Engineer

技術的負債

📻 今選んだ近道が将来の変更に生む追加コスト

どのようなものか

技術的負債は、システムの設計やコードで近道を選んだことによる、将来の追加コストです。今日の時間は節約できても、その近道を解消するまで、後の変更が遅く、危険になったり、不可能になったりします。英語のtechnical debtは、チーム内でtech debtと略すこともあります。負債という言葉は比喩です。近道が借金のように振る舞うため選ばれました。今何かを得る代わりに、それが残る限り、後から費用を請求されます。

この考えは、Ward Cunninghamが1992年のOOPSLAで発表した、Smalltalk製のポートフォリオ管理システムWyCashについての経験報告に由来します。少人数のチームが約四メガバイトのソース全体の概要を把握し、成熟した部分は何度も見直したり、書き直したりしていました。報告は、その整理によって段階的な成長を着実に続けられたと論じています。重大な落とし穴は、その整理をしないこと、つまり動作し、顧客も満足しているのに、整えられたことのないコードです。Cunninghamは「最初に書いたままのコードを出荷することは、借金をするようなものだ」と一行で表しました。報告自体は単に負債と、その類推について述べています。後の研究が二語の名称を使い、Googleの2015年の論文は、この比喩をWard Cunninghamによる1992年のものとしています。

2016年4月にDagstuhlへ集まり、用語の範囲を合意しようとした研究者たちは、作業上の定義を作りました。そこでは負債を、短期的な労力を節約する一方で、将来の変更を高くつかせ、一部を不可能にする状態を残す、設計や実装上の要素の集まりとしています。影響はシステム内部の品質、特に保守や進化のしやすさに限定します。これにより、まだ作っていない機能とは区別されます。負債のコストは、システムを変える人にかかり、その人たちを通してのみ利用者へ届きます。同じ報告は、金融の語彙も使います。元本は、近道で節約したもの、または今、よりよい解決策を作るのにかかる費用です。利息には二種類あります。繰り返し発生する利息は、開発速度の低下、誘発される不具合、品質の損失という形

で支払い、累積する利息は、不十分なコードの上に新たなソフトウェアを構築するたびに増える追加コストです。負債は主にコードにありますが、文書、テスト、既知の不具合にもあります。

概念を判断に生かす

「技術的負債とは？」を実際の判断に生かしましょう。その概念を、開発または購入するシステムに結び付けて考えるお手伝いをします。 まず全体像を設計し、各部分のつながりと費用を、コードを書き始める前に明らかにします。

サービスを見る お問い合わせ →

用語の背後にある前提を検証する

解説とリンク先の実績を参考に、プロジェクトについての質問を整理してください。ツールを選ぶ前に、前提を一緒に検討できます。 14部門を支える共通インフラ計画 — 14 共通インフラ計画の対象となった研究部門

14部門を支える共通インフラ計画 お問い合わせ →

なぜ重要なのか

意図して負債を引き受けることが、よい判断になる場合もあります。Cunninghamの報告は、少量ならチームを速く進められるが、コードを書き直して早めに返すことが条件だと述べています。Googleの論文も、借り入れが適切な戦略になることは多いとしています。両方が警告するのは、負債を残すことです。報告では、不十分なコードのために失う一分一分という時間で利息を払い、誰も整理しない実装は、技術組織全体を止めるほど重くなり得るとします。その前に、プログラムは一人では把握できないものとなり、一部分だけを知るプログラマーが生まれ、最後には方向転換できない製品になります。

問題は、負債が選択の結果とは限らないことです。Dagstuhlの報告は、しばしば意図せずに生じ、開発者が不意を突かれるため、原因となった判断からずっと後の保守作業で見つかることが多いと述べています。原因は日常的です。納期の圧力、重要な人の不在、判断時点で技術的な機能について十分な情報がなかったことなどです。Googleの論文は、隠れた負債こそ、気付かないうちに複利で膨らむ危険なものだと付け加えています。

この比喩は、開発者と管理側に共通の言葉も与えます。Dagstuhlの報告は、強力な意思疎通の仕組みと呼び、参加者が示す将来像には、不具合や新機能と同じ慎重さで負債を管理し、開発者の懸念と管理側の判断を結び付ける方法も含まれています。セミナーで紹介された研究は、そのチームへの効果を示します。Graziela Tonin、Alfredo Goldman、Carolyn Seamanは、合計60人からなる複数の学生チームに、負債を明示的に特定し、追跡するよう求めました。見える一覧によって、顧客とリファクタリングの時間を交渉し、プロジェクト内の未成熟な部分を記録し、コード品質が改善しているか確認できました。メンバーの多く

は、意思疎通が増え、負債が本当に必要かをよく考え、リファクタリングも増えたと答えています。著者たちは、負債を引き受ける判断を話し合った結果、ほかの人が代われない専門家が減ったチームもあったと付け加えています。

実際の活用例

負債は、チームが仕事を計画する場所に書くことで管理できるようになります。この記事のために作った台帳の項目なら、次の程度でも十分です。

```
TD-14        請求書の合計を三か所で計算している
成果物:      billing/web, billing/export, reports/monthly
原因:        公開日が固定され、規則を共通モジュールへ移すより
             コピーするほうが早かった
元本:        テストも含め、規則を一つのモジュールへ移すのに約2日
利息:        税規則が変わるたび三回修正・テストし、直近五回の変更で
             一回、コピーの修正が漏れた
次の対応:    次の税規則変更時に返済する。電子請求書の出力は
             今四半期の予定だが、四つ目のコピーにしてはならない
担当:        請求処理チーム
```

項目は、Dagstuhlの報告のモデルに従っています。具体的な成果物に結び付け、原因、元本、利息を持ちます。利息を書くと、請求コードが乱雑だという感覚が、比較して判断できるコストに変わり、台帳が役立ちます。税規則が年四回変わり、変更ごとに作業とテストが一日余計にかかるなら、元本の二日分は六か月以内に回収できます。予定する出力機能は、Dagstuhlのいう累積する利息です。重複したルールの上に新しいものを作るたび、後で直す費用が増えます。逆に、コードを凍結し、もう誰も触らない見込みなら、同じ項目が、負債を残す判断の根拠になります。

負債はコードだけの問題ではありません。このサイトの社内プロセスの刷新で8,000時間を節約した実績記事は、創業者が以前Mappitallで行った仕事の記録です。技術的負債が読者の作業を遅らせていないかと問い、仕事の進め方に積み重なる負債を説明しています。設計ではなく継ぎ足しで成長したソフトウェア構成、動いてはいるが効率の悪いシステム、人が待つか過負荷になる日程管理です。緊急事態ではなかったため、そのままにされていました。2015年から2022年の仕事で、構成、システム、日程管理を作り直し、約8,000時間を節約しました。記事は、その一部を、チームの運営方法に変更を定着させた効果だとしています。恒久化しない改善は、次第に以前のやり方へ戻ってしまうためです。

問題が起きるところ

何もかもが技術的負債と呼ばれる。 言葉が広まると、古いライブラリーから未実装の機能、同僚の書き方まで、あらゆる不満に同じラベルが付きます。Dagstuhlの報告は、この危険を明示し、明確な定義で対処するよう求めています。将来の変更に費用がかかる近道に限定し、不具合は不具合、要望は機能として記録します。原因と利息を示せない台帳項目は受け付けません。

負債が、人の頭の中にしかない。 書き残さなかった負債は、次にコードを触る人が、なぜ時間がかかるのかも分からず払うことになります。バックログのそばに台帳を置き、システムの一部を変える前に、Toninの研究の学生チームが行ったように確認します。時間配分を決める人に見えるのは、その人が見る場所に書いた項目だけです。

利息がかからない場所の元本を返している。 誰も変更しないモジュールを整理すると、働いた気にはなりますが見返りはなく、毎週変わるコードは請求し続けます。利息は人が作業する場所でだけ増えるので、各部分の変更頻度と、その上にこれから何を作るかで台帳を並べ、負担の大きい項目から返します。

返済を、大規模な書き直しまで待つ。 何年も残した負債には、システム全体の置き換えで対処しがちですが、古いものが費用を請求し続ける間に、新しいプロジェクトも自分の負債を抱えます。Cunninghamは早期の返済を勧め、Googleの論文は、リファクタリング、単体テストの強化、未使用コードの削除、依存関係の削減、APIの絞り込み、文書の改善という一般的な方法を挙げています。どれも機能の追加ではなく、次の変更を安くするためのものです。それを必要とする変更の近くで、通常業務の一部として予定します。

実験が本番に残る。 本番コード内の条件分岐で代案を試すことは、個々の実験としては安く済みます。Googleの論文は、その分岐が積み上がり、全組み合わせのテストが困難または不可能になる様子を説明しています。Knight Capitalも挙げ、古い実験用コードの経路が想定外の動作をしたためとみられる問題で、システムが45分間に4億6,500万ドルを失ったと述べています。実験用の分岐とフラグを定期的に見直し、使わないものは削除し、コードを増やすのと同じだけ減らすことも評価します。

ツールの点数を、そのまま負債と考える。 静的解析は重複行や長い関数を数えられ、ダッシュボードはその数を残高のように見せます。しかしGoogleの論文は、比喩から継続的に追える厳密な指標は得られず、チームがまだ速く進めることも負債が少ない証拠にはならないと明言しています。DagstuhlでForrest Shullは、変更コストの上昇や開発速度の低下などの症状と、プロジェクトをまたいで

一貫した相関を持つ、単一の検出ルール群はなかったと報告しています。ツールの指摘は、自分たちの変更と不具合の履歴に照らし、費用を予測できるルールだけを残します。新メンバーをどれほど速く戦力にできるかなど、論文自体が示す問いのほうが、よい判断材料になります。

参考資料

- The WyCash Portfolio Management System — 負債の比喩が最初に登場する、Ward CunninghamのOOPSLA '92経験報告です。
- ソフトウェア工学における技術的負債の管理 — Dagstuhl Seminar 16162の報告で、作業上の定義、元本、利息、明示的な負債一覧の研究を含みます。
- 機械学習システムに潜む技術的負債 — GoogleのSculleyらによる2015年の論文で、負債の蓄積、返済方法、測定の難しさを論じています。
- Refactoring — Googleの論文が返済方法の最初に挙げる、小さく、振る舞いを変えない変更についてのMartin Fowlerのサイトです。

用語集

実際の制約に合わせて選ぶ

業務の流れ、既存のインターフェース、変更を必要とする利用者をお知らせください。検証可能な納品内容と、お客様のチームが保守を続けられる引き継ぎ内容を定められます。

料金を見る お問い合わせ →

用語を実務に結び付ける

「技術的負債とは？」がお客様の組織にどう当てはまるかをご相談ください。現在のシステムの具体例が一つあれば、話を始められます。

サービスを見る

welcome@engineer.company +45 50 10 61 84 デンマーク、コペンハーゲン

このページを持ち歩く

連絡先カード

心を込めて丁寧に制作： D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

著作権 © 2025 – 現在 · Engineer ApS · すべての権利を留保します。

印刷元： https://engineer.company/ja/glossary/technical-debt/