ソフトウェア工学

2025年9月22日

海谷 治彦

目次

- 講義の目標
- 単位認定について
- 参考書
- 授業の進め方
- なぜプログラミングは難しいか
- Java, UML, Eclipse
- 受講上の注意

講義の目標

- ソフトウェアを分析・設計してから，プログラムを開発するような人に受講生がなること.
  - いきなり，エディタ(もしくはIDE)でコードを書くのは今後はNG.
  - AIアシストはどうかなぁ.
- 特にオブジェクト指向開発ができること.

- 大規模ソフトウェア開発に有効な各種技法を知る.

単位認定について

- 演習(プログラミングやモデリング)の結果で判断します.
  - 大体，4回前後.
- 中間試験や期末試験は，
  - 今年は行わない予定です.

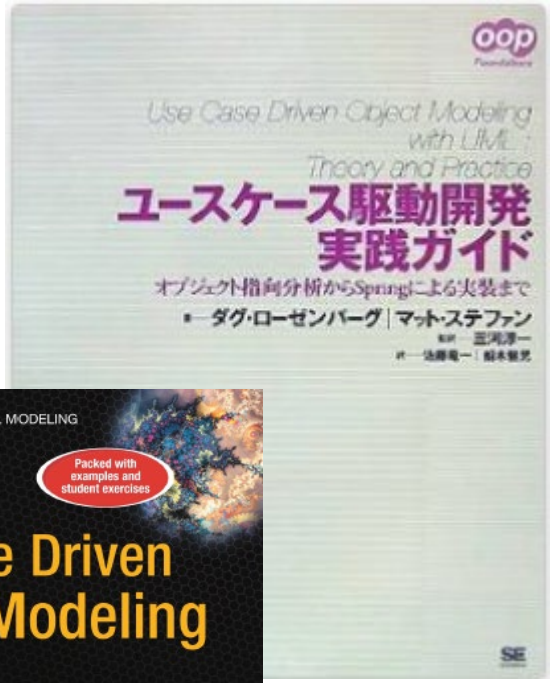
- できれば，JavaやC#等のポピュラーなオブジェクト指向言語知ってると嬉しいが，知らなくても仕方ない.

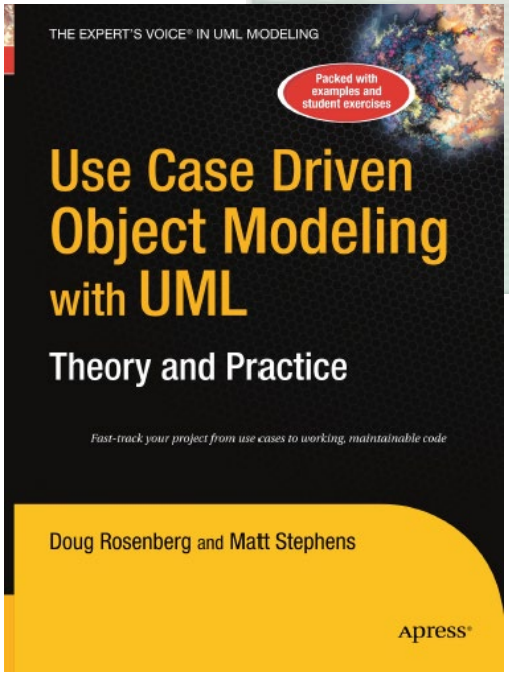
オフィスアワーについて

- 原則，毎週 火曜日 4限
- 場所 20-409

参考書

- ICONIXという手法の解説本です.
- まぁ, この分野ではポピュラーではないかと思います.
- 高いので買う必要はありません.





授業の進め方

- 基本的にシラバスにそって講義やります.
  - 情報システム論と少し中身を入れ替えます.
- 演習をやってもらいます.
  - プログラム，設計，たまには感想や考察.
  - 提出先は webclass の予定.

- スケジュール: webclass で確認してください.
  - たびたび更新されます.

ソフトウェア工学

- ソフトウェア工学は，産業としてのソフトウェアを開発する手順を系統化，自動化することを目的としてます．
  - ソフトウェアの軍事利用が発端だったようです．

- 系統化するための技法として今はオブジェクト指向が中心ですが，他にも色々な技法があります．

ソフトウェア工学の起源

- はじめてソフトウェア工学 という言葉が使われたのは、1968年のガルミッシュ・パルテンヒルケン（GAP，当時の西ドイツ）で開催されたNATO(北大西洋条約機構)の科学委員会。

ミュンヘン中央駅から，
GAPへ行く電車.
（2024年9月現在）

ソフトウェア工学

ってか，工学って何？

言葉の定義 by 広辞苑

- 工学:

  基礎科学を工業生産に応用して生産力を向上させるための応用的科学技術の総称。
- 工業:

  原料や粗製品を加工して有用なものとする産業。
- 生産:

  自然物に人力を加えて、人にとって有用な財を作り出し、もしくは獲得すること。
- 産業:

  生産を営む仕事、すなわち自然物に人力を加えて、その使用価値を創造し、また、これを増大するため、その形態を変更し、もしくはこれを移転する経済的行為。

言葉の定義 by 広辞苑

- 工学:
  基礎科学を工業生産[illegible]用[illegible]向上させるための応用的[illegible]
- 工業:
  原料や[illegible]。
- 生産
  自然[illegible]り出し、もしく[illegible]
- 産業
  生産を[illegible]を加えて、その使用価値を[illegible]増大するため、その形態を変更し、もしくはこれを[illegible]る経済的行為。

工業・工学の根幹は主観的

↓

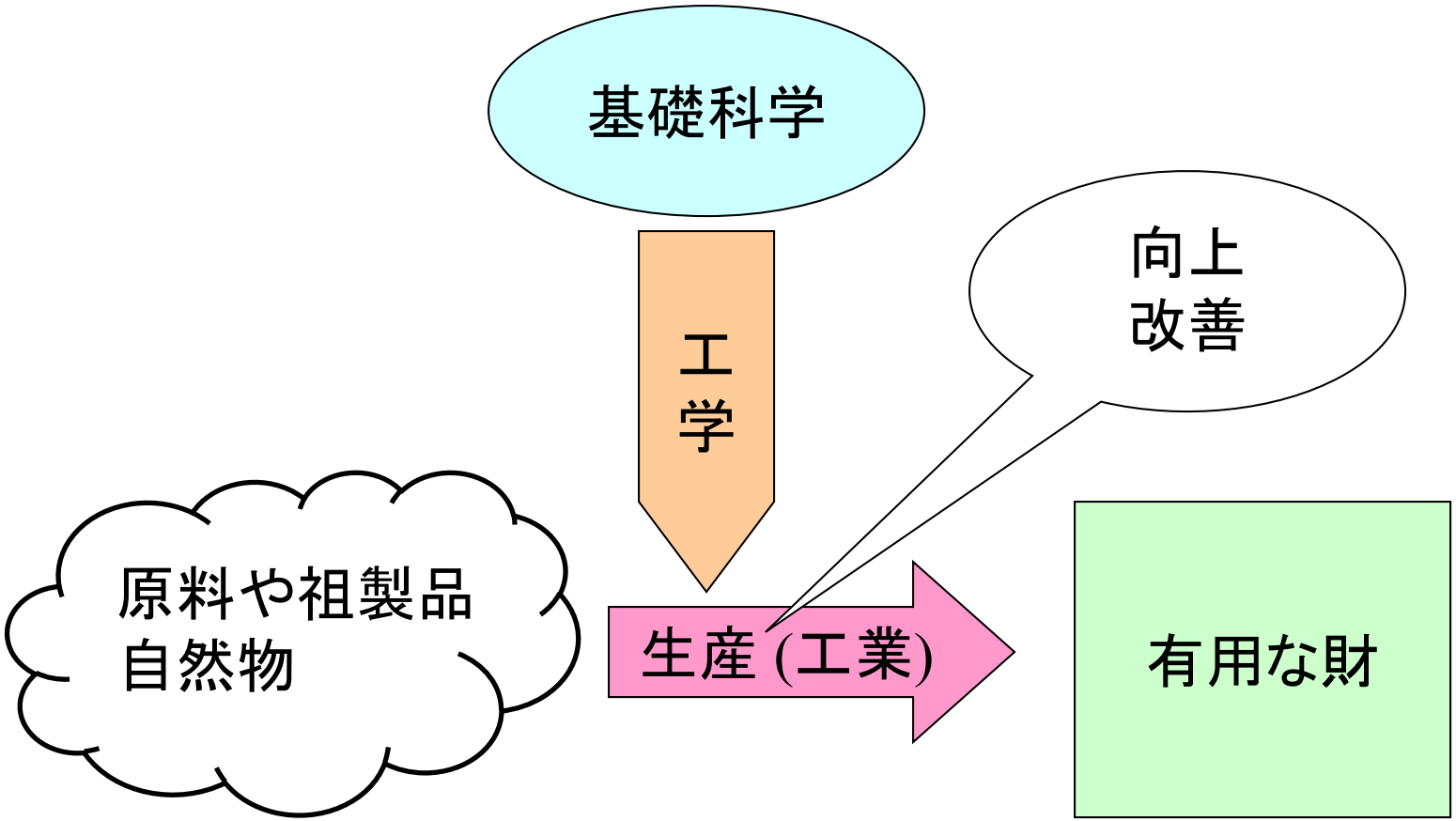
ヒトから見た
有用性，価値に
依存している．

科学

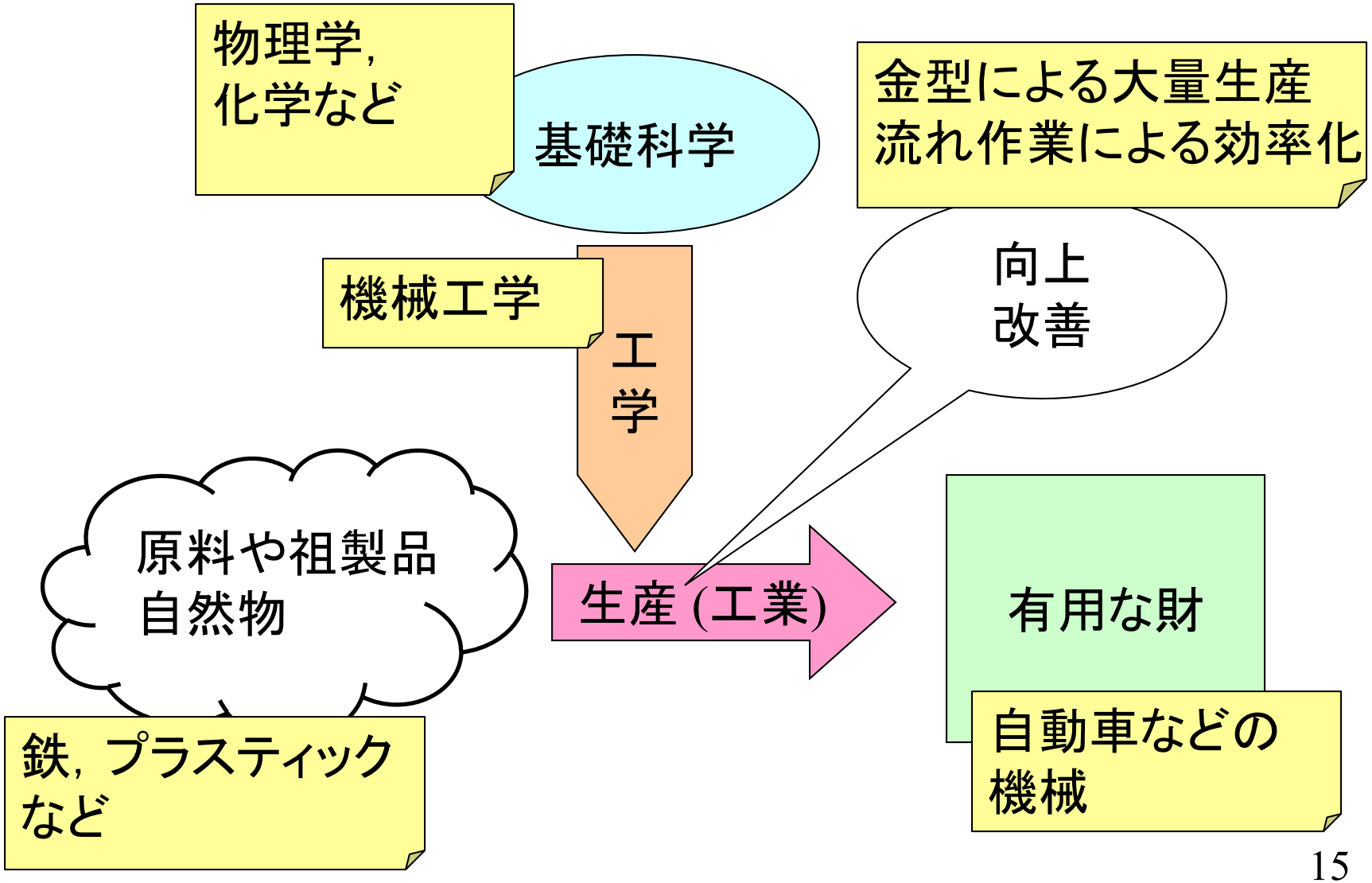
- 観察や実験など経験的手続きにより実証されたデータを論理的・数理的処理によって一般化した法則的・体系的知識。
- また、個別の専門分野に分かれた学問の総称。
- 物理学・化学・生物学などの自然科学が科学の典型であるとされるが、同様の方法によって研究される社会学・経済学・法学などの社会科学、心理学・言語学などの人間科学もある。
- 狭義では自然科学と同義。

[広辞苑 第七版]

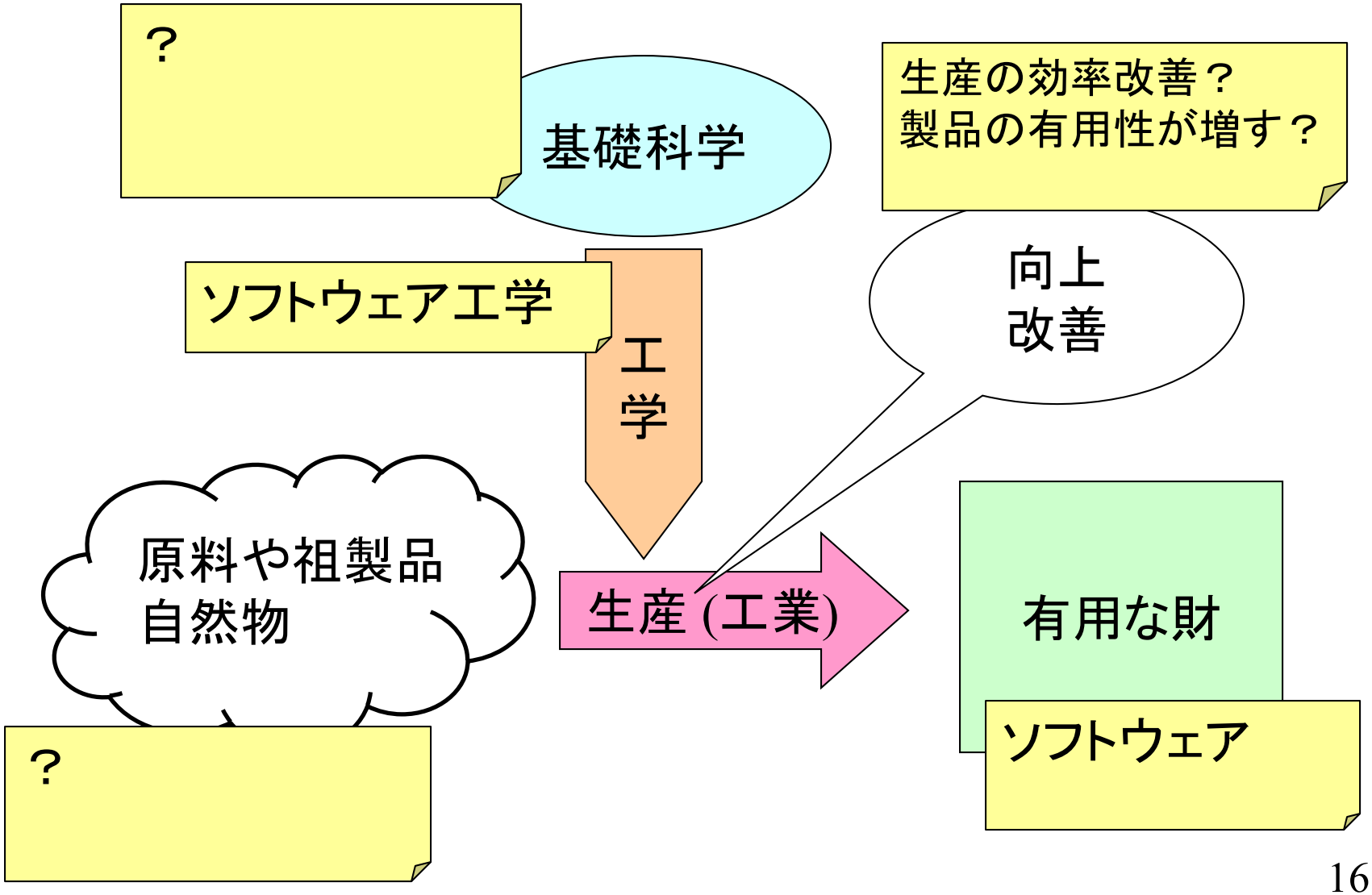
工学



工学と機械工学


物理学，
化学など
基礎科学
金型による大量生産
流れ作業による効率化
機械工学
工学
向上
改善
原料や祖製品
自然物
生産 (工業)
有用な財
鉄，プラスティック
など
自動車などの
機械

工学とソフトウェア開発



プログラミングは何故難しい？

その結論

- なぜプログラミングは難しいか？

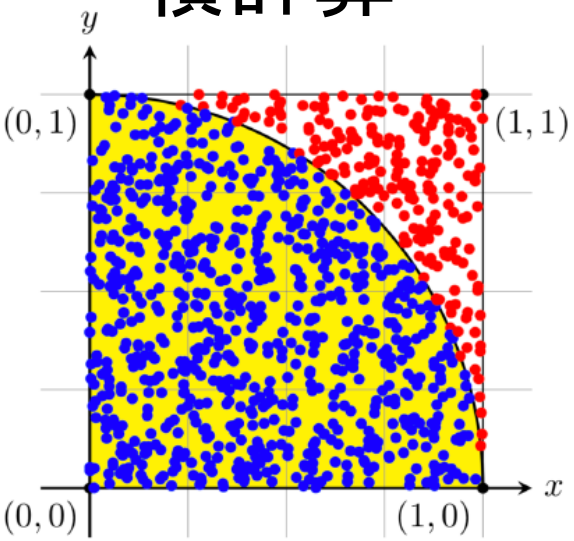
以下のどこかでつまづいている！

1. コンピュータに行わせたいことを理解
2. 理解したことを説明できるレベルまで整理
3. コンピュータにわかる言葉に翻訳

プログラミングとは何か？

- コンピュータにやらせたいことの手順を，コンピュータのわかる言葉で書く.

- 例: リテラシでやったモンテカルロ法での面積計算



- ダーツを，ある領域にランダムに投げ続ける.
- 領域は既知の面積.
- 求めたい面積範囲にダーツが入ったかを座標から計算できる.

理解の失敗例

- もし，以下にあるようなことを理解できなければ，面積計算のプログラムはできない！
  - ランダムにダーツを投げる.
  - ダーツの座標から面積範囲の外か内かを判断する.
- 同様に，銀行業務が理解できなければ，その業務支援ソフトウェアは作れない.

- やりたい事を，コンピュータができる事に対応付けることが理解では重要.

何故，考案ではなく理解か？

- コンピュータにやらせたいことの多くは，現実世界の業務や手順の一部である.
  - 放射性物質の飛散予測の計算の一部（全部）
  - 銀行業務の一部
- そのような業務や手順は，その道の専門家が考案する.
  - 原子力専門家，物理学専門家，気象学専門家
  - 銀行員
- 我々，コンピュータ技術者は，これらを理解し，計算機で(どれだけ)肩代わり可能か判断する.
  - 判断のためには理解が必須！

整理の失敗例

- 個々の処理手順はわかるが，それを一般化（整理）して，他人に説明できない．
  - 例えば，算数や数学にあるようにNやxみたいなパラメータを使って，計算手順を一般化できないとか．
- 銀行業務も個々の決済等の業務を一般化した手順として書けなければ，ソフトウェアを作れない．

整理のポイント

現実業務と計算機が可能なことのバランスが重要

- 現状の計算機(プログラム言語)で実現不可能な整理をしてもシステムはできない.
  - 微分方程式で飛散予測を整理できても，それをコンピュータで直接実行はできない.
- 業務と大きく剥離した形で整理しても，そもそも整理されているか確認しようが無くなる.
  - プログラムを直接見せられても分からない銀行員も多いだろう.
  - 結果として，計算機にやらせたいことが整理されているか，確認しようが無くなる.

説明相手は誰？

基本的に相手は二種類を想定する.

- 計算機
  - 計算機にやらせることを想定しているのだから，計算機に説明することを想定しないと.
  - 曖昧さや，直観は通じない.
- そもそもの業務専門家
  - コンピュータ技術者の理解が合っているかは，往々にして，専門家じゃないとわからない.
  - 専門家は人間だから，基本，長大なプログラムやアルゴリズム記述の意味はわからない.

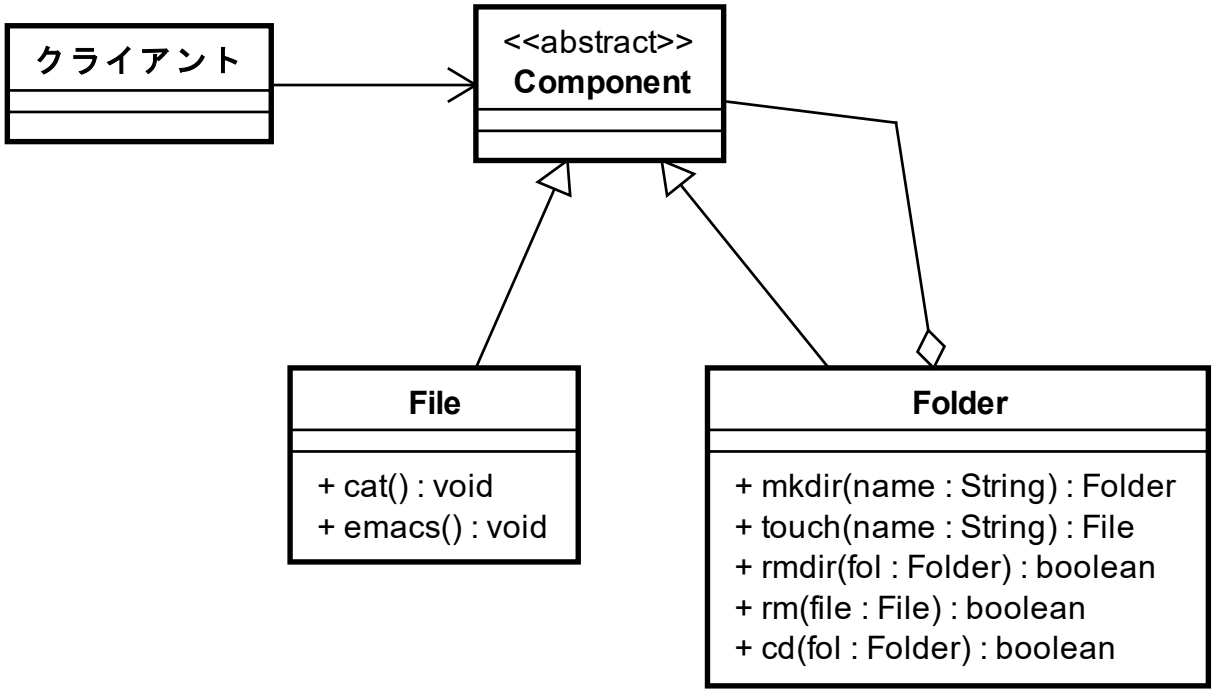
整理に創造は必要か？

- 現実的には，どう整理するかについて，新しい整理法や整理様式を新規に創造する必要は無い.
- 理解したことを，既存の様式を真似て，整理する技術をまずは覚え使えるようにすること.
- 例
  - PCのフォルダや組織階層等，階層的な構造(木構造)を整理するにはよく知られた様式が既に存在する.

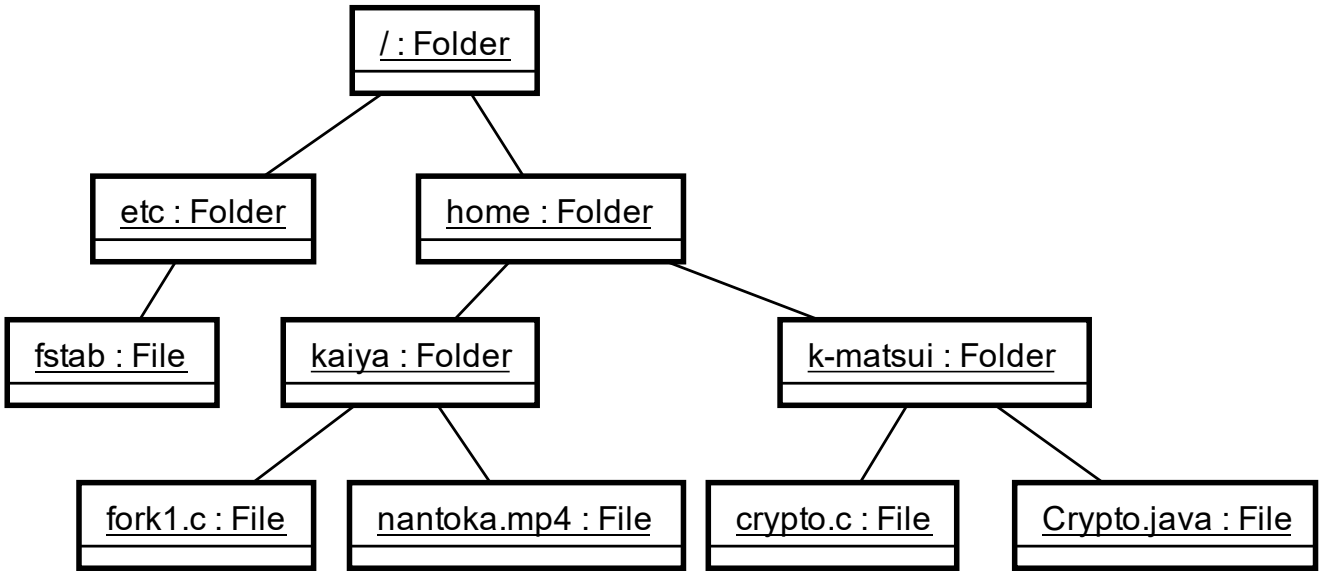
    (コンポジットパターン)
  - この様式を超える整理法を創造するのは多分無理.
- 既存の整理法を数多く知った上で，より良い整理法を創造することを最終的には目指してほしい.

コンポジットパターンの例

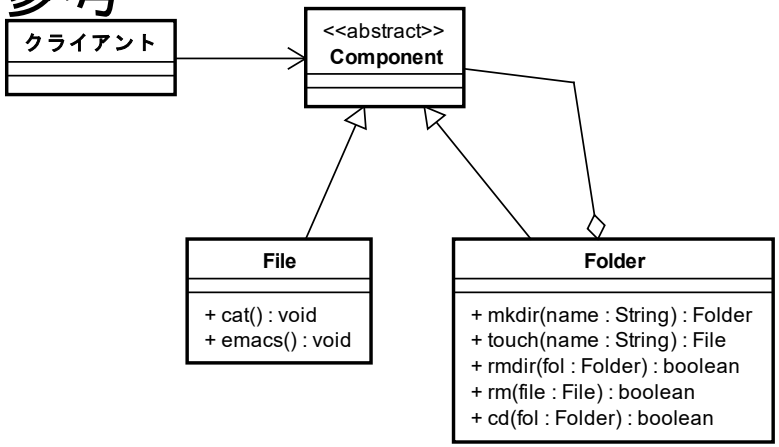
- UNIX風のコマンドをメソッドとして実装.
- 実際，UNIXでは，ファイルとフォルダの作成，削除のコマンドが異なるため，それも準拠.



インスタンスの例


/ : Folder
etc : Folder
home : Folder
fstab : File
kaiya : Folder
k-matsui : Folder
fork1.c : File
nantoka.mp4 : File
crypto.c : File
Crypto.java : File

参考


クライアント
<<abstract>>
Component
File
+ cat() : void
+ emacs() : void
Folder
+ mkdir(name : String) : Folder
+ touch(name : String) : File
+ rmdir(fol : Folder) : boolean
+ rm(file : File) : boolean
+ cd(fol : Folder) : boolean

翻訳の失敗例

- ずばり，プログラム言語を知らない．
  - この辺を補う図書は腐るほど出版されている．
  - 所謂，プログラム言語の授業は結構，この辺のみが重視されている．
- 言語を知っていても，一般化した手順の記述との対応が分からない．
  - 配列，リスト，スタック，木等は知っていても，それが「整理されたコンピュータにやらせたいこと」の何に対応するか分からない．
- 日本語を知っていても，コミュ力が無い人が居るのと同じ．

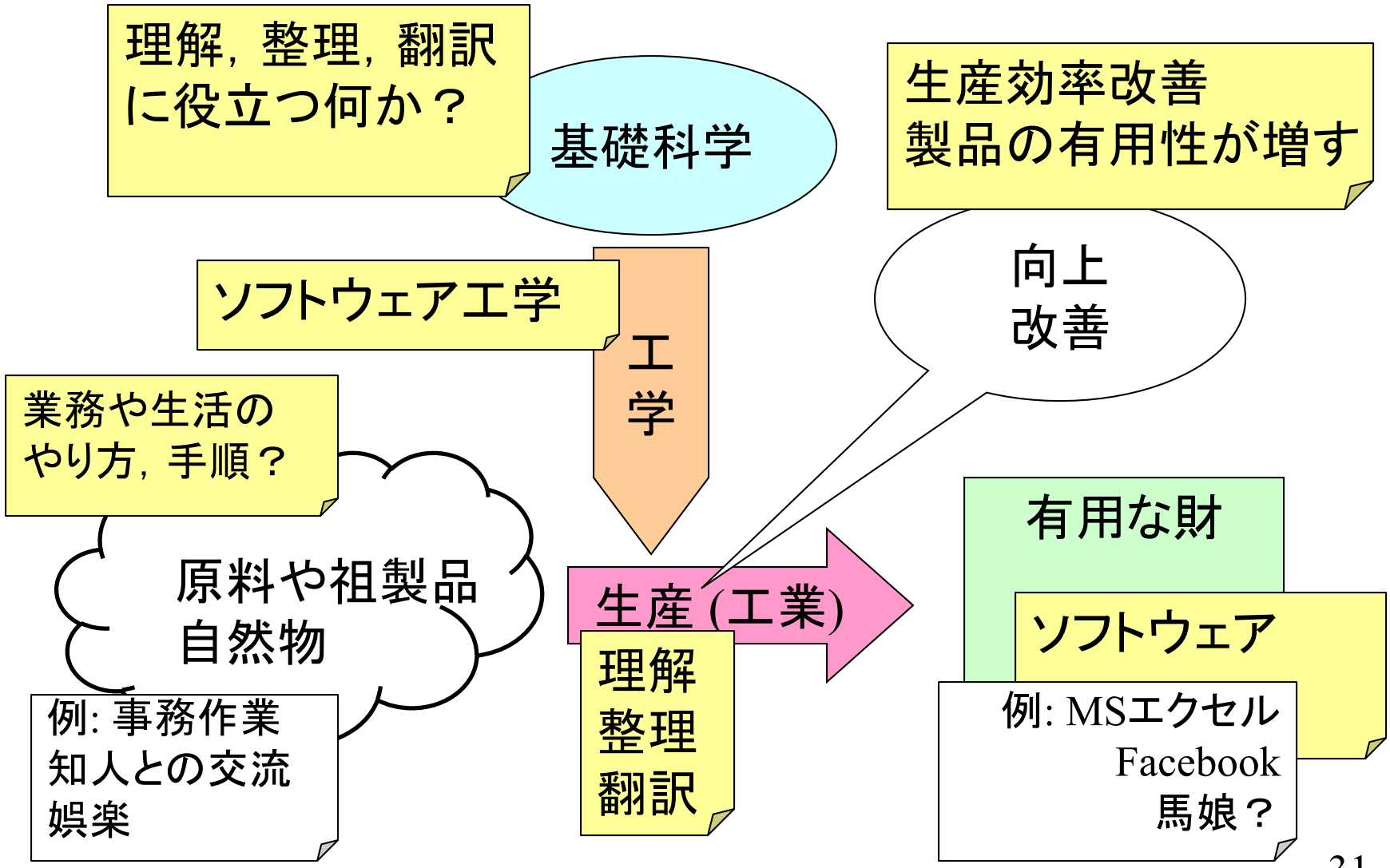
本当に難しいのは理解と整理

- 理解
  - 株取り引きの業務が理解できてますか？
  - ゲーム内での3D表示の人間の描画法を理解できてますか？
  - SPEEDI (スピーディ)の放射性物質の飛散予測法を理解できますか？
  - 迷惑メールとそれ以外との違いが理解できますか？
- 整理
  - 上記を個々の事例ではなく，一般化して整理できますか？（整理する既存法を知ってますか？）
  - 整理したことを人間と計算機の双方に都合よく説明できますか？

計算機屋は下請けか？

- 大筋で YES
  - 対象分野の一部もしくは全部を肩代わりする下請け業務.
- しかし，計算機技術により業務・生活の方が変化することも最近は多い.
  - 技術主導で世の中のあり方・やり方が変わる.
  - 例
    - 検索技術の発達による情報整理法の変化.
      - 図書館等 VS Webやデスクトップ検索
    - 携帯端末の発達による待ち合わせ法の変化.
      - いまどき，きっちり場所と時間を指定しなくても集まれる.
    - SNSの発達により，グループ，伴侶，家族等の在り方の変化.

工学とソフトウェア開発



基礎科学としての授業

- 理解
  - 本学科には無い・・・
- 整理
  - 数理論理学関係 (命題論理や述語論理)
  - オートマトン，アルゴリズム
- 翻訳
  - プログラミング系
  - OS, ネットワーク
  - 画像処理，暗号，データベース

ソフト開発の問題の種類

- 規模の問題
  - デカいものが多くなっている.
  - 結果，開発にかかわる人の種類も人数も多い.
- 量と多様性の問題
  - 多種多様なソフトが必要
- 依存関係の問題
  - 今日，単独で動作するソフトは少ない.
  - ライブラリや外部サービス等の更新・変更のチェックが必要.
- 品質の問題
  - 自動車，飛行機，原子力プラント等のためのソフト
  - 信頼性，セキュリティ，使い勝手の問題
- 早い変化の問題
  - 業務の変化が早いので，システムも迅速に変更しないといけない.

規模の問題

- 現実のプログラムは非常に大きい（長い）
  - OS 1億行以上のものもある
  - テキストエディタ 数十万行のものも

- こんなものを一人でgccとエディタだけで，ちまちま作っていては，完成できない．

- 複数人で共同して開発するための技法やツールが必要．

リアルなプログラム例

unzip.c

```
    G.filespecs = argc;
    G.xfilespecs = 0;

    if (argc > 0) {
        char **pp = argv-1;

        G.pfnames = argv;
        while (*++pp)
            if (strcmp(*pp, "-x") ==
                if (pp > argv) {
                    *pp = 0;                /* terminate G.pfnames */
                    G.filespecs = pp - G.pfnames;
                } else {
                    G.pfnames = (char **)fnames;  /* defaults */
                    G.filespecs = 0;
                }
                G.pxnames = pp + 1;       /* excluded-names ptr: _after_ -x */
                G.xfilespecs = argc - G.filespecs - 1;
                break;                     /* skip rest of args */
            }
        G.process_all_files = FALSE;
    } else
        G.process_all_files = TRUE;       /* for speed */

#else /* !SFX || SFX_EXDIR */             /* check for -x or -d */
```

C source file | length : 97,148 | lines : 2,656 | Ln : 1,158 | Col : 37 | Sel : 0 | 0 | Unix (LF) | ANSI | INS

アプリ unzip.exe を構築するには，このファイルを含む153個のファイルが必要．総行数は約8万2千行．

人種(職種)の問題

- 皆が開発者では無い.
  - 少なくとも，最終的には開発には関係ないユーザー(他人)が使うことになる.
- 開発者といっても色々いる.
  - プログラマだけではない.

- やはり，複数人が連携するための技法やツールが必要となる.

多様性の問題

- 自動車や建築物等とはくらべものにならない多様性.
  - 「移動する」,「居住する」等の共通となる目標はソフトには無い.
- 同種のソフトでも客や国に合わせての多用なカスタマイズ.
  - レジシステム一つとっても店舗や販売物によってかなり違う.

依存関係の問題

- 単純なアプリでさえ多くの外部ライブラリや言語処理系を利用している.
  - openssl, XMLの処理ライブラリ等
- 加えて，動作中の他のサービスに依存するソフトが今日では普通.
  - そもそもIPネットワーク(Internet)が無いと動かないものが多い.
  - 他，データベース，検索エンジン，AIエンジン
- 依存する外部の部品の更新で，アプリが動かなくなる恐れがある.

変化の速さ

- 業務や生活の中でソフトは使われる.
- 業務等の変化は以前より激しい.
- その変化にシステムも追従しないといけないが，（普通では）システムは自動追従できない.
- 手作業で追従するにしてもコストがかかる.

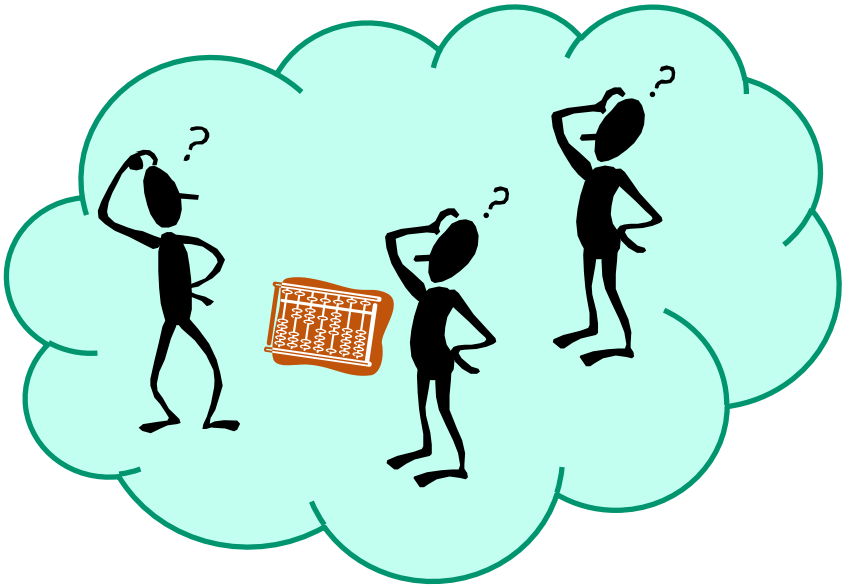
- ソフトを構成する部品自体の変化も早い.
  - 頻繁なバージョンアップと互換性の無さ.

模型（モデル）を使う

- いきなり実機で理解，整理，翻訳をしたらコストもかかるし危ない．
  - 飛行機だったら落ちてしまう・・・
- そこで，模型を使って，理解や整理を行うのが一般的．

- 模型のことをモデル，モデルを使っての机上の吟味をモデリングと呼ぶ．

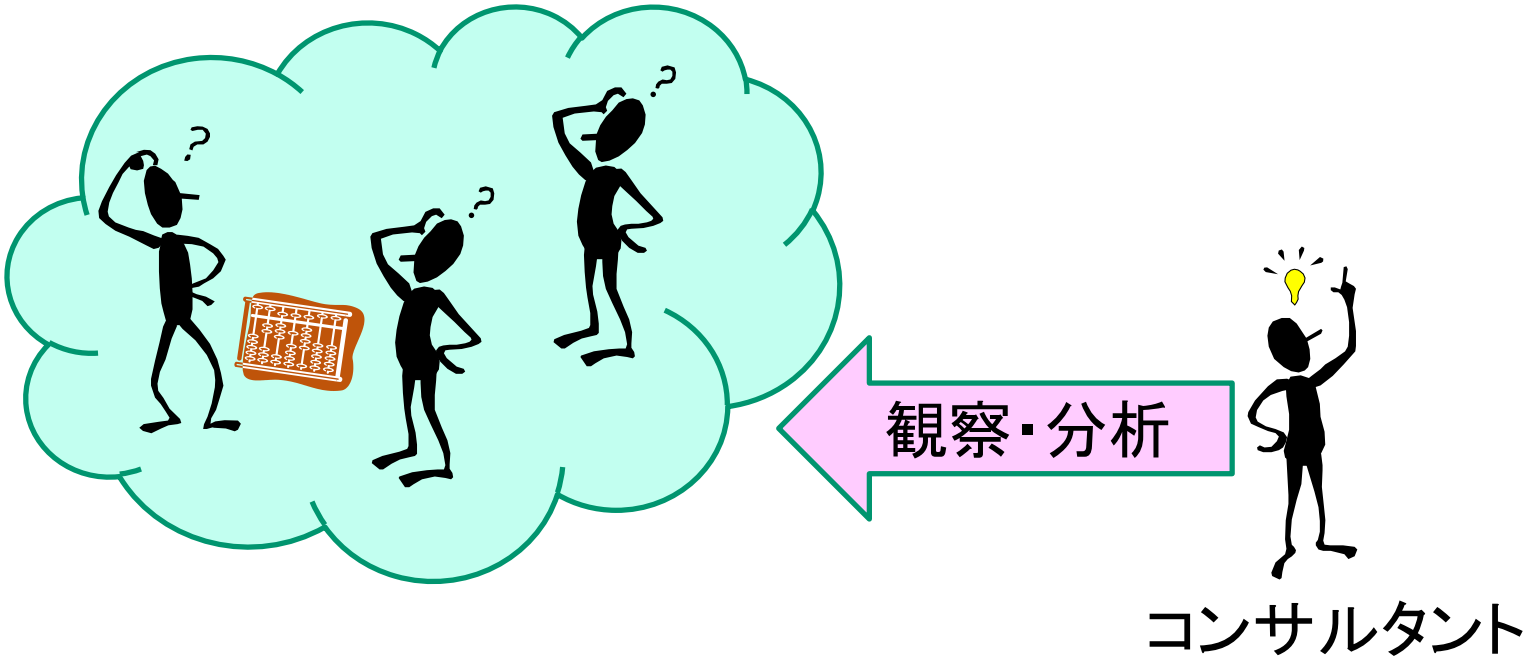
モデリングとは 1/4



仕事や生活と,
それにかかわる人々や道具

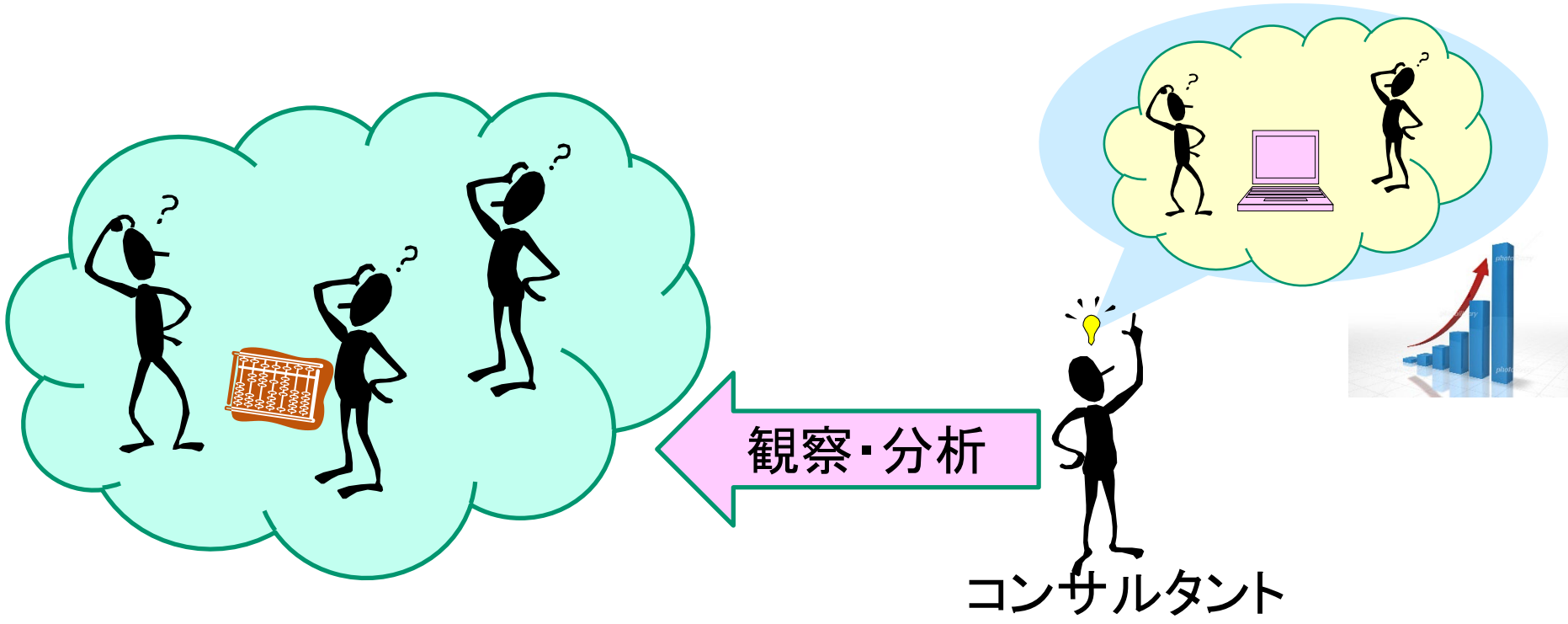
- 人々はなんらかの仕事や活動をしています.
  - 最初はソフトウェア，コンピュータとは関係なく.
- 仕事例: 銀行業務，物品販売，ホテル経営‥
- 生活例: 音楽鑑賞，ゲーム，育児，介護‥

モデリングとは 2/4



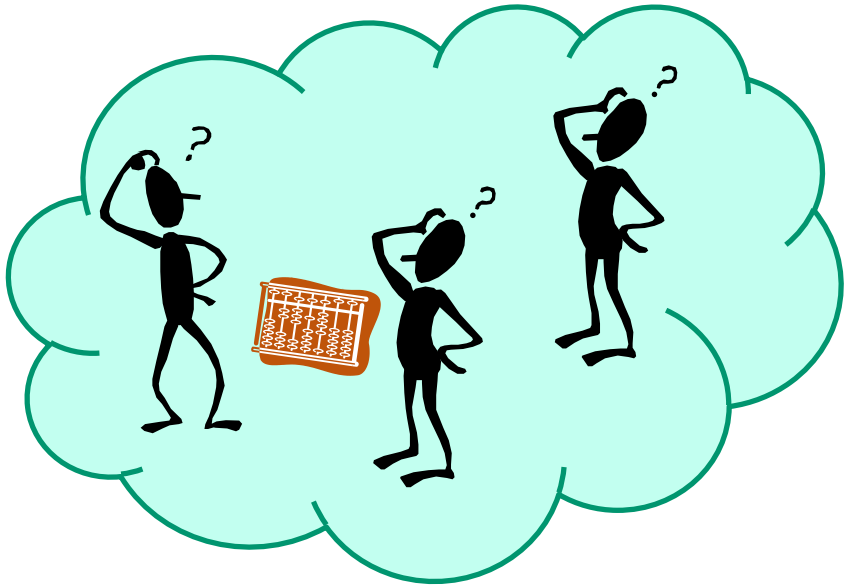
- コンサルタント等が仕事・生活を観察して・・・

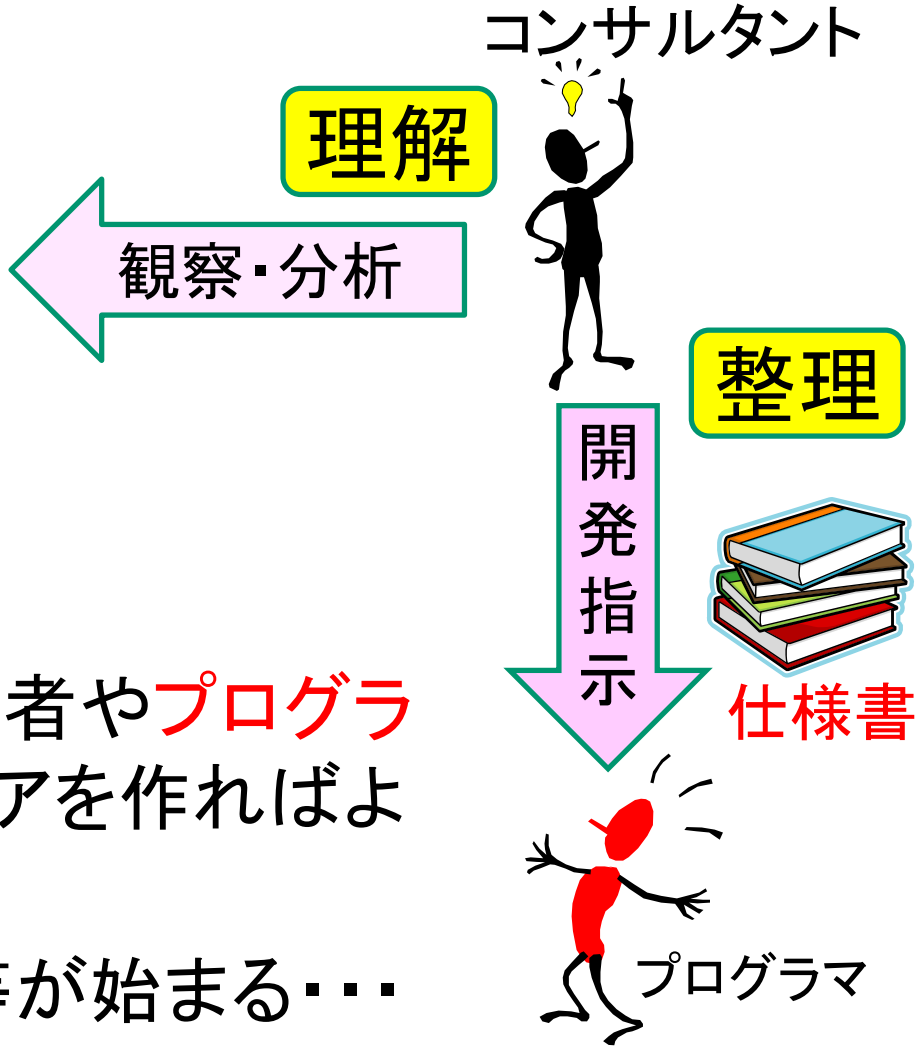
モデリングとは？ 3/4



- ソフトウェアをどこかに導入したほうが仕事や生活がより良くなるかどうかをコンサルタント等が検討します.

モデリングとは？ 4/4

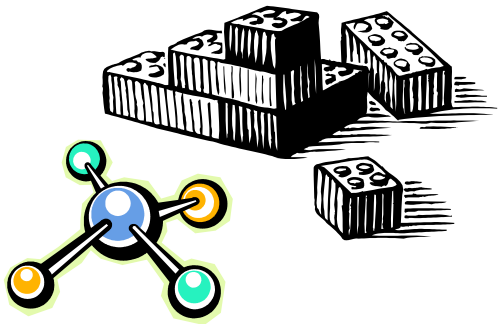




- 分析結果に基づき，開発者やプログラマ等に，どんなソフトウェアを作ればよいか指示します．
- その後，プログラミング等が始まる・・・

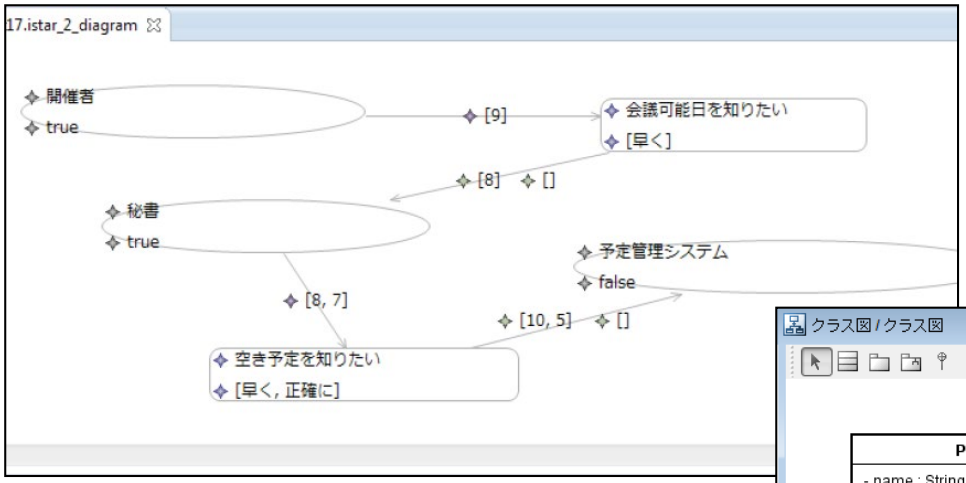
「理解」や「整理」をするためには？

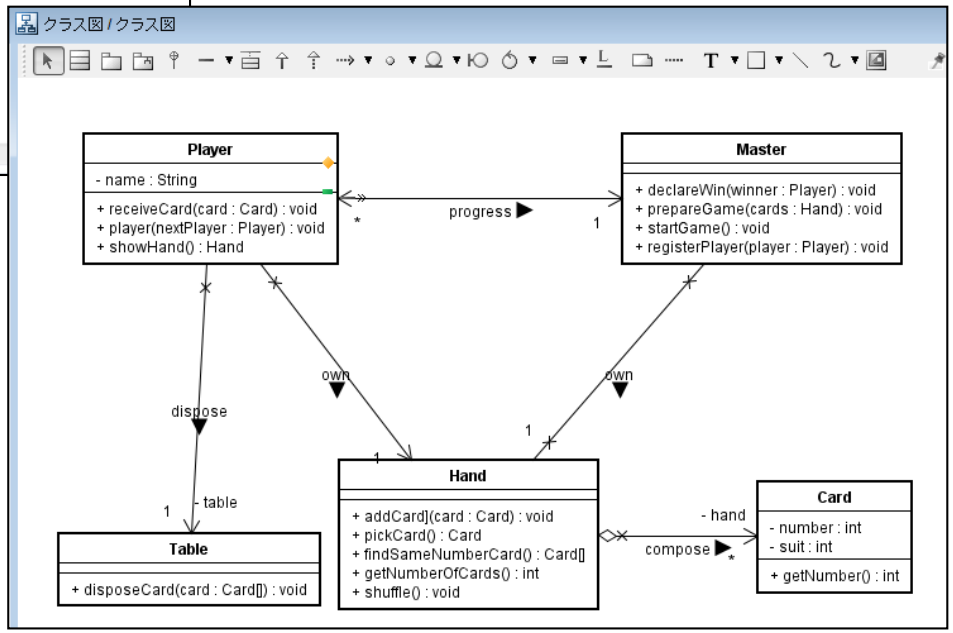
- ソフトウェアの導入を検討している業務や生活の模型を作って，
- 模型の上で，導入したらよさそうなソフトウェアを模擬的に導入し，
- 導入効果を吟味したりする．



- この模型のことを「モデル」，モデルを作ることを「モデリング」と呼ぶ．

モデルの例





ほとんどのモデルは上記のような図式言語で書かれる.

モデルの読者は誰？

- プログラマ等の開発者
  - モデルを「仕様」として用いる場合，当然，開発する人であるアーキテクト，ソフトウェア技術者，プログラマ等が読者となる.
- コンサルタント
  - モデルを書くのがコンサルタントだが，書いたモデルが正しいか随時吟味するため，当然，読者でもある.
- その他，利害関係者
  - 導入されるシステムを利用する人
  - システム導入で活動に影響(リストラとか)を受ける人
  - 導入で利益を得る人

要求と仕様の違い

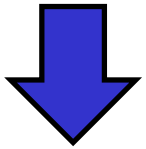
- 色々な人が色々な定義を提唱しているが，本講義では，M. Jackson の定義をベースにする.
- 要求 (Requirements)
  - 現実世界の事物がどうなってほしいかという人間の願望.
- 仕様 (Specification)
  - 要求を達成するために，現実世界に追加される情報システム(および付随する機械)の振る舞いの規定.
  - 仕様書の読者はプログラマ等の開発者である.
- 実務では要求と仕様は区別されないことがあるが，本講義では上記の定義を用いる.
- 尚，要求を「要件」と呼ぶ場合があるが，要件という用語は本講義では用いない.

要求と仕様の違いの例

業務・活動	要求	仕様
動物園等の入場者を管理する業務.	入場券を買った人だけ入場できるようにしたい.	● 入場券によってゲートが開く. ● 開いたゲートは一名のみ通過できる. ● 通過したらゲートは閉まる.
交差点の交通整理をする業務.	交差点で車等が衝突しないようにしたい.	● 交差する道の信号機を互い違いに赤・青にする. ● 双方の道が青になる時間帯をつくらない. ● 双方の道が赤になる時間帯を作る.
集中治療室(ICU)での患者の容態監視.	患者の容態が急変したら，看護師，医師に緊急連絡したい.	● 容態を示す情報(血圧等)を患者から常時得る. ● それぞれの情報の正常値を登録する. ● Etc…

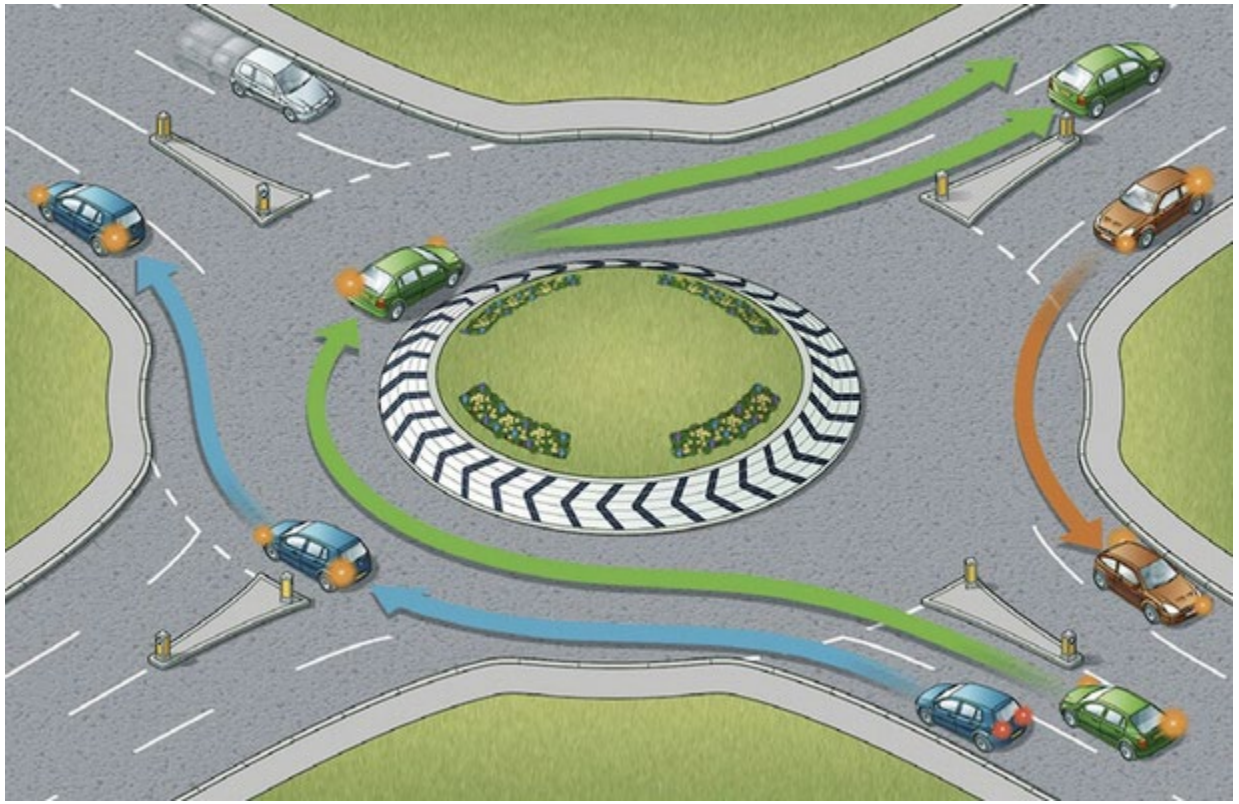
要求と仕様の関係

- 前頁の例からわかるように，要求を満たす仕様は複数存在しうる．(一個も存在しない場合もあるが)
- それぞれの仕様は特定の理論や技法や機器を利用している．
  - 個々の理論や技法に注視する研究室が多いでしょう．
- ある理論・技法・機器に過度に肩入れしていると，本来，それによって達成すべき要求を軽視してしまう場合がある．



- 常に要求を念頭におき，別の解決法(仕様)を考えてみましょう．

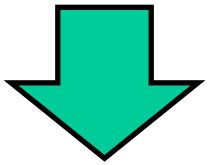
交差点の別仕様



Roundabout や rotary といわれる交差点，信号機は無い

本末転倒な例 1/2

- 以下の話は逸話で嘘ですが.
- 要求「宇宙船で手書きのメモをしたい」



- NASAの仕様
  - 「莫大な予算をかけて，重量が無くても書けるボールペンを開発！科学の勝利だ！」
  - 「莫大な予算をかけて，最新鋭の高信頼性タブレットを開発．科学の勝利だ！」
- ソビエト(現ロシア)の仕様

本末転倒な例 2/2

- 以下の話は逸話で嘘ですが.
- 要求「宇宙船で手書きのメモをしたい」
- NASAの仕様
  - 「莫大な予算をかけて，重量が無くても書けるボールペンを開発！科学の勝利だ！」
  - 「莫大な予算をかけて，高機能タレット」
- ソビエト(現ロシア)の仕様
  - 「エンピツと紙でいいじゃん.」

「要求仕様書」,「RFP」

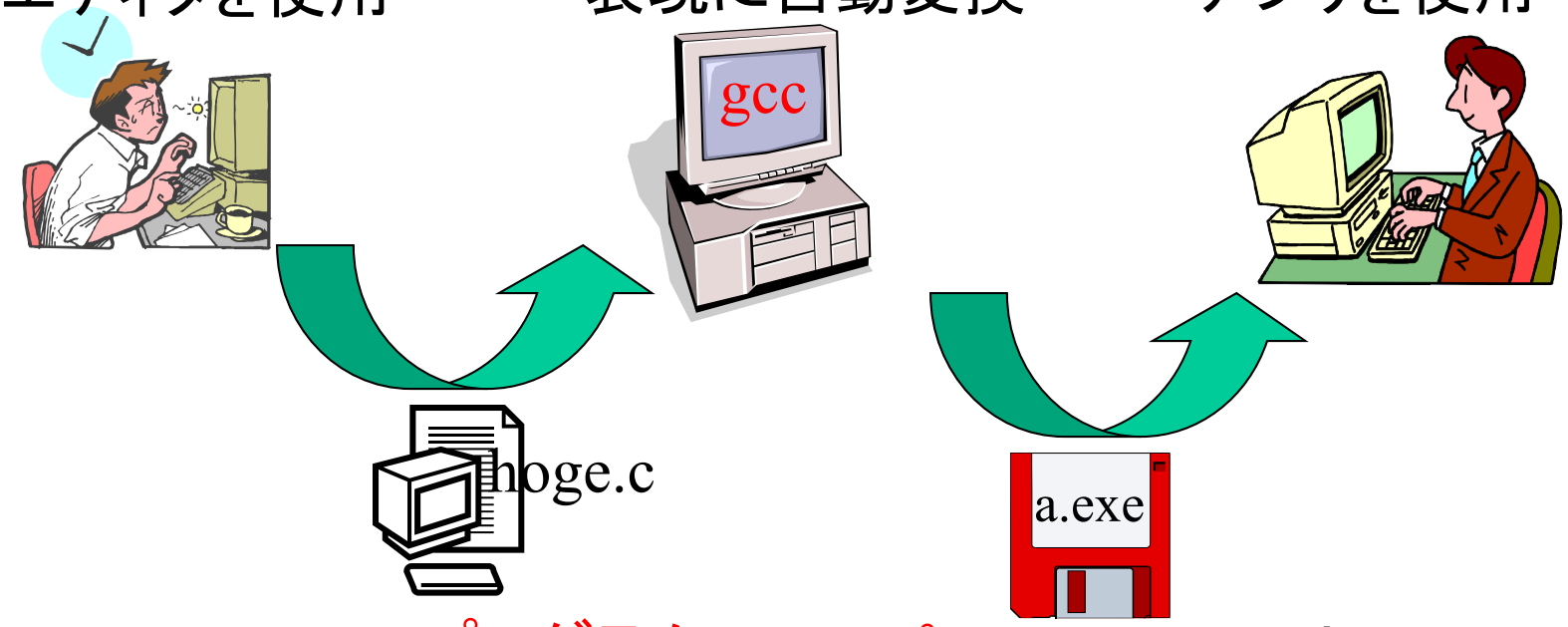
- 世の中には表題のような文書が多数ある.
- 要求仕様書
  - 前述の要求と仕様の双方が書かれたものが理想.
  - 実際は仕様が詳しく書かれており，要求の説明が不十分なものが多い.
- RFP
  - Request for Proposal
  - 内容は要求仕様書とほぼ同じ. むしろ，要求のみ書くのが理想.
  - 公募や入札等を通して，実際に開発する企業体を募集するという意味合いが大きい.
  - 要求仕様書と同じく，仕様は詳細であるが，要求が不足している問題をはらむ.

簡単なプログラムの開発

プログラミング
手作業
エディタを使用

コンパイル
機械が理解できる
表現に自動変換

実行
生成された
アプリを使用



(ソース)プログラム
(ソースコード
hoge.c 等)

アプリケーション (実行ファイル)
(ロードモジュール
a.exe 等)

(ソフトウェア)生産の3つの視点

- プロダクト
  - 何を作るのか，中間成果物を含めて.
  - 要求書，仕様書(ユースケース図)，設計書(クラス図)，コード，テストケース等
- プロセス
  - どんな手順で作るのか.
  - ICONIXも一つの例.
- プロジェクト
  - 誰が何をいつ作るのか？

有名なプロセスの例

- ウォーターフォールモデル
  - 要求分析，設計，実装，テストを順に行なう.
  - 基本，前段階が完了してから次にいく.
  - 文書駆動型ともいえる.
- スパイラルモデル
  - プロトタイプやシミュレーションを用いて，開発全工程を複数回，繰り返し，システム全体の問題点を段階的に洗い出す.
- 段階的開発モデル
  - システムを小さい部分に小分けして開発する.

それぞれの問題点

- ウォーターフォールモデル
  - 開発の最後にならないと問題が健在化しない.
  - 前段階のコストを既に消費しているのに，ソフトの完成に至らない危険がある.
- スパイラルモデル
  - プロトタイプやシミュレーションに手間とコストがかかる.
- 段階的開発モデル
  - とりあえずコーデイングするという素人と同じ開発状況に陥り，保守しにくいものになる.

アジャイル開発

- 契約や計画よりも，開発者と顧客の協調を重視する.
  - 場合によっては物理的に同じ部屋や場所に集まることを重視する.
- 計画の変更も積極的に受容する.
- 文書作成をなるだけ軽減する.
- 一般に小規模開発に向いているといわれる.

- 例 XP(エクストリームプログラミング), スクラム，リーン開発，DevOps
- ICONIXもアジャイルに分類されるとあるが，ウォーターフォール的なカラーも強い.

ソフトウェア開発とAI

- 理解
  - IoTデバイスから得られる監視情報から，人の要望や問題点を自動認識できるかもしれない.
- 整理
  - 理解した問題点の多数の事例を生成もしくは収集して一般化してくれるかもしれない.
- 翻訳
  - 既に実践投入されている.
    - Cursor, Windsurf 等

新人技術者とAI

- 新人技術者がやる，できるような仕事は既にAIが無料で実施してくれる.
- じゃぁ，人を雇うのを止める？？

- 10年後の会社のことを考えると愚かな選択と言う人もいる.
- 会社を10年周期でスクラップアンドビルドして，よそで育った人材を盗むという考え方もできる.

今後の話題について

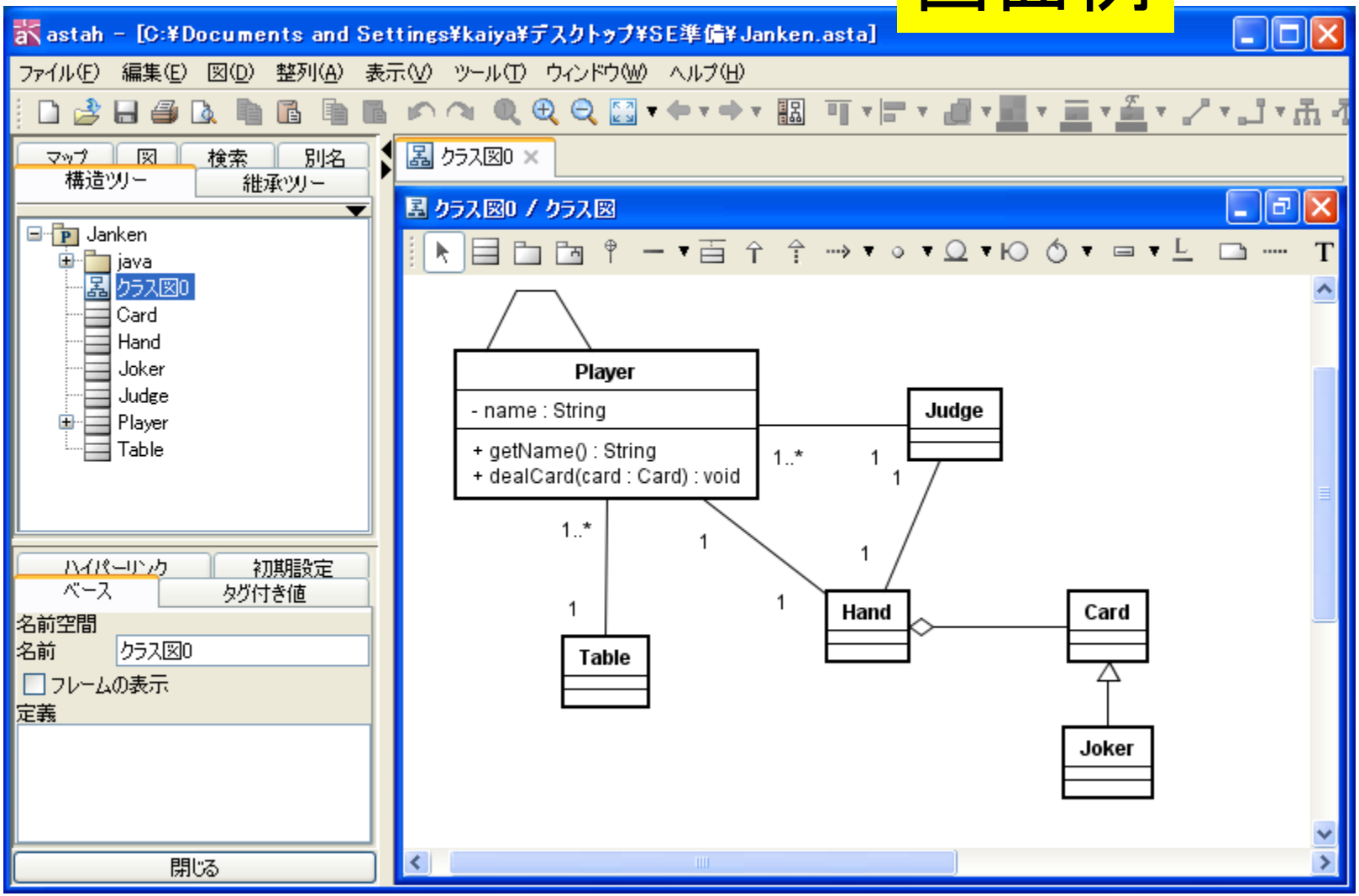
- 仕様，設計を表記するための記法.
  - 要求は情報システム論の方で中心的に行う.
- その記法の典型パターン.
- 設計を動くプログラムに変換する手法.
- 動作テスト.
- 機能以外の留意点 – 品質
- 再利用，保守，見積

astah

- モデルを描くためのお絵かきツール
- どのようなソフトにするかの設計をする際に利用する.
- この分野では珍しく made in Japan!

- 実装に近い形でソフトを設計すれば，コードのひな型も生成してくれる.
- 概念的な設計にも利用できる，例えばビジネスモデリング等.

画面例


astah - [C:¥Documents and Settings¥kaiya¥デスクトップ¥SE準備¥Janken.asta]
ファイル(F) 編集(E) 図(D) 整列(A) 表示(V) ツール(T) ウィンドウ(W) ヘルプ(H)
マップ 図 検索 別名
構造ツリー 継承ツリー
Janken
java
クラス図0
Card
Hand
Joker
Judge
Player
Table
ハイパーリンク 初期設定
ベース タグ付き値
名前空間
名前 クラス図0
フレームの表示
定義
閉じる
クラス図0
クラス図0 / クラス図
Player
- name : String
+ getName() : String
+ dealCard(card : Card) : void
Judge
Hand
Card
Table
Joker
1..*
1
1..*
1
1
1
1

受講上の注意

- 受講登録を行い，WebClass から授業ページ等の情報を得て，この授業のページを参照できるように必ずしておいてください．
- 自分のラップトップ(ノートパソコン)は必ず用意してください．
  - Windows PC を想定しています．
    - バージョンはupdateして最新にしてください．
  - MacでもOKのはずだが，Macの場合は自分で調べてね．

来週までの宿題

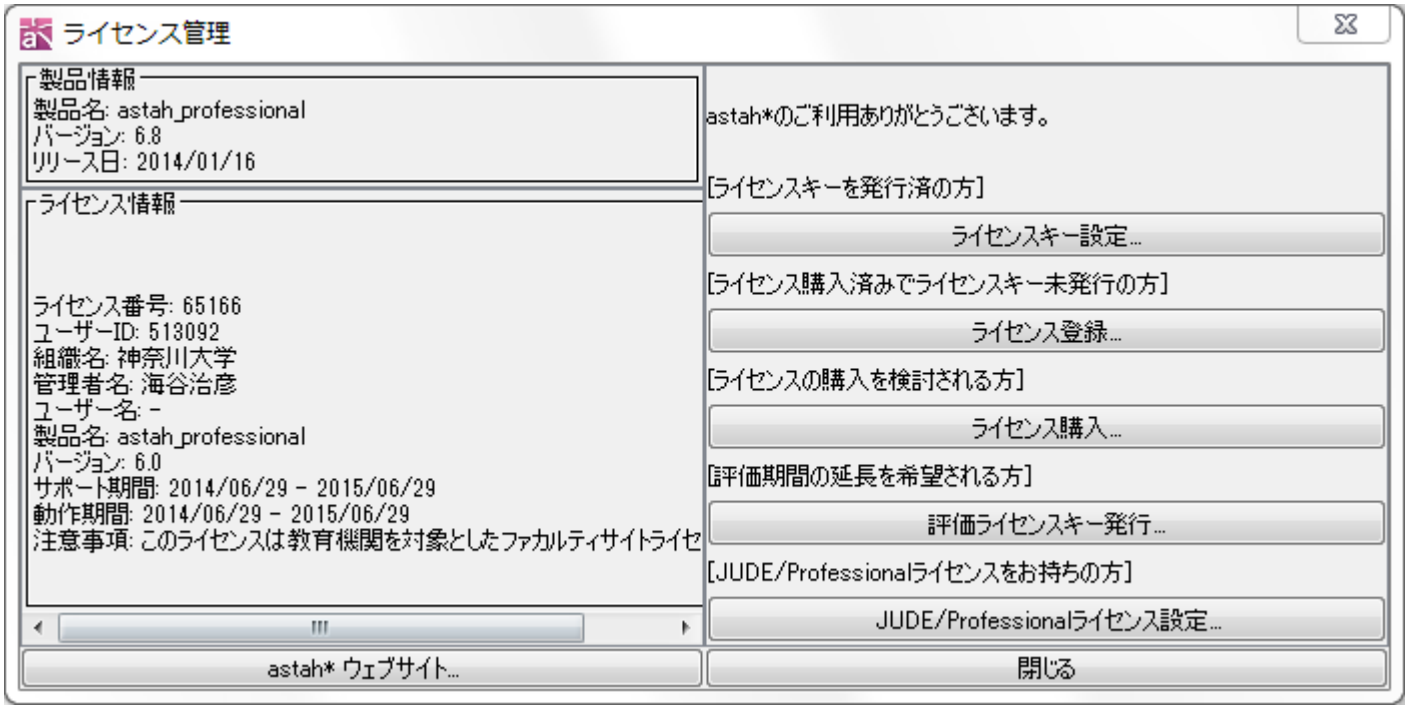
- astah モデリングツールをインストールしておいてください.
- 詳細な手順やデータ，URLは授業のページからリンクがありますが，次頁に概要を提示します.
- ライセンスファイル等はWebClassにおいてあります.

手順

1. WebClassからライセンスファイルをダウンロード(XML形式)
2. プログラムをダウンロード.
   - astah説明のページにリンクがあります.
3. プログラムを管理者権限でインストール.
4. 初回実行で, プログラムにライセンスファイルを与える. この際も管理者権限で実行が必要.
5. とりあえず, 図 > クラス図から, 適当に画をかいてみてください. (余力があれば)

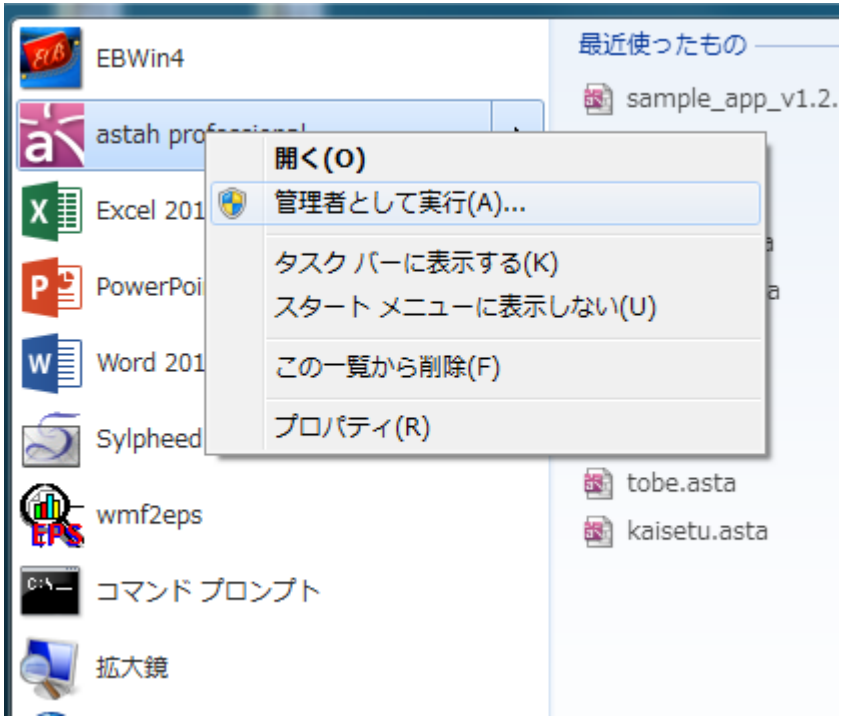
ライセンスファイルの付与

- ヘルプ > ライセンス設定 より，
- ライセンスキー設定ボタンを押して，ダウンロードしたファイルを選択.



管理者として実行

- メニューから，右クリックで以下のメニューを出す．
- Win10,11も似たようなものです．



本日は以上