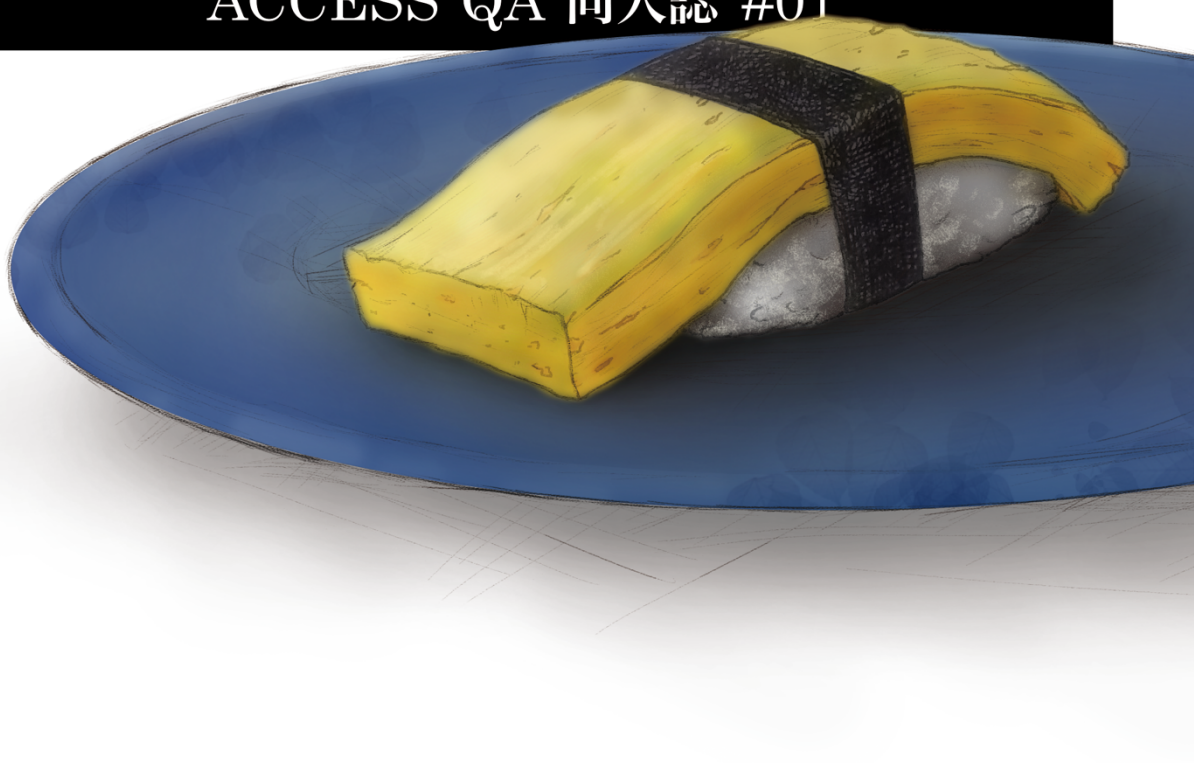
Tamago, kudasai!

異世界転生したら最強テスターになっちゃった

ACCESS QA 同人誌 #01



ACCESS

Quality

Assurance

目次

プロローグ

――――「どこだろう、ここ…」

目を覚ますと、知らない場所にいた。私は辺りを見回そうとして重大なことに気づく。
「に、人間になってる！？」
私はスライム。…だったはずだ。冒険者に殺された…はず…なのに。生き返った？人間として？体を切り裂かれたショックからか歯抜けになった記憶を必死であさる。物理的にも頭をかき回していると、突然背後から声がかけられた。
「あっ、探したよ、羽宮さん！」
駆け寄ってくるのは軽装の人間だ。残念ながら知り合いではない、はずだ。
違う、と言う前に手を掴まれ、引きずるように連れていかれる。
「駄目だよ、勝手に出て行ったりしちゃ！」
どうも怒っているようだ。生き返った途端殺されるようなことにならなくて良かったが。
相手の方が立場が上らしいので、とりあえず謝っておく。
「すみません…」

密集した建物の間を抜け、重そうな鉄の扉から屋内に入る。立つのがやっとの小部屋に入ると、ようやく手を放してもらえた。
名前も知らない相手はため息をついてこちらを見る。
「疲れてるかもしれないけど、今日は残業お願いすることになりそう」
「ざんぎょう…？」
「そうだよ、明日リリースだからね。今の消化具合だと全然終わらない」
（残業ってなんだろう）
分からない単語ばかりだが、目の前の相手に聞いていいものだろうか。
「あの…」

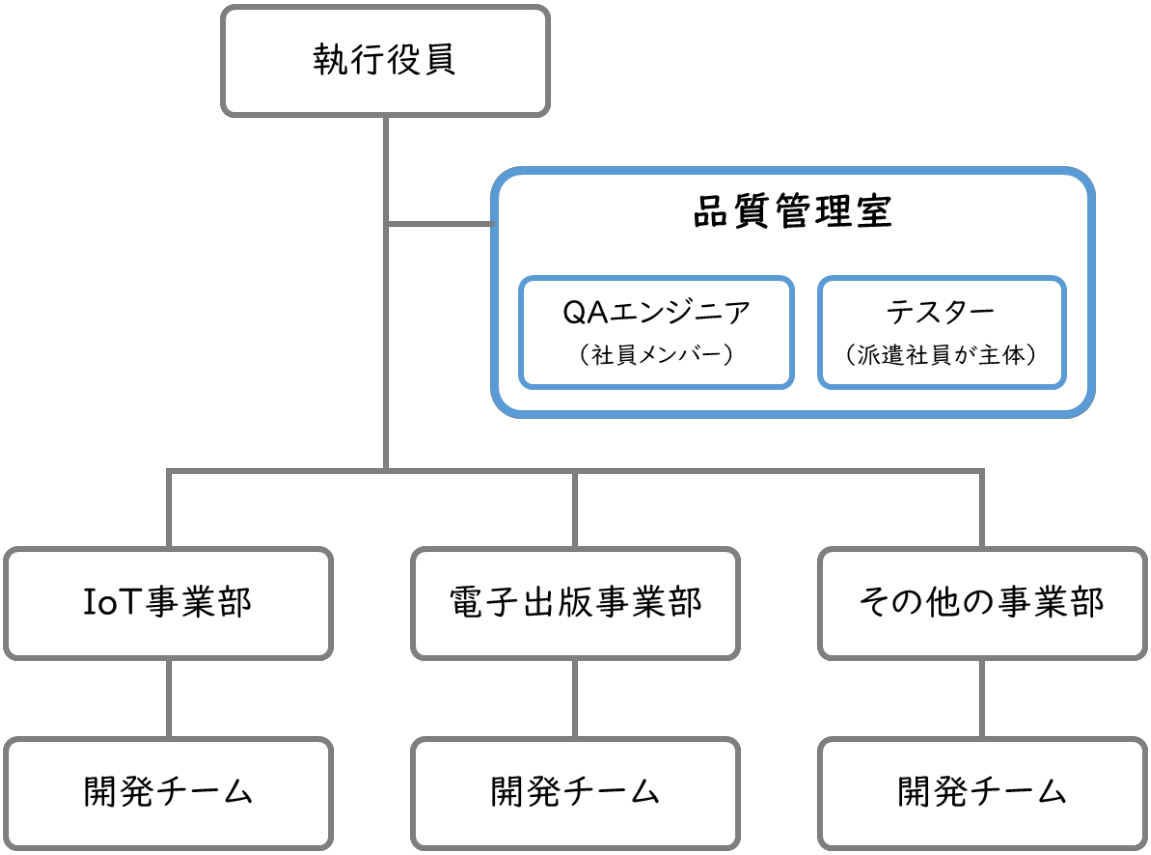
――――ピンポーン。

間延びした音と共に体にかかっていた荷重が消える。目的地に到着したらしい。
眩しく光る天井に灰色の床。たくさんの机と謎の装置。
一番近くにある椅子を引いて、相手が言った。

「さぁ座って、テストをお願いします！」

品質管理室にようこそ

株式会社 ACCESS の品質管理室（QA チーム）は、開発チームとは独立した組織となっていて、品質に関するあらゆる業務をしています。大勢のテスターさんを抱えているのも特徴です。秋葉原と神田にオフィスを構え、リモートワークも活用して働いています。



品質管理室の住人たち

個性豊かな社員メンバーを好きな寿司ネタとともに紹介します。

Makoto Wakabayashi

チーム一開発力がある
一人称が「ワシ」
に聞こえる

推しネタ：のどぐろ

Dai Matsui

とにかく仕事が早い
開発者に辛いものを
お見舞いする

推しネタ：トビッコ

Tsukasa Yanagi

QA チームの委員長
ライブに行きすぎて
業界人と間違えられる

推しネタ：いなり寿司

Shirou Asai

タフな案件をこなす戦士
彼女にアキバおつかい
ミッションを頼まれがち

推しネタ：納豆巻

Tatsuya Yamamoto

頼れるリーダー
見た目は若いが夕方に
なると疲れが顔に出がち

推しネタ：中トロ

Fusami Jono

磨き上げたスキルで
ざくざくバグを出す
長崎のレジェンド

推しネタ：ガリ

Kasumi Suzuki

EPUB と git の伝道師
社内外の各所に首と足
を突っ込んでいる

推しネタ：甘エビ

Koji Shiraishi

大ベテラン
48 時間走れるし
100km だって走れる

推しネタ：カンパチ

Akiko Murakami

自動テストやテスト
ツールを使いこなす

推しネタ：カンパチ

Maki Yoshida

貪欲に知識を吸収中
4 コマ漫画家の
よしだま先生である

**推しネタ：トロタクと
炙りサーモン**

Keisuke Irii

転職 3 回の何でも屋
壁を登ることしか
考えていない

推しネタ：マグロ赤身

Takashi Hosoya

絶賛修行中
本職は料理人
食材を求め海へ山へ

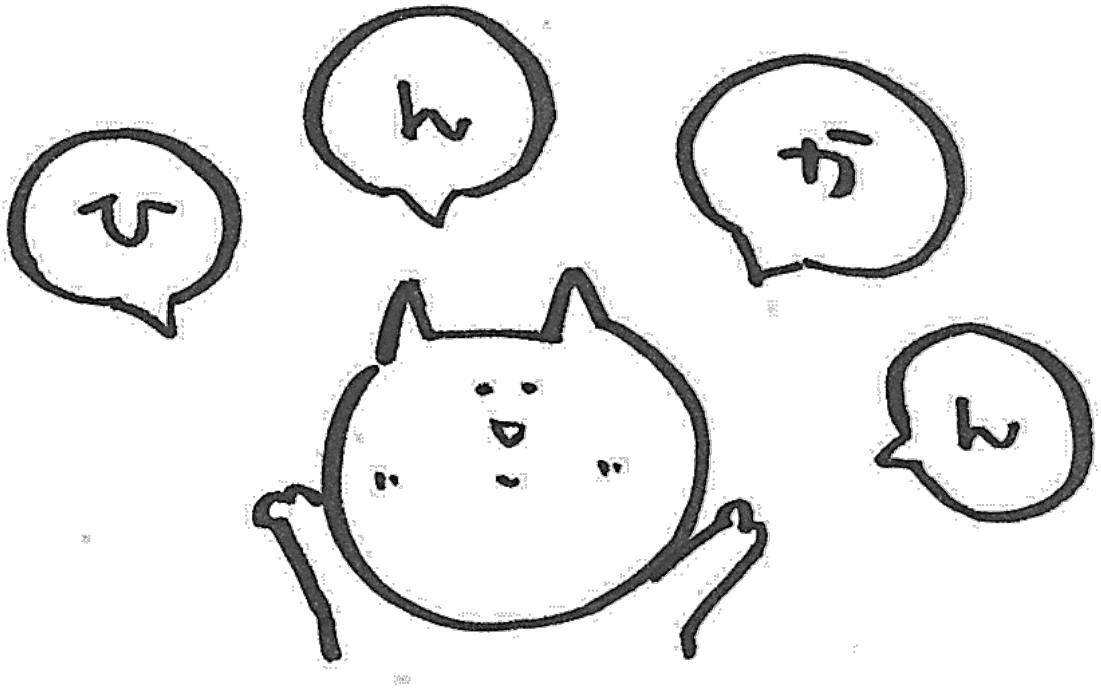
推しネタ：アナゴ

ある日の Good & New

品質管理室ではアイスブレーク、コミュニケーションの活性化を目的として毎朝 Good & New を話す取り組みをしています。ここではある日の Good & New を紹介します。

山本：長女が産まれた！2860g で母子ともに元気。妻退院まで長男の育児がワンオペになり、分かってはいたが大変…
白石：昨日も息子の水泳大会の応援に行ってきた。日曜の平泳ぎ 200M に続いて 100M も全国の基準タイムをクリア出来たのが Good。折角、京都まで行って 1 種目だけの参加だと寂しかったので良かった良かった。
鈴木：昨日の昼食は家族全員カレーだった。超偶然 New.
やなぎ：3 日連続でライブをみてきた。昨日行ってきたライブは、弾き語りのみのイベントで、弾き語りはミュージシャンそのものを出してくるので圧倒されたが、長丁場で少し疲れた。しかし、はじめてみるミュージシャンで気になった方もでて、良い体験ができた。Good また、出演者の中ではじめましての人に、お久しぶりですと人違いされたのが、モヤモヤした。
村上：昨夜何かを「明日の Good&New で言える！」と思ったはずなのだが、今朝起きたら何も覚えていなかった。のを New ということにする。。
浅井：スマホゲームの FGO のコラボでウエハースの第 5 段が発売された。多分依頼が来るだろうと先読みして少し買ったらシークレットのカードが出た。朝、彼女に報告したらとても喜んでくれた Good
若林：今日履いていくズボンのポケットを朝まさぐったら、50 円玉が入っていたのが Good
松井：Twitter でバズってた無印のすっぱいゼリーシチリアレモン味を買って食べてみた。言うほどすっぱくなかったけど、美味しいのは美味しいので Good（よしだまにお見舞いしたら悶絶してたが、気のせい）
城野：玄関の戸締まり忘れて寝ちゃったら、朝 4 時に蝉の叫びを耳元で聞いて起きるというイヤげな経験をした New ねこ、玄関勝手に開けて出ていけるのね。。。
細谷：連休に伊豆に釣り旅行に行ってきた。雨の合間を縫って釣りをしたが、赤潮が来てて大物は釣れなかった。早々に小物釣りに切り替えたらアジやらベラやらが釣れたので、それをその場で揚げて食べてみた。ベラは身が水っぽくておいしくなかったが、アジの美味しさを改めて思い知った。New
よしだま：昨日からあげ食べ放題にいってきた！Good 食べきれず、入井さんに 1 個押し付けてしまった…そのせいで入井さんは胃もたれをおこし、今日は休んでいます(嘘)
入井：休み

ひんかん！ ～品質管理室座談会～



品質管理室のメンバーで座談会をしてみました。メンバーには事前に以下のお題に回答してもらい、みんなの回答をまとめたものを見ながらワイワイおしゃべりしたものを編集して記事にしています。

❶ QA エンジニアになるまでの経緯を教えて！
❷ イチオシのテスト作業高速化術を教えて！
❸ テスト設計のコツを教えて！
❹ 開発の人と仲良くなるコツってある？
❺ 付録：品質管理室の人に必要なスキルって？

ACCESS 品質管理室の雰囲気を感じとってもらえればと思います。

❶ QA エンジニアになるまでの経緯を教えて!

アンケートの回答（一部抜粋）

A. 派遣でたまたま型
- 学生のときに就職氷河期なのに就職活動をさぼり続けていたら、卒業後に無職になってしまったので、とりあえず派遣登録してみたら紹介されたのが ACCESS のテスターだった。
- もともと Web 制作の個人請負の仕事をしていたので、そういうスキルセットを提示して派遣登録したら、たまたまテスターの仕事を紹介された。
- たまたま派遣テスターとして採用されたのがきっかけ。QA のことは何も知らずに働き始めた。

B. 運命のイタズラ型
- Mac で Illustrator や Photoshop が使えるということで、ある案件の UI 遷移図の作成要員として派遣されてきたのだが、その案件がなくなってしまったために仕方なくメーラー開発案件にテスターとして放り込まれたのがきっかけで今に至る。
- 保育士を辞めて Web デザイナーを目指して勉強していたが、派遣ではじめたテスターの仕事にそのまま定着してしまった。

C. 転職芸人型
- 開発→PMO→ラーメン屋→酒工場→テスター

D. 新卒採用型
- 開発採用で新卒入社したが、本配属先を選ぶときに品管を希望したから。

鈴木：「派遣でたまたま型」がやはり一番多いんですね。でも、ラーメン屋さんとかを経験してる人もいるみたいですよ(笑)
山本：それを言ったら僕も美術館で絵を運んだりしてましたよ(笑)
よしだま：いろいろとやってきた人が多いみたいですね。
山本：流れ着いちゃうのかな。怖いな（笑）
鈴木：JaSST の転職芸人でもこういう人いましたよね。
よしだま：私は保育士から Web デザイナーを目指す過程で派遣テスターになったらいつの間にか社員に…
入井：そもそもこの職業って昔は派遣以外で募集してなかったですよね。
浅井：派遣の中でもちょっと違うから難しい仕事って言われてましたね。

入井：人によってはすぐ辞めちゃう職業だから、辞めてない人たちになんで辞めないかを聞いてみましょうか。
よしだま：私は前職の仕事環境が酷すぎたので…自由にトイレに行けることに感動しました。
鈴木：それは深掘りすると別の問題になるので（汗）ほかの理由はどうでしょうか？
松井：居心地がいいから。人もあるし、環境も。
山本：松井さんは居心地のよさを大事にしてるよね。昔の ACCESS にはひどい働き方をしている人もいたから気持ちは分かります。
やなぎ：なんとなくですかね…仕事選びには「向き不向き」と「好き嫌い」の軸があって、好きではないけど向いていると感じたので。趣味で QA やっているタイプの人とは違いますね。
鈴木：えっ、みんなそうなの…？
山本：趣味と実益を兼ねていた方がいいんでしょうけど、そうじゃない人の方が多いんじゃないですかね。好きだけど向いていないと辛いので、それよりはいいと思いますけど。
若林：もともと品管に入りたいから、という人はいないですよね。誰かに必要とされてそこにいる、ってことなんじゃないですか。
山本：必要とされるのは大事ですよね！大きなモチベーションになる。

やなぎ：開発志望で入ってきたけど QA にい続ける人もいますよね。
よしだま：QA を辞めて開発に戻っても QA の知識が役に立っているって話は聞きました。JSTQB の知識でテストの漏れを指摘できるとか。

松井：まずテストっていう仕事があることを世間は知らないですよね。
入井：確かに。実際にやってみたら、ドラクエのレベル上げみたいに心を無にしてやれば意外と楽だなって印象がありましたね。
山本：テストってやれば終わるんだよね。手を動かせばいつかは終わる。プログラムはできないものはできない。
鈴木：バグが出て何度もやり直しとかになると終わらなくないですか？
山本：それは自分の責任じゃないし、身体的には辛いけど心理的には辛くない。もちろん、立場が上がっていくとそんなことも言っていられなくなるけどね…

やなぎ：テスターは細かい人が向いているってよくいわれますけど、実は大雑把な人とかいます？
よしだま：私、大雑把です。だからいつもやなぎさんに「ダメだよ」って言われてます。
やなぎ：(笑)
浅井：この仕事をしていたら、細かくなりますね。
よしだま：というか、若林さんも大雑把ですよね？
若林：何ですか急に（笑）細かいかどうかでいうと、細かくはないですけどね。

入井：細かさとあわせて、要領の良さも関係ありますよね。開発は遅れたら作業期間が延びますけど、テストはなぜか期日が変わらないことが多い。そんな環境で真面目に働き続けると普通に死んでしまうので、いい意味での手の抜き方・思い切りの判断は必須です。
若林：そういうときにどこまでテストするかの判断は難しいですよね。
山本：総合的な判断が必要だから難しいんだよね。技術要素や同類プロダクトとかお客さんの性格とか色々なことを知っておかないといけない。

❷ イチオシのテスト作業高速化術を教えて！

アンケートの回答（一部抜粋）

- 集中できる環境を作ること。お腹いっぱい、睡眠不足、過剰な労働量、やたら割り込みが入る、気がかりが多すぎると集中できず、結果、作業量は下がるので雑務をやる時間、集中する時間を分ける。
- キーボードショートカットとかクリップボードを管理するアプリを活用することですかね。チリも積もればマウンテンってやつです。
- カジュアルな自動化。
- 困ったことは過去に誰かが困っている、と考えてググって解決策を探す。
- 日頃の積み重ね。常に得た情報を整理し、それを取り出せる状況にしておけば、まぁまぁ効率的に作業を進められる。
- 生真面目にすべて取り組んでいたら終わらないことも多いので、力を入れるところ、抜くところを覚えるのが肝要。どこが力の出し入れどころかは仕様理解と経験によると思う。
- キーボードとマウスにこだわりを持つのが大事。
- テキストエディタ、CSV エディタ、Excel あたりを組み合わせると、テスト項目やテストデータの大量生産が高速にできるようになるのでおすすめ。文字置換で、タブ置換や改行置換を多用すると幸せな気持ちになれる。
- PowerShell や Linux のコマンドなど、なんとなくレベルでいいので使えるようになると色々はかどる。
- 集中できなくなったら一旦別の作業やメールチェック等をして切り替える。
- 複数端末活用。

鈴木：回答を見てみるとテスト専門の高速化っていうよりは、事務作業的なのが多いですよね。
入井：けっきょくチリツモで高速化するしかないですね。
やなぎ：作業をどう定型化していくかって話ですよね。
山本：テストって繰り返しが多いからね。チリも積もればマウンテン！

入井：複数端末活用はかなり有効じゃないでしょうか。プログラムの応答を人間が待つことの方が多いので。ものにもよるけど 3〜4 台ぐらいずらっと並べてどんどん順番に操作**していく**。もちろん事前に手順を把握しておかないといけないけど。
山本：付箋とか貼って区別するのも大事だよね。

❸ テスト設計のコツを教えて！

アンケートの回答（一部抜粋）

- 実際に顧客、エンドユーザーが行うであろう操作をイメージする。
- システムの概要をある程度は理解しておいて、アプリがどの様な動作をしているのか理解する。
- 仕様書を読むとき試験ケースを妄想しながら読む。
- この機能であれば、こういう不具合が出そう…など、試験コンテンツの内容や、その不具合を発見したときの開発さんの顔まで妄想しましょう。
- うーん、難しい。数をこなせば頭の中で勝手に構造化されます（たぶん）
- 情報を出しきって整理すること。分けて考えていいところは分ける（関連させない）こと。
- 情報収集と想像。
- 現状上手く出来ていない。逆にコツを聞きたい。
- テスト設計の元ネタのクオリティーをどれだけあげられるか、どれだけ豊富に用意できるかにすべてはかかっていて、それさえできていれば、あとは単純作業が多いんじゃなかろうか。
- コツかはわかりませんが…思いついたことを全部書き出す。疑問点や未確定なものもとりあえず書く。書いてから整理する。自分で観点を出した後、過去に似たようなプロジェクトがあれば比べてみる。仕様書の章番号やチケット#など、根拠になる資料もメモしていく。（抜け漏れ確認や後々の内容確認のため）
- むしろ聞きたいです… 三色ボールペン法とか試してみようかなとか思ったり…まだ手探りです。
- 過去のバグ（弱点）を参考にする。
- 仕様があいまいなときは、まずは期待動作を空にしてテストケースを作る。

鈴木：数をこなすのがコツ、はあんまり本に書いてないですよね
山本：まぁね、経験が必要ってことかなぁ。案件数とか、機能数とか？
頭の中で勝手に構造化される人とされない人はいるよね。
若林：何か想像しながら読むのがコツなんじゃないすかね
城野：不具合の改修をやってると、この機能にはこういう不具合が出るよねっていうのが積み上がっていくから、改修確認で数をこなすっていうのもあるかな
山本：設計だからノウハウがあるはずで、「情報を出し切る」「情報を整理する」ができればいい。情報を出し切るために知識や経験、想像力が必要ってことで。

城野：大体、テスト設計するときに仕様書はできあがってないことが多いじゃない？そうすると予測で「こう作るよね」って開発とすり合わせながらやっていかないといけない。ただ、聞いても答えは出てこないから、そこは想像力だよね
山本：テスト設計と言いつつプロダクトの設計をしちゃってることも多いよね。
エンドユーザーを想定したりして…非公式な仕様決定権があるから面倒なのかな
入井：QA が想像したものが期待動作でいいと思います。QA が「こうなるのが当然だ」と言い切ってあげないといつまでも期待動作が決まらない。
想像して補完して自信をもって言い切るところまで QA の業務として必要では。
山本：最終的にはお客さんの判断になることが多いけど、一次案としてはそうですね。
浅井：「こういう動きが正しいんですか？」ってお客さんに聞かれたときに困ったりしますよね。
やなぎ：要件定義のときに受け入れテスト・システムテストのことも考えて QA 側が仕様決めをリードしていくっていう手法はありますよね。

❹ 開発の人と仲良くなるコツってある？

アンケートの回答（一部抜粋）

- 質問しまくる（会話する機会が増える）
- 分からない範囲、分かる範囲を最初に展開してしまう
- チケット記載内容について遠慮せずに聞く
- できれば評価のチャットルームでも開発さんを交えた方が良い。間違っているときに拾い上げてくれます
- おしゃべり苦手な人も自分の仕事についての説明はちゃんとしてくれるし、理解したこと、ありがとうという気持ちを示すことでコミュニケーションは潤滑に回ります
- 開発プロセスを理解する（理解できなくても学ぼうとする）
- 開発として大変なことや状況を理解する
- 技術をある程度理解していると、こちらの話も聞いてくれるんじゃないかなと思う
- とくに敵対もしていないし、仲良くしなくても、同じ目標に向かって仕事しているチームなので、話し合えばよい
- 感謝の気持ちは惜しみなく伝える
- 仕事以外の世間話をする
- 一緒にタバコを吸いに行く・一緒にごはんに行く・一緒に飲みに行く・一緒に遊ぶ
- 開発に限りませんが、辛い食べ物をお見舞いすることですかね
- 話しにくい開発者はもちろんいるので、どういうタイプなのか見極めつつ、ベストな距離を探る
- まあ仲良くなくてもなんとかなる。質問すると多少仲良くなれる気がする。

入井：開発と仲良くならないと情報量に差が出たりしますよね。地味に大事な裏仕様みたいなものを教えてもらえたりとか。
若林：開発に限らず、テスターとも仲良くならないと情報が聞けないし、一般的なコミュニケーションは必要ですよね。

若林：辛い食べ物をお見舞いするってどういうことですか？
よしだま：これ回答したの絶対松井さんですよね！？
松井：とりあえずそれでコミュニケーションが発生しますよね。
鈴木：当たり屋ですか（笑）
山本：わさび鉄火（すごく辛い煎餅）ばらまいてましたよね。
入井：餌付けいいですよね。開発のところに行くときには毎回お菓子を持っていけばいいんですよ。
よしだま：糖質でいいコードが書けるかもしれませんよ！って羊羹持っていきます。

鈴木：仕事と関係ない話をする、って、世間話するのも難しくないですか？
入井：そもそもドライな関係の方がいい人もいるから。
山本：仲良くなくても話しかけやすければいいんじゃない？
やなぎ：メンバーだって認識してもらうのが大事ですよね。定例会議に顔を出したり。
浅井：とりあえず朝会で一緒にいれば最低限話す機会がでてくるんじゃないかなと思いますね。

村上：テスターさんとかなら自分の報告は自分でしてもらうと一員感が出ますよね。一緒に質問に行くとか、チケットでもテスターさんから質問してもらうとか。
入井：自分の質問なのに誰かに代わりに質問してもらう、みたいなのはよくないですね。
城野：仕様を説明できない子は引っ張っていってたよ。次からは一人で行って、って強制的に。
浅井：相手にもよりますよね。私が最初に質問に行った人が怖い人だったので、行きづらいこともあるんじゃないかなと…
城野：私は開発さんに不具合を報告するのが楽しみだったから、すごい満面の笑みで走っていってたなー。
山本：で、嫌がられてたんでしょ（笑）
城野：みんな目を反らすんですよね(笑)

入井：営業さんとか、直接開発と関係ないような部署の人と仲良くなるのもいいですね。視野が広がります。営業さんとかが無性に開発のことを心配している話を聞いたりすると、プロジェクトの報告の仕方に問題があるかもしれないなーって気づいたり。
よしだま：どこから情報を引っ張ってきてるんですか？
浅井：どういう情報を知りたいかによるかな。裏情報はタバコを吸いながら集めますね
入井：誰と誰が同期で仲がいいとかを覚えておくと使えますよ。誰に何を聞けばいいか、に役立つんですよね。前に同じ案件やってたとかも。
山本：そういうの覚えるの苦手なんだよな…

山本：開発の言うことを真に受けてもいけないんだよね。盛って話す人とネガティブに話す人がいるから、その人がどういう風に情報を出すのかを知ってないといけない…
そこまでしないといけないのか？浮気調査か？普通にオープンにしてほしい。
本来はコツとかなくても情報共有できるようにしたいですよね。
入井：いろんな考え方の人がいるからなかなか難しいですね。

ネコ師匠

**ひと鳴きすれば
バグを見つけるという
アダマンタイト級テスター。**

ネコテスター

テストをはじめて３年。
スキルアップのために書籍
を購入しては積みがち。

ウサテスター

テストをはじめたばかりの新
人。自宅にパソコンがなく、
すべてスマホで済ませる世代。

❺ 付録：品質管理室の人に必要なスキルって？

座談会でいろいろと話しているときに「品質管理室の人に必要なスキル」についての話題になりました。簡単にですが付録として、出てきた意見のいくつかを紹介します。

品管への入口

- テスターの経験
- 最低限の IT 知識
- IT への興味

プロセスの理解

- 開発プロセスの理解
- 品質保証プロセスの理解

プロセスを現場で回す

- プロジェクトマネジメントの実践
- テストマネジメントの実践
- リスクマネジメントの実践

仕様マスターになる

- ヒアリングスキル
- 情報整理スキル
- 言語化スキル
- 文書化スキル
- バグ情報の生き字引となること

テスト方面のスキル

- 作業の自動化、効率化
- テストの自動化、効率化
- 勘の良さ、バグ出し職人
- わかりやすい報告
- 見積上手、計画上手、設計上手
- 体力、集中力

製品・サービスのドメイン知識

- Web ブラウザーなら HTML、CSS、JavaScript、DOM、HTTP、SSL の知識
- EPUB なら組版、HTML、CSS、EPUB、JavaScript の知識

ソフトスキル

- コミュニケーションのスキル
- 語学のスキル
- リーダーシップのスキル
- ファシリテーションのスキル
- チームの一体感

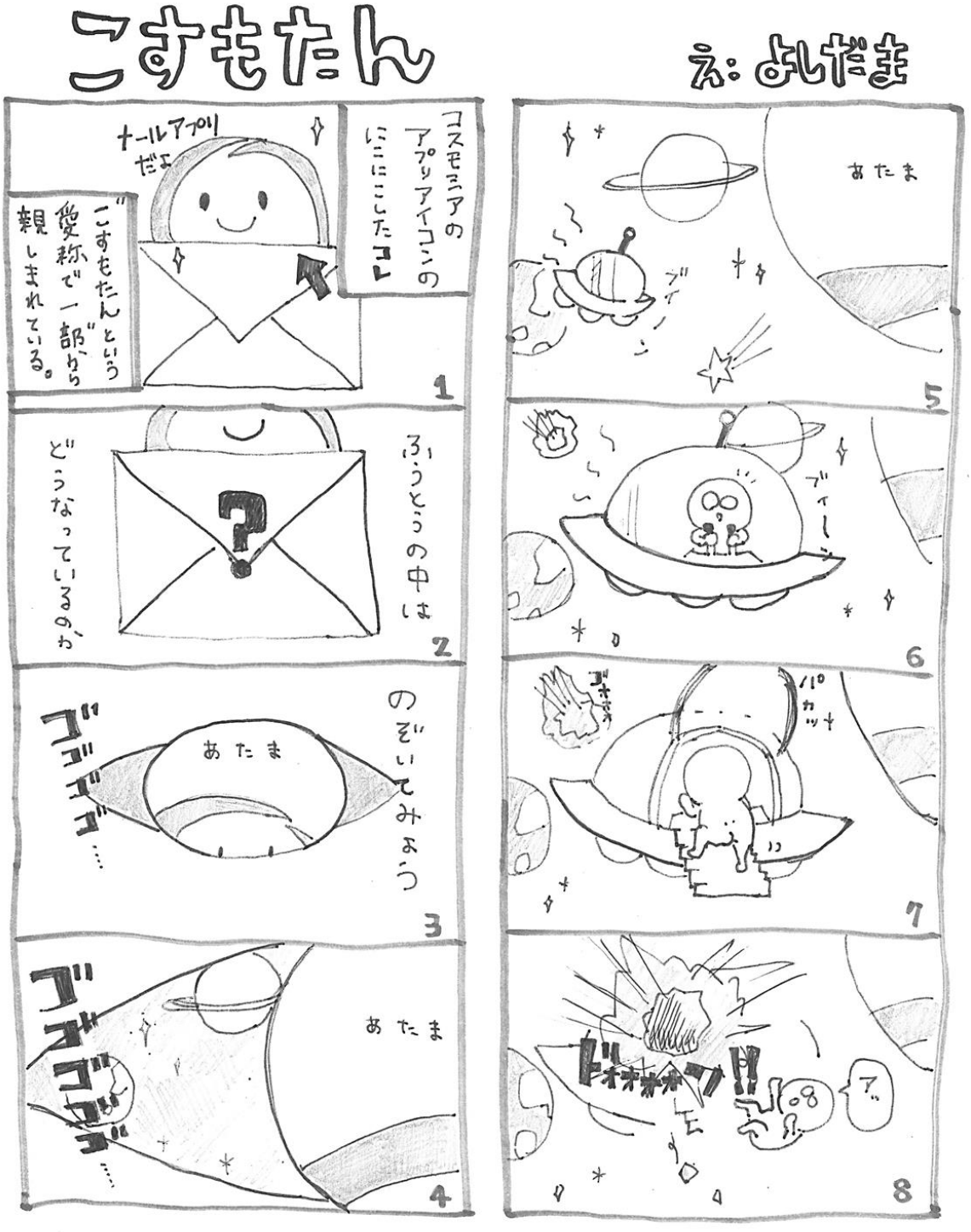
現場のテストにすぐ役立つスキル

- 開発環境構築および自分でビルドできるようにしておく
- Web ブラウザーの F12 開発者モードを使いこなせるようにしておく
- Linux のコマンドを使いこなせるようにしておく
- 正規表現がわかる
- シェルやバッチが書ける

他社に依存する仕様の知識

- スマホアプリをストアに公開する方法、ルール
- アプリ内課金や、定期購読周りの仕様

Google Play
で手に入れよう



↓↓先生へのおたよりはこちらまで↓↓

@yocidama

現場で使えるバグ曲線

書いた人
Keisuke Irii
モバイル、セキュリティ、EC、IoT と渡り歩いている QA おじさん

このコーナーでは殺伐としたお仕事の現場できっと役に立つバグ曲線作成テクニックを紹介します。

❶ バグ曲線をソフトウェアの成熟傾向の判断材料として使う

一般にバグ曲線というと、グラフの縦軸に累積バグ数、グラフの横軸に日付をとって作成されます。しかし漫然な作り方をしたバグ曲線は、ソフトウェアの成熟傾向を判断するための資料としては役に立ちません。

単純なバグ曲線が、ソフトウェアの成熟傾向を判断するための資料になれない大きな理由は以下の通りです。

- バグがたくさん出るようになったように見えて、実際にはテスト実行が本格化しただけかもしれないということを否定できない
- バグが出なくなったように見えて、実際にはテスト実行をやめただけかもしれないということを否定できない
- 営業日以外はバグが出ないため、バグ曲線が自然と横方向に間延びした感じになり、そのおかげでバグ曲線が寝たように見えていることを否定できない

バグ曲線の形を見て品質を判断したいのに、バグ曲線の形状自体に誤りがあれば、正しいつもりの判断も結果として誤りになってしまいますね。そこで、形状にこだわったバグ曲線を描くために以下の手順で作業します。

1. 「その日のバグ検出総数」を用意する
2. 「その日の総テスト実行工数」データを用意する
3. 「1 人日あたりバグ検出数」を算出する
4. 「1 人日あたりバグ検出数」の累積をグラフの縦軸にとる
5. 横軸には「その日の総テスト実行工数」が 0 でない日のみをプロットする

では、手順をひとつずつ説明していきます。

手順 1「その日のバグ検出総数」を用意する

「その日のバグ検出総数」はご利用中のバグトラッカーから抽出すればよいのですが、以下の点に注意してください。

- バグではないチケットを含まないようにする
- バグであっても重複報告のチケットを含まないようにする
- バグであっても品質測定の対象ではないものを含まないようにする（たとえば既存バグ）
- バグかどうかわからないものは含んでよいが、バグではないとわかった時点で除外する

手順 2「その日の総テスト実行工数」データを用意する

「その日の総テスト実行工数」も用意するには、どの程度厳格に記録するかは状況によりますが、以下のような方法があります。

- その日テストに参加しているテスト実行者の人数に所定の労働時間をかけ合わせる
- その日テストに参加しているテスト実行者に稼働時間を日報などで報告してもらい合算する
- その日テストに参加しているテスト実行者の稼働時間を勤怠管理システムから抽出して合算する

手順 3「1 人日あたりバグ検出数」の算出

「その日の総テスト実行工数」が算出できたら、「その日のバグ検出総数」を「その日の総テスト実行工数」で割ります。単純に割ると 1 人時間あたりのバグ検出数が算出されますが、そこに法定労働時間をかけ合わせて 1 人日あたりどのぐらいバグが出たのかという数字にした方が、感覚として理解しやすい指標になると思います。

「1 人日あたりバグ検出数」はまさに「いまどのぐらいバグが出やすいのか」を示した指標です。この指標はバグ曲線と同じグラフに棒グラフで共存させると、バグ検出の沈静化傾向がよりわかりやすくなります。

手順 4 「1 人日あたりバグ検出数」の累積をグラフの縦軸にとる

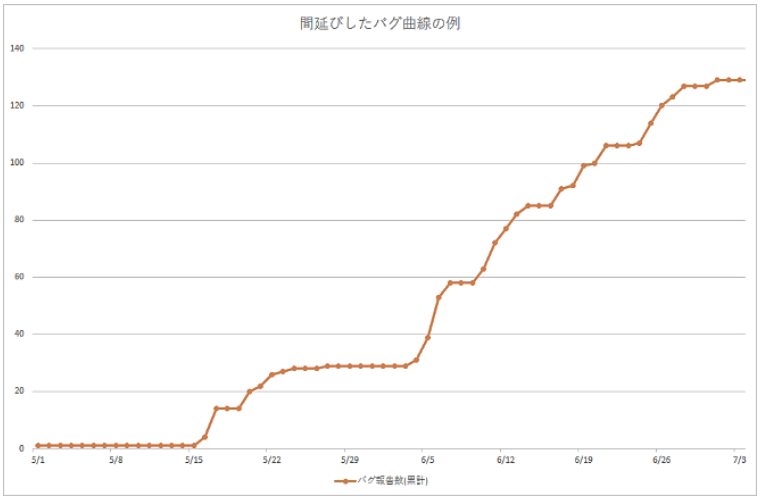
営業日ごとに算出された「1 人日あたりバグ検出数」を累積していき、この数値をグラフの縦軸に使ってバグ曲線を描画します。つまり、バグ曲線が表現しているのはバグの累積数の成長ではありません。バグの出やすさが終局に向かっているかを表したものです。

手順 5 グラフの横軸にはテスト実行工数が発生した日のみをプロットする

土日祝日をグラフの横軸から省くのではなく、テスト実行工数の発生した日（および未来においては発生する予定の日）だけを横軸にプロットするようにします。このようにすることで、「1 人日あたりバグ検出数が 0 の日」はテストをしたのに本当にバグが検出されなかった日であることがわかります。

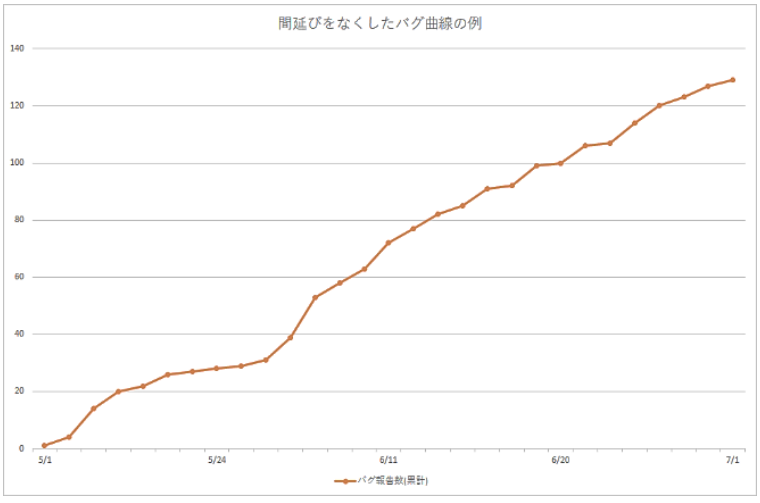
手順 5 の解説

下のグラフ（図 1）は単純にグラフの縦軸に累積バグ数、グラフの横軸に日付をとって作成したものです。長期間フラットになっている時期が見えますが、テストを実行しているのにバグが出ていないのか、テストを実行していないのかは不明です。また、土日になるたびにフラットな線が挿入されてしまっていますし、終盤だけを切り取ればグラフが寝ているようにも見えなくないです。



（図 1）間延びしたバグ曲線の例

前出のグラフ（図 1）を手順 5 に従い、「その日の総テスト実行工数」が 0 の日をグラフに含めないように加工すると下記（図 2）のようになります。実際のところはほぼ一直線に成長していて、品質としてまったく成熟していないことが分かります。これではまだ当分はテストを続けるしかないですね。

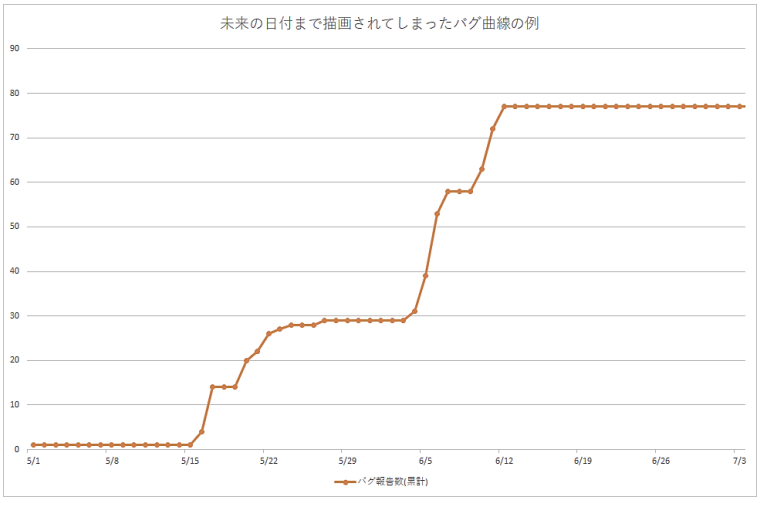


（図 2）手順 5 を使って間延びをなくしたバグ曲線の例

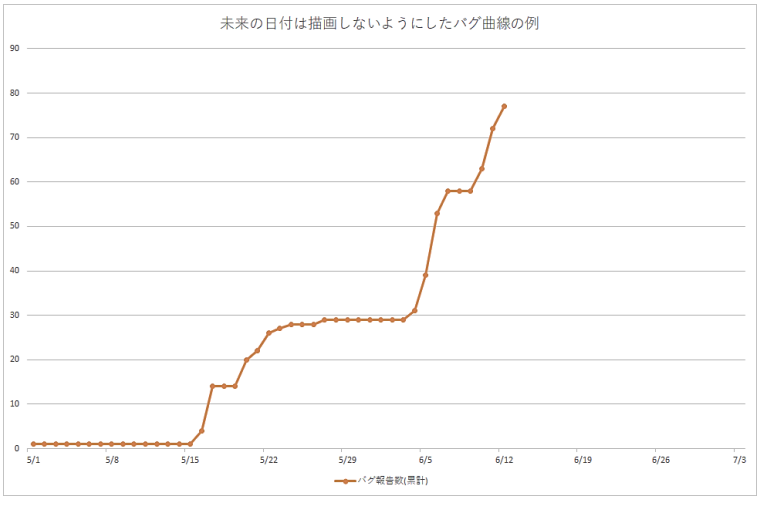
❷ バグ曲線をよりわかりやすくする

未来のバグ曲線を描画しないようにする

しばしば見かけるのが、テスト期間中なのにバグ曲線が現在日付よりも先まで描かれているバグ曲線です。このようなバグ曲線では現在の日付より先が水平線になってしまうため、なんとなく寝ているように見えてしまいます。細かいことですが、バグ曲線は現在日付が終端になるように描画します。



（図 3）未来のバグ曲線が描画されてしまっているグラフの例



（図 4）未来のバグ曲線を描画しないようにしたグラフの例

バグ曲線の表示期間を長くしすぎないようにする

長期間のプロダクト開発でひとつのバグ曲線を使っていると、規模ばかり大きくなって、何を示したいのかわからない資料になってしまいます。
そのようなときは以下のような単位で区切ってバグ曲線を分割すると効果的です。

- 会社に申請した開発プロジェクト単位
- 会社の四半期単位

このようにバグ曲線を分割することで生産性指標との突合せもしやすくなるという副次的な効果もあります。また、ベース品質が悪すぎるプロダクトであっても、現在の作業にいったんの終わりを設けることができます。

❸ バグ曲線をより簡単に作成する

バグ検出予測数はテスト実行開始後、数日経ってから正式なものを作成する

バグ検出予測数はテスト工数の見積に使用するにはとても有用ですが、テスト開始前に作成した値をそのままバグ曲線と一緒にプロットすることは推奨できません。たいていの場合すぐに実績と乖離してしまい、意味のない線が引かれるだけとなってしまいます。

一方、テストを始めてしまえばバグ検出予測数は立ちやすいものです。テスト実行が開始して数日待ってからバグ検出予測数を作成し、グラフ上にプロットする方が作業的にも無駄になりません。

バグ検出予測曲線を簡単に作る

せっかくバグ曲線を作っているのなら、いつまでテストを続ければバグ曲線が寝るのかを説明できるようになったほうがいいでしょう。そのためにはバグ検出数の実績をもとに、バグ検出予測曲線を描く必要があります。

そこでここでは気軽に「ほぼ」正確なバグ検出予測曲線を作成する手順を紹介します。

1. 本日までのバグ検出数の実績で描画されたバグ曲線を紙に印刷します。
2. 右手に鉛筆を軽く握り、ペン先をグラフの起点に当てます。
3. 手順 1 で描かれた既存のバグ曲線の傾向を描きながら、こんな感じになるだろうという強い確信のもとに、手首を柔らかく使って、既存線の延長に未来のバグ曲線を描きます。
4. 描かれた曲線を観察し、線がほぼ平行になったと思われる個所に該当する横軸の日付が品質の安定する時期です。

手順 3 を見てそんな馬鹿な、と思われたかもしれません。しかしこの手順で作成されたバグ報告の沈静化する時期の推測はおおむね信頼できるものです。当然ながら手順 1 の既存のバグ曲線が長ければ長いほど正確になります。

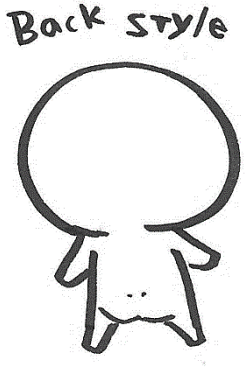
予想バグ曲線を作成することの本当の意味は何でしょうか。それは明らかに無謀なプロジェクトの計画を軌道修正するための道具として用いることにあります。明らかに無謀なものを訂正するには、100 点の予想ではなく、70 点の予想があれば十分です。予想は外れていいのですが、外れすぎてはいけませんし、予想を作るのに時間をかけすぎてもいけません。

❹ 組み合わせでバグ曲線をもっと便利な資料にする

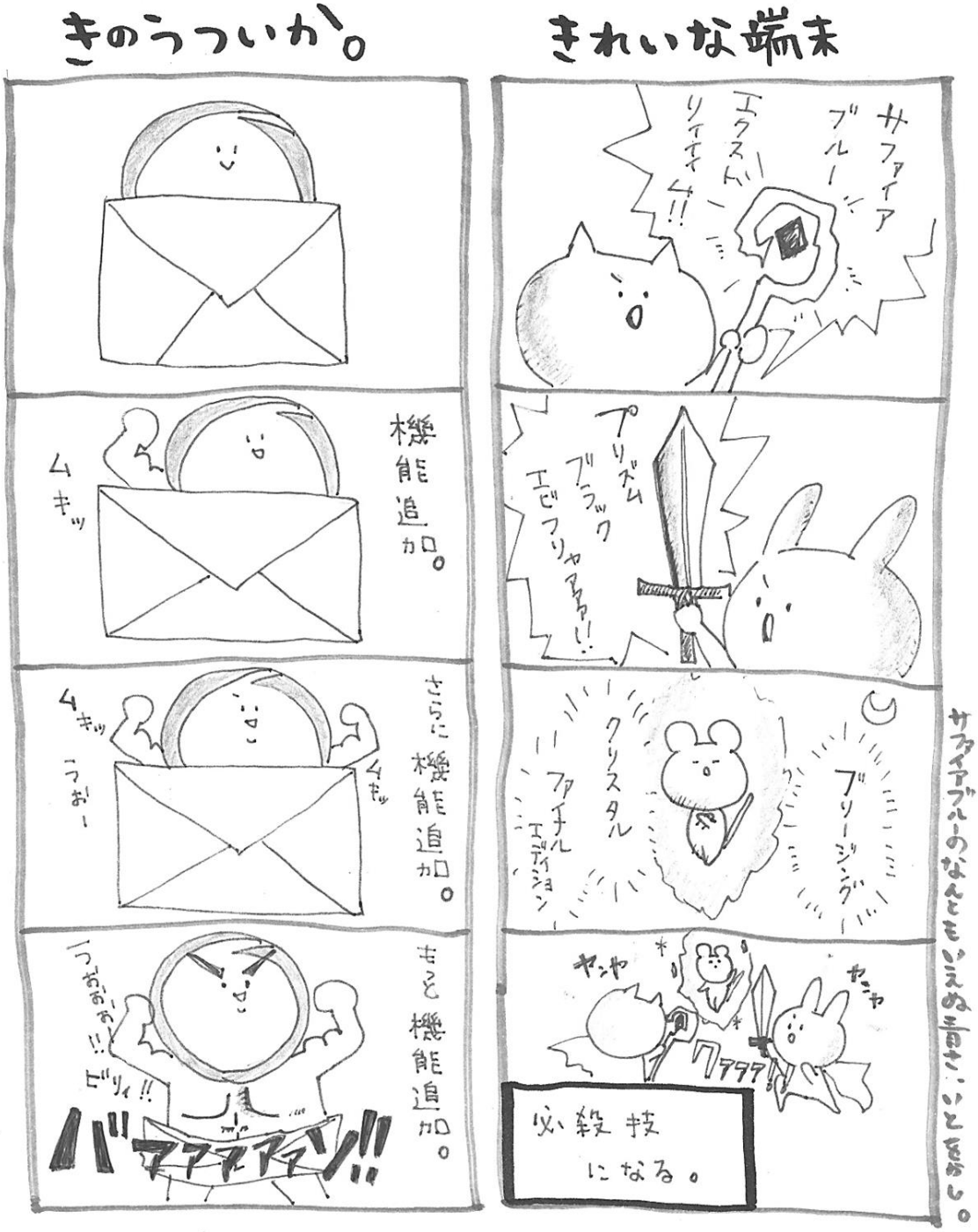
バグ曲線が描画されているグラフに以下のメトリクスもプロットしていくことで、より多目的に利用できる資料とすることができます。

要素が多すぎるグラフは見づらくなってしまうので、意図的に見せたいメトリクスのみを選択することが大切です。私のおすすめするメトリクスは以下の通りです。

メトリクス	グラフの種類	目的
バグ検出予測数	線グラフ	テスト進捗の確認
未実行テスト数	線グラフ	テスト進捗の確認
未修正バグ数	線グラフ	バグ改修進捗の確認
1人日あたりバグ検出数	棒グラフ	品質の確認
1人日あたりクリティカルバグ検出数	棒グラフ	品質の確認



え：よしだま


きのうついか。
きれいな端末
サファイアブルー
エクスパンション!!
プリズム
ブラック
エビフリャアアア!!
ブリージング
クリスタル
ファイナル
エディション
ヤッヤ
ヤッヤ
777
必殺技になる。
サファイアブルーのなんともいえぬ青さ。いとをかし。
機能追加。
ムキッ
さらに機能追加。
ムキッ
ムキッ
こおー
さらに機能追加。
こおおおお!!
ビリィ!!
バアアアアン!!


Bugってドキドキ♡
第1話 出会い
私の名前は 羽宮 零
今日から 花の高校 一年生
あぅん
もぅ 初日から ちこくしそう
あの…
これ おとしたよ
タイプC ケーブルかな？
ドキン…！
はっ…はい ありがとう ございます…
ドキドキの高校生活スタート!!（つづく）
第2話 おんこんふぃぎゅれーしょんちぇんじ
朝のドキドキの正体を 確かめたくて 知恵袋で 質問してみた
落としものを ひろって くれた学校の せんぱい をみてから，ドキドキが とまりません。 なぜでしょうか…？
ズバリ！ それは恋ですね。
Fall in love ♡
…!!
キャーッ
そんな!!
あっ
アプリが 繰り返し 停止しています
縦横切替で 落ちてる
この気持ちは 強制終了なんてできない…！（つづく）

テスターのための Git コマンド入門

❶ はじめに

書いた人
Kasumi Suzuki
EPUB と git の伝道師
社内外の各所に首と足
を突っ込んでいる

ここでは、以下のような方を対象に Git コマンドの使い方を解説します。

- テスターなどの非開発者（基本的にソースコードは読む専門）
- ローカル PC にソースコードが必要（ビルドするためなど）
- Git を十分に理解する時間がない
- Git はあまり使わないので今だけ手順が分かればいい

紹介するのは Git のごく一部ですが、少しでも触れるようになれば、Web の解説記事もスムーズに分かるようになると思います。
「全然わからない！怖い！」という状態から「これはやってもいいんだ」「怖くないよ」にレベルアップする助けになれば幸いです。

この記事では以下のコマンドを紹介します。

- 最初に使う `git clone`
- ソースコードの状態を確認する `git status`, `git log -1`
- ソースコードを最新にする `git pull`
- ソースコードを綺麗にする `git reset -hard`, `git checkout` `git restore`（git v2.23.0）, `git switch`（git v2.23.0）
- ソースコードを最新以外のバージョンにする `git checkout`

❷ Git をはじめよう

Git、面倒だな？

新しいことを勉強するにはコストがかかります。慣れるまでは失敗もします。「やらずに済ませたい」と考えるのは間違っていません。

CI 環境を充実させれば関係者全員が便利に使えますので、まずは CI 環境の構築・充実ができないか、開発チームに相談してみましょう。

- CI を最大限活用しよう
- 最新バージョンのビルド
- リリース物のビルド
- 任意のバージョン（ブランチ・タグ）のビルド

使えれば便利な Git コマンド

使わずに済めばそれに越したことはないのですが、一度 Git を使えるようになると後々まで役立つと思います。頑張って習得してみましょう！

Git を使えると…

- 直接的に役立つ
  - ✧ 任意のバージョンのビルドができる
  - ✧ 一時的にソースコードを書き換えてテストができる
- 間接的に役立つ
  - ✧ GitHub で公開されているツールを使ってみるハードルが下がる
  - ✧ テストデータやドキュメントを管理するためのリポジトリを作ったりできる

Git コマンドを使えると…

- GUI ツールは SourceTree とか色々あるので、それを使ってもいいと思う
  - ブランチツリーや diff が見やすいとか便利な機能がたくさんある
- コマンドを覚えておくとサーバーなど GUI がない環境でも使える
  - このときささっと apt とか yum で便利なツールを入れられるような人は普通に Git コマンドも使えると思う…
- CI/CD スキルの一つとして
- ツール要因のトラブルから解放される
  - 使っているツールが違うと話が通じない問題
  - 同じツールでも話が通じない問題
    - バージョン・OS が違うとデザインが違う
    - 言語設定の違い（英語メニュー、日本語メニュー）
  - ツール自体の問題
    - ツールのバグ
    - インストール・アップデートトラブル
    - ストレージやメモリの圧迫（カツカツでマシンを使っている人限定）

怖がらずに Git を使うために

使いたい！けど事故ってしまったらどうしよう…という不安は、以下のような対策で軽減しましょう。

- GitHub, GitLab など権限設定できる場合、まずは「Read」権限だけをもらうようにしましょう。push できないので安心です。
- 最新の Git（v2.23.0）をインストールしましょう
  - git switch, git restore など分かりやすくて便利な機能が使えます
- Git の理解を深めましょう
  - ベタですが「サルでもわかる Git 入門」（https://backlog.com/ja/git-tutorial/）
  - 「LearnGitBranching」（http://k.swd.cc/learnGitBranching-ja/）は実際にコミットやブランチ操作ができるのでおすすめです
  - 自分の PC 内にリポジトリをつくることもできます（`git init`）

事前に覚えておくべきコマンド

Git コマンドを使うには、Windows のコマンドプロンプトや macOS のターミナルなどのコマンドを少しだけ覚えておく必要があります。

pwd（Windows の場合 dir）	現在位置を確認する
ls（Windows の場合 dir）	現在位置にあるファイル・フォルダ一覧を表示する
cd フォルダ名	～に移動する（change directory の意）
cd ..	ひとつ上の階層のフォルダに移動する
mkdir フォルダ名	フォルダを作成する（make directory の意）

この 5 つが使えれば、以下のようなことができます。

1. `pwd` または `dir`（現在位置を確認）
2. `ls` または `dir`（現在位置のフォルダを確認）
3. `mkdir code`（ソースコードを置くフォルダを作成）
4. `cd code`（code フォルダに移動）
5. （Git コマンドを使ってソースコードを管理！）
6. `cd ..`（code フォルダから出て元の位置に戻る）

Git のインストールとプロキシの設定

macOS には Git がはじめからインストールされています。Windows に Git をインストールするには「git Windows インストール」などでググってみてください。色々な記事が見つかるかと思います。

社内ネットワークにプロキシがある場合、インストール完了後に設定しておきましょう。プロキシ設定は分かりにくいので、不安があれば早めに誰かに助けを求めましょう。

```
git config --global http.proxy proxy.example.com
git config --global https.proxy proxy.example.com
```

❸ Git をつかおう

ふんわり用語集

さて、これからいよいよ Git コマンドについて解説していきますが、その前にいくつかの用語を簡単に定義しておきます。ここでの説明はあまり正しくはないのですが、「限定的な使い方しかしない」という前提で単純なものにしています。

ローカル	自分の PC のこと。
リモート	自分の PC 以外のこと。GitHub だったり社内サーバーだったり。
リポジトリ	ソースコードのかたまり。コードを書き始めてから今までの全ての変更履歴が記録されている（ので、どの時点のソースコードでも参照できる）
コミット	リポジトリに記録されているひとつひとつの変更のこと。
コミットメッセージ	どんな変更だったかの説明。「xx というバグを直した」とか「xx のバグが直りきっていなかったので追加修正」とか。
コミットハッシュ	バージョン番号のようなもの。リポジトリに記録されている全ての変更履歴の中から「この時点のソースコードを見たい」と指定するための ID。違う変更に同じ ID が付いてしまったら困るので、SHA-1 という人間にはわかりにくい文字列になってしまっているが、あまり怖がらなくて大丈夫。
ブランチ	変更が積み重なってできた一本の履歴。 標準版とカスタマイズ版を開発している場合、標準ブランチとカスタマイズブランチがあったりする。 他にも開発ブランチ（開発・テスト用）とリリースブランチ（テストして OK だった修正だけ入れる）とか。 「どのブランチを見ればいいですか？」などと質問すると Git デキる感が出るかも…

最初の一歩 git clone

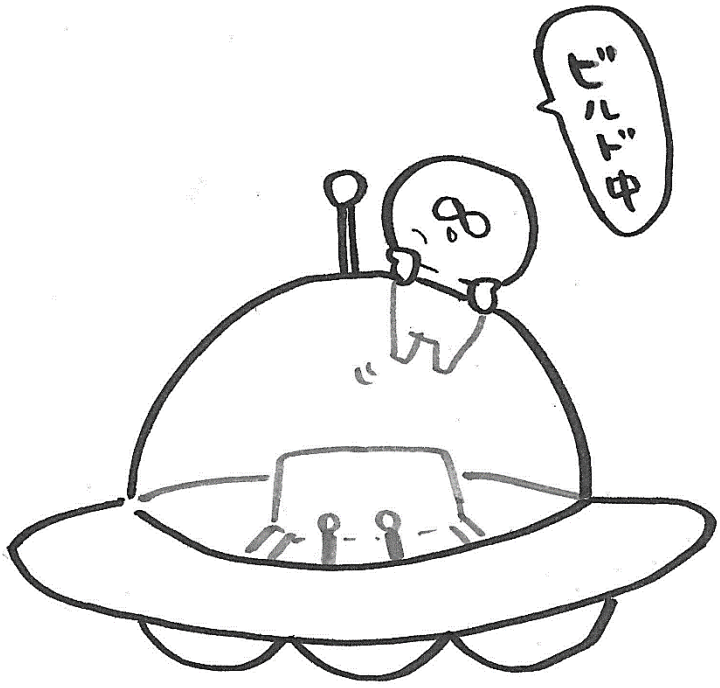
```
cd リポジトリをおきたい場所
git clone リポジトリのアドレス
```

まずはともあれ、リモートにあるオリジナルのリポジトリをローカルにクローン（コピー）してきましょう。リポジトリのアドレスは例えば GitHub だったらこんな感じです。

```
https://GitHub.com/ユーザ名/リポジトリ名.git
git@GitHub.com:ユーザ名/リポジトリ名.git
```

※このとき、リポジトリにアクセスしていい人かどうかの確認のため、ユーザ名やパスワードの入力を求められることがあります。PC のユーザとは別のアカウント情報が必要な場合が多いので、誰かに確認しましょう。

うまくいっていれば、リポジトリ名のフォルダができているはずです。ls または dir コマンドで確認してみましょう。



ソースコードの状態を確認する git log, git status

```
git log -1 今見ているバージョンを確認
git status 今のステータスを確認
```

状態確認はとても重要です。

- 自分がどのバージョンのソースコードを見ているのか
- 古いバージョン・間違ったバージョンで確認していないか？
- ソースコードに意図しない変更が入っていないか
- 何かの拍子にファイルを削除・上書きしてしまっていないか？

それほど複雑な手順ではないので、Git を使い始めの頃はしょっちゅう確認するようにしましょう。

```
git log -1
```

今見ているバージョンのコミットハッシュやコミットメッセージを確認できます。

「-1」の部分の数字を増やすと今見ているバージョンの一つ前、二つ前、…と履歴をさかのぼることもできます。(が、大事なのは「今、自分がどのバージョンを見ているのか」です。履歴を見るだけならブラウザーで可能なことが多いと思います)

```
git status
```

今のソースコードの状態（ステータス）を表示します。

- PC の操作ミス
- ビルド時に自動でファイルが生成された
- テストのために一時的にソースコードを書き換えたが、戻し忘れた

などの理由で、ソースコードは意図せず変更されてしまうことがあります。

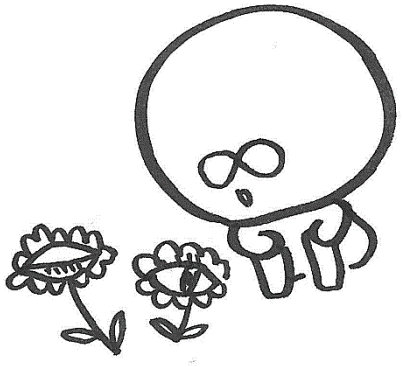
正しくない（オリジナルとは違う）ソースコードをテストしても意味がないことが多いので、ソースコードに変更がないかはよく確認しましょう。

```
nothing to commit, working tree clean
```

とあれば大丈夫です。

```
Changes not staged for commit:
modified: ファイル名（ファイルが変更されています）
deleted: ファイル名（ファイルが削除されています）
Untracked files: ファイル名（ファイルが追加されています）
```

などとあった場合、変更があるので、対処しましょう。
後述の「ソースコードを綺麗にする」を参照してください。



ソースコードを最新にする git pull

```
cd リポジトリ名のフォルダ
git status
git pull
```

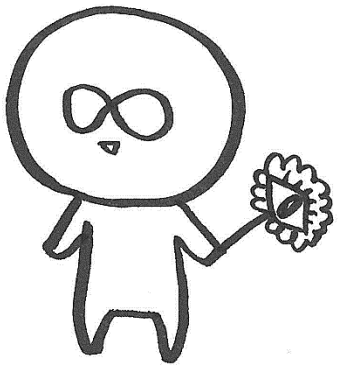
クローンするとその時点での最新のソースコードがダウンロードされますが、その後に追加された変更は自分で取得（プル）しに行く必要があります。

git pull 実行時の注意

`git pull` を実行する前に `git status` でリポジトリの状態を確認しましょう。（前章参照）

以下のようなメッセージが表示されていたら対処が必要です。「ソースコードを綺麗にする」を参照してください。

- Changes not staged for commit:（変更されているファイルがある）
- HEAD detached at 英数字（今見ているのがブランチの最新でない）



ソースコードを綺麗にする（元に戻す）git reset

```
git reset --hard
git checkout ブランチ名
```

※ git v2.23.0 では以下も使えます。

```
git restore ファイル名
git switch ブランチ名
```

`git status` で

- Changes not staged for commit:
- HEAD detached at コミットハッシュ（英数字）

と表示されるときは、ソースコードがローカルで変更されていたり、ブランチの流れから外れてしまっている可能性があります。

意図的に変えているのであれば別ですが、そのままでは作業に支障が出るおそれがあるので以下の手順で元に戻しましょう。

「Changes not staged for commit:」への対処

- modified, deleted がある場合 → `git reset --hard`, または `git restore ファイル名`
- Untracked files がある場合 → 手動でファイルを削除または別フォルダへ移動
- `git status` で綺麗になったか確認
- nothing to commit, working tree clean が表示されれば OK

「HEAD detached at コミットハッシュ」への対処

- 戻りたいブランチ（master や dev など、開発の本流になっているブランチ）名を確認
- `git checkout ブランチ名`, または `git switch ブランチ名` を実行
- `git status` で綺麗になったか確認
- nothing to commit, working tree clean が表示されれば OK

上記の対応を行っても元に戻せないときは詳しい人に見てもらいましょう。

ソースコードを最新以外のバージョンにする git checkout

```
git checkout ブランチ名またはコミットハッシュ
```

※ git v2.23.0 では以下も使えます。

```
git switch ブランチ名
```

ソースコードを最新以外のバージョンに変更するには、「ソースコードを綺麗にする」で紹介した `git checkout` コマンドを使います。二つのバージョン指定方法（ブランチ名、コミットハッシュ）の違いは

- ブランチを指定して変更→車線変更
- コミットハッシュを指定して変更→すごろくのマスを指定して移動

と捉えると分かりやすいかもしれません。このコマンドを使うと「少し前のバージョンの挙動を確認」「新機能確認用の一時的なブランチをビルド」などができるようになります。

git checkout は準備が肝心

トラブルを避けるために、これまでに紹介したコマンドで準備をしておきましょう。

- ソースコードを綺麗にしておく
- ソースコードを最新にしておく

いちいち面倒に感じるかもしれませんが、Git の理解が進めばこのような準備が不要なケース・必要なケースが区別できるようになると思いますので、最初は耐えてください。（むしろ積極的にトラブルを起こして解決に苦労した方が理解が深まるという説もありますが…）

目的のブランチ名やコミットハッシュを確認する

準備ができたら、次は目的地を確認しましょう。Redmine のチケットを読んだり、GitHub・Gitlab のコミットページを見たりで調べられると思います。調べたブランチ名やコミットハッシュはコピーしておくと後でペーストすればいいのでオススメです。

目的地に移動する（チェックアウトする）

ここまで来れば後は移動するだけです。下記のコマンドでバージョンを変更しましょう。

```
git checkout ブランチ名またはコミットハッシュ
```

※ git v2.23.0 では以下も使えます。

```
git switch ブランチ名
```

移動できたかどうかは git status で確認です。

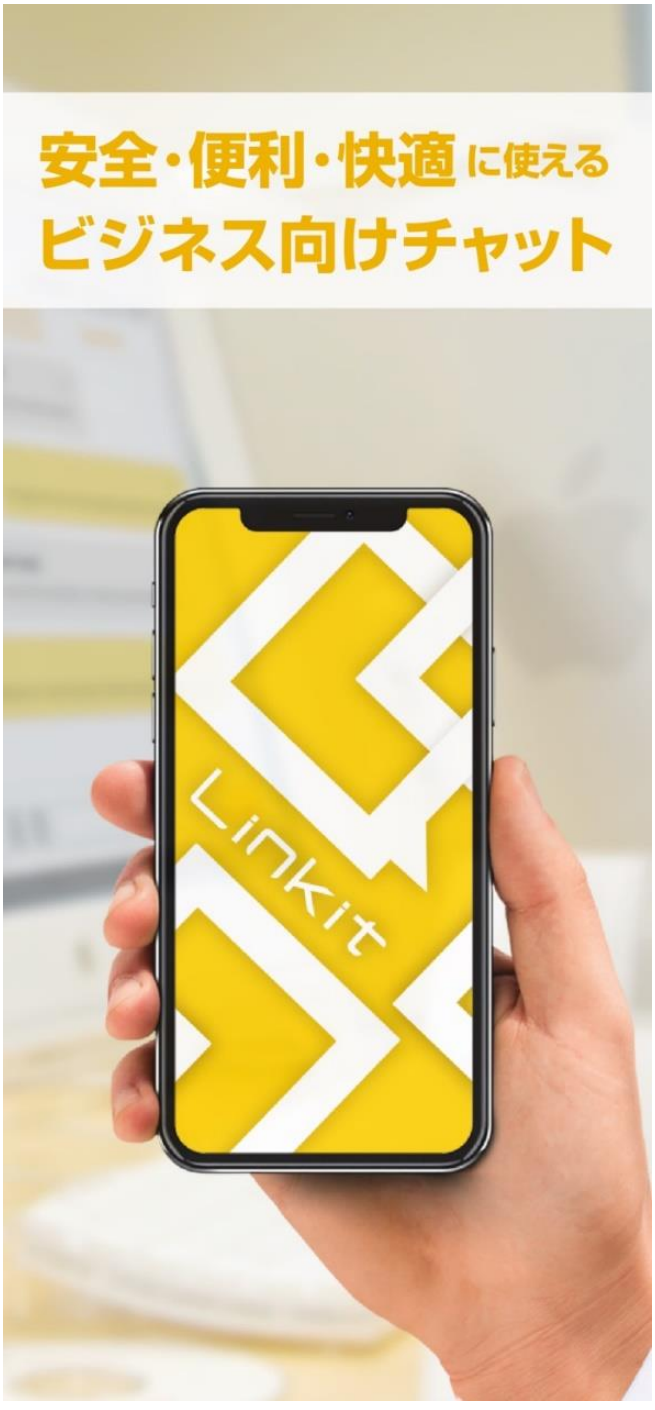
元のブランチに戻る

おうちに帰るまでが遠足です。確認が終わったらもう一度 `git checkout` コマンドを使って、本流のブランチに戻っておきましょう。ソースコードを綺麗にしておくと、次回スムーズに使えます。

```
git checkout 戻りたいブランチ（master や dev など、本流のブランチ）
```

※ git v2.23.0 では以下も使えます。

```
git switch ブランチ名
```



Google Play
で手に入れよう

App Store
からダウンロード

最適な QA 組織とは?

最適な QA 組織とは何でしょう。過去を振り返りながら、
品質管理室のリーダーが語るコーナーです。

書いた人
Tatsuya Yamamoto
頼れるリーダー
二人目が生まれたばかり
ただいま育休中

ACCESS の現 QA 組織

ACCESS 本社の現在の QA チームは開発部とは独立したひとつの組織として存在していて、社内の様々なプロジェクトの QA マネジメントやテストを担っています。

1 ヶ月のうちに 20～30 のプロジェクトが平行して動いていますが、ひとつひとつのプロジェクトの規模は大小様々で、期間も数日間で終わるものから数か月かかるものまであります。各プロジェクトを 1 名～数名で担当し、1 名が複数プロジェクトを兼任することも多いです。

QA チームの役割は、テスト作業の見積もりから、テスト戦略、テスト計画、テスト設計、テスト項目作成、テスト実施、バグ分析、品質報告といった QA に関わること全般の他に、リスク管理やプロジェクト終了後の振り返りのファシリテートや、オフショアの品質改善、セキュリティチェッカー等の開発ツール普及活動など多岐にわたります。プロジェクトに対する QA 活動については常に全部を漏れなくやっているわけではなく、プロジェクトの要求に応じて一部だけを担うことも多いです。

成り立ちの経緯

このような組織になったのには必然的な理由がありました。

プロジェクトの規模が小さくなった

まずひとつ目は昔と比べて多くのプロジェクトの規模が小さくなったということ。

かつて i-mode ブラウザーをはじめとした組み込みブラウザーを多く開発していたころは大規模なプロジェクトが多く、数十名のテストチームがプロダクトに紐づき、数年にわたって維持されていました。メンバーの採用もプロダクトの要請に従って必要なスキルを定義して募集をかけていました。テスターはほとんどが派遣社員か業務委託です。同じプロダクトに継続してあたることで経験値を積み、より効率よくテストをこなせるようになってきます。テスト戦略もプロダクトごとに作りこまれていました。

このころ部署間のテスト人材交流は少なく、基本的にプロダクトが終了するとテストチームは解放されていました。一部の高スキルなメンバーは解放するのがもったいないのでちょうどそのときに需要があった部署に異動することもありましたが、需要があるかどうかは運任せでした。

そこに転機が訪れます。組み込みブラウザーの需要が激減し、事業構造を変えていかざるを得ない状況になってしまいました。そうなった原因は、極々簡単にいうと、iPhone と Android の登場です。いわゆるガラケーはインターネットに接続するための専用のソフトウェア群が個別に必要で、ブラウザーSDK を作れる ACCESS は大いに潤いましたが、スマホには初めから標準でブラウザーが入っていました。ACCESS はスマホ向けの OS も作っていましたが、Apple や Google に勝ち目はありませんでした。ブラウザー以外のプロダクトを生み出す必要に迫られ、多種多様な分野に手を出し、いくつものプロダクトが出ては消えていくことを繰り返していきました。

このような状況になると必然的にプロジェクトの規模は小さくなり、短命に終わるものも多くなります。せっかく確保したテストチームもすぐに解放することになるが、すぐに別種のプロジェクトが立ち上がるとまた募集をかける、ということをしているとものすごく効率が悪くなってしまいます。外部リソースというものは当たるも八卦当たらぬも八卦、運よくいい人が来れば良いですが、数日でバックレするような人が来ないとも限らないのです。業務委託にしても最初は次の受注につなげるためにその会社の中の一軍が来るけれど、次の案件では気が緩んで二軍三軍をよこしたりします。せっかく良い委託先を見つけたと思って他部署に紹介したりもしたのに、二軍をつかまされてしまった部署から苦情を受けた

こともありました。世の中そんなものです。優秀な人はパレートの法則で 2 割しかいないのです。

そのことが身に染みたころ、優秀な派遣のテスターは担当プロジェクトが終わるとともに解放するよりも、何とか他部署に異動させてでも維持して、来るべき次の案件に備えるのが得策という考え方になっていきました。ひとつのプロダクトが終了しても、別のプロダクトのテストチームに吸収され、また別のテストチームと合併し、次第に一か所に集まっていったのです。新たなプロダクトが立ち上がると、そこから人を出して作業にあたり、終わると元に戻ってくるということを繰り返すようになりました。

それにソフトウェア開発というものは往々にして遅延します。実装が遅延を重ねテスト期間が後ろにずれたり、お客様の都合で突然リリースが早まったりします。規模が大きいプロジェクトならそれなりに予算があるので多少作業に穴が開いたりしてテスターが暇になっても何とか適当な作業を作ってごまかすことができたし、突然人手が必要になっても人数が多ければ皆で残業したり他の作業を一時的に止めたりして何とか乗り越えられたりもしました。

しかしプロジェクトの規模が小さいと予算も少なく、ちょっとでも作業が遅延するとすぐに赤字になるし、そもそもテスターが 1 名しかいなかったら他に止める作業など存在せず、残業させようにも負荷がかかりすぎて疲弊させてしまいます。来週すぐに人が必要と言われてホイホイ出せるような会社は探せばあるのかもしれないが、出てきた人は当然そのプロダクトのことは 1 ミリも知らず、教えられる人がいるならその人がやったほうが早いという状況になります。

だったら社内で何とか人をやりくりするほうが効率的ですし、あるプロジェクトが遅れたら別のプロジェクトの作業をやれば、無駄にコストを消費することもありません。複数のプロジェクトでテスターを融通できる状況であれば、どこかのプロジェクトはちょっとくらい止めても何とかなるので、そこから人を引っこ抜いてくれば緊急対応もなんのその。いや、実際は結構きびしい社内調整をすることになるのですが、少なくともコントロールの範囲内で事が進む安心感はあります。このような必然性があって、テストチームは 1 つにまとまっていくことになりました。

プロジェクトの分野が増えた

ふたつ目はプロジェクトの分野が増えたということ。

組み込みブラウザー全盛だったころは、プロジェクトの分野といってもたいていはブラウザー等の組み込み向けソフトウェアを何かの機器に移植しカスタマイズするためのプロジェクトでした。技術分野も似通っていてテストに必要なスキルセットも一定の範囲に収まっていました。

その後ブラウザー以外に手を出し始めると、Android アプリ、iOS アプリ、Web サービス、バックエンドと扱う技術分野や対象とする業界も増えていき混沌としていきました。

違う部署で平行して別の Android アプリを作るような場合、それぞれでテストしていると同じようなところでバグが出たりします。プラットフォームごとにバグが出やすい部分というものがあります。さらにプラットフォームは当社の都合はお構いなしにどんどんアップデートされ、公式情報にないところで勝手に機能が追加されたり変わったり削除されたりします。

そんな状況だと、あるプロジェクトで経験したトラブルを他のプロジェクトに共有したほうが効率は良い。かつては部署間のテストチームの交流はほとんどなかったが、各部署のテストチームが情報交換するような定例会ができて徐々に交流するようになりました。

新たに経験する分野でどのような品質目標を立てるべきかもこのような場で知恵を出し合っていきました。事業構造が不安定になった状況では各部署に散らばった QA 担当者は助けを求めるように集結するのはここでも必然だったのです。

QA 組織の誕生へ

やがて組織としても QA 担当者を集めた部署ができることになり、プロジェクト間でテストに関わるノウハウの共有が行われるようになりました。人数が増えることでプロジェクトに対するテスターアサインの柔軟性が高まり、様々な種類のプロジェクトに関わる機会が増えることでより広範囲なノウハウの蓄積が行われるようになりました。

誰かが現在の QA 組織の形を意識的にデザインし変化させたのではなく、社内の開発チームの要請にそのときどきで応えていたら結果的に現在の組織になっていた、というのが正直なところです。ただし流動性が無ければ変化は起きません。常に状況に適応させよう変化させようという意識が皆それぞれにあって色々動き回ったから、最適な平衡状態に落ち着いていくというということなのかもしれません。

現 QA 組織の特色

流れ着いた現 QA 組織ではありますが、強みが多いことも分かりました。

ノウハウの横展開ができる

社内のあらゆるプロジェクトと関りを持つので、あるプロジェクトで試した手法がうまく行った場合、別のプロジェクトにそれを紹介することが日常的に行えます。

開発者というものはプログラムに関するノウハウは共有したとしても、品質を高める手法やプロセスに関する改善事例などは共有されにくい傾向にあるように思います。

QA 組織はプロジェクトに入り込んで実践を通じてノウハウを共有することができるので効果的です。

コミュニケーションハブとして機能する

開発チームが機能単位で分かれていたり、国内の外部委託やオフショアと共同で作業に当たることも多いが、それらチーム間でコミュニケーション不全による問題が発生する頻度がとても多い。要件や仕様変更が正確に伝わらなかったり、あるチームで発生した課題が他のチームに伝わらず解決が遅れることもあります。

QA 組織はそれら全てのチームとやり取りしてテストを組み立て実行するか過程で、コミュニケーション不全に気付くことができ、情報がうまく流れるよう助けることができます。

プロジェクト間の品質に一貫性を持たせられる

開発部門の内部に組み込まれている QA 組織の場合は、所属部門の利害や思惑や雰囲気に左右され、品質判断にもバイアスがかかります。同じ会社なのにある部門は品質基準が厳しいが、隣の部門ではアジャイルの名のもとに基準が緩くなったりします。世間から見たら同じ会社なのに製品の出来栄えがバラバラでは、安心して仕事を任せようという気にならないはずです。信頼の根源にあるのは期待の再現性だと考えます。

1 つの QA 組織がプロジェクトに横断的に関わることで、ACCESS として常に再現すべき一定の品質をプロジェクトに織り込むことができます。

かといって課題がないわけではありません。
やりたくてもできていないことはマウンテンほどあるのです。

テスターや QA 担当が集まってできた QA 組織という生い立ちから、自動化や SET (Software Engineer in Test)、DevOps のようなエンジニアリング寄りのスキルが弱い。開発エンジニアの中から QA を組織することも必要ですが、現状その流れにはなっていません。かつては開発者の中にもテストを強みとしていた人達もいましたが、残念ながら当時の社内に活躍の場が作れず外に旅立って行ってしまいました。ここは今後考えていきたいです。

おわりに

何年か前友人に「ACCESS はベンチャー企業の集りみたいだね」と言われたことがありました。確かに業界も技術も異なる事業をいくつも同時に立ち上げている中小企業はそんなにないのかもしれません。大企業で複数の事業をやっているところは多いが、そちらとはまた事情が違いそうです。

「ベンチャーの集り」のような会社の QA 組織は、それぞれが独立したベンチャーのような社内プロジェクトに対して、その局面に合わせた最適な品質保証プラットフォームを提供し、ACCESS として信頼される一定の品質を確保することで、ACCESS が社会に受け入れられつつ継続して新たな価値を生み出し続けられるようにすることが、求められる役割なのではないかと思っています。(了)

あとがき

こういうのを書くのは初めてだったのですが、ここ数年を振り返る良い機会になりました。今も少しずつ変化し続けているので、また数年後振り返ってみたいなと思いました。（山本）

楽しく書きました！特に座談会はメンバーのことを知れてよかったです。普段から役に立つのか立たないのか分からないことばかり考えていますが、それが誰かの足しになれれば嬉しいです。（鈴木）

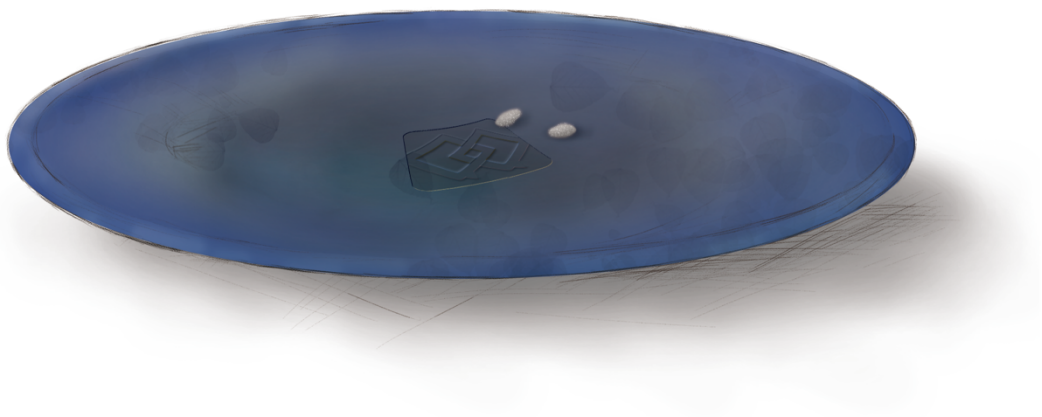
たまご色と米粒に思いを込めました。
お皿の模様が気になった方は、是非「Linkit」でご検索を。
（村上）

4 コマに登場する宇宙人キャラ「こすもじん」が気に入っています。
これからもたくさん描いてあげたいです！
（よしだま）

なぜ異世界で、なぜ寿司なのか？最強テスターの正体とは！？
それは次回作で明らかになるとかならないとか。
「別に、技術書典８にも出てしまっても構わんのだろう？」（入井）

ACCESS QA 同人誌 #01
『異世界転生したら最強テスターになっちゃった』

発行者　株式会社 ACCESS 技術書典同好会
著者　山本竜哉　鈴木香澄　吉田真季　入井圭介
表紙デザイン　村上亜希子
協力　株式会社 ACCESS 品質管理室
発行年月日　令和元年九月二十二日
印刷所　ねこのしっぽ



Gochisousamadeshita.