エンジニアノウハウ集

実際にやってきたこと

古城智雅

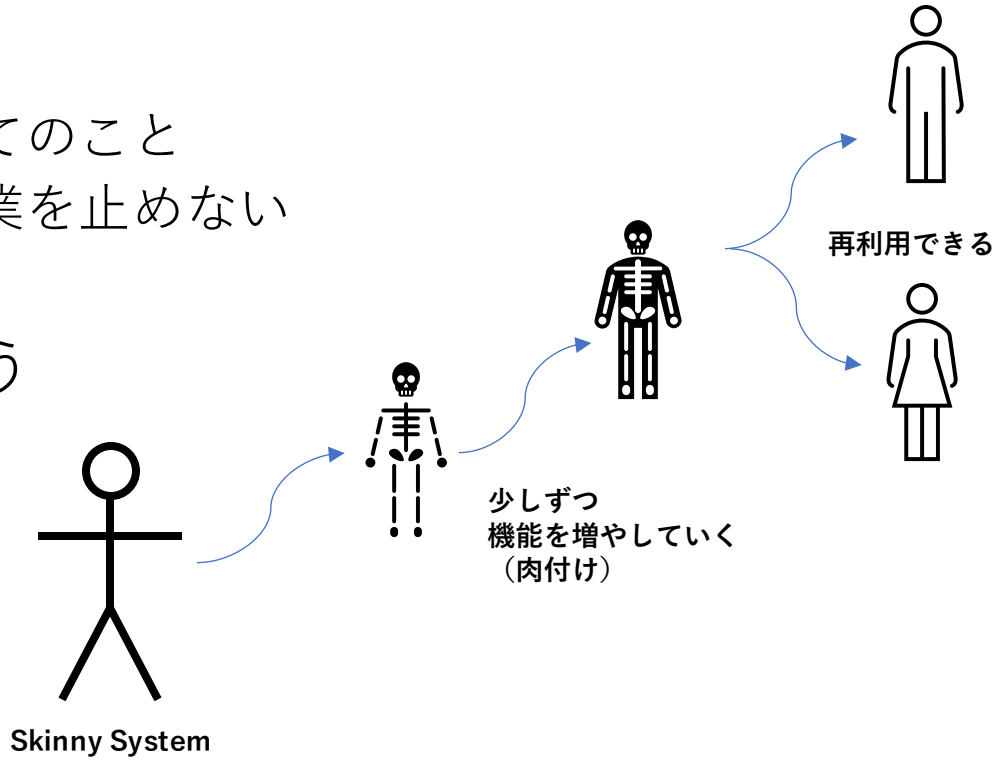
<Tomomasa Kojo>

予定の半分で完成を目指す

- スケジュールには失敗することをあらかじめ含んでおく
  - 半分で完成を目指せば失敗してもリカバリ可能
  - 失敗を乗り越えたものは良いものです
  - 無事に失敗せずに早く終わったら性能を上げるリファクタリングをすれば良い
- 作業期間が短いなら簡単な仕組みを考えるべき
  - 既存の技術をうまく流用（OSSとか）
- つまり見積もりは全速力の倍ぐらいは確保しておく
  - 作業時間をベストタイムで見積もるな
  - 100m走でマラソンしたら死ぬ（でもそういう人多いでしょ）
- 完了予想時間は必ず着手してから申告せよ
  - 人は未来の時間を過信する
  - あなたが明日予定している作業量と、昨日やった作業量は同じですか？

完成に近づけるサイクル

- 部分を完成して次に進むより、未完成の全体を完成に近づける
  - スキニーモデルに肉付けしよう
- そしてサイクルを繰り返して完成度を上げていこう
- 完璧でなくてもいいから動くモノを出す
  - これは「商品」という事ではなく「部品」としてのこと
  - 自分のリリース物の利用者（他の開発者）の作業を止めない
- 早く終わったら、まずは休んで英気を養おう



コードは再利用されるつもりで

- あなたが作るコードは誰かが再利用する前提で
- APIにはアーキテクチャの意図が現れなければいけない
- APIは適切か？API変更は利用者側のコード変更を強要する
- スキニーモデルでもAPIは確定してスタブ実装すること
- 提供するAPIはテスト済みが最低限の礼儀（スタブ実装でも）
  - テスト済みの実績あるコードを利用するから、利用者はテスト範囲を狭められる
  - だから提供するあなたのコードは安定動作している必要がある

百聞は一見に如かず

- エンジニアは画を描こう
- 意図が固まらないと画にはならない
- 会話だけの空中戦は誤解のもと
- 向かい合って言いあうより、一緒に画を眺めるスタンスで
- 特にテレコンやSlackで設計を進めるときには必須

構造を常に予想する

- 自分がこの機能を実現するならどうやるか、それは美しいか？
- そして実際の構造を調べて、予想と答え合わせする
  - 違いを明確にする
  - 気づかなかったPros/Consがあったはず
- これを常時無意識で考えるクセをつける
  - なんなら自分の仕事以外の領域でも良い
  - 広く浅くをきっかけに広く深くなればいい
- 底力ってのは複数の領域の掛け算ができるか
  - ぶっちゃけこの習慣がいちばん効く
  - 未知の技術の理解が段違いに早くなる

予想するために必要なこと

- 勉強しないと視野が狭くなる
- 勉強しないとわずかな知識だけで理解しようとするので、あり得ない仕組みを仮定して「分からない」を埋め立て始める
- 分からないものの多さが分からないから、なんでも知っている気分になってしまう
- ハンマーしか持っていなければ、全ての問題は釘に見えるものです

- だから構造を予想したら必ず答え合わせをすること
- そして知らなかった仕組みを知ること

デバッグは想像力

- 動的、静的、両面から不具合の動きを見る
- 仕組みを知れば想像力のレベルが上がる
- 仕組みに対する解像度がどれくらい高いか？
  - 例えば「Linuxコンソールでキーボードを押して文字が表示される」現象をどれくらい分解して説明出来る？
  - 知らなくてもシステムとして利用すれば機能の実装はできる
  - でも不具合が起きたときどこまで特定出来るかは理解の分解能による
- 結果から原因を遡るのは難しいが、逆は非常にシンプルなことが多い
  - 予想した原因が結果に結びつくまでにレアな条件が必要なら、多分間違っている

全体の作業を一気通貫で考える

- なぜこんな要求が来たかを確認しよう
  - 自分に与えられた領域だけを実装をするというスタンスではダメ
  - もしかしたらもっと適切な機能やAPIがあるかもしれない
- こんなの素人でも作れるというシンプルさでちょうど良い
  - 「面倒くさい」と言った時は多分やること（工数）が見えている
  - 「がんばります」と言ったときは目処が立っていないので要注意

- 特に低レイヤーほど「なに作れば良いか？」を利用者に聞いてはいけない
- 「どう使いたいか？何に使いたいか？」を聞いてAPIを提供するべき

測定による見える化

- 必ずベンチマークを取ってトレードオフを考える
- 大きなボトルネックからターゲットにする
- そのボトルネックを解消するために必要なことは？
  - システムのリソースが足りない？
  - 工数が足りない？
  - 時間が足りない？
- これらを解消するために動くべき人は誰？
  - エンジニア？
  - マネージャー？
- 見つかったボトルネックが生まれた理由は明確にすること

必ず運用を考える

- 面倒なことは誰もやらない
  - ハードルを上げすぎるとくぐるヤツが出てくる
  - 簡単でもやらない人がいることを意識する
  - 通知すればみんなが対応するとか夢を見てはいけない
  - だから定型化して自動化する
- それでもわざわざ手を動かして作業が必要なら、どういう利益（不利益）があるのかを提示する必要がある
  - 管理するためにルールを増やすというのはウマくない
- 人手が掛かる運用に永続性はない

約束は守る

- 特に納期は絶対
- 開発初期は性能を落としてでも納期は守る
  - あなたのリリースを待つ開発者がいる
  - 特に低レイヤーになるほど待っている人が多いことを意識する
  - 約束を守り続けると信頼になる
- 性能を上げるのは次の段階、まずは動くものを出す
- 納期に合わせてスキニーモデルのスタブをどれだけ実装できるかと考える

納期か、品質か、コストか

- イテレーションの前半では納期を最重視
  - 開発の前半ほどスケジュールが甘い傾向がある
- 後半からは品質、もちろん性能も含む
- コストを理由にこれらの2つを犠牲にしない方がよい
  - 過去を振り返ってもろくなことにならない

- 機能充分、価格最低が基本
  - 機能充分を実現する方に知恵を絞りたい

近しい技術知識を繋げていこう

- 例えばLinuxなら
  - ドライバ→Kernel→例外処理→タスクスイッチ→プロセス→メモリ
  - Kernel→メモリ→プロセス→userlandの知識
  - Kernel→ファイルシステム→SDIO→NAND
  - ARM→AXI→I/O→例外処理→TrustZone→セキュリティ
  - 書き切れないぐらい縦横に関係することがたくさん
- 直接は役に立たなくても仕組みに興味を持とう
  - TrustZoneは例外処理とタスクスイッチとAXI等々を知らないと理解出来ない
  - TrustZoneは暗号モジュールではなく、アーキテクチャだと言えるとカッコイイ
- 仕組みを知ると次の段階の理解が早くなる
- それはいつか武器になる、そして武器を増やそう

段取り重要

- プロジェクトの進め方だけではなく「作り方」も段取りが重要
- 他の人の作業完了を待つようではもったいない
  - 個人の作業速度を上げるよりパイプラインを増やそう（できるだけ並行作業）
  - でもどうしても相互関係で相手の作業完了を必要とすることもある
  - だからスキニーモデルが重要になる
  - APIをスタブで提供するというのはプロジェクトを前進させる有効な手段です
  - 初回リリースでAPI決まってないのは（もちろんエラーケースも含めて）、設計が出来ていないということ
- リーダーの仕事だと丸投げせずに作る側も全体を考えながら段取りをとるクセをつけておきましょう
- あなたもいつかリーダーになって泣くことになるので

作業時間にも汎用バッファを

- 2時間/週ぐらいは予定を入れない時間を確保するのがオススメ
  - 考える時間を確保して、先のことを考えるクセをつける
  - 遅れた作業の巻き返しに使ってもいい

- 毎日業務終了前30分ぐらいはラップアップしよう
  - 明日の午前中のゴールデンタイムに何をするか事前にリストアップしておく
  - 午前中は手を動かすのに最適の時間、すぐに作業に入れるようにしておく
  - 会議入れるのはもったいない

- 効率が大好きなエンジニアとライフハックは相性がいい
  - 時短して得られる時間は「考えること」に利用しよう
  - 時間単位の作業を効率化して、空いた時間を「作業量を増やす」ことに使うのはうまくない

失敗から学ぶ

- 全力でやった失敗から学ぶことは多い
  - 振り返りやポストモーテムは絶対にやる（個人でもやる）
- 失敗することが前提で段取る
  - だからスケジュール半分で完成を目指す
- 社外に影響を与えない失敗なら受け入れよう
  - 逆に社外に影響を与えるレベルなら早めにアラートを
- 悪い報告にいちばん価値がある
  - 良い報告は遅れても構わない
  - 報告を受ける側になってもこの意識を持とう
- 失敗したときにこそ次のチャンスを与えるべき
  - 失敗に価値が生まれるのはリベンジするとき

リーダーや上司は笑ってろ

- 笑っているのはリーダーの義務
- しかめっ面しているリーダーに悪い報告はしにくい
- 悪い報告にこそ価値があるのだから、報告を受ける側はいつも笑って、悪い報告が出やすい雰囲気を作るべき
- よく言う「報・連・相」は、それを実現しやすい環境を作るということに軸足を置くべき

修羅場に強い人とは

- 修羅場を「修羅場っぽく」しない
- どんなに窮地に追いやられても、人間関係だけは壊さない配慮を最後まで通せる人
- 怒っている人に正面からぶつかっていくのではなく、いかにさらりと回避できるか
- 狡猾さと肝の座り方が必要
- 人と人を向かいあわせない、人と人を同じ方向に向かせる
- これらを意識して経験すること

グレーゾーンは黙ってやれ、うまくやれ

- グレーゾーンの判断を仰いだとき「OK」となるわけがない
  - だから自分で責任の取れる範囲は自分で判断しろ
- グレーゾーンには手早くできるとか意味があるはず
  - 悪いことをやれってことではない
  - 明文化されていない部分の落としどころをうまく探せ
- 社員は例外の判断を自分でやれ、上にいちいち聞かない
  - 判断を委譲するために担当者になっているハズ
  - 逆に担当者に任せるということは判断も任せよう
- ただし自分で取れる範囲の責任は自分で取れ
  - それを超えた責任はリーダーや上長に任せよう

知っている人を知ろう

- 仕事のスキマを埋める人は必要
- こぼれている仕事を拾える人になろう
- give and takeのgiveは多めに
- あなたのこれまで仕事をしてきた結果と信頼が人脈になる
  - 人脈とは「あなたを頼ってくれる人たち」のこと
  - 人脈で仕事をするわけではない

コミュニケーションの増やし方

- コミュニケーションは交換ではなく、お互いの贈与
- 知識や経験は渡しても減らないので、コミュニケーションするとお互いに増える
- そうやって仕入れた知識や経験が別の場所で使えるのでさらにコミュニケーションしやすくなる

- 「勝ちたい」とか「負けたくない」というのはいい
- 「損したくない」というのはオススメできない
  - 知識や経験が共有ではなく独占になる

エンジニアとしての審美眼

- 違和感を大事に
  - 直感はハズレるけど違和感はハズレない
- キレイはバカにできない
  - 正しいものを大量に見ないと審美眼は得られない
  - 枯れたOSSコードなどを読みましょう、大量に
- センスとはこれまでの経験から最善手を無意識に選択した結果
  - だから経験と手数を増やそう
  - 「質より量」を目指す時期は必要、思ったより長期間必要

全てを疑え、特に自分自身を疑え

- たとえ現象がレアケースでも原因がレアである保証はない
- 「ここは大丈夫」というところがいちばん危ない
- KernelやOSSなど他責のバグより自責を疑え
- 関数の返値は必ずチェック、見ない理由ある？

会議は先読みの訓練の場

- 若い頃に同行する会議では発言することもなく、なぜ呼ばれているか分からなくなることもある
- 内職するのもアリだけど「先読み」の訓練をしてみては？
  - この人はどんな立場でしゃべっている？
  - この人のメリットは？デメリットは？
  - この人の目的は？
  - この人の背景となる組織の目的は？
  - こちらのメリットは？デメリットは？
  - こちらの目的は？

  - 逆に言えばこれらが分かるようなメモや議事録が取れるようになると完璧

議論では相手の話を聞く

- 説得するには相手の言葉と理屈を使って話せないとダメだから
- 自分の言葉と理屈だけでは、同じ知識を持っている人にしか通用しない

- 最後は必ずラップアップして決まったことを全員で再確認
- 曖昧にしておきたい人達はここでもにょもにょする
- こういう時に曖昧にした事は必ず終盤でトラブルのもとになるので突き詰めよう（こういうところの忖度はいらない）

常に手抜きを考える

- ひたすら言われた通りに作業するだけではダメ
- 同じ作業を3回やったら自動化を考えよう
- 10分の作業を、1時間かけて自動化して、100回まわす
- ミスが許されない仕事は機械にやらせる、ミスをするのは人の仕事です

上手いひと

- 「どんな時、どうすれば良いか」といった知識は誰でも簡単に学べる
- 一番難しいのは「今がどんな時か」を感知すること
- これは知識としては学べない
- 経験と圧倒的な練習量が必要
- 現在の位置や状態を的確に把握できれば、もう「上手い」も同然

この仕事をする限り勉強は終わらない

- データは情報ではない
- 情報は知識ではない
- 知識は知恵ではない
- 困ったとき（トラブル時）に役に立つのは知恵です

- 答えを知りたがるか、仕組みを知りたがるか
- 5年、10年経ったとき、この違いは大きいです

好きこそものの上手なれ

- 最高の戦略は努力が娯楽化すること
- Try & Errorの回数をどれだけ増やせるか

- 同様に憧れというのも、無限のエンジンとなる
- ほめて伸ばすのはこういうのが理想的（甘やかしとは違う）
- また負けず嫌いというのも同じで大変良き

- 見れば奮い立つ、ああなりたいという存在
- 追いかけたい背中が隣にあるというのは幸運なこと
- そういう人はいますか？そういう人になれていますか？

理想

- 技術のピタゴラスイッチ
- 既存の技術の組み合わせで、簡単に、楽しく、便利に暮らす
- 組み合わせはキレイに切れて、キレイに繋がる
- 技術の流行が変わっても、その部分を取り替えてアップデート

- 不便を感じている人がいるのなら、それは工学の敗北です
- 負けたままで、いいの？

おまけ：裏ノウハウ集

- 「ご存じの通り～」といいながらプレゼンすると質問が減る
  - 「分からない人います？」とすると質問が減る
  - 「気になるところあります？」とすると質問が増える
- 正論は守りのときに持ち出そう
  - 正論で殴り合うと落としどころがなくなる
  - 頭を下げるのは勝っているとき
- リーダーは「上に厳しく下に甘く」してればだいたい回る
  - 逆をする人はリーダーになってはならない
- 夜に設計して、朝に実装する
  - 夜に書いたラブレターは翌朝読み返して出せというのは設計にも当てはまる
- リーダーは「命令を出す人」ではなく「どうしたらメンバーが命令通りに動くのか」を考える人です
  - 進捗確認すれば人が動くと思っているようでは少し足りません
  - 自発的に動く仕組みを考えるのが仕事です

おまけ：裏ノウハウ集

- 人は資産でありコストではない、資産とコストを混同しないように
- 「上手くいったら優勝、失敗しても準優勝」これぐらいの気持ちでちょうどいい
- 人がいちばん騙されるのは過去の自分の経験
- AかBかで悩んでいるとき、全く別のCがより良い答えであることが多い
- 決断したらそれが正しかったかクヨクヨ悩むより、その決断が正しくなるように動け
- 10万円と1000円のワインを混ぜると5万500円のワインが2本出来るワケがないのであるが、人員に関してはそういう計算が起こるので気をつけろ
- 仕様を決めてから納期を決めろ、逆が多いので注意せよ、これはお客様のためである
- わかり合えない人とは距離を取る
- 好きな人には親切に、嫌いな人には丁寧に
- 頭を使う仕事をする人は睡眠が重要、寝るとだいたいの問題は解決する

おまけ：裏ノウハウ集

- エンジニア座敷わらし説は真実だと思う（面白い仕事と良い環境をお供えすると優秀なエンジニアが寄ってくる、逆もまた然り）
- 20時過ぎて書いているのはコードではなくバグ

改訂履歴

- 2024/10/29　初版（古城智雅）
- 2024/11/01　修正（古城智雅）
  - 「予想するために必要なこと」に加筆修正
  - 「グレーゾーンは黙ってやれ、うまくやれ」に加筆修正
  - 「知っている人を知ろう」に加筆修正
  - 「常に手抜きを考える」に加筆修正
  - 「上手いひと」を追加
  - 「おまけ：裏ノウハウ集」を追加
- 2024/11/12　修正（古城智雅）
  - 「作業時間にも汎用バッファを」を追加
  - 「おまけ：裏ノウハウ集」に加筆修正