

5.8 良書との出会い方

インターネットを使えば多くの情報を得られるようになりましたが、書籍の重要性も失われていません。書籍には以下のようなメリットがあります。

- 体系的に情報がまとめられている
- 複数の人が校正や校閲に参加しており信頼性が高い

同じような内容を扱っている技術書の中にも「良書」と呼ばれるものがあります。世間一般で言われる良書は難解なものが多く、私たちにとっての良書になるとは限りません。本節では「私たちそれぞれにとっての良書」と出会うために、筆者が気をつけていることを順番に説明していきます。

1. 入門・中級・上級の違い

技術書には大きく「入門」「中級」「上級」向けにレベルが分かれています。自身の求める情報がどのレベルに該当するのかを事前に決めておく必要があります。「中級」以上のものは、前提となる知識などが割愛されている場合が多く、いきなり取り組むと理解に苦しみ挫折する可能性があります。また、日本国内の技術書は入門レベルが大半を占めています。Amazonなどの通販サイトでは、洋書の技術書を扱っているため、中級以上を探し求める場合は、洋書を検討するのも良いでしょう^{†1}。

2. 「はじめに」と「目次」

「はじめに」は、著者が書籍を執筆した意図や読者に「このように読んで欲しい」というメッセージが記載されています。共感できる場合、楽しく読める可能性が高いでしょう。「目次」は構成や内容を把握することができます。自分が求める情報や興味を持つ部分を容易に確認することができます。

3. 「あとがき」と「版・刷数」

書籍の後ろの「あとがき」には、本を読み通した読者に「今後このように活用して欲しい」という著者の願いや「このような不満があったのではないだろうか?」というメッセージが記載されています。共感できるものは、楽しく読める可能性が高いです。

また、最後のページに記載されている「版・刷」が多いものは、それだけ多くの人に読

まれている証拠です。発売から間もない書籍は初版のため意味がありませんが、初版発売から1年以上経過している本などは参考になります。

4. 本文のパッと見の印象

書籍をペラペラとめくりながら印象をつかみます。本文には、技術的な内容以外にも多くの情報があります。筆者が書籍の印象を決める要因は以下のよう�습니다。

- 色使いに嫌悪感がないか
- 文章の表現に嫌悪感がないか
- 文字のサイズ、間隔、強調などに嫌悪感がないか
- 図解やイラストがわかりやすいか

筆者は「嫌悪感がないか」という視点で見ることが多いです。例えば、筆者はモノクロや寒色系の色が使われた書籍を好み、暖色系が使われた書籍は苦手です^{†2}。これらは、常に目に入り、読むことが苦痛になることがあるため避けるようにします。

5. 正誤表

書籍では修正履歴として、インターネット上に正誤表が公開されています。購入しようしている「版・刷」で修正されているか確認しておきましょう。また、正誤内容が非常に多い場合は、著者・編集者ともに正しく取り組んでいるのかや正誤表に記載のないページの正確性に懐疑心が芽生えることがあります。特にプログラミング言語のような模写が必要な場合は、誤った記載により悩む時間が増えるため、正誤内容が多い場合は避けるようにしています。

6. ダウンロード配布物

技術書には、説明に用いられているサンプルコードや設定ファイルなどをダウンロード配布している場合があります。コピー＆ペーストで手っ取り早く動作確認したい場合などは重宝するため、ダウンロード配布物の確認をしています。

良書は、私たちを助けてくれる強い味方です。「私たちそれぞれにとっての良書」と巡り会いましょう！

†1 えらそうに洋書と書いていますが、筆者は英語を読むことも喋ることもできません。しかし、過去には仕事で辞書を用いながら洋書を読む必要に迫られたこともあります。英語が苦手な場合も、選択肢として持っておく方が良いでしょう。

†2 これは好みによるものです。また、暖色系が使われていても、その他の内容に嫌悪感などがなければ購入することもあります。

5.9 手取り額10%

私たちは、技術的なものからマネジメントに及ぶまで、仕事に必要となる知識をつけるために学習を行います。仕事に直結するような研修費用・資格受験費用・書籍代などは、組織が負担してくれますが、自身の興味のある分野など自発的に行う学習は、自分自身から学習費を捻出しなければなりません。

自主的な学習にかかる費用の一例を挙げてみましょう。

- PC・周辺機器の費用（ディスプレイ・キーボード・モニタなど）
- 検証機器の費用（サーバーやネットワーク機器など）
- クラウド利用料、ライセンス費用
- 書籍代、資格受験費用
- 勉強会、交流会（ITエンジニア飲み会など）の参加費

筆者は、ITに限らず何かに触れる機会は、将来の成長に向けた投資と考えています^{†1}。しかし、無制限に投資ができるわけではありません。生活費や将来への貯金も確保しなければなりません。このように考え出すと、学習のために「何かを試してみよう！」と思つても、どの程度のお金を使っていいのか悩み、なかなか行動に移せないことがあります。

そこで筆者が考えだしたのが「手取り額10%ルール」です^{†2}。

手取り額10%ルール

手取り額10%ルールは「毎月の手取りの10%は、悩むことなく学習費として使って良い」というルールです。このルールは以下の4つの約束から成り立っています。

1. 月の手取りが20万円であれば2万円を使える（手取りの10%）
2. 余った分は繰越金として貯め続ける
3. 賞与などの大幅増額のときはPCやディスプレイなど高額購入のチャンス！
4. どうしても足りない時は一時的に増額してもよい

この10%という割合に特別な意味はありません。5%でも20%でも構いません。当時の筆者の収入から生活費や交際費など毎月必要となる金額を差し引いた後、学習費として使え

る余剰資金が10%程度だった結果です。大切なのは、楽しい生活を維持するためのお金を差し引いた後の余剰金が、どの程度あるかを把握することです。

学習費を決めるメリット

手取り10%ルールのメリットは、「悩む頻度の削減」と「機会損失の削減」です。

筆者は、読書が好きで、大型書店でさまざまな書籍を物色することも大好きです。インターネットとは異なり、ちょっと目についた書籍を手に取ったり、平置きされている人気書籍を発見したり楽しい時間を過ごせます。しかし「ちょっと欲しい本だけれど、価格が高い」や「欲しい本が多すぎるのではないか」といった悩みがよくありました。学習費があらかじめ決まっていれば、価格が多少高い場合や冊数が多いと感じた場合でも即断して購入でき、悩む頻度は減少しました。また、後で考えて買おうとして結局買わなかつたということも減り、良書と出会える機会も増えたと感じます。期待ほどの内容ではなかった場合でも「学習費の範囲内であるから」と深く傷つくことも少なくなりました。

本との出会いは一期一会と表現されることがあります、他の人間関係などすべての出会いも一期一会です。せっかく「何かを試してみよう！」と思ったとき、手持ちのお金で諦めてしまうよりは、少々損をする可能性があっても、やってみる方がトータルで見れば良い経験となることが多いと感じます。

なお、決めた学習費は上限値ではありません。10%は「悩まず使っていい金額」です。当月分と繰越分を使い切っても、さらに必要であれば、貯金残高等を考慮し増額します。しかし、割合を毎月無闇に変更してはルールの意味がありません。最初は手探りに変更することがありますが、一度割合を決めたら当面はそのままにしましょう。変更を検討するのは、仕事内容（学習すべき内容）が変わったり転職などにより収入が大きく変動したときです。

最後に、既婚者の人たちが手取り額を基準にするのは難しいかもしれません。その際は、お小遣いを基準にしてみることをおすすめします^{†3}。

†1 ITの学習だけに限らず、もう少し広い解釈をしています。「同僚との飲み会」や「行ったことのない場所へ行ってみる」なども投資対象です。仕事に直接関連はしませんが、これらの経験のいくつかは、仕事のあり方を考えたり、人間関係の形成に役立つことがあります。

†2 このルールは会社員時代の筆者が考えたものです。現在は自営業という毎月が不安定な生活のため、残念ながらこのルールは使用できなくなりました。現在では昨年度の所得金額から一定の割合を学習費としています。

†3 筆者は結婚した後、仕事の学習に必要な費用の話をしてお小遣いを増額してもらったことがあります。

5.10 検索エンジンへの聞き方

検索エンジンは、インターネット上の情報（Webサイトや関連する画像など）を収集し、私たちがWebブラウザの検索ボックスに入力したキーワードに基づいて検索結果を表示してくれる重要なツールです。私たちが何かわからないことを人に聞くとき、聞き方を間違えると期待した回答を得られないように、検索エンジンも聞き方により回答（検索結果）は異なります。本節では、筆者がGoogle検索エンジンを使って、技術的なことを調べるときに期待した回答を得るためにしている4つの聞き方を紹介します^{†1}。

検索演算子を使って聞く

Googleの検索エンジンでは、検索演算子を使うことにより検索結果の精度を高めることができます。筆者がよく使う検索演算子まとめました（表5-10-1）。

検索方法 ^{†2}	書き方（例）	説明
検索から語句を除外	word1 -word2	除外する語句の前に「-」をつける
完全一致を検索	“word1”	単語や語句を二重引用符で囲む
OR検索	word1 OR word2	各検索クエリの間に「OR」を置く
AND検索	word1 word2	各検索クエリの間にスペースを置く

表5-10-1 筆者がよく使う検索演算子

英語で聞く

技術情報を探す際、日本語圏よりも英語圏の方が情報が多いです。英語圏の検索結果を出す方法はいくつかありますが、一番簡単な方法は以下のURLからGoogle検索を行う方法です。このURLをお気に入りに追加しておけば、簡単に英語圏の検索を行うことができます（表5-10-2）。検索に指定するキーワードは英語で入力してください。

<https://www.google.com/?gl=us&hl=en&pws=0>

項目	値	意味
gl	us	国にus(米国)を指定している
hl	en	言語にen(英語)を指定している
pws	0	パーソナライズド検索を0(無効)にしている

表5-10-2 米国圏を検索するURLパラメーター

時間に聞く

Googleのデフォルトの検索結果では時間に関する指定をしていません。「直近の新しい情報だけ表示したい」「過去の一定期間の情報だけ表示したい」場合は、ツールの「期間を指定」のメニューから行えます（画像5-10-1の①と②）。

画像に聞く

プロトコル構造やソフトウェアコンポーネントの関係などは、図解して解説されているものを見る方が理解が早まります。その場合は「画像」をクリックすることで、図形やグラフで説明しているサイトを見ることができます（画像5-10-1の③）。



画像5-10-1 期間指定・画像で検索

今まで見つけられなかつたものがあれば、これらの聞き方を試してみましょう。これまでよりも、欲しかった多くの回答を得られるはずです。

†1 本節の内容以外にもGoogleの検索方法はさまざまあります。興味がある方は以下を参照してください。

Google検索ヘルプ。Google検索の結果を絞り込む。2023年9月9日。

<https://support.google.com/websearch/answer/2466433>

Google検索ヘルプ。Googleで検索オプションを使用する。2023年9月9日。

<https://support.google.com/websearch/answer/35890>

†2 「検索方法」の名称は筆者がわかりやすく書いたもので公式の呼称ではありません。Chromeブラウザの検索オプションでは以下のように表記されています。

- ・検索から語句を除外：含めいないワード
- ・完全一致を検索：語順も含め完全一致
- ・OR検索：いずれかのキーワードを含む
- ・AND検索：すべてのキーワードを含む

第

VI

章

無能のための時間管理

本章は、無能なITエンジニアとして生きていくための時間管理（タイムマネジメント）についての教訓になります。

プロジェクトでは、納期やタスク単位で期限が設けられており、私たちが詳細に管理する必要はないでしょう。

しかし、私たちのタスクを期限内に達成するためには、個人レベルでの時間管理が必要です。

1 時間管理術ができない理由

2 予定は日で考える

3 目的と期限の明確化

4 ズレを許容する準備

5 実績と私のスケジュール

6 空き時間なのか、待ち時間なのか

7 私の集中できる時間

8 無駄な時間？私には必要です

9 ヒトは1週間、モノは1ヶ月

10 残業は悪か

6.1 時間管理術ができない理由

時間管理（タイムマネジメント）は「仕事を効率化し、無駄な作業を減らし残業もなくす素晴らしい考え方だ！」と多くのビジネス書やインターネットの記事で語られています。確かに、それらの記事には素晴らしい考えが含まれますが、多くの場合ビジネス全般へ向けた内容が多く、表現が抽象的で私たちが活用するには具体例が不足しています。ITエンジニア向けの時間管理術を説明した書籍として「エンジニアのための時間管理術^{†1}」がありますが、以下の理由から私たちが活用するのは難しいと感じています^{†2}。

- ターゲット層がシステム管理者である
- 日本の請負構造的な働き方は考慮されていない
- 著者のトーマス氏が有能であるがために実践できているのではないか・・・

筆者が、時間管理術ができないと考える理由は他にもあります。私たちの働き方は時間管理に向かない要素が多いのです。以下に、その例を挙げておきましょう。

ルーチンがほぼない

時間管理術には、ルーチンと呼ばれる定型的な作業時間を効率的に短縮する方法論があります。しかし、私たちの仕事には、ルーチンに該当する仕事が、ほとんどありません^{†3}。メールの確認などルーチンとなりうる一部のタスクはありますが、これらを短縮して時間を有効活用するのは難しいでしょう。

不確定要素の多さ

時間管理術の基本的な考えは、個々のタスクごとに作業時間を見積もり、計画的に実施することです。それこそ分単位の見積もりが必要になります。しかし、私たちの仕事には多くの不確定要素があります。設計や実装において問題が発生することはよくありますし、新しい技術を取り扱う場合には予想外の遅延が生じて数日単位で狂いがでてくることもあります。また、複数プロジェクトを同時に担当している場合、これらの不確定要素はさらに大きくなり、タスクの正確な時間見積もりは非常に困難です。

障害連絡・問い合わせ連絡がスケジュールを壊す

複雑な障害連絡や即答できない問い合わせ内容は、対応や調査に半日や1日を費やすこ

とも少なくありません。朝一に1日のスケジュールを計画しても、このような突発的な連絡があれば、一瞬にして計画が狂ってしまいます。

やる気のぶれの激しさ

筆者特有の理由かもしれませんのが、やる気というものは毎日一定ではありません。やる気のある日は多くのタスクをこなすことができますが、やる気のない日はパフォーマンスが著しく低下します。私たちは人間であるため、コンピュータのように毎日同じパフォーマンスを発揮することは難しいでしょう。

筆者は、プロジェクトの変更や仕事の内容が変わる環境で、1日のスケジュールの計画を考えることが困難であり、一般的な時間管理術を実行し続けることが難しいと気づきました。しかし、時間管理術を試行する中で「日単位の管理」と「時間単位の管理」という2つの方針を導き出すことができました。

日単位の管理を行う

1つの方針は、自分のみで「完結する」タスクに関しては「日単位で管理」を行うことです。1日の中で他者に影響せず、期限までに結果が用意できれば良いタスクについては、日単位でスケジュールを立てます。自分で完結できるため、かなり緩い管理方法です。多少のスケジュールの狂いは残業時間などを使って対処します。

時間単位で管理を行う

2つの方針は、自分のみで「完結しない」タスクに関しては「時間単位で管理」を行うことです。顧客先のリリース作業や会議など、1日の中で影響を受ける他者が存在するタスクについては、分や時間を単位として管理します。他者に影響を与えるため、厳密に管理します。また他者に影響するため、正確な時間配分が必要です。

この2つの方針は、一般的な時間管理術が強調するような効率化は実現できませんが、スケジュールで悩む時間を減少させ、他者へ影響を与えないようにするのに役立ちます。

†1 トマス・リモンチェリ、エンジニアのための時間管理術、株式会社オライリー・ジャパン、2006年10月

†2 「エンジニアのための時間管理術」の内容を批判しているわけではありません。本書の中でも、影響を受けていることが多いです。少々古い書籍のため登場する単語やツールが古くなっているものの、お時間のある方はご一読をお勧めします。

†3 1日の多くをルーチン作業に使っている読者がいたのであれば申し訳ありません・・・。

6.2 予定は日で考える

私たちの仕事には「タスクAは○月×日まで」のように期限が設けられています。リリース作業や会議など関係者が多いタスクは、15分や30分単位でタイムスケジュールが組まれることもありますが、多くのタスクは日付による期限が設定されています。

自分のタスクを計画をする際には、タイムスケジュールが必須のタスク以外は、1日～数日単位で考えるのが良いでしょう。「今日の○時～×時まで」のように短期的なスケジュールは避けるべきです。このような短期的なスケジュールは、時間が足りない際に後続のスケジュールを組み直す必要があるからです。

来週中にやろうなどの長期的なスケジュールも避けるべきです。長期的なスケジュールは、プロジェクト全体の管理には役立ちますが、個人が処理するレベルのタスクに対してはスケジュールの機能を果たしません。

以下は、筆者が1日のスケジュールで実行している手順です。スケジュール全体はプロジェクトで管理されており、タスクが割り当てられていることを前提としています。

1. ルーチン作業を行う

仕事を始める際に、ルーチン作業や短時間で終わるような、所要時間が明確なタスクを終わらせます。これは、チャットやメールの確認・返信、バックアップなどのステータスチェックなどです。

2. 期限の近いものに着手する

最初に取り組むタスクは期限の近いものです。後回しにしても期限は伸びません。簡単なものだからと後回しにしてはいけません。簡単なものであれば、なおさら早期に着手して完了させるべきです。同じ日が期限のタスクが複数ある場合は、他者が待つ可能性の高いものから取り掛かります。

3. 次に期限の近いものに着手する

次に取り組むタスクは、次に期限が近いタスクです。同じサイクルで期限の近いタスクから順次取り掛かります。苦手なタスクや面倒なタスクを後回しにしてはいけません。

4. 未知数のタスクに着手する（1日の終わりまでに必ず着手する）

1日の終わりまでに未知数のタスクに取り掛かる時間を確保します。筆者は、1日の終わりに着手することが多いです。未知数のタスクには以下のようなものがあります。

- 期限や優先度が不明確なタスク
- 初めてのものなど時間が推定できないタスク^{†1}

これらのタスクは、担当は決まっているが期限が定まっていない場合や、未経験のものや調査が必要な場合などが該当するでしょう。この時間は、タスクの難易度を把握するための時間です。今後どの程度の期間を必要としそうか、または「大丈夫そう」「非常に難しそう」などの感覚を掴みます。これにかける時間はタスクの内容やその日のやる気によって異なります。筆者は、30分程度のことであれば、他の急ぎのタスクがなければ、数時間取り組むこともあります。

5. 明日の予定を立てる

タスクの進捗状況の更新など、その日に行ったことをまとめ、翌日の予定を立てます。翌日中におおよそどの程度タスクを終わらせたいかのリストを作成します。翌朝の朝礼などで進捗報告をする必要がある場合は、共有する情報をまとめておきます。

朝に予定を立てる方法論もありますが、当日の方が記憶が鮮明であり、朝はバタバタすることが多いため、1日の終わりに翌日の予定を立てることをお勧めします。また、予定は業務時間内（7～8時間）で終わるように計画すべきです。

残念なことに、これらのスケジュールに突発的な依頼やトラブルのタスクが入ることがあります。予定していたタスクをこなせない場合は、それらのタスクを翌日に持ち越すか残業で対応します。しかし、残業は残業代という幾つかの幸せをもたらしますが、それ以上の多くの不幸をもたらします。残業は、最終手段であることを覚えておきましょう。

^{†1} どの程度の精度にするかによって難易度は変わりますが、筆者は1日単位で見積もりができないタスクを「未知のタスク」として分類します。

(例)

以下の例では、タスクAは見積もれていますが、タスクBは未知のタスクです。

タスクA：（確信をもって）これは3日で終わるな

タスクB：これは3日・・・いや4日かかるか・・・うーん・・・

6.3 目的と期限の明確化

多くの場合、私たちはプロジェクトという単位でチームを組んで仕事を進めます。複数のプロジェクトを同時に進行し、異なる役割を果たしていることもあるでしょう。すべてのプロジェクトが順調に進行しているとき、私たちは平穏に過ごすことができます。同時に進行するプロジェクトは、多少の混乱を引き起こすことがあります、順調であれば人間らしい生活を送れるはずです（順調でも人間らしくない生活なら、何かが間違っているですね！）。

これは、プロジェクトの目的や期限が要件定義書やプロジェクト計画書に明示され、プロジェクト管理ツールなどによって、私たちのタスクと期限が管理されているからです。

しかし、私たちは日々、多少とは言い切れない混乱の中にいます。以下は、私たちを混乱に陥れる要因となる突発的なことです。

- 想定外の仕様変更に対応
- プロジェクト外からの対応依頼
- 既存システムのトラブル対応依頼

これらのタスクはさまざまなところからもたらされます。チャット・メール・電話、もしかすると背後からの声かもしれません。そして、こう言われるのです。「よしなに！（もしくはASAP！！^{†1}）」。

おそらく、これらのタスクを断るのは難しいことが多いでしょう。そのタスクが断ることができるものならば、そもそも連絡がこなったはずです。私たちが混乱を少しでも抑えるためにできることは「目的」と「期限」をすぐに確認し明確化することです。

目的の明確化

予定がないタスクを依頼される際、依頼内容を実施するための背景や目的が省かれ「簡素な内容だけ」が伝えられることがあります。まず、私たちはその目的を明確にする必要があります。可能であればその目的に至った背景や経緯も明確にすべきです。

背景や目的が明確になれば、依頼された内容よりも簡単な解決策を見つけだせるかもしれません。トラブル対応においても、一時的な対応、恒久的な対応、原因の究明など、対応の範囲によって準備も時間も大きく変わってくるでしょう。

期限の明確化

目的を明確にしたら、次に期限を確認しましょう。トラブル対応の期限は難しいところですが、すでに復旧はしており「原因の究明をする」場合、数日など長期間を確保できる可能性もあります。

ただし、突発的なタスクに対しては、依頼者も明確な期限を持ち合わせていないことが多いです。期限を確認しても「可能な限り早く」「うーん、わからないなあ」のどちらかの返答があるでしょう。その場合は、私たちから「xx日まではどうでしょう？」などのように、具体的な期限を提案しましょう。

遅すぎると言われるかもしれませんが「もっと早くしなければいけない」ことは把握できます。期限をできるだけはつきりさせることが大切です。

「目的」と「期限」の明確化は、タスクに着手する前に、可能な限り早く行うべきです。私たちは多忙かもしれません、依頼をしてきた人たちも同じくそれ以上に多忙なはずです。依頼者は、依頼したことによって対応が即座に開始していると考えているかもしれません。

時間が経過した後に目的や期限を確認しようとすると、依頼者はイライラし私たちに不満をためるでしょう。さらに悪いことに、依頼者自身が詳細な内容を忘れてしまっている可能性もあります。

私たちは、これらの突発的なタスクへの対応よりも、本来決まっていたタスクを進めたいはずです。それでも「目的」と「期限」は早急に確認しましょう。

後回しにしても、その依頼が無くなることはありません。

† I ASAP (As Soon As Possible)。「できるだけ早く」という意味です。

6.4 ズレを許容する準備

私たちのタスクには、期限という時間的制約があります。しかし、どのタスクでも不具合やバグなど予期せぬ事象に遭遇し、予想以上の時間を要し本来必要とされていた時間とズレが発生する場合があります。そのため、責務範囲となるタスクにはズレを許容する準備、一般的に「バッファ^{†1}」と呼ばれる余力を持つことが必要です。

バッファを持つ方法について、以下に筆者の2つの考え方を紹介します。

自分が時間を決めることができるタスクの場合

1つ目の考え方は「自分が時間を決めることができる」タスクにバッファを持つ方法です。「どれくらい時間かかりそう?」と言われ、自分でタスクの期限や所要時間を決める場合です。この際は「経験のあるもの」「経験のないもの」の視点で考えます。

1.経験のあるもの

過去に同様のタスクを行った際の所要時間を振り返ります。例えば、以前に同じようなタスクに5時間かかった場合「5時間」を基準にします。過去の実績を基に1.5倍し「 $5h \times 1.5 = 7.5h$ 」で7.5時間とします。2.5時間がバッファになります。

この方法は、単純でわかりやすいのですが、2つの欠点があります。

- 10分などの短すぎるタスクは、バッファが意味をなさないほど短くなる
- 何十日もかかるタスクは、バッファが必要以上に長くなる

そのため、短いタスクについては、その性質（トラブルに発生する要因や対応に必要となる時間）を考慮し、10分などを追加します。タスクと同様の時間をバッファとして持つのは不自然なため、複数のタスクのトータルのバッファとして1時間などにすると自然になります。長いタスクの場合は、バッファを半日や1日など、やや大きな時間幅で追加すると自然になります。

2.経験のないもの

経験のないタスクは、難しい場合がありますが、以下のアプローチで考えます。

1. 類似タスクの実績者や残っている資料を参考にする
2. タスクに関連した技術情報の入手性を調べる
3. 短時間で試せそうなものは、概要部分だけでも動かし難易度を予測する

これら3つと現在の自分の知識レベルを考慮し勘案します。特に2つ目の入手性の容易さと情報量は、タスクにかかる時間を大きく左右するため念入りに調べます。これらの情報がない場合は、手探りかトライアンドエラーになるため、関係者に事情を説明しタスク調査の優先順位をあげるか、残業時間が多く使うなどの交渉をして、調査するための時間を確保しなければなりません。

すでに時間が決まっているタスクの場合

2つ目の考え方は「すでに時間が決まっている」タスクにバッファを持つ方法です。他の人がすでにタスクの期限や時間を決めてしまっている場合でも、正当な理由があれば変更できることがあります。多用はできませんが、重要なタスクの所要時間の算出が誤っている場合などは修正しなければなりません。

1. 自身の算出

すでに決まっている時間と自分の算出した時間を比較します。多少のズレであれば許容します。

2. 根拠の確認

ズレが大きい場合は、そのタスクの時間を割り当てた担当者へ根拠を確認します。タスクの内容への認識が異なるか、どちらかに把握できていない情報などが存在している可能性があるためです。

3. 影響範囲の確認

認識の相違が発見された場合や譲れない理由で時間の変更をしたい場合には、時間変更による影響範囲を確認します。この段階ではメンバーレベルで正確な影響範囲の把握は難しいため、自身の考えられる範囲で影響度を確認します。

4. 交渉

これまでの内容をまとめ、関係者に事情を説明し時間の調整を行います。

実際のところ、ここまでスムーズにバッファを持てることは稀です。多くの場合、望むバッファより短いか全くバッファを持てないことが多いでしょう。

それでも、バッファを持つための労力を惜しんではありません。ズレの許容されないギリギリのスケジュールは、多くの場合、悲惨な結果をもたらします。

†1 バッファ (buffer)。本節では、スケジュールにおける「時間的な余力」という意味で使用しています。

6.5 実績と私のスケジュール

私たちがタスクを割り当てる際、過去の実績に基づいて同じ所要時間で割り当てられることがあります。実績があることから、同程度の所要時間でできると期待されています。過去の実績が、類似機能の実装などで小さい範囲の端的なタスクであれば、タスクの所要時間に大きな差異は生じないでしょう。もし大きな差異が生じるのであれば、類似機能ではなかったか担当者の技術レベルに大きな差がある場合です。

しかし、同じタスク内容、同じ技術要素、同じ技術レベルであっても、複数の手順や関係者が関与する場合、所要時間に大きなばらつきが生じることがあります。

こうしたタスクには以下のようなものが該当します。

- ソフトウェアのリリース、バージョンアップ作業
- 計画停電対応
- 監視や運用などのルーチンワーク

これらのタスクは一定の周期で実施され、過去の実績を参考としやすいものです。手順書なども用意されていることが多いため、誰でも同じ所要時間で行えると想定されがちになります。

しかし、過去の実績があり、手順書が用意されていてもタスクの所要時間が変わる要因があることに気をつけなければなりません。

環境の慣れによる違い

組織や環境が異なる場合、運用ルールや特定の操作が影響を及ぼす範囲は異なります。初めてその環境で作業を行う場合、それらの違いを注意深く確認しながら行うため必要となる時間は増えることでしょう。顧客の運用中のシステムを想像してください。必要なときのみ操作する私たちより、普段から操作している顧客の方がスムーズに操作が行えるようなものです。

これは、顧客側に長期間常駐していたり、アカウントSEとして長期のノウハウを蓄積したエンジニアからタスクを引き継ぐ際によく発生します。

経験度合いによる違い

初めて対応するときは、手順書を細部まで見ながら1つずつ処理をしていきます。何度もこのタスクの経験のある人は、手順書の多くを読み飛ばし、押さえるべき手順のみを注視しているはずです。

これは、ルーチンワークなど短い周期で行われるタスクの引き継ぎでよく発生します。このタスクの経験者からすると、手順書があり日常的な作業になっているため、細かい説明がないことがほとんどです。

使用ツールによる違い

手順書が用意されている場合でも、その中に細かい多数のステップがある場合は、一部の処理をスクリプトやツールを使用して簡略化または自動化していることがあります。これらが手順書に明記されていれば良いのですが、ドキュメントや履歴として残っておらず、担当者内で暗黙的に行われていることがあります。この場合、スクリプトなどが使われている部分を、手順書の通りに手作業で行えば、実績と大きな差が発生します。

これは、古いシステムと同じ人物が長く担当している場合や、導入したばかりで本格的な運用に乗り切っていないシステムでよく発生します。

人間関係による違い

複数の人が連携して一連の作業を進める場合、逐次確認を行いながら進める必要があります。この場合「良好な人間関係が築けている人」や「コミュニケーション能力が高い人」たちが行えば短時間となりますが、それらがない人が行うと極端に多くの時間が必要になります。

これは、顧客側の担当者の変更や私たちの部署異動などにより、人間関係が大きく変わった際によく発生します。

「自動化すれば良いのでは？」との意見もあるでしょうが、頻度が少ない場合や自動化が複雑になるタスクは、自動化の対象から外れることがあります。そのようなタスクは、今後も人間が行うタスクとして残り続けると筆者は考えています。

もし、私たちがタスクを振られた場合、安易に過去の実績を信じてはいけません。「自分ならどのようなスケジュールで進められるか」を意識しておかなければ、実績と大きな差異が生じます。

6.6 空き時間なのか、待ち時間なのか

私たちのタスクの間には、他者のタスクが挟まることがあります。以下の順番で組まれたタスクがあるとしましょう。

1. 私のタスク
2. Aさんのタスク（私のタスクに影響しない）
3. Bさんのタスク（私のタスクに影響する）
4. 私のタスク

AさんやBさんがタスクを処理している間、私たちは稼働していない時間になります。この稼働していない時間が「空き時間」「待ち時間」のどちらなのかを意識しておかなければいけません。

空き時間

空き時間は自由な時間です。「Aさんのタスク」のように「私のタスク」に影響がなく待っている時間になります。このような「空き時間」は、自分の好きに使うことができます。別のプロジェクトのタスクを進めたり、雑務を片付けるなど、私たちが思うままに使える時間です。

待ち時間

待ち時間は自由な時間に見えますが、完全に自由にしてはいけない時間です。「Bさんのタスク」のように「私のタスク」に影響する（もしくは、影響する可能性がある）処理を他者が行っている間の待っている時間になります。このような「待ち時間」は、ただ待つだけの時間ではありません。

短時間の場合は、Bさんがどのようなことをしているのか注視し、自分のタスクに関するログや動作を逐一確認するべきです。

数十日かかるような長時間の場合も「待ち時間」です。「Bさんのタスク」の進捗状況や進捗報告に注視しておかなければなりません。

「待ち時間」を意識的に過ごせば、問題が発生した場合も早期に対処できます。「Bさんのタスク」に問題が出た場合、影響範囲内である私たちが呼び出される可能性はかなり高いでしょう。

このような問題を避けるために、PERT図やPDM図^{†1}で作成したクリティカルパスがあるじゃないか！という意見もありますが、私たちの現実にあるタスクにおいて「空き時間」「待ち時間」を把握するためにはいくつかの課題があります。

- クリティカルパスの役割は一連のタスクの最長経路の発見である
- クリティカルパスは私たちのが実施する細かなタスクまで網羅できない
- タスクの影響範囲やリスク評価を表現できない
- そもそもクリティカルパスが作成されないことも多い

WBS^{†2}やタスク管理表などでも同じような課題を抱えています^{†3}。

実際のところ「空き時間」「待ち時間」を正確に判断することは難しいものです。他者のタスクが、私たちにどのような影響を及ぼすかを完全に把握できなければ、判断できないからです。また「Aさんのタスク」が「Bさんのタスク」に関与しているなど、依存関係がある場合、判断が一層難しくなります。

しかし、幸運なことにいくつかの「待ち時間」を見つけることは、比較的容易です。私たちのタスクが影響を受ける他者のタスクは多く存在し、また私たちのタスクの多くも他者のタスクへ影響を与えるからです。

私たちがタスクを抱えた時、以下のことを考えましょう。

- そのタスクは、誰のタスクから影響を受けるか
- そのタスクは、誰のタスクに影響を与えるか
- 稼働していない時間は、「待ち時間」になりそうか
- 「待ち時間」にできることは何か

明日は「空き時間」と「待ち時間」どちらが多いでしょうか？

†1 クリティカルパスを視覚的にわかりやすいように表現するための図法の種類です。

PERT : Program Evaluation and Review Technique

PDM : Precedence Diagram Method

†2 WBS (work breakdown structure) はプロジェクトに必要なタスクをまとめるものです。本書では、WBSにガントチャートを追加したものやWBS辞書などもWBSと表記しています。

†3 筆者の考えになりますが、個人レベルのタスクの全てに対して、プロジェクト管理手法を適用するのは非現実的だと考えています。私たちが個人で管理するタスクと、プロジェクトで管理するタスクでは粒度が異なるためです。もし、すべてを管理しようと相当な時間と労力が必要になるはずです。

6.7 私の集中できる時間

筆者が通勤中にニュース記事を読んでいると、早朝出勤に関する記事を発見しました^{†1}。その記事には、以下のようなことが掲載されていました。

- 朝の早い時間は集中力が高い状態で仕事に取り組める
- 効率的に仕事を終わらせることができる
- その結果、早く帰宅できる

当時、全く共感できなかったことを覚えています。当時の筆者が集中できる時間は、定時を過ぎた18:00以降から終電までだったからです。早朝出勤を否定するつもりはありません。その記事には、通勤のラッシュ時間を回避できることなど納得できるものも掲載されていました。しかし、筆者の生活スタイルとは大きく異なるものでした。

私たちが集中できる時間を把握しておくことは大切です。難しい問題や初めてのことを取り組むときなどは、ある程度まとまった時間に集中する必要があるためです。本節では、自分が集中できる時間を発見するための、筆者の経験談を紹介します。

進捗の違いによる発見

プロジェクトが開始する少し前は、多くの集中する時間を取ります。プロジェクトを進めるために必要な情報がある程度集まっており、顧客への説明も開始していないため問い合わせもありません。

開始直後や進行中は集中できません。多くの認識違いが見つかり、多数の問い合わせ対応もあります。

終盤は、また集中できる時間取れるようになります。必要なタスクはほとんど終わっているからです（間違いをしていなければ！）。

働き方による発見

筆者が会社員のとき、日中は知識も時間も足りない多くのタスクに追われ、頻繁に会議も開催されていたため、集中できる時間は定時後でした。定時後は、会議も電話も少なくなるからです。

フリーランスとして働き出してからは、集中できる時間は日中です。会議は必要最低限

参加すればよく、日中の多くの時間を技術的なタスクに割り当てることができます（知識が足りていれば！）。

年齢による発見

筆者が若かりし頃は、夜に頭が冴え渡りました。金曜日の夜～土曜日の朝にかけては、ゴールデンタイムと言っていいほど集中できました。

現在はどうでしょう。日中で脳は疲れ切り、夜はほとんどの機能を停止します。筆者の脳が機能をフル回転できるのは、11:00～15:00ごろです。

内容に違いによる発見

現在の筆者が仕事に集中できるのは、平日のみです。土日に仕事をすると「なぜ休みの日にこんなことを？」という疑問で埋め尽くされ集中できません。仕事は、可能な限り平日に終わらせたいと考えています。

興味のある学習は、土日に集中できます。本を読んだり、何かを試す楽しい時間を過ごせます。仕事の難しい話や悩んでいることの問い合わせもありません。

気分による発見

現在の筆者は、長時間労働も徹夜も苦手になりましたが、難題なタスクの解決の糸口を見つけたときや恐ろしく順調に進捗しているとき、一種のランナーズハイになります。ランナーズハイの集中力は、筆者にとって一番集中できる時間帯です。この時は、時間のことを忘れて仕事をし続けることにしています。

これらは筆者の体験でしかありませんが、現在の自分が置かれている状況で「集中できる時間」を把握できれば、自分の全力で取り組む時間や残り時間で可能な最大の成果も把握しやすくなります。

「集中できる時間」は、プロジェクトの進捗や年齢もしくは働き方などにより変化します。筆者は、1つのプロジェクトが終わったタイミングで見直すようにしています。

皆さんの「集中できる時間」はいつでしょうか？その時間は、難題なタスクに取り組む最良の時間になります。

†1 2015年には、政府が「ゆう活（正式名称は、「夏の生活スタイル変革」）」と通称した、朝早くから働き始める 것을推奨하는活動があつたため覚えのある読者も多いことでしょう。

6.8 無駄な時間？私には必要です

筆者は、よく仕事が遅いと言われます。仕事が遅い原因には大きく3つあります。

資料の意図がわからない

タスクに着手するために、内容や前提条件を確認するため設計書などを読みますが、説明が不十分で意図が読み取れない場合があります。関連した設計との食い違いが発生している場合は、各設計者の認識や目的とすることが間違っている可能性を考えなければいけません。こうした考える時間や関連者へ確認に多くの時間を使います。

業務上の制度・運用ルールがわからない

手順書には実施すべき指示が書かれていることが一般的でしょう。指示の中で自身の知識では理解できないことを発見することがあります。技術的な内容であれば、それらを理解するために調べる時間が必要になります。業務上に関する処理の場合には、制度や運用ルールが深く関わっていることがあります。この場合は、関連する業務や運用ルールの意図を読み取ったり、関係者へ確認する時間が必要です。

何が正解かわからない

初めての技術を仕事で使用する場合、とりあえずの正常な動作を確認するのは比較的容易です。しかし、要望に沿うような動作や想定外の動作が発生した場合、どのように対処すべきかがわからぬことがあります。こうした場合、構成を変更してみたり、処理内容を変えてみたり、擬似的に想定外の動作をさせて確認することに多くの時間を使います。

これらはタスクの期限が守れる場合、業務時間の中で行います。交渉で業務時間と認められなければ、業務時間外を使ってでも行います。これらの中には、無視して取り組んでも問題となることもあります。実際に多くの時間を使いました。そして、現在の筆者もこれらに多くの時間を使うことがあります。

時間を使うことによって、問題を未然に防ぎ感謝されることもあるれば、時間の浪費や予算を無駄に使っていると指摘されることもあります。

しかし、これらは筆者にとって「私の責務範囲としてやりきった」というために必要なものばかりです。時間を惜しんで失敗した場合、筆者は謝罪することしかできず、大きな後悔の念に駆られることでしょう。

これらの実践を推奨するわけではありません。大切なのは「遅い原因を説明できる」とと「必要であったと確信できる」ことです。

もし、仕事が遅いと指摘を受ける場合、遅い原因を説明できるようにしてください。仕事に時間を要する原因は、他にも以下のような要素が考えられます。

- 知識や経験不足によるもの
- タスクの目的や優先順位が明確でない
- 異なるタスクが多すぎる（スイッチングコストが高い）

説明をした後に反論されることもあります。しかし、説明後もなお必要だったものと確信できるならば、その時点の私たちにとって必要だったものです。無理に反論し、相手を納得させる必要はありません。必要だったものという事実を自分で認めるだけで良いのです。必要だったものは、発生するはずだった問題から私たちを助け、経験として蓄積されていきます。

自分に必要なものと確信するために、注意することが2つあります。

期限は守る

1つ目は、期限を守ることです。どれほど必要なことであったとしても期限を過ぎてしまっては、使った時間で成し得たことも、相手からの信頼も失います。期限を過ぎるのであれば事前の相談が必要です。

質問は解決への近道

2つ目は、関係者や有識者などに質問することで解決する問題も多いことです。もし、質問をせず自分で時間かけていては、誰からも浪費時間と見做されてしまうでしょう。

仕事に使っている時間は無駄な時間ですか？それとも、自分にとって必要だと確信できる時間ですか？

6.9 ヒトは1週間、モノは1ヶ月

私たちがスケジュールを考える場合、自分で完結するタスクであれば、好きなようにスケジュールすれば良いでしょう。朝早くから働くか、夜遅くまで働くかは、組織の規則に従っている範囲では、私たちの自由です。

しかし、自分で完結しない場合、好きにしてはいけないことが2つあります。

- 他者に依頼する場合の「人の手配（ヒトは1週間）」
- 必要な物を購入する場合の「物の手配（モノは1ヶ月）」

詳細は後述しますが「1週間」「1ヶ月」という期間は、参考値であり常に正しいとは限りません。

ヒトは1週間

私たちは他者に何らかのタスクを依頼することができます。これは同じチームメンバーや同僚、顧客に向けてかもしれませんが「〇〇の準備をお願いします」などのように依頼をすることがあります。依頼される側は、数週間先までのスケジュールを組んでおり、土日の予定も決まっているかもしれません。そのような状況で「次の日までお願ひします」と依頼するとどうでしょうか？おそらく多くの人たちは不快に感じるでしょう。筆者は金曜日の夕方に、翌週月曜日までのタスクを依頼されると憤りを感じます。

「ヒトの手配」を考える際には、遅くとも1週間前には相手に依頼すべきです。小さなタスクであれば、すでに組まれているスケジュールの間に柔軟に対応してくれるはずです。依頼のタスクに5営業日も必要ですか？そうであれば、1週間前は遅すぎます。1ヶ月前に依頼しても不快に感じるかもしれません。

これに正解はありません。依頼するタスクの量や人間関係によっても変わってきます。大切なことは、相手にもスケジュールがあり、多くの場合すでに別のタスクで埋まっていることです。そして前日に依頼すれば、彼らをほぼ確実に怒らせることでしょう。

モノは1ヶ月

現代では、さまざまなモノを比較的短時間で入手できるようになりました。クラウドサービスは、数十分もあれば利用を開始できます。ソフトウェアに関しては、オンラインで

ライセンスを購入すれば、すぐに使い始めることができます。しかし、実際に仕事でモノを入手するには、多くの時間が必要です。

モノを入手するときに、よくありそうなことを例にしてみましょう。

まず、営業担当者から見積もりを取得しましょう。少なくとも数日はかかります^{†1}。海外製品の場合、本社へ確認を取る必要があればもっと多くの時間が必要かもしれません。見積もりを取得した後、承認を得るため稟議を提出すると、承認者の不在が多く5日も経つてしまいました・・・。

「モノの手配」は遅くとも1ヶ月前には着手しましょう。しかし、この期間にも正解はありません。エンタープライズ機器のように特定の要件が必要な場合は、1ヶ月前では遅すぎるでしょう。これは例えクラウドサービスの場合でも、その利用の是非の検討と承認を得るために多くの時間が必要になります。少なくとも数分後や翌日に入手できると考えてはいけません。

では、ヒト・モノの手配は早いことが正解でしょうか？「早すぎる手配」も失敗の元になります。いくつか誇張した例を挙げてみましょう。

10年後の旅行の予約をしました。おそらく筆者ならば1年も経たずに忘れていました。新しいパソコンを購入し10年後に納品します。箱を開けるころには、サポート期限は切れ、正常に動作するかも怪しいものです。

これらの正解はケースバイケースです。組織文化や社会情勢にも大きく左右されます。共通することは、ヒト・モノの手配には、相当の時間が必要であることです。ヒト・モノの手配が必要であることがわかった時点で、それらを入手するために必要となる時間の算出と関係者へ事前連絡などの根回しをしておくべきです。

私たちにとってどれほど緊急度が高く必要だったとしても、世界は私たちを中心として回ってはいないのです。

^{†1} これは誇張しすぎて語弊があるかもしれません。依頼から数時間後に見積もりを提出する営業の方も多くいることでしょう。

6.10 残業は悪か

筆者は、残業は善か？悪か？と問われれば「悪である」と答えます。残業は、私たちからさまざまなものを持ち去ります。睡眠時間を奪い、集中力を奪い、心と身体に負担をかけ不健康にもします。プライベートでは、友人や家族と過ごす時間を奪うだけでなく、関係を壊すこともあります。

ただし、筆者は残業を「絶対的な悪」とは考えていません。現在でも必要な場合には残業をしています（この文章を書いているのは20:00ごろです！残業が発生している！）。残業が絶対的な悪であるならば、働き方改革^{†1}で残業が規制される前に残業は消え去り、都内のビジネス街の夜景は素晴らしい景色ではなかつたでしょう。

筆者が、残業が絶対的な悪ではないと考える理由を挙げておきます。

仕事を終わらせる

筆者は、仕事が遅いです。準備にも実行にも多くの時間を必要とします。それでも、残業をすることにより、短納期の仕事や急に入った緊急度の高い仕事を終えることができました。残業がなければ、約束の期日までに何も成し遂げられないITエンジニアになっていたことでしょう。

失敗を抑止する

時間に追われる中で行う仕事には、漏れが発生する可能性が高くなります。資料の誤字脱字、設計や手順書の記載漏れなどです。筆者は、資料作成も実装も非常に遅いです。資料のブラッシュアップや技術的な仕様の調査は、ほとんど残業でしていました。これらは、本番システムに関する多くの失敗するかもしれないから筆者を救ってくれました。

人間関係の形成になる

残業を共にする同僚たちは、多くの困難な問題に直面し、解決策を模索しています。同じチームメンバーまたは外部のメンバーでも、共に困難な問題に取り組むことで少なからず絆や団結力が生まれます^{†2}。筆者が現在も連絡を取り合っている元同僚の多くは、共に残業を経験した人たちです。

成長の機会になる

筆者は技術を理解したり身につけることにおいて多くの時間を必要とします。しかし、残業をしなければいけない状況のときは、仕事に没頭しなければいけないときでもあります。1日の大半を仕事や技術に触れながら過ごします。学ぶことがいくら遅いと言っても否応なく少なからずのことが身に付きます。

心の余裕を生む

残業は心に負担をかけると述べましたが、ときには余裕を与えてくれます。設計や実装中に、ちょっと気になる点があるとき、残業することができればその点を見逃さずに問題を修正できるかもしれません。しかし、残業ができない場合、その点を見過ごしてしまい、後でトラブルの原因となることもあります。

これらには異なる意見もあるでしょう。仕事を効率的に進め、最短時間で最大の成果をだすことができれば、残業は必要ないはずです。先ほど挙げた筆者の考えも、日中業務の中でできる人も多くいます。しかし、筆者には無理でした。そのため現在でも残業をすることがあります。

残業は絶対的な悪ではありませんが、悪であることに変わりはありません。筆者の考えも絶対的な悪ではない側面であって、善の部分ではありません。残業を美德とすることもありません。しかし「残業は悪いもの」だからと残業を全面的に否定し、時間が無いことを理由に自身の責務を放棄したり、期日を守らないことは「悪」であると考えます。

私たちは残業が悪であり、最後の手段であることを忘れてはいけません。しかし、手段が残っているにもかからず、責務を放棄することも悪であることを忘れてはいけません。

†1 正式名称は「働き方改革を推進するための関係法律の整備に関する法律」です。個々の事情に応じた多様で柔軟な働き方を自分で選択できるようにするための改革で、時間外労働の上限規制が設けられました。

†2 筆者はこの考えを妻（当時は彼女）に話したことがあります、理解はしてもらえませんでした。ITエンジニアの読者は理解してくれる信じています。



第

VII

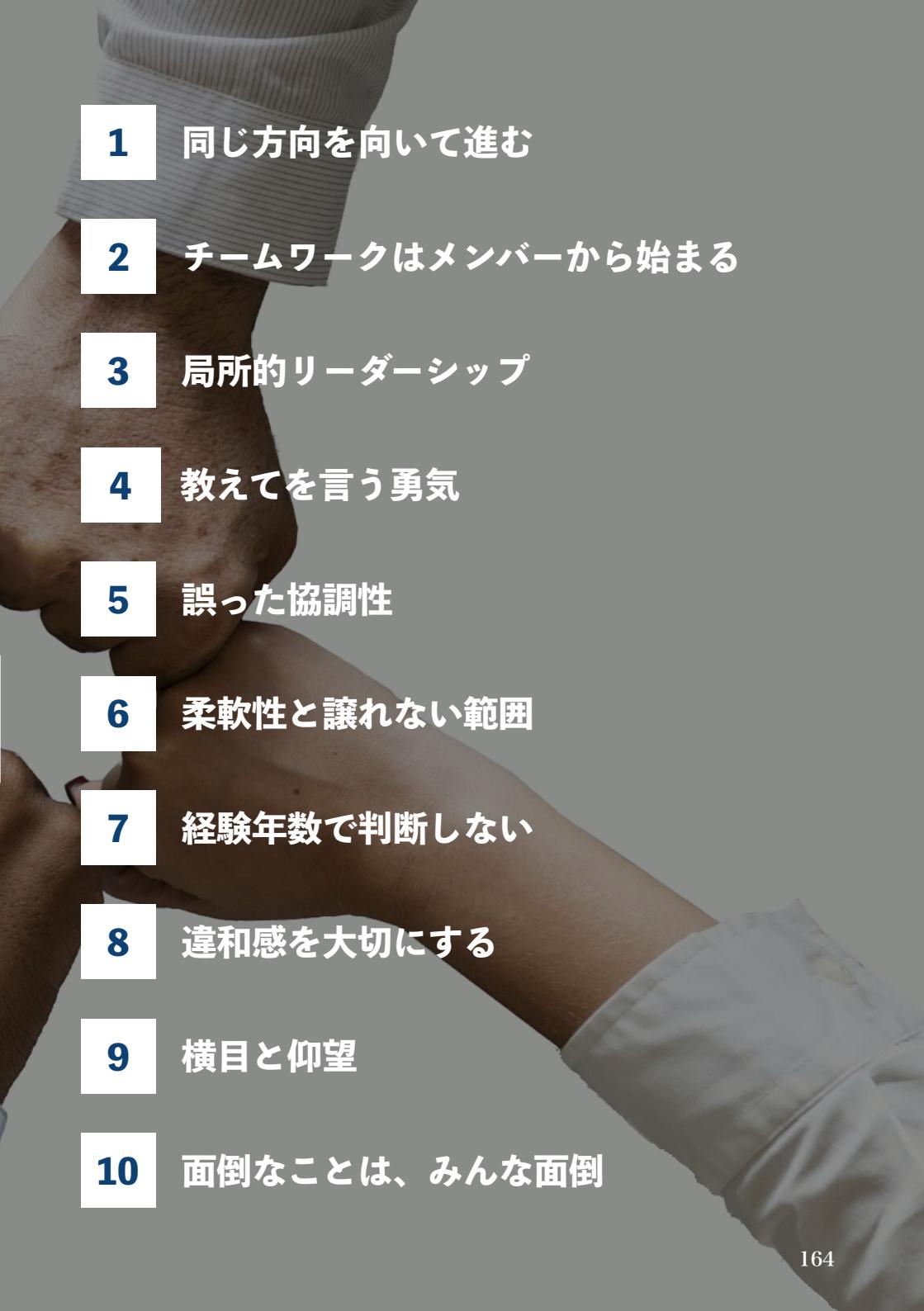
章

無能のためのチームワーク

本章は、無能なITエンジニアとして生きていくための
チームワークについての教訓になります。

プロジェクトチームとして複数の人たちと協力する際、
チームワークは非常に大切です。

私たちがチームのメンバーとして貢献できることは、
技術以外にも多く存在します。



1 同じ方向を向いて進む

2 チームワークはメンバーから始まる

3 局所的リーダーシップ

4 教えてを言う勇気

5 誤った協調性

6 柔軟性と譲れない範囲

7 経験年数で判断しない

8 違和感を大切にする

9 横目と仰望

10 面倒なことは、みんな面倒

7.1 同じ方向を向いて進む

筆者は、プロジェクトマネジメントとは「カルガモ親子の行進^{†1}」のようなものだと考えています。「カルガモの親」はPMやPL、「カルガモの子どもである雛」が私たちメンバーです。

プロジェクトにはスタートとゴールがあり、全員でゴールを目指すものです。しかし、PMやPLが正しい方向を示していたとしても、私たちは雛のようにフラフラと別の方向に歩くことがあります。彼らは、そんな私たちを見つけては、正しい方向に軌道修正してくれます。そう考えると「カルガモ親子の行進」はしつくりくるのではないしょうか？

テレビで「カルガモ親子の行進」を見ていると、親は大変そうだと感じます。

1. 後ろを振り返っては雛たちの数を確認する
2. 間違った方向に進む雛がいないか確認する
3. 間違っている雛がいれば軌道修正する

これらはPMやPLがプロジェクトで行っていることです。その他にもさまざまなマネジメント業務がありますが、私たちメンバーが迷子にならないように気を配ってくれています。ゴールの決定と方向の指示、間違いを修正する役割はマネジメントにありますが、私たち自身が間違った方向に進まないようにする方法がいくつかあります。

プロジェクト計画書を見る

プロジェクトがスタートする際に使用される資料を見るようにしましょう。プロジェクト計画書やそれに類似する資料が存在すれば、大チャンスです。プロジェクト計画書は、プロジェクトを開始するために必要なRFI（情報提供依頼書）、RFP（提案依頼書）、要件定義書などの難解な資料を、私たちにも理解しやすい内容でまとめてくれています。

更新される管理資料を見る

プロジェクトを管理するための資料はさまざまですが、進捗報告・課題管理・WBS^{†2}など、更新される可能性のある資料を定期的に（できれば毎日）確認しましょう。これらの管理方法は自社のシステムやクラウドサービス、オフィスファイルなど複数ありますが、更新履歴の確認程度であれば、それほど時間はかかるないでしょう。更新の内容も自身に

関連する部分だけを確認すれば、少ない時間で行えるはずです。これらの変更は、後で私たちの追加タスクとして降りかかってくる可能性が高いものです。

他社・他部門の資料を見る

プロジェクトでは複数の組織や部門が関与する場合があります。それぞれが作成している設計書や手順書などの資料を積極的に見るようにならう。多くの関係者が関わるプロジェクトでは、資料の品質にばらつきが発生します。事前の取り決めがない場合は、大きなばらつきが発生し、後で修正が必要になる可能性が高まります。プロジェクト内で同じ工程レベルの、別のチームが作成している類似資料があれば、品質を確認してから作成することで大きなばらつきを抑えることができます。

コストを見る

私たちのコスト（費用）を見ておきましょう。私たちは問題の解決や品質の向上のために残業や休日出勤を行うことがあります。多くのPM・PL・上司たちは私たちの時間外労働を嫌がります。私たちが余分に働けば働くほどプロジェクトや部門の利益が減少するからです。コストの管理は上司やPMの役割かもしれません。自身のコストを把握しておくことは、交渉や調整にも役立つことがあります。

メンバーの中には、知らないことを理由に歩くことを諦めたり、怒って別の方向に進んでしまう人たちがいます。彼らの多忙さを考えると、その不満に頷ける部分もありますが、「知らない」の多くは「知ろうとしなかった」ことが多くを占めます。

マネジメントを行う人たちも人間のため、全能ではありません。私たちが「知ろうとする」とことで「こっちだよ」と言われなくとも同じ方向に歩けることもあります。

†1 ITのプロジェクトチームを「船員」や「二人三脚」で例えられることもありますが、筆者はピンと来ない表現でした。それらは、全員で間違った方向に進むか、全員でゴールできるかのどちらかだからです。実際のプロジェクトでは、さまざまな人が全く違う方向に進んでしまうこともあります。テレビで「カルガモ親子の行進」をみているときに、別の方向に進む雛をみてしつくりきたことを覚えています。

†2 WBS (work breakdown structure) はプロジェクトに必要なタスクをまとめるものです。本書では、WBSにガントチャートを追加したものやWBS辞書などもWBSと表記しています。

7.2 チームワークはメンバーから始まる

私たちの多くは、プロジェクトチームというさまざまな役割を持つメンバーたちのチームで仕事をしています。各メンバーが必要なスキルを満たしていることも大切ですが、もう1つ大切なことはチームワークです。チームワークは、私たちのパフォーマンスとチームのコミュニケーションの効率を高め、新たな知識を得る機会を提供してくれます。

素晴らしいチームワークを築く役割は、PMやPLなどのマネジメントにあります。彼らはチームビルディングのため多くのことを考え実践してくれます。

しかし、チームワークは単に提供されるものではありません。筆者はむしろ、チームワークはメンバーたち自身が築くものだと考えています。PMやPLがどれほどチームビルディングに尽力しようとも、メンバーが積極的に関与しなければ、チームワークは形成できなければなりません。そして、チームワークの恩恵を最大限享受するのもメンバー自身です。

私たちメンバーが主軸となって、素晴らしいチームワークを築き、維持するために必要なとなる、いくつかの方法を紹介しましょう。

チーム内に敵を作ってはならない

敵はチームワークにとって必要です。共通の敵はチームの団結力を強め、鼓舞する原動力となります。気をつけなければならないのは、私たちはしばしば誤った敵を作ってしまうことです。

- 技術力や問題解決能力の低いメンバー
- コミュニケーションが苦手なメンバー
- 無理難題を持ちかけてくるPMやPLたち

これらは敵ではありません。本当の敵は、体制や方法論・技術的課題などです。チームワークに敵は必要ですが、チーム内に敵を作ってはいけません。

情報提供を惜しんではならない

私たちは技術に関する多くの情報を持っています。経験値のみならず、参考となるURL、過去に行った試行錯誤の履歴、リファレンスや組織内のナレッジに蓄積された情報

などです。何か問題を抱え悩んでいるメンバーがいる場合、これらの情報提供を惜しむべきではありません^{†1}。これらの情報は私たちの知的資産ですが、提供しても資産は減少しません。これらの情報が私たちを助けてくれたように、多くのメンバーも助けてくれるはずです。

疑わなければならぬ

PMやPL、メンバーたちは、ときおり難解な専門用語や小難しいロジックを交えて設計や実装に関する議論をします。このような場合でも、私たちは疑いを持って聞くべきです。疑念は私たちをより注意深くし、理解を深め新しい知識を得る機会になります。また、誤りや齟齬を見つけた場合は、それを指摘し修正することもできるでしょう。

不満は伝えなければならぬ

プロジェクトの進行中に、いくつかの不満を抱くことでしょう。これらの不満はチーム内で伝えていくべきです。同じ不満を抱くメンバーを見つけられるかもしれません。PMやPLたちは、私たちが不満を抱くことを事前に排除しているはずですが、彼らは超能力者ではありません。私たちが見たり感じた不満を彼らが知る方法は、私たちの言葉を通じてしかありません。それらはプロジェクト全体の改善に繋がることもあります。

拒否の理由が忙しさであってはならない

メンバーの間でもタスクの交換や依頼を行うことがあります。特に小さいタスクは、PMやPLたちが関与することはほとんどないでしょう。このような場合、忙しさを理由に拒否をするべきではありません。拒否する本当の理由は、以下のようなもののはずです。

- 他の優先タスクがある
- 経験不足で成果の品質に不安がある
- 大切なプライベートの用事がある

拒否する場合は、正確な理由を伝えなければなりません。

どれほど素晴らしいPMやPLたちも、彼らだけでは素晴らしいチームワークを築くことはできません。チームワークは私たち全員のものだからです。

^{†1} 組織内のナレッジなどは、守秘義務や社外秘になる情報も多いです。社外のメンバーに共有する場合は、事前に注意深く確認してから提供する必要があります。

7.3 局所的リーダーシップ

リーダーシップを簡潔に説明するなら「組織や集団をまとめる統率力」と言えます。プロジェクトにおいては、PMはプロジェクト全体を統率するためリーダーシップを発揮し、PLはチームを統率するためにリーダーシップを発揮します。

しかし、多くのプロジェクトを経験した人であれば、彼らが完璧なリーダーではないことを理解しているでしょう。彼らは生まれつきのリーダーではなく、役割を果たすために、リーダーシップを発揮することに尽力しています。

リーダーシップはPMやPLの専売特許ではありません。彼らほどではありませんが、私たちメンバーもリーダーシップを発揮できることがあります。筆者は、このようなメンバーの発揮できる小さなリーダーシップを「局所的リーダーシップ^{†1}」と呼んでいます。

筆者が見た「局所的リーダーシップ」を発揮したメンバーたちを紹介しましょう。

技術によるリーダーシップ

とあるプロジェクトで技術的な問題が発覚し、その問題は複数の技術要素が関わり、設計の修正に及ぶことがわかりました。あるメンバーは、問題の原因を特定し、各技術のPLたちを集めて修正が必要な箇所を詳細に説明し、問題を解決しました。

彼は、自分の技術領域だけでなく、さまざまなIT分野が大好きな技術オタクでした。

代理によるリーダーシップ

PLが不在の際、あるメンバーが代理として会議に参加して顧客からの質問に対応し、調整が必要な課題などをまとめました。急ぎの課題は担当のメンバーへ引き渡し、残りの課題と進捗についてはPLと連携しました。

彼は、普段から話すことが好きで、いろんな人たちと雑談をするITエンジニアでした。

育成によるリーダーシップ

育成は大変です。技術的な用語や複雑な仕組みについて、多くの時間を使って説明しても、教えられた側の成長はいつも期待値より遅いものです。あるメンバーは、新しく入った新人メンバーに、開発環境や技術的な説明を難しい言葉を分かりやすく言い換え、とき

には手書きのメモを使い説明していました。また、同じ質問にも説明の仕方を変えて、伝わるように努めました。

彼自身も技術に精通していたわけではありませんが、講師の経験があり人に教えることが得意でした。

モチベーションによるリーダーシップ

メンバーたちはいつも多くの悩みを抱えています。納期や技術的な課題などにイライラすることもあります。非常に厳しい要件と納期のプロジェクトで、あるメンバーは周囲に笑顔で接し、雑談をしてきて苦難や不満をチームと共有しました。彼の明るい笑顔に触発され、筆者たちメンバーも笑いながら雑談していました。そして、筆者たちメンバーは、笑ってプロジェクトを終えることができました。

彼は若く、経験は浅かったものの、技術や仕事を心から楽しんでいました。

サポートによるリーダーシップ

ひどく単調で、非常に時間のかかるタスクがありました。メンバーの誰かが行わなければならぬタスクでしたが、優先度は低くタスク管理もされていませんでした。誰もが見て見ぬふりをしていましたが、あるメンバーがそのタスクをすべて終わらせていました。

彼は技術の習得が遅いと言われていましたが、細かな点に気がつき、継続的な努力を惜しまない姿勢を持って取り組める人物でした。

ここで紹介した素晴らしい局所的リーダーシップを發揮した彼らは、全て別の人物であり、PMやPLではなくメンバーの一人でした。

私たちは、自身の局所的リーダーシップを知ることで自身の長所を知り、チームに貢献する機会を増やすことができます。

†1 筆者の作った造語です。

7.4 教えてを言う勇気

最初に言っておくと、筆者は「教えてもらう」ことが好きです。以下のような、さまざまな場面で他者から教えてもらう機会を楽しんでいます。

- ITエンジニア向けカンファレンスや技術研修
- 組織内で開催される勉強会
- 先輩や後輩が知っている役立つ技術や方法

「教えてもらう」機会は、私たちに多くのメリットをもたらします。

- 新しい技術の初歩的な段階をスムーズに取り組める
- 自身では調べなかつたであろう情報を得ることができる
- 自身では発想できなかつたアイデアを得ることができる

新人や初学者の場合、社内研修や先輩からの指導など、待つばうけでも教えてくれる機会は多いです。しかし、経験を積むにつれて、このような機会は減少していきます。

そのような段階で大切なのは、自分から「教えて」と伝えることです。しかし「教えて」が言えない人も少なからずいます。また「教えて」が言えず「わかった振り」をしてしまう人もいます。これらの行動は、学ぶ機会を逃すだけでなく、大きなトラブルを引き起こす可能性を高めることもあります。

「教えて」が言えない理由はいくつか考えられます。

何を教えて欲しいかわからない

初めてのことを学び始めた時によく見られる理由です。

「わからない」ことは分かっているものの、「わからない」ことが多すぎて、その中で何について教えてほしいのか自分でも理解できていない場合です。この場合は、2つのステップが必要です。

1つ目のステップは「わからない」を細分化することです。細分化されたものは単語レベルになることもあります。細分化したものを1つずつ自分で調べていきます。調べても理解できない、もしくは細分化したものを繋ぎ合わせることができないものを、リストアップしておきます。

2つ目のステップは、細分化して調べてもわからないリストアップしたものを1つずつ「教えて」と伝えることです。単語の意味がわからないのか、それらの関連性がわからないのかなど、どの視点でわからないかも併せて伝えましょう。

無知と思われる・怒られる

IT技術の中には、単語や仕組みなど暗黙の了解として広がっていることがあります。これを知らないことが無知であると感じさせ「教えて」を躊躇わせることがあります。しかし「聞くは一時の恥、聞かぬは一生の恥」です。自分の知識や理解を広げるために積極的に質問していきましょう。

知らないことを嘲笑されたり、叱りを受けることもあるかもしれません。彼らは、私たちから学ぶ機会を失わせようとしている人たちです。以後は、他の人に聞けばよいでしょう。

経験年数・プライドによるもの

経験年数が長くなると、プライドを持ちがちであり、技術担当者としての自信から、自分より経験の浅い人や若い人たちに「教えて」を躊躇うことがあります。自信を持つことは大切ですが、優越感に浸るプライドは害となります。

そのような優越感を持つためのプライドは、学びの機会を逃し、他者との協力を妨げることがあるため、捨て去るべきです。

私たちは、学ばなければ仕事ができないことも、学ぶことが大変なことも知っています。同時に他のメンバーは、私たちが学び、さまざまな仕事を遂行することを期待しています。

私たちの経験年数などに関係なく、教えて欲しいことを明確にし「教えて」と伝えれば、多くの人々は喜んで教えてくれるはずです。

7.5 誤った協調性

協調性とは、一般的に「利害や立場が異なる人たちが協力し合う性質・性格」と言われています。就職活動の自己PRなどでも協調性が重視され、多くの場面でアピールポイントになるほど、世の中では協調性が好まれているようです。筆者の経験からも、協調性を重んじる人は多くいる感じます。

では、私たちに協調性は必要でしょうか？筆者の考えでは必要です。私たちは多くの場合、プロジェクトチームなどの複数人の協力体制で仕事を行うからです。協調性は、チームで仕事を円滑に進める上で大切なものです。しかし、注意が必要なのは、しばしば「誤った協調性」を発揮することがある点です。

「誤った協調性」には次のようなものがあります。

無言という誤り

無言により周囲の意見に「暗黙的に了承した」とすることがあります。

- 既定方針で覆すことができないという気持ちから無言になる
- 経験の長い人物の発言に、自分が指摘することはないと考えて無言になる

既定方針でも、既定の中に大きな問題が潜んでいる可能性があります。経験が長いからと言って、見落としがないとは断定できないでしょう。私たちの考える全てが網羅され、疑念もなく納得できる場合は無用な発言は控えるべきです。しかし、考える前に黙ってしまうのは、協調性ではなく「考えることの放棄」になります。

正しさの追求による誤り

私たちは、正しくあろうとします。技術的ベストプラクティスを採用し、より良い実装を目指し、運用時の負荷を下げようとします。周囲に正しさを主張し、理解が得られないと怒り出すこともあります。それらは、技術的には正しいことが多いでしょう。

しかし、他者に受け入れられない正しさは、往々にして「技術的な観点からだけの正しさ」です。プロジェクト全体を考慮すると、予算や人材、スケジュールなどの理由から、正しいとは言えないこともあります。そのような、周囲の人たちから納得を得られない中

で技術だけの正しさを追い求めると「わがままな人」になってしまいます。

教育という誤り

私たちが指導者側の立場になったとき、スキルの向上や教育の一環として、新たな技術や難易度の高いプロジェクトに参画を依頼したり、難しいタスクを割り振ったりすることがあります。これにより、指導を受ける人たちの成長を手助けをしようと努めます。これらの成長に協力しようとする協調にも注意が必要です。

私たちは指導者として、教育の目的や意図を正確に伝え、教育に必要となる前提の知識やノウハウを伝えられているでしょうか？もしできていなければ、教育の放棄と同じです。指導を受ける人たちを崖から突き落とし、自力で這い上がってくることを期待しているようなものです。

逆に、私たちが指導を受ける立場の場合、教えられることや指示を待ち続けていいでしょうか？もし待っているだけであれば、親からの餌を待つ雛鳥と同じです。

筆者は、協調性についていくつか違和感を覚えることがあります。

- 協調性が「ある」か「ない」の両極端で語られる
- 協調性が性格によるものだと語られる
- 協調と同調が同じ意味のように使われる

私たちに必要な協調性とは「協力し合う意識」だと考えています。この意識は、どのように協力できるか考えることで養うことができます。程度の差はあるかもしれませんのが、協調性がまったくない状態は避けられるでしょう。

また、協調性は何にでも同調するような「イエスマン」になることではありません。

私たちがチームで協調するとき「誤った協調性」を発揮してはいけません。協調には「協力」と「異なる意見を尊重」する意識が必要です。

7.6 柔軟性と譲れない範囲

私たちは、単に指示通りのタスクを遂行するだけでなく、さまざまな変更に対応する必要があります。

大きな変更への対応は、契約内容や将来の顧客関係などを考慮し、PMや上層部によって決定します。私たちに関する小さな変更には「柔軟性」と「譲れない範囲」の指針を持つことが必要です。この指針は、個人のアイデンティティを保ちつつ、チームの協力関係を高めることに役立ちます。

本節では、筆者の考えている「柔軟性」と「譲れない範囲」について紹介します。

「柔軟性」とは、困難でも対応可能であることです。面倒で時間のかかることかもしれませんが、努力によって対応できることを意味します。柔軟性が必要となる状況には、以下のようなものがあります。

技術要素への対応

プロジェクトの進行中には、技術関連の変更が生じることがあります。プログラミング言語やプロダクト自体の変更は稀ですが^{†1}、これらに関連するライブラリの変更や予定でいなかつたプロダクトの機能を新たに使用するような変更はあるでしょう。もしくは、初期の提案が変更になり、設計や実装を変更しなければならないこともあります。

これらの変更は、期間を考慮する必要はありますが、頑張れば対応できるものが多いです。

タスクの優先順位への対応

タスクの優先順位が変更されることがあります。これらの変更は、チーム内の問題や顧客からの要望など、さまざまな理由によりもたらされます。多くの場合、急に新たなタスクが発生したり、既存のタスクの期限が短く変更されます。

残念ながら、これらの変更は緊急性が高いことが多いにもかからず、他の既存のタスクの期限が伸びることは、ほとんどありません。そのため、残業時間や休日出勤が必要になりますが、それだけ必要となるタスクのため柔軟に対応する必要があります。

「譲れない範囲」とは、私たちに大きな不利益をもたらす可能性がある、または理不尽な変更への線引きです。これらの変更は、努力で対応できるものもありますが、柔軟な対応の前に、交渉を実施すべきものです。交渉すべきものには、以下のようなものがあります。

期限への対応

プロジェクトの納期と同様に、私たちに割り当てられた小さなタスクの期限も大切なものです。小さなタスクの遅延は、プロジェクトの破綻に直結しないかもしれません、私たちの信頼を失墜させます。

理不尽なタスクの期限変更は受け入れ難い場合があります。本来10日後が期限のタスクが急に2日後に変更されたり、今日発生したタスクの期限が翌日までというのは、安易に許容できるものではありません。この場合には、詳細な理由を確認し、期限の延長などに関する交渉が必要です。

品質と信頼性への対応

私たちが担当するタスクの成果に対しての品質や信頼性を低下させる指示があることがあります。例えば、以下のようなものです。

- 予算が不足しているため、設計書は省いて良い
- 時間が不足しているため、テストは最小限で良い
- とりあえず動くものが必要なので、最短でお願い

これらの指示は、成果の品質や信頼性を下げるばかりではなく、私たちの信用も損なう可能性が高いものです。予算や時間の不足も理解できる部分ですが、最低限の設計書やテストをするリソースの確保や時間的な余裕を得るための交渉が必要です。

これらの上層部の関わらない小さな変更はチーム内で議論的になります。変更は、少なからずの負荷を個人やチーム全体へもたらすためです。このようなチーム内の議論の際、私たちが「柔軟性」と「譲れない範囲」の指針を持っておくことで、対応の方針を決める議論の助けになるでしょう。

†1 筆者は、プロジェクト開始後にプログラミング言語やプロダクトそのものが変わることはない信じていました。しかし、設計時に提案内容の要望が満たせず、プログラミング言語やプロダクトが異なるものに変更となった経験があります。事実は小説より奇なりです。

7.7 経験年数で判断しない

ITエンジニアの評価の指標の1つに「経験年数」があります。転職サイトを見ても、必須事項に「○○の経験が○年以上」という要件が記載されていることが多いでしょう。また、私たちもチームに新たな人が加わる際、その経験年数を基準に評価し、一喜一憂することが少なくありません。

- PMが初めてという人であれば、プロジェクトが円滑に進むのか不安になる
- PMを長年経験した人であれば、安泰だと楽観視する
- 未経験のITエンジニアであれば、教育コストがかかること悲観する
- ベテランのITエンジニアであれば、救世主が現れたと期待する

これらの感情は、一定の根拠があるかもしれません、必ずしも期待通りではありません。良くも悪くも裏切られることがあります。

「良い裏切り」には以下のようないものがあります。

別分野からのPM

IT業界ではあるものの、全く異なる分野から異動してきた人物が、初めてPMの役割を担うことになりました。筆者は、スケジュールを適切に管理できるのか、技術的な会話を理解してくれるのかなど、不安を覚えました。

初期の段階では、彼の技術的な知識は不足しているように見受けられました。しかし、彼は素晴らしい柔軟性を持っており、不足している知識を積極的に学び、理解することに努めました。また、メンバーの技術的な発言にも耳を傾け、理解をしようとしてくれていることが感じられました。

時間が経つにつれ、筆者の不安は杞憂に過ぎないことがわかりました。彼はプロジェクトを成功に導くことに全力を尽くしてくれました。筆者が尊敬するPMの一人です。

未経験の若手ITエンジニア

とあるプロジェクトで人員が不足しており、追加要員として未経験の若手ITエンジニアがアサインされました。筆者は、彼がプロジェクトに参加する際に詳細な説明や指導に多くの時間を費やすかもしれない不安になりました。

しかし、驚くべきことに、彼は非常に積極的に学ぶ情熱を備えていました。プロジェクトに参加した直後から、日々の業務と自己学習から多くのことを学び成長していきました。また、他のメンバーの手伝いを惜しまず行い、チームワークを高めることも同時に行いました。わずか1年後には、チームで中心的な役割を果たすようになりました。

「悪い裏切り」には以下のようなものがあります。

プロフェッショナルを名乗るPL

技術もマネジメント経験も豊富と言われる人物がPLとしてアサインされました。筆者たちが対応している炎上気味だった案件に救世主として登場し、多くある課題を次々に解決してくれることを期待しました。

しかし、彼は技術的な理解が遅く、顧客の無理難題をそのまま持ち帰り、解決策を提供できない伝書鳩のような存在となり、炎上を加速させました。

長い経験を持つITエンジニア

とあるプロジェクトで、設計要員として長い経験を持つITエンジニアがアサインされました。メンバーの中でも一番長く豊富な経験があったため、筆者たちは喜んで彼を迎えることになりました。

しかし、彼の設計書は難解な記述が多く、他の設計者から見ても整合性に欠けるものでした。最終的には、他の設計者が再設計を行いました。設計経験も豊富と聞いていましたが、筆者たちが期待した設計経験と齟齬があったことは間違いません。

プロジェクトへの貢献度は、実際に協力し合ってみないと把握するのは難しいものです。技術的な観点だけでなく、人間関係やチームの相性も関係してきます。経験年数は重要な指標の1つですが、それだけを根拠にして一喜一憂してはいけません。

では、私たちはどうすればいいのでしょうか？筆者が持ち得る答えは、早い段階から実際の協力を始めることです。私たちが経験年数を評価する側であろうと、評価を受ける側であろうと、実際に協力しはじめて、早期に相互理解を深め、お互いに改善する時間を確保できるはずです。

7.8 違和感を大切にする

違和感は、プロジェクトのあらゆる場面で現れます。以下は筆者が違和感を覚えることのある場面の一例です。

- 提案書や設計書を見たり、作っているとき
- 実装しているとき
- 他者のレビューを聞いているとき

これらのとき、明確には言葉にできないけれども「ん？なんかおかしくないか？」という漠然とした違和感を覚えることがあります。この違和感は、他者へ対してのみではなく自身に対しても現れます。

この違和感を軽視しないことが大切です。違和感が言葉に表せないほど微かなものであっても、それは過去の経験や知識、あるいは考察に基づいて生まれたものです。これらの違和感は、ときに問題の兆候を発見する手がかりにもなります。

違和感を覚えた際に行う3つのアプローチを紹介しましょう。

違和感を突き止める

「何に違和感があるのか」を突き止めるため調査を行います。違和感を覚えた資料や実装の詳細な解説や仕様調査、動かせる環境がある場合は検証を行い調べます。筆者はこの方法を一番優先としていますが、いくつかの課題もあります。

1. 本来優先すべきタスクがあるため、使える時間が限られる
2. 理由を明確にできないため、残業申請の承認を得ることが難しい
3. 周囲に無駄な時間を過ごしていると認識される

取れる手段は3つです。良い手段ではありませんが違和感を放置せずにすみます。

1. 本来のタスクを後回しにして、時間を割り当てる
2. 残業を認めもらえるように根気強く依頼する
3. 早朝や定時後の業務外の時間を使ってでも突き止める

とりあえず聞いてみる

違和感の先が自分自身でも他者の場合でも、他者に共有してみることです。

自身への違和感であれば、仲の良く話しやすいメンバーに「ここって、なんかおかしくない？」と聞いてみます。ポイントは「仲の良いメンバー」です。仲の良くない人は漠然とした質問を嫌う傾向があるためです。

他者への違和感であれば、メンバーや顧客に漠然としていることを前提に「この辺りになんか違和感があるんですが・・・皆さんないですか？」のように聞いてみます。このアプローチは非常に勇気のいることです。漠然とした問い合わせは相手に伝わりにくいばかりでなく、問い合わせた私たちの評価を下げる可能性もあります。また、お互いに一定の信頼関係がなければできないことです。しかし、顧客側のファシリティや顧客管理のシステムに関する部分には、違和感を覚えることが少なからずあります。「とりあえず聞いてみる」ことにより、他者が新たな視点を提供してくれる可能性があります。

忘れないようにする

違和感を突き止める時間も確保できず、とりあえず聞いてみることも難しい場合は、「忘れない」ようにします。忘れないようにする方法はさまざまですが、以下のようなものがあります。

- 資料の目につくところにわかりやすく注釈を入れる
- アノテーションコメントとして記録する
- 個人用のタスクとして管理しておく

普段から目にとまりやすいところに忘れないように残しておき、後日時間を確保できた際に振り返られるようにしておきます。

違和感は放置していても問題のないケースもあります。周囲からすれば後回しでいいと一蹴されることもあります。しかし、違和感が問題に発展する可能性があることを忘れてはいけません。

筆者の経験上、問題が顕在化した際に解決に費やす時間は、違和感を突き止めるために費やした時間を上回ります。この解決のための時間は、私たちも含め他のメンバーたちが費やすかもしれない時間です。

7.9 横目と仰望

ITエンジニアに限らず社会人をしていると、俯瞰する（高所から下方を見渡す）ことが大切と言われることがあります。俯瞰することは、思いもしなかった問題の防止や改善につながる機会を与えてくれます。

しかし、私たちが俯瞰すべきことは、自身の責務範囲に止めておき、プロジェクト全体を俯瞰することは避けるべきです。プロジェクト全体を俯瞰するために必要な「高所へ羽ばたく翼」や「高所から見渡す目」を手にいれるためには、多くの権限と時間を必要とします。また、技術的な知見のみならず、全チームの状況、都度遷移する顧客との関係性、経営的な思惑なども必要になります。

プロジェクト全体の俯瞰は、PMやPLたちに任せておきましょう。彼らは、大空から私たちを見渡してくれています。

私たちには、プロジェクト全体の俯瞰は必要なく、自身の責務範囲を俯瞰する視点が必要ですが、他にも必要な視点があります。それは「横目」と「仰望（ぎょうぼう）」です。

横目の視点

横目とは「横をちらつと見る」ことです。「横」には2つあります。

1つ目は、責務範囲を同じくするチーム内です。同じチームの人たちは、私たちと共に通ずる技術を使い、似た仕事を行っています。私たちと同じように、日々悩みを抱えています。このようなチーム内でちらつと見合ったり、雑談でも話してみたりすることは、悩みの共有と解決の糸口を発見できる機会になります。

2つ目は、責務範囲に隣接しているチームです。隣接しているチームは、技術上の連携を密にしているチームです。私たちの責務範囲によって、隣接するチームは異なります。

- サーバチームであればネットワークチーム
- データベースチームであれば業務チームかバックエンドチーム
- バックエンドチームであればフロントエンドチーム

隣接チーム間は、同じチーム内以上に認識のずれが大きく、問題の発生しやすい部分です。お互いのチームを意識することは、大きな問題を未然に防ぐ機会になります。

PMやPLたちは、私たちを高所から俯瞰する目を持っていますが、木の密集する森の中を透視する目は持っていない。森の中を見渡すのは、森と同じ視点にいる私たちの方が容易に行えるはずです。

仰望の視点

仰望（ぎょうぼう）とは「顔を上に向けて仰ぎ見る」ことです。私たちは、高所を羽ばたくPMやPLたちを仰望しておく必要があります。

彼らは俯瞰するために常に飛び回っていますが、しばしば私たちの視界から消えたり、私たちを見る目を忘れることがあります。顧客からの呼び出しで急に外出したり、私たちの課題が忘れられていたなどは、多くの人が経験していることでしょう。

彼らに助けを求めるとき、私たちを見る余裕があるのか意識しなければなりません。彼らが、他のことを見ることに集中している際に助けを求めれば、彼らをイライラさせ、私たちも残念な気持ちになります。

私たちがプロジェクトで必要な視点は3つです。

1. 自身の責務範囲を俯瞰する視点
2. 同チームと隣接チームを横目する視点
3. PMやPLたちを仰望する視点

プロジェクト全体の俯瞰は必要ありません。私たちに関係するところだけを見ておきましょう。

7.10 面倒なことは、みんな面倒

面倒なことは、ほとんどの人にとって嫌なものです。しかし、「面倒くさがり」はプログラマに向いているそうです^{†1}。プログラマによれば、周りの優秀なITエンジニアを見ていると、面倒くさがりが多いように感じます。彼らは、便利な技術やツールを探し、インフラを構築し、コードを書き、多くの処理を自動化し、面倒な作業をコンピュータに任せます。驚くことに、これらに彼らは笑顔で取り組み、プライベートの時間を割いてまで取り組むこともあります。

しかし、仕事の中には、彼らも取り扱からない面倒なことがあります。

- デスクに配置された電話の対応
- 荷物の受取や発送
- 古い設計書や手順書のドキュメント更新
- 新しいメンバーのPCの手配やセットアップ
- 会議調整やデータセンターの入館申請
- その他の評価基準とならないような雑務

これらは、ITを使っても解決することが難しい面倒なことです。しかし、無視もできません。誰かが電話でなければなりませんし、古いドキュメントをそのままにしておけば問題が生じるかもしれません。新しいメンバーに環境を与えなければ仕事はできません。

このような「みんなが面倒だと感じつつもやらなければならないこと」に私たちは取り掛かるべきです。技術的にチームに貢献することが難しい場合でも、これらの面倒な仕事に取り組むことで、貢献することができます。また、思わぬ副産物を得ることができるかもしれません。

人間関係の形成

面倒なことというものは、ほとんどの人が面倒であることを認識しています。想像してみてください。私たちが面倒だと感じるタスクを次々に処理してくれる人が近くにいるとしたら、おそらく感謝の言葉を送るでしょう。

新しい仕事

ITエンジニアたちは、常に仕事に追われています。できるだけ多くの仕事を他の人に任せたいと考えています。そのとき、以下のような2人がいるとします。

1. 何をやっているかわからない人
2. 率先して雑務を処理している人

どちらにタスクを依頼したいですか？おそらく後者でしょう。こうして依頼された仕事は、少なからず技術的な要素を含み、私たちの成長につながることでしょう。

これらを「古い考え方」や「非効率な考え方」と感じる人もいるかもしれません。筆者の経験を共有しておきましょう。

筆者が、ITエンジニアになったばかりのころの主な仕事は、電話当番とオフィスの掃除でした。当時の筆者は、ITの技術や仕事がどんなものかもわかつておらず、タスクが割り振られることもありませんでした。

当時の先輩たちは忙しく、デスクにいなることがよくありました。自然と電話対応の機会が増えました。電話当番だけでは暇すぎたので、オフィスの掃除も行っていました。当時のオフィスはお世辞にも綺麗とは言えなかつたため、掃除できるところを見つけることが容易でした。

初めて技術的な仕事を行ったのは、筆者が電話当番と掃除をしていること知っていた先輩から依頼されたタスクでした。

そして、筆者が新人などと働く際に仕事をお願いしたいと思うのも、待ちぼうけではなく、自分のできることを率先して行う人たちです。

空いた時間に、みんなが面倒だと感じる仕事を探ししましょう。それを見つけたら、積極的に取り組みましょう！

†1 プログラミング言語Perlの開発者ラリー・ウォール氏がプログラマの三大美徳を提倡して広まったと思われる考え方です。三代美徳とは「怠惰」「短気」「傲慢」のことであり、プログラマに必要とされる「効率や再利用性の重視」「処理速度の追求」「品質にかける自尊心」をなぞらえたものです。

Wikipedia. プログラマ. 2023年9月8日 05:51

<https://ja.wikipedia.org/wiki/プログラマ#プログラマの三大美徳>

第

VIII

章

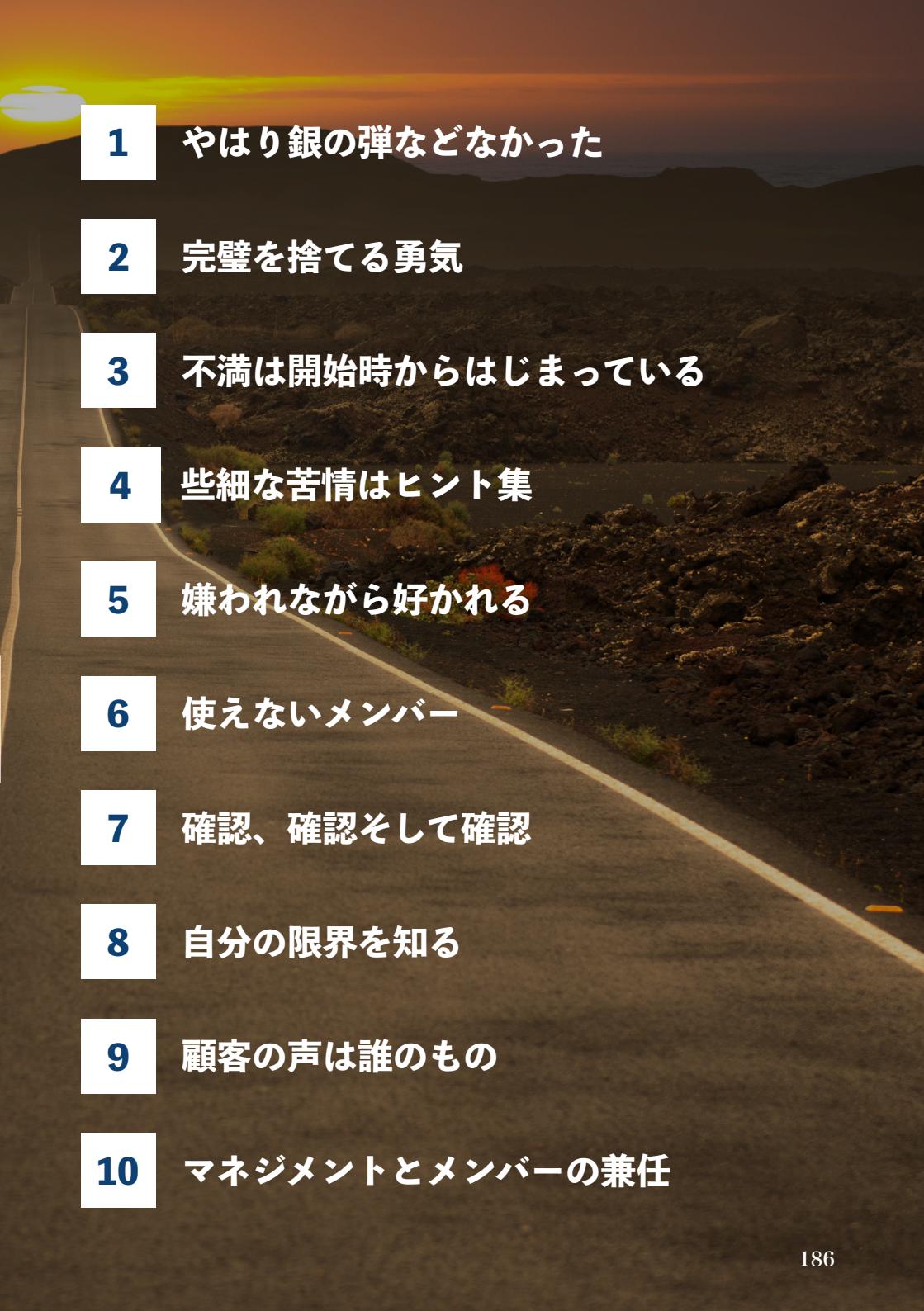
無能のためのマネジメント

本章は、無能なITエンジニアとして生きていくためのプロジェクトマネジメントについての教訓になります。

本章のマネジメントには、プロジェクトマネージャーと
プロジェクトリーダーが含まれます。

そして、本章に限り「私たち」はメンバーではなく
マネジメントの役割を指します。

私たちがマネジメントを行う際は、
ITエンジニアとは異なる視点が必要です。



1 やはり銀の弾などなかった

2 完璧を捨てる勇気

3 不満は開始時からはじまっている

4 些細な苦情はヒント集

5 嫌われながら好かれる

6 使えないメンバー

7 確認、確認そして確認

8 自分の限界を知る

9 顧客の声は誰のもの

10 マネジメントとメンバーの兼任

8.1 やはり銀の弾などなかった

「銀の弾などない」は、フレデリック・P・ブルックス、Jr氏の1986年に著した論文「No Silver Bullet - Essence and Accident in Software Engineering」により広く知られるようになり、皆さんも耳にしたことがあるかもしれません。筆者は、同氏の「人月の神話^{†1}」で知りました。「人月の神話」の「銀の弾などない」理由の多くが、ソフトウェアを主軸とした技術的な要因について述べられています^{†2}。しかし、筆者は人的な要因により、プロジェクトマネジメントにおいても「銀の弾などなかった」ことを実感しました。

現代では、プロジェクトマネジメントに関する多くの書籍や情報があり、学ぶことは比較的容易になりました。しかし、これらの書籍を読んでも「銀の弾」を見つけることができませんでした。

メンバーの人的要因に関して「銀の弾がなかった」経験を紹介しましょう。

性格によるもの

プロジェクトの進行中に、仕様変更の依頼がありました。技術的な問題はありませんでしたが、納期を守るためには、何度かの徹夜と休日出勤が必要になることは容易に想像ができました。この情報を聞いたあるメンバーは、鬼の形相になり怒りに任せて机を殴り始めました。筆者は目を合わせないようにしたことを覚えています。しばらく時間が経つてから、冷静になった彼と話をすすることができました。

これは幸運に恵まれていました。もし、彼がその後もずっと怒り続けていれば、どうすればよいのでしょうか？今でもその解決策は見つかっていません。

年齢によるもの

若い頃にマネジメントをしていた際、見落としや技術的な懸念点があると、経験豊富な年上のメンバーたちが次々と指摘をしてくれました。年齢を重ね若いメンバーが増えた現在は、そのような機会が少なくなりました。彼らの中には「自分より経験のある年上が言っているのだから間違いない。言われた通りにしよう。」と考える人たちがいます。そのような人がいないか注視しなければなりません。

また、私たちも年齢を重ね他者の見解を取り入れようとせず、自分の意見を論拠なく突

き通そうとすることがあります。ときには感情的になることもあります、冷静さを欠いてはいけません。

私たちは年齢とともに多くの経験をしていますが、メンバーの方が専門性が高く詳しいこともあります。私たちには彼らの助けが必要です。

時代変化によるもの

近年では、リモートワークの拡大や残業規制の強化により、ITエンジニアの働き方が大きく変わりました。コミュニケーション方法も変化し、チャットやWeb会議が増加しました。私たちは、メンバーたちと密に連携する方法論をより考える必要があります。以前のように、過度な残業や休日出勤で乗り切ることも不可能です。当日に残業を依頼すれば、舌打ちが返ってくるかもしれません。

このようなメンバーの人的要因は、大きな組織や大規模なプロジェクトであれば、簡単に対応できるケースもあります。予算があり、優秀な人材獲得や、要員の追加・変更を柔軟に行える可能性があるからです。

小さな組織や小規模なプロジェクトでは、人的要因はプロジェクトに致命的なダメージをもたらすことがあります。メンバーが不満を抱き、プロジェクトを去ったり退職した場合、早々に新たな人材を獲得することは難しいでしょう。その場合、既存のメンバーか私たち自身が穴を埋めなければなりません。

フレデリック氏は「銀の弾などない」の解決策の要点について以下のように述べています。

複雑性こそ私たちの取り組んでいる仕事であり、
複雑性が私たちを制約しているものだ。^{†3}

プロジェクトマネジメントの人的要因による問題解決も複雑であり「銀の弾」はありませんが、これらに取り組むことも私たちの仕事であることを忘れてはいけません。

†1 筆者を持っているものは少し古いものです。現在は、カバーが一新された「人月の神話」が発売されています

フレデリック・P・ブルックス, Jr. 人月の神話[新装版] 狼人間を撃つ銀の弾はない. 株式会社ピアソン・エデュケーション. 2002年11月

†2 誤解が生まれそうなため、補足しておきます。技術的な要因が多いとしたのは「人月の神話」の以下の章に限定した解釈です。「人月の神話」は、人に焦点を当てた内容についても多くの記述されています。

第16章 銀の弾などない – ソフトウェアエンジニアリングの本質と偶有的事項

第17章 「銀の弾などない」再発射

†3 「人月の神話」. 213ページ

8.2 完璧を捨てる勇気

私たちがマネジメントの役割を果たす際、多くのプレッシャーに立ち向かう必要があります。プロジェクトのゴールが明確で進むべき方向が確定していても、その道のりには予想できない数々のリスクが潜んでいます。私たちはしばしば「失敗してはいけないプレッシャー」と「未知のリスク」からくる恐れに取り憑かれ、完璧を求めてしまうことがあります。しかし、その完璧を捨てる勇気が必要になります。

すべてはわからない

完璧を求めすぎると、メンバーや私たち自身に過度な負荷をかけることがあります。特に新しい技術が導入される場合、この問題が顕著です。新たな技術に関する問題が、設計や実装の段階で登場すると、プロジェクトの要望を満たしている場合でも、他に最適な方法があるのか、本当にそれが最適解なのかなどを徹底的に検証しようとすることがあります。そして、仕様が変更された際に備えて、適切な対策を講じるために、さまざまなシナリオやパターンについて分析しようとします（仕様変更の連絡はきていないのに！）。

これらの徹底的な調査や確認作業は、リスク管理に必要なことですが、然るべき予算や期間があつてできることです。私たちからメンバーに対して、あらゆるリスクに対処するように強要をすべきではありません。また、私たち自身も必要以上に技術的な詳細を理解しようとすべきではありません。

多くの場合、メンバーは専門的な知識を持っており、これらの技術や詳細に関して深く考えています。

支援と実行

メンバーたちが問題に直面した場合、私たちに助けを求めてくることがあります。もし、その問題が技術的な内容の場合、私たちは解決策の実行をすべきではありません。エラーログを解析し、コードを読み、解決策を見つけ出し、問題の解決を自ら行うことは避けるべきです。

その代わりに支援をしましょう。参考になる情報を提供したり、問題の解決に向けた調査方法などを伝えます。もしくは、その問題の領域に詳しいメンバーを特定し、支援の調整を行います。私たちが、解決の実行をしてしまうと、私たちの時間を費やすばかりではなく、メンバーの成長の機会を奪います。

支援は私たちの仕事ですが、実行はメンバーの仕事です。

時間は常に進んでいる

トラブルが発生した場合、対応方法に複数の選択肢が存在することがあります。その際、他にも完璧な対応方法があるのではないかと探し始めることができます。調査や確認、判断材料の収集には多くの時間がかかります。その間にも、プロジェクトの残り時間は減少し、メンバーの本来の進捗が遅れることになります。

こうした状況では、その時点で完璧を求めるのではなく、すでに存在している対応方法を選択するか、仮の対応を実施し後日に恒久的な対応を行うなどの判断が必要です。

判断を保留する時間を長引かせ、私たち自身がプロジェクト進行の妨げにならないようにならぬよう。

失敗やリスクを恐れることは、大切なことです。恐れは私たちに、さまざまな考え方を与えてくれます。しかし、その恐れを克服するために、完璧を求めてはいけません。完璧の追求が、新たなリスクと失敗の要因になりかねないからです。

では、どのようにすれば完璧を求めずにリスクを管理し、恐れを克服できるのでしょうか？筆者の考えは「致命的な誤った選択を避ける」ことです。完璧な選択肢を探すよりも、致命的な誤った選択を避ける方がより現実的です。いくつかの小さな誤りをするかもしれません、致命的な誤りでなければ、プロジェクトの中で対処することができるでしょう。

完璧なバグのないシステムを作るプロジェクトを想像してください。すでにそのプロジェクトは失敗しています。

8.3 不満は開始時からはじまっている

メンバーは、PMやPLの役割である私たちに多くの不満を抱いています。理由や解決方法は多岐にわたるため、すべてを述べることは出来ませんが、メンバーの不満はプロジェクト開始時からすでに始まっています。本節では、内部キックオフミーティング^{†1}の際に、不満を減らすために気を付けることを紹介します。

資料の事前共有

ミーティングに使用するキックオフ資料やプロジェクト計画書に加えて、入手している資料を可能な限り共有しましょう。面倒だからとファイルサーバにあるフォルダを丸つとzipファイルにしてはいけません。共有された側は混乱して見ることをやめてしまします。

「キックオフで説明する資料」「重要資料」「参考資料」などに分類し、共有された側が、どの順番で見ればいいのか把握できるリスト一覧を作成しておきましょう。それらをフォルダなどで整理したものをzipファイルにして共有しましょう^{†2}。参加者はミーティングまでに概要を把握することで、安心して私たちの説明を聞いてくれるはずです。

自己紹介と他己紹介

ミーティングの冒頭で、いきなり資料の説明をしてはいけません。多くの異なる組織や部署からの参加者がいるはずです。私たちはすでに全体の体制を把握していますが、彼らは誰が参加し、それぞれの役割は何か不安に思っています。

資料の途中で説明するかもしれません、最初にしておけば急遽参加した人たちも安心できるでしょう。私たちの紹介と、各人員がどのような役割で参加しているのか紹介しましょう。それぞれの代表者に挨拶してもらうのもいいかもしれません。おそらく5分もかからないはずです。

顧客情報の共有

参加者は、資料から顧客の企業名・業界・組織規模などは把握できていますが、顧客側の担当者の性格や特徴については知りません。例えば、その担当者が社交的なのか、技術に精通しているのか、怒りっぽい人なのかななど文章で表すことが難しい部分もあります。

このような資料からは伝わらない顧客情報を共有しましょう。私たちが顧客担当者にまだ会っていない場合でも、営業担当者などから情報を収集し、その情報を共有することは

できます。彼らが顧客と会議をするとき、アプローチの参考になるはずです。

スケジュールの確認

参加者たちは多忙で、常にスケジュールに気を使っています。私たちは、スケジュールが完成している場合でも「スケジュールを確認しておいてください」で終わらせず、マイルストーン程度は一緒に確認しておきましょう。変更の可能性がある期間や重要となる期間も伝えておけば、彼らはその付近のスケジュールに対して意識して行動することができます。

テンプレートとルールの共有

おそらく、私たちがまず依頼するのは、設計の準備の着手です。彼らは私たちが依頼をせずとも設計書の叩き台を早く作りたくてウズウズしています。

必要となるドキュメントのテンプレートを共有するか、独自フォーマットで良いのであれば、その旨を明確に伝えましょう。また、用語集は出来ているところまでを、同じタイミングで共有しておくと良いでしょう。ドキュメントの精度が判断できる参考資料が用意できればさらに良いです。

また、連絡体制やコミュニケーション方法、プロジェクトで使うサービスやツールの運用ルールについて共有しておきましょう。プロジェクト計画書に含まれている場合でも、体制と方法を明確に説明し共有できれば、この後の混乱を避けることができます。

懸念事項の確認

説明の終わったときや区切りのいいところで、参加者に懸念事項がないか確認します。即座に回答できなくとも、彼らの多くの不安や疑念を取り除くことができます。また、多くの質問に対応できるよう、会議時間は余力を持つておく必要があります。「伝えるだけ伝えて、時間がなくなりました」という状況は最悪のパターンです。

これからもまだ多くの不満がでてくるでしょうが、プロジェクト開始時の不満は少なくなっているはずです。

†1 キックオフミーティングは、プロジェクト開始時に関係者が集まり、プロジェクトが目指すべきゴールや目的、情報を共有するためのミーティングです。通常、顧客も関係者に含まれますが、本節では顧客以外のメンバー・ビジネスパートナーとなる人たちが集まるミーティングを内部キックオフミーティングと表記しています。

†2 近年では、zipファイルではなくクラウドストレージなどのサービスを使って共有するプロジェクトも当たり前になってきました。その場合でも、分類が必要なことに変わりはありません。見る側が把握しやすいように整理しましょう。

8.4 些細な苦情はヒント集

苦情は面倒なものです。苦情を受け入れ、解決するために時間を割かなければいけません。その間に私たちの貴重な時間が奪われていきます。しかし、苦情にはその時点での不満や問題が内包されていると捉えることができます。

こうした苦情を聴くことで、問題の根本的な解決に繋がることもあります。課題管理に記録されたり、顧客から大きな声で発信された苦情は問題を未然に防ぐ機会となる一方で、目立たない些細な苦情も同様に問題を回避する手助けになることがあります。

○○ツール使いたいのになあ

雑談の中から発見できた些細な苦情の事例です。

とあるプロジェクトでは、開発環境を設定済みのPC端末が、各メンバーに配布されました。プロジェクトが始まってしばらく経った後、メンバーたちの間で「○○ツールが使えれば便利なのになあ」という声が耳に入ってきました。なぜそのツールを使わないのか質問したところ、いくつかの回答を得ました。

- 類似ツールは開発環境に導入されており文句をいわれるかもしれない
- 新しいツールを導入するときの申請などが必要なのかわからない
- 申請が必要だったとき、すごく手間になる（かもしれない）

筆者は気になり、新しいツールを導入するときに、どのような申請が必要になるか確認しました。理由があれば簡単な申請書を記入し、管理部署へ提出すれば複雑な審査などもなく認められることがわかりました。メンバーに他に必要なツールはないか確認し、いくつかのアプリケーションリストを入手しました。いずれも無償で利用できるツールだったため、チームを代表してそれらの申請をしたところ、数日後には使用できるようになりました。

この申請は知っている人にとっては当たり前のことがでしたが、筆者を含めチーム内で申請方法を知っている人はいませんでした。他のメンバーたちは、面倒なためそれらを調べようとはしませんでした。

開発の効率にどれほど貢献できたかは定かではありませんが、メンバーたちのフラストレーションを下げられたことは確信しています。

別にその案じゃなくてもよくな?

顧客の小さな発言から発見できた些細な苦情の事例です。

顧客が利用しているシステムの作業計画レビューを行いました。筆者たちの案としては、業務影響の可能性がある作業だったため、夜間を予定していました。反対意見が出ることもなく、レビューは進行していましたが、最後のまとめを説明している際に、顧客側担当者の一人が遅れて参加してきました。

まとめの説明を聞いた彼は「なんで夜間に作業するのか?」と別の担当者に、ぼそりと聞いていました。明確な発言ではありませんでしたが、念の為に筆者は再度説明をしました。そこで、彼と話してわかったのは「事前に通達すれば業務を一時停止して作業ができる」とのことでした。

彼からすると事前通達の調整よりも、夜間立ち会いする調整の方が面倒だったのでしょうか。筆者たちとしても、日中作業の方が問題発生時の対応する体制を整えやすかったため、案を日中作業に変更し対応しました。

些細な苦情を無視したり、面倒くさいと感じることがありますが、それらは問題解決の貴重なヒントとなり得ます。

同時に、些細な苦情には感情的な要素などの雑音も紛れていることに気をつけなければいけません。

例えば、ツールに関する苦情の場合「高額なツールを試してみたい」という個人的な興味による苦情であれば、筆者は運用ルールを調べることはしなかったかもしれません。作業時間に関する苦情の場合「絶対に業務に影響を出すな」という苦情であれば、作業時間を相談する前に「絶対は存在しない」ことを説明しなければいけませんでした。

些細な苦情を見つけましたか?聞かぬふりや面倒と思う前に、耳を傾けてみましょう。それは私たちを助けてくれるヒントかもしません。

8.5 嫌われながら好かれる

筆者は、人から嫌われることが「イヤ」です。今後一切関与しない人でもない限り、良好な関係を築きたいと考えています。特にメンバーから嫌われるの「すごくイヤ」です。プロジェクトの成功に向けて、家族と同じくそれ以上に日々の時間を過ごさなければならぬからです。楽しく笑いながら一緒に仕事をしたいと考えています。

それでも、仲良くしようとして馴れ合いに甘んじて「プロジェクトが失敗」することは、これらの中でも「一番イヤ[†]」です。

私たちは、プロジェクトを失敗させないようにすればするほど、メンバーから嫌われる機会が増えています。

以下は、嫌われそうな依頼の一例です。

- 仕様変更による設計、実装の変更依頼
- 急遽発生した想定外のタスク依頼
- 作業工数の不足によるオーバーワークの依頼
- 上記に関連する残業や休日出勤の依頼

これらの依頼は、嫌われてしまうかもしれません、プロジェクトを失敗させないために必要になるかもしれないことです。私たちが何か依頼するときに、いくつかの気を付けるべきことがあります。

理由を明確に伝える

メンバーが嫌がることを依頼するとき「顧客が言ったから」や「全員が頑張らなければならないから」などを理由にしてはいけません。依頼の背景には、顧客側の運用やビジネスの変化や私たちの想定不足などの理由があるはずです。明確な理由なく想定外のタスクが生まれることはあります。

これらの理由を冷静に説明し、メンバーに伝えなければいけません。メンバーの不満は解消しないかもしれません、私たちが「伝書鳩」と思われることは、なくなるはずです。

必ず「ありがとう」と「ごめんなさい」

メンバーが嫌がることを依頼するときは「ありがとう」を伝えましょう。彼らは、プロジェクト成功のために想定してなかつた嫌なことに立ち向かおうとしています。最大限の感謝を送らなければなりません。

また、同時に「ごめんなさい」も伝えましょう。誰もが嫌がる突発的な依頼の多くは、予測ができない技術的な不具合を除けば、私たちの「交渉の失敗」や「考慮不足」から発生しています。私たちの力が及ばなかつたことや過失を認めましょう^{†2}。

逃げない

タスクを依頼した後も、私たちの役目が終わって早く帰ろうなどとは考えなてはいけません。無理に残業をする必要はありませんが、メンバーたちは依頼された新しいタスクに関連する確認事項や、それに伴う新たな問題について質問したり相談したいことがあるでしょう。私たちも一緒に向かい合わなければいけません。

また、依頼を受けたメンバーは残業時間が多くなり、上司から叱責されているかもしれません。顧客からは技術や手法の内容について、質問攻めに遭うかもしれません。私たちが依頼したために窮地に立たされています。私たちは窮地に立たされるメンバーを常に支え、守る役割を果たさなければなりません。

どんなプロジェクトにも3つの問題があり、それらの問題は「1に人、2に人、3に人」と言われます^{†3}。私たちは、プロジェクトの仕様上の問題や技術的な問題の前に、人の問題に立ち向かわなければいけません。

私たちがメンバーに向かい合っている限り、PMやPLとして嫌われることがあっても、「人」として嫌われることはないでしょう。そして「人」として嫌われない限り、メンバーたちはプロジェクトの成功に向けて歩み続けてくれるはずです。

^{†1} これらを含めてもっと嫌なことは、私たち自身も含めて、プロジェクトの関係者で心が病んでしまったり、仕事に絶望する人が出てしまうことです。

^{†2} 謝罪は、ときに悪い方向に向かうことがあります。筆者の妻は、子どもによく言っていることがあります。「謝ればなんでも許してもらえると思っているでしょ！！」。過失を認めることは必要ですが、免罪符ではありません。

^{†3} エドワード・ヨーダン、デスマーチ 第2版 ソフトウェア開発プロジェクトはなぜ混乱するのか、日経BP社、2006年5月、95ページ

8.6 使えないメンバー

プロジェクトを推進していると「使えないメンバー」が現れることがあります。私たちが発見するかもしれませんし、他のメンバーの苦情から発見するかもしれません。使えないメンバーとされる理由は、いくつかあります。

- 遅刻や早退、提出物の期限を過ぎることが多い（多くの場合、報連相^{†1}はない）
- 他のメンバーと比較して成果や稼働が極端に低い
- タスクが終わっていないにも関わらず定時帰宅や休んだりする

このような場合、私たちはいくつかの対応を試みます（表8-6-1）。

対応	内容
対話/協力	原因を特定し、改善に向けた協力関係を築くため、本人や他のメンバーと対話する。
注意喚起	改善を促すために、本人に対して注意を喚起する。これは、対話と異なり権威行使することが含まれ、関係や状況を悪化させる可能性があることに留意する必要がある。
退場勧告	プロジェクトからの退場を本人に促す。これは本人に対して行う場合だけでなく、本人の上司や組織関係者に対してや、それら複数に及ぶこともある。客観的な正当な理由が必要となる。
放置	どの対応も行わず、問題をそのまま放置し、これまで通りとする。小規模プロジェクトや小さな組織など、人員の補充が難しい場合にこのような対応となることがある。他のメンバーのフラストレーションをため続けることがあるため留意する必要がある。

表8-6-1 使えないメンバーへの対応

どの対応が正しいかは状況によって異なりますが、慎重に検討しなければなりません。使えないメンバーが現れた際、私たちや他のメンバーは感情的になり、多くのことを見失いがちです。対応に動く前に検討すべきことには、以下のようなことがあります。

病気によるもの

うつ病・睡眠障害・ADHDなど、病気や障害が起因している可能性があります。気をつけなければいけないのは、本人が自覚していない場合もあることです。また、吃音症などで、周囲が不当に評価をしている可能性もあります。本人と他のメンバーが病気や障害への理解が必要になります^{†2}。

家庭事情などプライベートによるもの

家族の介護や長期入院の対応、もしくは失恋など、プライベートな問題が業務へ影響している可能性があります。こうした問題は、心身を疲弊させますが、即時的な解決策は、ほとんどありません。私たちができることは、仕事の負荷が高くなりすぎないようにするか、休みを取得しやすくする環境を整える程度です。

本人のモチベーションによるもの

「やりがいを感じない」や「収入など待遇面の不満」などにより、モチベーションが低下していることがあります。「達成感のあるタスクを依頼する」や「必要としていることを伝える」などできますが、私たちだけで対応できることは、限られています。

チーム内の人間関係によるもの

メンバーからの嫌がらせや、ときには悪質ないじめによる可能性があります。チーム内に発言力の大きな人や一定の実力があり評価されている人がいる場合に、発生しがちになります。彼らに周囲のメンバーが同調してしまい、嫌がらせやいじめに発展することもあります。問題の原因を明らかにし、解決に取り組む必要があります。

使えないメンバーに対応する際、無力感を感じるはずです。私たちが、どれほど慎重になったとしても、本人や他のメンバーと信頼関係がなければ、原因に辿り着くことすらできません。彼らは、本心を隠し表向きの会話をすることがあるからです。

また、多くのプロジェクトは有期であり、数ヶ月～数年程度でチームは解散します。人材育成のような長期的戦略は適用できず、そのような人たちを救うことはプロジェクトの目的でもありません。プロジェクトの中で対応できることは、ほんのわずかです。

†1 報連相（ほうれんそう）。報告・連絡・相談の3つの言葉を合わせたビジネス用語です。

†2 これには、大きなジレンマがあります。病気や障害を把握するには、本人との信頼関係がなければ突き止められないことです。そして、そのような信頼関係があれば、問題となる前に防ぐことができた可能性が高いことです。

また、病気や障害の理解を得るために、少なくともチーム内で知つてもらう必要がありますが、多くの場合、周囲に知られることを嫌がります。

8.7 確認、確認、そして確認

プロジェクトマネジメントは、スコープ・スケジュール・コミュニケーションなどさまざまなものを管理（Management）します^{†1}。これらの管理は大切ですが、私たちがより大切にすべきことは「確認」です。

確認を継続的に行うことで、問題を早期に発見し、回避できる機会を増やすことができます。確認は、毎週などの定例会議で行うと考える人もいますが、私たちが日常的に確認すべきことを紹介しましょう。

課題の確認

課題管理表は毎日確認しましょう。

オフィスソフトやクラウドサービスなどを使用して課題を管理しているはずです。顧客が関与しないような、内部向けの課題に関してはメンバーが起票することも多いでしょう。これらの課題を毎日確認することで、彼らが困っていることを、いち早く知ることができます。また、私たちが忘れてしまっている課題を思い出す機会にもなります。

注意が必要なのは、メンバーの報告する課題は、彼ら視点に偏っていることが多い点です。プロジェクト全体や関係者に与える影響を考慮していなかったり、技術以外の解決策がある場合も、技術を使ってどうにかしようと必死に悩んでいることもあります。

進捗の確認

進捗をメンバーたちへ直接尋ねることで確認しましょう。

スケジュールに記載された進捗率を鵜呑みにしてはいけません。メンバーたちの中には、タスクを着手した瞬間に進捗率を50%にしたり、期限が近づくと90%にする人が少なからずいます。それらの多くは、細かなエラー処理やドキュメントの更新などの関連する作業は進捗率に含まれていません。

しかし、日常的に定型文で「本当の進捗率はどれくらいですか？」などと尋ねてはいけません。そんなことを日々確認すれば、彼らはフラストレーションを溜め、私たちと会話することや顔を合わせることを嫌に感じるでしょう。

筆者は、オフィスをブラブラしつつ、メンバーに会うと「進捗どう？大変？」などと立ち話をするようにしています。これにより、良くも悪くも問題を発見できことがあります。残念なことにリモートワークが増え、この方法を使える機会は減少しました。リモ

一トワーク向けの方法として、チャットで「あー！（自分の）進捗が悪い！そっちはどう？」などと聞くようにしています。筆者の進捗が悪くないことはほとんどありません。いつでも使える良い聞き方を見つけました。

時間の確認

時間の確認には、2つあります。

1つ目は、メンバーがタスクに要した時間です。タスクに要した時間が想定より超過していれば、スキルに見合わないタスクに取り組んでいる可能性があります。今後のタスクにも多くの時間を必要とするかもしれません。要した時間が想定より少なすぎる場合、そのメンバーにとって簡単すぎたか一部の要件を見落としている（一部未実装のまま放置など）可能性があります。

2つ目は、メンバーの勤務時間です。勤務時間を確認する権限は、私たちに無いかもしれませんのが、メンバーの労働状況をできる限り知ろうとしましょう。「他の案件の調子はどう？」などと聞いてみると良いかもしれません。彼らの中には複数のプロジェクトを抱えたり、勤勉な人たちも多く、異常とも言える長時間労働をしていることがあります。そのような状態を放置しておくことは、彼らにとっても私たちにとっても良いことではありません。

顧客への確認

顧客の中には忘れっぽい人がいます。私たち「に」依頼した内容は詳細に覚えていませんが、私たち「が」依頼した内容は放置されることがあります。毎日確認しては、私たちは嫌がらせをする悪者になってしまうため、ときおり確認しましょう。他の用件と合わせて確認すると良いでしょう。次の定例会議までの間でも顧客とメールで連絡を取り合っているはずです。面倒なので、次の定例会議で確認しますか？結果を得られるのは、その次の次あたりの定例会議になるはずです。

管理不足はプロジェクトの失敗につながりますが、確認不足も同じ以上に失敗につながる可能性があります。私たちが、毎日積極的に「確認」を意識して過ごせば、多くの失敗要因を排除することができるでしょう。

†1 本節では、文脈の分かりやすさから「管理」と直訳して使っていますが、プロジェクトの「マネジメント」を「管理」と捉えるのは誤りです。プロジェクトマネジメントには「プロジェクトを成功させるために必要となる」計画や実行のすべてを含み、管理することが主軸ではないからです。

8.8 自分の限界を知る

私たちは、メンバーたちの心身が限界を超てしまわないように注意を払います。彼らが限界を超てしまい欠員がでると、タスクは未着手か他のメンバーに重くのしかかってきます。代わりの人員が見つかった場合でも、迎え入れる準備やその人が仕事とチームに慣れるまでには相応の時間が必要になり、遅延の要因になります。

それと同時に、私たち自身も限界を超えないように注意を払う必要があります。私たちは、マネジメントという役割の重圧から、私たち自身の限界をしばしば忘れてしまうことがあります。

私たちが限界を超てしまい、プロジェクトから一時的に離脱したり完全に退場してしまうことは、メンバーが退場するよりもプロジェクトに大きな混乱をもたらします。

メンバーたちはプロジェクトを維持するために奔走するか、判断する役割がないため事実上の凍結状態になつたります。新しいPMやPLは、私たちが管理していた資料や状況を再度整理し立て直しに多くの時間を使います。

プロジェクト開始時にサブPMやサブPLを組み込んでおけば、多くの場合は救われます。しかし、サブが事前に存在するプロジェクトは、大規模などの重要なプロジェクトに限られるでしょう。システムと同じように冗長化には大きなコストがかかります。

筆者が「体」と「心」の限界を知るためのサインには、次のようなものがあります。

体の限界のサイン

- （意志に反して）半日以上寝続けてしまう
- 日中に急激な睡魔に襲われる
- 動けないほど関節が痛む^{†1}

心の限界を知る

- 全く疲れなくなる
- 顔や手の一部に痙攣のような震えができる
- 文が読めない（理解するために何度も読み直す）

自分の限界のサインを見つけた場合は、無理にでも休暇をとるなど、回復をはからなければなりません。休むことによる迷惑を気にして回復を後回しにすれば、より多くの迷惑をかけることになります。

そして、もし限界を超えてしまったときの準備も普段から意識しておきましょう。限界ではない場合の不慮の事故や家庭の事情で仕事が行えなくなったときにも、多くの人の助けになります。

更新があるものは常に最新にしておく

更新が必要となる資料などは常に最新に保ち、フォルダ分けやバージョン管理など他者が見てもわかりやすいように整理して管理しましょう。特に、顧客やメンバーと個別に話した内容は、見落とされがちになります。

普段から会話しておく

普段からメンバーと会話をしましょう。これらの会話の中には、メンバーが普段意識しないような、プロジェクトの目的や現在の状況の話が自然と含まれることでしょう。指示だけする関係であれば、メンバーたちは指示が無くなった途端、突っ立ったカカシになる可能性が高いです。少しでも情報があれば、そこから模索してくれるかもしれません。

連絡を取れるようにしておく

体の限界だけの場合^{†2}、連絡が取れる準備をしておきましょう。プロジェクトの内容や資料の場所は、頭の中にはいっています。スマートフォンの充電に気をつければ十分です。新しいPMやPL、メンバーたちの悩む時間を軽減できるはずです。

限界のサインは、限界を超えるラインの一歩手前か限界のライン上に立っている状態です。最後にして最悪の一歩を踏み出さないよう、自分の限界を知るようにしましょう。

†1 筆者は学生の頃にしていた運動により体を色々と壊しています。日常生活に影響はありませんが、疲労が限界に近づくと、壊れた部分の関節が痛み出す傾向があります。

†2 身体的な疲労や怪我などの一時的な問題の場合で、かつ組織の規則の範囲の前提です。また、心が限界なのであれば、一切の連絡を遮断する方が良いでしょう。上司など一部の人にプライベートの電話番号を教えておき、仕事に関する機器の電源は切っておくことを推奨します。

8.9 顧客の声は誰のもの

マネジメントをしている私たちは、常に顧客と対話し続けます。顧客が話すプロジェクトに関する良い話も悪い話も、私たちの耳に届きますが、これらの声は私たち個人に対しではなく、プロジェクト全体に対してのものです。

感謝の声

私たちはしばしば、顧客から「ありがとう」の「感謝の声」を聞きます。

- プロジェクトを完遂したとき
- 顧客の無理難題な要望を実現したとき
- 大きなトラブルを解決したとき

これらは、メンバーが頑張ってくれたからこそ得られた声です。メンバーにも顧客からの感謝の声を届けましょう。1.5倍くらい誇張した表現で伝えてもいいかもしれません。メンバーのチームは解散してしまっていますか？メールでもチャットでもいいので伝えましょう！

メンバーに届く顧客の声は、以下のようなものが多く、感謝の声を聞くことはほとんどありません。

- マニュアルや仕様書にあることを質問てくる声
- 技術的可否を問う怪訝な声
- 不具合や障害に怒り狂った声

感謝の声は、メンバーの満足度を向上させ、次のプロジェクトも頑張ってくれるはずです。そして、私たちからも「ありがとう」の「感謝の声」をメンバーたちに送りましょう。彼ら無くしてはプロジェクトが完遂することはなかったのですから。

苦情の声

私たちは、顧客から「苦情の声」も聞きます。苦情は、2つに大別されます。

1. プロジェクト全体に関する苦情
2. メンバーなどの個人に関する苦情

プロジェクト全体に関する苦情であったなら、それが落ち度であったことを確認し、謝罪しましょう。プロジェクト全体の管理者としての責務です。

しかし、稀にメンバーを名指しした苦情の声を聞くこともあります。謝罪の前に、なぜメンバーの名指しにまで至ったかのか確認しましょう。メンバー個人の非ではなく、私たちの伝達方法や指示に誤りがあったのかもしれません。もしくは、過度な労働や人選の誤りなど体制による問題の場合もあります。

このような個人を名指しした苦情は、顧客側内部で尾ひれが付いて広まる可能性があります。それだけは、避けなければなりません。

では、個人への「苦情の声」は、本人に届けるべきでしょうか？苦情の理由や本人の性格にもありますが、筆者は基本的に伝えるべきだと考えています。もし、本人の落ち度であれば、改善の機会になります。体制的な問題など、本人の落ち度でない場合でも、他の経路であやふやな情報が本人に届くよりは良いと考えているためです。

伝え方はさまざまですが、どのような理由であっても、私たちがメンバーを責めるべきではありません。その苦情を防ぐことができる「何か」があったとすれば、それを発見できる可能性は私たちが一番高かったはずです。

マネジメントをしていると、顧客の声を誤って受け取ることがあります。「感謝の声」のような耳あたりの良い言葉は、自分が頑張ったから「自分に送られた言葉だ」と受け取ります。「苦情の声」のように、自分に言われたくない言葉は、その原因をメンバー個人にすり替えようとなります。

顧客の声は常にプロジェクト全体のものです。私たちのものであり、メンバーのものであります。そして、多くの場合メンバーは私たちを経由するしか聞く方法がありません。

顧客からの声を聞きましたか？時間をとってメンバーたちに伝えにいきましょう！とても大切で重要度の高いタスクです！

8.10 マネジメントとメンバーの兼任

私たちは「PM（PL）兼エンジニア」のように、マネジメント業務とメンバー業務を兼任することがあるかもしれません。兼任は「危険である」とする声もある一方で、プレイングマネージャー^{†1}という言葉があるほど、需要が高いことが伺えます^{†2}。現在も小規模案件では、以下のような理由でプレイングマネージャーを見る機会は少なくありません。

- 人員が確保ができない
- 低予算プロジェクトである
- 両方の経験があるから、同時にできるでしょう（人件費の削減）

私たちがプレイングマネージャーとなるとき、意識しておくべきことがあります。

メンバーである前にPM（PL）である

私たちは、メンバーである前に、PMやPLの役割を優先しなければいけません。メンバーは他にもいますが、マネジメントを行う人は私たちしかいません。メンバーとして技術の楽しさやスキルの成長を優先すると、多くのメンバーは進むべき方向を見失います。技術への興味や楽しさを一時的に忘れましょう。

スケジュールはメンバーに任せる

兼任の場合、マネジメントに必要な情報も技術的なスキルもあるため、スケジュールの作成が容易な場合があります。しかし、スケジュールの妥当性や最終的な確認は、メンバーに行いましょう。スケジュールの案を作成することは良いことですが、スケジュールの根拠を説明し、メンバーに確認をして調整しましょう。「スケジュール作ったからよろしく！」は絶対にしてはいけません。多くのメンバーは不満を抱くことになります。

余力を持つ選択をする

以下のような理由で技術的なタスクを担うことはやめておきましょう。

- 難しそうだから挑戦したい
- やったことがないため興味がある
- 勉強しているため仕事で使ってみたい

これらのような経験のない技術要素は、知らなかつたことの調査や対応、トラブルなどの解決に多くの時間が必要になるケースが多いためです。

プレイングマネージャーは、可能な限り常に余力を持つ選択をしなければなりません。マネジメント業務かメンバー業務のどちらか一方に余力がなくなれば、プロジェクト全体のボトルネックになります。少々難易度が高くとも、メンバーを信頼し彼らに任せるようにしましょう。

新しいプレイングマネージャーを作るつもりで

私たちがプレイングマネージャーとして働くということは、組織がプレイングマネージャーという働き方を良しとしています。こうした組織は、プレイングマネージャーが必要なプロジェクトを多く生み出し、それを担える人材をいつも欲しています。

メンバーの中で、プレイングマネージャーに向いている人を探し、本人の意志を尊重しつつ了承を得られれば、マネジメントのタスクを少しお願いしてみましょう。現在のプロジェクトのみならず、今後の新規プロジェクトでも私たちを助けてくれるはずです。

プレイングマネージャーが、貧乏くじなポジションであると考える人も少なくありません。マネジメントとメンバーとしてのポジションになるため、どちらを優先するかなどの悩みは増え、責務範囲とプレッシャーも大きくなります。マネジメントか技術のどちらかの問題が発生すれば残業時間も多くなります。それに加え、技術のスキルが伸びる機会も減少します。

その中でも筆者は、以下のようなメリットも見出しています。

- ITエンジニアでも、マネジメントの経験を積む機会を得られる
- 他の組織や他部署など新しい人たちと出会う機会を得られる
- マネジメントとメンバーの両方の考え方や悩みを理解できるようになる

これらは、それまでになかった新しい価値観や考え方の視点を与えてくれました。

プレイングマネージャーに任命されましたか？ITエンジニアとして、技術的な成長の機会は少ないかもしれません、悲觀することはありません。技術以外の大切なことを学ぶ多くの機会を得ることができます。

†1 筆者の知っている限りでは、プレイングマネージャーは日本国内で通じる言葉です。海外では「Player manager」や「Player coach」という言葉がありますが、スポーツのチームに使われるもので、ITのプロジェクトで使われてはいないようです。

†2 一般的にプレイングマネージャーの「マネージャー」は課長などの役職を伴う人を指す場合も多くあります。本節では、ITのプロジェクトをまとめる役割をマネージャーと解釈して使用しています。

第

IX

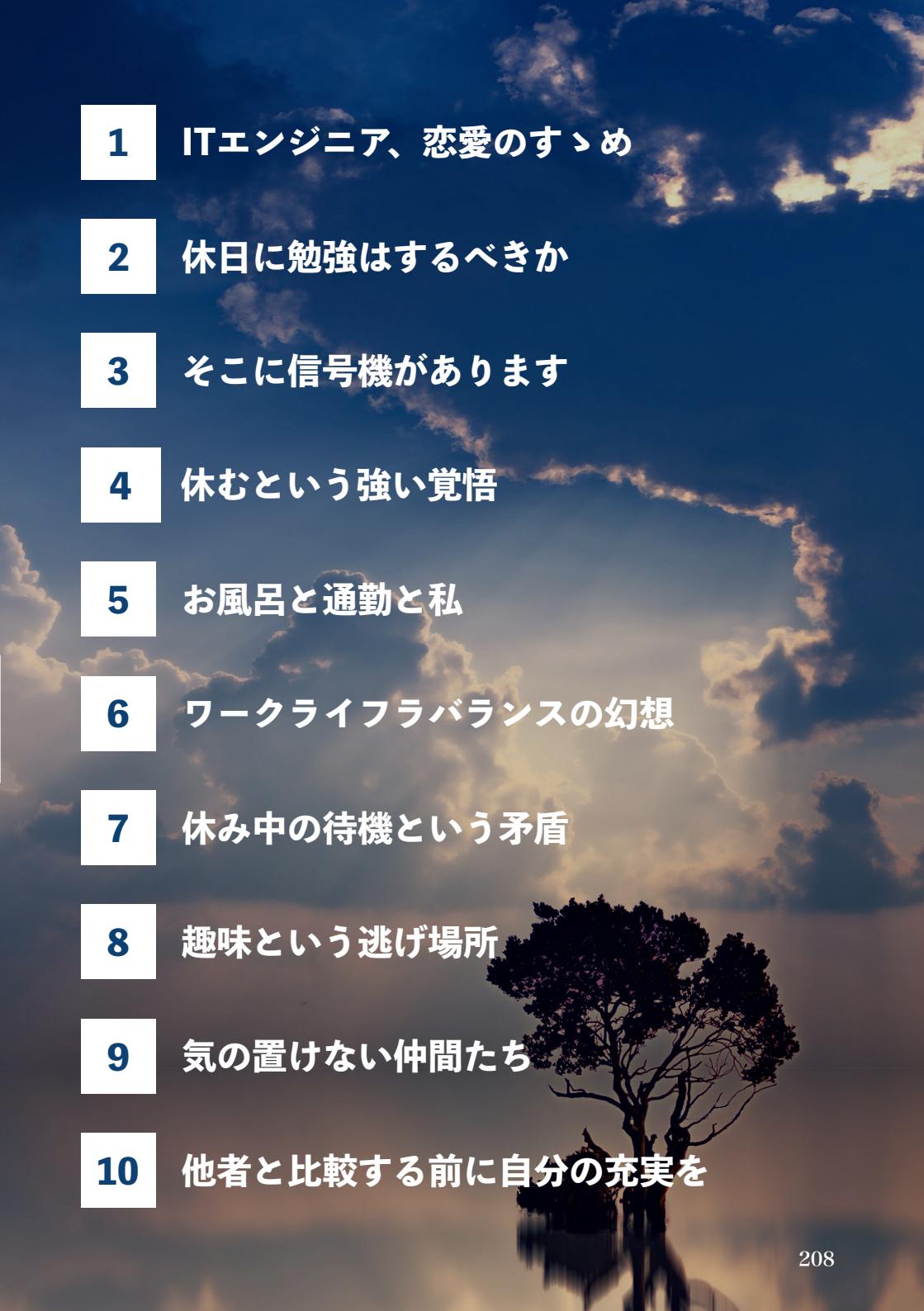
章

無能のためのプライベート

本章は、無能なITエンジニアとして生きていくための
プライベートについての教訓になります。

休みなどのプライベートな時間は大切ですが、
完全に安息したプライベートの時間を過ごせることは
少ないかもしれません。

プライベートを仕事に支配されないように
考える必要があります。



1 ITエンジニア、恋愛のすゝめ

2 休日に勉強はするべきか

3 そこに信号機があります

4 休むという強い覚悟

5 お風呂と通勤と私

6 ワークライフバランスの幻想

7 休み中の待機という矛盾

8 趣味という逃げ場所

9 気の置けない仲間たち

10 他者と比較する前に自分の充実を

9.1 ITエンジニア、恋愛のすゝめ

本節は、恋愛経験のない独身男性に向けたものになっています。既婚者や恋愛経験のある人は、読み飛ばしても構わないでしょう。女性の方は、独身者であれ既婚者であれ読み飛ばすことをお勧めします。筆者の男性視点からの見解を述べており、多くのフラストレーションを感じるはずです。

また、本節に限り「私たち」は「恋愛経験のない独身男性」、「彼女たち」は「恋人か妻」を指しています。

恋愛や結婚が素晴らしいものかどうかは個人によって変わりますが、私たちの中で恋愛や結婚を望み、行動しようとしていれば、筆者の経験上3つのことを覚えておかなければなりません。

お金は私たちだけのものではない

私たちの収入は、当然ながら私たちのものですが、彼女たちの多くはそう思っていないかもしれません。交際費や結婚後の共同生活の費用だけの話ではありません。例えば、私たちのお小遣いであったとしても、新しいパソコンや高額なグラフィックカードに何万円も使おうとした際、彼女たちは怪訝な顔をするでしょう。仕事に必要な参考書が5千円や1万円近くすれば、それらを不思議な目で眺め、積み本^{†1}をすれば早く読まないのかと催促されるでしょう。

私たちは、稟議書の準備と承認に多くの時間を費やすはずです。

時間は私たちだけのものではない

彼女たちの多くは、記念日や時間に敏感です。私たちが夜や休日に、仕事や技術の学習に多くの時間を使うことに嫉妬します。また、過剰な残業や緊急トラブルに徹夜対応することを仕事とは信じず、浮気を疑われるかもしれません^{†2}。記念日を穏やかに過ごすために、顧客や多くの人たちが関わる日程調整にも気を遣わなければならないでしょう。仕事関係の飲み会や気が重くなる出張、学習のためのカンファレンスへの参加などを遊びだと考えるかもしれません。

これらの理解を彼女たちから得ることは、組織から休日出勤や残業の申請の承認を得るよりも困難な場合があります。

私たちは新しい悩みを抱える

私たちは仕事で多くの悩みを抱えますが、プライベートでも多くの悩みを抱えることになります。仕事で厳しい局面を乗り越えたり、顧客や上司からの叱責で心がボロボロに疲れた後、帰宅すると、彼女たちからも心をボロボロにされるかもしれません。それは怒鳴り声だったり、無言の圧力として現れるかもしれません、私たちを攻撃してきます。これらの状況の多くは些細な理由から始まり、その内容もエディタ戦争^{†3}のようにひどくつまらないものです。私たちは怒鳴り返したり、静寂を貫き通したり、ときには謝つたりと、さまざまなアクションを行いますが、体と心を疲弊させることは間違ありません。

紹介した3つの事例は、筆者の経験した中でも相互理解と調和までに多くの労力と時間を費やしたものでした。これら以外にも些細な問題は多くありました。そして、これからも増え続け、その度に調和を図ることでしょう。

独身時代の筆者は、結婚にそこまで関心がなく、ITエンジニアとして仕事を続けることができれば良いと考えていました。しかし、幸いにも最良の妻と出会い、子どもを授かるなど多くの幸福を経験することができました。

「恋愛や結婚は良いものか」に対する筆者の最終的な答えが出るのは、今わの際と考えていますが、現時点では良いものであると考えています。しかし、良くするためには、私たちと彼女たちの多くの時間と歩み寄りが必要になるでしょう。

私たちは、答えのない未知のプロジェクトに挑戦しようとしています。

†1 購入した本を読まずに、部屋の片隅などに積んだままにしている状態のことです。筆者の自室には、常に5冊ほどの積み本が存在します。

†2 筆者の妻は、IT業界とは全く関係の無い仕事をしていました。その妻曰く、ITエンジニアの働き方はおかしいそうです。当時の筆者の周囲には、ITエンジニアか過剰労働の友人が多いため麻痺していました。

†3 プログラミングを行う人たちが「どのテキストエディタが一番良いか」をテーマにする論争です。この戦争は、特にvi愛好派とEmacs愛好派で熾烈な戦いを繰り広げました。

9.2 休日に勉強はするべきか

筆者は年を重ね、ITエンジニア志望者や初学者から、次のような質問を受けることが増えました。

「休日やプライベートの時間も勉強をするべきなのでしょうか？」

この質問は、筆者も若かりし頃に感じた悩みです。現在のところ、筆者なりに行き着いた回答は「休日やプライベートな時間を使った勉強は、条件付きでるべき」です。

ITエンジニアの仕事は知識産業です。定着している知識のみで継続できる仕事はほとんどなく、新しいものが次々と出てきます。もちろん、必要な知識を業務中に習得できるならば、定時後や休日などのプライベートの時間を使う必要はないでしょう。しかし、筆者の経験上、私たちが業務時間でこれらの新しい知識領域をカバーすることは、ほぼ不可能と考えています。

そして、プライベートの勉強は今後何十年も続くものです。長期的に行うためには、続けられる条件を決めておくことが大切です。筆者の「プライベートの時間を使って勉強するときの条件」を紹介しましょう。

条件1. まずは業務に直結すること

勉強を行うときは、業務に直結することを優先して行います。これらの勉強は以下のようなものです。

- 業務で使っている技術で、作りたいものを作ってみる
- 業務で使っている技術、もしくは関連した技術の参考書を読んでみる
- 業務に関連するわからない仕組みや用語を調べてみる

これらの勉強は、業務で活用できる機会が多く、無駄になりにくいものばかりです。

条件2. 興味のある分野も触る

業務以外でもインターネットやSNSで見つけた興味のある分野を学びます。手を動かす

ことあれば、参考程度に情報を見るだけのこともあります。モダンな技術だけではなく、現在は使われていないような技術も対象です。とりあえず興味があり、楽しそうと思えるものに触れます。

条件3. 勉強時間は決めない

「1日〇時間勉強する」のような時間を決めません。時間を決めてしまうと、やりたくない時もダラダラと続けたり、時間を守れなかったことに自分自身で落胆することがあるからです。そのときの意欲に応じて、1日1時間のことであれば、1日10時間することもあります。また、すべてを理解したり、試すこともしません。勉強をする意欲が続く時間で、できるところまでします。

条件4. つらいときはしない

人生の中で「つらい」ときは必ず訪れます。激務であったり闘病であったり、家族の不幸など、さまざまなつらいことが発生します。そうでなくとも、自身の理解の及ばなさや学習の遅さに、つらいと感じることもあるでしょう。

このようにつらいと感じるときは、一旦勉強をするという行為を忘れ去ります。長く続く人生、少しくらい勉強をさぼっても大きな問題はないでしょう。

条件5. 最優先はプライベート

プライベートは重要です。私たちは生きるために仕事をしているのであり、仕事や勉強をするために生きているわけではありません。家族との約束・仲の良い友人と遊ぶことなどは、勉強よりも優先することです。

筆者は、勉強を一種の保険だと考えています。将来的に使うことがあるかもしれません。逆に簡単にできるツールやサービスが登場したり、廃れて使わないかもしれません。それは、その時にならなければわからないものです。もしもの時のために、知識を積み立てていくのです。

また、勉強は最優先事項ではありません。保険の支払いのために、生活に困窮するほどお金を支払う人はいないでしょう。私たちの勉強も無理をしてまで積み立てるものではありません。余力がある時間に積み立てていくものです。

9.3 そこに信号があります

皆さんの周りに信号機はあるでしょうか？筆者の自宅の近くには、4つの信号機があります。信号機にも、定周期式・感応式・押しボタン式など複数のものがあります。これらは制御システムとして、高度なアルゴリズムが実装されていることでしょう。

本節で述べたいことは、信号機の難解なアルゴリズムの解説ではありません。普段過ごしているプライベートの中にも、ITの技術に関連する要素や助けとなることを見つけることができるということです。筆者が出会ったITに関連する一例を紹介しましょう。

考え方的なもの

信号機のアルゴリズムは複雑すぎて、筆者が理解することも、想像することも難易度が高いものです。もっと簡単な例にしましょう。筆者の自宅の階段には、下の階と上の階に照明のスイッチがあります。何度か往復しながら、照明のON・OFFをすれば、排他的論理和（XOR）という論理回路の動作原理を理解できるでしょう^{†1}。

技術的なもの

筆者が訪れた飲食店に、広告用の大きなディスプレイがありました。そこには、広告表示の上にポップアップとメッセージが表示されていました。

「このサイトのセキュリティ証明書の取り消し情報は、使用できません。」

どうやらWindowsのWebブラウザで、ポップアップが表示されているようでした。ポップアップのデザインからWindows10のように見受けられました。

このような光景は、次世代自販機・電光掲示板・ATMなどでも目にすることがあります。いずれも開発には携わっていませんが、これらの機器でOSが動いていることを知ることができます。また、提供しているサービスによっては、私たちの操作履歴などをネットワークを介してどこかへ蓄積しているかもしれません。これらの仕組みは、インターネットで調べると仕組みを説明したサイトを見つけられます。

デザイン的なもの

筆者は漫画や専門書など、本をよく読みます。それらの表紙や帯、中身に至るまで多彩で独特的なデザインに溢れています。また、長距離ドライブでは、斬新なデザインの看板を

発見します。美術館を訪れると、見たことのないような表現方法を知ることができます。多くは建物の内装も筆者を楽しませてくれます。これらは、筆者が資料を作成する際に、いくつかのアイデアを提供してくれます。

体験的なもの

コロナ禍以降、店内にメニューを置かずに店内の席から自身のスマートフォンを使って注文できるモバイルオーダーができる飲食店も増えてきました。筆者は、コロナ禍に初めてモバイルオーダーのお店を利用しました。感染リスクを少なくするため、店員との会話を避け、好きなタイミングで料理の注文が行えるモバイルオーダーという仕組みは、素晴らしいUX（ユーザー エクスペリエンス）を提供してくれました。

運用的なもの

筆者は、スーパーで無人レジを利用します。よく利用するスーパーの無人レジでは、ほとんどの操作は筆者が行いますが、年齢確認などがあると「無人レジの近くにずっと立っている従業員」が対応してくれます。たまに利用する他のスーパーでは「レジの従業員」が対応することが多いです。

スーパーごとに時間帯や呼ばれる頻度を考慮し、そのスーパーにとって一番良いと考えられた、異なる運用方法を採用していることがわかります。

これらの視点は、現在の業務に直接関連していないかもしれません。現在抱えている問題を解決してくれるものでもないかもしれません。しかし、少し異なる視点から物事を観ることは、新しいITの知識を得る機会になったり、将来的に役立つアイデアを見つける手助けとなります。

ずっとこのような考え方をすると、プライベートが疲れるものになってしまいますが、ふと思い出したときに少し違う視点で物事を視つめることを思い出してください。

今、あなたの周りには何が「見え」ますか？もう一度「見て」ください。明日か数ヶ月後か数年後かに、あなたの助けになることかもしれません。

†1 照明のスイッチは物理的であり、中身は3路スイッチで実装されていると思いますが、論理の理解の助けにはなるでしょう。

9.4 休むという強い覚悟

休日は、土日のような法定休日や祝祭日、有給休暇など、仕事を行わない時間として与えられています。しかし、私たちは悩み事を解消するために休日を仕事の時間に割り当てることがあります。

- 休日を使い進捗の遅れを取り戻す
- 動かないコードの原因を休日に探す
- 長期休暇中に資料を作成する
- 休日にタスクの進捗確認や資料の更新を行う

上記のことを楽しく過ごしているのであれば、筆者から何もいうことはありませんが、おそらくほとんどの人は不快に思っていることでしょう。楽しめていれば無能にはなっていないはずです。

日々悩み続けて仕事をしている私たちに、休日は必要なものなのです。私たちの身体と心を癒してくれます。友人と過ごせば楽しい時間になり、異性と過ごせばより仲を深めることができます。家族や子どもと過ごせば、素晴らしい思い出になります。

「絶対にすべての休日を休み続けろ」と主張しているわけではありません。プロジェクトというチームで動いているため、スケジュールは無視できません。特にリリース作業や移行作業などは、土日などの休日が多いため休むことが難しいこともあります。

そのような中で、私たちが休日を取得し、休みとして穏やかに過ごすためには「休む覚悟」が必要です。覚悟とは「悪い事態を予測して準備しておくこと」です。覚悟には、以下のようことがあります。

休むことによる影響

私たちが休むことによる影響を把握しておく必要があります。休むことによってタスクの遅延が発生する場合、影響が自分だけであれば、次の週に残業するなどで対応することができます。しかし、他者のタスクに影響を及ぼす場合は、休むことを諦めるか事前に相談し、了承を得てから休む必要があるでしょう。

また、休日に私たちが直接必要なタスクではない場合も、他者のタスクの影響により私

たちが必要になるかもしれません。例えば、他のメンバーが作業でトラブルに遭遇し、私たちが必要な場合は、休日であろうとも電話で呼び出しが発生するでしょう。

不在時のルール

私たちが不在時の対応ルールが決まっているかを確認する必要があります。休日のメールには自動応答機能などを使うかもしれません、緊急度の高い場合は電話連絡が来るかもしれません。私たちの不在時に、PMやPL、上司が対応してくれるルールが事前に決まっているのであれば、安心して休むことができるでしょう。

可能であれば、私たちの代わりに対応できるメンバーを代理者とするのが一番良い方法です。これはチーム内のルールで定められていたり、その都度メンバーで相談して決めたりとさまざまですが、代理者がいれば大きな問題が発生することは少ないでしょう。

私たちの分身

代理者がいる場合であっても、組織内に私たちの簡易な分身を作つておく必要があります。簡易な分身は、進捗や作業の履歴・メール・課題管理などの情報です。自身に関わる情報を他者が簡単に理解できる形で整理して残しておけば、その情報は私たちの分身になります。他者が何かを確認したい場合も、私たち自身に確認せず、分身に確認してくれるでしょう。

休む期間の長さによっては、これら以外のこと必要になるかもしれません「休む覚悟」があれば、必要な準備を見つけることは難しくないでしょう。しかし、覚悟することを忘れてしまうと、プロジェクトが順調なときできえ、休日に仕事を割り当てる習慣が身についてしまうことがあります。

休む覚悟によって、私たちは素晴らしい時間を過ごせます。休日の旅行中に呼び戻されたり、映画を見ている最中に退席することもなくなります。

次の休日を休む覚悟はありますか？覚悟に必要なことを把握し準備できていますか？

9.5 お風呂と通勤と私

1日24時間という時間は、すべての人に平等に与えられた資源だと言われています。この事実は疑いの余地がありませんが、筆者には少し異なることがあります。仕事の納期が迫ったり、新しいスキルを覚えたり、問題を解決したりする必要がある場合、他の人よりも多くの時間を必要とします。

架空の人物「Aさん」の忙しいときを例にしてみましょう。

1. 7:00 早めに出社し仕事をする
2. 18:00 進捗が悪いため残業をする
3. 23:00 新たなエラーが発生し、徹夜する
4. ----- 翌日 -----
5. 翌5:00 始発で帰宅しお風呂に入る
6. 翌8:00 出社し仕事をする

経験や問題解決のスキルが不足している場合、これらの状況では多くの時間を必要とします。このような忙しい時によく口にするのが「時間が足りない！」という言葉です。他の人と同様に1日24時間あるはずなのに、時間が不足してしまうのです。

しかし、この中にも、平等に近づけるために使える時間があります。

お風呂の時間

ゆっくりとお風呂に浸かることが好きです。おおよそ30分～1時間弱ほど入浴します。入浴中にパソコンは使えませんが、設計書の内容や直前の作業内容、エラー内容などは、頭に入っています。30分間考える時間があれば、間違っていた理由や次に試したいことを数パターン程度は考えられそうです。

通勤時間

通勤時間のうち、電車に乗っているのは、およそ1時間です。座れるかどうかは半々ですが、スマートフォンを持っています。立っている場合でも、お風呂で思いついた試したことやエラーコードの詳細をインターネットで調べることができます。有用な情報があ

れば、URLをメールかチャットで自分宛に送信することもできます。

時間が足りないと思っていた状況から、1時間30分ほど確保し、エラーの解決に取り組む時間を前倒しできました。

誤解なきように伝えておきますが、本章は「無能のためのプライベート」です。業務時間ではないため残業代は発生しません。そして、この例はさまざまな法令や組織の規則に反するものであり、推奨できるものではありません。

本節で述べたいのは、30分のような短い時間でも何かしらに活用できる可能性があることです。仕事のために充てることもできれば、動画をみたり漫画を読んだりなどの趣味の時間にもなります。難点としては、急には身につかないことです。その場に応じた集中の仕方などが体に馴染むまで反復する必要があります。

筆者は現在のところ、短い時間を以下のように過ごしています（表9-5-1）。

タイミング	時間	過ごし方
お風呂	30分～1時間	仕事の悩んでいることの再考や今日の振り返り
電車移動	30分	読書（集中が不要な本） or SNS
出張移動	数時間	読書（集中が必要な本） or SNS
子どもの寝かしつけ	30分～1時間	読書（集中が不要な本） or SNS
自分が寝るまで	30分～1時間	読書（集中が必要な本） or SNS

表9-5-1 筆者の短い時間の使い方

1日24時間は平等に与えられますが、時間が足りないことは多くあります。しかし、空いている時間は好きなように使うことができます。

9.6 ワークライフバランスの幻想

私たちの中には、激動のIT時代を乗り越えた人たちも多いことでしょう。筆者もIT業界が、新3K^{†1}や7K^{†2}と呼ばれた時代を経験し、少しばかり大変な期間を過ごしたのではないかと感じています。その頃の日々は過酷で、ワークライフバランスという言葉はまだ一般的ではありませんでした。

近年では、IT業界でもワークライフバランスやホワイト化などが進み、ITエンジニアがより人間らしい生活を実現できる機会が増えているように感じます。

内閣府の定義によれば「ワークライフバランス」は次のように述べられています^{†3}。

誰もがやりがいや充実感を感じながら働き、仕事上の責任を果たす一方で、

子育て・介護の時間や、家庭、地域、自己啓発等にかかる

個人の時間を持つて健康で豊かな生活ができるよう、今こそ、

社会全体で仕事と生活の双方の調和の実現を希求していかなければならない。

ホワイト財団による「ホワイト企業」の定義は次の通りです^{†4}。

私たちが考える「ホワイト企業」とは、
いわゆる世間で言われている「ブラック企業ではない企業」ではなく
「家族に入社を勧めたい次世代に残していきたい」企業を指します。

これらの定義や理念、実現に向けた取り組みは、素晴らしいものです。私たちの人生や今後の子どもたちの人生を楽しく豊かにすることでしょう。そして、現代において、このような組織や職場も少なからず存在することは事実です。

しかし、筆者がワークライフバランスは幻想であると考える2つのことがあります。

与えられるものではない

ワークライフバランスやホワイト企業のメリットの部分は、与えられるだけのものではありません。国の運営も組織の運営もボランティア活動ではなく必ずお金が必要です。そのため、私たち労働者は以下のようなことを実現させなければなりません。

- 限られた時間内で成果を出し続ける
- 次世代の育成の取り組みを持続的に行う
- 同時に生活（プライベート）を充実させる

これらを実現するのは難しいことでしょう。特に成果は、短い期間で最大の成果を求めるのははずです。

一定以上の優秀さが必要

ワークライフバランスの取れた組織やホワイト企業として知られている場で働くITエンジニアを見たことがあります。簡潔に言えば、彼らは非常に優秀なITエンジニアたちでした。彼らは、困難な問題にも臆せず立ち向かい、解決し続けていました。新しい技術などを次々と習得し、それでいて充実感がありました。

ホワイト企業とされる組織も増えてきましたが、私たちがそこに所属することで「楽に過ごせる」と考えるのは大きな間違いです。また、自身の組織にワークライフバランスのとれた環境やホワイト企業を求めるのも間違いです。それらは、私たちが成果をもたらすことができて、実現できるものだからです。

ブラック企業を容認するわけではありません。過去の残業や徹夜地獄のような日々に戻りたいとも思いません。しかし、一部の人たちが囁くような「楽に過ごせる」職場や仕事はないことを認識しておくべきです。

では、私たちはどうすればいいのでしょうか？残念ながら答えはありません。各自がどこを目指すかは、それぞれの選択に委ねられています。

筆者にとっては、ワークライフバランスは幻想であり、幻想を見るよりも目の前にある仕事に集中し続けることが、安寧に過ごすための答えです。

†1 ブルーカラーの職種について使われていた3K（きつい・汚い・危険）をIT業界に対して使われるようになったのが新3K（きつい・帰れない・給与が安い）です。

†2 新3Kに、規則が厳しい・休暇がとれない・化粧がのらない・結婚できないの4つを加えたものです。

†3 「仕事と生活の調和」推進サイト ワーク・ライフ・バランスの実現にむけて。仕事と生活の調和（ワーク・ライフ・バランス）憲章。2023年09月16日

https://www.cao.go.jp/wlb/government/20barrier_html/20html/charter.html

†4 一般財団法人 日本次世代企業普及機構（ホワイト財団）。ホワイト企業認定について。2023年09月16日

<https://jws-japan.or.jp/recognition/>

9.7 休み中の待機という矛盾

ITエンジニアの仕事の中には「待機」と呼ばれるものがあります。

- SESエンジニアの次の案件の調整中の期間
- 現地作業の結果報告待ち（問題があれば対応のため出動）
- 夜間・休日などの監視当番

待機時間は、半日や1日の短期間のこともありますが、数日以上の長期間に及ぶこともあります。これらの待機時間は、状況によってプライベートの時間が仕事に部分的な制約を受けることがあります。

待機時間にオフィスや客先に出勤していれば、業務時間であり給与も支給されることが多いでしょう。しかし、自宅待機などの場合は実稼働していないため、少額の手当や無給であることもあります。

たとえ少額な手当や無給の場合でも、仕事のために待機しているため、遊びのための外出は難しく、飲酒もできません。インドアな趣味をしていても、いつ連絡がくるのかと気になり集中できないことが多いでしょう。このように、プライベートの活動が制限されることがあります。

このような待機時間は、普段の業務時間中に行うほど優先度は高くないけれど、やりたいと思っていた仕事に取り組むのがおすすめです。筆者が、待機時間にしていることを紹介しましょう。

仕事から学んだことの整理

普段の仕事で得た技術の知識やノウハウを整理して蓄積することは、業務の優先度としては低くなります。以下のような地味に時間が必要になるものは、待機時間を活用できます。

- 知識を文章や図などの形で整理し、ナレッジとしてまとめる
- 参考になるURLやリファレンス情報を整理する
- 技術を使う際の前提条件を詳しく調べる

学習した内容の実践

参考書やインターネットの情報で新しい技術を「こんなものなんだ」と把握するのは簡単ですが、実際に動かして確認するには、環境の準備・コードを書くなど、時間要する場合があります。そのようなタスクに、待機時間を有効に利用することができます。

他プロジェクトの資料準備

プロジェクトが1つとは限りません。複数プロジェクトを担当している場合などは、他のプロジェクトの資料作成に必要な準備をします。

- 必要な情報や素材探し
- 図の作成やタイトルの作成
- 資料の骨格になる叩き台の作成

これらの地味な作業を終わらせておけば、業務時間は資料を作ることに専念できます。

新しくチャレンジしたいことの準備・着手

新しいことに挑戦したいアイデアがある場合、それに向けての準備や初めの部分を待機中に取り掛かっても良いかもしれません。これらは、想像は簡単にできても準備や着手には時間を要するものです。本書の執筆も構想自体は以前からありましたが、仕事がない期間（都合よく解釈すれば待機時間！）を使って執筆を開始しました。

注意点として、以下のようなタイミングも待機と呼ばれる場合がありますが、これらは待機時間ではありません。

- リリース後の本番利用開始日
- システム移行後の初の営業日

これらの時間は待機ではなく、業務に直結する重要な時間です。現地で待機していれば利用者の動向に目を凝らし、リモート待機であればリソースの負荷やログの出力を注視しなければいけません。これは通常、業務時間として扱われるはずですが、そうでない場合も問題が発生した際に迅速に対処できるよう常に緊張感を持たなければなりません。

待機時間は、連絡に怯えて何もせず過ごすのではなく、普段の私たちを助ける時間にしましょう。

9.8 趣味という逃げ場所

趣味は、私たちにワクワクする興奮や樂しみを与えてくれ、楽しい時間を提供してくれるものです。「趣味に逃げる」という表現は、ネガティブに受け取られることもあるかもしれませんが「安心して避難できる場所」と考えるなら、ネガティブな意味は全く感じないでしょう。その場所は、私たちにとって素晴らしい場所になるはずです。

逃げ場所が存在しないと、私たちは心の災害にどのように対応すべきか悩むことになります。例えば、美しい海を眺めていたとき津波が迫ってくるのを知った場合、私たちはその津波を眺め続けるでしょうか？それとも勇敢に立ち向かうでしょうか？私たちは、安心して避難できる場所へ逃げる選択をするでしょう。

ITエンジニアの中には「仕事に関連することが趣味」という人たちも少なくありません。それらの趣味は、仕事とうまく作用します。仕事自体が趣味であれば、日々興味のある発見を追求し、趣味を通じてお金を稼ぐことができます。技術の学習が趣味の場合も同様で、楽しみながら仕事に活用できる知識を獲得し、他の人たちよりも一歩も二歩も先を歩めることでしょう。

ITエンジニアが仕事に関連することを趣味にすることが、素晴らしいことのように語られることがあります。手放しで賞賛できるものではありません。それがうまく作用するのは、私たちの心に災害が発生していない間だけです。

私たちは「仕事に関連しない趣味」を逃げ場所として持っておくべきです。私たちは、1日の内で仕事を少なくとも7時間、多い時は十数時間しています。これに仕事に関連した趣味を追加すると、合計時間は膨大なものになります。心が災害に襲われた際、逃げ場所がなければ、仕事は憂鬱になり仕事に関連した趣味も苦痛になるでしょう。筆者の経験をいくつか紹介しましょう。

過度なストレスの軽減

仕事の中で、極度のプレッシャーに数ヶ月間さらされたことがあります。その仕事を乗り越えることはできましたが、次回も同じようなプレッシャーにさらされる可能性に怯え、パソコンを見ることすら憂鬱になり、仕事に関連した趣味に取り掛かる気力は全く沸きませんでした。この時、筆者には「小説を読む」という趣味がありました。一定の期間

の定時後や休日は、パソコンなどを一切触らず小説を貪り読みながら過ごし、ストレスを軽減させることができました。

自己肯定感の向上

筆者は最近「娘と遊ぶ」という趣味が増えました。彼女は慌ただしく動き、泣いたり笑ったりを繰り返し筆者を困惑させることもありますが、ITや仕事をことを忘れさせてくれます。「コレナニー？コレナニー？」とさまざまなものを見たりたがり、筆者がその問い合わせると、満面の笑顔で満足そうになります。IT関連の学習を趣味とする際、楽しいと同時に他の人と比較して自分は遅れているという劣等感を抱くことがありますが、娘と遊んでいる時は、自己肯定感を高め、自分の存在意義を感じることができます。

新しい出会い

筆者の運営している「ITエンジニアのまとめ (<https://itenginner-matome.net/>)」は、仕事に嫌気がさしていたときに「まとめサイトを見る趣味」から自分で作れるんだろうか？という好奇心によって生まれました^{†1}。このサイトを多くの人に知つてもらうためにX（旧Twitter）も始めました。知らなかつた技術とともに、フォロワーという新しい出会いを得ることができました。

筆者には、他にも仕事に関連しない趣味があります。これらをしているときは、仕事を忘れ、楽しい時間を過ごすことができます。

- 国内旅行（海外は緊張するため国内旅行を好みます）
- 料理をしたり、食べたりする
- 映画やアニメを見る
- （気が向いたときに）運動する
- DIYで何か作つてみる

仕事に関係のない趣味を持っていたら、その趣味の良さをSNSで投稿してみてください。私たちの多くが、その投稿を見つけるはずです。

^{†1} 当時の筆者の仕事は、Web関連の技術とは関係のない仕事でした。

9.9 気の置けない仲間たち

気の置けないとは「気配りや遠慮をしなくてよい」ということです。

仲間とは「一緒に物事をする間柄」にある人たちです。

つまり、気の置けない仲間とは、私たちが遠慮せず話すことができ、私たちの悩みに一緒にになって悩み、考えてくれる人たちです。これらは同僚かもしれませんし、配偶者や両親などの家族、友人たちかもしれません。

私たちは、日々多くの悩みを抱えています。仕事のこと、技術のこと、プライベートなことなど、悩みが溢れています。悩みを遠慮なく打ち明け相談し、一緒になって考えてくれる人たちを持つことは大切です。彼らは必ずしも私たちに明確な解決方法を与えてくれるわけではありませんが、多くの気づきを与えてくれます。

筆者に気づきを与えてくれた例をいくつか紹介しましょう。

仕事の悩みへの気づき

筆者は技術力による仕事への貢献度の低さに悩んでいました。同僚たちは、多くの仕事を短い時間で終わらせていましたが、それに比べ筆者は多くの残業を使っても終わらせることができませんでした。この悩みを、プライベートでも親しい関係にあった「気の置けない先輩」に相談したことがあります。

彼は、まず筆者の仕事における強みや良い点を多く並べてくれました。それに続いて、筆者の悪い点や改善が必要な部分についても率直に指摘しました。当時の筆者は、自身の仕事が純粋に技術的な貢献で評価されると考えており、人に対する考慮が欠けていました。それは、技術を扱う同僚に対してだけでなく、営業担当やときには顧客に対してのときもありました。

彼の指摘は、筆者の技術力を高めることはませんでしたが、技術以外でも貢献できることがあることに気づかせてくれました。

限界値への気づき

妻は、恋人である時から私たちの仕事がいかに大変であるかを理解していました。筆者

の帰宅の多くが終電であったり、徹夜明けに始発で帰宅しシャワーをしてすぐ出社していましたことを知っていました。仕事で大きな失敗をし、叱責を受けた後は帰宅後に大きなストレスを感じて泣くことがよくあることも知っていました。

それでも仕事のことに助言などしなかった彼女が、筆者に提案をしてきたことがあります。「もし、これ以上仕事をするなら離婚する。でも、その前に心療内科を受診して欲しい。」と言ったのです。妻の言葉で筆者は心療内科を受診しました。

心療内科の医師は、いくつかの質問への筆者の答えを聞いた後、即座に仕事から距離をおき、可能ならば退職を考えるべきだと言いました。筆者は心の限界を迎えていました。

心の限界に気づいたのは、筆者本人ではなく妻でした。

気の置けない仲間には注意も必要です。同僚は私たちの仕事を知っており、家族や友人は私たちのプライベートの過ごし方を知っています。両親であれば私たちの性格まで知っていることでしょう。

しかし、彼ら全員が気の置けない仲間とは限りません。一緒にいる時間が長いことや旧知の仲であればいいわけではないからです。私たちが、本心を打ち明けることができ、一緒に考えてくれる人でなければなりません。

逆に言えば「最近友人になった人」や「インターネット上の付き合いの人」も気の置けない仲間たちになるかもしれません。幸いなことに、私たちはこのような機会を得るツールの使い方に長けています。

筆者は「人は最後は結局一人」だとも考えています。何かに悩み判断するとき、他者との完全な共有は不可能であり、最終的には自身で決断しなければならないからです。それでも、気の置けない仲間たちは、私たちが決断するための新たな気づきを与えてくれ、ときには失敗につながる決断を再考する機会を与えてくれます。

9.10 他者と比較する前に自分の充実を

私たちは、しばしば自分と他者を比較し、喜んだり悩んだりします。特に仕事では、技術的な知識やスキルなど比較を行えることが多いため、他者との比較が頻繁に発生します。心理学者レオン・フェスティンガー氏によると「人は他者と比較し自分の評価」を行うのだそうです^{†1}。

この考えを、私たちは経験的に知っています。学生時代にはテストの点数を比較し、社会人になると収入や仕事の成果を比較しています。他者との比較は、自分のポジションや市場価値を把握したり、目指す方向を探すためにも必要なものです。しかし、比較には上方比較と下方比較があり、これらのデメリットは私たちのプライベートに悪い影響を与えることがあります（表9-10-1）。

比較の種類	上方比較	下方比較
比較対象	自分より優れた人	自分より劣った人
メリット	モチベーションを高めたり、より高い目標を持ち自身の成長の機会となる。	自己評価を向上させたり、他者より優れていると感じることでストレスを軽減できる。
デメリット	自己評価を低下させたり、他者より劣っていることにストレスを感じることがある。	自己満足を過剰に高めたり、自己の問題から逃避し、改善や成長の機会を逃すことがある。

表9-10-1 比較の種類とメリット・デメリット

上方比較と下方比較のデメリットが過剰に影響を及ぼさないように、筆者が気をつけていることを紹介しましょう。

仕事による比較

上方比較となる人々は、素早く設計や実装を行い、トラブルにも迅速に対応します。それらの激務を行いつつ、多数の資格を取得している人もいます。しかし、彼らがそこにあるまでには、多大な時間と努力を費やし、苦難を乗り越えているはずです。彼らに追いつくために無謀な努力やプライベート時間を犠牲にする必要はないでしょう。私たちに必

要なことは、自身の責務範囲の仕事を果たすことだからです。

下方比較となる人たちは、一部の側面で私たちより劣っているかもしれません、それが、私たちが学習を怠つたり、仕事を手抜きしたりする理由にはなりません。彼らは、他の部分で優れているかもしれません。

収入による比較

上方比較となる人たちは、高い収入を得て豊かな生活をしているかもしれません、それらの仕事は楽ではないでしょう。過度なプレッシャーに耐え、プライベートの時間を使っていることもあります。筆者は家族が安心して生活できる収入があれば、安心感を得られます。

下方比較となる人たちは、私たちより収入が低いかもしれません、その代わりに余暇の時間を有意義に過ごすことに喜びを感じているかもしれません。また、彼らの収入が低かったとしても筆者の生活の豊かさには全く関係しません。

過去の自分との比較

頑張っていた時期の自分と上方比較をし、なぜ現在の自分があの頃のように頑張れないのかと自問自答することがあります。しかし、その差異は若さや健康の違いによるものであることも理解しています。また、現在の筆者は、過去の自分が持つことが難しかったプライベートの時間が多く持てています。

頑張れていなかった時期の自分と下方比較をし、現在頑張っている自分に安心することがあります。しかし、過去の自分とは年齢も生活基盤の状況も異なります。現在の仕事の役割の達成と家族との時間の充実に焦点を当てる必要があります。

これらの比較は、人によって異なりますが、ほとんどの人が何らかの形で行っています。人生が順調な場合、比較はモチベーションや目標設定に役立つことがあります、苦しい状況であったり「こんなはずではなかった」と感じる場合は、デメリットが顕著に現れます。

そのような時には、他者との比較ではなく「現在の自分が充実するにはどうすればよいか」を考えることが大切です。

†1 Wikipedia. 社会的比較理論. 2023年9月4日 00:23
<https://ja.wikipedia.org/wiki/社会的比較理論>

第

章

X

無能のための未来

本章は、無能なITエンジニアとして生きていくための
未来についての教訓になります。

未来には希望と同時に恐怖も存在します。
私たちは、急速に変化するITの世界で、
正しいか間違っているかもわからない中から
自分の未来を選択しなくてはなりません。

正しさは誰にもわかりませんが、
間違った選択をしないようにしなければなりません。

1 成長による停滞、成長なき後退

2 無くなるのではない、変わるものだ

3 バズワードの危険性

4 具体的な未来は考えない

5 35歳定年説は崩れたものの・・・

6 転職の葛藤

7 スカスカな職務経歴書

8 独立という考え方

9 高学歴化への備え

10 未来よりもイマを大切に

10.1 成長による停滞、成長なき後退

筆者がITエンジニアとして働く中で、とある先輩のITエンジニアから助言されたことがあります。覚えている限りでは、彼の言葉は以下のようなものでした。

IT業界で働き続けたいなら、新しいことも古いことも勉強し続けてください。

今の知識だけで仕事を続けたらどうなると思いますか？

業界の流れに取り残され仕事がなくなります。

勉強を続けてやっと求められるができるようになります。

もし、優秀になりたいのであれば、周囲の人たちよりももっと勉強してください。

優秀な人たちもずっと勉強をしています。

君がどこを目指すのかはわからないけれど、

この業界にいるつもりならずっと勉強してください。

筆者はこれを「成長による停滞、成長なき後退」と心の中に刻んでいます。彼の助言から学んだ大切なことが2つあります。

ITエンジニアには継続的な成長が必要

1つ目は、勉強を続ける（成長を続ける）必要性を確信できたことです。助言を得た後、働く中で出会ったITエンジニアたちに「普段から勉強をしていますか？」と聞いて回った時期がありました。勉強の方法や使う時間は異なりましたが「この人のようなITエンジニアになりたい」と思える人たちは、絶えず何かしらの勉強を行っていました。

成長とは新しいことだけではない

2つ目は、成長は新しいことだけでなく、古いことについても起こり得ることです。多くのITエンジニアは新しい技術やトレンドな技術を勉強しようとしますが、古い技術や局所的に使われている技術だとしても成長はできるのです。

筆者は新しい技術も古い技術にも詳しいレベルには至っていませんが、両方において広く浅い知識はあります。この知識は、現在も筆者の仕事を助け続けてくれています。

そして、筆者自身のITエンジニアとしての経験から、もう1つ気づいたことがあります。

私たちが働き続けるためには「停滞するための成長だけ」が必要ということです。前進するような成長は必要ありません。私たちに必要なのは、前進ではなく停滞し続けることです。

停滞とは、その時代や状況の仕事において必要となる最低限を維持できていることをします。この停滞のラインは、数年後には時代の進みとともに最低限を下回ります。数年後までに、その時代に停滞するために必要となる成長だけを行うという考え方です。成長は必要ですが、停滞するよりも大きな成長は必要ありません。

ITの流れについていくために勉強し続けることは不可欠ですが、今後の何十年続くITエンジニア人生において前進し続けることが重荷になることがあります。前進に必要となる過度な勉強は、時に心や体を壊し、停滞を続けることも困難にします。

筆者が前進をしようとして失敗した例は以下のようないました。

- 新しい技術が多すぎて何から手をつけてよいかわからなくなる
- 1つの技術をとっても奥が深く身につけるには多くの時間がかかる
- 前進しているITエンジニアたちに追従できない劣等感を日々感じる

筆者は現在も勉強は行っていますが、それらは新しい技術やトレンドな技術、資格を取得するための勉強ではなく以下のようないます。

- 目の前にある仕事に必要となる技術的な勉強
- 仕事で必要となる技術以外の勉強
- 過去に関わらず興味のある勉強

これらは前進したり優秀になるには少なすぎるのですが、停滞するためには十分なものでした。筆者は、未来でも停滞し続けることを最優先で取り組んでいるでしょう。

私たちは、今後の未来に向けて停滞し続けるための勉強を続けなければなりません。しかし、前進する必要はありません。

10.2 無くなるのではない、変わるもの

自分の仕事が無くなることに恐怖する人がいます。無くなりそうな仕事について、面白おかしく伝えるメディアの報道を聞いたり、IT系の雑誌で見たりすると、酷く恐怖し未来へ不安を感じことがあります。

ITの分野は進化による変化が多くあり、無くなってしまった技術やプロダクト、ベンダーが多数存在することは事実ですが、それに恐れを感じる必要はないと考えています。

無くなることは少ない

現在の私たちが使っている技術が、突如別の技術によって完全に取って代わられるることは、少ないと考えています。

2000年代初頭～中盤ごろに、WindowsがLinuxに置き換わると言われたこともありましたが、Windowsは現在でもクライアントOSとして圧倒的なシェアを保っています。サーバーOSでは、WindowsとLinuxはそれぞれ得意な分野で共存しており、必要に応じて使い分けられています。

2010年代前後に仮想化技術やクラウドサービスが流行り出したころ、オンプレミスのハードウェアは減少し運用が楽になり、一部のITエンジニアは不要になるとの意見がありました。仮想化やクラウドは多くの企業で使われていますが、現在も運用負荷の軽減とITエンジニアの獲得に奔走しています。また、ハードウェア上で稼働するレガシー形式のOSすら無くなるには至っていません。

2010年後半～現代では、AIは飛躍的な進化を果たしました。特に大規模言語モデルの登場により、プログラマの職業が無くなるとも言われていますが、AIの出力結果には、不正確なことも多くプログラマは依然として不可欠です。また、現在のAIブームは3回目^{†1}ですが、これまでにプログラマが消滅したことはありませんでした。

無くなったものは変わったもの

確かに、無になった技術やプロダクトもありますが、それは新たな需要に対応するためには変化した結果です。

Adobe Flashは無くなりましたが、代わりにHTML・CSS・JavaScriptが必要を満たしています。無くなったプログラミング言語やフレームワークもありますが、新しい需要が生まれたために別のプログラミング言語やフレームワークが誕生しました。メインフレームやCOBOLも消える傾向にありますが、今後はオープンシステムに変わることでしょう。そして、無くなるにもまだ相当の時間を要しそうです。

筆者はこのような「無くなる」ことに対して、楽観していることが2つあります。

1つ目に、私たちの仕事が無くなる技術に依存していないことです。新しい技術に取って代わられることがあっても、本来の需要は残ります。そして、その変化も数年以上の時間が必要であり、先行者の知識をインターネットや書籍などで得ることができます。最先端の立場にいる必要はなく、仕事に必要となる変化に対応していければ、生き残れると考えています。

2つ目に、完全に無くなることは極めて稀のことです。1990年代までのハードウェアやプロトコル、プログラミング言語の仕様などの覇権争いをしていた頃と比較すると、現代は比較的安定しています。基礎技術や考え方方が拡張されたり、組み合わされることはありますが、完全に別のものに取って代わることは少ないでしょう。TCP/IPが消える未来は想像できませんし、現在のプログラミング言語の仕様が全く通用しなくなるのも遠い未来のように思われます。一度身についた知識が無駄になることはないと考えています。

「無くなる」ことへの注意点としては、私たちが使う開発の支援などを担うツール類は特に変化が激しいことです。これらのツールは便利ですが、ツールの使い方を覚るために多大な時間を使うべきではありません。数年のうちに別のツールが出てくる可能性が高く、また新たに使い方を覚えなければいけないためです。必要な使い方はその時に調べれば十分だと考えています。

これからも新しいITの技術や職業は次々と登場し、それに伴い無くなることがあるのは事実ですが、多くは変わっているものです。私たちが恐れるべきことは、無くなることではなく、仕事に必要となる変化に着いていけなくなることでしょう。

†1 マイケル ウルドリッジ. AI技術史 考える機械への道とディープラーニング. 株式会社インプレス. 2022年3月. VIページ

10.3 バズワードの危険性

ITエンジニアの世界は、常にバズワードという言葉の嵐にさらされています。バズワードは「専門的そうに聞こえるが、抽象的で不明確な部分がある言葉」です。これらの言葉は、しばしば新興の考え方や技術と関連づけられ、業界のトレンドとなることがあります。バズワードの一覧は見つけられませんでしたが、2023年のテクノロジーに関するトレンドワードには、以下のようなものがあります（表10-3-1^{†1}）。

No	ワード	No	ワード
1	普遍化するAI（人工知能）	6	NLP（自然言語処理）
2	量子コンピューティング	7	グリーンテクノロジー
3	XR（クロスリアティリティ） IoTとエッジコンピューティング	8	サイバーセキュリティ
4	ロボティクスとオートメーション	9	デジタルツインズ
5	バイオテクノロジーと遺伝子工学	10	メタバース

表10-3-1 2023年のテックトレンド

バズワードがトレンドワードになることや、反対にトレンドワードがバズワードになることもあります^{†2}。最近ではAI（人工知能）・DX（デジタルトランスフォーメーション）・Web3.0などはバズワードだったと言えるでしょう。

バズワードを把握しておくことは有用です。バズワードは多くの人の議論の的になり、顧客や経営者はバズワードに注目しています。そして、私たちはしばしば彼らからバズワードに関する技術的観点の優位性などの質問を受けることがあるからです。また、バズワードは、AIエンジニアやDXエンジニアなど新しい仕事や職業を生み出すことがあります。バズワードが生み出した仕事をする人は「先進的で」「高い給与で」「渴望されている」人材のように謳われることがあります。

これらは、必ずしも間違いではありませんが、そのような仕事を無闇に目指すべきではない理由がいくつかあります。

新しすぎる

バズワードに関する技術は比較的新しいものです。それらの多くは定石的な扱い方がない、ビジネスへの活用方法も模索段階です。学ぶ教材や情報も少なく、先進的なために専門の指導者もほとんどいません。自分でその技術の活用方法を確立させるために多くの労力を注ぐ覚悟が必要です。

そもそも出遅れている

バズワードに関連した仕事は、収入面などが高待遇で求人されることがあります。これらに対応できる人の多くは、バズワード化する以前からその分野で活動していた人たちです。バズワードを知ってから目指したのでは出遅れています。追いつくことも可能かもしれませんが、過酷な競争になるでしょう。

消えやすい

バズワードは新しい言葉が毎年誕生し、一時的に注目を集めているだけのことがあります。今後、大きく発展する可能性がある一方、消えやすいものです。学んだことは無駄にはならないでしょうが、現在の実務に時間を割く方が良いこともあるでしょう。いずれにしろ、安定とは程遠いところにあります。

かつては、クラウド・ビッグデータ・RPA・SaaSなどもバズワードでした。これらのバズワードは、ITエンジニアの新しい働き方や仕事の道を広め、扱う技術や仕事をさらに便利に発展させ続けています。一方でバズワードは、マーケティングに悪用され実態を伴うこともなく消えることもあります。

バズワードに可能性を見出そうとするとき、私たちの未来を完全に託すことは危険です。私たちの未来を託せるのは常に私たち自身です。

†1 トレンドワードは以下のURLを参考に筆者が日本語訳したものです。

LinkedIn. Unveiling the Future: Top 10 Tech Trends In 2023. 2023年4月5日

<https://www.linkedin.com/pulse/unveiling-future-top-10-tech-trends-2023-maham-shafiq>

†2 トレンドワードとバズワードの明確な線引きは難しいものです。本節では同様の意味として使用していますが、筆者は以下のような違いがあると考えています。

バズワード：特定の業界やコミュニティの中で使われ議論の的になるもの。トレンドワードより先進的なことが多いが、抽象的で具体的な定義が不足している。

トレンドワード：バズワードよりも業界やコミュニティが広がり一般に浸透しつつあるもの。バズワードよりも具体性がでてきているもの。また、季節などの期間に関連する言葉として使用される場合もある。

10.4 具体的な未来は考えない

「将来どうなつていいか？」のフレーズは、子ども時代では夢を持つために、大人時代では就職や転職活動・組織の評価面談でよく耳にするものです。筆者が、ITエンジニアとして働きたいと就職活動していたときは、以下のような未来を考えていました。

- 特定の分野で社内トップクラスの実力を身につけたい
- 上流工程から下流工程まで幅広くこなせるようになりたい
- プロジェクトマネジメントが出来るようになりたい
- 海外でITエンジニアとして働きたい
- 後輩の育成に貢献したい
- いずれは起業に挑戦してみたい

これらの目標のうち、部分的に達成できたものもありますが、筆者の行動ではなく、行っていた仕事から派生して運良く実現したものばかりです。実現できなかった目標は、筆者の能力や努力が不足していたためでしょう。

具体的な未来を考えることは、今後の計画や方向性を定める役割を果たしますが、筆者はさまざまなものを見ることによるメリットもいくつか発見しました。

価値観は変わるもの

価値観は、時代とともに常に変わっていきます。個人の価値観だけでなく、組織や社会の価値観も変わります。5年後や10年後の価値観を予測するのは難しいでしょう。筆者は、ITエンジニアという仕事が天職であり最も優先すべきものと考えていました。現在では楽しくはありますが、天職とは考えていません。また、仕事の優先順位も以前ほど高くありません。

プレッシャー・ストレスの軽減

具体的な未来の目標に向かうのは、非常にプレッシャーのかかることです。常にプレッシャーを感じ、目標を達成できないことで自己評価が低下することもあります。具体的な行動をしていないことがストレスとなり、ときには自分自身に失望したり人生に後悔したりします。

具体的な未来のビジョンを描かないことで、これらのストレスが軽減される可能性があります。

選択肢を広げる

具体的な未来を考えることは、私たちの選択肢を制約することもあります。ある目標に焦点を合わせることで、他の選択肢を見過ごしてしまうのです。しかし、私たちには多くの選択肢があり、選択肢は時間と共に変化します。

現在の目標以外にも、生き方にはさまざまな選択肢が存在します。読者の皆さんはITエンジニアを続けたいという気持ちで本書を読んでいるかもしれません、ITエンジニア以外の生き方もあるのです。

現在に集中

私たちにとってまず大切なことは、何年も先の未来ではなく、目の前にある仕事を終わらせることです。もしくは明日か明後日か、近い未来に期限が差し迫ったタスクを終わらせることです。5年後の未来を考えるより、現在に集中し仕事を遂行することで、達成感を得られる機会が増えるでしょう。

未来に向けた計画を立てることは大切ですが、具体的な未来のビジョンを持つことが常に有用なわけではありません。筆者は近い未来は具体的に考え、遠い未来はぼんやり考えています。

「来週はどうなっていたいですか？」

差し迫ったタスクを片付け、仕事を完遂させたいですね。次の土日は休んで娘と遊びたいです。夜は穴子の煮付けを作つてビールを飲みたいです。そのために、仕事を無事に終わらせるための準備を進めています。

「5年後どうなっていたいですか？」

目の前の仕事を完遂できる人でいたいですね。家族で暮らせる収入を維持しながら、健康で過ごせていれば幸せです。できれば、大学に通いたいと考えています。

これは組織からすると酷くつまらない回答かもしれません、筆者が楽しい未来を描くために必要なものです。

10.5 35年定年説は崩れたものの・・・

2000年代頃からIT業界で広まった、ITエンジニアの「35歳定年説^{†1}」が崩壊したことは周知の事実でしょう。35歳定年説の出自は定かではありませんが、以下の理由からだつたと記憶しています。

- 体力が追いつかない（当時は2～3徹夜が恒常化していた）
- 学習能力が低下し新しいスキル習得が間に合わない
- エンジニア以上にマネジメント層の需要が高かった
- 年上のエンジニアに指示を出すことを嫌がる人がいる（と言われていた）
- 汎用システムからオープンシステムへの波に乗れない人がいると考えられていた

また、専門分野をもったプロフェッショナルなITエンジニアは当てはまらず、凡庸なITエンジニアたちが危ないと認識されていたように記憶しています。

しかし、現在では40代や50代、さらには60代でも前線でバリバリ働くITエンジニアが多くいます。再雇用やシニアエンジニアの求人も増えてきたように思います。私たちが60代以降もITエンジニアとして働くことは夢物語ではなく、実現可能なものでしょう。私たちももう35歳定年説にビクビクと怯える必要はなくなりました！

ただし、私たちが年齢を重ねITエンジニアとして働き続けるために意識しておくべきこともあります。

体力の低下

年齢を重ねると、体力の低下は避けられない現実です。笑顔で徹夜をしたり、夜遅くまで取り組める体力は20～30代で失われることでしょう。40代になれば、早く就寝したくなるほど、体力はなくなります。最近では、筆者も徹夜や長時間の残業を行うと体にダメージが蓄積されていることを実感し、回復までに相当の時間を要するようになりました。

ストイックな人は、激しい運動で強靭な体力作りをしつつ仕事をこなす人もいますが、私たちは少なくともウォーキングやストレッチなどをして、体力を可能な限り維持することに気をつけなければなりません。

学習時間と学習方法

年齢を重ねるにつれて、学習能力や記憶力が低下することは確かですが、それよりも大切なのは学習時間の確保です。年齢を重ねると、自分に使える時間が減少していきます。家族と過ごす時間・子どもの教育・親の介護・自身の病気・若手の育成など、私たちを取り巻くことに割り当てるべき時間が増えていきます。どの時間を学習に充てるべきか、どの学習方法が自分に合っているのかを意識する必要があります。

マネジメントの役割

年齢を重ねると、少なからずマネジメントの仕事が増えていきます。管理職になることもあれば、PM・PLのような役割を期待される機会も増えてきます。これらのような明確な役割でなくとも、少人数のチームや関係者をまとめる機会は徐々に増えることでしょう。回避策は強い意志を持って断る以外にありませんが、自身の成長の機会損失になるかもしれません。そのような役割が回ってくることを意識しておくべきです。

技術だけの限界

卓越した技術スキルや専門的な分野の知識を持っていない場合、技術力だけでITエンジニアを続けていくのは難しいと考えています。私たちは技術力に加えてプラスαが必要です。プラスαは、設計能力・マネジメント能力・業務や業界知識・教育能力・語学力などさまざまです。どれも高い能力が必須なわけではありませんが、仕事に活用できるプラスαを意識しておく必要があります。

ITエンジニアの年齢制限を恐れる必要はもはやありませんが、意識して備える必要があります。年齢による影響は、個人の状況や環境によって大きく異なります。筆者は、これらの要因を意識する前に年齢を重ね多くの時間を失いましたが、これから備えには間に合うと信じています。

そして、皆さんの備えも間に合うと信じています。本書を読んでいることから、これらも長い間ITエンジニアを続けたいと考えているでしょうから。

†1 「プログラマ35歳定年説」や「SE35歳定年説」とも言われます。ITのエンジニアとして働くのは、35歳頃が限界であることを表す言葉です。

10.6 転職の葛藤

ある程度の経験を積むと、転職の誘いが増えることがよくあります。誘ってくる人たちは、以下のようにさまざまです。

- 別の組織に転職した元同僚
- 起業を成功させた知人
- 一緒に働いたことのある顧客やビジネスパートナー

誘ってくる人たちはさまざまですが、多くの共通点があります。彼らは、現在の私たちの収入より高い金額を提示し、私たちが如何に必要な人材であるかを説明し、彼らの仕事が魅力的で楽しいことを熱く語ります。

これらは非常に魅力的なものばかりです。そのときまで転職のことなど考えていない場合でも、転職を考え出すほどです。私たちを素晴らしい場所に導いてくれる機会が訪れたと感じることでしょう。

しかし、このような魅力的な誘いに応じる際に、注意すべきことがあります。

リセットの覚悟

新しい職場に転職すると、これまでのさまざまなものがリセットされる覚悟をしなければなりません。

- 職場の人間関係
- 職場での評価
- 仕事のやり方

これらは短期間で築けるものではなく、これまでの蓄積によって得られているものです。新しく築きあげるには、時間も労力も必要です。人によっては、フラストレーションの原因となることもあるでしょう。これらを再度構築する覚悟が必要です。

ジョブチェンジのリスク

新しい職場で以前と全く同じ仕事を続けられるとは限りません。ITエンジニアからPMやコンサルタント、もしくは現在の業務とそれらを同時にこなすことがあるかもしれません

ん。自分のやりたいことであれば問題ありませんが、魅力的な条件に惑わされると、これらも大きなフラストレーションの原因になります。

お金と労力

転職に伴って収入が増えることは喜ばしいことですが、相応の成果を出すプレッシャーも高まります。特に働く地域や業界が同じ場合で収入が上がるときは、これまで以上の成果を求められていることを意識しなければいけません。

リファラルのジレンマ

誰かに誘われる場合、多くはリファラル採用になるでしょう。リファラルは相手側にとっても組織から紹介料をもらえることや見知った人と一緒に仕事ができるなどのメリットがあります。見知った人と仕事ができるのは私たちにもメリットになるでしょう。

しかし、リファラル採用が、私たちにもたらす2つのことも考えなければなりません。

1つ目は、私たちが期待されている以上の成果が出せない場合、紹介者に恥をかかせる可能性があることです。自分以外の人の期待も背負い成果を出し続けるというのは、非常に大変なことです。

2つ目は、辞めたい場合に辞めにくくなる可能性があることです。仕事や人間関係が良好でない場合などに早く辞めたい時にも、紹介者の立場を考えると辞めるには幾分かの勇気がいるでしょう。制度上は辞めても問題はないでしょうが、紹介者との関係性が悪化するかもしれません。

誘いを受けることは誇らしいことです。目指すところと合致するのであれば、どんどんチャレンジした方が良いと考えています。しかし、転職にはリスクも伴います。失敗した際に、再び転職するのは大変なことですし、家族がいればプレッシャーは相当なものになるはずです。

魅力的な言葉だけに耳を傾けず、葛藤に悩み抜きながら判断する必要があります。

10.7 スカスカな職務経歴

職務経歴は、同じ組織で働いている間は、大きな問題とはなりにくいものです。しかし、転職を考える際に、スカスカな職務経歴は問題となります。スカスカな経歴の理由はいくつか考えられます。

- 病気などにより仕事を休んだり空白期間が多いこと
- ルーチンワーク中心で記載できる内容が限られていること
- 自己学習を怠り、業務でも深い知識や経験が不足していること

筆者自身、これまでの職務経歴において大きな仕事の成果を上げたことはありません。また、働いていない空白期間は合計で数年以上にわたり、この期間に資格取得などもできていません。その結果、筆者の職務経歴書は内容の薄いスカスカなものとなっています。内容の薄い職務経歴書を、マシに見えるようにするテクニックは、インターネット上でも多くありますが、それらを駆使しても薄いままであることに変わりはありません。

筆者が、某大手転職エージェントを利用した際、担当者は職務経歴を見て「年齢の割にアピールできる経験がないですね。この年齢は厳しいです。」と指摘されたこともありました。

このようなスカスカな職務経歴書の場合、転職エージェントが積極的にサポートしてくれないことがあります。彼らも利益を追求しており、高収入を見込める人材に時間を使いたいと考えているからです。

私たちの中には、そのような厳しい状況でも転職をしなければならないことがあります。スカスカな経歴しかない場合でも、選択肢を広げるために役立った3つのことがあります。

日常の仕事を真面目に取り組む

日常の仕事へ真面目に取り組み続けることは、転職の際に役立つことがあります。

筆者が転職が必要になった際、転職していた元同僚や共に働いたことのあるエンジニアや営業の方に誘われたことがあります。彼らが誘ってくれた理由はさまざまでしたが、まとめると以下の2つに集約されます。

- 君の仕事の仕方を知っている
- 必要な技術は学べば覚えることができる

転職においては、高い技術力や過去の成果が評価基準の一部となります。特に転職エージェントのように、初対面の人たちが評価する指標の多くは、技術力や過去の実績が主軸になるでしょう。しかし、私たちと仕事で繋がっている人たちの中には、技術力や成果以外の日常の仕事への姿勢を評価基軸にしてくれる人たちもいます。

直接応募する

転職を希望する組織に直接応募することもできます。直接応募には以下のようなメリットがあります。

- 直接やりとりを行えるため志望意欲を伝えやすい
- 自分のペースで活動を進められる

組織側としても、採用時の成果報酬などの手数料をエージェントに支払う必要がないなどのメリットもあります。直接応募が転職に有利か不利かという議論もありますが、状況と人によるため断定できるものではありません。しかし、職務経験が限られている場合、熱意や志望意欲を直接伝えられる機会は大切です。自己PRや面接でのパフォーマンスに焦点を当てることにより、成功の可能性を高められるでしょう。

過度な要求はしない

世の中には、転職時に収入アップや自己の目標の実現など、複数の要望を実現できる人たちがいます。しかし、筆者が転職活動時に要求することは、そのときに大事な1つだけです。大事なことが労働時間や環境であれば、収入や業務内容は組織の意向を最大限許容します。大事なことが収入アップなのであれば、労働時間や環境は組織の意向を最大限許容します。

組織側も利益になる人材を欲しているのであって、スカスカな経歴しかないにも関わらず、あらゆる要求をしてくる人材は不要と考えるでしょう。

高い技術力と大きな成果を持つことは素晴らしいことです。私たちの未来の選択肢を広げ、転職の成功率を高めます。しかし、それら以外にも私たちを評価してくれる指標はあります。

10.8 独立という考え方

ITエンジニアは、フリーランス^{†1}として独立しやすい仕事だと言えます。近年では、ITエンジニアのフリーランス向けの仲介エージェントも増加しており、専門的な技術とパソコン1台、いくらかの税務知識があれば、明日からでもフリーランスになれるでしょう。

しかし、フリーランスとして独立を考えている場合、少なくとも以下の3つについて熟考する必要があります。

年収と年商の違い

「フリーランスになれば年収が上がる」と表現されることがあります、正確にはフリーランスは年商（1年間の売上）になります。会社員の年収と比較すると金額は高くなることが多いかもしれませんが「年商は売上」です。年商の中から経費・各種税金・国民健康保険・国民年金などを支払わなくてはなりません。経費は控除になりますが、会社員であれば会社が支払っているものです。

また、会社員でも給与所得控除や特定支出控除など使える制度が多くあります。これらを勘案し実質的な手取り額を比較しなければ、後悔することでしょう。

会社員のメリット

会社員のメリットは計り知れません。

- 税金や経費などの事務処理は会社側が実施
- 社会保険（健康保険・厚生年金）の半分が会社負担
- 雇用保険や手当など福利厚生がある
- 繙続的な案件確保・技術知識習得の機会を得ることができる
- 仕事で発生した損害賠償は基本的に会社が負う
- 安定的な収入を得ることができる
- 社会的信用が高い

これらのメリットは、会社員としての働き方に伴うものであり、フリーランスとして独立する際には、これらの利点を放棄することになります。

続けることの難しさ

フリーランスの生存率は低いと言われることもありますが、明確な数値は提示されていません。筆者は資金面から見ると、ITエンジニアとしてフリーランスを続けることは他の業種と比較すると難しいことではないと考えています。人件費や設備投資が圧倒的に少なく、仲介エージェントなど仕事を獲得する最低限の土台があるからです。それでも、筆者がITエンジニアのフリーランスを続けることが難しいと感じることが3つあります。

1つ目は、不安定な収入です。フリーランスの多くはプロジェクトや契約に依存して収入を得ています。契約がない場合、収入は0円になる可能性があります。また、仲介エージェントとの関係も安定的に続くとは限りません。年齢とともに参画できる案件は減少するかもしれませんし、新規顧客を見つけるのも容易なことではありません。特に、家庭を持つなど生活費が増えると、収入の不安定さに対するプレッシャーはかなり大きいものになります。

2つ目は、変化への対応です。システムやITの技術は急速に進化しており、これらの変化に対応するには組織というバックグラウンドがあるほうが、触れたり学ぶ機会は圧倒的に多く、容易でしょう。また「コロナ禍による売上変動」や「インボイス制度」など、ITの技術以外の社会的変化へも対応していかなければなりません。

3つ目は、事業拡大と持続の困難さです。フリーランスになるとITの技術（もしくは時間）の切り売りです。これらは年齢とともに難しくなります。アプリを収益化したり、従業員を雇うなどで事象を拡大し収益を保つ必要がありますが、いずれも簡単なことではありません。

フリーランスという働き方には多くのメリットがありますが、デメリットもあります。そして、これらのメリット・デメリットは時代とともに変化していくものです。

独立を考える際は、現時点と未来のことを考慮し判断する必要があります。

†1 フリーランスとはなんなのか？については、筆者の運営する以下の記事にまとめています。

ITエンジニアのまとめ。【ポンプの戯言】ITエンジニアがフリーランスになる理由. 2021年10月27日
<https://itenginner-matome.net/archives/35373>

10.9 高学歴化への備え

ITエンジニアという職業は、学生たちの間でますます人気を集めています。同時に、多くの組織も優秀なITエンジニア人材の獲得に奔走しています。これからITエンジニア人材になる学生たちの実態を、公的な資料を基に状況を考察してみましょう。

需要から見るIT人材の状況

2019年の経済産業省の「IT人材需給に関する調査^{†1}」によれば、2030年に約16（低位シナリオ）～79万人（高位シナリオ）のIT人材が不足すると試算されています^{†2}。また、新卒 IT 人材の就職率も2017年の6.1%（約3.3万人）から2030年は7.8%（約3.6万人）に増加すると試算されています^{†3}。これらのデータから、IT人材への需要が高く、学生から人気が高いことがうかがえます。

教育から見るIT人材の状況

文部科学省の「令和4年度学校基本調査^{†4}」によれば、18歳の大学(学部)進学は、令和4年で56.6%と過去最高水準であり^{†5}、少子化にもかかわらず大学全体の在学者数も過去最多となっています^{†6}。

また、大学以外の教育面でも「プログラミング教育の必修化」と言われる大きな変化がありました。2020年に小学生、2021年に中学生、2022年に高校生で実施されています。実際の指導内容はプログラミングではなく「情報活用能力の育成・ICT活用」です。高校生で実施される情報Iや情報IIの中には、現場で働く私たちでも知らないような知識や同時に理解できない内容が含まれていると感じます^{†7}。

大学卒業や情報教育を受けたからといって、彼らがIT人材として優秀と判断するのは早計ですが、プログラミングなどを得意とする新卒者が増え、組織もそのような人材を採用することが増えている傾向にあると感じています。彼らは、これからの近い未来に高度な知識やモダンな技術を携え、新人として私たちの前に現れる可能性は高いでしょう。

彼らの登場に対して、私たちは「技術的な備え」と「技術以外の備え」が必要です。

技術的な備え

私たちも彼らに遅れを取らないように、ある程度の技術的な知識をアップデートする必

要があります。ただし、単に教科書的な内容だけを繰り返しても不利になります。教科書的な内容に使える時間は、圧倒的に彼らの方が多いのです。

幸いなことに、私たちはチームが残したナレッジや現場で必要とされる考え方につれての機会に溢れています。特に以下のようなものは学生では触れる機会が少ないものです。

- パフォーマンスや運用性を考慮した設計・実装方法
- 高額なコストが必要な機器やサービスの知識
- レガシーな技術とモダンな技術の両方を考慮すること

また、私たちはこれまでの業務の経験から、使用した技術の進化や変化を少なからず見てきました。この経験は、新しいものを学ぶ際の土台になり、理解を深めたり早める手助けになってくれるはずです。

技術以外の備え

技術的なこと以外でも備えられることがあります。以下のようないことは、仕事でなければ経験が難しいものです。

- 仕事に必要な見積もりの仕方や予算獲得をする方法
- 無理難題な仕事を獲得する営業やPMとの関係を築く方法
- 怒り狂う顧客を落ち着かせ良好な関係を築く方法

これらの方法を完全に使いこなしている人はいないでしょうが、失敗も含めて経験してきました。技術的なことではありませんが、仕事で必要となるスキルです。

高学歴や情報教育を受けた新世代を恐れる必要はないでしょう。彼らは私たちの仲間です。そして、私たちが備えたことを彼らに共有できれば、さらに頼もしい仲間になってくれるはずです。

†1 経済産業省、IT人材需要に関する調査 - 調査報告書2019年3月、2023年9月25日
https://www.meti.go.jp/policy/it_policy/jinzai/houkokusyo.pdf

†2 同資料20ページの内容になります。

†3 同資料9~10ページの内容になります。

†4 文部科学省、令和4年度学校基本調査（確定値）について公表します令和4年12月12日、2023年9月25日
https://www.mext.go.jp/content/20221221-mxt_chousa01-000024177_001.pdf

†5 同資料5ページの内容になります。

†6 同資料2ページの内容になります。

†7 以下のサイトに詳しい内容が掲載されています。

文部科学省、高等学校情報科に関する特設ページ、2023年9月25日

https://www.mext.go.jp/a_menu/shotou/zyouhou/detail/1416746.htm

10.10 未来よりもイマを大切に

ここまで読んでいただいた皆さんの多くが、これからもITエンジニアとしての道を歩みたいと強く願っていることでしょう。これからもITエンジニアの道を歩み続ける人、新たに歩み始める皆さんに、ぜひお伝えしたいことがあります。それは「未来よりもイマ（現在）を大切にしましょう」ということです。

筆者が「イマ」を大切にするために考えている3つのことを紹介します。

喜びの共有

ITの進化とビジネスへの活用の拡大に伴い、システムはますます大規模で複雑になっています。同時に、ITエンジニアとしては、以前は難しいとされていたことも、便利なツールやサービスの登場により比較的容易に行えるようになりました。この変化と共にプロジェクトの規模は小さくなり^{†1}、同時に進行するプロジェクトが増え、新しいプロジェクトが立ち上がるサイクルは早くなっているように感じます。一方で、年間を通してプロジェクトが完了した後に、一息つく時間は減少しているようにも思えます。

また、リモートワークが急速に普及し、出社や外出の減少により対面で話す機会が減少しました。さまざまなプロジェクトが続く中で、対面でのコミュニケーションが少なくなり、プロジェクト関係者が一堂に集まって「喜びの共有」をする機会が減っていると感じます。

このような状況でも、プロジェクトの区切りなどでは「喜びの共有」に積極的に取り組むことが大切だと考えています。成功や困難を共に乗り越えた「イマ」の経験を共有することで、私たちだけではなく、関係者全体の喜びと成長をより豊かにすることができるようになります。

自己の成長の認識

成長はしばしば目に見える変化や大きな達成と結びつけられがちですが「イマ」この瞬間の自己の成長を感じることも大切です。私たちは日々成長しています。その成長は必ずしも技術力を高めることに限らず、新しい考え方や単語を1つ知っただけかもしれません。もしくは、ほんの小さなタスクを終わらせただけかもしれません。しかし、昨日の自分よりも「イマ」の自分の方が、新しい知識や経験（例え失敗の経験だとしても）を得ています。

す。

これらの「イマ」の成長を意識することで、自分の成長の軌跡を認識し、明日へ繋げていくことができるでしょう。

人生は長く、短い

ITエンジニアとしての道を歩み続けると、40年以上という長い歩みになります。技術を身につけることに時間がかかったり、他者よりも成長が遅いことに不安になるかもしれません。しかし、それらの時間は長いITエンジニア人生の道のりの中でわずかな瞬間にすぎません。その不安を感じる前に「イマ」を大切にし、知らないことを学び、仕事という挑戦の中で楽しさを見つけましょう。

一方、人生全体を考えると、短すぎるかもしれません。友人や家族などの大切な人たちと楽しく幸せな時間を過ごすことは大切ですが、そのような関係を築くには多くの時間を必要とします。また、楽しく幸せな時間はあつという間に過ぎ去ります。仕事に多くの時間を費やすことで、これらの大切な瞬間を逃すことのないようにしましょう。

筆者は、本書の教訓に気づくまでに多くの間違いをし、大切だったはずの「イマ」を多く失いました。現在でも後悔することはあります。しばしば、教訓を忘れ同じ過ちをおかしてしまうこともあります。そのときは、これまでに紹介した教訓を振り返るとともに「イマ」をどうするべきか考えています。

「イマ」を大切にし意識的に選択をすれば、より良い未来へ道を歩んでいくことができるでしょう。もし、間違った選択をしてしまった場合も、意識的に自分で選択した道であれば後悔は少なくなるはずです。

私たちの未来は無限ではありませんが、多くの選択肢が広がっています。未来で何を選ぶかは私たちの考え方と行動にかかっています。そして、出発点は過去でも未来でもなく「イマ」であることを忘れてはいけません。

†1 本節の「小さくなる」とは、1つのプロジェクトを行うメンバーの数のことを指しています。筆者の感覚でしかありませんが、システムに関する技術的難易度や予算は大きくなっていますが、それらを担当するチームの人数は少なくなっているように感じています。

付 章

ここまで教訓は、多くの人たちの助言や
叱咤激励の言葉から生まれました。

本章では、教訓が生まれるきっかけになった言葉と
その言葉を得たときの状況を紹介します。

第Ⅰ章 無能のための考え方

第Ⅱ章 無能のためのコミュニケーション

第Ⅲ章 無能のためのプレゼンテーション

第Ⅳ章 無能のための技術

第Ⅴ章 無能のための学習

第VI章 無能のための時間管理

第VII章 無能のためのチームワーク

第VIII章 無能のためのマネジメント

第IX章 無能のためのプライベート

第X章 無能のための未来

第Ⅰ章 無能のための考え方

1.1 仕事は問題を解決すること

＜仕事ってなんだと思う？＞

新人時代に先輩と居酒屋で飲んでいるときに、唐突に質問されました。当時の筆者の回答は「技術を使って利益を出すこと」だったと記憶しています。

1.2 自身の責務範囲を知る

＜お前の仕事なんだから＞

筆者が自分の範囲ではないと勝手に判断し拒否したときに、先輩から叱責されました。当時の筆者は、言われたことを行うことに精一杯で、自分の責務範囲を考えることを放棄していました。

1.3 責務範囲のモレとダブり

＜自分を守ることに必死だな＞

トラブル調査で、自分の技術領域を調べ尽くし別のチームへ丸投げした後、ほっとしている筆者を横で見ていた先輩から、落胆した声で言われました。丸投げされた方の気持ちを考えられていませんでした。

1.4 限りなく最適解に近いもの

＜それが一番いいんですか？＞

設計レビューの際に、PMから質問をされました。その時の設計は、正しくあろうとして他の影響範囲を考慮できていないものでした。振り返ると、その正しさも稚拙なものでした。

1.5 役割と上下関係

＜正しいと思ったことを言えばいい＞

若手メンバーとしてプロジェクトへ参画する際に、見送ってくれた上司が送ってくれた言葉です。当時の筆者は、言葉通り受け取り、好き放題言ってしまい、多くの敵を作ってしまったのは苦い経験です。

1.6 ウォーターフォールがもたらした最悪のもの

＜面倒だから黙るのか？＞

実装の途中に設計の間違いに気づき、設計者に修正を依頼するか先輩に相談したときに言わされました。当時の筆者は、設計者に文句をつけるようで伝えることを躊躇していました。

1.7 正しいモノサシを持つ

＜自分の価値は常に意識しつけよ＞

メンバーとして参画しているときに、同じチームのPLから言わされました。当時の筆者は、自分の技術力が、どの程度かすら把握できていませんでした。

1.8 水はモノサシで測れない

＜そんなの当たり前だろ＞

初めて関わる部署と一緒に仕事を行い、「当たり前」とされることを質問したときに叱責されました。質問したこと事態に後悔はありませんが、事前に調べたり考えることはできたのは事実でした。

1.9 究極の評価ではない

＜お前の残業時間無駄だな＞

毎晩遅くまで残業をしても、タスクが終わらない中で残業に文句をいう筆者を見た先輩から言わされました。ひどく傷つきましたが、仕事が一番遅かったことは事実でした。

1.10 不満を垂れる前に行動を

＜さっさと動け＞

慣れない現場であたふたとしている時に、叱責された言葉です。当時の筆者のスキルでは、難しいことではありましたが、この言葉によって「ボーッと待つ」ことは無くなりました。

第II章 無能のためのコミュニケーション

2.1 すべてのはじまり

＜で？ナニ？＞:(>

常に不機嫌そうな先輩に質問する際に毎回怒ったように言われました。彼が怒った雰囲気を出しているだけで、本当は怒っていないことに気づくのに時間がかかりました。

2.2 事実と感情

＜君はどう思う？＞

筆者があやふやな質問や話をする際に、多くの先輩から言われました。相手がどのように受け取るか考えずに好き勝手に話す傾向がありました。

2.3 コミュニケーションの種類

＜日本語下手くそか？＞

話すことも文章を書くことも苦手だったころに、よく叱責されていました。日本語を組み立てることが苦手だったため、相手をイライラさせることが多かった時期です。

2.4 一方通行は存在しない

＜これはどういうこと？＞

何かのレビューを行うときに、多くの人たちから言われたことです。当時の筆者は、相手は何を知りたいか？を全く考慮できていませんでした。

2.5 コミュニケーションに終わりはない

＜そういえば、あの件だけど・・・＞

何年も前に対応したことを唐突に質問された際の言葉です。思い出すのにかなりの時間を要しました。

2.6 ありがとう、ごめんなさい

＜まずは感謝しろ。謝るのはその後だ。＞

詳細は忘れてしまったのですが、何かの書籍の一文です。「日本人は謝るのは得意だけでも、感謝は苦手だ」というようなことを読んだときに、筆者自身にも当てはまる強く感じました。

2.7 怒っているのではなく、困っている

＜どうなってるんです！？>(>

多くの人から頂いた言葉です。筆者は怒られることが多かったので、嫌というほどこの言葉を聞きました。

2.8 綺麗な格好ではなく、似合う格好を

＜そのスーツ、サイズ合ってなくね？＞

自分の体型に合っていない、ダボダボのスーツを着ていたときに先輩から言われた言葉です。当時は適当に安いスーツを選んでいました。

2.9 リモートワークで失うもの

＜では、そういうことで。＞

Web会議が多くなり、締めの言葉としてよく聞くようになりました。筆者は寂しがり屋のため、その言葉を聞いてすぐミーティングが終わることに寂しさを感じます。

2.10 自分の時間、他者の時間

＜それは自分で考えろよ＞

若い頃、すぐに最終的な回答を求めようと質問を繰り返すことの多かった筆者に、先輩が叱責した言葉です。

第Ⅲ章 無能のためのプレゼンテーション

3.1 プrezentーションは自己紹介

＜君の考えを聞きたいな＞

筆者が、文字を読み上げるだけのプレゼンテーションをしていたときに諭された言葉です。当時の筆者は、間違ったことを伝えないように、必死に文字ばかりをみて、周りの人たちを見ることができませんでした。

3.2 文章の前にストーリーを考える

＜何を伝えたい？＞

筆者のメールを見た先輩が問い合わせてきた言葉です。上から読んでいくメールの性質を無視してメール文を作成していました。

3.3 大きさは2倍、早さは半分

＜ん？なんて？＞

プレゼンテーションが苦手で、マイクを使っても声が小さいころによく言われました。一時期は話すことには嫌気がさしていました。

3.4 立った！立ったわ！

＜お前の説明眠くなるんだよ＞

筆者が、単調な話し方しかできないときに同僚から得たフィードバックです。聴衆を楽しませたり、聞いてもらうための観点がありませんでした。

3.5 ホワイトボードは怖くない

＜書けよ＞

筆者が、少々面倒な技術的な内容を、言葉で必死に説明しているときに諭された言葉です。それからは、紙とペンを持ち歩くようになりました。

3.6 失礼という誤解

＜ちゃんと言いたいこと言えた？＞

急遽出席した会議で、口数少ない筆者に先輩が問いかけてきた言葉です。しどろもどろで自信がないことを気づかれていました。

3.7 基本はみっつ

＜情報整理できないの？＞

設計書を作り慣れていない同僚が言っていた言葉です。どのように作ったら見やすいだろうか？を考え始めて色々なテクニックの書籍を読んだ結果、3つを指標にすることに行きつきました。

3.8 内職はしない

＜・・・え？もう一度お願ひします＞

会議時に質問をよく聞き返す人を発見しました。よく見ていると会議中に画面を見てキーボードを打っていました。

3.9 長時間会議は悪か

＜時間が来たので今日はおしまいで＞

時間の決まった会議の締めで聞くことがありました。何も決まっていない状態で終わる会議に疑問を感じていました。そして、自分が会議を主催する際にも同じような言葉を使ってしまうことに嫌気がさしていました。

3.10 全員に愛される必要はない

＜誰に伝えたいの？＞

全員に理解して欲しいと考えて、特にアピールポイントがない資料をレビューしているときに指摘してもらった言葉です。

第IV章 無能のための技術

4.1 はじめてのものには、畏怖と敬意と楽しさを

＜楽しめない人とは仕事したくないね＞

とあるプロジェクトで、メンバーが主張していた言葉です。当時の筆者は、初めて触れるものには畏怖しか感じていませんでしたが、楽しむ必要もあると考える機会になりました。

4.2 動いた感動、理解した感動

＜プロなんだよね？＞

あまり技術の理解をしていなかった筆者が、顧客からの質問に答えられなかつたときに叱責された言葉です。

4.3 点で捉える

＜馬鹿でも暗記くらいはできるだろう？＞

技術の関連性を繋げられないことを相談したときに、先輩から激励された言葉です。確かに時間をかけねばある程度の知識を蓄えることは可能でした。

4.4 全く同じは存在しない

＜ちゃんと検証してるの？＞

とある作業に失敗して、顧客から叱責された言葉です。検証はしていましたが、考慮すべき違う点を考慮できていなかつたため、モヤモヤした気持ちだったことを覚えています。

4.5 おまじないという嘘

＜これはおまじないだから＞

C言語に触れたときに初めて聞いた言葉です。C言語ではありませんが、仕事の中で「おまじない」を信じたために、手痛い思いを経験してからは、この言葉を全く信用しなくなりました。

4.6 できないは出発点

＜それで？＞

筆者が、できない理由をひたすら説明し終わったときに、聞き返された言葉です。当時の筆者は、できない理由だけ考え、できるようにする方法を考えられていませんでした。

4.7 必要なものはモダンではない

＜うんうん。で、解決できるの？＞

とあるモダンな技術を採用したプロダクトの説明をした後に聞き返された言葉です。そもそも問題を把握できていなかった筆者は、的確に回答することができませんでした。

4.8 偽りのフルスタックエンジニア

＜俺、フルスタックだから＞

SNSなどでよく聞く言葉です。実際の案件でも、このような人を見ることはありますが、筆者はフルスタックと言われるより、出来ること出来ないことを話してくれる人に信頼を寄せます。

4.9 私たちは失敗する

＜ありえねーだろ！＞

筆者が失敗したときに叱責された言葉です。振り返ってみても、ありえない失敗でした。それは、事前に数十分ほど確認することで回避できる失敗でした。

4.10 未経験者と経験者の違い

＜まだまだだな＞

尊敬するITエンジニアが、その人自身の仕事の成果に対して評価した言葉です。当時の筆者は、それ以上何を求めるのか理解できませんでしたが、現在では経験者としての高みを目指していたことがわかるようになりました。

第V章 無能のための学習

5.1 模倣から始める

＜真似でも身につけたらお前の力だよ＞

自分の強みがなく悩んでいたときに、恩師が激励してくれた言葉です。当時の筆者は、独自の強みがあることが素晴らしいと考えていました。現在は、周りから模倣できることをすぐに見つけることができるようになりました。

5.2 資格だけでは身を滅ぼす

＜あの人資格持ってるから・・・＞

とあるメンバーが未経験の役割になったときに、なぜその役割を与えられたのかをPMに質問したときに得た言葉です。そのメンバー以外は誰も知見がなかったために、資格を持っているという理由でアサインされていました。

5.3 暗記ではなく、できることを知る

＜オプション？そんなの調べりやわかるだろ＞

筆者が、Linuxコマンドのオプションの暗記を試みていたときに、先輩から指摘された言葉です。今ではオプションを暗記しようなどと思うことは、ほとんどありません。

5.4 知らないを知る

＜知らないことを調べることはできないよ＞

どうすれば早く目的のことを調べることができるのか？と先輩に質問したときに得た言葉です。知らないことを知っていることは、知っていることと同じくらい大切であることに気づく機会になりました。

5.5 学びは歴史とともに

＜俺が○○歳くらいの頃はな～＞

上司や先輩と居酒屋へ行ったときによく聞いた言葉です。参考書に載っていないようなマニアックな知識や生々しい体験を聞くことができました。

5.6 学習媒体を使い分ける

＜それ、どこ情報？＞

調査結果などをまとめて報告したときに、確認のために聞かれた言葉です。情報の信頼性と鮮度の大切さを学ぶ機会になりました。

5.7 情報源は2つ以上

＜俺の言うことが絶対正しいとは限らない＞

先輩の口癖でした。その先輩自身は優秀なITエンジニアでしたが、疑問に思うなら他の人にも教えてもらうことも推奨していました。疑うことの大切さに気づく機会になりました。

5.8 良書との出会い方

＜これは良い本なんだよ＞

一般的な良書と言われる技術書を紹介してくれたときの言葉です。その技術書は難解すぎて、筆者の役に立ちませんでした。そのときに「自分にとっての良書ってなんだろう」と考え始める機会になりました。

5.9 手取り額の10%

＜趣味だし＞

ITの技術に関することに多くのお金をつぎ込んでいる人に「勿体なくないか？」と質問をしたときに得た言葉です。「趣味と考えれば、そこまで高いわけでもないか」と妙に納得したことを覚えています。

5.10 検索エンジンへの聞き方

＜ここに書いてるよ？ちゃんと調べた？＞

筆者が数時間調べてもわからなかつたことを先輩に質問し、10分ほど検索して回答に繋がるヒントを与えてくれたときの言葉です。愕然としましたが、調べる方法の大切さに気づきました。

第VI章 無能のための時間管理

6.1 時間管理術ができない理由

＜今日が終わる・・・＞

一時期、筆者が頭の中で繰り返していた言葉です。仕事の進捗がないのに終電の時間が迫ってくることに怯えて過ごしていました。

6.2 予定は日で考える

＜1時間もあれば終わるかな＞

先輩にとあるタスクにどの程度の時間がかかりそうか質問したときに得た言葉です。1時間と聞いていたタスクが1日かけて終わらなかつたときは泣きそうになりました。

6.3 目的と期限の明確化

＜よしなに＞

とあるプロジェクトで急遽発生したタスクのすべてに「よしなに」がついて頼まれることがありました。途中からタスクが溢れ出し、自分が主体になって明確化していく必要性に気づくことができました。

6.4 ズレを許容する準備

＜延長は認められません＞

すでに決まっているタスクの期限を伸ばそうと交渉したときの言葉です。この交渉をもつと早く行なっていれば延長も可能でしたが、そのときは判断が遅すぎました。交渉は可能な限り早くする大きさを学ぶ機会になりました。

6.5 実績と私のスケジュール

＜前任者はこれで終わったから今回も同じ感じで＞

とあるタスクを依頼されたときに言われました。当時の筆者はそのまま信じたところ、前任者の2倍以上の時間を要し、関係者に謝りながらタスクを処理していました。

6.6 空き時間なのか、待ち時間なのか

<ちょっとトラブルったから見て！>

複数のメンバーと共同で作業しているときに、筆者の作業が終わって待っているときに得た言葉です。当時の筆者は自分の作業が終わったことに安堵し、周りを見えていなかったためトラブルの発生状況を確認するために多くの時間を使いました。

6.7 私の集中できる時間

<早朝出社はメリットだらけだ>

ニュース記事から得た言葉です。教訓にも記載している通り、共感はできませんでしたが時間について考える機会になりました。

6.8 浪費時間？私には必要です

<なんでこんなに時間かかるの？>

仕事の中でよく問われた言葉です。理由を明確に説明できないときは、指摘だらけでしたが、理由を説明できれば納得してくれる人も多くいることがわかりました。

6.9 ヒトは1週間、モノは1ヶ月

<自分中心で世界が回ってると思ってる？>

筆者が忘れていたタスクを他者に依頼したときに叱責されたときの言葉です。依頼の仕方も悪かったですが、他者のスケジュールを考慮しないことが、どれほど相手を怒らせるか知る機会になりました。

6.10 残業は悪か

<無駄な残業など存在しない>

筆者の残業時間が多すぎることを指摘されている中で、先輩から励まされたときの言葉です。このおかげで心折れず乗り越えることができました。

第VII章 無能のためのチームワーク

7.1 同じ方向を向いて進む

＜そんなことも把握していないのかよ＞

思いついた疑問点をすべてPLへ質問して叱責された言葉です。全てを聞くのではなく、自分自身で確認をする大きさを気づく機会になりました。

7.2 チームワークはメンバーから始まる

＜誰もいうこと聞かないチームだな＞

PMが、ぼそっと愚痴をこぼしていたときの言葉です。チームワークをPMだけに任せるのは間違っているのではないか？と考え始める機会になりました。

7.3 局所的リーダーシップ

＜得意なものない？＞

得意な技術や分野がない筆者が問われた言葉です。当時は「ないです・・・」と言う回答しかできませんでしたが、自分が貢献できることを探し始める機会になりました。今も得意なものはありませんが、役に立てることは少なからずあります。

7.4 教えてを言う勇気

＜はい、知っています（本当は知らない）＞

筆者がタスクを依頼したときに、その技術を知っているか確認したときに得た言葉です。信じてタスクを依頼したところ、本当は知らず期限直前になって助けを求められた苦い思い出があります。

7.5 誤った協調性

＜・・・（無言）＞

確認を行った際に無言だった人が、問題につながるとわかった途端に「やっぱり〇〇すればよかった」と言い始めたことがあります。確認の際に言ってくれればと、酷くイライラしました。

7.6 柔軟性と譲れない範囲

＜いえ、私の範囲じゃないので＞

問題を相談する機会を設けるために、とあるメンバーに声をかけたときに得た言葉です。それは明らかに彼の範囲でしたが、最後まで納得してもらうことはできませんでした。

7.7 経験年数で判断しない

＜PMはじめました＞

とあるプロジェクトのPMの自己紹介で聞いた言葉です。筆者は酷く不安になったことを覚えています。プロジェクトは苦難の連続でしたが、PMは尽力してくれる真面目な人物で楽しい日々を過ごせました。

7.8 違和感を大切にする

＜そんなこと後でいいだろ＞

筆者が、違和感を突き止めるための調査を行っていることに気づいた先輩から叱責されたときの言葉です。相談なく判断し、本来のタスクも進んでいなかつたため、彼が怒るのは当然のことでした。

7.9 横目と仰望

＜じゃ、あとはお願ひね！＞

とあるメンバーが、案件を一時的に離脱するときに、その仕事を丸っと渡されたときの言葉です。当時の筆者は、自分のことしか見えておらず、急に渡された仕事の内容を理解するのに苦労しました。

7.10 面倒なことは、みんな面倒

＜お前、なんの役に立ってんの？＞

技術で貢献できていない筆者に問いかけられた言葉です。すぐに技術で貢献することは難しかったため、見つけた雑用をひたすら処理するようになりました。

第VIII章 無能のためのマネジメント

8.1 やはり銀の弾丸などなかった

＜あの人って難しいよね＞

チームの中の雑談で聞いた言葉です。筆者は「あの人」を難しい人だとは感じていなかつたのですが、他の人からすると難しく感じることを知り「人の難しさ」とはなんだろうかと考える機会を得ました。

8.2 完璧を捨てる勇気

＜その技術ってどういうことなの？これは？これは？＞

技術的な内容を全て理解したい思考のPMから言われたことです。理解を示す姿勢は嬉しいものですが、理解してもらうまで事細かに説明する時間が毎回発生し、本来のタスクを行う時間にまで影響することもありました。

8.3 不満は開始時からはじまっている

＜何が不満なの？＞

PMから問われた言葉であり、筆者自身がPMのときメンバーに問いかけた言葉でもあります。不満は至る所で発生していることを考える機会になりました。

8.4 些細な苦情はヒント集

＜また、面倒なこと言い出しやがった・・・＞

筆者がマネジメントの役割のとき、さまざまな問い合わせをもらったときに心の中で呟いてしまった言葉です。面倒と思ってしまったがために、対処できたはずの問題の原因を見逃してしまったことがあります。

8.5 嫌われながら好かれる

＜もう一緒に仕事したくないです:-)＞

筆者がマネジメントをしたプロジェクトが終わったとき、メンバーから笑顔で貰った言葉です。彼は、筆者の仕事の仕方に常に文句を言っていましたが、プライベートでは頻繁に遊ぶほど仲良く過ごしていました。

8.6 使えないメンバー

＜あいつは使えねえな＞

筆者がマネジメントをしているプロジェクトのメンバーから上がった言葉です。当時の筆者の力不足により、使えないメンバーと言われた人物を守ることができなかつたことは苦い経験です。

8.7 確認、確認そして確認

＜いや、そんなの知らないけど＞

筆者がメンバーのときPMから受けた言葉です。課題管理には起票していましたが、認識されていませんでした。筆者自身がマネジメントを行う際は、確認は常に行おうと心に決めた瞬間でした。

8.8 自分の限界を知る

＜限界なんで休みます＞

プロジェクトに耐えきれない多くの人から聞いた言葉であり、筆者自身も使った言葉です。限界を超えることは何も良いことがないことを経験しました。

8.9 顧客の声は誰のもの

＜お客様から感謝の言葉をいただきました。みんなありがとう！＞

筆者がメンバーのとき、PMからチーム全体に伝えられた言葉です。当時の筆者は、PMから感謝が伝えられることを嬉しく感じ、真似をしようと考えました。

8.10 マネジメントとメンバーの兼任

＜マネジメントもメンバーも不足してるからね＞

筆者がマネジメントとメンバーを兼任したとき上司から送られた言葉です。現在でもマネジメントとメンバーが足りているプロジェクトは少ないと感じます。

第IX章 無能のためのプライベート

9.1 ITエンジニア、恋愛のすゝめ

＜結婚したいんですよね＞

平日も休日も、多くの時間を仕事に費やす恋人のいない同僚が日々呟いていた言葉です。恋人ができたら少しは仕事を抑えるのか疑問に思っていましたが、筆者が退職するときも、がむしゃらに働き続けていました。

9.2 休日に勉強はするべきか

＜出来るならしなくていいんじゃない？＞

筆者が若い頃に、先輩に「休日に勉強したほうがいいのか？」を質問した際に得た言葉です。彼は休日にはまったく勉強をしないタイプでしたが、筆者には必要でした。

9.3 そこに信号機があります

＜日常にはアルゴリズムが溢れている＞

詳細は忘れてしまったのですが、技術系の書籍の中で発見した一文です。筆者は、仕事に関連することを、そのまま考えるのが苦手なため、日常の中から役立ちそうなことを探すようになりました。

9.4 休むという強い覚悟

＜なんでそんなに働くんです？＞

一緒に働いた新人から問われた言葉です。当時の筆者は、終電まで仕事をすることが多く、新人は定時で帰っていたため不思議に感じたそうです。1年後くらいには、その新人の残業時間は筆者を超えており、同じ問いを彼に投げかけたところ苦笑いしていました。

9.5 お風呂と通勤と私

＜時間がねえ＞

多忙な時期に、筆者が心の中で常に呟いていた言葉です。明日が来ることが怖くて毎日震えて過ごしていました。

9.6 ワークライフバランスの幻想

＜IT業界もホワイトになりましたね＞

飲み会で同僚と話している中でよく出てくる言葉です。昔の苦労を笑い話にしながら、徹夜などが少なくなった喜びを分かち合っています。

9.7 休み中の待機という矛盾

＜何かあつたら連絡するから＞

作業の結果待ちをするときによく聞く言葉です。筆者はこのようなとき、不安でプライベートなことが手につかなくなることに早い段階で気づきました。

9.8 趣味という逃げ場所

＜来週もまた仕事か・・・＞

筆者の趣味がIT関連のみで、休日がなかったときによく心の中で呟いていた言葉です。毎週この言葉をつぶやくのが嫌で、IT関連以外の趣味を探すようになりました。

9.9 気の置けない仲間たち

＜これ以上仕事をするなら離婚です＞

筆者が壊れかけていたときに、妻から伝えられた言葉です。妻の言葉によって、自分自身の心と家族を完全に壊さずに済みました。また、仕事を最優先にすることもなくなりました。

9.10 他者と比較する前に自分の充実を

＜なんでこんなに頑張ってるのに出来ないんだ＞

いくら頑張っても仕事ができないことに苦悩していた筆者が、心の中で呟いていた言葉です。途中からは出来ないことよりも、なぜこんなに苦しいのかと考えるようになり比較することが原因と気づくことができました。

第X章 無能のための未来

10.1 成長による停滞、成長なき後退

＜働きたいなら成長し続ける＞

筆者が新人のころに、先輩から助言してもらった言葉です。成長できない自分を苦しめる言葉にもなりましたが、どこまで成長すれば良いのか考える機会を多く与えてくれた言葉でもあります。

10.2 無くなるのではない、変わるもの

＜私の仕事がなくなります＞

組織的な体制が大きく変わると同時に、同僚がこぼした言葉です。当時の筆者も仕事がなくなることに怯えていましたが、結局のところ目の前の仕事を処理し続けるしかないという考え方になりました。

10.3 バズワードの危険性

＜○○エンジニアの年収が高い！＞

同僚が転職する際に理由を聞いて得た言葉です。彼はバズワードにあやかり転職に成功して、楽しく暮らしているようです。筆者は、彼のような技術力もバイタリティもないため、おそらく失敗していました。

10.4 具体的未来は考えない

＜5年後、どうなっていきですか？＞

転職の面談や人事評価のときに、よく聞いた言葉です。筆者は長いこと体裁を繕った回答しかできませんでした。今でも具体的な回答はありませんが、自己の中で指針を作ることができました。

10.5 35歳定年説は崩れたものの・・・

＜徹夜かあ・・・＞

年齢を重ねて徹夜が決まると同僚と顔を見合せながらこぼしていた愚痴です。体力を頼りにした働き方が出来なくなっていることを、実感しています。

10.6 転職の葛藤

＜うちに転職しない？＞

筆者が転職の誘いを受けたときの言葉です。多くはフランクに勧誘をしてきて、どれも魅力的な内容でした。転職をするときに考えることを整理する機会を得ることができました。

10.7 スカスカな職務経歴書

＜見てる人はちゃんと見てるよ＞

こちらも筆者が転職の誘いを受けたときの言葉です。空白期間が長く自信を喪失していたため、この言葉に救われました。

10.8 独立という考え方

＜独立したら年収増えていいですよね＞

筆者がフリーランスになってからよく聞く言葉です。独立=年収が上がると考える人が意外なほど多いことを知ることが出来ました。

10.9 高学歴化への備え

＜え？大学出てないんすか？＞

どこの大学出身か？と言う話の中で若手エンジニアから出た言葉です。周りを見回すと大学卒業者の若手ばかりなことに気がつきました。いつか大学に通いたいと考えていますが、まだまだ先になりそうです。

10.10 未来よりもイマを大切に

＜幸せってなんだと思う？＞

若い頃に居酒屋で同僚と呑んでいるときに問われた言葉です。当時は、筆者も同僚も高収入であることが幸せだと考えていました。現在では、高収入よりもイマが楽しければ幸せと考えています。

あとがき

本書では、筆者の経験をもとに「無能なITエンジニアが、無能なまま生き続けるために必要な100の教訓」を紹介しました。

これらの教訓を綺麗事や理想論と捉える人もいるかもしれません。教訓の中には、多くの時間が必要になることや精神論的なもの、ときには諦めが必要なことも含まれているため、人によっては不快に思う教訓があることは否定できません。

また筆者自身、常にすべての教訓を実施できているわけではありません。教訓に出てきたことでも状況により機微が変わってくるため、再度悩むことがあります。それでも、教訓を思い出すことにより問題解決の助けになっています。

教訓の中から、1つでも皆さんの助けになることが見つかれば、本書としての役割は果たせたのではないかと考えています。

ここからは、本書で書ききれなかった2つのことについてお伝えします。

なぜ無能なままなのか？

本書を読んで「なんで無能なままなの？」と思われた読者もいるでしょう。その理由は、これらの教訓には無能を脱却するために必要となる「人並み以上の成果を出す」という要素が欠落しているからです。これは、技術の理解や習得が遅く、人並みの成果も出せない筆者が生き残るために、以下を指針にして教訓を考えたためです。

- 無理はことはしない
- 時間をかけければできる
- 損失をださない

「無能なまま」を異なる言い方になると「人並みの成果は出せないが、組織やチームに存在を認めてもらえる」ことを主軸としています。無能を脱却するためには、これらの教訓とは別の次元で人並み以上の成果に繋げるための思考と行動が必要になります。残念ながら、筆者はその方法を未だ見つけられていません。

教訓の原点

無能なままで生き続ける方法を模索し始めたとき、指針とともに全ての教訓を考える原点がありました。その原点は、P.F.ドラッカー氏の「真摯さ（integrity）」です。ドラッカー氏は、経営やマネジメントの分野で有名ですが、彼の言う「真摯さ」が私たちの仕事でも役立つことに気づきました。特に以下の文が原点になりました。

真摯さは、とつてつけるわけにはいかない。すでに身につけていなければならない。
ごまかしがきかない。ともに働く者、特に部下に対しては、真摯であるかどうかは
二、三週間でわかる。無知や無能、態度の悪さや頼りなさには、寛大たりうる。
だが、真摯さの欠如は許さない。決して許さない。^{†1}

この文はマネジメントについてのものですが、筆者に「無能であったとしても真摯さを考え続ければ、生き残る方法が見つかるのではないか」という希望を与えてくれました。彼は「真摯さ」について以下のようにも述べています。

マネージャーにできなければならないことは、そのほとんどが教わらなくとも学ぶことができる。しかし、学ぶことのできない資質、後天的に獲得することのできない資質、始めから身につけていなければならない資質が、一つだけある。
才能ではない。真摯さである。^{†2}

彼は「真摯さ」は後天的に獲得できないと述べていますが、筆者は異なる考えを持っています。彼の言うような「真の真摯さ」は獲得できないかもしれません、筆者は真摯さに近いことを周囲にいる上司や先輩たちなどの多くの言葉により学ぶことができました。私たちが仕事で必要となる「真摯さ」は後天的に獲得できるものだと信じています。

私たちがIT業界で生き残り続けるのは簡単なことではありませんが、筆者は現在のところ生き残っています。これからも生き残るために、本書の教訓を振り返ることでしょう。

†1 P.F. ドラッカー. マネジメント【エッセンシャル版】—基本と原則. 2001年12月. 147ページ

†2 同書籍 130ページ

無能なボンブ (むのうなぼんぶ)

ITエンジニアを目指して田舎から上京し、自社開発会社、SIerなど数社を経験したのち、心の病に倒される。寛解後は再発を恐れながらフリーランスのITエンジニアとして幅狭く活躍中。可能な限り頑張らずに楽しく生きることに専念がある。

無能なITエンジニアのための100の教訓

2024年1月1日 初版 第1刷発行

著者	無能なボンブ
装丁	無能なボンブ
本文デザイン	無能なボンブ
レイアウト・図版	無能なボンブ
編集	無能なボンブ
発行者	無能なボンブ

本書の一部または全部を著作権法の定める範囲を超えて複写、複製、転載あるいは
ファイルに落とすことを禁じます。