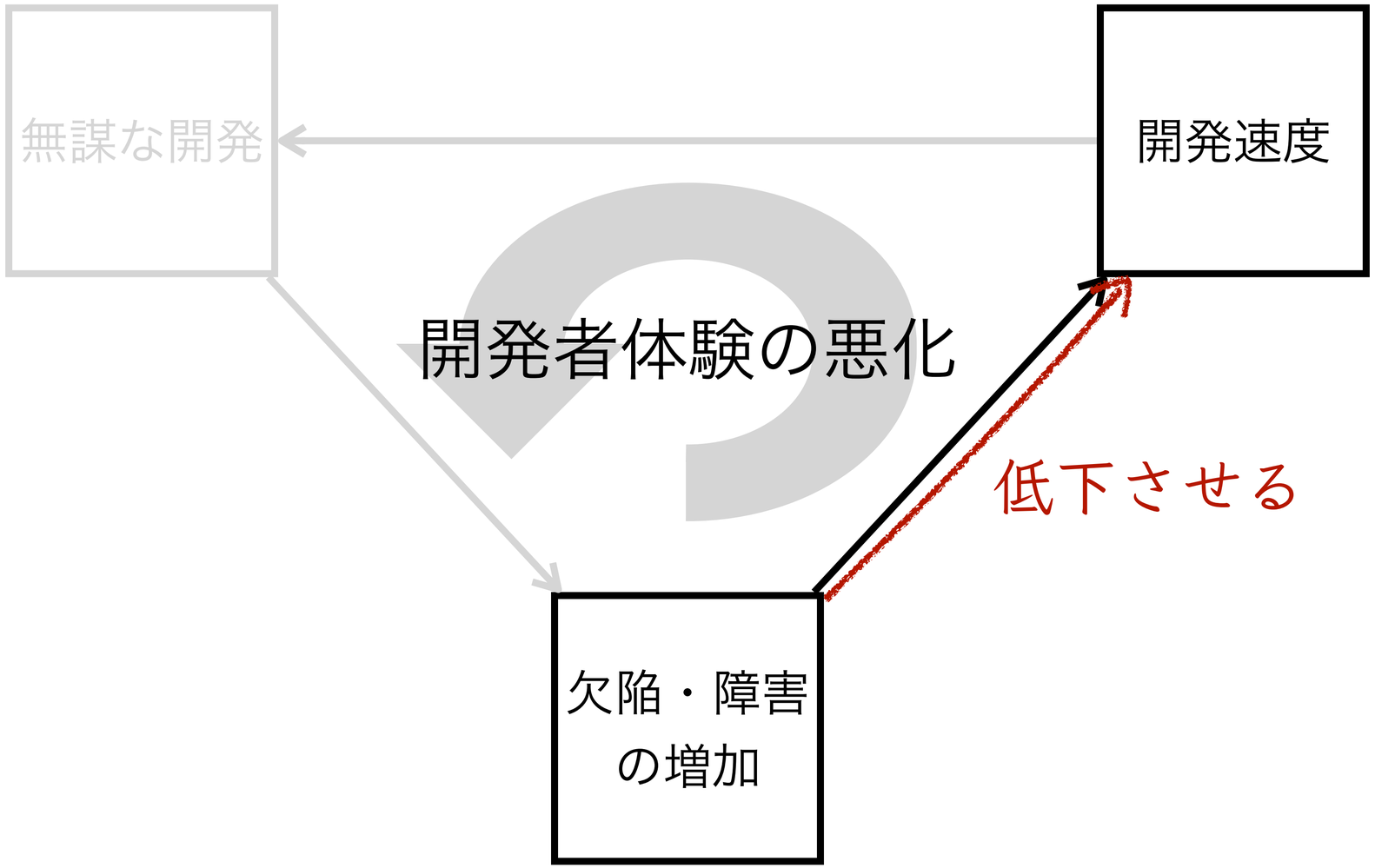
だけど……

しばらくすると、**もう一つの悪循環が動き出す**

無謀な開発が欠陥・障害を増加させる

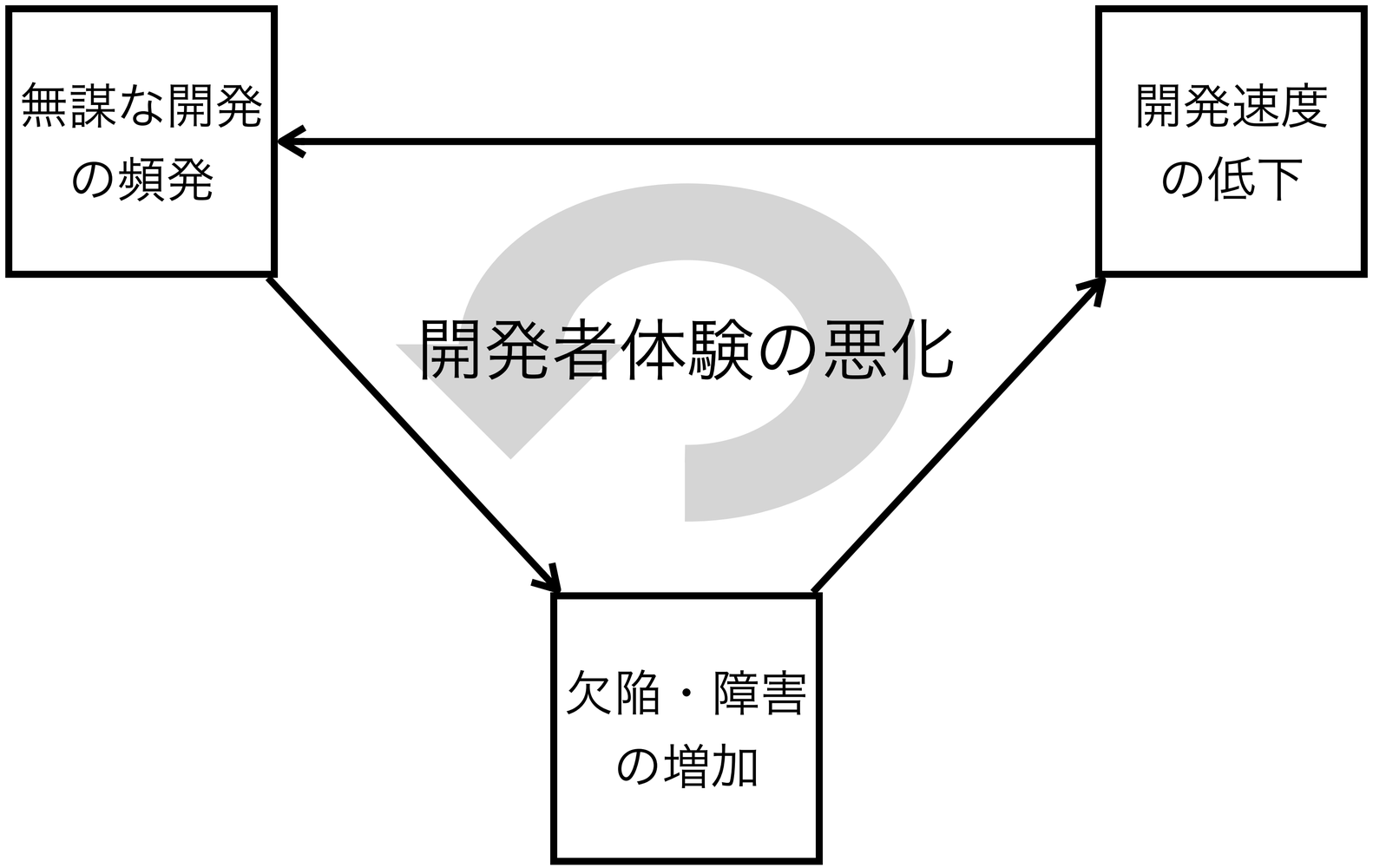


欠陥・障害の増加が開発速度を低下させる

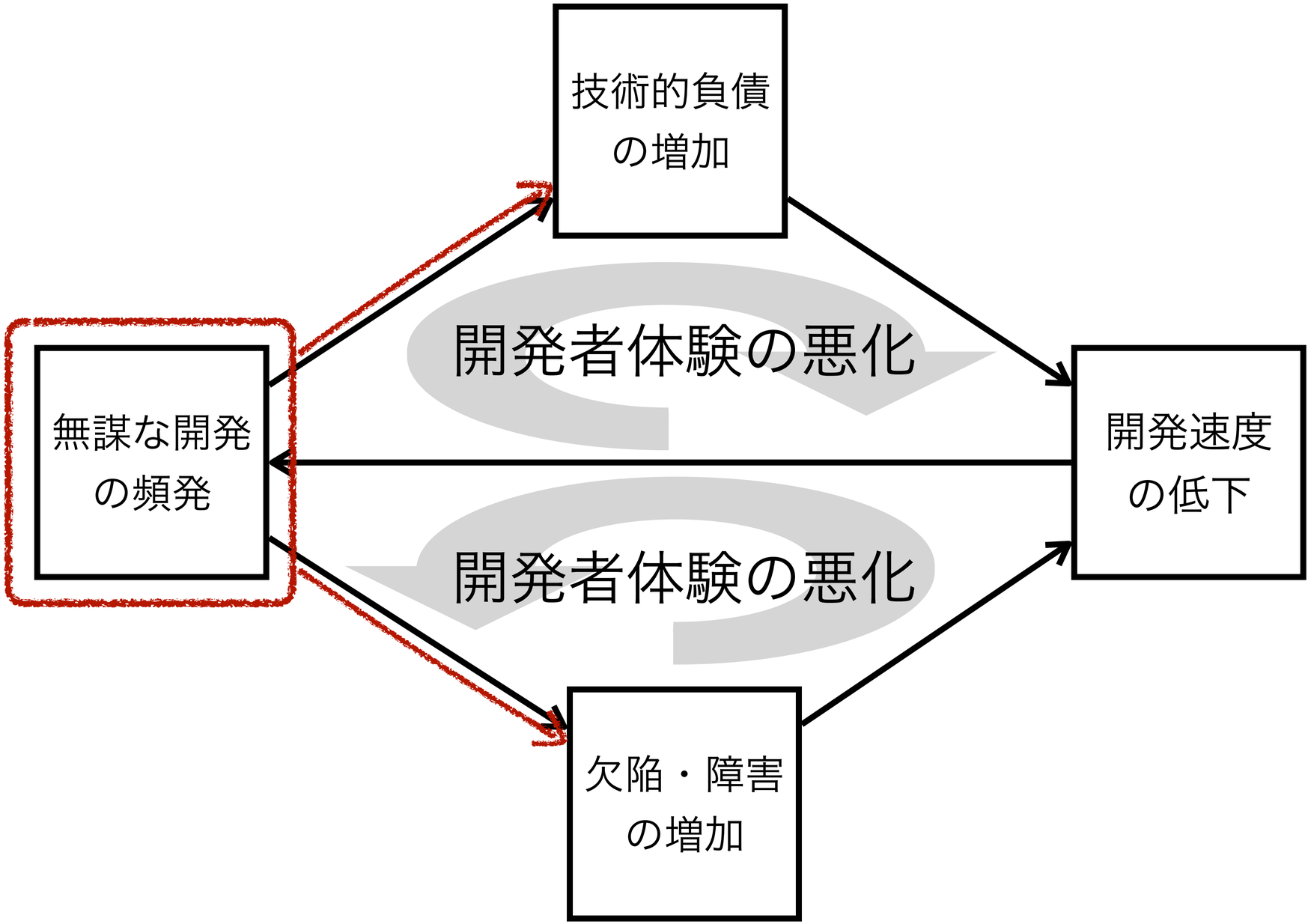


開発速度の低下が無謀な開発を頻発させる

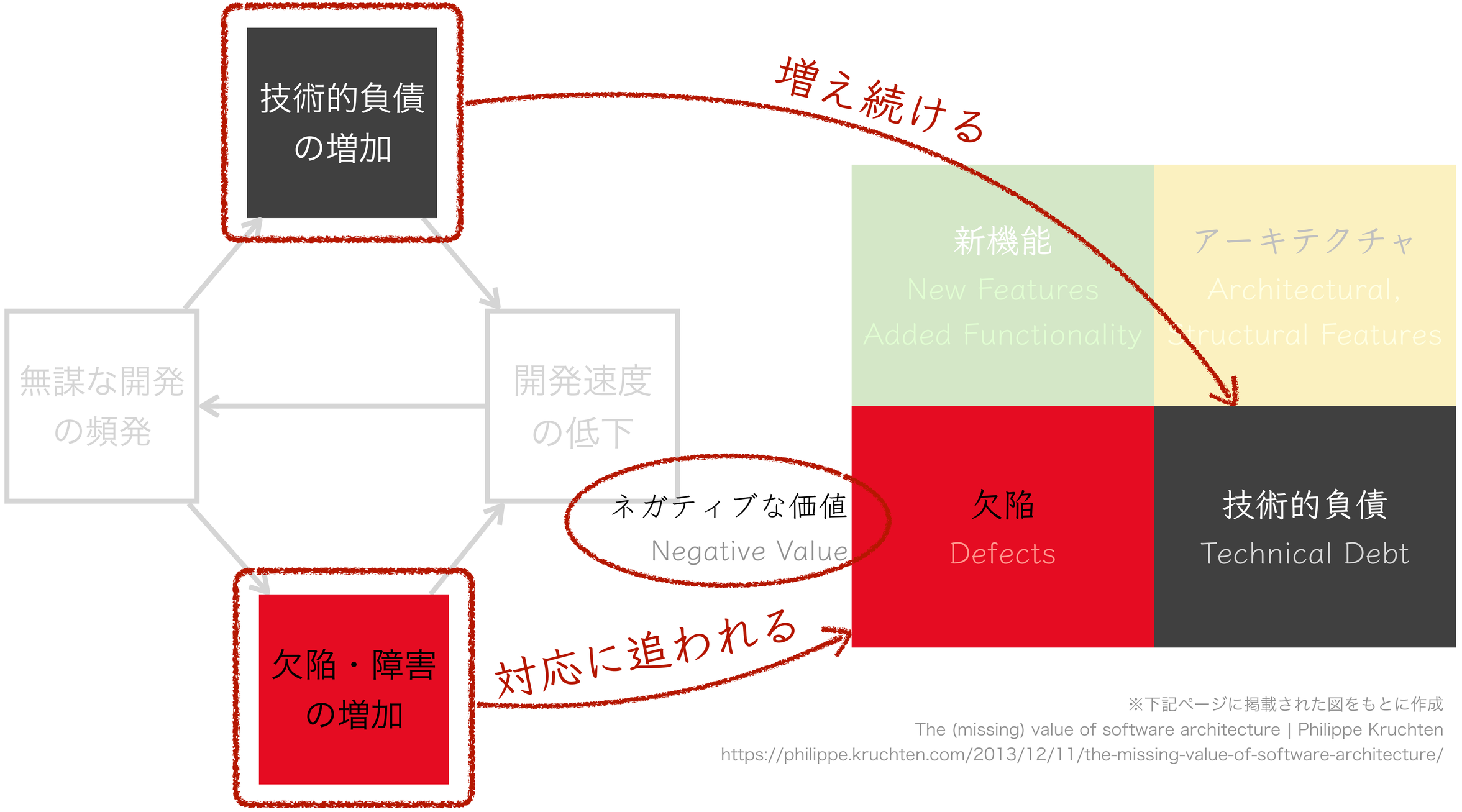
2つめの悪循環の完成



「無謀な開発」が2つの悪循環をまわし続けている



ユーザー体験も開発者体験もボロボロに



※下記ページに掲載された図をもとに作成
The (missing) value of software architecture | Philippe Kruchten
https://philippe.kruchten.com/2013/12/11/the-missing-value-of-software-architecture/

アジェンダ

- 2つの体験の似た特徴
- 振る舞いの価値、構造の価値
- 無謀な開発による悪循環
- 無謀な負債、慎重な負債
- 2つの体験の決定的な違い

構造の価値とは？

ソフトウェアをソフトにする

“優れたアーキテクチャがあれば、システムを容易に理解・開発・保守・デプロイできる。最終的な目的は、システムのライフタイムコストを最小限に抑え、プログラマの生産性を最大にすることである。”

『Clean Architecture』 第15章 アーキテクチャとは？

構造の価値を構成する主な特性

「ソフトウェアをソフトに維持し続ける」ための特性

保守性の3つの品質特性

- 理解容易性 ← リーダブル、コメント、ドキュメント
- 変更容易性 ← BDUFとENUFのバランス、適応度関数
- テスト容易性 ← テストピラミッドの下層へ
  （壊れにくくて統合環境が不要なテスト）

デリバリパフォーマンスに影響するアーキテクチャ特性

- テスト容易性
- デプロイ容易性 ← チームとアーキテクチャが疎結合

開発者なら構造の価値は「見える」のか？

開発者が構造の価値を正しく評価できるとは**限らない**

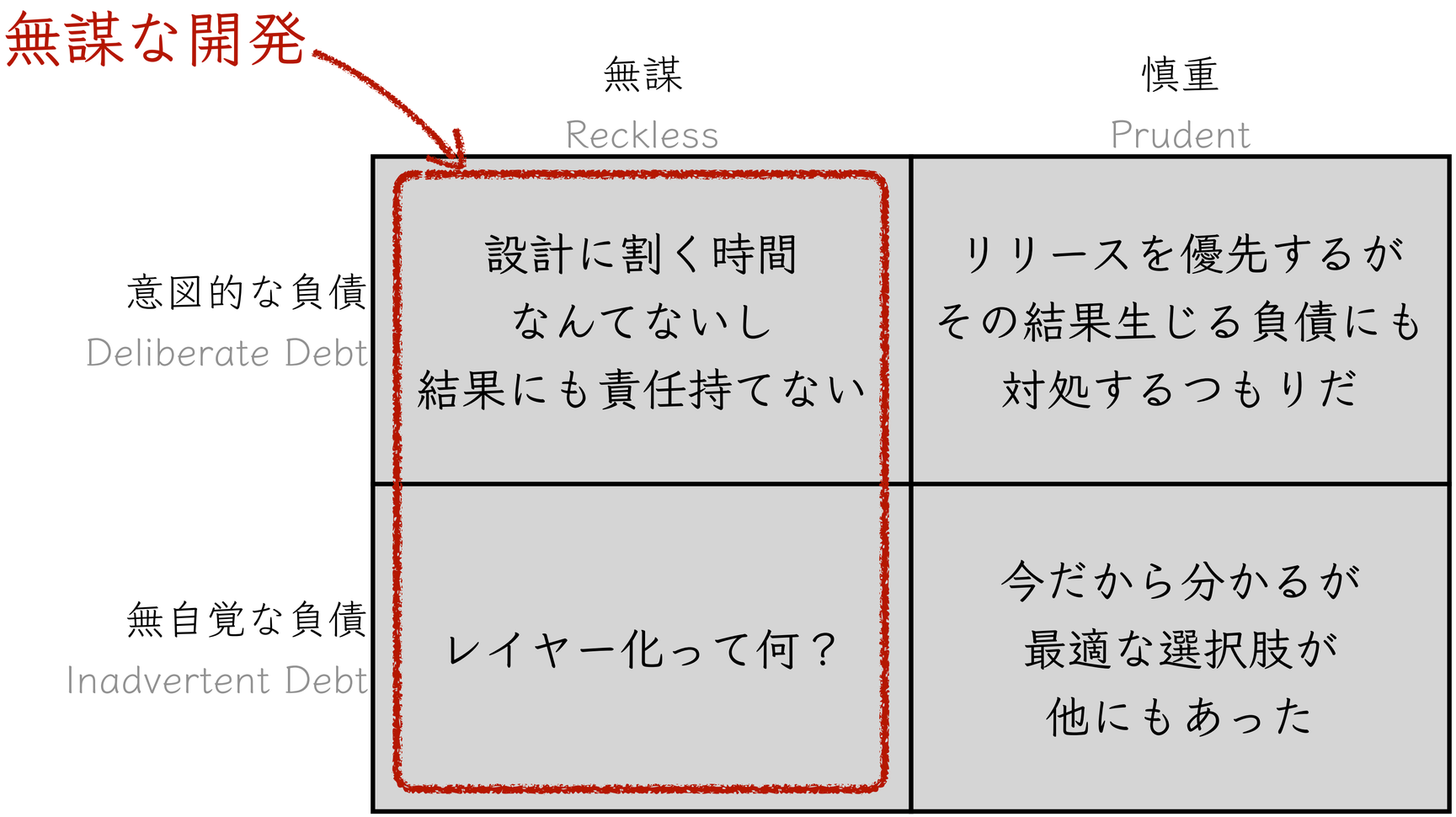
技術的負債はビジネスやユーザーの
不理解から生じるとは限らない

技術的負債が生み出される4つの理由

	無謀 Reckless	慎重 Prudent
意図的な負債 Deliberate Debt	設計に割く時間 なんてないし 結果にも責任持てない	リリースを優先するが その結果生じる負債にも 対処するつもりだ
無自覚な負債 Inadvertent Debt	レイヤー化って何？	今だから分かるが 最適な選択肢が 他にもあった

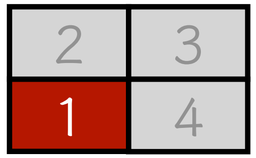
※下記ページに掲載された図をもとに作成
TechnicalDebtQuadrant | martinfowler.com
https://martinfowler.com/bliki/TechnicalDebtQuadrant.html

左側は「無謀な開発」による無謀な負債



※下記ページに掲載された図をもとに作成
TechnicalDebtQuadrant | martinfowler.com
https://martinfowler.com/bliki/TechnicalDebtQuadrant.html

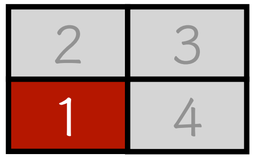
無謀 & 無自覚な負債



	無謀 Reckless	慎重 Prudent
意図的な負債 Deliberate Debt	設計に割く時間 なんてないし 結果にも責任持てない	リリースを優先するが その結果生じる負債にも 対処するつもりだ
無自覚な負債 Inadvertent Debt	(1) レイヤー化って何？	今だから分かるが 最適な選択肢が 他にもあった

※下記ページに掲載された図をもとに作成
TechnicalDebtQuadrant | martinfowler.com
https://martinfowler.com/bliki/TechnicalDebtQuadrant.html

無謀 & 無自覚な負債

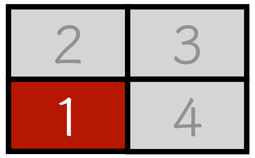


“A team **ignorant of design practices** is taking on its reckless debt without even realizing how much hock it's getting into.”

デザインプラクティスに無知なチームは、無謀な負債を背負うのだが、自らがどれほどの負債を背負っているかに**気づいていない**。

TechnicalDebtQuadrant | martinfowler.com
https://martinfowler.com/bliki/TechnicalDebtQuadrant.html

無謀 & 無自覚な負債

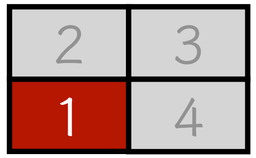


知識不足・スキル不足

負債が生じる原因：

- 優れたデザインプラクティスを知らない
- テストコードを書いたりリファクタリングするスキルを有していない

無謀 & 無自覚な負債

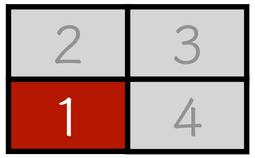


知識不足・スキル不足

負債に対する行動

- 負債の存在に気付いていないから返済しない
- あるいは、返済しようとして余計に負債を作り出してしまう

無謀 & 無自覚な負債



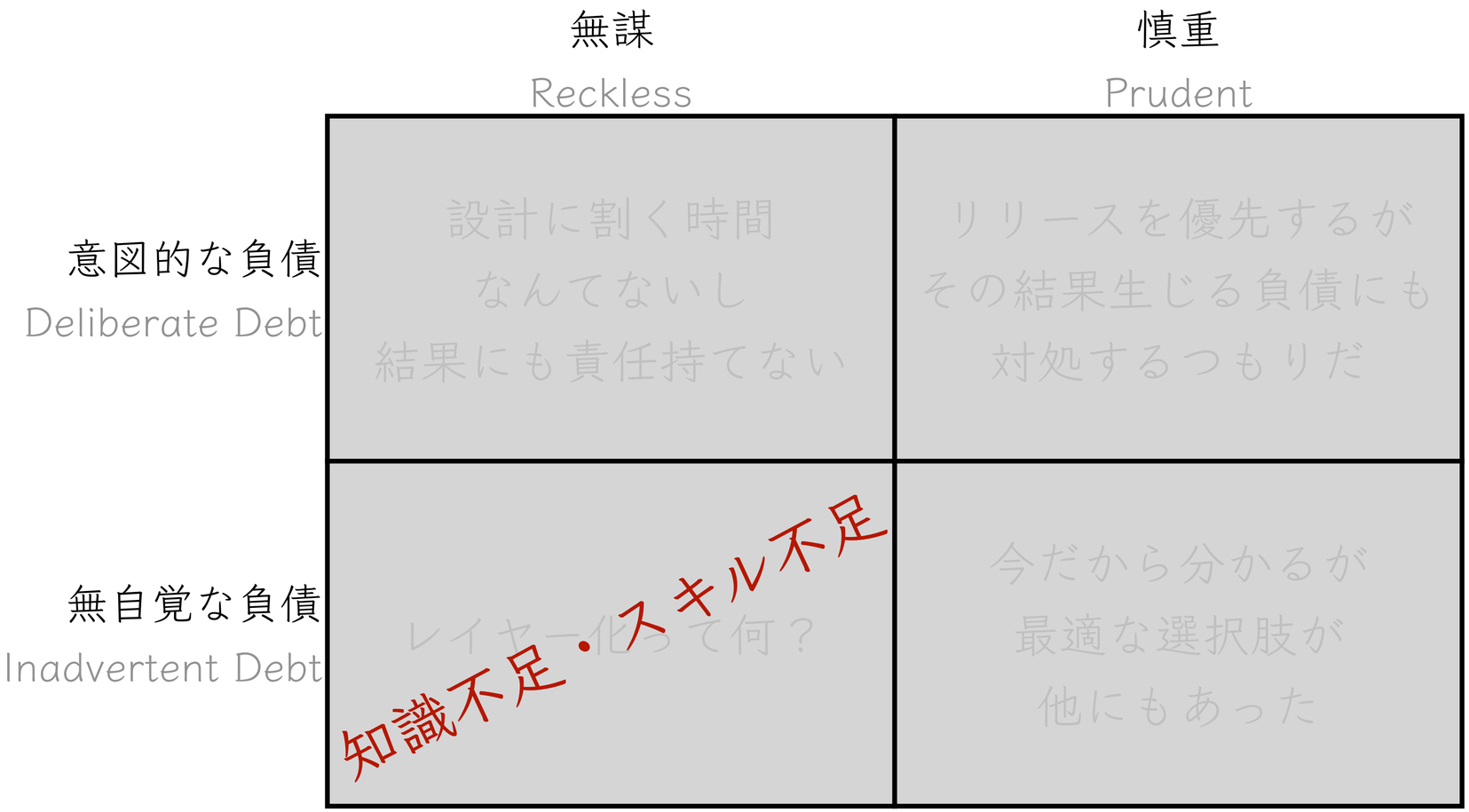
知識不足・スキル不足

問題の背景（解決すべき点）

- 日々の開発業務が忙しすぎて、開発者が知識やスキルを高める機会・環境がない組織
- 開発規模に合わせて大急ぎで人をかき集め、人数だけを揃えたプロジェクト
- 本番稼働による運用・保守からのフィードバックが得られない開発専任チーム

無謀 & 無自覚な負債は「知識不足・スキル不足」によって生じる

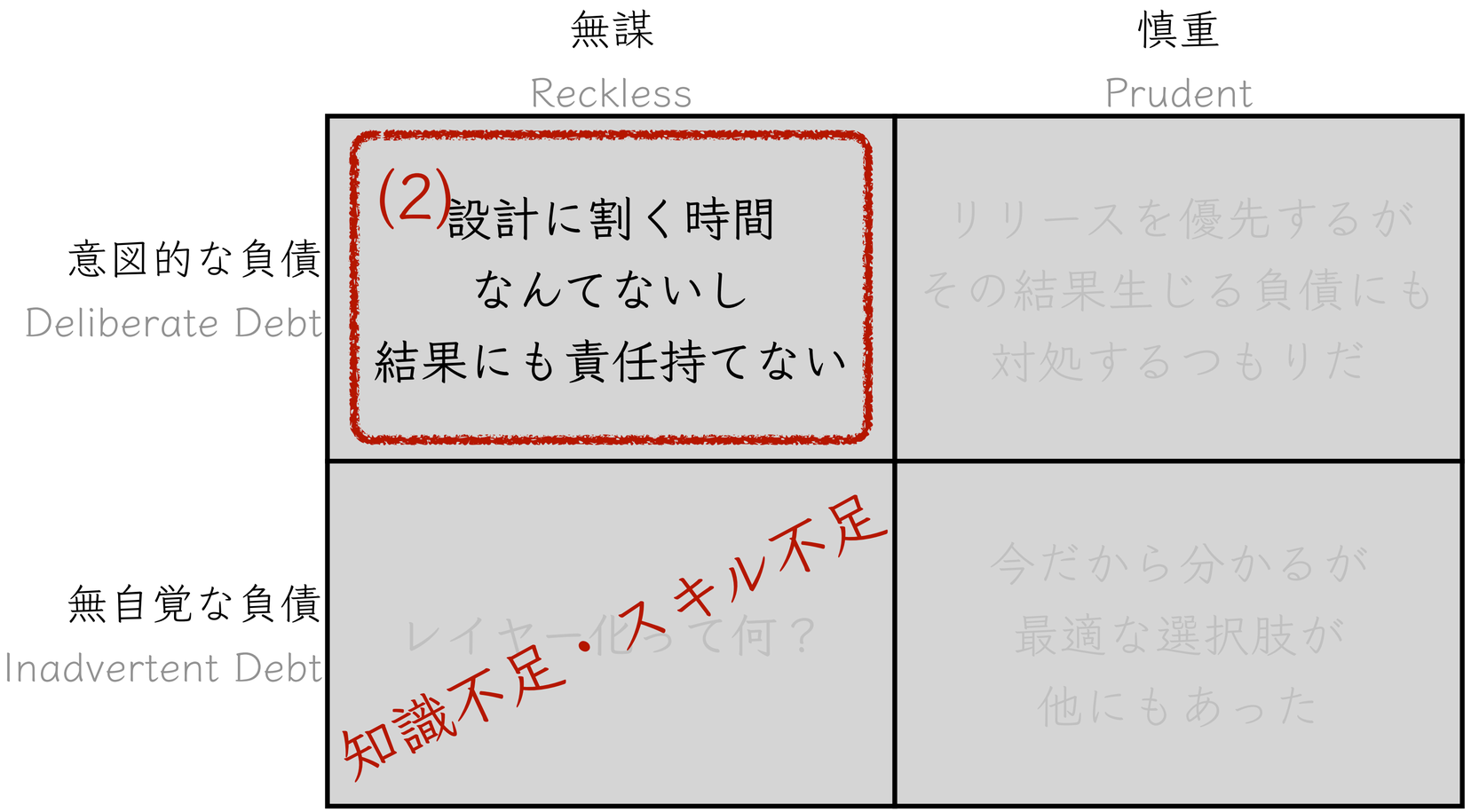
2	3
1	4



※下記ページに掲載された図をもとに作成
TechnicalDebtQuadrant | martinfowler.com
https://martinfowler.com/bliki/TechnicalDebtQuadrant.html

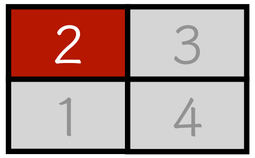
無謀 & 意図的な負債

2	3
1	4



※下記ページに掲載された図をもとに作成
TechnicalDebtQuadrant | martinfowler.com
https://martinfowler.com/bliki/TechnicalDebtQuadrant.html

無謀 & 意図的な負債



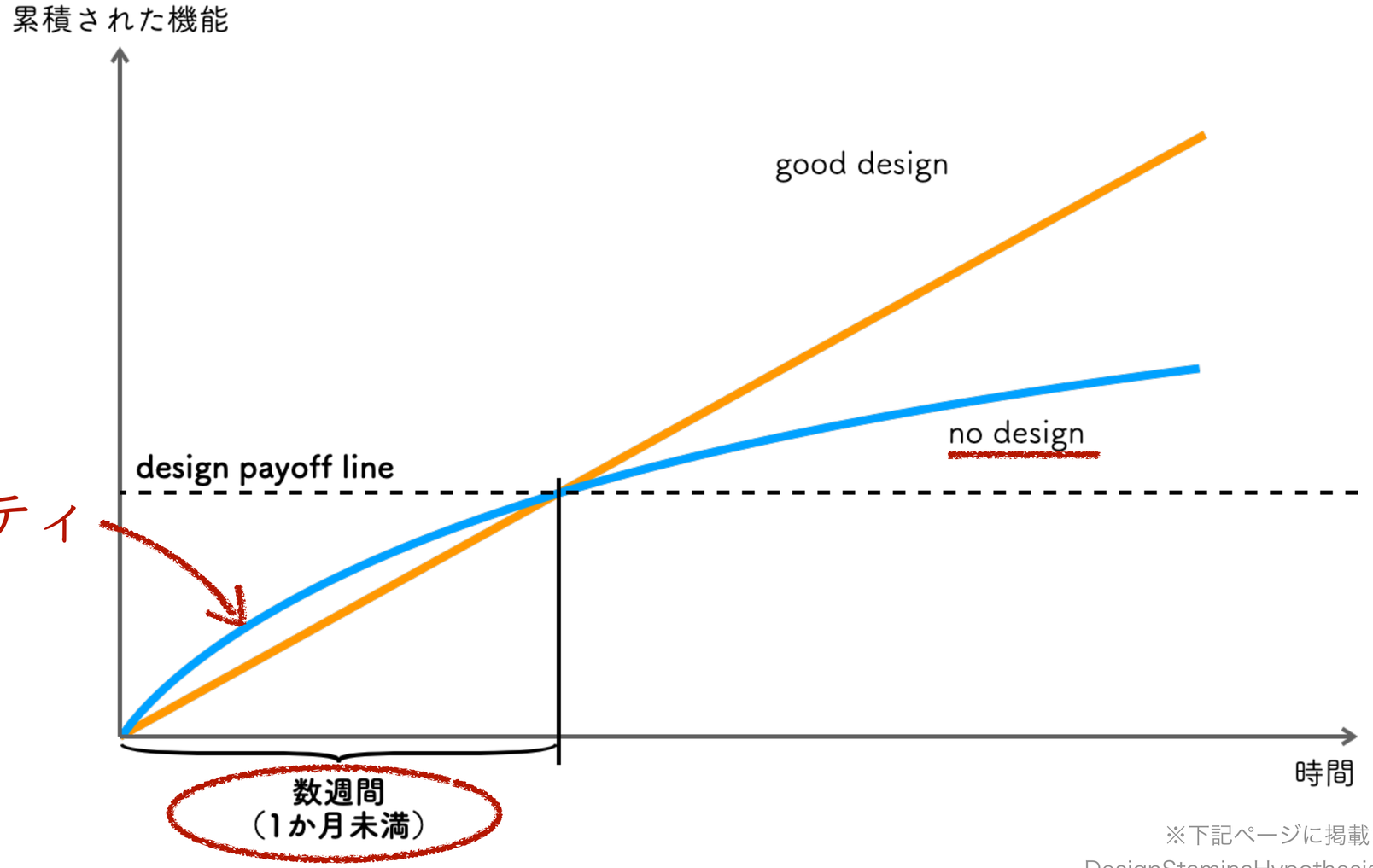
“A team may know about good design practices, even be capable of practicing them, but decide to go "quick and dirty" because they think they can't afford the time required to write clean code. I agree with Uncle Bob that this is usually a reckless debt, because people underestimate where the DesignPayoffLine is. The whole point of good design and clean code is to make you go faster”

チームは優れたデザインプラクティスを知っているし、それを実践する能力もあるが、「クイック&ダーティ」でやろうと決めている。彼らはクリーンなコードを書くための時間が無いと考えているのだ。このやり方はたいてい無謀な負債になるというアンクル・ボブの意見に私は賛成だ。みんな、DesignPayoffLineを過小評価しているのだ。優れたデザインとクリーンなコードの本質は、チームをより速くするということだ。

TechnicalDebtQuadrant | martinfowler.com
https://martinfowler.com/bliki/TechnicalDebtQuadrant.html

Design Payoff Lineを過小評価している

2	3
1	4

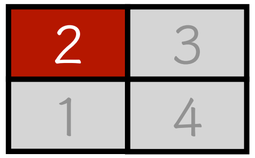


クイック&ダーティ

もっと先だと甘く見ている

※下記ページに掲載された図をもとに作成
DesignStaminaHypothesis | martinfowler.com
https://martinfowler.com/bliki/DesignStaminaHypothesis.html

スピードと品質

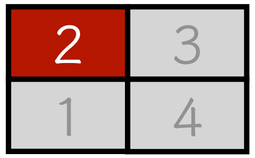


“A team may know about good design practices, even be capable of practicing them, but decide to go "quick and dirty" because they think they can't afford the time required to write clean code. I agree with Uncle Bob that this is usually a reckless debt, because people underestimate where the DesignPayoffLine is. The whole point of good design and clean code is to make you go faster”

チームは優れたデザインプラクティスを知っているし、それを実践する能力もあるが、「クイック&ダーティ」でやろうと決めている。彼らはクリーンなコードを書くための時間が無いと考えているのだ。このやり方はたいてい無謀な負債になるというアンクル・ボブの意見に私は賛成だ。みんな、DesignPayoffLineを過小評価しているのだ。優れたデザインとクリーンなコードの本質は、チームをより速くするということだ。

TechnicalDebtQuadrant | martinfowler.com
https://martinfowler.com/bliki/TechnicalDebtQuadrant.html

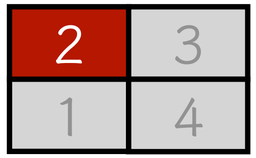
スピードと品質



~~品質と~~**~~引き換えに~~**スピードが得られる

品質を重視すれば

無謀 & 意図的な負債

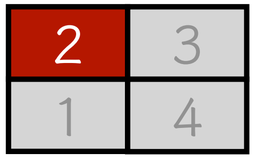


クイック&ダーティ

負債が生じる原因：

- Design Payoff Lineを過小評価している
- とりあえず「振る舞いが実現できればOK」という思考に陥り、本来やるべきことをやらずに突き進む

無謀 & 意図的な負債

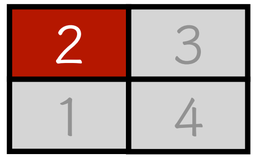


クイック&ダーティ

本来やるべきことをやらずに突き進む

↓ 技術的負債を生み出す

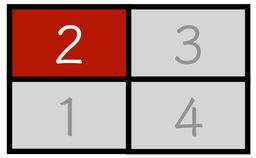
- 設計しない
- リファクタリングしない
- テストコードを書かない
- 適切なコメントを書かない
- 適切なドキュメントを作成しない



クイック&ダーティ

負債に対する行動

- あと回しを繰り返し、気付いた時には手遅れになる
- 負債にまみれるうち返済するモチベーションが失われてしまう



クイック&ダーティ

問題の背景（解決すべき点）

- 期日もスコープも**動かせない**、実質的にトレードオフスライダーが壊れたプロジェクト 時間、予算、品質、スコープすべて固定化
- 技術的負債にまみれた状態が**当たり前**になって、感覚が麻痺している開発チーム つまり「割れ窓」
- 引き継ぎなどで、コードに対するオーナーシップ意識が**欠落**した開発チーム 「前任者たちのせいだし」

無謀 & 意図的な負債は「クイック&ダーティ」によって生じる

2	3
1	4

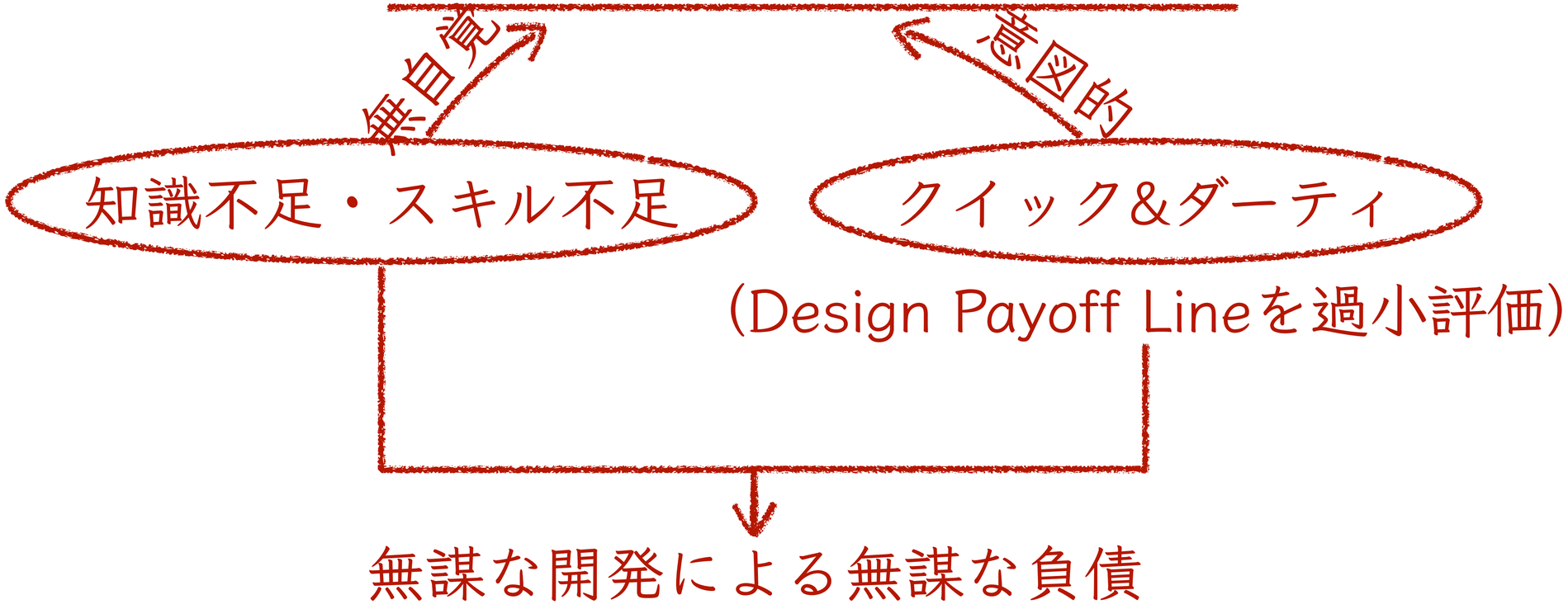
	無謀 Reckless	慎重 Prudent
意図的な負債 Deliberate Debt	設計に割く時間 なんてないし 結果にも責任持てない クイック&ダーティ	リリースを優先するが その結果生じる負債にも 対処するつもりだ
無自覚な負債 Inadvertent Debt	レイヤー化って何？ 知識不足・スキル不足	今だから分かるが 最適な選択肢が 他にもあった

※下記ページに掲載された図をもとに作成
TechnicalDebtQuadrant | martinfowler.com
https://martinfowler.com/bliki/TechnicalDebtQuadrant.html

無謀な開発による無謀な負債が生じる背景

（再掲）

開発者が構造の価値を正しく評価できるとは**限らない**



「悪い負債」と「良い負債」

技術的負債は必ずしも悪者ではない

悪い負債

Bad Debt

1週間以上経っても返済されない
古い負債

良い負債

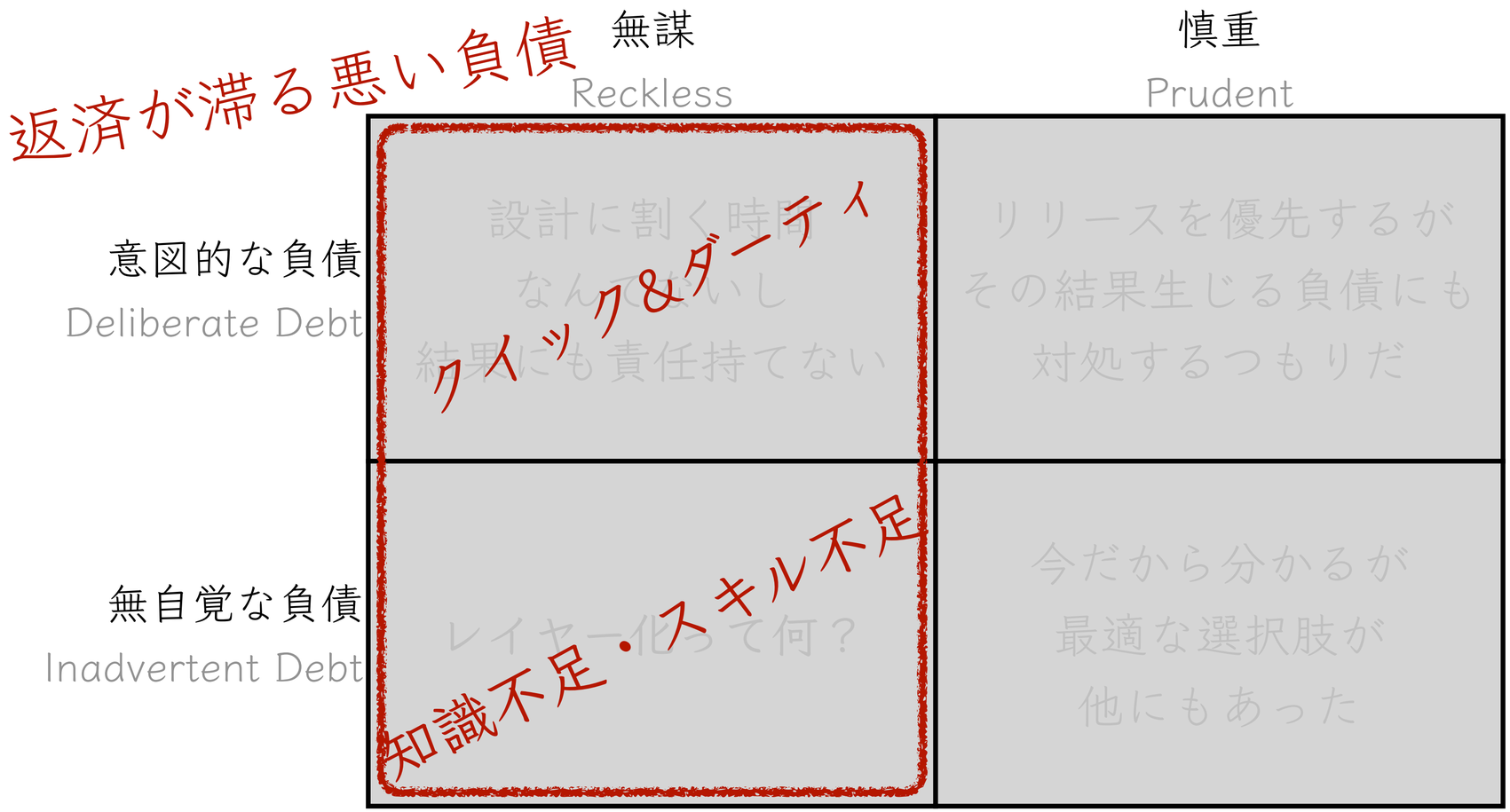
Good Debt

出来て一週間未満の
新しい負債

※参考ページ
Good and Bad Technical Debt (and how TDD helps) | Crisp's Blog
https://blog.crisp.se/2013/10/11/henrikkniberg/good-and-bad-technical-debt

無謀な負債は返済が滞る「悪い負債」

返済が滞る悪い負債



※下記ページに掲載された図をもとに作成
TechnicalDebtQuadrant | martinfowler.com
https://martinfowler.com/bliki/TechnicalDebtQuadrant.html

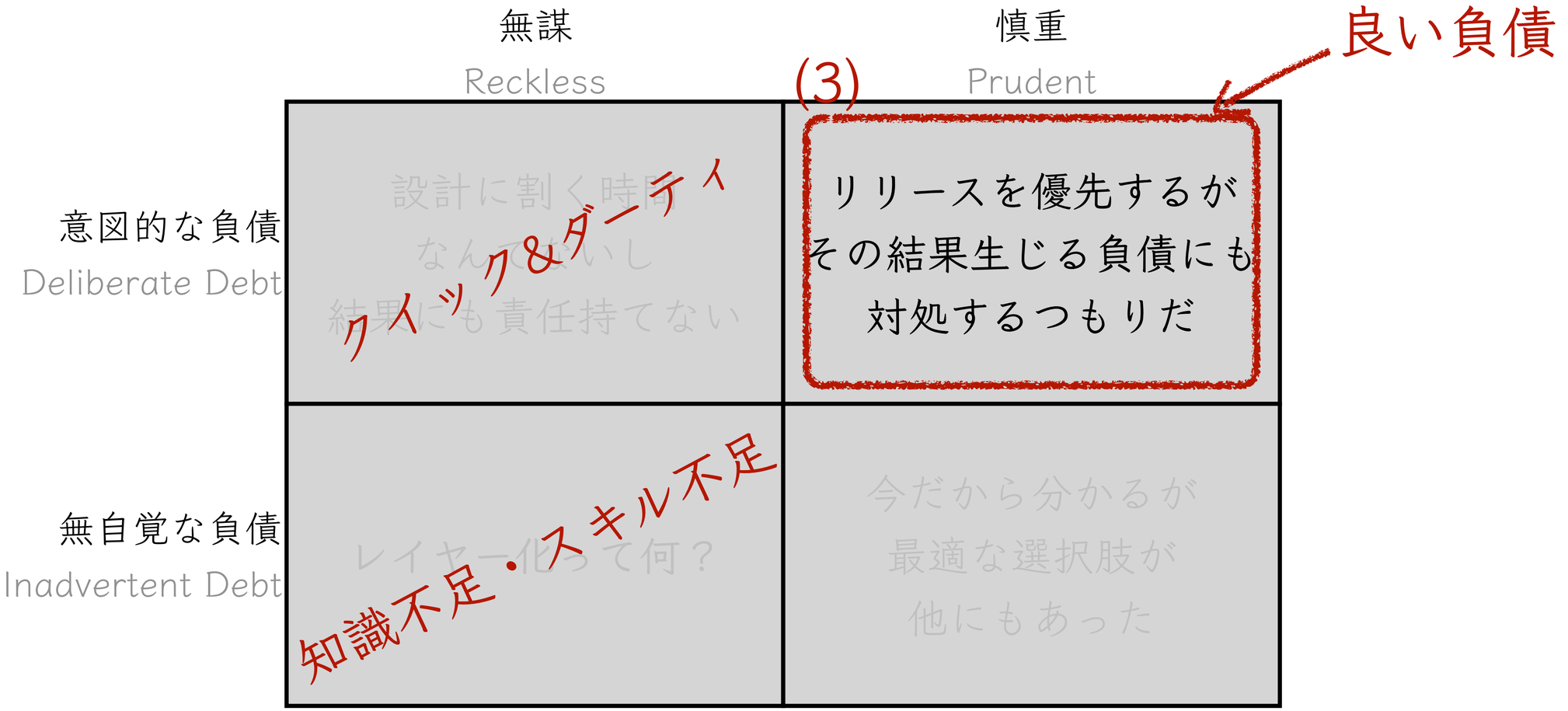
タイトルを正しく書き直す

悪い技術的負債

技術的負債は開発者体験を悪化させる

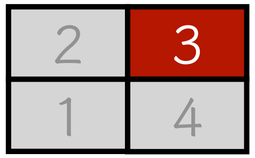
慎重 & 意図的な負債

2	3
1	4



※下記ページに掲載された図をもとに作成
TechnicalDebtQuadrant | martinfowler.com
https://martinfowler.com/bliki/TechnicalDebtQuadrant.html

慎重 & 意図的な負債

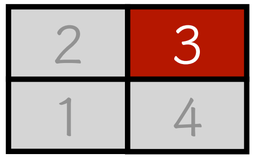


“The prudent debt example is deliberate because the team knows they are taking on a debt, and thus puts some thought as to whether the payoff for an earlier release is greater than the costs of paying it off.”

慎重な負債の例は、負債を負っていることをチームが認識していることと、早期のリリースによる見返り（payoff）が、それにより支払うことになるコストより大きいかどうかを考慮しているという点で、意図的だ。

Design Payoff Lineを
過小評価しない

慎重 & 意図的な負債



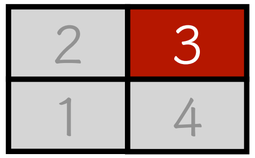
- 例1： 優れた構造の追求
  シンプルでエレガントなコードを書くための**実験**の過程で生まれる一時的な負債

- 例2： 優れた振る舞いの追求
  新しい機能をユーザーが気に入るか**フィードバック**を得ることを優先した短期間の負債

実験や試行錯誤の
過程で負債が生まれる

※参考ページ
Good and Bad Technical Debt (and how TDD helps) | Crisp's Blog
https://blog.crisp.se/2013/10/11/henrikkniberg/good-and-bad-technical-debt

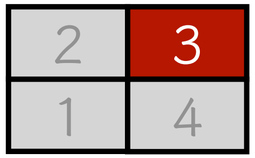
慎重 & 意図的な負債



実験や試行錯誤

負債が生じる原因：

- 優れた振る舞いや構造を導き出す過程で**一時的に**負債が生じる

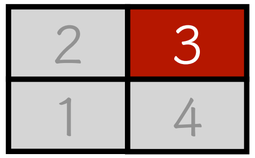


実験や試行錯誤

負債に対する行動

- 優れた構造にたどり着く過程で生じる負債は、一連のプロセスの中で解消されていく
- 優れた振る舞いにたどり着く過程で生じた負債は、次の機能に取り掛かるまでに返済される

慎重 & 意図的な負債



実験や試行錯誤

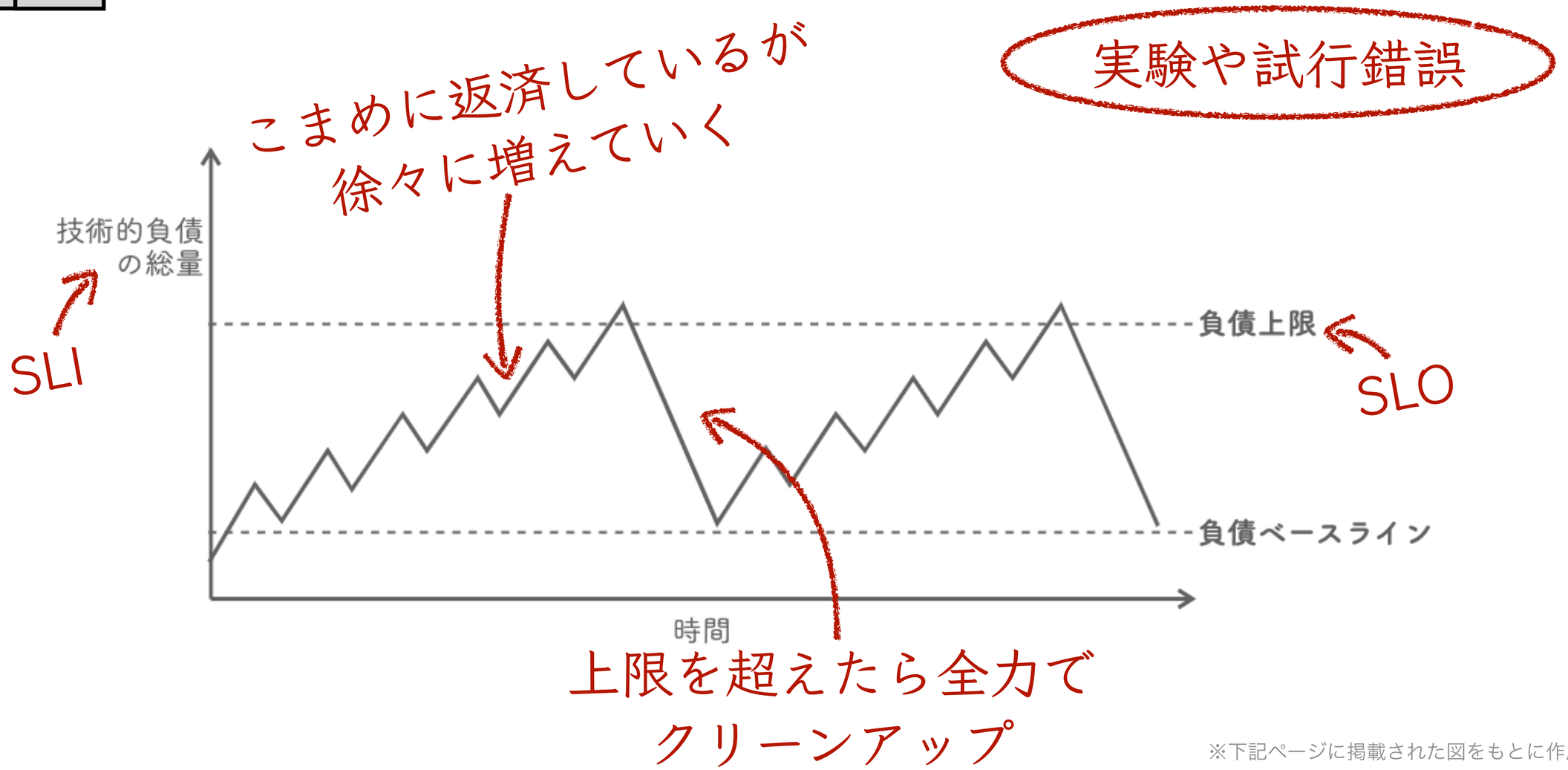
それでも負債は**徐々に蓄積されていく**

まるで経年劣化のように
(=悪い負債)

慎重 & 意図的な負債

2	3
1	4

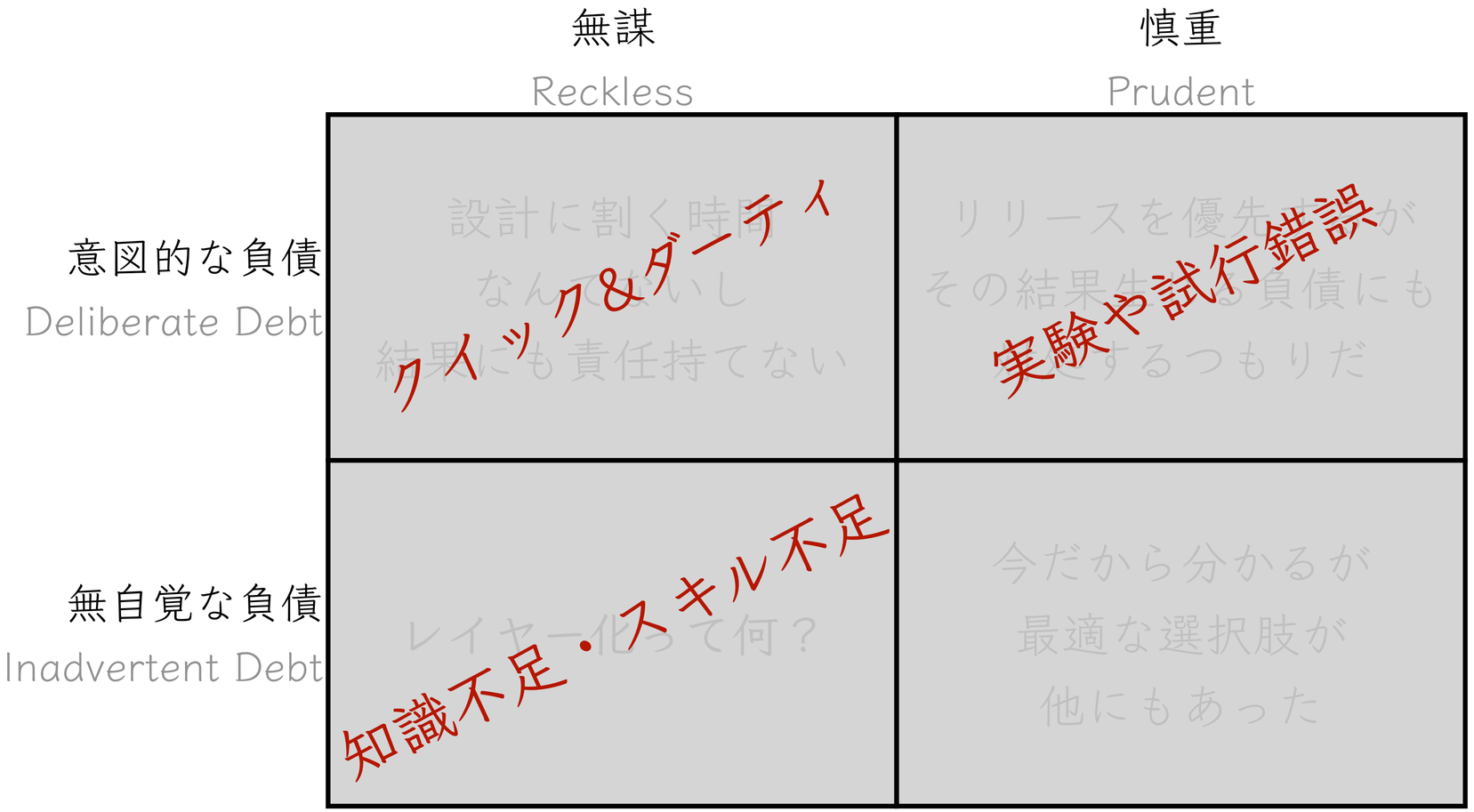
実験や試行錯誤



※下記ページに掲載された図をもとに作成
Good and Bad Technical Debt (and how TDD helps) | Crisp's Blog
https://blog.crisp.se/2013/10/11/henrikkniberg/good-and-bad-technical-debt

慎重 & 意図的な負債は「実験や試行錯誤」の中で生じる

2	3
1	4

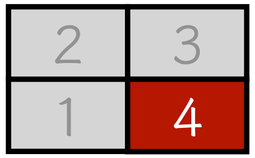


※下記ページに掲載された図をもとに作成

TechnicalDebtQuadrant | martinfowler.com

https://martinfowler.com/bliki/TechnicalDebtQuadrant.html

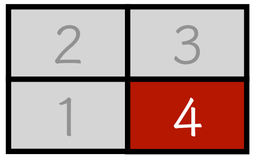
慎重 & 無自覚な負債



	無謀 Reckless	慎重 Prudent
意図的な負債 Deliberate Debt	クイック&ダーティ	実験や試行錯誤
無自覚な負債 Inadvertent Debt	知識不足・スキル不足	(4) 今だから分かるが 最適な選択肢が 他にもあった

※下記ページに掲載された図をもとに作成
TechnicalDebtQuadrant | martinfowler.com
https://martinfowler.com/bliki/TechnicalDebtQuadrant.html

慎重 & 無自覚な負債



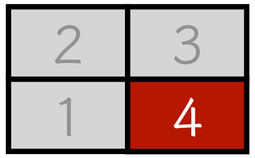
“I was chatting with a colleague recently about a project he'd just rolled off from. The project that delivered valuable software, the client was happy, and the code was clean. But he wasn't happy with the code. He felt the team had done a good job, but now they realize what the design ought to have been.”

最近、同僚がロールアウトさせたばかりのプロジェクトについて、彼と話した。そのプロジェクトは、価値あるソフトウェアを提供し、クライアントも満足したし、コードもクリーンだった。でも、彼はそのコードに満足していなかった。チームは良い仕事をしたと感じてはいたが、彼らは今になって設計がどうあるべきかを悟ったのだ。

↖ それまでは気付いていなかった

TechnicalDebtQuadrant | martinfowler.com
https://martinfowler.com/bliki/TechnicalDebtQuadrant.html

後から気付いた



知識との大きな乖離

負債が生じる原因：

- 長期間におよぶ継続的な変更の中で、システムに対する開発チームの理解が進んだ結果、チームの知識と現状のアーキテクチャの間に**大きな乖離**が生じた
- その乖離の解消に**リアーキテクティング**が必要となった

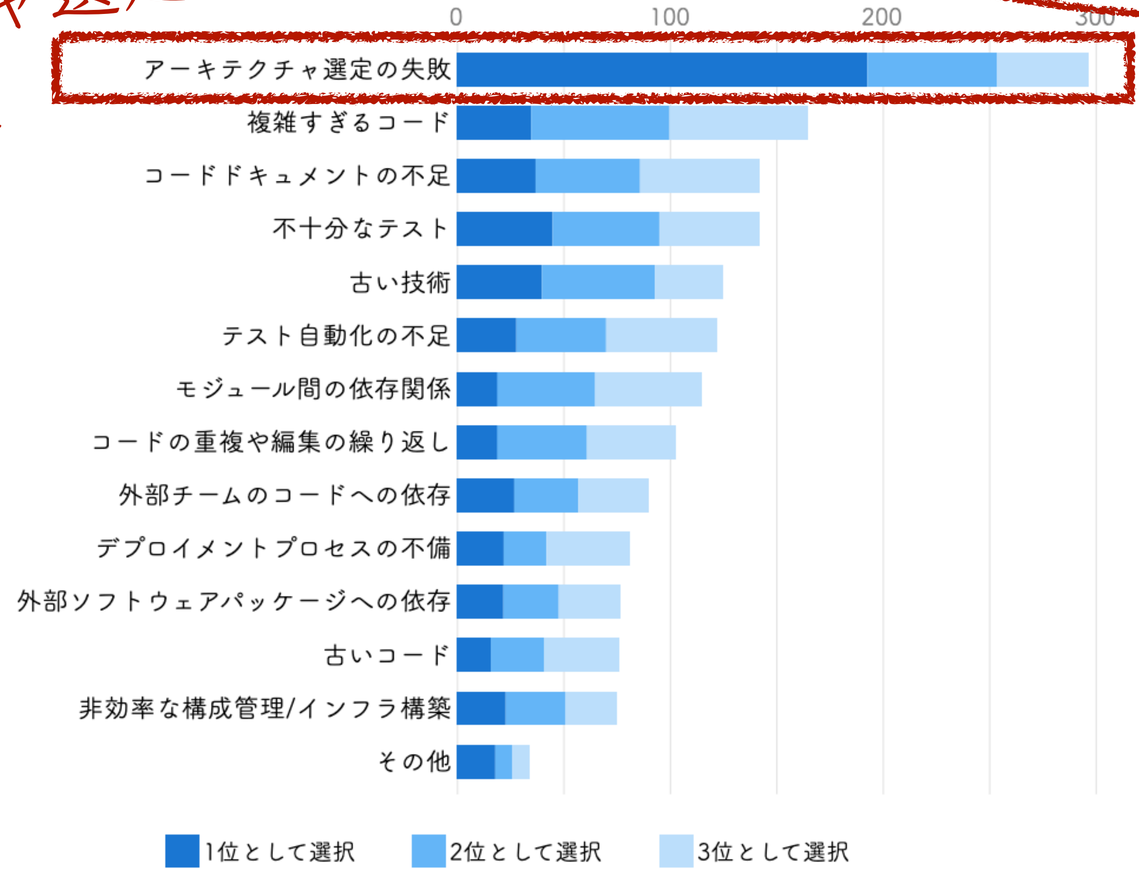
慎重 & 無自覚な負債

2	3
1	4

アーキテクチャ選定の失敗

技術的負債の原因

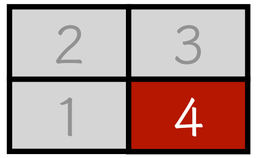
知識との大きな乖離



※下記ページに掲載された図をもとに作成

A Field Study of Technical Debt | Software Engineering Institute

https://insights.sei.cmu.edu/blog/a-field-study-of-technical-debt/

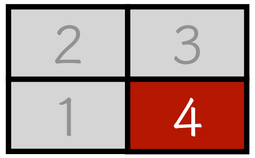


知識との大きな乖離

負債に対する行動

- 「後から変更することが難しい」と言われることも多いアーキテクチャを変えるのは、ビジネス判断としてもコストが大きすぎる
- チームは、ステークホルダーに返済の重要性を説こうと試みる

ビジネス判断が入るのはここ
他は開発チームの問題



知識との大きな乖離

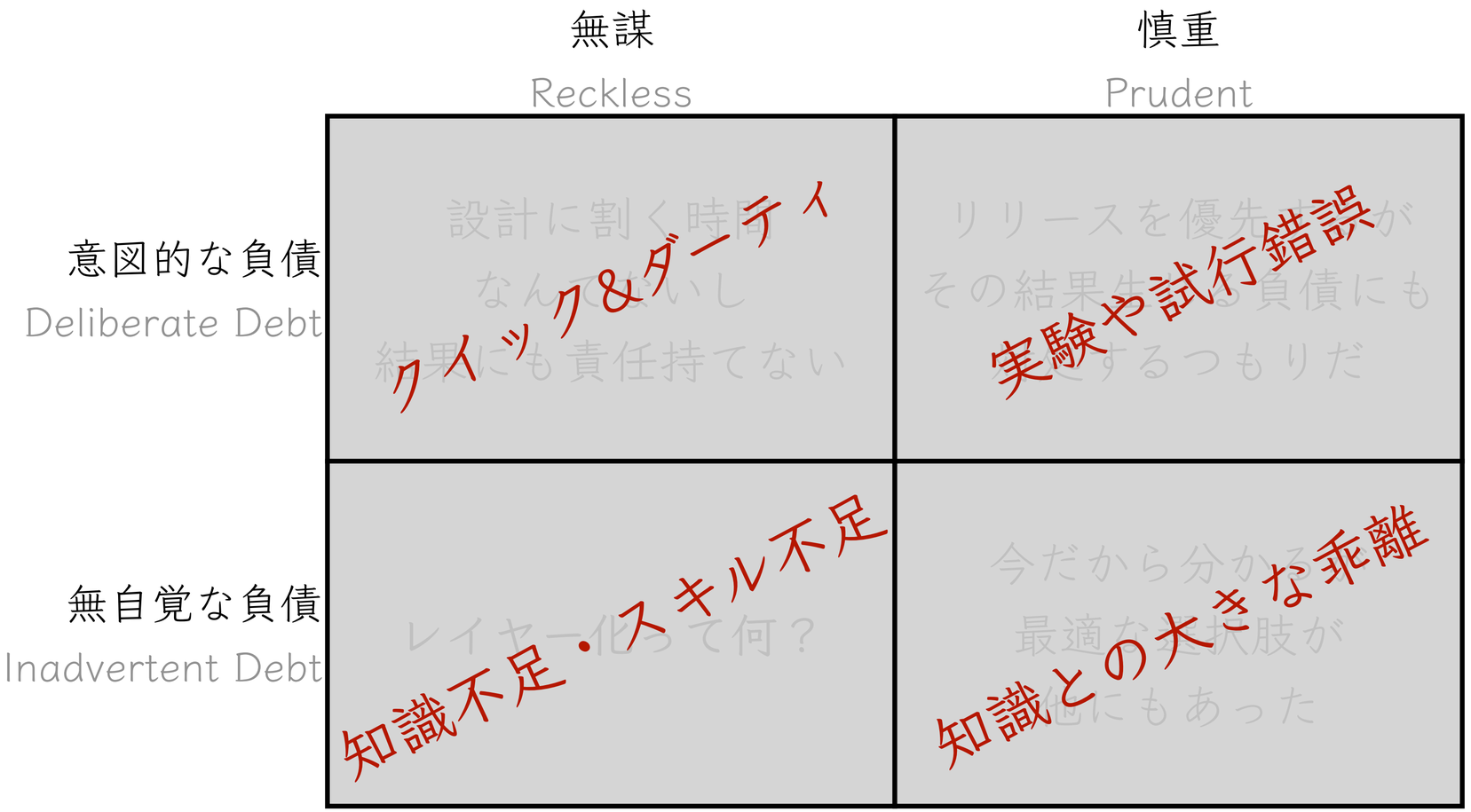
問題の背景（解決すべき点）

- （ステークホルダーに理解が得られないケース）
  日頃から、負債の状況やその影響が可視化・言語化されておらず、開発チームとステークホルダーの間で認識に大きな隔たりがある組織

ステークホルダーに構造が
「見えない」ままにしている

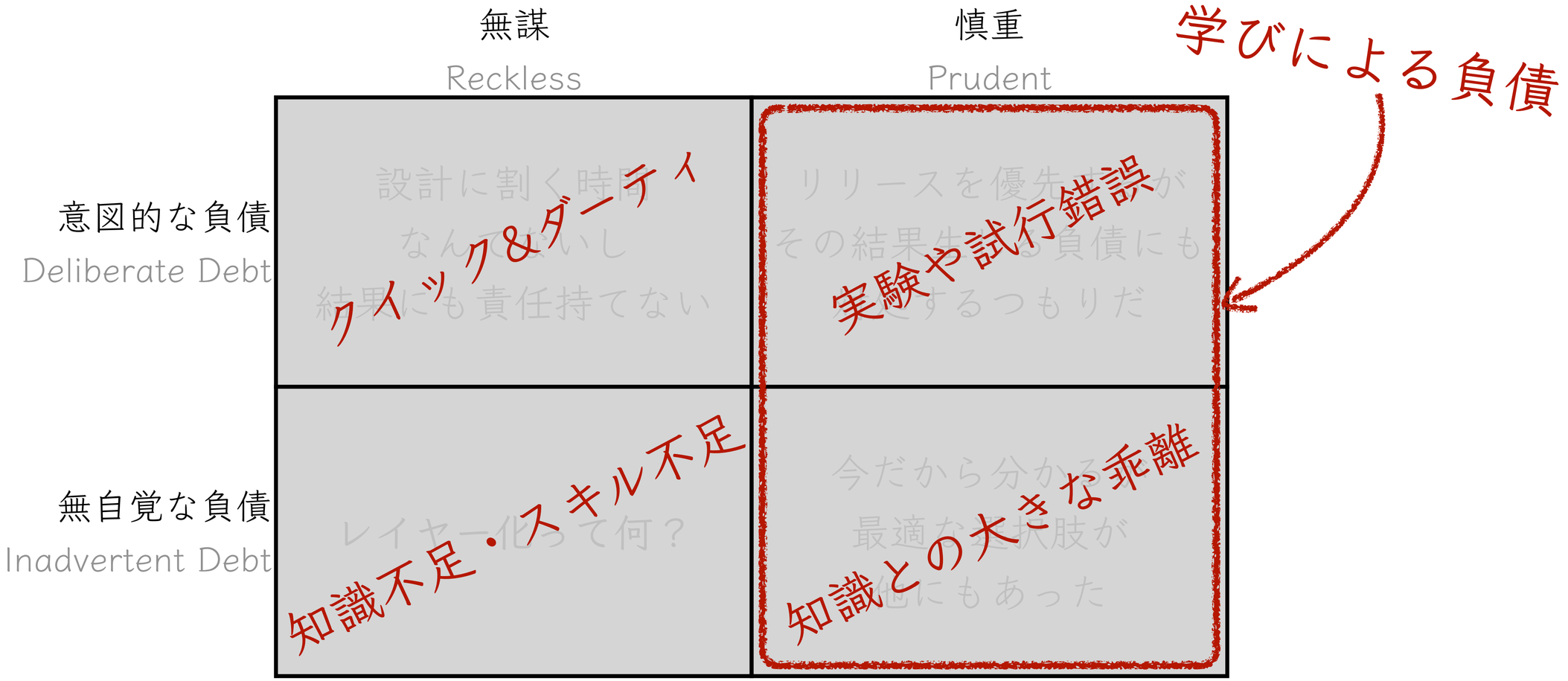
慎重 & 無自覚な負債は「知識との大きな乖離」によって生じる

2	3
1	4



※下記ページに掲載された図をもとに作成
TechnicalDebtQuadrant | martinfowler.com
https://martinfowler.com/bliki/TechnicalDebtQuadrant.html

慎重な負債による見返りは「学び」



※下記ページに掲載された図をもとに作成
TechnicalDebtQuadrant | martinfowler.com
https://martinfowler.com/bliki/TechnicalDebtQuadrant.html

負債が生じる4つの原因

クイック&ダーティ

実験や試行錯誤

知識不足・スキル不足

知識との大きな乖離

解決すべき点のまとめ

知識不足・スキル不足

- 日々の開発業務が忙しすぎて、開発者が知識やスキルを高める機会・環境がない組織
- 開発規模に合わせて大急ぎで人をかき集め、人数だけを揃えたプロジェクト
- 本番稼働による運用・保守からのフィードバックが得られない開発専任チーム

クイック&ダーティ

- 期日もスコープも動かせない、トレードオフスライダーが壊れたプロジェクト
- 技術的負債にまみれた状態が当たり前になって、感覚が麻痺している開発チーム
- 引き継ぎなどで、コードに対するオーナーシップ意識が欠落した開発チーム

知識との大きな乖離

- 日頃から、負債の状況やその影響が可視化・言語化されておらず、開発チームとステークホルダーの間で認識に大きな隔たりがある組織

アジェンダ

- 2つの体験の似た特徴
- 振る舞いの価値、構造の価値
- 無謀な開発による悪循環
- 無謀な負債、慎重な負債
- 2つの体験の決定的な違い

ソフトウェアプロダクトにまつわる2つの「体験」

ユーザー体験

UX: User eXperience

開発者体験

DX: Developer eXperience

よく似た特徴を持っている

似ているだけではなかった

2つの体験には決定的な違いがある

「体験を優れたものにするのは誰か？」

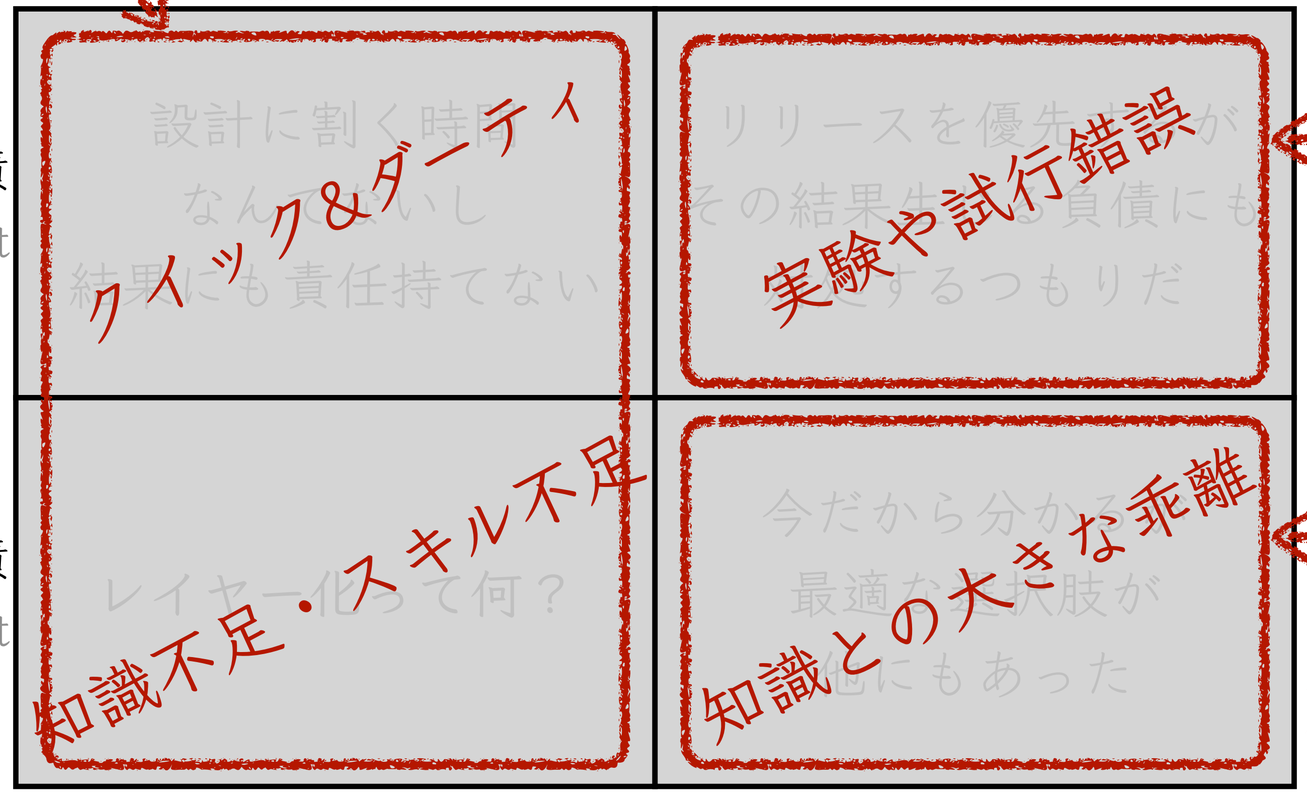
開発者体験を優れたものにするのは開発者自身

体験を優れたものにするのは誰か？

- ユーザー体験：ユーザー自身ではなく**プロダクトの提供者**
- 開発者体験　：多くは**開発者自身**

開発者体験を優れたものにすることに向き合う

無謀な開発の悪循環から抜け出す



常に最善を尽くす

ステークホルダーの理解を得る

※下記ページに掲載された図をもとに作成
TechnicalDebtQuadrant | martinfowler.com
https://martinfowler.com/bliki/TechnicalDebtQuadrant.html

開発者の責任

“アーキテクチャを後回しにすると、システムの開発コストはますます高くなり、システムの一部または全部が変更不能になるだろう。そのことを覚えておいてほしい。そのような状態が許されているようなら、ソフトウェア開発チームが自らが必要とするもののために、懸命に闘わなかったということだ。”

『Clean Architecture』 第2章 2つの価値のお話

懸命に闘おう