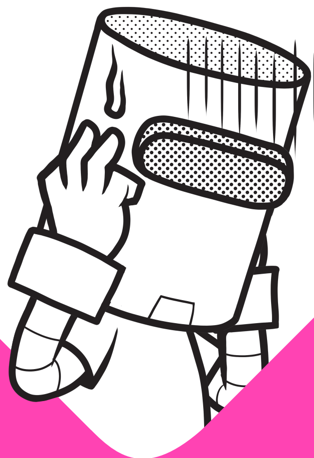
特別抜粋版

出石聡史
DEISHI SATOSHI

42の失敗事例で学ぶチーム開発のうまい進めかた



ソフトウェア開発現場の「失敗」集めてみた。

よくある落とし穴の回避策がわかる!

- 機能盛りすぎ「全部入りソフトウェア」
- 行間を読ませる「文学的仕様書」
- アクションしない「聞くだけ進捗会議」

やらかしたくないエンジニア必読!

SE SHOEISHA

ご案内

本資料は、2024 年 6 月 12 日発売の書籍『ソフトウェア開発現場の「失敗」集めてみた。42の失敗事例で学ぶチーム開発のうまい進めかた』から一部を抜粋し、特別に編集したものです。

全書は『ソフトウェア開発現場の「失敗」集めてみた。42 の失敗事例で学ぶチーム開発のうまい進めかた』でご覧いただけます。購入および、詳しい書籍情報は以下のリンクからどうぞ。

Amazon販売ページ

http://www.amazon.co.jp/dp/4798185183

書籍情報

https://www.shoeisha.co.jp/book/detail/9784798185187

本書を読み始める前に

●本書で扱う架空のプロジェクトについて

本書では失敗を追体験できるよう、架空のプロジェクトをベースにエピソードを紹介しています。また登場人物も明らかにフィクションであるとわかるように、全員ロボットにしていますが、それ自体重要な要素ではありませんので、あまり深く考えずに、読み進めていただければと思います。

◆エピソードの舞台、背景

本書の舞台となる、**株式会社ロボチェック**はロボ部品の生産をサポートする、検査機器のメーカーです。ロボ部品の生産において、計測、診断、調整をサポートし、品質をコントロールするためのハード、ソフトを含めたシステムを提供することが、株式会社ロボチェックの主な業務です。

ロボ部品を生産している企業には、パワータイプを得意とするA社や、耐久性に優れたB社、精度の高いC社などがあります。

◆ロボットアームの自動検査プロジェクト

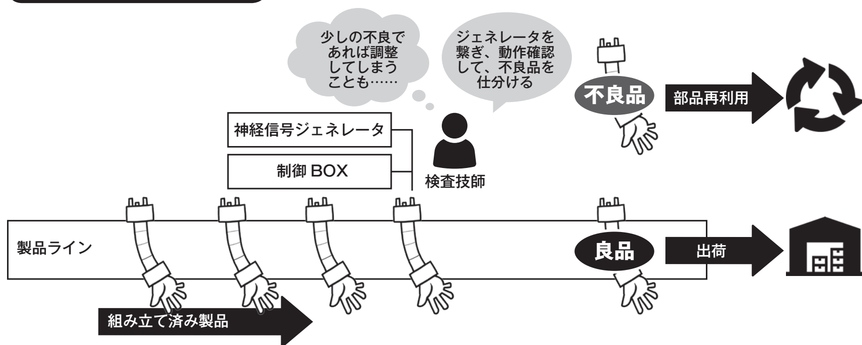
あるとき、C社から工場のライン増設に関する相談がロボチェック社に舞い込んできました。今までベテランの技師が行っていた検査工程を自動化し、かつ生産量を増やせないか、という相談です。

これはロボチェック社にとって大きなビジネスチャンスです。早速この要望に応えるべく「**アームチェッカープロジェクト**」を立ち上げました。もしこのプロジェクトが成功すれば、A社やB社も興味を持ってくれるはずです。

プロジェクトリーダーにはハルさんを抜擢しました。ハルさんはこれまで優れた技術者として成果を上げてきましたが、リーダーは初めての経験です。

そしてここからハルさん、チームメンバーのコーハイさんやシンジンさんたちの長い戦いが始まります。プロジェクトには様々な罠や落とし穴が待ち受けており、開発チームは数々の失敗に直面することになるのです……。

生産ラインでの検査工程



●本書における用語の使い方

本書において、各種用語はそれぞれ下記の定義で使用しています。

要望 顧客の望むシステムへの期待。顧客の言葉そのもの。

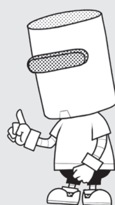
要求 システムに対する、インプットとアウトプットの期待。どのようなインプットに対して、どのようなアウトプットを期待するのか。要望を分析し、顧客の真の課題から効果的な要求を見出す。

要件 システムに対する、インプットとアウトプットの実態。どのようなインプットに対して、どのようなアウトプットが実際に返ってくるのか。システムの機能を外側から見たときの入出力を定義するもの。要求を実際にシステムとしてどう実現するか定義する。

課題 目標と現状との差異を埋めるための取り組み。作業を開始していないタスクも課題にあたる。

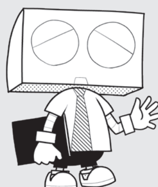
問題 目標と現状との差異そのもの。主に課題を実施したにもかかわらず、なお目標と現状との間に差異がある場合に用いる。

[主な登場人物]（というかロボ）



ハル

この本の主人公。高度な技術力と知識を持ち、技術者として事業に貢献してきたが、このたび新しいプロジェクトのリーダーに任命される。果たして無事、プロジェクトを成功に導けるのか！？

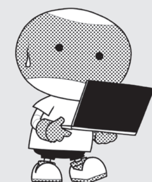


カチョー

ハルさんの上長。管理職。プロジェクトのオーナー。技術者としての実力を買ってハルさんをプロジェクトリーダーに任命する。ほんわかとした雰囲気とは裏腹に、多くの実績を上げている実力者。ハルさんによい経験をさせようと画策している。

ブチョー

開発部門の部長。ソフトウェアのことはあまり詳しくない。どちらかというと昔気質の管理職で、いつも無理難題を吹っかけてくる。部下を大事に考えている一方、組織の長としては厳しくありたいと思っている。

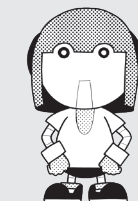


コーハイ

若手技術者。ソフトウェア工学の知識が豊富で、コーディングのスピードも速い。若手随一の技術力を誇るが、調子のよいところがあり、深く考える前に手を動かして、ミスをすることも多い。ハルさんを師匠として尊敬している。

シンジン

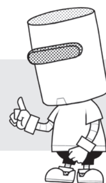
期待の新入社員。プログラミング経験はないが、持ち前の好奇心とコミュニケーション力を発揮して現在急成長中。理解力に優れ学ぶ力が高い。ドーナツが大好き。



ヒンシツ

製品の品質保証担当者。製品の品質には妥協を許さない。製品を世に出す最後の審判者としてのプライドがあり、あるべき姿を正論としてぶつけてくるので、いつも開発メンバーと衝突している。

Contents



はじめに～失敗続けて 25 年～ ………………………… iii

本書を読み始める前に ……………………………………… viii

Chapter 1 「企画」で失敗

Episode 01 なんでもできる「全部入りソフトウェア」……………… 002

Episode 02 みんなの願いをかなえたい「八方美人仕様」 ……… 008

Episode 03 顧客要望通りの「使えないソフトウェア」 ……………014

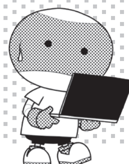
Episode 04 製品のことしか記載のない「足りない成果物」……… 020

Episode 05 ほっとくだけで大問題「新 OS 地獄」 ……………… 026

Episode 06 リーダーも新人も一緒「全員一人前計画」…………… 032

Column 何を、なぜ作るのかが最重要 ……………………… 038

Chapter 2 「仕様」で失敗



Episode 07 実装できない「ふんわり仕様」………………………… 040

Episode 08 解読が必要な「難読仕様書」 ………………………… 046

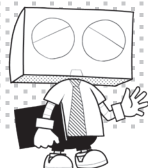
Episode 09 ユーザーを迷わす「オレオレ表記」………………… 052

Episode 10 知らんけど「知ったかぶり技術」 058
Episode 11 カタログだけで判断する「スペック厨導入」 064
Episode 12 行間を読ませる「文学的仕様書」 070
Episode 13 ゴールがあいまいな「おまかせ委託」 076
Episode 14 業界用語でイキる「玄人向け UI」 082
Column 仕様は間違いなく、間違って伝わっている 088

Chapter 3 「設計・実装」で失敗

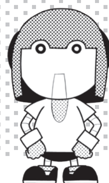
Episode 15 自在に解釈可能な「形だけインターフェース」 090
Episode 16 自分だけの都合で変更する「自己中改造」 096
Episode 17 リリース版が復元できない「不完全リポジトリ」 102
Episode 18 もう開発環境がない「伝説のオーパーツ」 108
Episode 19 夜な夜な数える byte 数「メモリ怪談」 114
Episode 20 つい自分でやってしまう「経験値泥棒」 120
Episode 21 修正が新たなバグを生む「バグ無間地獄」 126
Episode 22 今がすべて「動けばいいじゃん症候群」 132
Episode 23 チームを守れない「ノンポリリーダー」 138
Column 設計書には思いをこめて 144

Chapter 4 「進捗管理」で失敗



Episode 24 アクションしない「聞くだけ進捗会議」……146
Episode 25 残業も計画に織り込む「時間泥棒」……152
Episode 26 会議が会議を呼ぶ「増殖する会議」……158
Episode 27 また責められる「怖い会議」……164
Episode 28 職場が戦場になる「不機嫌なチーム」……170
Episode 29 メールが業務の起点「メールドリブンワークスタイル」…176
Episode 30 変更されない「完璧な計画書」……182
Episode 31 施策を打ち続ける「カイゼンマニア」……188
Episode 32 世の中どうあれ「初期企画至上主義」……194
Episode 33 リリース直前に発覚「ステルス課題」……200
Column 会議5分前のエンターテイナー……206

Chapter 5 「品質評価」で失敗



Episode 34 バグが出ない「開発者バイアス」……208
Episode 35 作ってみてのお楽しみ「出たとこ性能」……214
Episode 36 すべてのバグを許さない「ゼロバグ出荷」……220

Episode 37 こっそり直す「ステルス修正」…… 226
Episode 38 リリース日が来たので「とにかく出荷」…… 232
Column リリース時の品質レベルを定義する…… 238

Chapter 6 「リリース後」に失敗

Episode 39 出荷したので解散「出しっぱなしプロジェクト」…… 240
Episode 40 正しい動作かわからない「ライセンスの迷宮」…… 246
Episode 41 問題は出ないはず「ノーログ戦法」…… 252
Episode 42 犯人を追い込む「お呼びでない名探偵」…… 258
Column 市場問題は誰が対応するのか？ …… 264

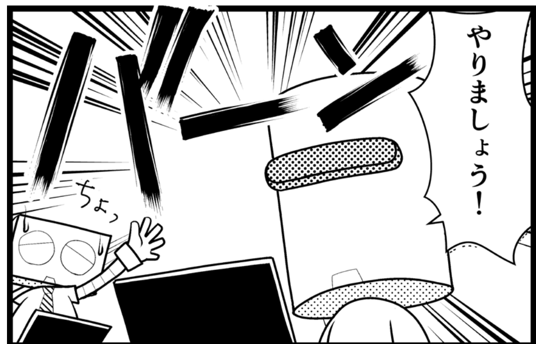
アームチェッカープロジェクト憲章 …… 265
株式会社ロボチェック施設探訪 …… 268
おわりに～それでもソフトウェア開発が好き～ …… 270

Episode

01

なんでもできる「全部入りソフトウェア」

機能がてんこ盛りで実装が間に合わない



デスマーチ。ソフトウェア開発に携わったことがある人なら、必ず聞いたことのある呪いの言葉です。「自分はそうはさせないぞ！」と思いつつも、気がつけばその渦中に巻き込まれています。実はこのデスマーチ、望まずして最初から計画に織り込まれている場合があるのです。いつの間にか開発期間に見合わない、機能がてんこ盛りになったソフトウェア開発を任されているとしたら、すでに黄色信号かもしれません。

すべての要求に応えてしまう

株式会社ロボチェック社は、このたび新しい製品企画「**アームチェッカー**」のプロジェクトを立ち上げました。ハルさんが新任プロジェクトリーダーとして、最初に取り掛かるのが**企画提案**です。現時点では「ロボアーム製造における検査の自動化」という、ざっくりとしたイメージしかないため、どのような製品にして、予算はいくらで、いつ発売するのか、具体的な企画をまとめて経営層の承認を得なければなりません。早速マーケティングや事業に関わるメンバーを招集し、詳細を詰めていくことにしました。

必要なのは検査の自動化なんだけど、実際に重要顧客のラインを見たら、作業者が検査時にアームを調整して不良品も良品にしているのよ。だから自動調整もできなきゃ今より生産性が下がるかもしれないんだ。

ええ！ 自動調整も必要なんですね。それは技術課題だなあ。でも以前は「関節の滑らかさが見たい」と言っていませんでしたっけ？

それは別の顧客のことかな。それで、重要なのが納期！ 実はその重要顧客が今度ラインを拡張するらしくてね。稼働予定の2027年に間に合えば結構な数が出そう。逆にこれを逃すと売り上げは非常に厳しくなる。

じゃあ、2026年12月ぐらいにリリース必須と……。

あと担当者は検査結果を手入力していたから、データをサーバに直接保存できるようにしてほしいな。自動化しても手作業があったらムダじゃん？

それはそうでしょうねえ……（あれ、工場内でネット使えるのかな？）

どの要求も重要で、外せないように感じます。さらに、他の販売メンバーからも顧客要望を聞くと、検査した結果はすぐに印刷したいとか、測定デバイスの通信仕様を公開してほしいとか、アーム検査のための神経信号ジェネレータや制御BOXは○○社と△△社が多く使われているけど、B社は□□社のものを導入しているようだから対応よろしくとか、とにかく言いたい放題、多種多様な要望が出てきます。詳しく話を聞けば聞くほど、それぞれの会社で状況は異なり、各々の要望にも確かな理由がありそうです。仕方がありません。とりあえず受け取った要求のすべてを実現できそうな

システムイメージを作り、これまた絶対といわれた納期に間に合うように、開発日程を引いて企画書に記載しました。

とりあえずリリース日に間に合うように日程計画立てたけど、大丈夫かな。まあ2年もあれば何とかなるよね。

ところがこれが大間違い。機能実現のために見積もりをはるかに超える工数が必要となり、想定外の技術課題も発生。仕様不明な機能の設計に多くの時間を要し、非機能要件も簡単には達成できず、アーキテクチャ崩れにより想定以上のバグが頻出。結局、機能削減の上に3か月もの遅れとなる、プロジェクト最初の大きな失敗となるのでした。

ソフトウェア企画の難しさ

そもそも仕様はどうやって決まるのでしょうか。顧客の依頼、委託によって、顧客の要望を満たすように作る案件であれば、納得のいくまで顧客と仕様を詰めることになります。

一方、ビジネスのために、自社のブランドを冠して広く一般に売り出すソフトウェアは、いったいどのような仕様にすればいいのでしょうか。この場合、先ほどのような「顧客」という絶対的な正解がありません。

そして、正解がないために、つい「あれも欲しいね！」「これもあったらいいよね？」と仕様が膨らんでいき、当初想定していた規模をはるかに上回る巨大なソフトウェアになっていきます。

あいまいな顧客像

いや、でもちょっと待ってください。確か最初のコンセプトはとてもシンプルだったはずです。「ロボアーム製造における検査の自動化」だけです。とすると、実際に検査をしているベテランの作業者に話を聞いて作ればいいので、前述の「顧客の要望に

沿ってそのまま作る案件」のように見えます。
　では、どこの作業者に話を聞けばいいのでしょう？ 例えばA社の作業者は「親指の力が出るかどうか」に注目していますが、B社は「関節の滑らかな動きを検査できるかどうか」が重要と考えています。C社に至っては「動きの悪い箇所を自分で手直しできるかどうか」が重要なようです。おや、それって検査＋調整工程になっていませんか？
　ソフトウェア開発はビジネスです。作ったものを、様々な企業に導入してもらわなければ売り上げが立ちません。しかし、A社の要望だけをかなえる企画ではA社にしか売れません。親指の力を検査できても、動作の調整ができないとC社には導入してもらえないのです。

とりあえずいろんなケースに対応できるよう、全部対応しておくか。いやでもすべてのケースは想定しきれないぞ？

ステークホルダーからの圧力

　また自社のビジネス用ソフトウェアとなると、販売方面からもたくさん要望が出てきます。実際に製品を売っているのは販売のメンバーですから、彼らにとって売りやすいスペックが必要です。想定顧客の声を聴くだけではなく、販売に関わるメンバーの意見もビジネスには重要なのです。
　しかし、販売やマーケティングの面々は、とにかくいろいろな機能を欲しがる傾向があり、あれもこれもとスペックを積んできます。競合と差別化できる機能が多いほど、彼らのセールストークもやりやすくなるので当然です。こうしてあいまいな顧客像、販売やマーケティングからの強い要望によって、気がつくと想定外の巨大ソフトウェアが構想されることになります。

だが発売時期はすでに決まっている

　そして、もう1つのポイントは、この巨大なソフトウェアをいつリリースするのかということです。これには「投資対効果」の話が絡んできます。ソフトウェアのリリース日はマーケティングの観点から売り上げに繋がるように設定されます。
　例えば、C社の新型ロボアームの生産が再来年度末からスタートするとしましょう。

とすると、アームチェッカーはその前にリリースしていないとC社には1台も売れません。普通に考えれば、自動検査装置に投資するタイミングは新製品のラインが立ち上がる前になるためです。

日程が最重要なのはわかるけど、こんなに多機能でリッチなソフトを本当にこの期間で作れるのかなぁ……。

そしてデスマーチは計画される

顧客やステークホルダーの要求をかなえるリッチなスペック。ビジネス上最高のタイミングでのリリース。実際の開発期間や難易度を勘案せず、これらの条件を最優先にしてしまうと、デスマーチを呼び寄せてしまいます。

内心「いやいやこれ無理でしょ」と思っていても、ステークホルダーの強い声には逆らえず、「何とか工夫してやりきるしかない！」という雰囲気がどこからともなく流れます。チームみんなで知恵を出して、足りない工数やら予算やら開発方針やらのやりくりをし、ギリギリ「ハッピーケースなら何とかなるかもしれない」という**楽観的**

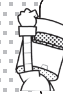
図 各要望と現実の開発日程が合わない

期待のもと、この危険なスケジュールにコミットしてしまうのです。そうです。ここが失敗のポイントです。

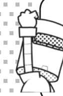
　このように、過度なスペックの開発を請け負わされそうになったとき、リーダーとしてはカッコつけたりいい顔をしたりすることは不要です。**しっかり実現性を精査し、可能な日程を提示する**のです。現時点で明確にはできない先々の日程はリスクとしてステークホルダーに理解してもらいましょう。彼らとの交渉でできることは3つ。**機能を削るか、日程を見直すか、リソース（人材）を増強するか**です。しかし、人材の増強は常に効果的な対策とは言えません。無計画に人員を増やすことは、コミュニケーションのパスが増加し、作業量が増えるだけでなく、新たな人材を教育するためのコストもかかります。まずは当初のコンセプトに立ち戻り、顧客の価値にフォーカスして、もっとシンプルなソフトウェアにできないか、仕様を見直すのがよいでしょう。ただし、日程に関する極端な水増しは技術者としての信頼を落としますのでご注意を。

まとめ

失敗：機能や日程の要求を満たすことが可能か精査をせず、スケジュールの遅延を招いた



回避策❶：開発メンバーも企画に参画し、早期に要求の実現可能性を提示する



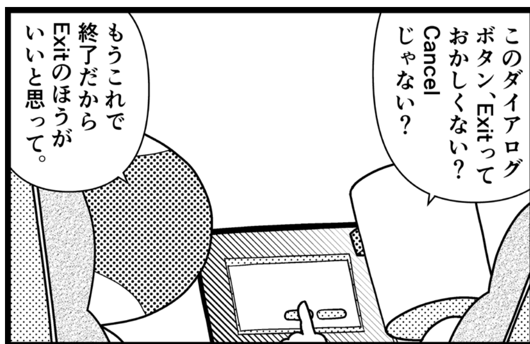
回避策❷：実現可能性が低い場合には、機能か日程の見直しを行う

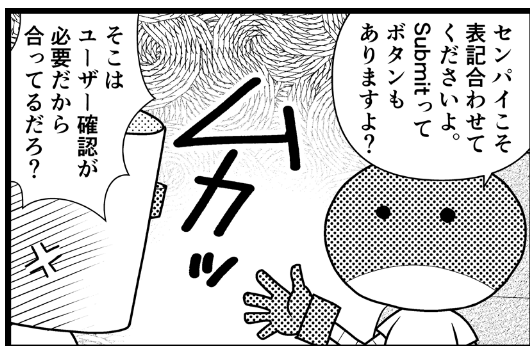
Episode

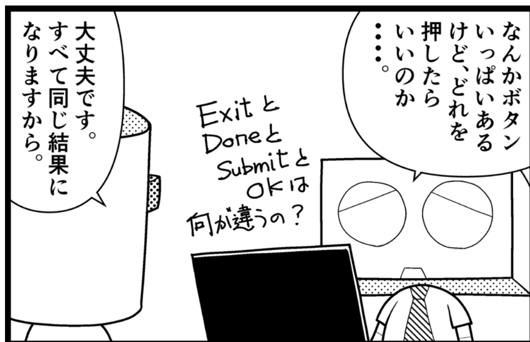
09

ユーザーを迷わす「オレオレ表記」

自分のルールで表記を決める







複数のメンバーで開発しているとどうしても表記の揺らぎが出てきます。UI上のボタンの表記1つを取っても、ExitなのかCancelなのかDoneなのかOKなのか。こうしたソフトウェアに関する用語の揺らぎは、ユーザーを惑わし、操作の手を止めさせ、業務ストレスの基となる「不満の種」です。そのダイアログのボタンは、果たしてどれを押せばユーザーの望む結果が得られるのでしょう？

表記の揺らぎが重篤な不具合に繋がる

アームチェッカーの開発もようやくα版までこぎつけたときの話です。プロジェクトは品質保証の部門に開発の早い段階から第三者評価をお願いすることにしました。もちろんこの時点では重要仕様に関わる機能しか実装できていませんが、気がついたところは遠慮なくリストアップしてもらうようにお願いしています。

ねえねえ、このボタン押して大丈夫？ 警告アイコンついているダイアログなんだけど、OKボタンしかないよ。何か怖いんだけど……。

押してダメなボタンなんてないっスよ。OKボタン押すと閉じるだけっスよ？

でも警告が出ているから、OKじゃないわよ。こっちのエラーダイアログにはExitってボタンになっているし。これ押したらアプリ終わっちゃうの？

いやいや、終わらないっス。Exitボタンでダイアログが閉じるだけっス。

表記が異なるのに何でどっちもダイアログ閉じるのよ！ 閉じるならCloseじゃないの？ しかも警告とかエラー出ているじゃない。ほんとに閉じていいの？

いやそこはハケンさんとシンジンさんとで手分けしたところで……（もうこのぐらいどっちでもいいんじゃない？ わかるっしょ）

コーハイさん、そんな小さいところどうでもいいだろうと、このときの指摘を放置していました。ところがこの後、最終リリース前というタイミングで、この文言が重篤な不具合の指摘を受けてしまいました。

表記の揺らぎで、ユーザーは操作ミスを起こしやすく、重要なデータ損失に繋がる可能性もあり、リリースは承認できないと言われたよ。全面的に用語のチェックだ。これは思ったより厳しいぞ……。

何しろ表記は日本語だけではありません。各国の用語に翻訳し、それぞれが正しいかどうか、改めてチェックする必要もあります。また用語が変わるのであれば取説類も作り直しです。取説に記載したスクリーンショットも全部取り直しになります。と

ても1日2日でできる作業ではありませんし、印刷した取説は廃棄することになりますから金額的にも大きな損失です。ああ、最初から用語の使い方を決めていたら……。

表記の揺らぎはユーザーを惑わせる

複数の技術者が作業をしていると、どうしても表記の揺らぎが出てきますよね。意味が通じれば少々揺らいでも……と思う気持ちもわかりますが、特にUI上での表記の揺らぎは、使用者を困惑させ、操作ミスを誘発することで重篤な不具合を招き、業務に大きな問題を招くこともあります。

現にダイアログのボタンに関しては、押すといったい何が始まるのか全くわからないものがあります。少し例を挙げてみましょう。次の図は筆者が実際に出会って困惑したダイアログを再現したものです。

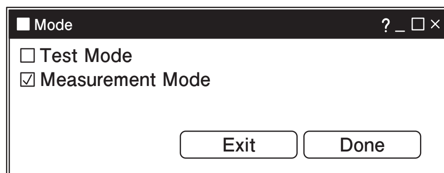


図 Exit と Done

まずExitとDoneがあるものです。どちらのボタンを押してもダイアログが閉じる気がするのですが、どういう状態でダイアログが閉じるのか、いまひとつ確信が持てません。おそらくExitのほうはダイアログで設定した状態をなかったことにして閉じる（今チェックしたMeasurement ModeはOFFになる）。Doneのほうは設定した状態を反映して閉じる（今チェックしたMeasurement ModeはONになる）ということだろうと予想されます。しかし、仮に「いやこのソフトでは逆ですよ」といわれても「明確にそれはおかしい！」と反論するほどでもないでしょう。つまりこの表記では、ユーザーはどちらのボタンを押すと、どのような結果になるのか明確に判断ができません。

また「OK」という表記もよく見ます。前述の例でDoneの代わりにOKとボタンに書

かれていた場合、クリックすると何が起こるのでしょうか。おそらく、「このダイアログの状態でOKです。チェックされた項目を反映してね」ということだと思うのですが、単に現在の設定を見せたかっただけで、クリックしてもチェック内容は反映されないかもしれません。

ユーザーに間違いが少なくなるよう単語を選ぶなら、例えばダイアログを開いたときの状態に戻して閉じるのはCancel、ダイアログで設定した変更を反映したいならSubmit、が望ましいでしょう。

次の例は「次」です。なぜか次へ（Next）が左にあります。こうしたUIの位置の違いも、ある意味表記の揺らぎの1つです。

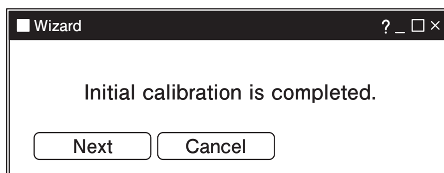


図 左にあるNext

PCの世界では右に行くほど未来、左が過去という習わしになっていますので、次の画面に遷移したい場合、ユーザーは無意識に右側にあるボタンをクリックしてしまいます。この例ではCancelボタンが右にあるので、まだ被害は少ないかもしれません。しかし、例えばDeleteのようなボタンが右にあったらどうでしょう。それこそ大事なデータを損失してしまう、重篤な不具合に繋がります。

また、意外に多いのが何もボタンがない例です。表記がないのに表記の揺らぎとは？と思うかもしれませんが、他では表記があるのにとある場面だけなぜか表記がない、というのも表記の揺らぎの1つです。

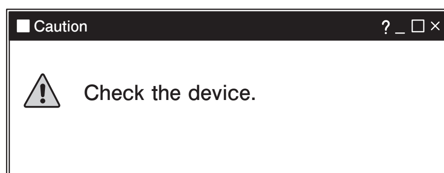


図 ボタンがない

おそらく、右上の×を押して閉じるのか、ダイアログ外をクリックしたら閉じるのだろうと推測できますが、この例のようにCautionマークがついていると、ちょっと戸惑いますよね。この後、どのように操作したらよいか困惑してしまいます。この場合は、もう少し次のアクションがわかる文章を記載するとか、ワーニングに対するヘルプボタンをつけるとか、せめてダイアログを閉じるCloseボタンぐらいつけてあげてもよいかもしれません。

こんな実装は普通しないよね……ここまで全部指示しないといけないのかなあ。でも常識っていうのは人によって違うし、確かに指示がなければ、できるだけ楽な方法で実装するのもわかるしなあ。

ちょっとしたストレスがソフトの評価を大きく下げる

これらの例はとっても小さい話ではあるのですが、こういった細かいところがきちんとできているかどうかでソフトウェアの使いやすさは左右されます。人間ほんのちょっとしたことでも自分の思うようにならないと、ついイラっとしてしまうもの。それはソフトウェアのユーザーも同様です。

今回の失敗ポイントは用語とその使い方を規定しなかったことです。用語を実装者任せにしたために、表記に揺らぎが出て、その結果ユーザーを迷わすソフトになったのです。

ソフトウェア全体を通して用語が統一されている、誤解を生まないようなボタンの表記になっている、常にCancelボタンがあって操作前の状態に戻れるなど、操作に迷いがないように些細なことがしっかりと考えられたソフトウェアは、安心して使うことができます。

用語集を整備する

まず、用語集を整備することが肝心です。このソフトを通じて使う用語はこれと決めましょう。そしてその画面で何をユーザーに促すのか、よく考えて用語を用いましょう。この用語集、実はそのドメインを表す知識となります。一般的な用語でもこのドメインではこう使う、また特殊な用語であってもほかの言葉では言い表せられないため、あえて使う。そういった知識が積み重なった重要なデータベースなのです。

用語集はソフトを作るときだけではなく、ドキュメント類を作ったり翻訳するとき

にも役立ちますし、チーム全員のドメイン知識を底上げするための教育にも使えます。用語集は社内Wikiを使って、誰でも閲覧編集できるようにするといいでしょう。

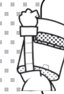
ユーザビリティを点検する

また実装する前にプロトタイプ（手書きのUIモックみたいなものでOKです）を作り、デザイナーの方や、品質保証の方と一緒にユーザビリティの点検を行うことが効果的です。ドメイン知識が少なく、一般のユーザー視点で評価していただける方は特に重要です。作る側の「当たり前」が実は当たり前ではなく、ただユーザーに戸惑いを与えるような、わけのわからないUIになっているかもしれません。ぜひ早めに協力いただくようにしましょう。

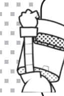
こうしてユーザビリティに関して検討した結果を、UI仕様としてまとめておけば、使用性に関する問題が減ると共に、再利用によってソフトウェア間の統一性も取ることができます。

まとめ

失敗：用語を統一せず実装者に任せたため、使いにくいソフトになった



回避策❶：用語集を作る



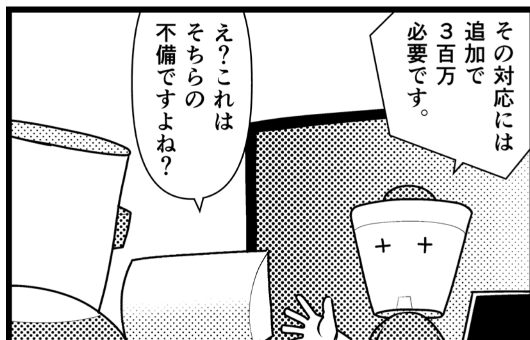
回避策❷：デザイン、品質保証の部署とユーザビリティの点検を行い、UI仕様としてまとめる

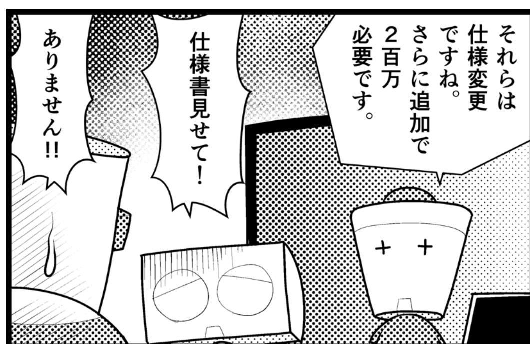
Episode

12

行間を読ませる「文学的仕様書」

仕様書にすべてを記載することはできない





仕様も固まり、利用するプラットフォームやコンポーネントも決まり、作るものがより明確になってきました。後は決めたことをしっかり仕様書にしたため、設計に入る……というところなのですが、仕様をすべて文章として書き下すのは非常に大変です。どうしても当たり前の部分は記載を省くことになりますが、人や文化によってこの「当たり前」は異なるため、様々なトラブルを呼び込みます。

業務委託で当たり前は通じない

アームチェッカーのPCアプリケーションは社外のベンダーに業務を委託して進めています。仕様はすべてロボチェック社内で作成し、設計・実装はロボゴーシステムズ社に依頼しています。ちょうど先ほど、外注先のロボゴーシステムズから成果物の納品があり、シンジンさんが内容をチェックしていたのですが……。

これおかしくないですか？ 複数のデータファイルを選んでアプリにドラッグ＆ドロップしても、1つしか開かないです。

むむ、確かに。不具合として指摘しましょう。でも仕様書には……うーん、複数ファイルのドラッグ＆ドロップ動作については記載がないなあ。この不具合を先方のミスと言い切るのも厳しいかなあ。

これはどうですか？ 測定デバイスを接続していないと、データが表示されないのですけど。

それも困るなあ。測定デバイスに繋がっていないPCでもデータは確認したいよ。とはいえこれも仕様書に記載はない……。

あとそれから……。

動作を細かくチェックしていくと、次々と想定と異なるところが見つかってきました。数えてみると想定の倍以上の不具合が発生しています。

不具合件数が倍以上、ということは今後の修正日程も倍以上かかることを示しています。そうなると明らかにリリースには間に合いません。しかもその不具合の半分は仕様書に記載がないか、仕様書の記載があいまいな箇所で、委託先の問題とも言い切れません。つまり仕様が不明確な部分については「ロボチェック社からの仕様変更」扱いとなり、追加費用が発生する可能性があります。

どうしよう。日程遅れも厳しいけど、追加費用なんてないよ。費用はカチョーさんと相談して、何とか確保してもらうとして……、ロボゴーシステムズさんとは今後の進め方を調整だなあ。日程前倒しでお願いとなると先方の人員も追加になるから、さらに費用がかかるのかなあ。

次々と課題が押し寄せます。とにかく日程を守れる作戦をカチョーさんやロボゴーシステムズさんと相談です。いったいどうしてこんなことになったのか。

外注は難しい

ソフトウェアの規模はどんどん大きくなっており、プロパー社員だけで作り上げることは難しくなってきました。もちろんアジャイル手法のように、少人数で小さく作って順次顧客に価値を提供していくやり方はあるのですが、ソフトウェアの性格によっては、それなりに機能がそろわないと販売は難しい、という場合もあります（ゲーム開発など）。そこで、ソフトウェア開発の一部を社外の業者に委託する（外注する）ということになります。今回のアームチェッカーに関しても、PC上のアプリケーションは外注することにしました。

実はソフトウェアの外注には様々な失敗が待ち受けており、1つのエピソードではすべてを語り尽くせません。今回は数多い外注委託失敗の中でも最も遭遇確率の高い「仕様は理解されていない」という失敗をご紹介します。

世界は暗黙知で動いている

ソフトウェア開発はコミュニケーションが重要です。そして外注委託先とのコミュニケーションにおいて最も核となるものが仕様書です。業務委託の内容そのものですから、重要極まりありません。

ただし、ソフトウェアの仕様をすべて仕様書にしたためるというのは現実的に不可能です。何もかも正確に記載しようとすると膨大な時間や労力がかかる上に、その膨大な仕様書を読み込むほうも膨大な時間や労力を要するでしょう。したがってどうしても当たり前の部分は記載を省き、多かれ少なかれ作者の思いを行間から読んでもらう「文学的仕様書」になってしまいます。

「当たり前動作」は当たり前ではない

これが同じ社員同士、同じチームで働く者同士であれば、記載をある程度省いても、

何となく、どのように作ったらいいのかわかるものです。これは暗黙知というもので、一緒に同じ仕事に従事することによって、お互いが共通に持つ明文化されていない知識が生まれます。

例えば、アームチェッカーアプリの仕様書に「エクスプローラーからデータファイルをドラッグ＆ドロップしたら、そのファイルを開くことができる」と書かれていたとしましょう。これがコーハイさんやシンジンさんなど、同じ会社、同じプロジェクトチームのメンバーであれば「複数ファイルをドラッグ＆ドロップしたら、そのドロップされた複数ファイルすべてが開く」と理解できることでしょう。それが共通の体験を持つことで培った「暗黙知」というものです。しかし、社外の委託先の立場に立ってみると、この記載だけでは何が正解かわかりません。

- ドロップされたすべてのファイルが開く
- エラーにする（複数のファイルは扱えない）
- 先頭ファイルだけ開く（何が先頭？）
- どうするか聞いてくる（ダイアログでユーザーに確認）
- 何も起こらない（無言）

どの選択肢を選ぶかは、委託先の企業的文化や価値観における常識次第なのです。特に海外に委託する場合は、国内と全く異なる文化を持っていますから、思わぬところで齟齬が発生します。筆者は、実際に国外の外注先から「先頭ファイルだけ開く」実装で納品された経験があります。

これが今回の失敗です。**仕様書に書いていない「当たり前動作」が当たり前ではない**ということを理解していなかったので、想定外の実装でアプリケーションが納品されてしまいました。しっかりと仕様書に書き込んだつもりでも、自分たちにとってはあまりにも当たり前すぎて、無意識のうちに省いた記載が多々あります。納品されてから「どうしてこうなった？？？」となることも少なくないでしょう。

そりゃあ仕様書に書いてないほうが悪いのだけど、そんなところまで書く時間ないよ。というか、何でわからなかったら聞いてくれないのよー。

仕様は正確に伝わっていないことを理解する

外注委託先と委託元との間では**「仕様は正確に伝わってはいない」**と思うことが大事です。これは別に委託先が悪いわけではありませんし、委託元がサボっているわけでもありません。どうやっても伝わらないことはあるもので、それが当たり前なのです。そう思って業務を進めることで、お互い早めに問題に気がつきます。

仕様書を委託先に渡したらあとはお任せ、では必ず齟齬が発生します。仕様書の行間を伝えることが大事です。仕様書に記載していないことは顔を合わせて仕様の説明を行うことも効果的でしょう。機能の説明だけではなく、なぜその機能が必要なのか説明をすることで、多くの勘違いを防げます。委託先側も不明なところがあったら委託元に素直に聞くことが大事です。そういった**丁寧なコミュニケーションでしか解決できない**ことがたくさんあります。

お互いの理解を確認する

齟齬や誤解は無意識のうちに発生するため、委託先では気がつかないことがあります。面倒でも委託先に簡単な設計書を作ってもらうとか、手書きのモックアップを作ってもらうなど、できるだけ早めに仕様書の理解を確かめられる工夫をしましょう。また設計や実装に入ってからは実装できた部分を早めに確認しましょう。短く期間を区切って、イテレーティブな納品計画を立て、実装に間違いがないか確かめながら進めるとよいでしょう。

これは委託先との間に限った話ではなく、開発と販売との間にも齟齬は発生します。開発中できるだけ早期に、ステークホルダー間で動作の確認することが、これらの齟齬に対するリスクヘッジになります。

異なる会社、異なる組織同士、まだ十分にわかり合えていないかもしれない。だからこれからしっかりお互いを知っていこう……という、友情漫画のような心持ちが、成功の秘訣です。

お互いの理解や信頼は一日にして築くことはできません。小さく業務を回し、成功体験を共有して、少しずつ信頼関係を構築していきましょう。

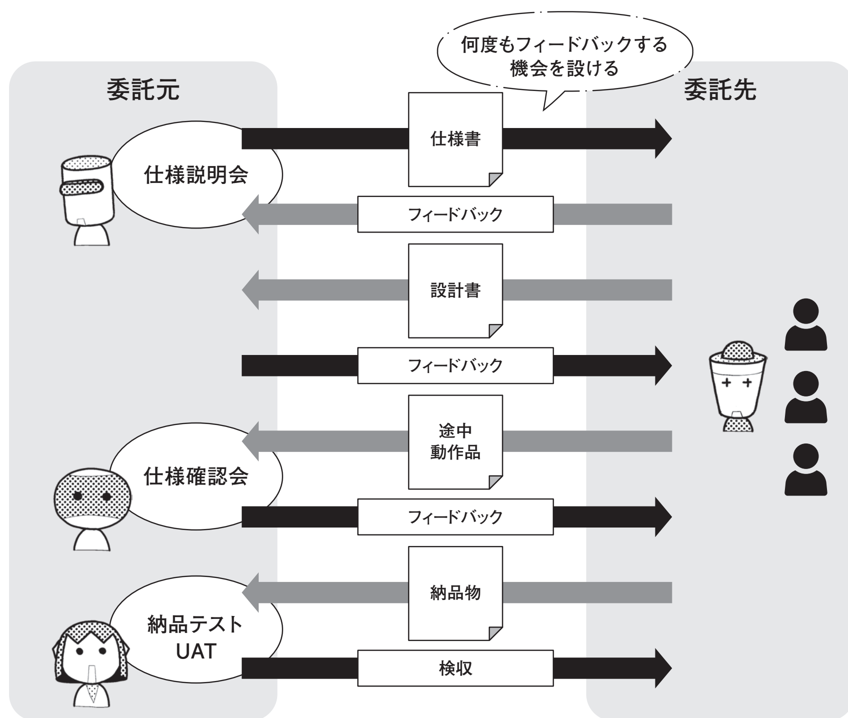
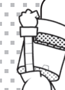


図 丁寧なコミュニケーション

まとめ

失敗：仕様書だけを委託先に渡し、期待と異なるものが納品された



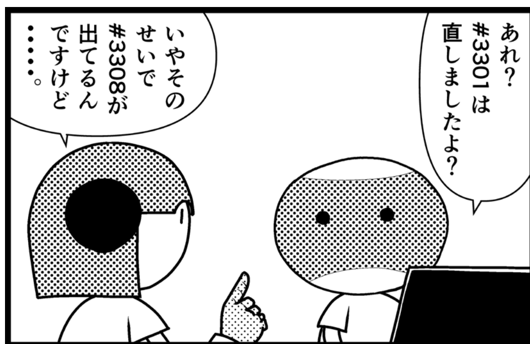
回避策：「仕様は正確に伝わっていない」前提を持ち、コミュニケーションの機会を増やす

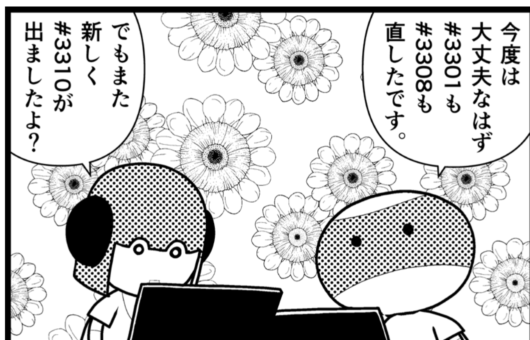
Episode

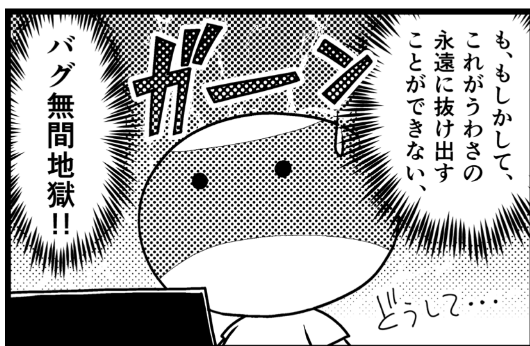
21

修正が新たなバグを生む「バグ無間地獄」

バグの症状だけを見て対症療法する







実装を進めていけば必ずバグが出ます。どんなに優れた技術者でも、誰かの作った部分と処理がかみあわずに、不整合が発生するものです。仕様や約束事がうまく通じていなかったり、どこかでアーキテクチャに沿っていない実装がなされていたりで、なかなかすんなりとは動きません。そんなとき、時間がないからとバグの症状だけを見て対症療法を行うと、後々さらに問題の大きなバグに化けることがあります。

終わらないバグ修正

アームチェッカーもようやく全機能の実装が完了し、品質保証部門での出荷前評価を実施しています。ところが評価途中にもかかわらず、すでに想定の倍以上の不具合が発生しています。進め方を見直したり、評価人員を追加したり、バグ修正の助っ人を頼んだりしているものの、バグは増えていく一方です。コーハイさんも毎日できるだけ多くのバグを修正しようとしているのですが……。

測定デバイスにデータファイルを渡したとき、最後の1行だけが読み込めないバグ、確かに直っているけど、今度は設定データの読み込みで失敗するわよ。

ええ？ おかしいなあ。設定データは触ってないけどなあ。これ以上このバグに付き合っている時間はないけど、うーん仕方ない。

コーハイさん、新しく発生した設定データの読み込みバグも、不具合の発生している箇所を確認し、さっさと修正してしまいました。時間も限られているため、急いで次の大きな不具合に取り掛かっていたのですが……。

ちょっと！ 今までなかった不具合がアプリのほうで出ているわよ。何か変なことしたでしょ。

言いがかりだなあ。アプリなんて触ってないっスよ。どこで出てます？ 設定ファイルの読み込み？ ……あ。

コーハイさん、実は先ほどの設定データの読み込みバグを修正するために、設定データファイルのフォーマット自体を変更してしまいました。この設定データファイルはアプリでも利用するため、フォーマットを変更したことで、アプリで動かなくなってしまったのです。

アプリを新フォーマットに対応させるのも一案ですが、そもそもアプリは業務委託して社外のベンダーに作ってもらっています。それでなくてもアプリも想定以上のバグがあふれていますし、今から委託先に追加費用のかかる変更をお願いするわけにはいきません。それどころかフォーマットを変えることで、アプリに新たな不具合を誘発する可能性もあります。

コーハイさん、しぶしぶデータフォーマットを元の状態に戻し、再度バグ修正を実施することにしました。1つでも余分にバグ修正をこなしたいときなのに3つも4つも手戻りが発生してしまいました。それどころかアプリのほうも追加評価で余計な工数がかかっています。

一番手間のかからない方法で修正したのに、10倍以上の手間がかかることになってしまった……。大失敗っス。

バグ修正がバグを呼ぶ

多くのソフトウェア技術者にとってデバッグは面倒くさくて、やりたくない作業の筆頭です（中にはデバッグが一番楽しいという人もいますが）。最近は開発環境やエディタに、バグを防ぐAIサポートがついており、簡単なミスはほとんど発生しなくなりました。しかし、それゆえ逆に根が深く、いやらしいバグが悪目立ちするようになりました。バグの原因探しは面倒だし、1つのバグだけにこれ以上時間もかけられない……そんな場面では、つい、簡単な方法でバグ修正を済ませてしまいがちです。

これが今回の失敗です。バグに対して**対症療法的な変更を行った結果、新たなバグを呼び寄せ、想定の数倍時間を使ってしまった**のです。新たなバグもまた対症療法的に変更すると、またさらに別のバグが増えるという状況が続き、結果的にリリースを延期せざるを得ない事態になります。最初から面倒くさがらずにちゃんと分析して原因に対処していれば……。

これはいわゆるサイドエフェクト（副作用）というやつです。バグ修正のサイドエフェクトには十分な注意が必要です。こんな副作用が出るならバグを直さないほうがよかった……ということもあります。副作用には次のような事例があります。

1. **ファイルの読み込み不具合のため、ファイルの最後に改行を入れて保存するように変更したら、どこかでクラッシュするようになった（勝手な仕様変更）**
2. **データファイルの一部が仕様と異なっていると思ったのでファイルを修正したら、他の関数から読み込めなくなった（仕様の理解不足）**

3. パラメータの値によって発生する演算バグを修正したら、その演算を利用している他の処理で問題が発生した（バグの再利用）
4. ある処理の不具合を修正したら、正常に動いていたはずの別の処理で不具合が発生した（アーキテクチャ崩れ）
5. AとBという2種類の機器が繋がるアプリにおいて、Aを繋いだときに発生する不具合を直したら、Bが動かなくなった（すり合わせ問題）
6. コードを修正したが、実はバグではなかった（仕様書のバグ、もしくは仕様書がない）

意外と6のケースもよくあります。バグではないのに動作を変更してしまい、その結果バグになるという切ない話です。システムの動作としてどうあるべきか、チーフアーキテクトや関係者間で合意を取っていくしかない場合もあります。

修正の影響範囲がわからない

バグの多くはソフトウェアの仕様や構造が理解されていないために発生します。本質的にバグがどこで発生しているのか、修正のためにはどこを直すべきなのか、直したときの影響範囲はどこまでか、ということを十分把握していないと様々なサイドエフェクトに遭遇します。

特に、ソフトウェアアーキテクチャ自体が悪い場合や、アーキテクチャが実装者に理解をされず崩れている場合などに、サイドエフェクトは多発します。
各機能でロジックが一貫していない、コピペで似たような（ただしちょっとだけ違う）機能が大量にある、グローバル変数などに頼っていて影響の範囲がつかめない、というような状況なら、すでに悪いパターンにはまっています。
中でもコピペは致命的です。何しろ一か所直しても全部は直らない、かといってコピペ部分すべてに同じ修正を適用したら別のバグが出る（コピペごとにちょっとずつ違うので、同じ修正では直らない）という具合で、いつまでたってもバグが収まりません。

ブロック図を描こう

こんな地獄を招かないために、まずアーキテクチャを設計し、図にしておくことが大事です。図はUMLなど厳密なものでなくても構いません。手書きでいいので、とにかく図にしましょう。

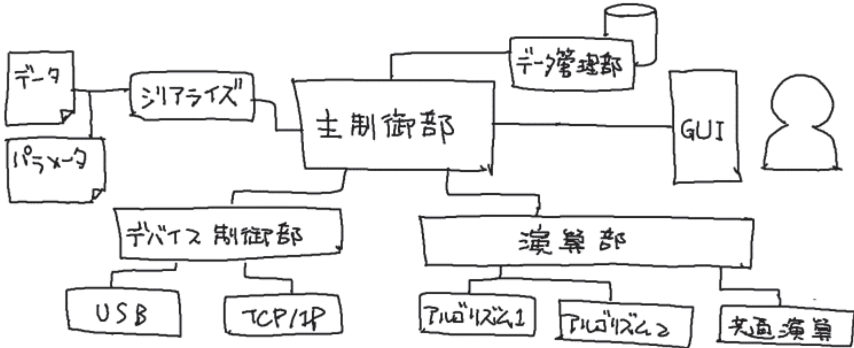


図 手書きの機能ブロック図

図はアームチェッカー用アプリの機能ブロック図です。図のクオリティは置いておいて、言いたいことはわかりますよね。こんなテキトーな図でも、あーアルゴリズムはいくつもプラグインできるようになっているのねとか、共通演算は同じプラグインでまとまっているのでそっちを使うのねとか、デバイス制御は通信方式によらず、同じように見えるのねとか、様々な設計意図を理解できます。もし共通演算部を修正したら、アルゴリズム全体に影響ありそうだなーという想像もできます。機能とその関係性がわかるのが重要です。特定の箇所を修正した場合、どの範囲まで確認しないといけないのかが、ピンとくるようになります。

まずバグの原因を突き止め、その原因に対して修正を行うことが原則です。その修正によってどの機能にまで影響が及ぶのかを図で確認し、問題が出ないかどうかをテストします。

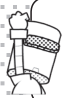
アーキテクチャを共有する

当然、図は描くだけではだめで、共有して利用しないといけません。各機能の実装担当者がそれぞれの考えでコーディングやバグ修正を行うので、放っておくとアーキテクチャは簡単に崩れます。図を描いたら、説明会を開くのが効果的です。全員が同じ図を共有できていれば、バグ修正時のサイドエフェクトも少しはましになるでしょう。バグの発生頻度も下がるかもしれません。

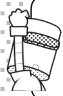
また図も最初から完璧を目指さず、一旦30分ぐらいでクイックに描いて共有し、随時詳細化や細分化を進めていくとよいでしょう。まずは図を描く敷居を下げることが重要です。どうせ最初から完璧を目指しても、プロジェクトを進めるうちに、構造の欠陥が見えたり、変更の必要性が出てきたりするものです。そして図は常に最新版を全員が共有できている状態に保ちましょう。図を改定したのであれば、改めて説明会を実施することも必要です（サーバに置いたから共有できている、と思い込むことは1つの失敗の形です）。

まとめ

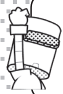
失敗：バグに対して対症療法的な変更を行ったため、新たなバグを呼び寄せ、結果的にリリースの遅延を招いた



回避策❶：アーキテクチャ図、ブロック図を作成し共有する



回避策❷：バグ修正は真の原因に対して行う



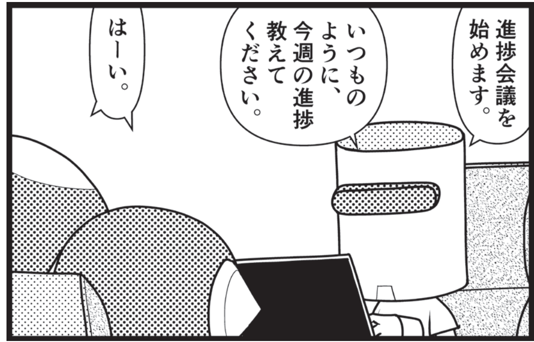
回避策❸：バグ修正の影響範囲を確認し、十分なテストを実施する

Episode

24

アクションしない「聞くだけ進捗会議」

面倒を起こしたくない心が面倒を起こす



これまで設計や実装にまつわる失敗を紹介してきました。上流ステージほど重大な失敗が、それとはわからない顔をして待ち受けています。Chapter4ではそういったヤバさに気づけない、という失敗を取り上げます。毎月毎週毎日進捗をチェックしているのにヤバイ状況に気がつけない。いや気がついているのに何もしない。実はプロジェクトを失敗させる最も効果的な方法は「何もしない」ことなのです。

すべてが順調に見える

また少し時間を戻して、アームチェッカープロジェクトの初期のころです。ハルさんはチームリーダーとしてチームの進捗を把握するため、週に一回進捗報告会を開いています。このころはまだすべてが順調に見えていました。

えーと今のところ予定通りっス。パラメータ調整アルゴリズムの実装が終わったところで、今は評価してもらっているところです。

こちらも予定通り、実装は進んでいます。外注さんのほうも、予定通り来週にはPCアプリのα版を納品しますって。

ほい。今週もおつかれさん。今週はすべて順調っと。

何事もなく進捗を終え、穏やかで平和な日々に感謝するハルさん。しかし、その次の週、重要機能を確認していた品質保証のヒンシツさんから、不具合の指摘が飛んできました。実はアームチェッカープロジェクトでは、重要機能に関して早期に品質保証部門の確認を行い、技術課題を早めに対処する施策を打っていたのです。

ちょっと調整アルゴリズム全然動いていないじゃない。むしろ調整前より悪くなっているわよ。ちゃんと確認したの？ 時々小指だけ計算が終わらないし。あとPCアプリのα版も全然動いてないし。そもそも測定デバイスと繋がらないってどういうこと？

ええ！ いやちゃんと動いたって言って……あーいや動いているとは言っていなかったかな？ うーん。こちらでも状況確認します。

何だか進捗会議で聞いたイメージと違います。順調に進んでいると思ったのに、ちっとも進んでいないどころか、課題だらけです。

ああ、すみません。いや確かに予定通り実装は終わってたんスけど、テスト用データでしか見ていなくて。実際のデータを食べさせると次々演算が発散しているみたいっス。おかしいなあ。アルゴリズムチームと相談します。

実は外注さんに測定デバイス本体を渡せていなくて。先方が通信仕様から作った測定デバイスのモックでしか接続確認していないそうです。

聞けば聞くほど、ちっとも予定通りではなかったことがわかってきました。しかもこれらの課題はもっと前に気づけたはずですし、事前に気づけば何とかなったことばかりです。先週予定通りと報告したところなのに、複数の技術課題の発覚で、現時点でどのくらいの遅れになるのか見当がつかない状況です。急遽それぞれの課題に対し、有識者を集めて対策会議です。

しまったー。もうちょい深く話聞いていれば何とかなったかもしれないのに。ううう。今週の進捗報告会、憂鬱だなあ。

気がつくと地獄の一丁目だった

順調に思われたソフトウェア開発ですが、リリース直前になって急に大変な問題が頻発することがあります。気がつくと実装が間に合っておらず1か月遅れになりますとか、実はパフォーマンスが出ていなくてどうしましょうとか、とある条件下ではアルゴリズムが破綻しますとか、重篤な問題が次から次に押し寄せます。

「毎月毎週毎日のように進捗会議していたのに、自分はどうしてわからなかったのか？」という困惑と後悔が同時に押し寄せてきますが、もう手遅れです。今となってはとにかくできることをやるしかないのですが、そもそもどうしてこんな大変な課題を見逃したのでしょうか。

絶対おかしい！ 毎週の進捗会議では順調なイメージしかなかったのに、1か月以上の遅れが急に出てきた。どこで見逃したのかなあ……。

なぜ気づかないのか

進捗会議の場で課題に気がつかないのは、次のような理由があるからです。

- 課題が挙がってこない
- 課題が挙がっているけど見えていない
- 課題を見つけようとしていない

課題が挙がってこない

課題が挙がってこないのは、タスクの粒度が大きすぎることが1つの原因ですが、組織の中に「言えない雰囲気」が漂っていることも関係しています。できていないなんて言えない、という無言の圧力にメンバーが負けて「はい順調です。今週末にはできると思います」と言って、こっそり一人で頑張ってしまうというケースです。

逆にリーダーがそもそも課題を聞いていない場合もあります。これは進捗会議あるあるなのですが「今週は何しましたか？」と作業しか聞かない場合です。これをすると「今週は予定通り実装しました」という返事が返ってきて終了です。たとえ機能が動作していなくても実装は完了したのですから間違いではありません。作ったけど動かない、という課題を聞き出せていません。

課題が挙がっているけど見えていない

またせっかく課題が挙がってきていても、その課題の実情がよく見えていないケースもあります。例えば「実装したのですが動いていません。でもちょっと見直せばすぐ動くと思います」という見込みでしかない報告がそのパターンです。こういう報告は、それなら2、3時間あれば動くから問題ないよね、と軽く流されてしまい、記録にも残りません。「後でその動いていない状況を見せてもらえる？」とちょっとした変化に気づいて、アクションを起こさないと、とんでもない問題を見逃すことになるかもしれません。

課題を見つけようとしていない

そしてそもそも課題を見つけようとしていないケースもあります。先の「実装したけど動いていない」ケースは、これにも該当します。実装が終わったから作業は完了として、実は動作確認をしていない可能性があります。自分であえて課題を出そうとせず、そっとタスクを閉じているのです。

これはタスク完了の定義があいまい、かつ共有できていないことにも問題があります。明確に「コードを実装しただけでは完了ではありません、少なくとも単体テストを実施し、想定の動作をすることと、他に新たなバグが生じていないことを確認でき

たら完了です」というような約束をすべきです。

これらのケースは誰が悪いというよりも、チーム全体の「問題を起こしたくない」というムードから発生します。誰だって面倒は起こしたくないのです。希望的観測にすがって、つい「何とかなるよね」と考えてしまいますし、課題に蓋をして見えないようにしてしまいます。リーダーもついつい「順調ってことでいいよね」と波風を立てないようにしてしまいます。

ここが失敗のポイントです。進捗会議で**作業だけを確認し、課題の発生を捉えていない**ため、多くの課題が急に沸き起こったかのように見えるのです。

進捗会議の神髄は、できるだけ**早く課題を芽のうちから掘り起こす**ことです。作業の進捗自体はJIRAとかAzure DevOpsとかに任せて、必要なときに確認すればよいのです。プロジェクトにどのような課題が発生しているのか、新たな課題の火の手が上がっていないか、素早く捉えることが肝心です。

課題の発見を喜ぶ

「できるだけ早く課題が見つかる」ことをチームの価値としましょう。メンバーがき

図 早期に課題を見つけアクションに繋げる

ちんと課題を報告してくれたらリーダーは「早く課題がわかってよかった！」と共に喜ぶことが肝心です。

そして、見つけた課題には必ずアクションを設定し、実行に移します。いくら課題を見つけてもアクションしないのなら、課題は解消できませんし、報告するモチベーションにも繋がりません。

担当者から「今週あまり時間が取れていなくて実装が少し遅れています」という報告があったとしても、もしかしたら何らかの課題が潜んでいるのでは？ とアンテナを上げ、状況を見極めた上で、適切なアクションに代える必要があります。

とても面倒ですよね。何だかんだエネルギーが必要です。そうです。何かを変えるには相当なエネルギーが必要ですから、ついついまだ問題ではない、そのうち解決するはずと信じて、何もしないほうを選んでしまいがちになります。これはぜひ覚えておいてほしいのですが、**何もしないことが失敗への一番の近道**です。どんなプロジェクトでも必ず課題をはらんでいます。ですから何もしないでいるということは、すなわち課題を放置していることと同義なのです。

いや一別に何も対策することなんてないですよ？ 順調です、というときが要注意です。きっと何か重要な課題を見逃しています。リーダーを任された以上、もう楽な道はないとあきらめましょう。できるだけ早期に問題を発見し、傷が浅いうちに手を打つことが、結果的に一番楽な方法なのです。

まとめ

失敗：進捗会議で課題を見逃し、大きな遅延を招いて日程が守れなかった

回避策❶：進捗会議では課題を見つけることにフォーカスし、早期発見に価値を置く

回避策❷：課題に対してはアクションを設定する