筑波大学
University of Tsukuba

JEITA 組込み系ソフトウェア・ワークショップ 2014
組込み系開発の実践的モデリング

組込ソフトの開発現場におけるモデリング事例

～モデル化と反復プロセスを組織的に推進する～

久我 雅人

筑波大学 University of Tsukuba

私とモデル化技術のかかわり

- T社時代
  - 前期：設計図等の技術を知らなかった時代
  - 後期：設計図面を使い始めた時代
- R社時代
  - モデル化技術によって高生産性を発揮した時代
- D社時代
  - 異質なモデル化技術に遭遇した時代
- そして現在
  - 再び、モデル化技術で品質確保する

T社時代の開発のやり方（前期）
★T社初の本格的MFPのコントローラ・ソフト開発

筑波大学 University of Tsukuba

- 手法・・・といっても、特にない。
  - 専任のアーキテクトが決めたアーキテクチャにしたがって設計していました。
  - そのころは、まだ「いかに機能を盛り込むか」が重点課題でした。
  - 品質についての意識は低かったと思います。
  - はじめは要求分析の責任者、途中からUIアプリケーションの設計リーダーを兼ねました。
- 苦労した点
  - 品質をどのように確保してよいかわからず、作りこみレベルが不十分なまま総合試験を始めてしまったり・・・
  - しかも私の管理が甘く、基本設計から詳細設計へうつる段階で、内容をキチンとレビューできていなかったり・・・
  - そのため、UIだけで500件以上のバグを抱えていました。

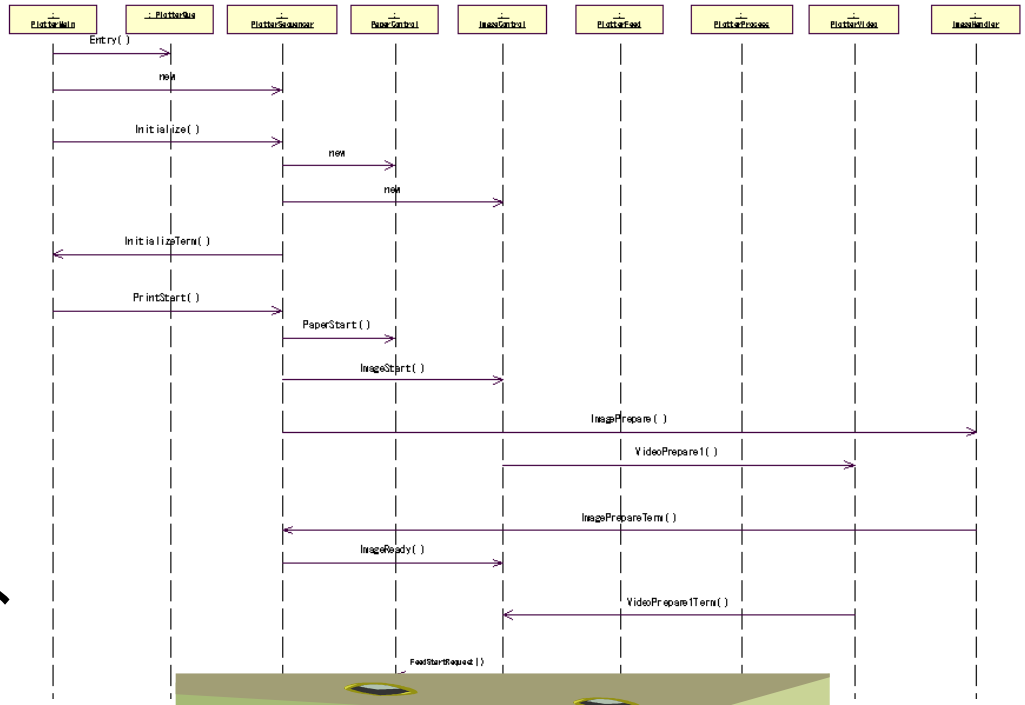
T社時代の開発のやり方（後期）
★T社初の本格的MFPのコントローラ・ソフト開発

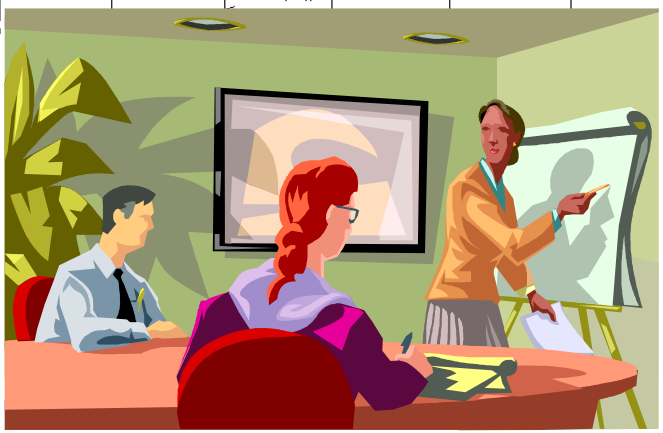
筑波大学 University of Tsukuba

- 何が問題だったか？
  - アーキテクチャを意識していない
    - アーキテクチャとソースコードが結びついていない
  - 問題が起きると、何が問題か？をつきとめるまでが大変
    - つきとめるための道具も持っていなかった
    - その結果、必ず担当者に頼り、担当者が手一杯だと人を投入して解決しようとしていた
  - アーキテクチャに問題があるのでは？と気づいたときが転機だった
    - 機能開発＋バグ対策と並行してアーキテクチャの再構築を進めた
    - 一気に不具合が収束した
- アーキテクチャ（とそれを表現すること）の重要性を知った！

ソフトウェア設計図との出会い

- いまにして思えば・・・
  - タスク間のメッセージのやりとりと、各タスクのメッセージ受信時の処理を図にすることが契機に
  - 毎朝、ホワイトボードでチームメンバーが集まり、タスク間のやりとり（＝設計の動的側面）を図にしながら議論・整合
  - モデル化へのいわば≪胎動≫の時期？？





筑波大学 University of Tsukuba

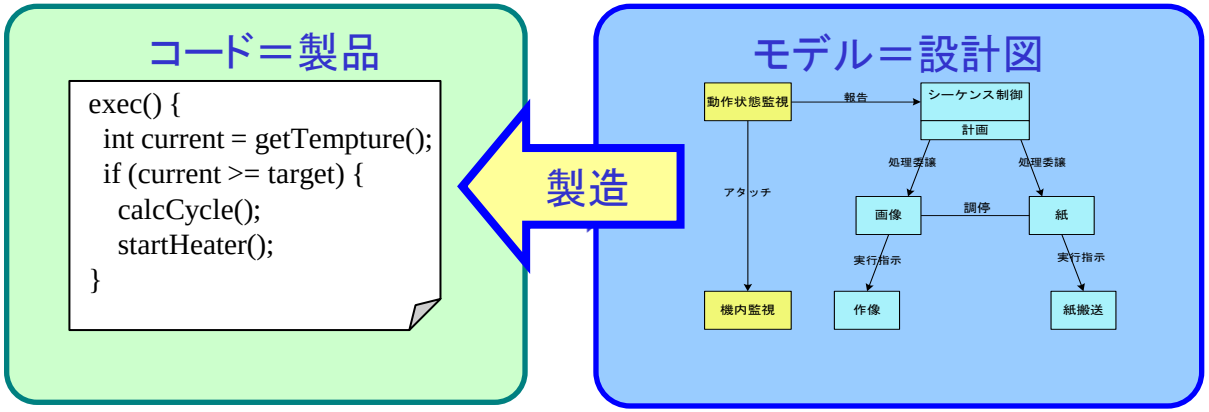
R社でモデル化技術と出会う

★オブジェクト指向技術の本格導入

- オブジェクト指向開発手法をとりいれたモデル開発
  - 抱えていた問題点
    - 仕様変更による2次障害仕様変更の工数を低減したい
    - ソースコードの再利用ではむしろ悪くなる品質を救いたい
  - この課題解決のため、当時の上司が提案
    - でも、ほんとうは上司がとにかく新しいことをやってみたかっただけなんじゃ…？
- 自分の経験から改善を意図していたことは？
  - しっかりした構造（アーキテクチャ）を作ること
  - 管理者の目に品質が見えるようにすること
  - 設計仕様書が目に見える形になればよい
  - ・・・というよりも、理解したことを形に表すことが成功につながった・・・そして、「議論する」土壌を培った

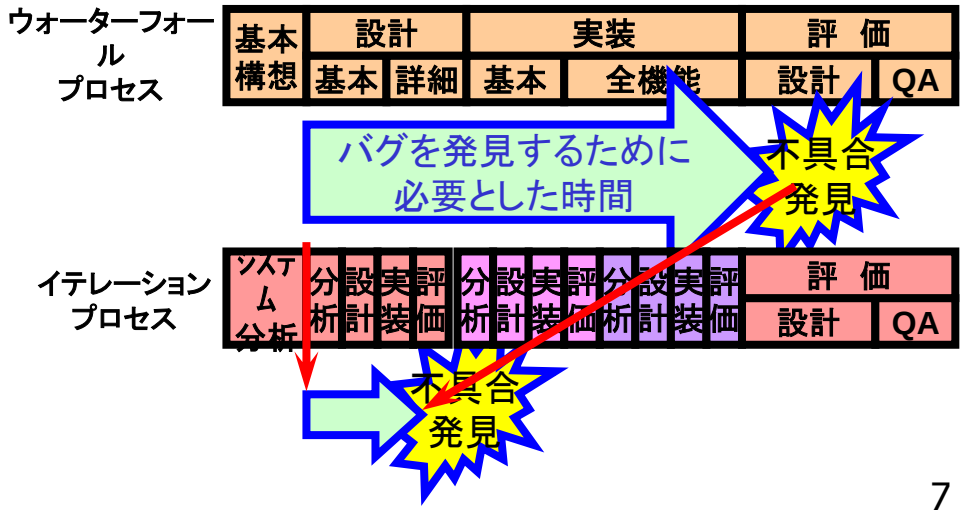
モデル化と反復開発の導入

オブジェクト指向ベースのモデリング ＋反復開発（イテレーション開発プロセス）



- 問題点を早期に発見できる＝リスク管理が容易

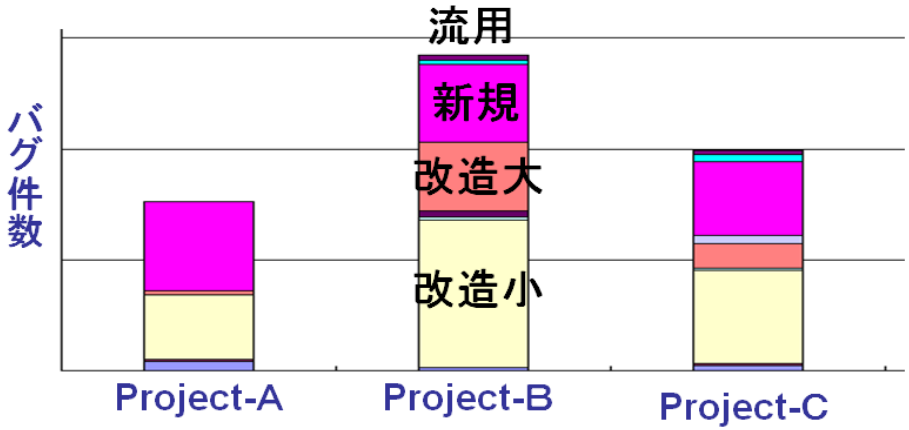
- 設計図（モデル）とコード（製品）が一致
  - 上流での品質確保がやりやすくなる
- ソースコード中心の開発（＝職人による手作り）から、モデル中心の開発（＝誰でも設計図から作れる）へ



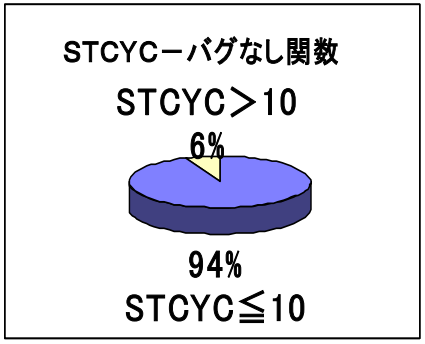
モデル化技術導入の背景

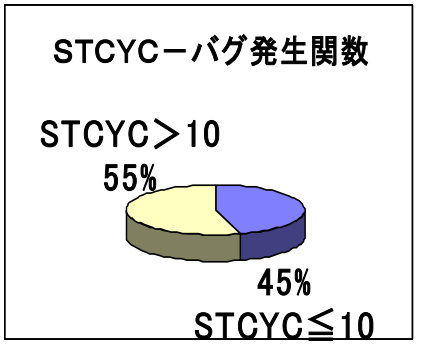
- 抱えていた問題点
  - 仕様変更による2次障害仕様変更の工数
    - 担当者を変更できない
  - ソースコードの再利用ではむしろ品質が悪くなる
    - 手軽で簡単だが全体を知らないと副作用を発生しやすい
    - 技術が開発者を通じてしか伝わらない
      - ＝属人性が高い

- ソースコードの流用では再利用を繰り返すほど品質が低下する

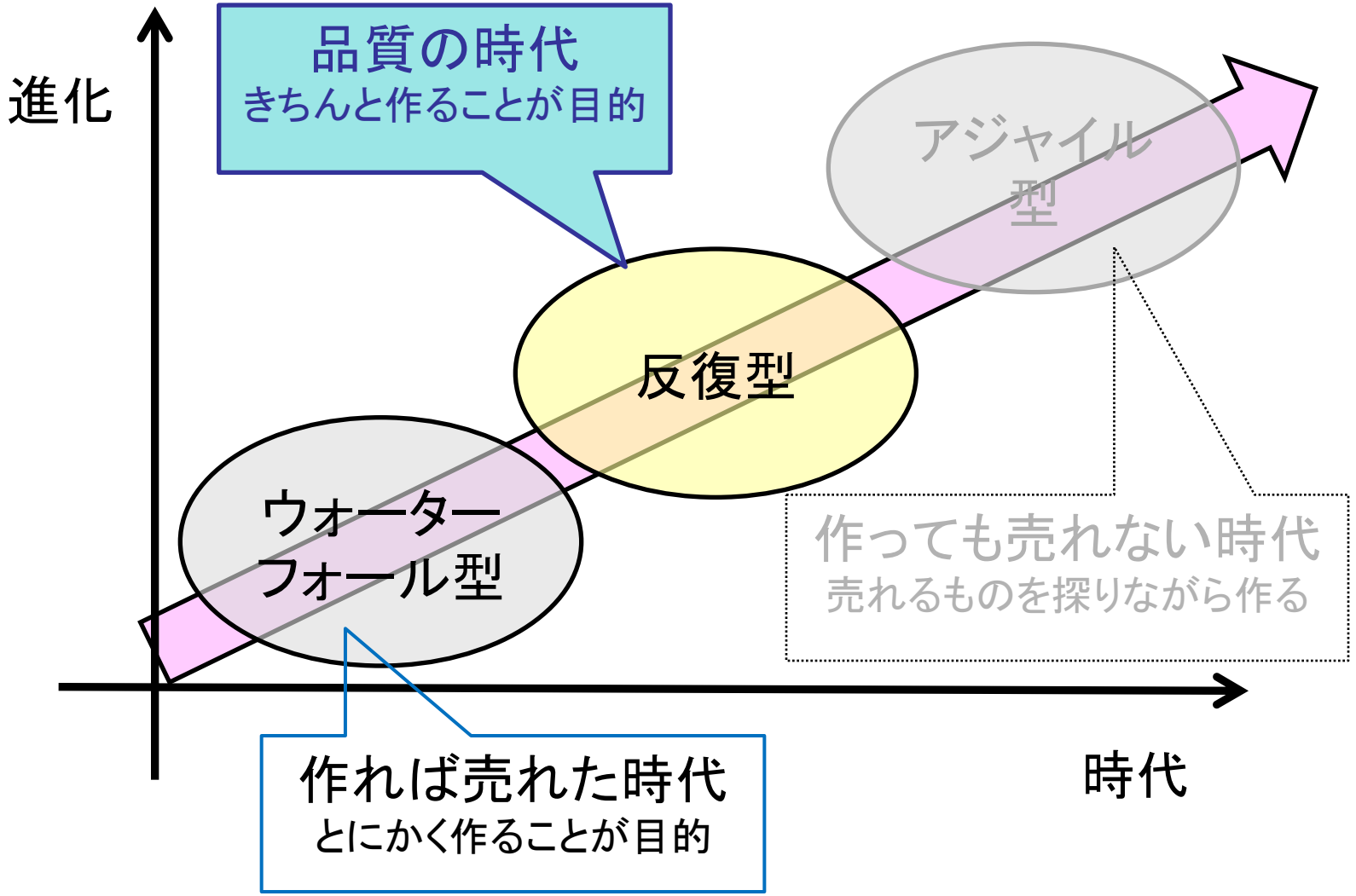


- ソフトウェアの複雑度が増加すると不具合が増加する傾向がある





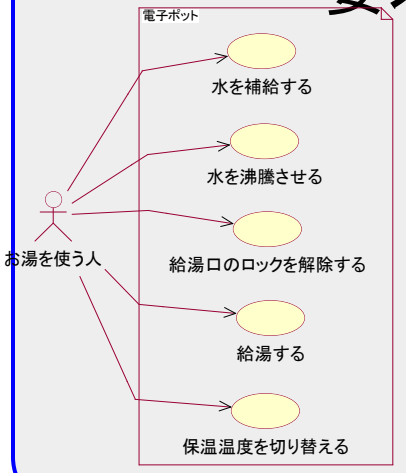
反復開発が時代の要請だった



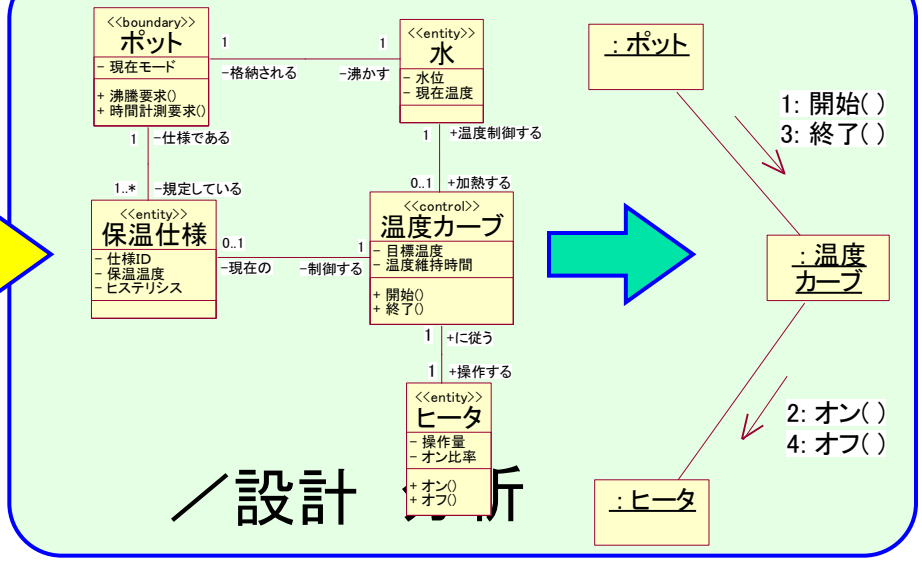
モデリングの工程

筑波大学 University of Tsukuba

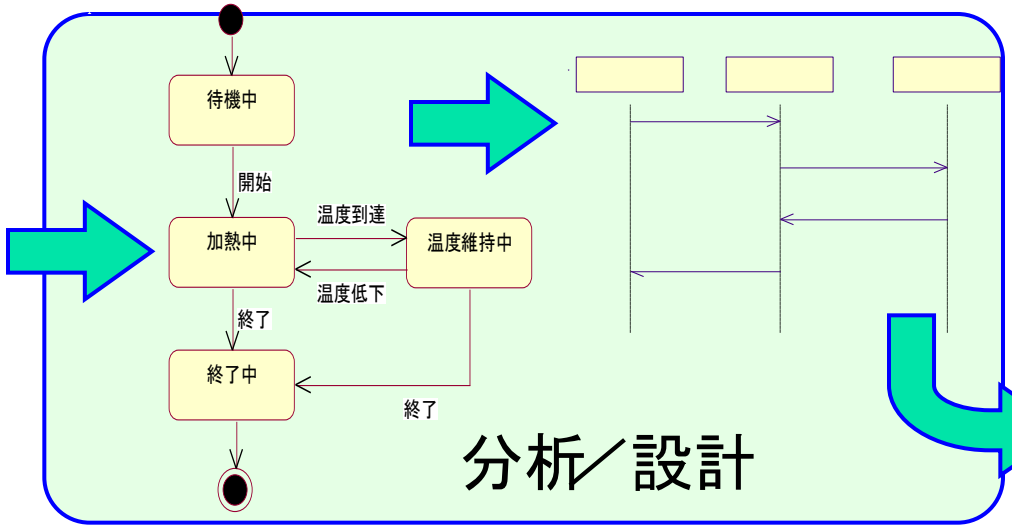
要求分析



ユースケース記述



／設計

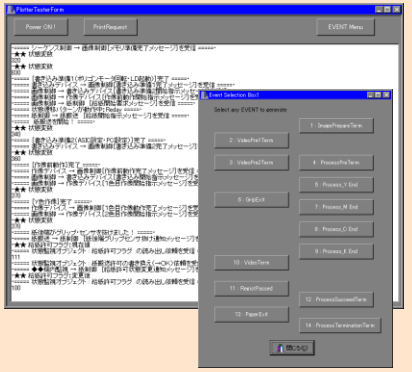


分析／設計

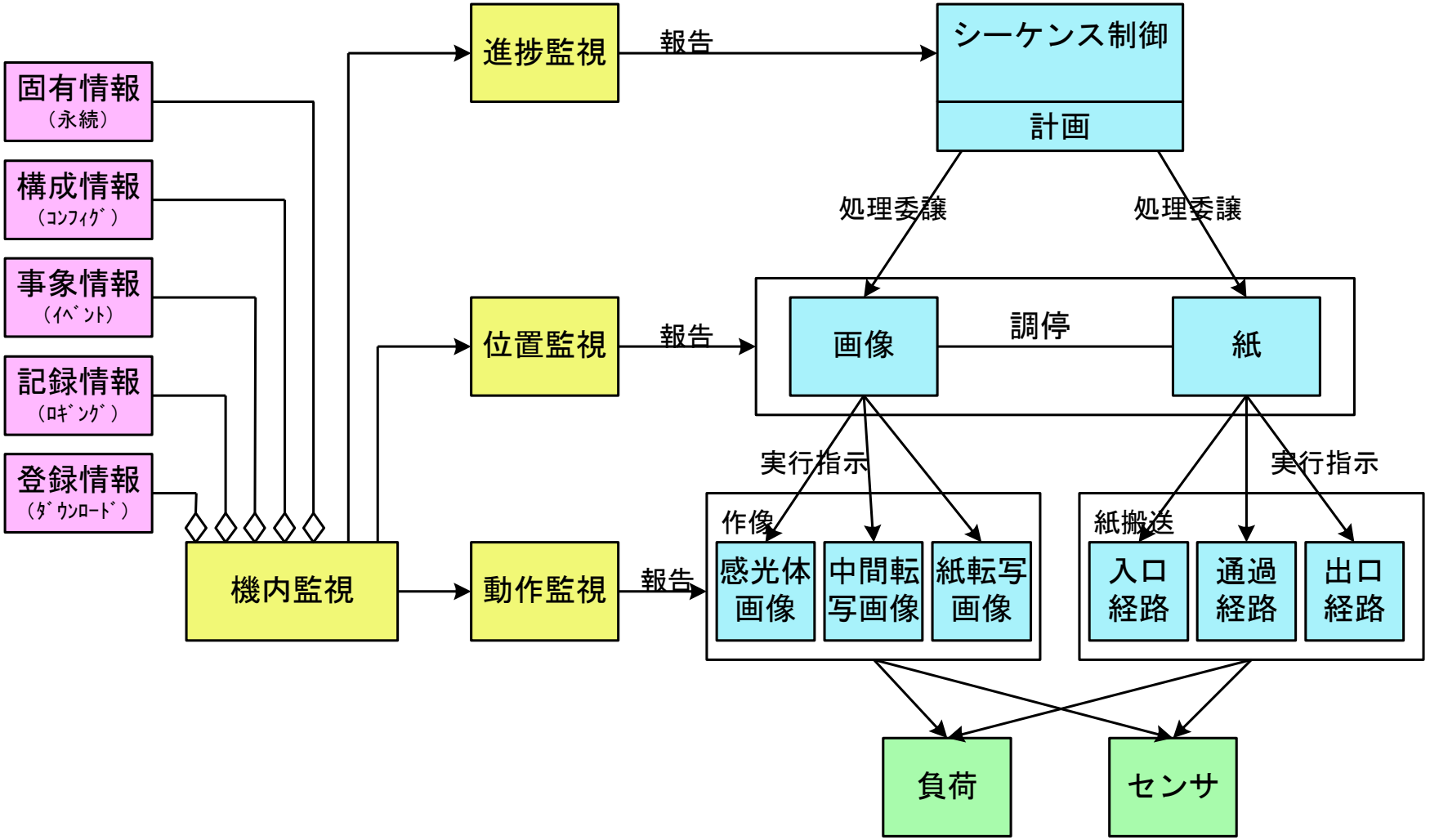
実装

ソースコード

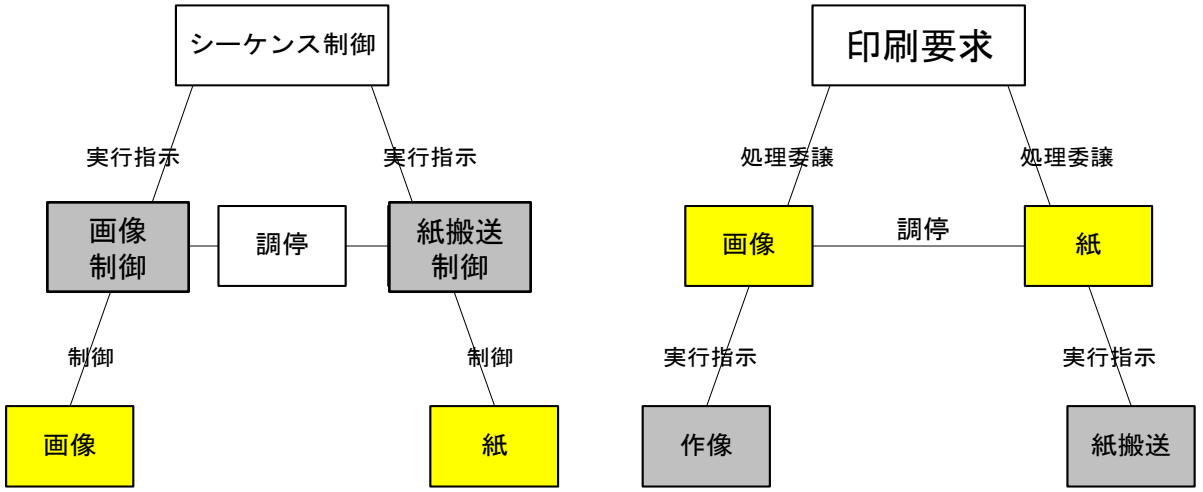
テスト



【事例1】カラーMFPの制御ソフトの全体像（初期の分析モデル）



オブジェクト指向によるモデル化で どこが改善されたか（1）

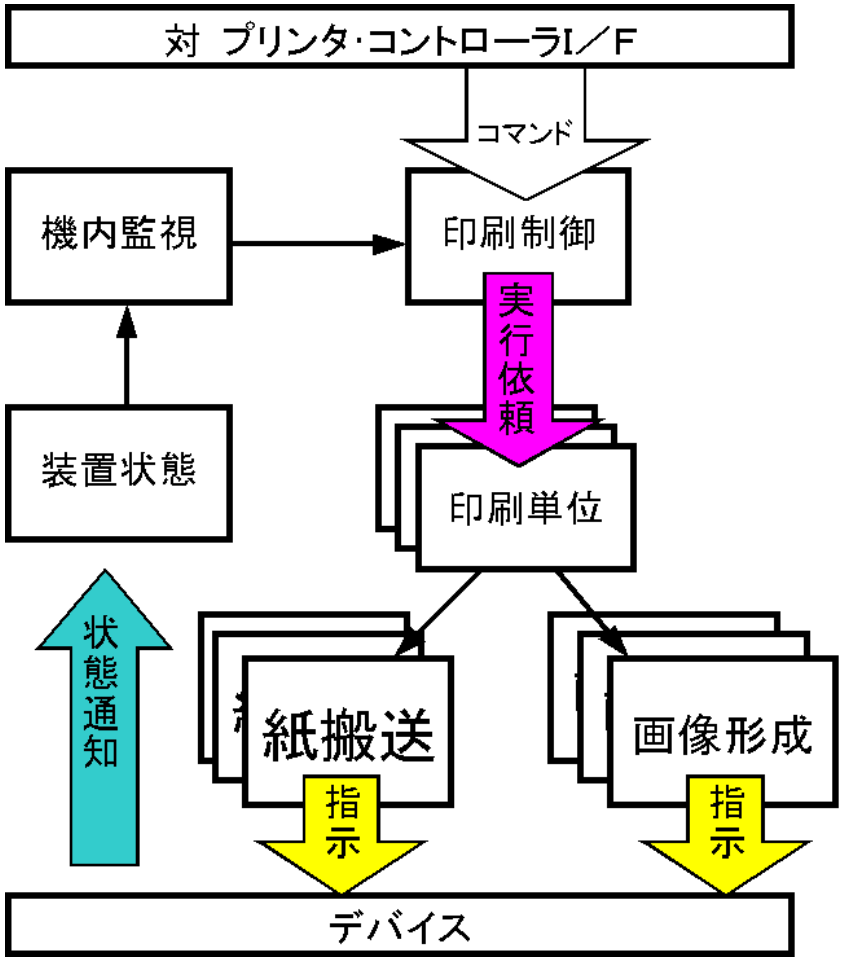


従来手法によるソフトウェアを図式化したもの　　オブジェクト指向アプローチによるモデル開発

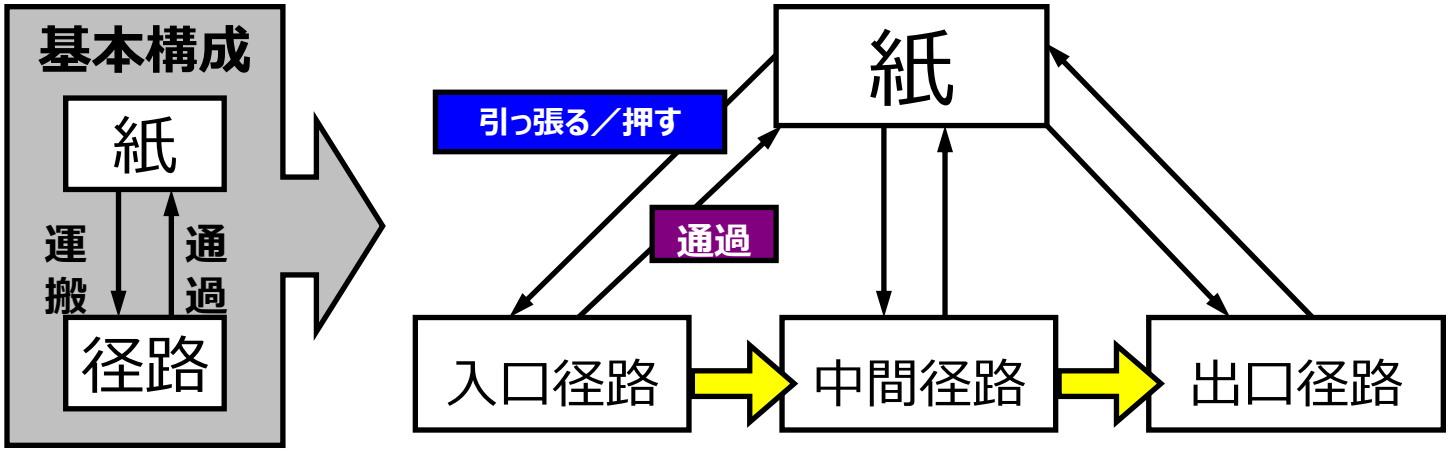
- 左側のような形従来モデルでは，「シーケンス制御」で，「作像」と「紙搬送」へ順次実行処理を発行し，その二つのタイミングを調停するような存在も必要になるので，複雑な構成となる．
- また，このモデルでは，「画像」や「紙」は実行処理の結果として動作するだけなので，モデルとしての意味（責務）は持ち合わせないものとなる．つまり，「画像」や「紙」は制御されるだけの存在となる．
- 従来モデルの問題点は，制御タイミングの処理部分がモデルの中心部分である「作像」と「紙搬送」部分に実装されていることである．すなわち，この部分が変動部となり，変更による修正範囲が広くなってしまうため，品質が劣化するだけでなく，異なる機種への展開が困難になる．つまり再利用性が低い．

オブジェクト指向によるモデル化で どこが改善されたか（2）

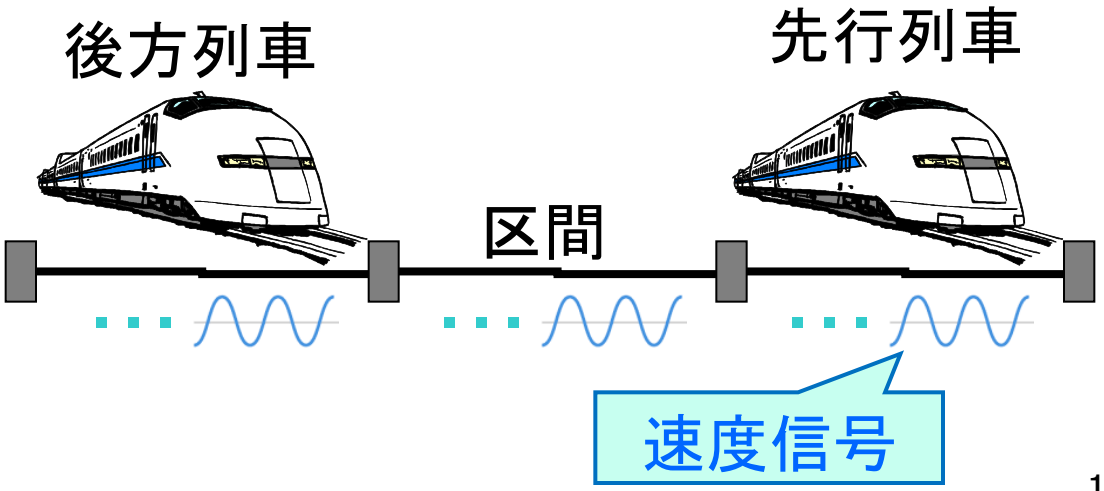
- これに対し，オブジェクト指向的アプローチで作成したモデル（前ページの右側）は，「印刷要求」が「画像」と「紙」に処理を委譲する構造となり，「画像」と「紙」は自分自身の状態を遷移させるために，作像や紙搬送へ実行指示を発行する．「作像」や「紙搬送」は，指示された動作を単純に実行するだけ，という役割分担となる．
- すなわち，前頁右側のモデルは，まずモデルの中心部分が抽象的な概念同士が連携して動作する，という汎用性の高い構造になっており，ちょっとしたタイミングの変更は，「作像」や「紙搬送」というモデルの下位層（知的情報を持ち合わせない）で対応できるため，局所的な修正で済む，という大きな特長がある．これにより，複数の機種にも適用可能なモデル化ができる．つまり再利用性が高い．



【事例2】カラーMFPの紙搬送制御部分のモデル（初期の分析モデル）

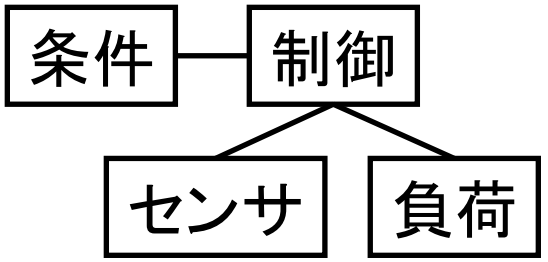


- 概念上の「紙」は「論理的な経路」を運ばれていく過程で自身の状態を変化させる
- 概念上の「紙」は「論理的な経路」１つに対し１枚しか入らない
- １区間に必ず１編成しか入れない新幹線のATCに似ている



モデリングは誤解されている

- モデル化でよくある誤解
  - 一般化しようとする
    - 一般化は、モデル化に結果見えてくるもの
      - モデリング時に一般化しようとすると、ドメインの特徴に関係なくどれも同じモデルになる



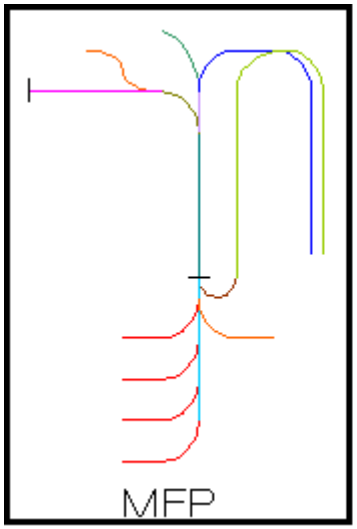
  - 抽象化の誤解
    - 抽象的な表現でまとめようとする
    - 組込では「制御対象」に着目し、その様子をモデル化するとうまく行く
    - ドメインの特徴を素直にモデリングしたほうがよい

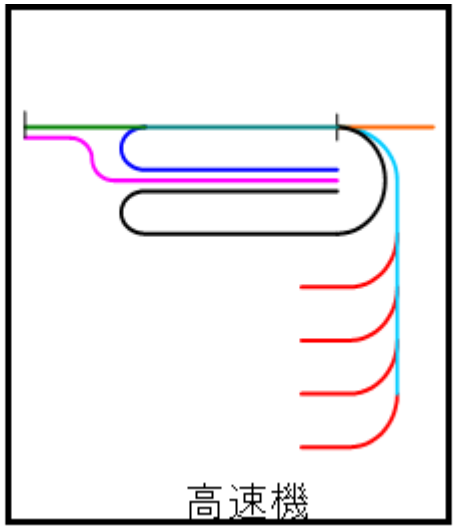
【事例3】カラーMFPの紙搬送制御部分の固定変動設計とソフト部品化

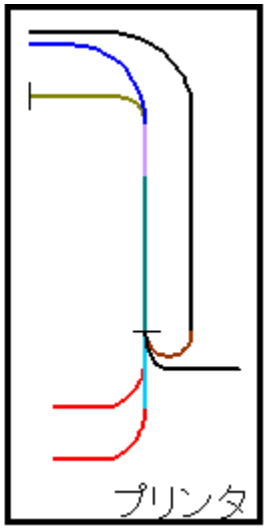
筑波大学 University of Tsukuba

- メカの構成、組み合わせ、寸法などのレイアウトは千差万別
  - メカ構成をそのままモデリングするのはNG
  - 物理構造ではなく、論理構造でとらえるべき

- メカ構成の影響を受けるもの
  - 紙詰まり検知の間隔
  - センサの数や位置
  - 両面／後処理加工の種類や機構







- これらは全て変動部か？
  - やることは毎機種同じ
  - やり方も経験上同じにできる
- → ならばルール化しよう！
  ・・・固定部としてモデリングする

この部分は当日会場でお見せします

紙間制御方式をルール化し、構造的に捉える。

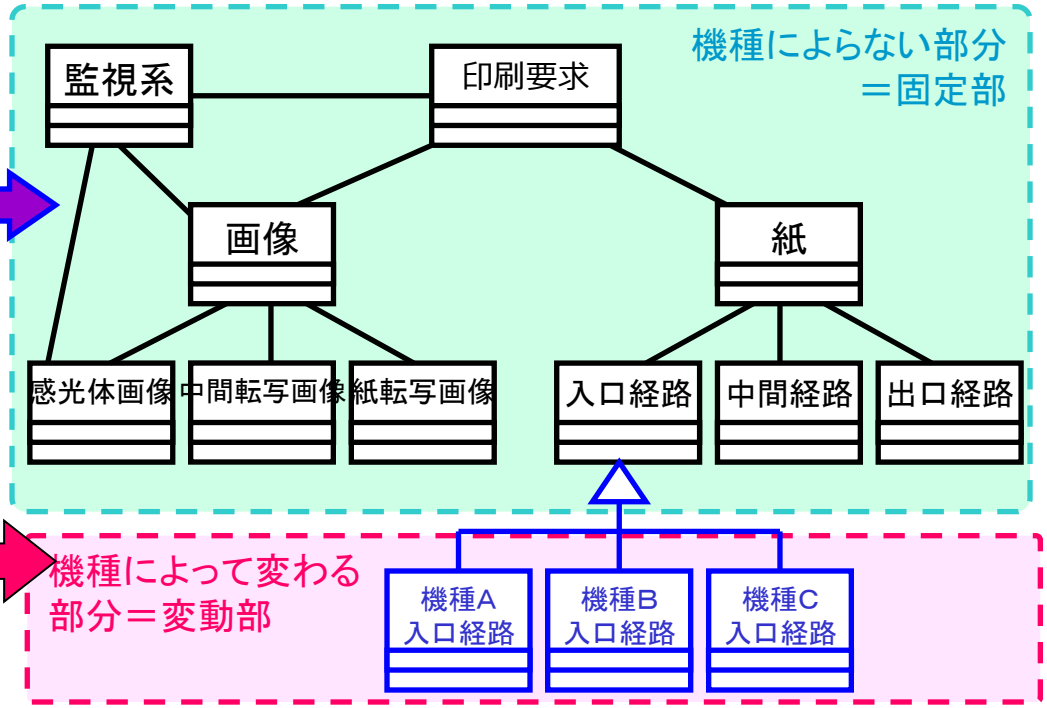
【事例4】固定変動技術を利用した大規模再利用の基本的な考え方

まず系列部分（「基本系列」「代替系列」「例外系列」）のみでモデル化

UC名	消耗品切れを判定する
概　要	印刷ごとに、××センサーと、補給部の固有情報から、消耗品切れを判定する。
アクター	1. センサー1 2. 操作部 3. 作像部
事前条件	・印刷動作が正常に行われていること
事後条件	・消耗品切れ判定が終了していること
基本系列	1. ***** 2. *****
代替系列	
例外系列	
備　考	

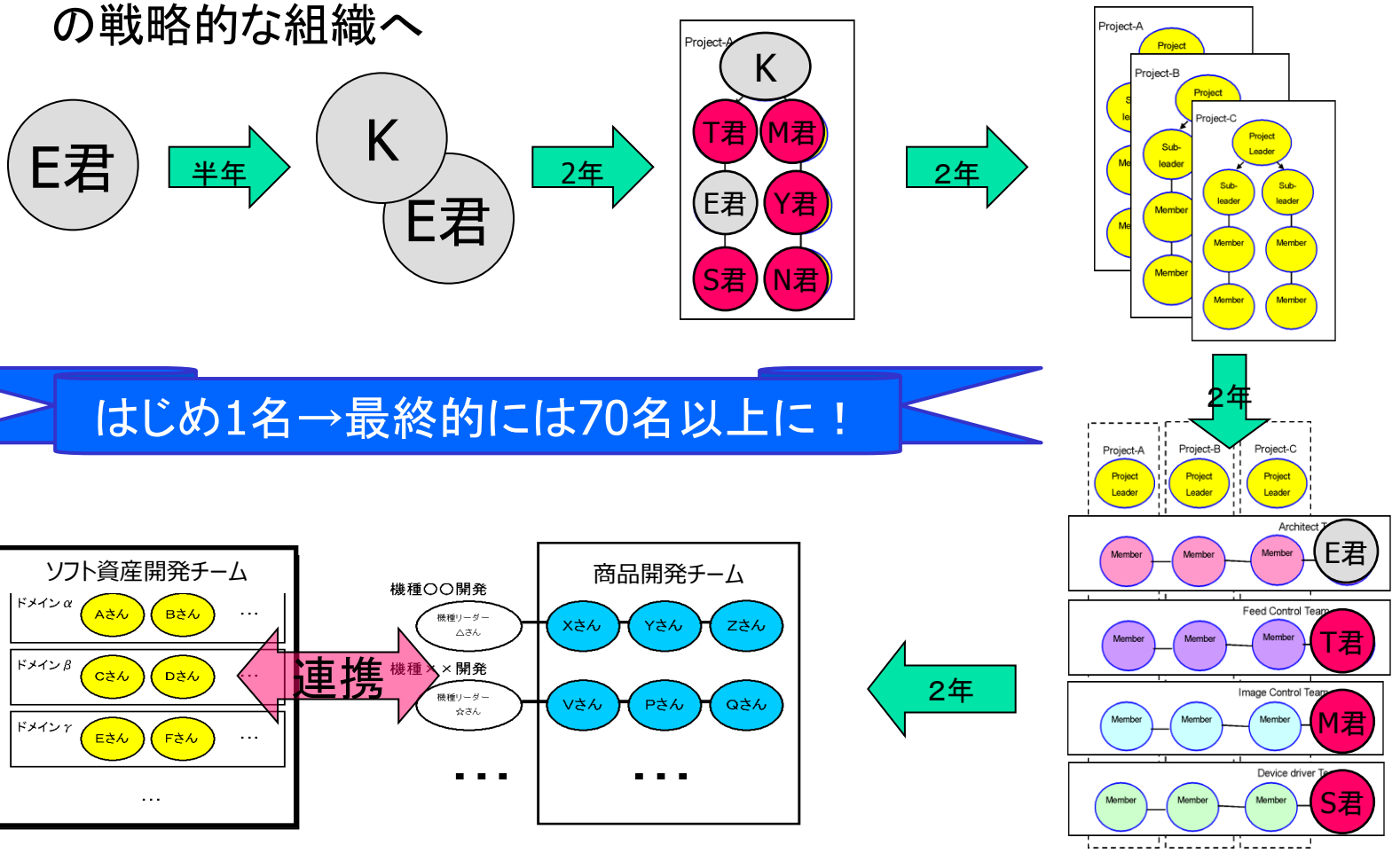
つづいて備考部分を加えてモデル化

- ユースケースをもとにまず固定部をモデル化し、
- 備考部分もモデル化して機種変動部とする

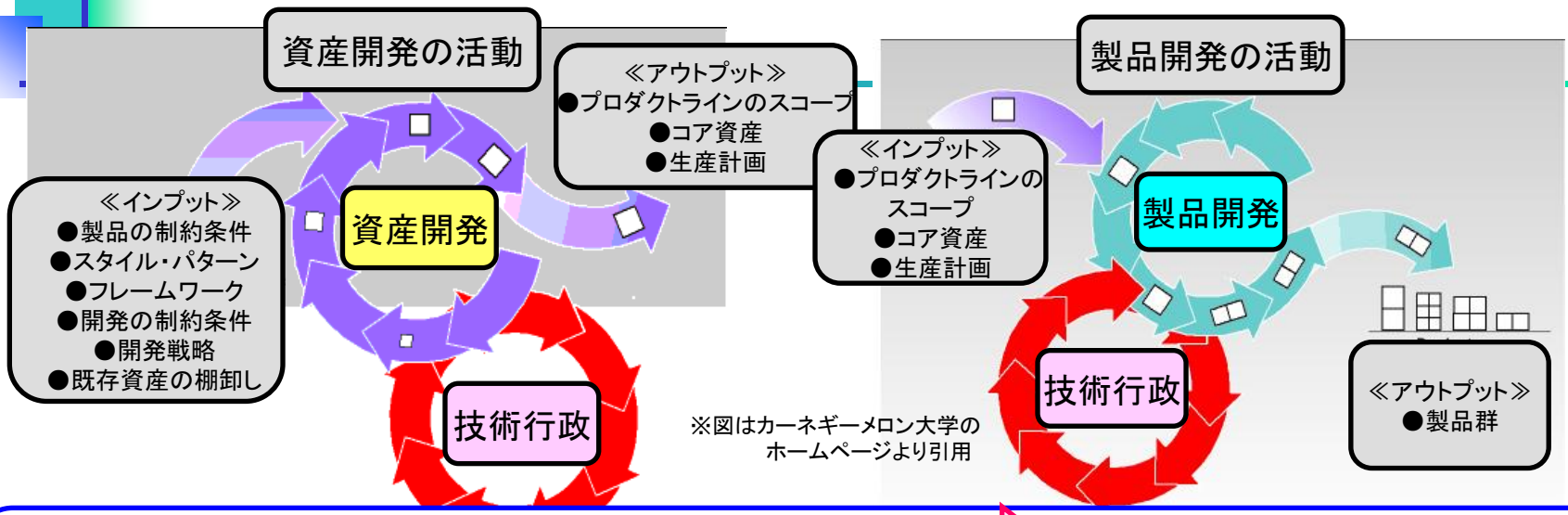


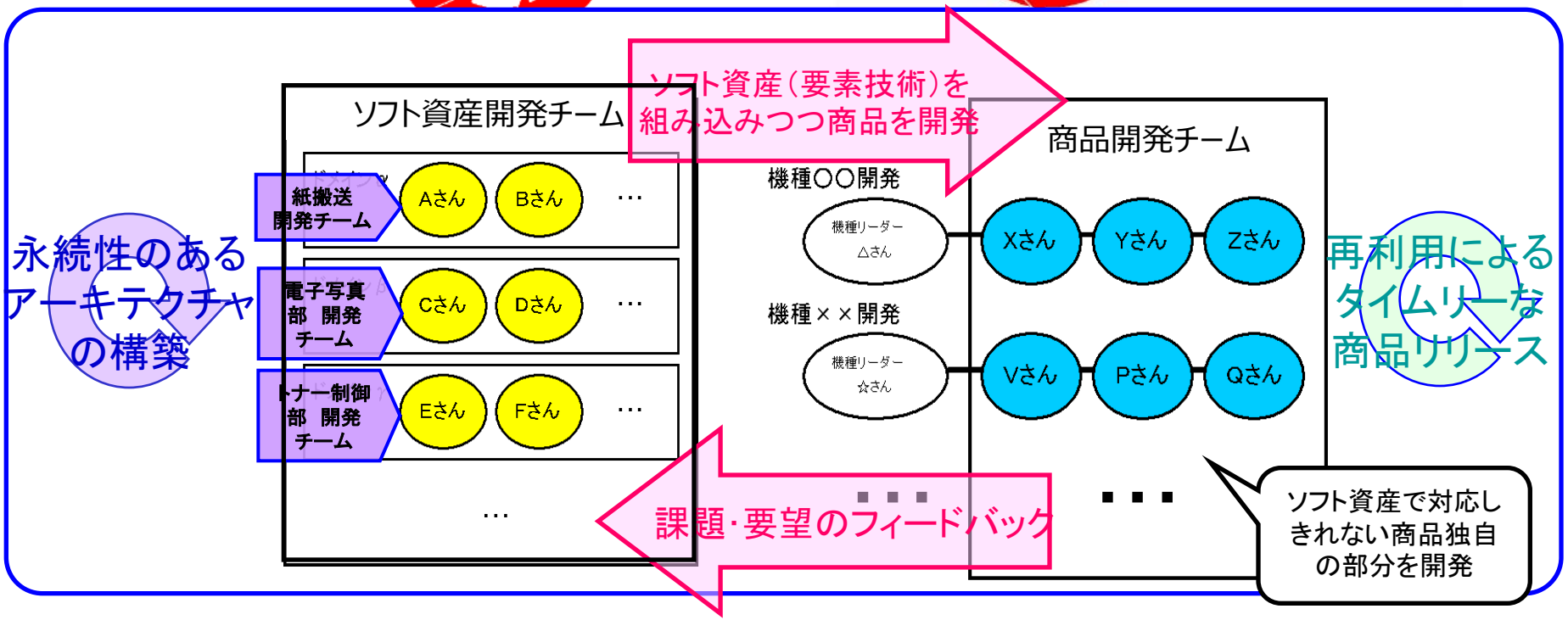
モデル化のレベルと開発者の組織化

- フレームワークの開発を進めるため、機種開発中心のいわゆる縦割り組織から機能中心の横割り組織へ・・・最終的にはSPLEを実行するための戦略的な組織へ



【事例6】ソフトウェアプロダクトライン開発の導入





モデル化によるチームの変化

- モデル化によって、設計図面を中心に議論ができる
  - モデル化によって何が変わったか？
    - 誰でも設計内容を理解できるようになった
    - 誰でも議論に参加できるようになった
- チームのレベルが向上した！
  - 個人のスキルに頼らず、チームとしての生産性に自然と目を向けるようになった
  - 高度な平凡性を有する質の高いチームになった

スポーツに例えれば、選手が監督の指示を待ってプレイするようなことが減り・・・いわば選手一人一人がそれぞれの戦術眼を有し、ある程度自分で判断して次のプレイを選択して実行するようになった

モデル化･･･新たな挑戦

工作機械メーカーのソフト開発部門にて モデル化を導入

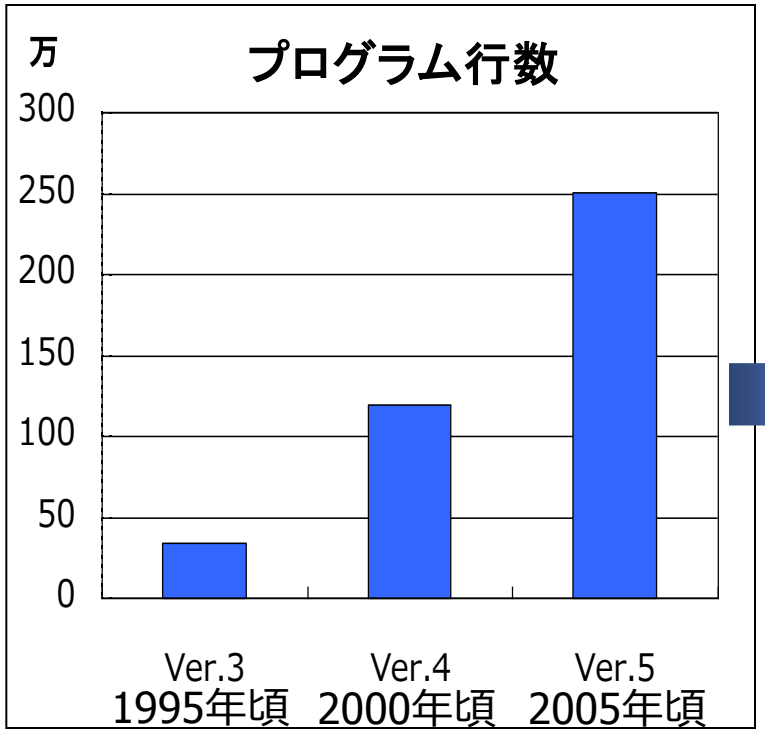
- つねに納期遅れ
- つねに低品質
  - 何か技術や手法を導入しても効果が出ない

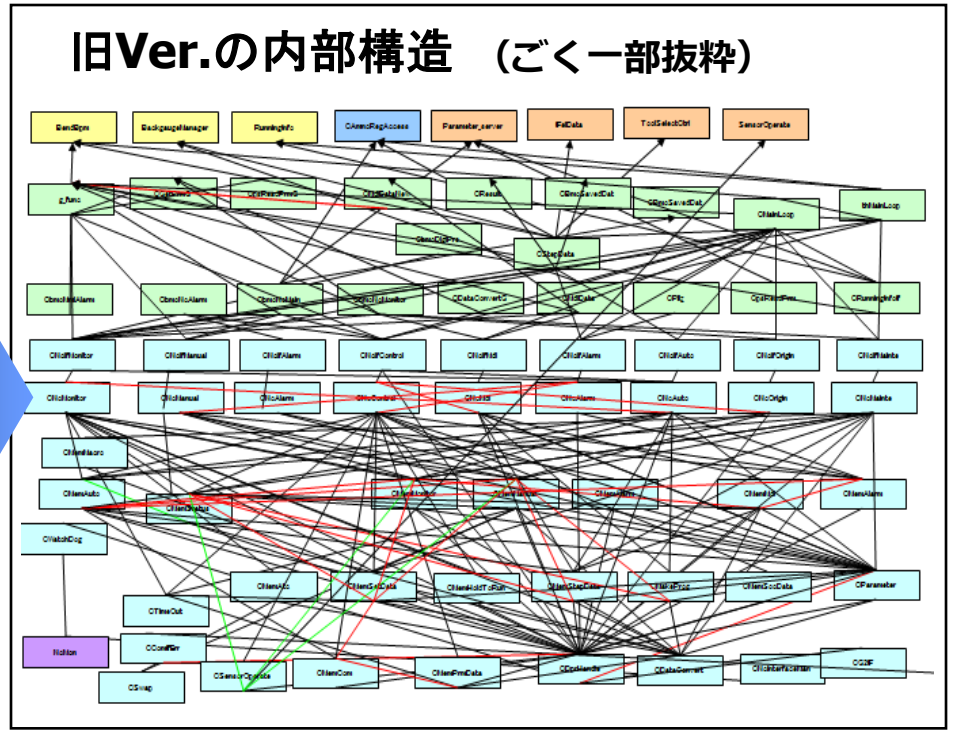
- モデル化技術を『道具』としてきちんと設計する風土を作る
  - 一緒にモデリングをしながら、チームの組織化を進める
- 5つのプロジェクトへ技術と手法を順次導入

【事例1】板金曲げマシンのコントローラー ソフトのモデル化

筑波大学 University of Tsukuba

従来ソフトの問題･･･高機能化・複雑化していくソフトウエア





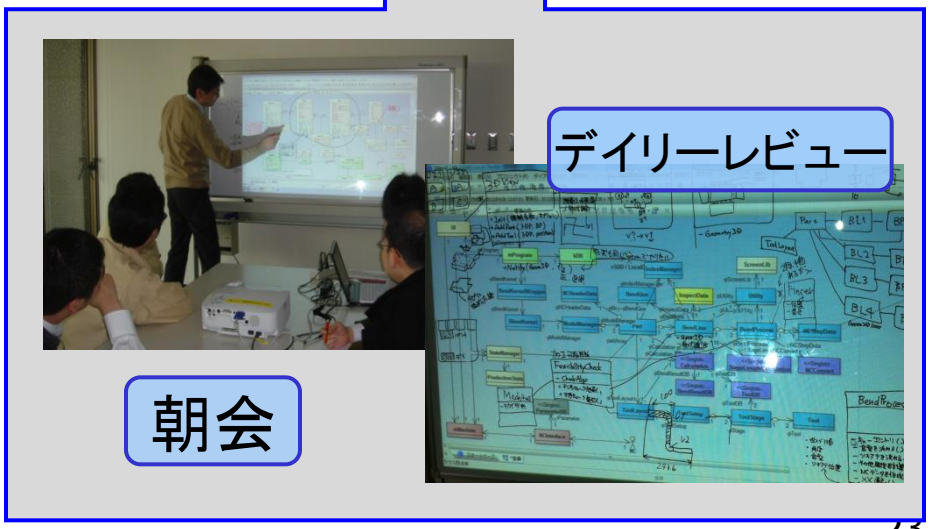
- プログラムの見通しが悪く、開発・テストに時間がかかるようになっていた
  - １ヶ所の変更が多くの機能に影響を与えるようになっていた

従来の手法では品質が確保できなくなってきた。

【事例1】板金曲げマシンのコントローラーソフトをモデル化

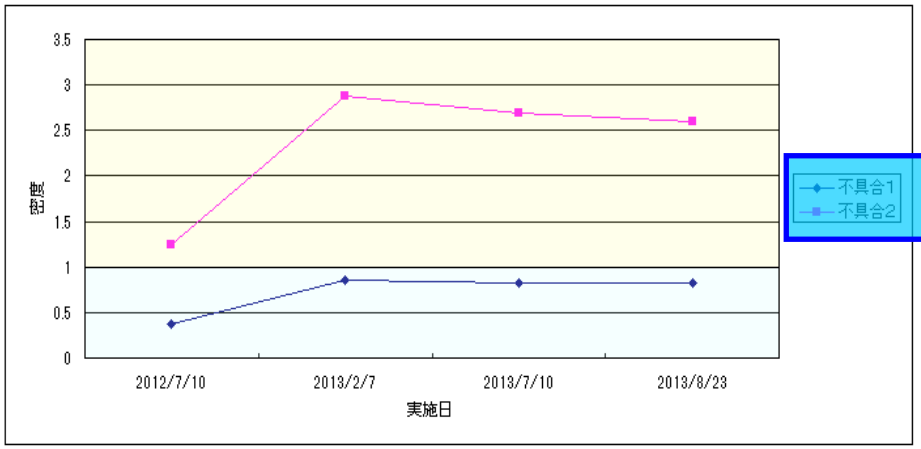
- やはり組込系では、制御対象の特徴をいかに記述できる（＝抽象化）か がカギ
  - 従来は曲げ機械コントローラの機能や処理に着目して設計していた
  - モデル化にあたり、対象となるもの＝板金が素の板の状態から、何回か曲げられて製品に仕上がるまでの様子を表現することにした
    - いずれも、「金型と角度を決めて加圧する」ことの繰り返しであることを「究明」
    - これをモデル化することで一気に「設計内容」が見えるようになった

この部分は当日会場でお見せします

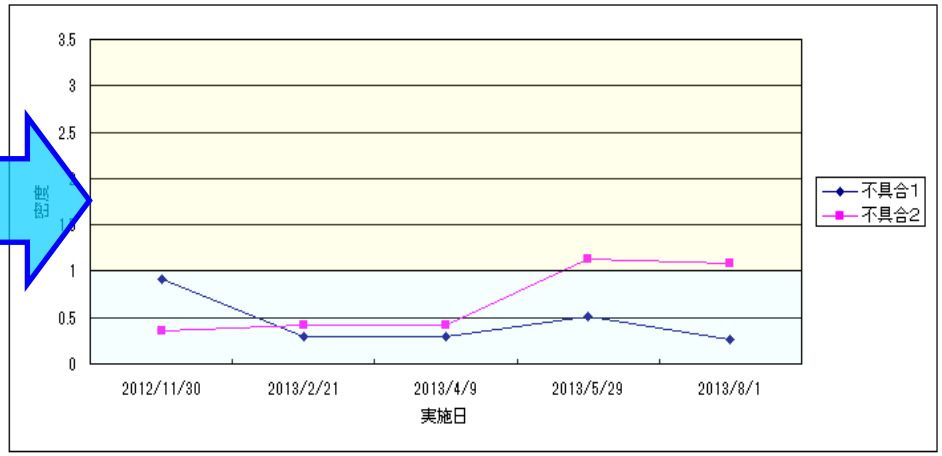


【事例1】モデル化と反復開発の適用効果（品質面）

- ソフトウェアの品質が向上
  - 静的指標の変化
    - 適用前のソフト よりどの程度改善されたか、を見ると･･･



モデル化適用前



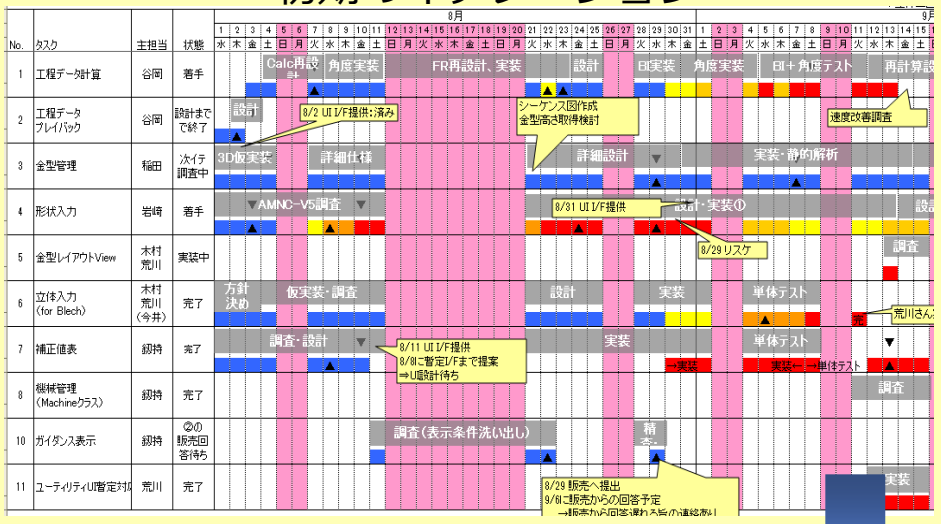
モデル化適用後

品質が大幅に改善されていることがわかる

【事例1】モデル化と反復開発の適用効果（工数面／納期面）

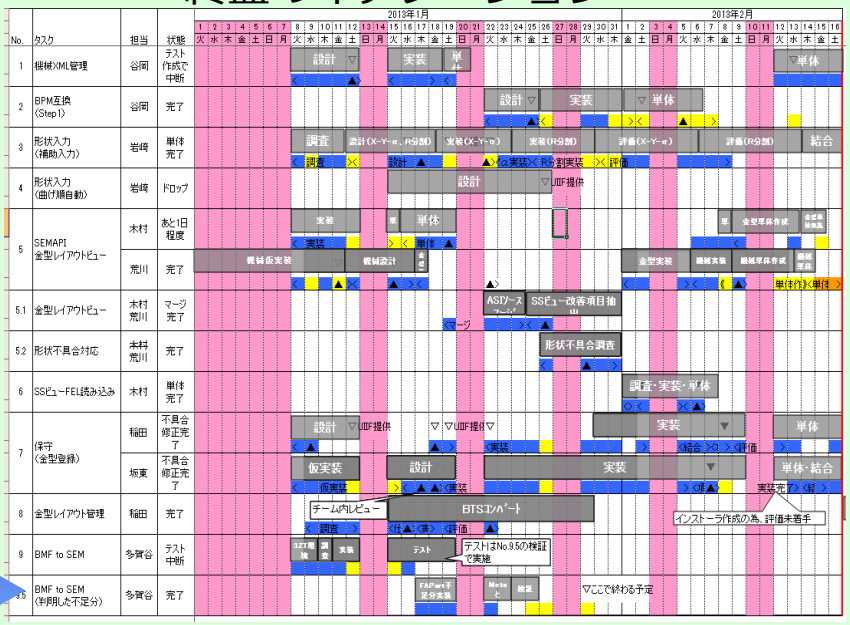
- 見積りの精度が上がり、リリース日が約束できるようになってきた

初期のイテレーション



まだ計画に対する遅れが多い

終盤のイテレーション



ほぼ計画どおりに進捗できるように成長

- 数値に基づいた正確な見積りが出せるようになってきた。
  - 実績を蓄積することにより、自分たちの生産性が数値でわかった。
- 突発的な先行リリース要求に対しても、適切な対応ができるようになった。
  - 見積りを根拠に、「代わりに機能A、Bを落とすことで対応可能」などの回答ができるようになってきた。

納期遅延が解消

工数も削減

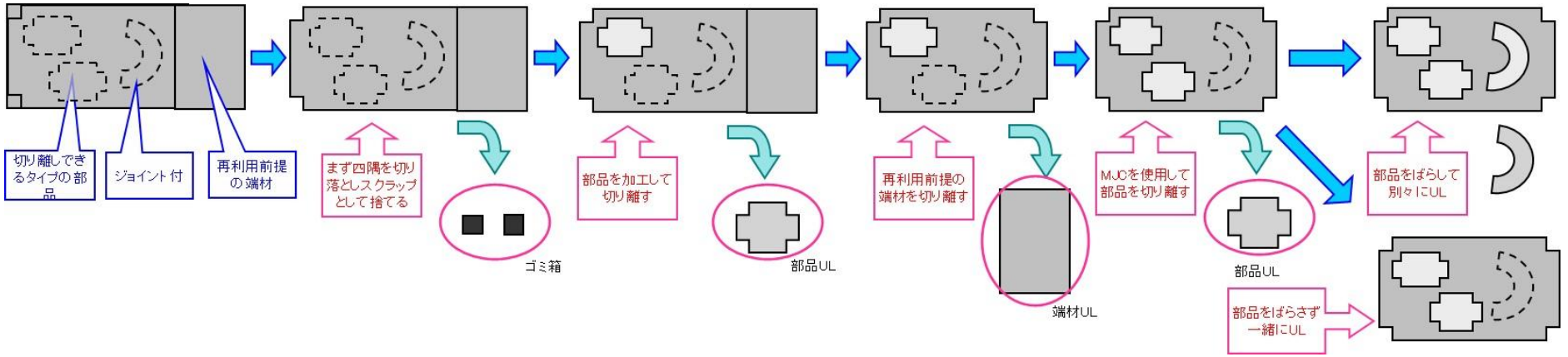
【事例2】自動倉庫システムの課題

- 板金を倉庫から取り出し、加工を施した後、部品の切り離しを行って集積、再び倉庫へ戻すシステム
- ユーザー先ごとに設備構成が異なる
  - 設備構成が異なると各工程で実施する内容が少しずつ異なる
  - これを１つのソフトウエアで実現する必要がある

部品加工と集積のパターンが無数にあり、個別のプログラミングと保守が大変

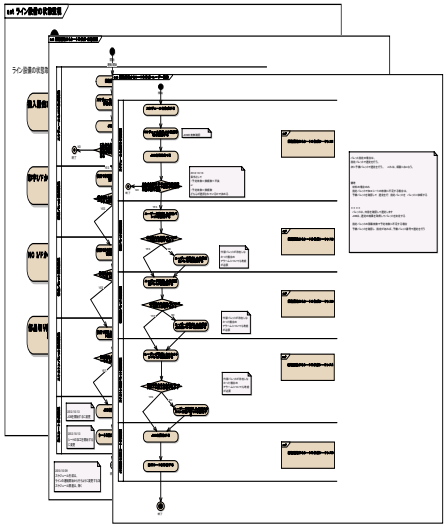
【事例2】自動倉庫システムのモデル化

- 膨大な数のパターンに惑わされて、個別の事象にこだわっても特徴は見えない
  - すべての事象を俯瞰するとそこにルールが見えてくる
  - ルールをモデル化する

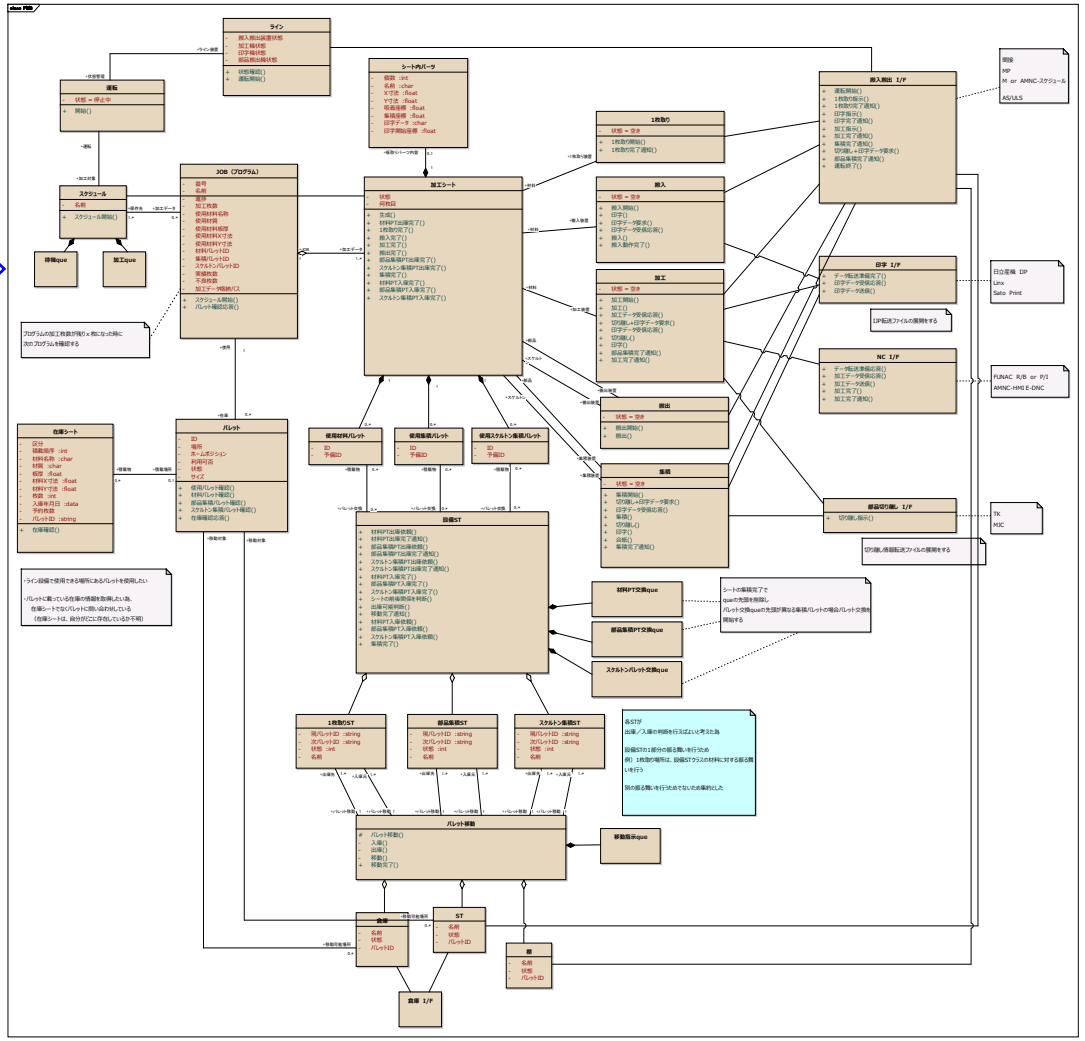


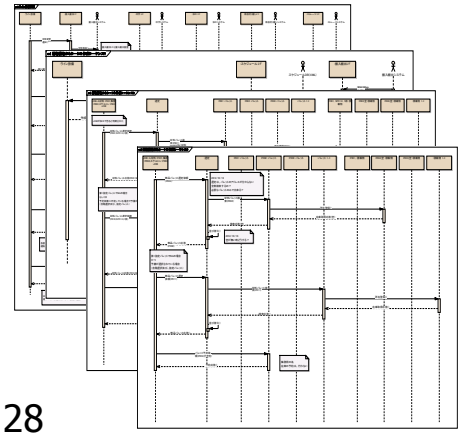
【事例2】自動倉庫システムのモデル化

- 試行錯誤しながら作り始めた**設計図**（クラス図 等 ）



クラス図
を起こす



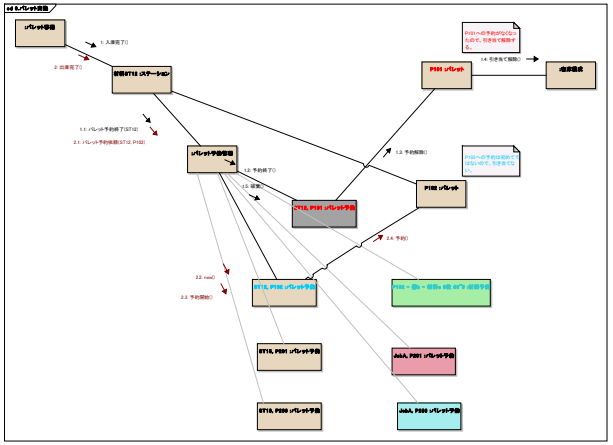


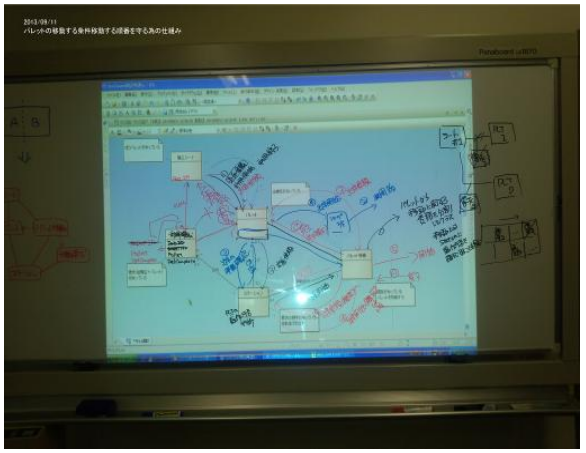
シーケンス
図で検証

モデル化を契機にチームが成長

本人たちの感想から

- モデル化の効果
  - 要求内容から仕様を早く理解できるようになった
    - 「～がしたい」という内容から、システムが行う内容「～が必要」を考え理解するようになった
  - ソフトウェアの構造を図面化することで、メンバーがソフトウェアの全体を理解できるようになり、検討した内容で意見交換が行えるようになった
    - 図面を見ることで、確認すべき場所や影響範囲がすぐにわかるようになり、スムーズに設計が行えるようになった





ソフトウェア資産を活用した戦略的な開発へ

- モデリング（図面化）することによるメリット
  - ソフトウェアの構造が目に見える
    - アーキテクチャを崩さずに機能追加できる
    - アーキテクチャの再利用で品質が安定する
  - 設計図（＝モデル図）とソースコードのひも付けができる
    - ソフトウェアを図面をもとにつくれる
    - 図面からソースコードを（半自動的に）生成できる
      - 上流の検討に工数を割くことができる
  - モデルの共有効果
    - モデル図で変更点と影響範囲を全員が特定可能
    - いままでフルテストまたはノーテストだったものが、適切なテスト計画を策定して実施できる
      - 商品設計の工数が75分の1になった例もあった