

「機能概要書」でもっとステークホルダーに歩み寄る開発へ！

2026/08/28

株式会社オージス総研 組み込みソリューション部 島崎 雄貴

第28回 組み込みシステム技術に関するサマーワークショップ(SWEST28)

はじめに

皆さんは普段、どのような仕様書を書いているのでしょうか？
「要求仕様書」？ それとも「機能仕様書」？ あるいは「設計仕様書」？

せっかく仕様書を書き上げたのに、ろくに見てもらえなかったり、
「分からない」と言われてしまった経験はないでしょうか？

あるいは一旦承認してもらったのに、結局覆されてしまったり
大変な思いをした経験はないでしょうか？

はじめに

今回は実業務の中で編み出した「機能概要書」を軸にして
「作り方を、つくりかえた」経験についてお話しします。

仕様を決めるプロセスをスムーズに進めて、「みんながうれしい」開発
に一步近づくヒントになれば幸いです。

自己紹介

株式会社オージス総研
インダストリ&ファイナンス事業部 組み込みソリューション部

島崎 雄貴



X @shimazakky

■ 経歴

- 2010年オージス総研に新卒で入社、組み込みソリューション部に配属
- 以降、産業機器メーカーをお客様としてソフトウェア開発に従事
- 現在は主にプロジェクトマネジメントとレビューを担当しつつ、研修教材開発などにも携わる

■ SWESTとの関わり

- SWEST25で初参加し、今回で4回目の参加

機能概要書の誕生ストーリー

前提の共有

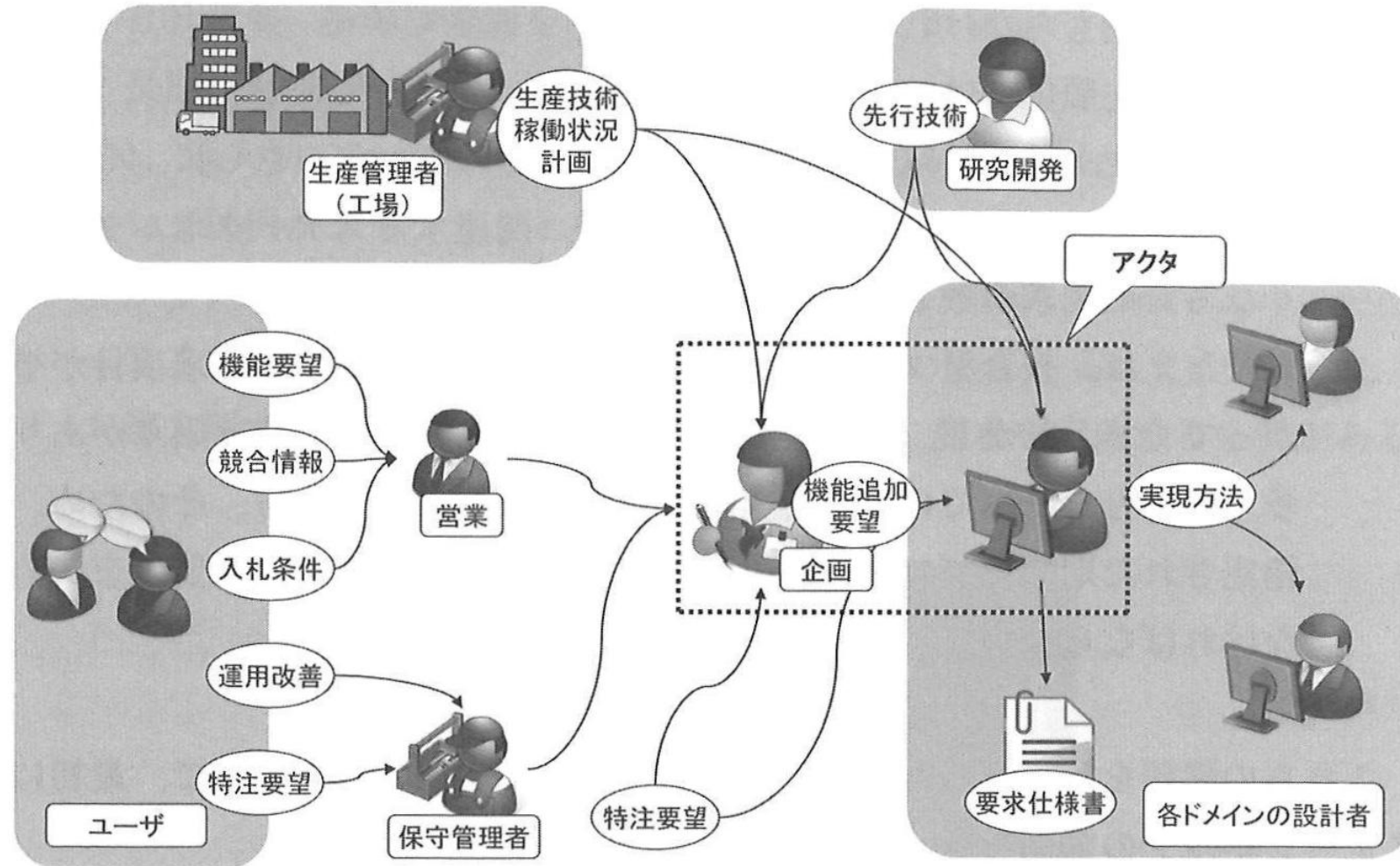
- 一口に「組み込み」と言っても環境は様々
 - 今回の話がフィットしない場合もあるかと思いますがご容赦ください
- とある産業機器メーカーの開発現場でのお話
 - 工場で使われるような、ある種の加工機を開発
 - 私はソフト開発チームにベンダーの立場で参画
- 開発プロジェクトのほとんどはいわゆる「派生開発」
 - バージョンアップでの新機能追加、不具合修正が主
 - 新機種も基本的に既存機種をベースに作る
 - 完全に作り変えるのは10年に一度ぐらい



前提の共有

- プロダクト要求は、商品企画を通じて開発に伝えられる
 - エンドユーザーや設備メーカーの要求が営業を通じて商品企画に集められる
 - また、社内の製造部門や、保守担当部門からの要求も商品企画に集められる
 - 商品企画がとりまとめた要求をもとに、新商品やメジャーバージョンアップの企画が作られる
- その他、大口ユーザーからの要望なども入ってくる
 - 「こういうことができるなら大量に買ってもらえるが、半年後に納品するときに間に合うか？」といった話もしばしばある

プロダクト要求に関連するステークホルダー



出典: (一社)情報サービス産業協会 REBOK企画WG 編. 要求工学実践ガイド. REBOK シリーズ2, 近代科学社, 2014.

機能仕様書「以前」からの出発

- 時は十数年前、最初は体系的な機能仕様書がない状態からのスタート
 - 変更したところだけの説明書と、変更したところだけのテスト
 - 体系的な仕様書はなく、事実上マニュアルが仕様書
 - それでも、小規模な改修であれば「なんとか」なっていた
- しかし、イチから新機種を開発するときに限界を迎えた
 - 新しいハード × 新しいソフト × 新しい仕様
 - 何が正解か分からない状態で、テストが不十分に
 - 多数の不具合が発生し、修正に追われた

テストベースとしての、詳細な機能仕様書

- どうか新機種を発売した後、反省を踏まえて仕様書を整備した
 - テストが作れないという課題から出発して改善に取り組み
 - お客様や第三者評価会社とともに、機能仕様書の整備にあたった
 - テストケースを書いていると出てくる「こういう場合はどうなるのか？」という疑問に答えられるような、「検証」(Verification)に適した機能仕様書になった
- 機能仕様書の記述が充実するとともに、テストも充実
 - 新規不具合は明らかに減少していった

これで「めでたしめでたし」かと思いきや…

詳細すぎる機能仕様書の弊害

新機能の開発で、機能仕様書を改訂して企画部門に持っていくと…

■「細かくて読み切れない」と言われてしまった

- 書く側は頭に入っているので読めるが、頭に入っていない状態で読むのは辛かった
- 漏れは少ないかもしれないが、全体像を俯瞰するのが難しい
- エラーになる詳細な条件などは、はっきり言って企画部門は興味がない

■ 資料を事前に読んできてもらうのは無理

- なぜならば、一日中会議で埋まっている
- どうしても会議の中で伝えきらなければならない

さて、どうしよう？



機能概要書の「発明」

■ 何かいい方法はないものか？

- アイデアを求めて、USDM、XDDP、アジャイル、要求工学などの本を読み漁る日々

■ 最終的にヒントになったのは中央官庁の政策説明資料

- 何十ページにも及ぶ詳細資料とは別に、数ページの「概要版」が置かれている
- 忙しい人に素早く理解してもらうために作られたもの
- これを見てひらめいたのが「機能概要書」

※ USDM = Universal Specification Describing Manner
清水吉男氏が提唱した要求仕様の表現方法

※ XDDP = eXtreme Derivative Development Process
同じく、清水吉男氏が提唱した派生開発に特化した開発プロセス

機能概要書の「発明」

- 「見て欲しいところだけ見てもらう」という発想に大転換
 - 既存の枠組みにとらわれず、思い切って「成果物を増やす」ということをやってみた
 - 些末な詳細や「重箱の隅」は一旦置いておいて、本筋の部分に集中して議論
 - そこで合意した内容をもとに、詳細な仕様書を作ることにした
 - 逆に企画の興味がないところは、開発側の一存で決めさせてもらう
- 忙しい人との合意を「一発」で取るために進化
 - 今日この場でOKをもらわなければ、一週間遅れてしまう
 - 細かい点は残したとしても、その場で大筋での合意を取り付けるという目的で作成

機能概要書の「発明」

■ 最初は商品企画と「握る」ために作ったが…

- 意外と他のシーンでも役に立ち、次第になくてはならない存在に
- 例えば:
 - マニュアルを作るときの元ネタに
 - 試作品をユーザーに提供するときの簡易マニュアルに
 - テスト設計のレビューを品質保証部門にしてもらう際の「頭出し」に

■ 作る成果物は増えるが、十分お釣りがくる

- 書くことによって作る機能の解像度が上がり、課題が事前に見つかる
- 事前に課題をつぶすことで、後続する開発がスムーズに進む

では、その「機能概要書」とはどんなものなのか？

機能概要書の書き方と具体例

具体例の題材について

- 実案件の資料は出すことができないため「電源制御ボックス」を題材に説明する
 - COVID-19の対応でリモートワークを余儀なくされた際に活躍したもの
 - 4口のコンセントがあり、遠隔で開発基板の電源をON/OFFすることができる
 - 今回はこれに「コンセント占有機能」をつけるというシチュエーションで例示

(参考)製品の概要

- 電源制御ボックスは、組み込みソフトウェアのリモート開発をサポートするために、開発ボードの電源を遠隔制御するものである。
 - AC 100 Vのコンセントを4口備えており、TCP/IPを経由したコマンド送信(プロトコル非公開)によってON/OFFすることが可能。
 - コマンド送信はPC上のGUIクライアントソフトから行う。



クライアントソフト

コマンド送信



機能概要書の目次例

■ 機能の目的

- この機能は、何のためのものなのか？ どうなったらうれしいのか？

■ 基本的仕様

- どうやって目的を達成しようとしているのか？

■ 設定項目

- この機能には、どんなインプットが必要なのか？

■ エラー仕様

- どんなときにエラーが発生するべきなのか？

■ インターフェース仕様

- どんなインターフェース(GUI, CUI, API, etc.)なのか？

■ 制約

- どうしても我慢してほしいところは？

この2つが特に大事

機能の目的

- この機能は何を達成しようとしているのか、その目的を端的に書く
 - － それを達成することがなぜ必要なのかという背景とともに表現する
- 「機能の目的」を書く目的
 - － 誰にとって、どんな価値があるのかを明らかにする
 - － 開発側と依頼側の間での認識のズレを解消する
 - － 依頼側の持っている情報をさらに引き出す

参考にしたもの：

『アジャイルサムライ』のインセプションデッキより、「我われはなぜここにいるのか？」

「コンセント占有機能」の場合

機能の目的

まず目的を端的に、一文で書く
(一石二鳥のときは2つ書く)

- 使用中のコンセントを他者によって操作されることを防止する。
 - 電源制御ボックスには複数のコンセントを備えており、複数人で同時に作業することができる。
 - しかし、操作ミスや故意によって、現在使用しているコンセントが他の人に操作されてしまう可能性がある。
 - 特にファームウェアの書き換え中や、重要な操作を行っている最中に電源がOFFされた場合には、被害が大きくなる可能性がある。

なぜそれが欲しいのか、どうなったらゴールなのかを語る

ここで開発者と依頼者の認識が合っていることをしっかり確認する
目的が明確になっていることで、仕様や設計を的確に判断できる

基本的仕様

- 機能仕様のうち、ステークホルダーが興味のある領域を中心に記載する
 - ユーザーが認識でき、ユーザー体験を形作る「システムの輪郭」
- どういったアプローチで目的を達成しようとするのかを説明する
 - 根幹的なコンセプトの説明から始めて、徐々に具体的な振る舞いや例外の説明に移る（cf.『Clean Architecture』における「方針」と「詳細」）
- なるべく画面イメージには頼らず、言葉で説明する
 - 言葉の説明だけですんなり理解できる仕様は、おそらく自然な形をしいたい仕様
 - 言い換えれば、電話口で説明できるような仕様が理想
 - 実際、コールセンターで問い合わせに回答するときに伝えやすいというメリットも

参考にしたもの：

『アジャイルサムライ』のインセプションデッキより、「パッケージデザイン」

基本的仕様

- USDM的には、ここで書かれるものが「要求」と「仕様」に相当する
 - 「基本的仕様」を書く過程で、目的を達成するための「仮説」としての要求が展開される
 - 「仕様」のレベルで網羅することは目指さないが、ステークホルダーの興味がある部分については重点的に掘り下げる
 - しかし、後述するように一旦設計レベルまでイメージした上で書くことで解像度が上がる
- 画面に依存しない抽象的な使用手順を示す
 - ユースケース記述に相当するが、あまり形式にはこだわらない
 - 使い方をイメージできて、ステークホルダーからの質問が引き出せるようになっていればよい
- とはいえ、あまりにもイメージが湧かない場合は一度画面を見せるのもあり
 - このあたりは担当者の匙加減

「コンセント占有機能」の場合

基本的仕様（1/5）

根本的なコンセプトを最初に説明する
主要な概念は英語も決めておきたい

- 使用中のコンセントを「占有」(Occupy, Occupation)することによって、他の人が操作できないようにする。
 - 使用する際は、まず対象のコンセントに対して占有操作を行う。
 - 先に他の人が占有した場合、占有操作は失敗する。
 - 強制OFFが有効な場合、占有操作は失敗する。
 - 占有に成功した後でしかON/OFF操作を行えない。
 - 例外として、電源制御ボックス本体での強制OFFは占有を無視して実行される。
 - 占有している間は、占有を実行したクライアント以外からの操作を拒否する。
 - 占有を終了するときには、原則として占有解除操作を行う。

← 影響し合う機能との優先順位

↑ どうやって実現する？
実現方法を考えておく

具体的に使い方がイメージできるように説明する

5

「コンセント占有機能」の場合

基本的仕様（2/5） 個別の仕様には、できるだけ理由も表現する（USDMを参考に）

- 占有を開始するときに占有者のユーザ名を登録し、誰が占有しているか分かるようにする。
(理由)占有者が分からないと、やむを得ず電源を切るときや、譲ってほしいときに誰に連絡すればよいか分からないため。
- クライアントソフトには、各コンセントの占有状態、占有者のユーザ名を表示する。
(理由)どのコンセントが空いているか、誰が占有しているかが分かるようにするため。

「コンセント占有機能」の場合

基本的仕様（3/5）

使われる状況を想像しながら、
喜ばれそうな仕様を考えていく

- 1ユーザーが複数のコンセントを同時に占有することができる。
（理由）複数の機材を組み合わせで使用することがあるため。
- それぞれのコンセントに対する占有は別々に扱われ、個別に占有解除することができる。
- それぞれの占有に対してコメントを記入することができ、コメントは他のユーザーから参照できる。
 - 最大255文字までのコメントを記入可能（改行可）。
 - 機材の種類、用途や使用終了予定時刻などを記入することを想定する。

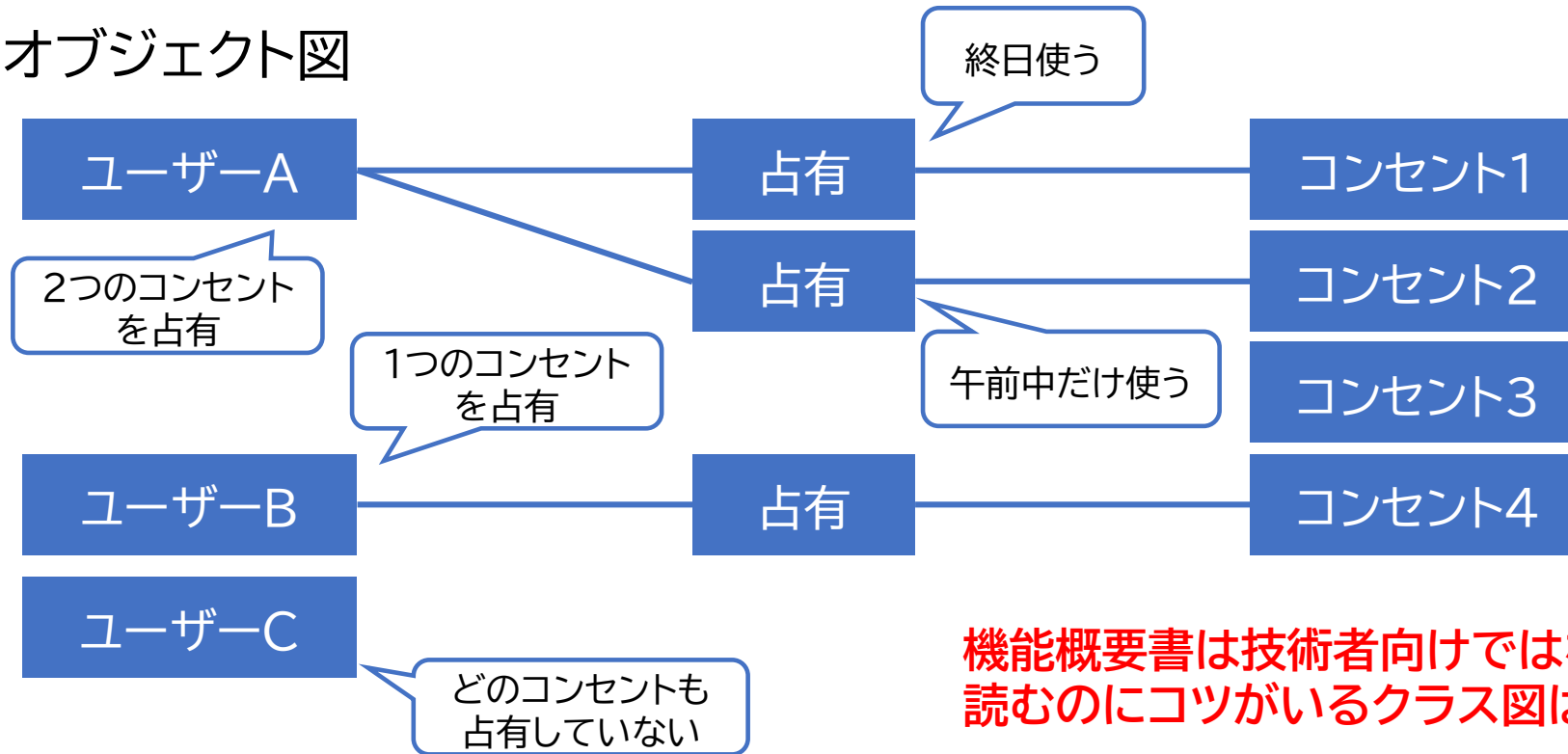
↑ 元々のソフトはコマンドを単発で送る作りだったが、
他の人の使用状況を見ることを考えると、定期的に通信する必要があるようだ

「コンセント占有機能」の場合

基本的仕様（4/5）

オブジェクト図やステートマシン図なども効果的であれば適宜使用

オブジェクト図



機能概要書は技術者向けではないので、読むのにコツがいるクラス図は非推奨

「コンセント占有機能」の場合

基本的仕様（5/5）

こんなときはどうなるべきか？
と想像を巡らせて書く

- 次の場合には、占有は解除される。
（理由）占有したまま作業終了すると他の人が使えなくなるため。
 - 占有者が占有解除操作をしたとき
 - 占有者がクライアントソフトを終了したとき
 - クライアントソフトから一定時間アクセスがなかったとき
 - 強制OFF機能を使用したとき
- 占有が解除されても、コンセントのON/OFF状態は維持する。
（理由）操作後にクライアントPCを終了することもあるため。また、通信途絶により電源OFFされることは避けたいため。

そういえばタイムアウトも
考慮が必要か…

↑ 通信状況が悪くなったときに勝手に遮断したら、かえってまずそうだ
（アップデート中に電源が落ちる可能性があり、機能の目的に反する）

9

設定項目

- 設定項目(あるいは入出力項目)の名称、有効範囲、初期値などを記載
- できれば設定項目や主要な概念の英語名をこの時点で決めておきたい
 - 多言語化するときどれぐらい画面の幅を取るかは意外と盲点
 - 英語よりもフランス語やスペイン語、ドイツ語のほうが長くなる傾向があるため、それらの言語に翻訳する可能性があるならさらに要注意
 - ここで英語を決めておくことでソースコード中の用語がバラバラになることを防げる
 - 日本語しか決まっていないと、ソースコード上の名前を決めるときにそれぞれの担当者が勝手に英訳して揃わなくなることがある

「コンセント占有機能」の場合

設定項目

単に画面実装のためではなく、
ドメインモデルを構築するという意識で

項目名	必須	設定範囲	初期値	説明
ユーザー名 User Name	Y	1～64文字	前回終了時の設定 (※初回は空欄)	ソフト全体の設定。 空欄のままで占有操作を行おうとするとエラーメッセージが表示される。
コメント Note	N	0～255文字	空欄	各「占有」ごとの設定。 用途や占有期間などを入力することで、他の人に状況を伝えられる。 占有を開始するときに入力し、占有中に編集することもできる。

要検討：
終了予定時刻は別項目とするか、自由記述の中で表現してもらうか
別項目にすると、占有解除忘れのアラームなどを実装しやすい。
ただし、必須入力にすると煩雑と受け取られる可能性もあり

その場で議論して決めたいことは吹き出しにしておく

エラー仕様

- どんなときにエラーになってほしいのか？ を検討する
- エラーになってほしいケースと、ユーザーの対処について記載する
 - 動作を止めていいのか、ログに残すだけのほうがいいのか？
- エラーコードや、エラーメッセージはまだ固まっていなくてもよい
 - 開発都合でエラーを分けたり統合したりすることがあるはず
 - これらは最終的にマニュアルに反映されれば問題ない
 - 決まり次第、機能概要書にも反映する
- 設計上必要な「ありえないエラー」は、ここでは一旦置いておく
 - ソースコード上から考えると、「実際には起こらないはず」のエラーも出てくるが…
 - ここでは、あくまでユーザーが対処可能なものに集中する

「コンセント占有機能」の場合

エラー仕様

ユーザーに自己解決してもらうには、
どういう対処ができるか？ で考える

コード	エラー種類	原因	対処
TBA	ユーザー名の問題	ユーザー名が未入力になっている	ユーザー名を入力する
TBA	占有に失敗した	他のユーザが先に占有している	占有者に連絡して占有解除してもらう
TBA	操作が拒否された	通信途絶などで占有が解除された	クライアントソフトを再起動し、再度占有する
TBA	通信がタイムアウトした	応答が返ってこなかった	通信状態を確認し、クライアントソフトを再起動する
TBA	強制OFFされた	メンテナンスのため強制OFFされた	翌日以降に使用する 必要なら管理者に連絡する

エラーコードやエラーメッセージが決まっていなくても、
どんな種類のエラーがあるかが決まればテスト分析が進められるし
マニュアルに書くべき内容の解像度も高まる

インターフェース仕様

- ここでようやく画面や通信、APIの仕様に触れる
 - あくまでこれらは「詳細」
 - 中心となるコンセプトが固まってからでなければ、先に画面を決めても振り回される
 - 画面仕様は最後まで調整が入るつもりで、画面に依存しない設計を心掛ける
- 画面に置かれる要素や配置、画面遷移などは示すが、ここで最終決定ではない
 - 「細かいところは後から変えられる」ようにしておいて「基本的にはOK」という判断を勝ち取る
 - 設計上も、インターフェースの詳細に依存しないように作る
 - 画面の細かいところにこだわって先に進めなくなるのは問題

「コンセント占有機能」の場合

画面仕様

文字の大きさなどは後で調整が入るものとして、
大まかなレイアウト、主要な見せ方について確認

接続画面

新規追加

ユーザー名

テスト太郎

接続

自分が占有している場合「占有解除」になる
占有開始時はダイアログでコメント入力する
他の人が占有しているときは占有開始できない

操作画面

基本的に同じコンセントを連続
で操作するので、操作対象を
ラジオボタンで選択する

ユーザー名: テスト太郎

占有解除



No.1

状態:ON

占有中(自分)

詳細...

占有解除



No.2

状態:OFF

占有中(自分)

詳細...

占有...



No.3

状態:OFF

空き

詳細...

占有...



No.4

状態:OFF

占有中(開発次郎)

詳細...

占有していないコンセント
は選択できない

ON

OFF

占有時のコメント等は
別ダイアログで表示

制約

■ 言いにくいことこそ、最初に伝えておく

- 「今の仕組みではやりにくいこと」、「期待に応えられないこと」はどうしても出てくる
- できないこと、諦めることは最初にしっかり認識してもらう
 - 言いにくいからと言わないままにしていると、後で大炎上の元に
 - 今は言われていなくても、後から「やってくれ」と言われそうなことを先回りしておく
- その代わり、代替策や緩和策を誠心誠意考えて提示する

■ 思いつかなくても、何かあるはず

- 何も思いつかないなら、ユーザーの立場になりきれていないのかも？
- 勝手にどんどん期待値を上げてみると、「さすがにそこまではやらない」ということが見つかるかも

参考にしたもの：

『アジャイルサムライ』のインセプションデッキより、「やらないことリスト」

「コンセント占有機能」の場合

制約

互換性がなくなる、特定の条件は対応不可、etc.
プロジェクトや製品に重大な影響がある事項は「やらない」と宣言

- 旧バージョンのクライアントを使用した場合は電源ON/OFF操作ができず、失敗したことも把握できない。
 - サーバーへの接続自体はできるが、旧バージョンのソフトには操作が拒否された場合の通知を実装していないため、ユーザーから見ると操作が空振りしたように見える。
 - リリース時に旧バージョンのユーザーに対して、バージョンアップを呼びかけることで対処する。
- 送信元の確認は簡易的なもので、なりすまし対策は行っていない。
 - 社内LANで運用することを前提としており、送信元の厳密な検証や、リプレイ攻撃の対策は行っていない。

← 残念ながら、受け入れてもらうしかない

↑ 後付けもできるが、また互換性がなくなるかもしれない
今回「認証エラー」だけ先に入れておけば、導入時の問い合わせが減らせるかも？

設計上の問題で今後もできる見通しがいいのか、あるいは
予算と時間があれば後から解消できる制約なのかを示すと、
交渉もやりやすい

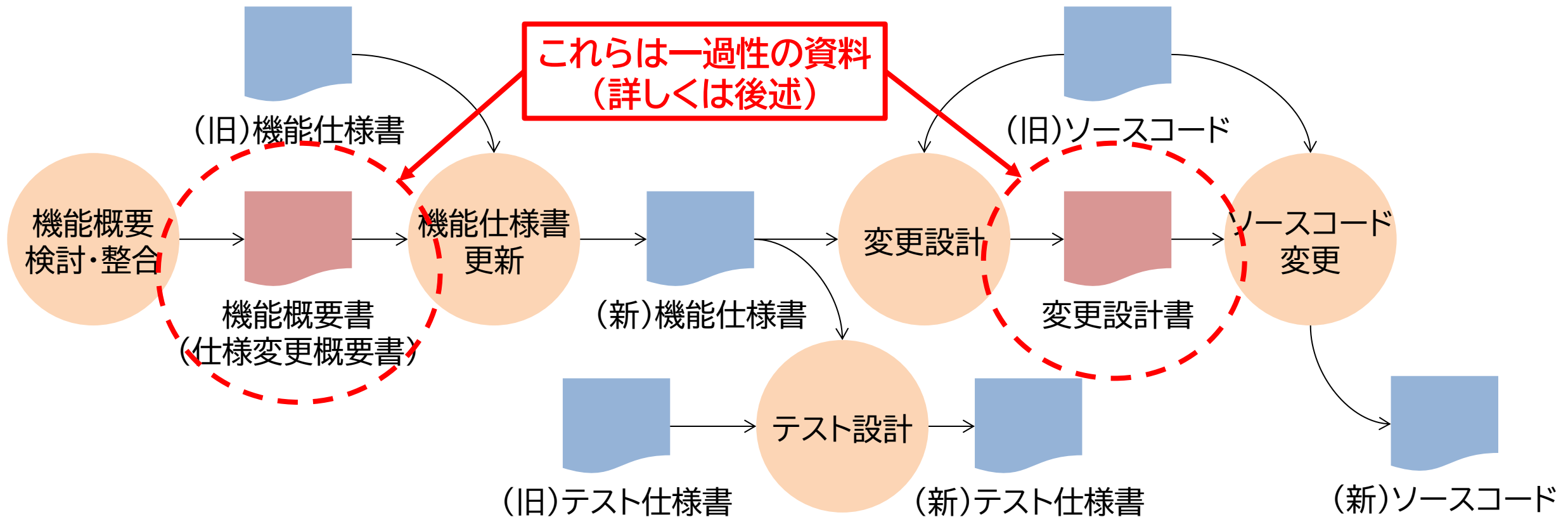
機能概要書の運用

基本的に「要件」単位で作る

- ここで言う「要件」とは、独立してリリース可能な価値提供の単位
 - 次のリリースに載せるか、外すかの判断が可能な単位
 - プロダクトバックログのような形で積まれて、次のリリースに載せるかを議論する
 - 「要件」の例
 - 旧機種ファイル変換機能
 - 通信履歴機能
 - ○○画面の操作性改善
 - 関係の近い要件をまとめて一つの機能として、一つの機能概要書にまとめる場合も
- 「仕様変更」も機能追加と同様に扱う
 - XDDPにおける「変更要求仕様書」と同様
 - 仕様変更の場合は「仕様変更概要書」という名前のほうがしっくりくることも

機能概要書は、一連の開発の起点になる

- まず機能概要書を作り、それを元に機能仕様書や設計書を作る
 - 機能仕様書は、機能概要書で書かれた内容を全て含んだ上で、さらに詳細化・具体化する
 - 機能仕様書が正式な仕様書であり、継続メンテナンスする対象



機能概要書を書きながら、設計を想像する

- 実現のイメージを伴わない仕様は、絵空事になりがち
 - 企画だけで考えた仕様は、実現困難なことがしばしばある
 - 開発側で考える以上、少なくとも実現する方法があることは確認しておきたい
- 特に派生開発の場合、「今のシステムの上で実現すること」は巨大な隠れ要求
 - 互換性を捨てれば簡単に実現できるが、今からでは難しいということもよくある
- 設計内容自体は機能概要書には入れないが、別でまとめておく
 - 仮に「概略設計書」と呼ぶ
 - ざっくり書いたものを開発チーム内でピアレビューして、これで実現できそうかを確認する

機能概要書を書きながら、設計を想像する

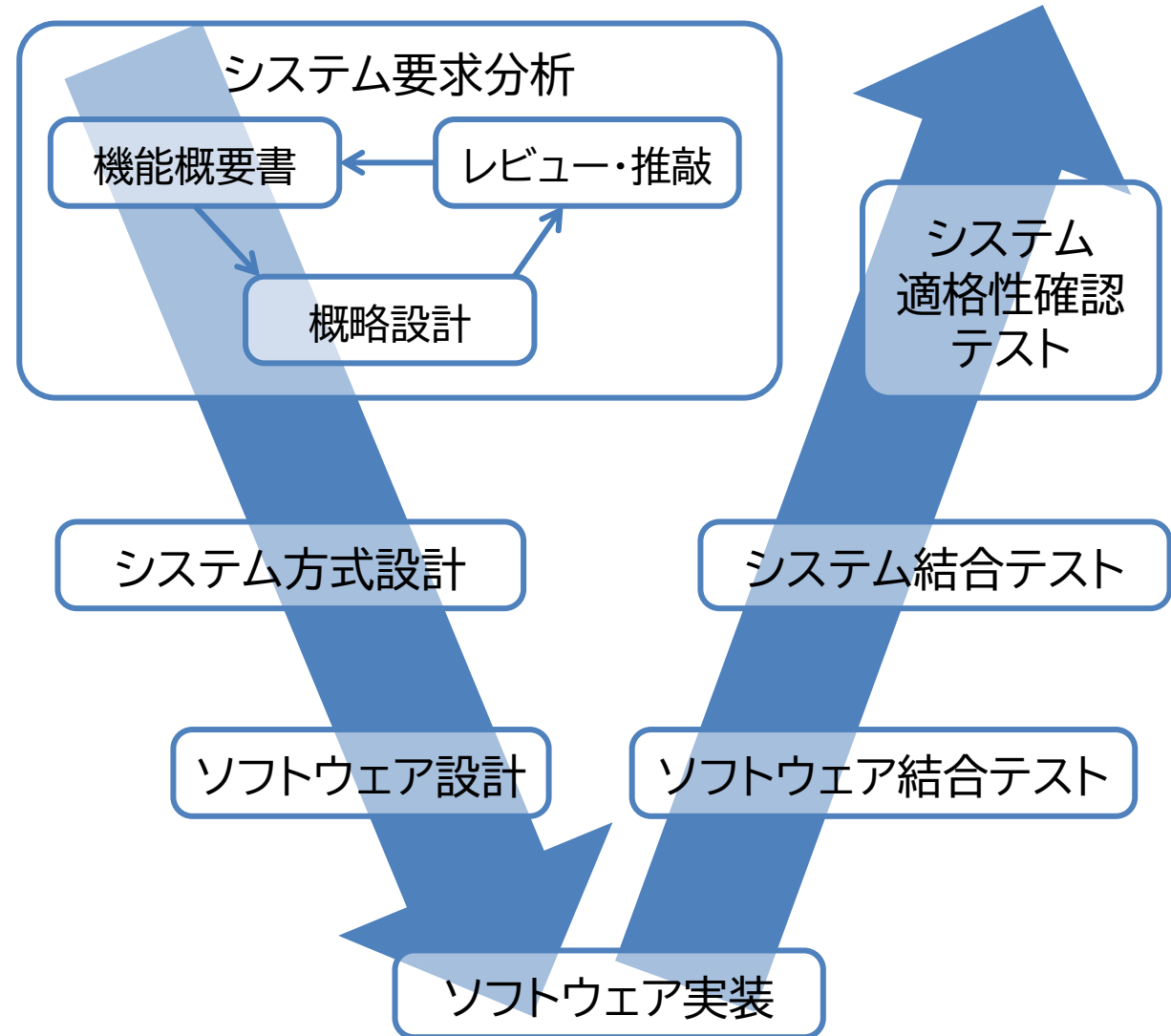
- 呑んでもらわなければならない制約や、譲歩できるところについて考えておく
 - 「この制約が受け入れられないなら破綻する」という条件を把握しておく
 - 多少譲歩した案としてプランB、できればプランCも考えておく
 - 打ち合わせの場ですぐにプランB、プランCを繰り出して合意に持ち込む準備をしておく



機能概要書を書きながら、設計を想像する

■ 「小さなV字」をぐるぐる回す

- 機能概要書を書き起こすにつれて、設計のイメージも浮かんでくる
- 設計のイメージが浮かぶと、仕様の具体的な姿も浮かび上がってくる
- その結果、「意外と影響が大きいこと」、「どうしても難しいこと」も見えてくる
- 全体としてはウォーターフォールでありながら、探索する期間を含んでいる
- 機能概要書のうちなら、試行錯誤のコストは低い



並行してプロジェクト計画も作成

■ トレーサビリティマトリックス(TM)を作る

- XDDPの方法を簡略化したもの
- 「要件」をいくつかの「サブ要件」に分けて、どのモジュールやインターフェースが担当するかをマトリックスで表す
 - 「サブ要件」が全部実現できたら要件が実現できるように分解する
 - これがシステム全体を見渡す、簡易的な設計書にもなる
- 列挙するモジュールは10個程度になるようにうまく調節する
 - デプロイ単位や、開発チームの境界なども分界点の候補

サブ要件	内容	関係モジュール/インターフェース		
		クライアント	プロトコル	本体
セッション管理	セッション開始、終了を知らせる	○	○	○
状態取得コマンド	各コンセントの占有状態、占有者などを応答するコマンドを追加する	○	○	○
占有状態表示	占有者などの情報を表示する	○	-	-
...	...	...	...	...

並行してプロジェクト計画も作成

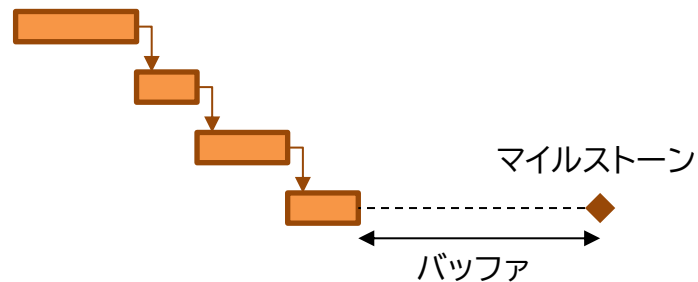
- サブ要件を見ながら工数・工期を見積もり、WBSにする
 - 最初からあまり細かく分解しすぎない
 - 工数・工期を見積もるときは「確度」の意識を揃えること
 - 例えば確度70%の見積もり、確度90%の見積もりが混在していると正しく把握できない
 - バッファの持ち方については後述
 - あくまで「今のメンバー」での見積もりであることに留意
 - 時間が経って人が入れ替わると、同じ見積もり工数で対応できるとは限らない

並行してプロジェクト計画も作成

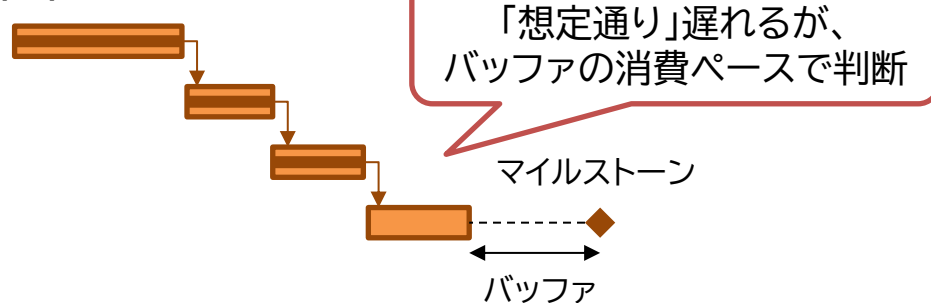
■ CCPM(クリティカルチェーン・プロジェクトマネジメント)を活用

- 個々のタスクにはバッファを積まず、確度50%に揃える
- バッファは一番最後にまとめて持つ
- 日々、最新の状況をガントチャートに反映する
 - ・ イナズマ線よりも、実際いつまでかかりそうなのかを見えるようにするのが大事

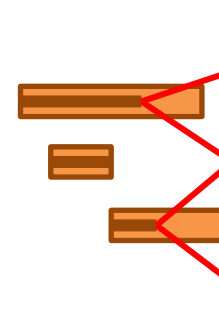
計画初期のガントチャート



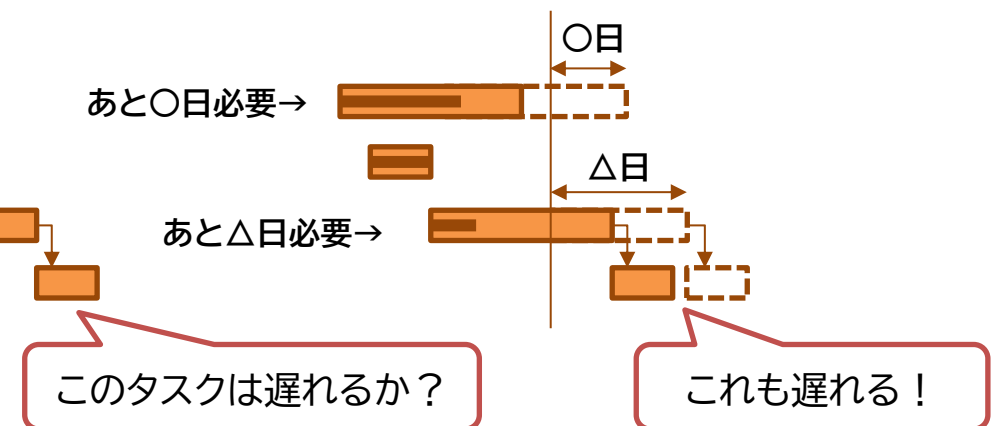
進行中のガントチャート



稲妻線管理のガントチャート



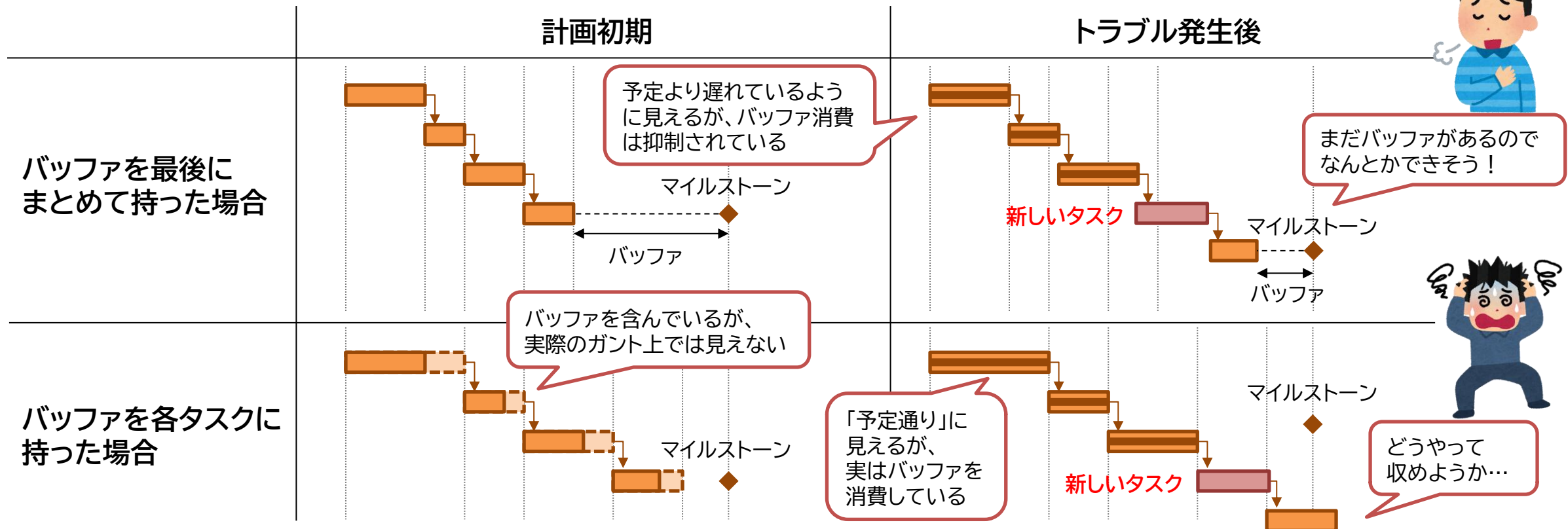
残日数で修正したガントチャート



並行してプロジェクト計画も作成

■ プロジェクトが遅れるとき、タスクは「伸びる」というより「増える」

- 経験上、遅れるときは大抵予想外の事態が起きて新しいタスクが増える
- 新しいタスクが増えた時に、バッファが各タスクに散らばっていると混乱する
- バッファが1箇所にまとまっていれば、影響が一目瞭然で手当てもしやすい



実践のポイント

まず「方針」のブレを抑え込むこと

■ 概要レベルですり合わせできなければ、先に進めるわけがない

- 機能概要書は全ての起点になるもの
- 機能概要書の合意が遅れると当然焦りを感じるが、ここをおざなりにしてしまうと後で苦しむ
- 逆に機能概要書の段階でかなり見通せていれば、後は「作るだけ」になる
- 本当に認識が合っているならレビューも一発で通るはずなので、まずは作ってみよう

■ 最初に合わせるべきところ

- 「機能の目的」について関係者の認識が合っているか？
- 「基本的仕様」の根本的なコンセプトが受け入れられるか？
- 方針が合っていない懸念がある場合は、まずこの部分だけレビューしてもらう

なぜ「要求」という言葉を使わなかったのか

- 機能概要書を通して行っていることは、一般的には「要件定義」に該当する
 - しかし管理単位こそ「要件」であるものの、「要求」という言葉をあまり使っていない
- 意図的に外したわけではないが、「要求」という言葉がしっくりこなかった
 - 「誰が」「誰に/何に」要求しているのか？
 - 直接やりとりする相手は企画部門だが、本当の要求元は別にいる
 - 「ソフトウェア/システムに対しての」要求か、「開発チームに対しての」要求か？
 - 「実現するために、結果的に求められること」は要求したという実感はない
 - 結果的に出てきた仕様がOKかどうかは気になるが、そうしたいと言ったわけではない
 - 「そんなことを要求した覚えはない」けど要求元として書かれる？
- 「要求」という言葉をやめれば「要求元」と「要求先」の対立がなくなるかも？
 - いいものを作って、ビジネスを成功させたいという目的は同じはず

なぜ開発側が作るべきなのか

- 内容的には要求に相当するならば、依頼側が作ってもいいように思えるが…
- 開発側が作ることで得られるメリット
 - そもそも実現不可能な仕様が提示されて、時間を浪費することを防げる
 - 開発者が理解した内容を書くことで、認識に齟齬があればレビューで見える
 - 今のアーキテクチャで無理なく実現できる仕様を提案できる
- 目的が達成できるなら、システムにとって自然な形で実現されるのがよい
 - システムの設計思想を無視した仕様を考えたとして、ユーザーにとってもわかりにくい
 - 早く開発ができて、仕様も分かりやすくなるならそれが望ましい

なぜ「パワポ」で作るのか

- 当時の時代背景もあるが、「パワポ」のメリットはある
- 打ち合わせの場でリアルタイムに書き込みするのに適している
 - 本文の編集、吹き出し、ノートなど、その場での書き込み手段が豊富
 - その場で意見をまとめ上げるためのツールとして非常に強力
- スライドを投影して、口頭で説明する前提で作る意味
 - 1時間の会議の前半30分で説明して、後半30分で議論できるボリュームにする
 - 収まらない場合は一部を切り出すことも検討する
 - 頭から読んでいって理解できるように構成する必要がある
 - アンチパターン: 全部読めば確かに書いてあるが、よく読まないで理解できない
 - 文字が小さすぎると読めないため、1ページの情報量が自然と制限される
 - どこでページを分けるか？ もっと簡潔に表現できないか？

機能概要書は「使い捨て」にする

- ドキュメントが増えていくと、メンテナンスコストが増大してしまう
- そこで、機能概要書は「使い捨て」の資料にする
 - 機能概要書は、機能がリリースされたときに役割を終える
 - もちろん本当に捨てるわけではなくアーカイブしておき、その機能ができた経緯を知りたいときに参照する
 - その代わり、機能概要書に書かれた内容は漏らさずに機能仕様書などの継続メンテナンスされる資料に盛り込む
- 設計書も同様に「今回の変更」に特化した内容にする
 - トレーサビリティマトリックスから導かれる、各モジュールのタスクリストも簡易的な設計書として役立つ
 - これらはXDDPの「変更要求仕様書」、「変更設計書」の考え方を取り入れている

ただし、機能がリリースされるまではメンテナンスする

■ フィードバックしないデメリット

- 機能概要書を作り始めた当初は、メンテナンス工数を嫌って機能仕様書以降の成果物に作業が移ってからは、仕様に変更があっても機能概要書まではフィードバックしなかった
- しかし、いざ説明したいときに最終仕様が載っている資料として使えずに困った

■ リリースするまではフィードバックすることで、いつでも最新の仕様を説明できる資料が手元にある状態になる

- リリース後はマニュアルが役割を引き継ぐ

機能概要書のテンプレートを成長させる

■ 機能概要書は製品の特徴に合わせてカスタマイズすべき

- 何度も機能開発を繰り返していく中で、それぞれの製品に最適な形にテンプレートを更新する
 - アンチパターン: 製品の特徴に関係なく全社で標準化
- 前回検討が漏れて手戻りが起きたところがあれば、補うようにテンプレートを更新する
 - 例えば過去バージョンで作成されたファイルとの互換性の検討を忘れていたとする
 - テンプレートに「互換性」のページを追加し、「過去バージョンで作成されたファイルを開いたとき」という項目を書いておく
 - それにより、次の機能を開発するときに互換性の観点を忘れずに検討することができる
- テンプレートが成長するほど、次回の仕様の品質が良くなっていく

生成AI時代の開発に向けて

生成AI時代の開発に対して思うこと

- 今年(2026年)6月に開催された「AI要件定義サミット」に参加しての所感
 - 要件定義、基本設計、詳細設計… と段階的に詳細化するのでは、AI駆動開発の恩恵を十分に受けることはできない
 - 組み込み開発でどこまでAIに自動で開発させられるかは未知数だが、設計的な要素も含めて「どうしたいのか」を早期に詳細化し、AIが走り出せるようにする必要がある
- 機能概要書とAI駆動開発は、本来相性がよいはず
 - ステークホルダーが関心のあるところは深く掘り下げて成果物を「縛る」
 - 勝手に決めてよいところは、あえて書かない
 - このような性質は、AI駆動開発と本来相性がよいと考えられる

生成AI時代の開発に対して思うこと

- すぐに動くソフトウェアが作れるようになったが…
 - 一見アジャイル開発に近づいたように思えるが、そこには少し疑問がある
 - 「使う人」あるいは「作りたい人」が自分で作る分には問題ないが、人に使ってもらうものを作るときにはどうだろうか？
 - 早く作れるようになったとしても、ユーザーが望むものが作れているのだろうか？
- 「品質は誰かにとっての価値である」(G.M.ワインバーグ)
 - 「品質は相対的である。ある人にとっての良い品質は、別の人にとっては悪い品質であるかもしれない。」
 - 「わたしたちは品質が合理的に設定できると思い込みたがるが、それはほとんどつねに政治的/感情的な問題である。」
- 早く作れるがゆえに、むしろユーザーを置き去りにしていないか？
 - 価値を受け取る、つまり良し悪しを判断するにも実は時間がかかる

「アジャイルソフトウェア開発宣言」に立ち返ってみる

みんなツールや
ドキュメントの話を
している…

私たちは、ソフトウェア開発の実践
あるいは実践を手助けをする活動を通じて、
よりよい開発方法を見つけだそうとしている。
この活動を通して、私たちは以下の価値に至った。

「個人」の価値、
「対話」の価値は
どうなった？

プロセスやツールよりも個人と対話を、
包括的なドキュメントよりも動くソフトウェアを、
契約交渉よりも顧客との協調を、
計画に従うことよりも変化への対応を、
価値とする。すなわち、左記のことがらに価値があることを
認めながらも、私たちは右記のことがらにより価値をおく。

Kent Beck
Mike Beedle
Arie van Bennekum
Alistair Cockburn
Ward Cunningham
Martin Fowler

James Grenning
Jim Highsmith
Andrew Hunt
Ron Jeffries
Jon Kern
Brian Marick

Robert C. Martin
Steve Mellor
Ken Schwaber
Jeff Sutherland
Dave Thomas

© 2001, 上記の著者たち

この宣言は、この注意書きも含めた形で全文を含めることを条件に自由にコピーしてよい。

機能概要書は「人との対話」を促すツール

■ 機能概要書のレビューは思いをぶつけあう場

- 開発チームとしては要望を尊重しつつも、今の設計思想にフィットする形の提案をする
- しかし、ステークホルダーとしては受け入れにくい場合もあるはず
- 機能の目的を達成し、それによってビジネスゴールを達成するために、対話を通じて落としどころを見つけていく
- それは、設計思想を進化させることでもある

■ 資料のすみずみに血を通わせることで、信頼関係を作る

- 生成された資料をただ読み上げるだけでは、ステークホルダーとの信頼は生まれない
- 機能概要書の作成は単なる作業ではなく、「この仕様であるべきだ」という確信を高める工程でもある
- 真剣に考えた内容だからこそ、耳を貸してもらえることもある

生成AIの助けを借りてもよいが、自分の思いをしっかり持つ

生成AIの使いどころ①:レビュー前のブラッシュアップ

- 機能概要書のレビューを受ける前に、突っ込まれそうなポイントを探してもらう
 - 自分一人で書いていると、書き忘れていることや暗黙の前提に気づかない
 - 生成AIに第三者として見てもらうことで、自分の思い込みや検討が甘いところに気づくことができる
- あくまで仮想のステークホルダーにすぎないことに注意
 - AIに言われるがままに直しても、それが受け入れられるとは限らない
 - まったく違う方向性の指摘を受けることもありうる
 - あくまで参考に留めること

生成AIの使いどころ②:実現性・リスクの調査

- 機能概要書を書く上で、大きなリスクになるのが実現性の確保
 - こちらから提案した仕様が実現できなければ、プロジェクトは遅れ、信用も失う
- 生成AIに現行のソースコードや設計資料を読み込ませることで、実現性の判断がしやすくなる
 - 今までは製品を知り尽くした設計者でなければ、できるかできないかをすぐに判断することが難しかった
 - 知り尽くした人でも、できると思ったがうまくいかないということがある
 - そこで生成AIを使うことで、まだ習熟していない製品についても実現性の判断が容易になる
- 課題:複数のソフト・ハードが絡むシステムでどこまで判断できるか？
 - ソースコード以外にシステムモデリングの成果物があれば、AIに正確な解釈をさせられるか？
 - 今後検証していきたいテーマ

「仕様駆動開発」への接続

- 機能概要書の内容は、「仕様駆動開発」のインプットとしても使える
 - 機能概要書の内容をMarkdown化すれば、Spec Kitのインプットとして渡すことも可能
 - 仕様駆動開発のために、わざわざ書き直す必要はない
- 課題:PPTXファイルをMarkdownにうまく変換できるか？
 - オープンソースではPandoc, pptx2mdなどの選択肢があるが、本文の変換は概ねうまくできるものの、図形描画機能で作成した図の変換はうまくいかない
 - Microsoft 365 Copilotで図形を画像に変換するように頼むと、ある程度うまく行く
 - 内部で独自Skillを使っていそうなので、おそらくOfficeのAPIを駆使すればできるはず
- 課題:仕様を詳細化する過程で出てきた内容の、機能概要書へのフィードバック
 - Specifyした結果のうち、ステークホルダーと合意すべき内容が出てくればフィードバックが必要だが、フィードバックすべき内容のピックアップもAIの助けを借りたい

まとめ

まとめ

■ 機能概要書を作るとうれしいこと

- 忙しいステークホルダーに、限られた打ち合わせの時間でOKをもらいやすくなる
- マニュアルの元ネタ、品証への説明などいろいろな役に立つ
- 共通の目的を起点に展開することで、関係者を同じ方向を向く仲間にする
- 小さいV字を回して、早期に問題発見・解決できる

■ 生成AI時代も、機能概要書の思想は役立つ

- 「AI駆動開発」や「仕様駆動開発」との相性も悪くない
- 生成AI時代は、「やりたいこと」をどれだけ解像度高くできるかがポイント
- 機能概要書というスタイルは、その実現にきっと役立つはず

最後に少し宣伝

- 今回ご紹介したのは、当社メンバーが支援する組み込みソフトウェア開発の現場で、現場の課題に応じて解決策をご提案し、実践した一例です。
- オージス総研では開発現場の様々な課題に対して、理想論だけではない、組織の実情に応じた解決策をご提案します。
- また、「実技つきコンサル」として当社メンバーが開発チームに入って自ら実践し、定着までサポートすることも可能です。

組み込みシステム・ソフトウェア開発のお困りごとがあれば、
ぜひ一度、お気軽にお問合せください。

当社エンジニア/コンサルタントが直接ヒアリングさせていただきます。
(お問合せの際に「SWESTの資料を見た」とお伝えいただくと助かります)

ご清聴ありがとうございました

参考文献

- (一社)情報サービス産業協会 REBOK企画WG 編. 要求工学実践ガイド. 近代科学社, 2014, 244p. (REBOKシリーズ, 2)
- Jonathan Rasmusson. アジャイルサムライ——達人開発者への道. 西村直人, 角谷信太郎監訳, 近藤修平, 角掛拓未訳, オーム社, 2011, 336p.
- 清水吉男. 【改訂第2版】[入門+実践]要求を仕様化する技術・表現する技術～仕様が書けていますか?. 技術評論社, 2010, 382p.
- 清水吉男. 「派生開発」を成功させるプロセス改善の技術と極意. 技術評論社, 2009, 416p.
- Robert C.Martin. Clean Architecture——達人に学ぶソフトウェアの構造と設計. 角征典, 高木正弘訳, KADOKAWA, 2018, 352p.
- G.M.ワインバーグ. ワインバーグのシステム思考法. 大野尙郎監訳, 共立出版, 1994, 340p. (ソフトウェア文化を創る, 1)