

CAUA シンポジウム 2023  
AI時代のプログラミング  
～10年後の大学教育に向けて～

# 生成AIはソフトウェア工学の問題を 解決できるのか

早稲田大学 理工学術院  
深澤 良彰

2023.11.28  
TKP東京駅カンファレンスセンター

# Agenda

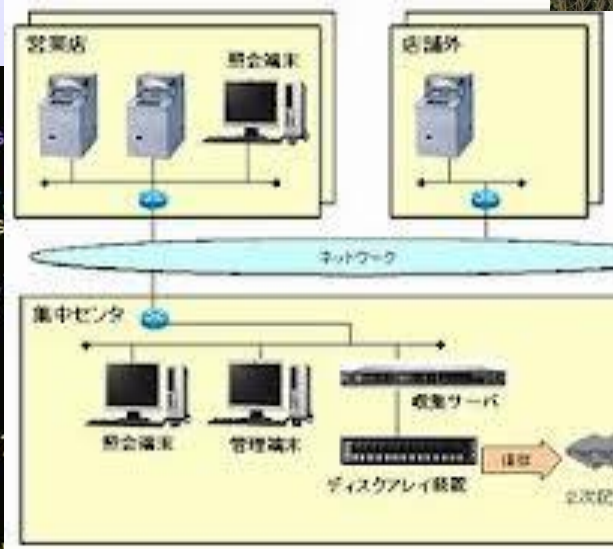
- ・ ソフトウェア工学とは？
- ・ 生成AIによるプログラムコードの自動生成
  - その得意／不得意
- ・ ソフトウェア工学が対象とするようなソフトウェアの開発における生成AI
  - 「普通の」生成AIだと・・・
    - ・ プロンプトエンジニアリングの重要性
  - いわゆる「コード生成AI」だと・・・
    - ・ コード生成AIで解決可能なこと
- ・ モデルベースソフトウェア開発と生成AI
- ・ 生成AIによって解決された点と解決されていない点
- ・ おわりに

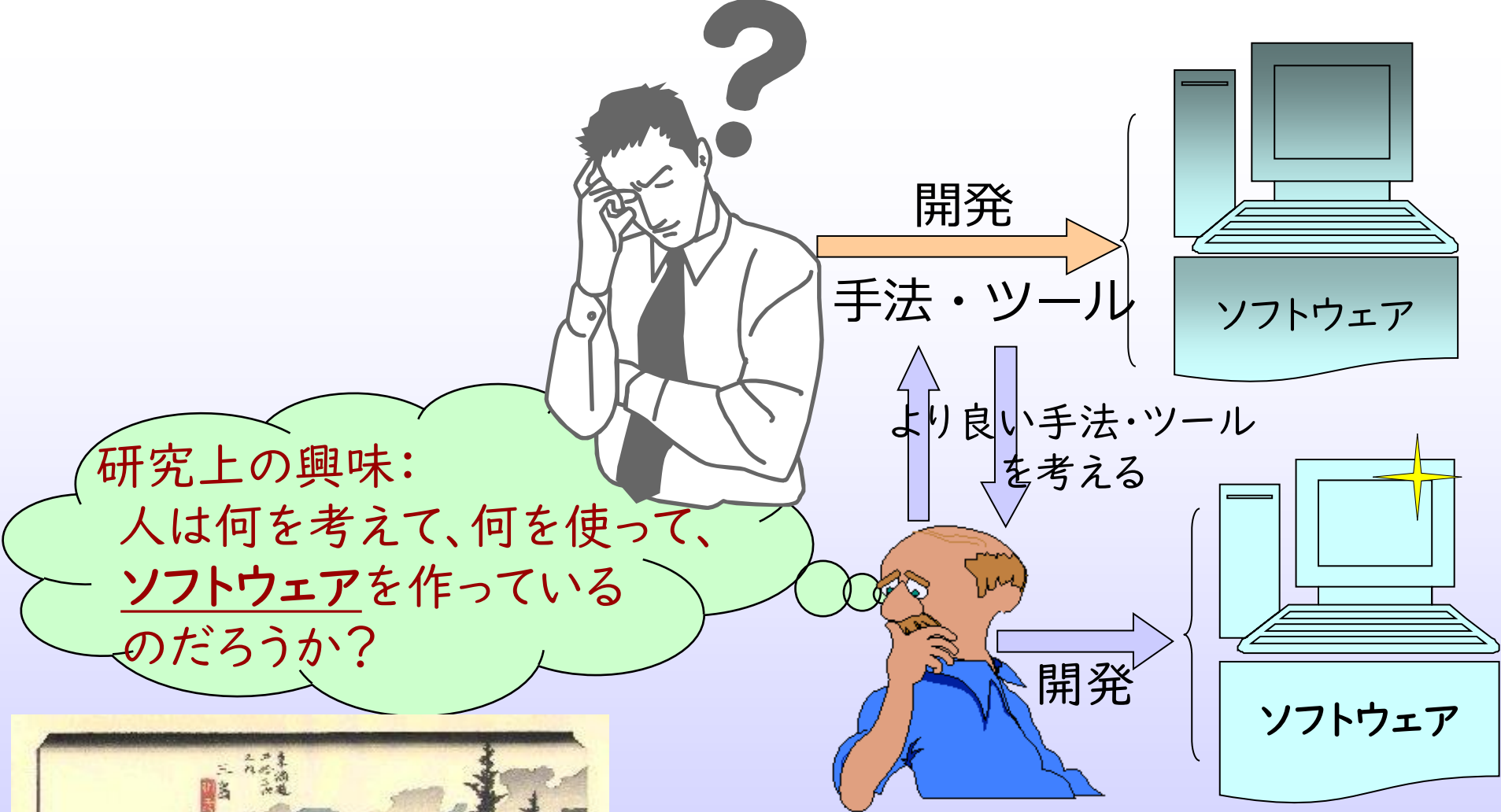
# 私の興味

「人は何を考えて、何を使って、モノをつくっているのだろうか？」



```
@var boolean  
define('PSI_INTERNAL_XML', false)  
if (version_compare("5.2", PHP  
die("PHP 5.2 or greater is  
}  
if (!extension_loaded("pcres"  
die("phpSysInfo requires  
properly.");  
}  
require_once APP_ROOT.'/inc  
// Load configuration  
require_once APP_ROOT.'/co  
defined('PSI_CONFIG_FILE') || defined('PSI_...  
.../templates/html/error_config.html");
```





「駕籠に乗る人、担ぐ人、  
そのまた草鞋を作る人」



# 「ソフトウェア工学 (software engineering)」とは？

- ・ 起源

- 1968年、ガルミッシュ・パルテンキルヘン(当時西ドイツ)で開催された北大西洋条約機構(NATO)の科学委員会で提唱

1968年:メキシコ五輪  
少年ジャンプ創刊  
3億円事件発生

- ・ 当時の状況:ソフトウェアの危機(1960年代末~)

ソフトウェア需要の増大



ソフトウェア技術者の不足



プログラムの質の低下

ソフトウェア技術者の高賃金化

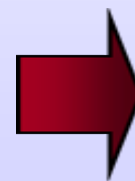
- ・ (私の)ソフトウェア工学の定義

良いソフトウェアを、簡単に作成するための  
方法論、ツールを開発する学問(工学)

# 生成AI登場の衝撃

誰にでも上手に  
できるかとい  
うと、それは違  
う

- ・ 生成AIができること(できると言われていること)
  - 文章の生成(レポートや感想文などの作成)
  - 文章の要約
  - 文章の翻訳・校正
  - 文章の改善
  - ブレインストーミング、アイデア出し
  - テストやクイズの作成と採点
  - プログラムコードの作成と検証
  - ゲームの作成
  - 研究の実施(文献や情報の検索、データの分析)
  - 作詞・作曲、画像・映像の生成



私の『夢』でした!



# プログラムコード生成の特質

- ・ 生成AIにおけるコード生成は、文章の生成等と同様に、LLMとTransformerが基礎
  - 生成AIは、所詮は、過去に開発したもの（今回の場合には、プログラムコード）を再利用したもの
- ・ LLM中に大量のプログラムコードが必要
  - GitHubなどの大規模なオープンソースプログラムプラットフォームを利用
  - しかし、このように蓄積されているプログラムには偏りがある



- 生成できるコードにも偏りがある

生成AIにも得意／不得意がある

- ・ 得意分野:世の中で使われているほとんどのソフトウェアは、データベースへのデータの登録／検索／表示／消去システム
- ・ 不得意分野:競技プログラミング
  - 参加者全員に同一の課題を出題し、より早く与えられた要求を満足するプログラムを正確に記述することを競う
  - コンピュータサイエンスや数学の知識を必要とする問題が多出

# 不得意分野

試験	GPT3.5	GPT4.0
AMC 10	36/150 (10 <sup>th</sup> ~19 <sup>th</sup> )	30/150 (6 <sup>th</sup> ~12 <sup>th</sup> )
AMC 12	30/150 (4 <sup>th</sup> ~8 <sup>th</sup> )	60/150 (45 <sup>th</sup> ~66 <sup>th</sup> )
Codeforces Rating	260 (below 5 <sup>th</sup> )	392 (below 5 <sup>th</sup> )
Leetcode (Medium)	8/80	21/80

- AMC 10/12
  - the American Mathematics Competitions for students in grades 10/12 and below
  - 算術、代数、確率など中等学校の数学的な問題解決のテスト
- Codeforces
  - 競技プログラミングコンテストを主催するウェブサイト
- Leetcode
  - GAFAといったIT企業のコーディング面接で使われたり、それと類似する問題を解くことができる学習サイト

幸いにも(?), ソフトウェア工学の対象とは言い難い



単なるプログラムコードはソフトウェア工学の対象ではない

## ソフトウェア工学の対象

ソフトウェア≠プログラム

- ・ 大規模なソフトウェア
- ・ 多人数で開発するソフトウェア
- ・ 開発者と利用者が異なるソフトウェア
- ・ 使い捨てでないソフトウェア

早稲田大学 基幹理工学部 情報理工学科 必修科目「ソフトウェア工学A」  
(深澤良彰、鷺崎弘宜担当) 第1回目の講義資料より



こんなソフトウェアを生成AIで本当に作れるのか？

# …ということで得意分野でやってみる…

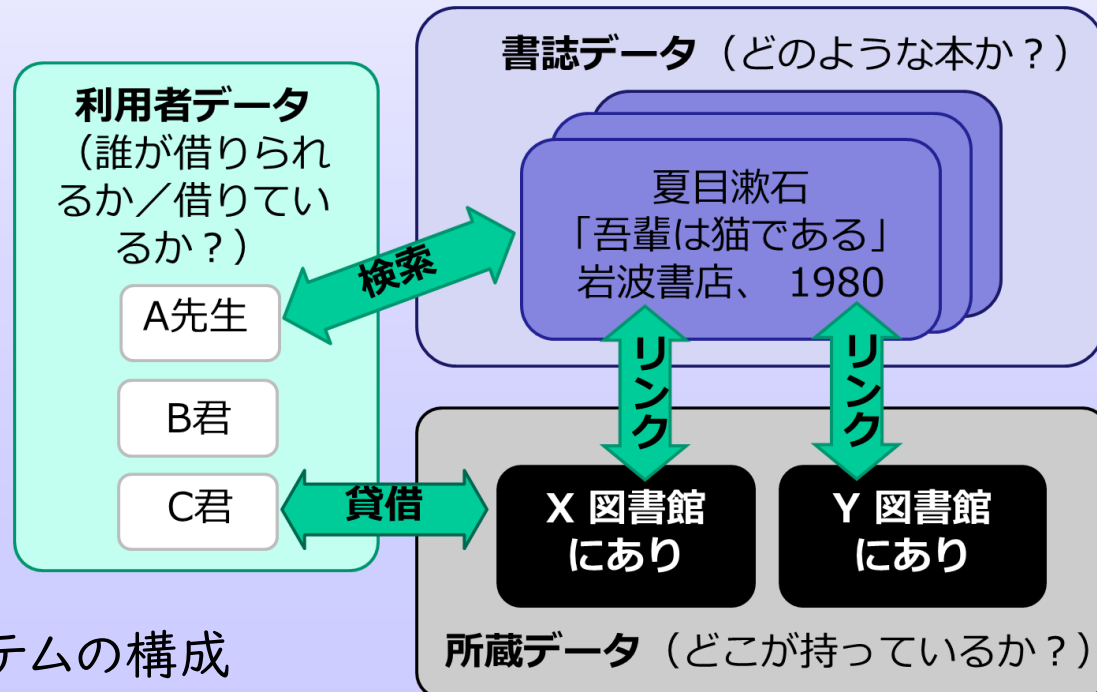
- 生成AIは、所詮は、過去に開発したもの（今回の場合には、プログラム）を再利用したもの



- 世の中で使われているほとんどのソフトウェアは、データベースへのデータの登録／検索／表示／消去システム

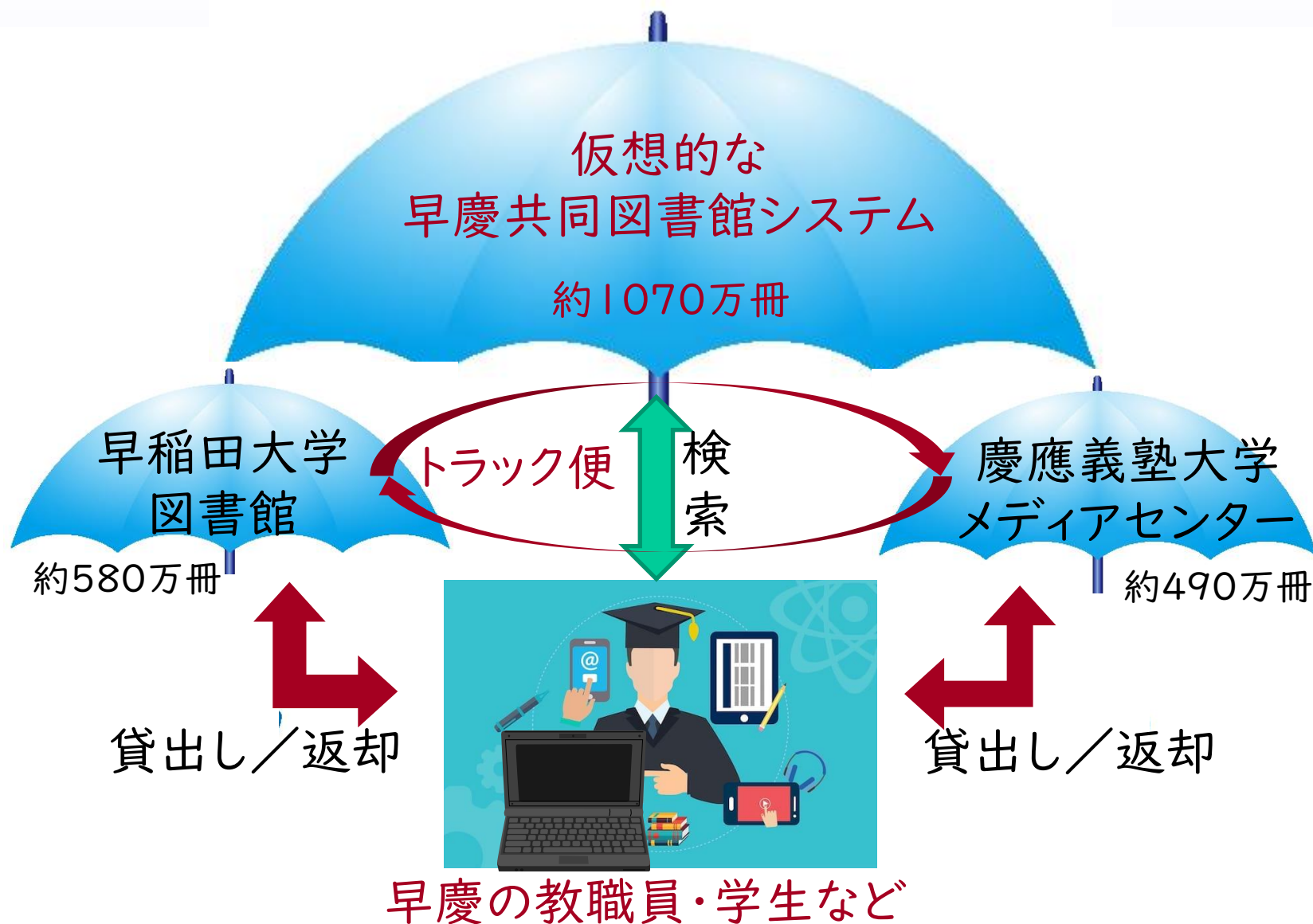


- 例題：Pythonによる図書館システムの開発

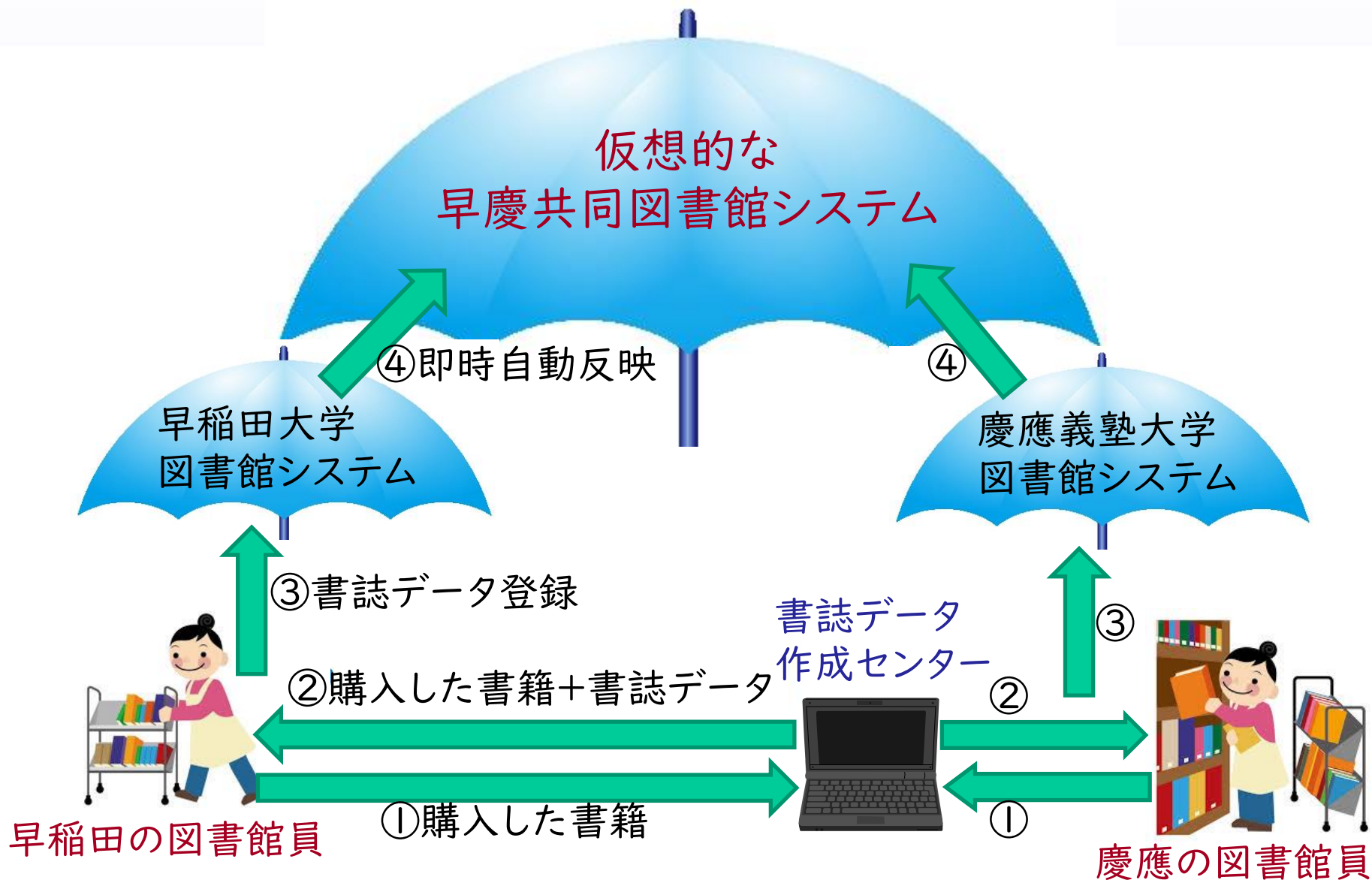


図書館システムの構成

# 早慶図書館システムの構成(利用者目線)



# 早慶図書館システムの構成(図書館員目線)



# Pythonによる図書館システムのChatGPTによる開発

- ・ 「Pythonで図書館のシステムを作ってください」
  - Book class, Patron class, Library classを含む62行のプログラムを生成
  - 『このコード例は、基本的な図書館システムの要素を含んでいます。ただし、実際のシステムでは、データベースやGUI、ユーザー認証、複雑な検索などが必要な場合があります。要件に合わせてコードを拡張して調整してください。』
- ・ 「これに利用者のデータベースを組み込んでください。データベースシステムとしては、SQLを使ってください。なお、利用者は、教職員と学生で分けてください」
  - SQLiteを使用してデータベースを管理し、利用者情報を保存
  - ユーザ情報には、名前とユーザタイプ(studentまたはteacher)が包含
  - 前記のプログラム以外に、58行のプログラムを生成
- ・ 「図書の貸出と返却する機能を付加してください」
  - 前記のプログラム以外に、74行のプログラムを生成



こんな調子で、ソフトウェア工学が対象とするソフトウェアが作れるのか？

# 続き

- ・ 少し専門的なことも聞いてみる

- 「書誌情報はMARC21を用いてください」

- ・ 「MARC21は、図書館などの情報機関で書誌情報を記述・交換するための標準的なフォーマットです。…ここでは、PymarcというPython用のMARC21ライブラリを使用して、MARC21形式の書誌情報を処理する方法を示します。」
    - ・ 前記のプログラム以外に、86行のプログラムを生成

さすが  
ChatGPT!

今の早稲田の  
図書館システム  
は20万行ほど  
のクラウドベ  
ースのプロダクト

↓  
こんな調子では、前述したようなソフトウェアは作れそうにない

規模的にも、機能的にも…

↓  
何が悪いのか？

2000個の  
プロンプト？

どのように  
分担？

↓  
私が悪いのか？ ChatGPTが悪いのか？ 所詮は無理なのか？



# 私が悪いのか？

プロンプトとプロンプトエンジニアリングについての知識がないからである

  
じゃあ、勉強してみよう!

前回のCAUA FORUMで、「  
一歩進んだ生成AI ユーザ  
になるためには？」と聞いたら

- ・ ソフトウェア開発者にとってのプロンプト
  - 生成AIに特定のプログラムを生成させるためのコードブロック、個々のコード行、または開発者が書いた自然言語コメント、統合開発環境 (IDE) 上の指示
- ・ ソフトウェア開発者にとってのプロンプトエンジニアリング
  - プロンプトエンジニアリングは、言語モデル (LMs) を効率的に使用するためのプロンプトを開発および最適化するための新しい学問分野
  - 命令 (プロンプト) の出し方次第で得られるプログラムが異なる  
⇒ 最適なものを入力することが必要
  - プロンプトを使いこなせないと、意図した通りの回答を得られない

# 「Prompt Engineering Guide」

- Introduction
  - LLM設定
  - 基本的なプロンプティング
  - プロンプトの要素
  - プロンプトをデザインする一般的なTips
  - プロンプトの例
- Techniques
  - Zero-shotプロンプティング
  - Few-shotプロンプティング
  - Chain-of-Thoughtプロンプティング
  - 自己整合性 (Self-Consistency)
  - 知識生成プロンプティング
  - Tree of Thoughts
  - Retrieval Augmented Generation
  - Automatic Reasoning and Tool-use
  - 自動プロンプトエンジニア
  - アクティブプロンプト
  - 方向性刺激プロンプティング
  - ReAct
  - マルチモーダルCoT
  - グラフプロンプト (GraphPrompts)
- Applications
  - PAL (プログラム支援言語モデル)
  - データ生成
  - Generating Code
  - Graduate Job Classification Case Study
  - Prompt Function
- Models
  - Flan
  - ChatGPT
  - LLaMA
  - GPT-4
  - Model Collection
- Risks & Misuses
  - 敵対的Prompting
  - 事実性
  - バイアス
- Papers/Tools
- Notebooks/Datasets
- Additional Readings

# プロンプトエンジニアリングの成果 (のいくつか)

- ・ コンテキスト内学習 (in-context learning)
  - 与えられた例文の文脈から学んだ上で新しい文章を生成すること
    - ・ 固有のタスクに応じた訓練やファインチューニングは永続的
    - ・ コンテキスト内学習による学習はログオフまでの一時的
- ・ 思考連鎖 (chain-of-thought, CoT)
  - 生成AIが最終的な答えを出す前に、一連の中間ステップとして問題を解くことを可能にする技術
  - 複数のステップからなる問題に答えるようにモデルを誘導することによって推論能力を向上
- ・ 最小から最大へのプロンプト (least-to-most prompting)
  - ある問題に対する部分問題を列挙し、その後、これらを順番に解くようにモデルに要求
- ・ 自己反復プロンプト (self-refine prompting)
  - 以下を満足できるようになりまで繰り返す
    - ・ 生成AIに問題を解くように要求
    - ・ 次に生成AIにその解答を批評するように要求
    - ・ それを受け取った後、再び問題と解答と批評を考慮して問題を解くように要求

# プロンプティングは最終兵器と聞いていたが...

- ・ 効果的なプロンプトによって、入力者が指定した内容に沿っていることで、利用者の判断も容易に
- ・ 断片的知識であって、思考の手順になっていない
  - 「エンジニアリング」になっていない
- ・ 生成AIにやりたいことを伝えた上で、それを効果的に指示するためのプロンプトをその生成AIに教えてもらうのが、シンプルかつ効果的な方法？



きちんとした要求仕様、設計仕様を書いて、生成AIに入力

ソフトウェア工学的に言うと、  
これができないから困っている


## ChatGPTが悪いのか？

- ・ 一般的な（汎用の）生成AIを用いるのではなくて、ソフトウェア開発向けの生成AIを使うべきでは？



- ・ 代表的なコード生成AI
  - GitHub Copilot
  - OpenAI Codex
- ・ GitHub
  - 3.3億\* リポジトリをホスティング
  - 9,400万人の開発者が利用
  - 400万の組織が利用

## GitHub Copilotとは？

- ・ ソースコード共有プラットフォームGitHubを開発・運営するGitHub社
  - ・ 2021年6月29日：GitHub Copilotのテクニカルレビューの開始を発表[1]
  - ・ 2022年6月21日：一般公開[2]  
ChatGPTの発表よりも前
  - ・ 対応プログラミング言語  
Python、JavaScript、TypeScript、Ruby、Go、C#、C++
  - ・ 対応開発環境  
Visual Studio Code、Visual Studio、Neovim、JetBrains
  - ・ 一言で言えば、
    - 「生成AIを搭載したコーディングアシスタント」
    - 「ペアプログラミングの相棒を生成AIがしてくれる」
- 
- 開発者のワークフローを加速し、開発者が日々のコーディング作業に取り組む方法を革新

[1] <https://github.blog/2021-06-29-introducing-github-copilot-ai-pair-programmer/>

[2] <https://github.blog/2022-06-21-github-copilot-is-generally-available-to-all-developers/>



# ペアプログラミングとは？

目的：短時間でチームのスキルを向上、ソフトウェアの品質も向上



ナビゲーター  
経験豊富な  
ベテラン

指示

生成AIが  
代替

ドライバー  
多くの場合  
初心者

作業環境：

2人分のキーボード・マウスなどを用意  
1台のディスプレイを2人で共有・操作

# GitHub Copilotの使い方

- ・ いわゆる「Auto Complete機能」
  - 従来からIDE等に装備
  - 大規模言語モデル (Large Language Models、LLM) を用いて、関連性の高い、より長いコードを生成

- ・ 「Auto Complete機能」というからには、最初に何かを入力する必要
  - 関数名を考えて入力⇒Copilot がそれに続くコードを提案
  - コメントとして作成したいコードの説明を書き、「function」まで書いて待つ⇒Copilotがコードを提案



- ・ このあとは、
  - 『Alt+]』で次の提案に進む、『Alt+[』で前の提案に戻る
  - 1単語ごとに提案を受入れ『Ctrl + 右矢印キー』、1単語ごとに確定した提案を元に戻す『Ctrl + 左矢印キー』
  - これらの繰り返し

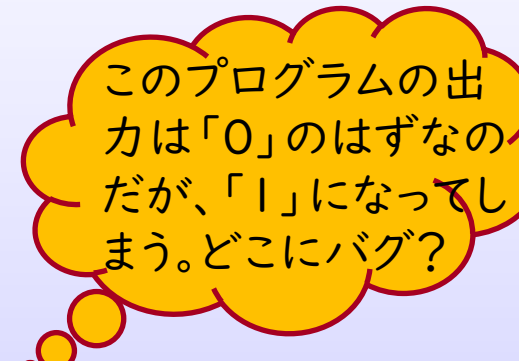
## GitHub Copilotの欠点(1/2)

- GitHub Copilotは、通常の生成AIと同様に、LLMとTransformerが基礎



- 「知りません!」とは言わずに、言語モデルが何かを言ってしまう(幻覚・錯覚(Hallucination))
  - 生成されたプログラムが誤っている可能性
  - テストが必要

- テストもGitHub Copilotが自動生成可能
  - ということは、テストプログラムにも幻覚がある可能性



単体テストは容易だが...

デバッグもしてくれるが...

どんなテストをどの位行えばよいかというソフトウェア工学上の大問題については、未解決

- ツールも存在
  - 例: Machinet

- テスト実行時に変数等の入力が必要な場合には、適切な値を提案

## GitHub Copilotの欠点(2/2)

- ・ 開発者が提供する自然言語のプロンプトが英語でない、あるいは文法的に正しくない場合、GitHub Copilot はうまく機能しない可能性
  - 公開されているソースのコメント等は主に英語であるため
    - ・ 外国製生成AIと同じ
  - 英語を話さない人が使う場合は、サービスの質が低下する可能性
- ・ プロジェクト内の他の場所、さらには同じファイル内で定義された便利な関数を利用できない可能性
  - GitHub Copilot は非常に限られたコンテキストだけを保持
  - 改良の余地
- ・ ライブラリや言語の古い使用法や非推奨の使用法を示唆する可能性



すべて、開発者の責任

## …ここまでの内容って…

小さなプロンプトによって小さなプログラムの作成は可能

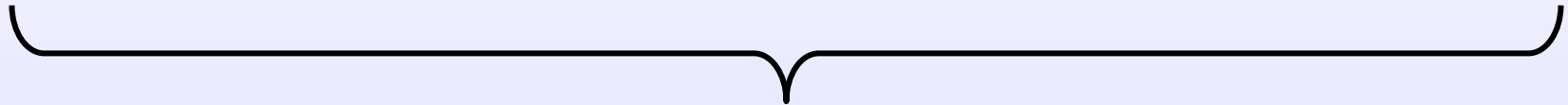


大規模ソフトへの対応は難しい

生成AIをペアプログラミングの優れた相棒にすることができる



ソフトウェア工学の問題としては、プログラミングの問題は全体として小さい



これだけでは、ソフトウェア工学の諸課題が解決したとは言い難い



ソフトウェアの開発・保守のライフサイクルに戻ってみよう



より難しい要求獲得・要求分析は？ 保守は？



再び普通の生成AIに戻ってみる…

# モデルベースソフトウェア開発

## 概念モデル

- 対象業務の世界を構成する概念と概念間の関係を表すモデル

## 要求モデル

- 顧客がシステムに望む事柄を表すモデル

## 分析モデル

- 実装方法を関知せずに、対象業務についてシステム化する方法を表すモデル


## 設計モデル

- システム化する事柄と、その実装方法を詳細に表すモデル


## 実装モデル

- プログラムソースコード、(コンパイル後の)プログラム、プログラムの計算機上の配置


GitHub Copilotなどの  
コード生成AIの守備範囲




座席とは・・・  
予約とは・・・



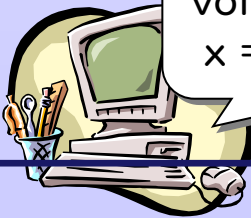
座席をオンラインで予約できるように・・・



画面から希望座席を指定すると  
・・・



CGIでScreenからDB上のSheetをselect()すると



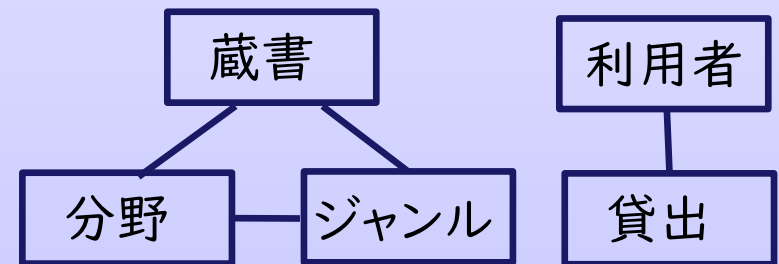
```
class Sheet{  
void select()  
x = y + 1;  
}
```



# 「図書館ドメインを分析し、概念モデルを作成してください」

- ・ 図書館ドメインの概念モデルは、このドメイン内の主要な概念とそれらの関係を視覚的に表現するものです。以下に、図書館ドメインの概念モデルの例を示します。
- ・ 概念モデルの要素:
  - ・ 蔵書 (Item):
    - 属性: タイトル、著者、出版社、出版年、ISBNなど
    - 関連: 分野、ジャンル、著者、出版物
  - ・ 分野 (Category):
    - 属性: 分野名、説明
    - 関連: 蔵書、ジャンル
  - ・ ジャンル (Genre):
    - 属性: ジャンル名、説明
    - 関連: 蔵書、分野
  - ・ 利用者 (User):
    - 属性: ユーザーID、名前、連絡先情報
    - 関連: 貸出、予約、貸出履歴
  - ・ 貸出 (Loan):
    - 属性: 貸出ID、貸出日、返却日
    - 関連: 利用者、蔵書 ...

→ ドメインが明確であれば、図示ができないことを除いて、ドメイン分析はできる  
・ 図示は別なツールで行えばよい



# 「これをもとに要求モデルを作ってください。」

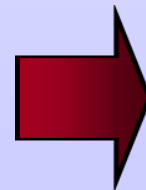
- 理解しやすい要求モデルを作成しましょう。以下に、図書館システムの要求モデルの一部を示します。要求モデルは、システムに対する期待される機能や動作を要約するものです。以下の要求は一部ですが、参考になるでしょう。

## 1. ユーザーアクセス要求:

- ログイン (Login):
  - 利用者はアカウントにログインできる。
- 新規アカウント登録 (User Registration):
  - 新規利用者はアカウントを作成できる。

## 2. 蔵書管理要求:

- 蔵書検索 (Search for Items):
  - 利用者はカタログを検索し、蔵書を見つけられる。
- 蔵書詳細表示 (View Item Details):
  - 利用者は蔵書の詳細情報 (タイトル、著者、説明など) を閲覧できる。
- 蔵書貸出 (Check Out Item):
  - 利用者は蔵書を貸出できる。
- 蔵書返却 (Return Item):
  - 利用者は貸出した蔵書を返却できる。
- 蔵書予約 (Reserve Item):
  - 利用者は蔵書を予約できる。



要求モデルも作れてしまう?

・生成AIが作る要求モデル?  
・問題は、「特徴」をどのように組み込むか

・結局、それはプロンプトで

# 生成AIによって、何が解決し、 何が解決していないのか？(1/2)

- ・ 生成AIの宿命
  - 一般的な質問から、一般的な答えを生成することしかできない
    - ・ それで十分な場合
      - 例:ドメイン分析
    - ・ それでは不十分な場合
      - 例:要求分析



特殊性の反映

- ・ 企業の特特殊性
- ・ システムの特特殊性 (特徴)



## 徹底したプロンプティング

例:既存のシステム設計書、過去に人手で実施したレビューの記録(指摘と修正内容)、その観点などを、生成AIのプロンプトとして入力



チェック項目

- ・ 仕様の抜けや漏れ
- ・ 入力値の範囲の妥当性

# 生成AIによって、何が解決し、 何が解決していないのか？(2/2)

- ・ 「知りません!」とは言わずに、言語モデルが何かを言ってしまう(幻覚・錯覚(Hallucination))
  - AIが生成したものを点検する必要有
    - ・ 命題「Iから人力で作業するのと比較すると、明らかに効率的」の真偽
  - 「点検の際にもAIを活用可能」, 「間違いが見つかったら、それをAIに指摘して修正」
    - ・ でもそこにも幻覚がある可能性
    - ・ AIが生成したものを点検する必要有



・誤りは生成AIの  
責任ではない



・きちんとした評価  
・きちんとした教育

- ・ ソフトウェアの開発についてはともかく、保守については、まだほとんど言及されていない

## ここまでのまとめ

- ・ 生成AIの登場:新たなヒューマン・インタラクションの提供の開始



- ・ 数々の生成AIベースのソフトウェアツールが出現
  - ソフトウェア開発のライフサイクルのどこでどのようなツールを利用するのか?
    - ・ 一般的に使い方は簡単
    - ・ できるだけ利用することにより生産性を上げる努力
    - ・ 後ろに控えている生成AIシステムの利用条件に注意
      - セキュリティ、個人情報保護・・・

個人的危惧:  
日本のソフトウェア  
開発企業が乗り遅  
れていないか?

- ・ 生成AIの本質的問題点については、常に留意
  - 生成AIを使いこなすのに最も重要なものはプロンプト
  - 幻覚は常について回る

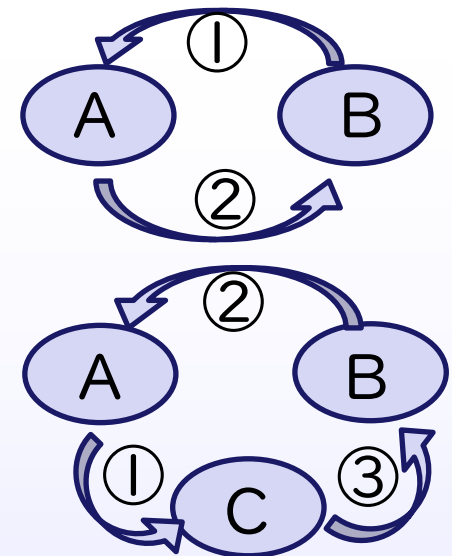


新しいソフトウェア開発時代の夜明けの到来  
ただし、あくまでも「ツール」であることの留意

## 教育の視点から

昇順に並び替えるバブルソートのプログラムをPythonで書いてください。

```
for i in range(n - 1):  
    for j in range(n - i - 1):  
        if arr[j] > arr[j + 1]:  
            arr[j], arr[j + 1] = arr[j + 1], arr[j]
```



素晴らしいが、Pythonだけ

先週の授業で教えた値のスワップの方法を使ってください。

『申し訳ありませんが、私は直近の授業や過去の特定の出来事に関する情報を持っていないため、具体的な先週の授業内容については把握していません。』

```
# 値のスワップ  
temp = arr[j]  
arr[j] = arr[j + 1]  
arr[j + 1] = temp
```



# おわりに

- ・ CAUAの面白さ
  - 出自は伊藤忠テクノソリューションズ(CTC)のユーザ会
  - ほとんどCTCの宣伝になっていない
    - ・ 事務局長のキャラ
      - 私(当時、早稲田大学メディアネットワークセンター所長)との初対面の時:「プログラミング言語Rubyの授業をやりませんか?」
      - 大学ICT推進協議会年次大会:「サーバなんか他社と変わらないので、CAUAのブースを」
- ・ 私事ですが・・・
  - 2019年7月:後藤滋樹先生の後任として3代目CAUA会長拝命
  - 2024年3月:定年(70歳)により早稲田大学退職予定
    - ・ 10回のフォーラム、シンポジウム
    - ・ 今回が最後のシンポジウム



Thank you...



なお、本資料は、深澤自身の独断と偏見に基づいている部分が多く、決して早稲田大学の統一的理解ではありません。

ご質問・ご意見等は  
fukazawa@waseda.jp まで