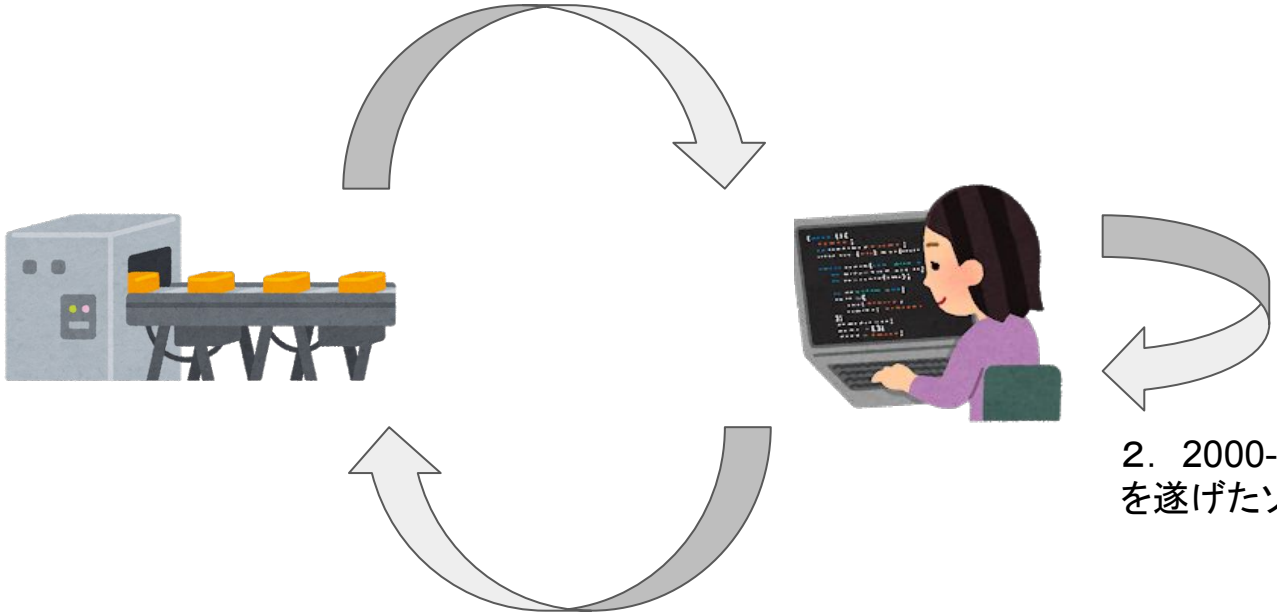
統計数理研究所 オープンハウス連携イベント
データサイエンスが描き出す「モノづくり」の未来シナリオ
～産学連携シンポジウム～

ものづくりとソフトウェア
-- DevOpsとSoftware 2.0

6/17, 2021

丸山宏
花王/東大/PFN
@maruyama on twitter

1．20世紀後半：ものづくりに学んだソフトウェア開発



2．2000-現在：独自の進化を遂げたソフトウェア開発

3．これから：ソフトウェア開発に学ぶものづくり

1．20世紀後半：ものづくりに学んだソフトウェア開発

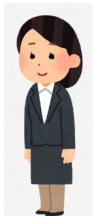
ものづくりの大前提：設計・生産と利用の分離

設計

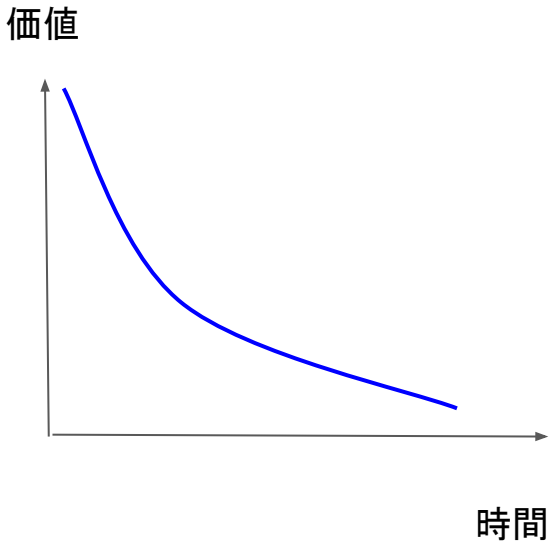
生産

開発・製造と
利用の完全な
分離

利用

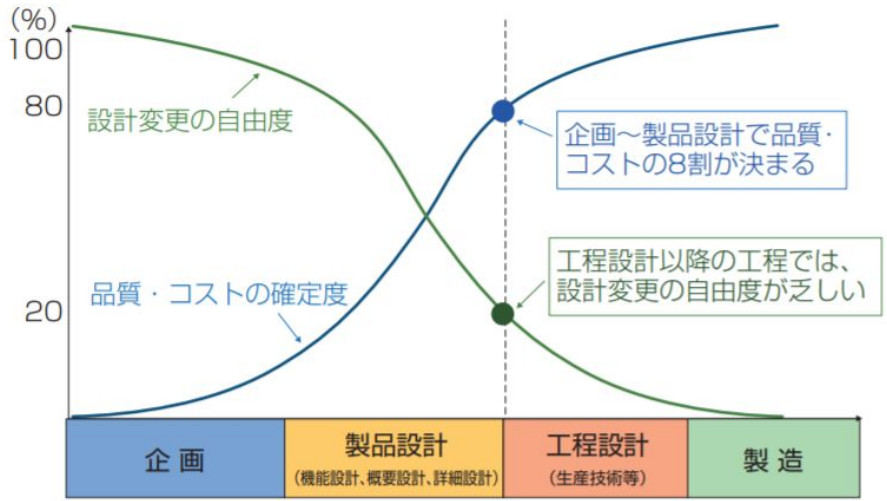


出荷してからの手戻りはほぼ不可能



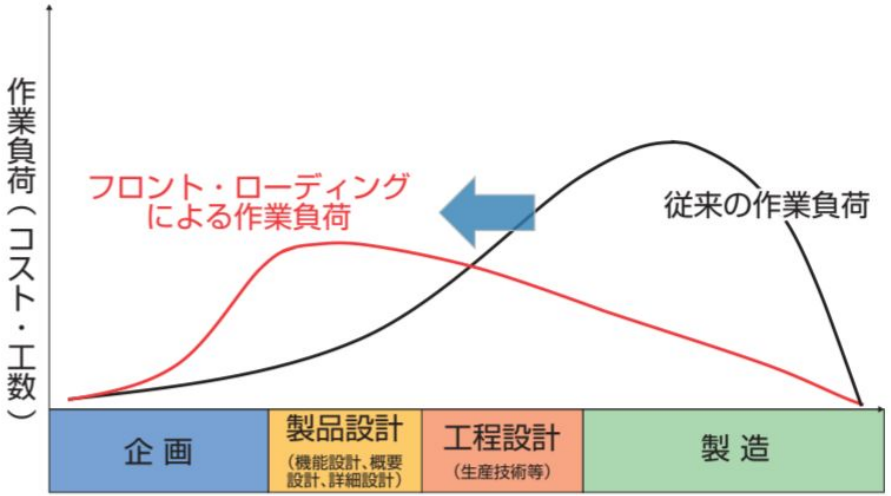
設計・生産においても手戻りは高コスト → フロントローディング

図 132-1　仕様変更の自由度と品質・コストの確定度



資料：日野三十四「実践　エンジニアリング・チェーン・マネジメント：IoT で設計開発革新」P.14 図 0-4 を参考に、経済産業省作成

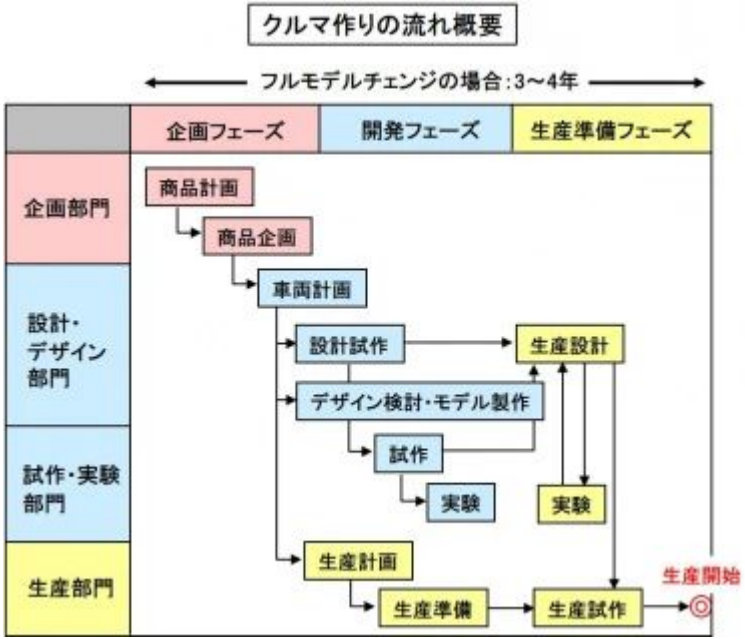
図 132-2　フロントローディングによる作業負荷の軽減



資料：日野三十四「実践　エンジニアリング・チェーン・マネジメント：IoT で設計開発革新」P.14 図 0-4 を参考に、経済産業省作成

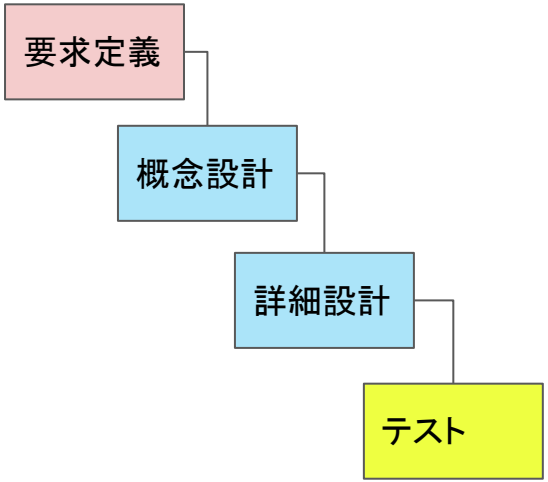
出典：経済産業省 2020年版ものづくり白書p73
https://www.meti.go.jp/report/whitepaper/mono/2020/honbun_pdf/index.html

ものづくりに学んだ20世紀のソフトウェア開発



出典：https://clicccar.com/2020/03/07/958117/

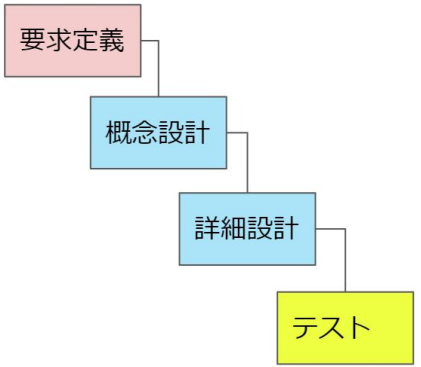
ウォーターフォール開発モデル



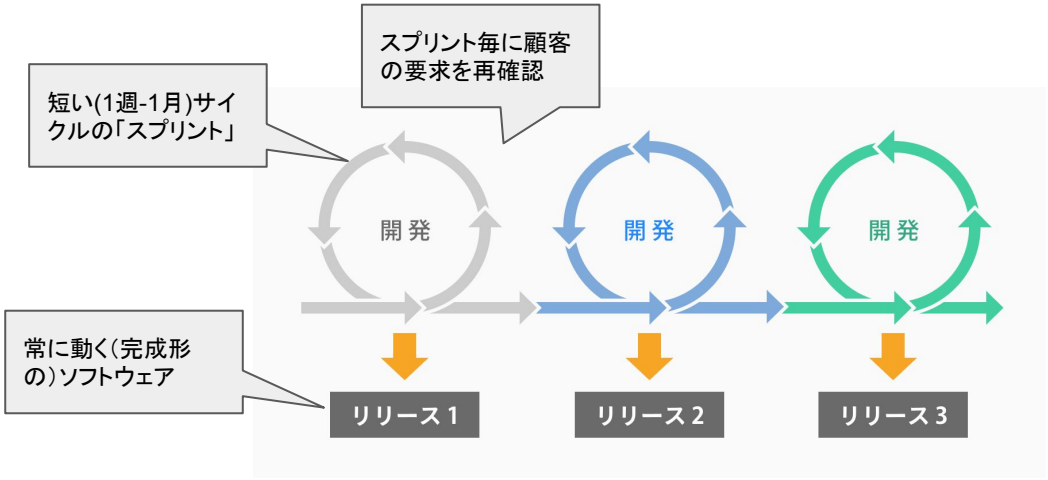
できるだけ手戻りを発生させない！

ソフトウェア開発における不確実性

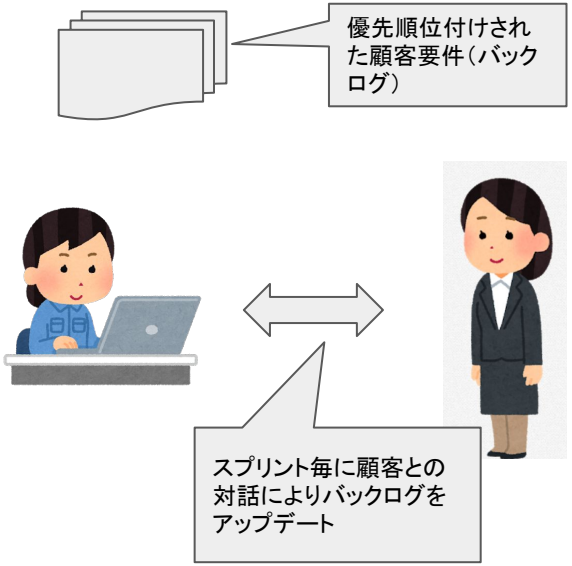
- 要求定義は顧客の真の期待を表現しているか
  - 各ステークホルダの合意ができているか
- 開発の各段階の機能は、上位の要求を正しく反映しているか（バグはないか）
- できあがったシステムは非機能要求を満たすか（性能・品質・使いやすさ・セキュリティ・メンテナンスのしやすさなど）
- 開発にかかる工数の見積もりは正しいか（デスマーチに依存していないか）
- システムは事前に想定された通りの環境で運用されるか
- 将来起きる可能性のある新たな要求は何か
- ：



不確実性に対するソフトウェア工学の叡智：アジャイル開発



出典：https://backlog.com/ja/blog/what-is-agile-and-waterfall/



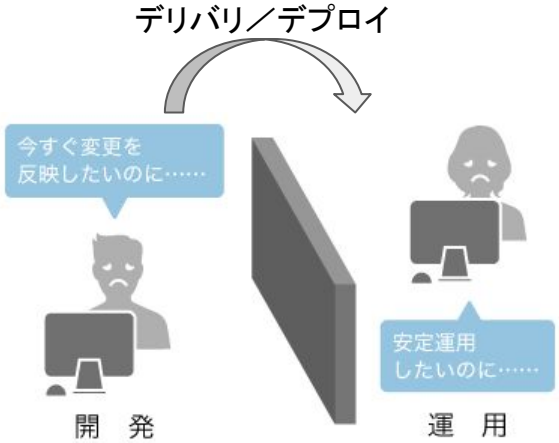
プロセスやツールよりも**個人と対話**を。
包括的なドキュメントよりも**動くソフトウェア**を。
契約交渉よりも**顧客との協調**を。
計画に従うことよりも**変化への対応**を。

アジャイル・マニフェスト (2001)　https://agilemanifesto.org/

QCD*は妥協しないが、機能は優先度に応じてできるものだけを実装する

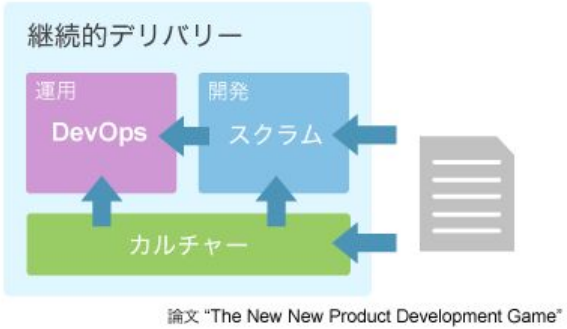
*QCD: Quality(品質), Cost, Delivery(納期)

開発と運用の一体化：DevOps



amazonでは、1時間に1,000回以上リリースされる（2012年当時）

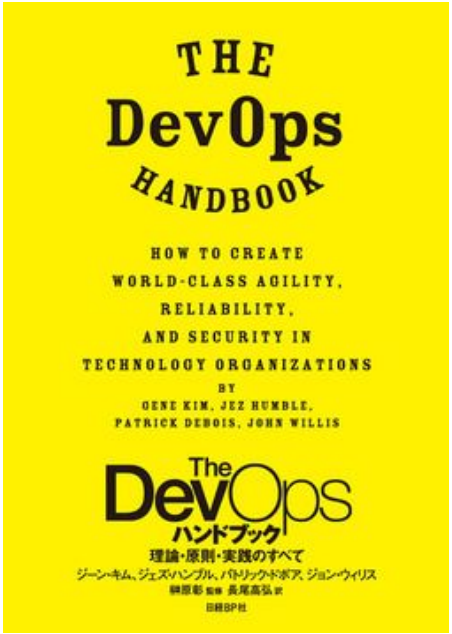
amazon.com　Google



- 継続的インテグレーション
- ソフトウェア定義インフラ
- デリバリ・パイプライン/自動デプロイメント
- カナリアリリース、A/Bテスト
- 意図的な障害の注入
- ：

出典：http://www.atmarkit.co.jp/ait/articles/1307/02/news002.html

DevOpsへの3つの道



ISBN-13 : 978-4822285487

1. フローの原則
   開発 → デプロイ → 顧客体験のパスを高速にする

2. フィードバックの原則
   あらゆるステージで、問題の発生と同時にフィードバックを行う

3. 継続的な学習と実験の原則
   成功と失敗の両方から組織として学習する文化を持つ

リードタイムの短縮！

林南八氏の言葉*

- TPSとは稼働率向上を目指したものではない
- 生産における品質の作り込みとは、問題のあるワークを次工程に流さないこと
  - **自働化**:「止める」「止まる」
  - **JIT**: リードタイムを短縮することで、正常・異常がその場でわかる
- 生産においては「1個流し」が基本

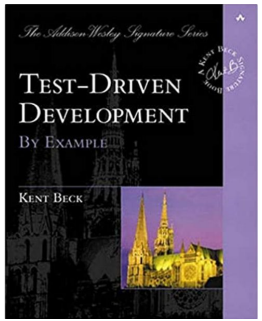
トヨタ生産方式の精神がDevOpsに受け継がれている！

* ビデオ「年末年始期間限定公開:トヨタ生産方式の本質とは？」（ 2021年1月）より

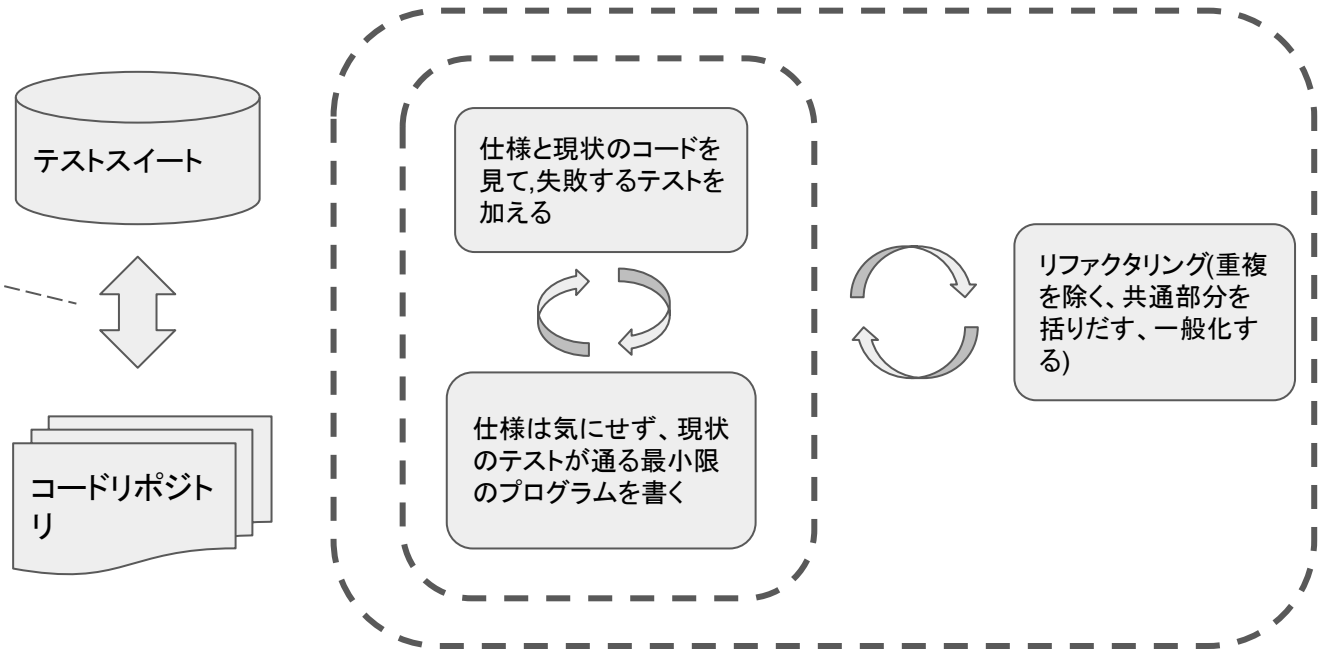
２．独自の進化を遂げたソフトウェア開発

テスト駆動開発（TDD）

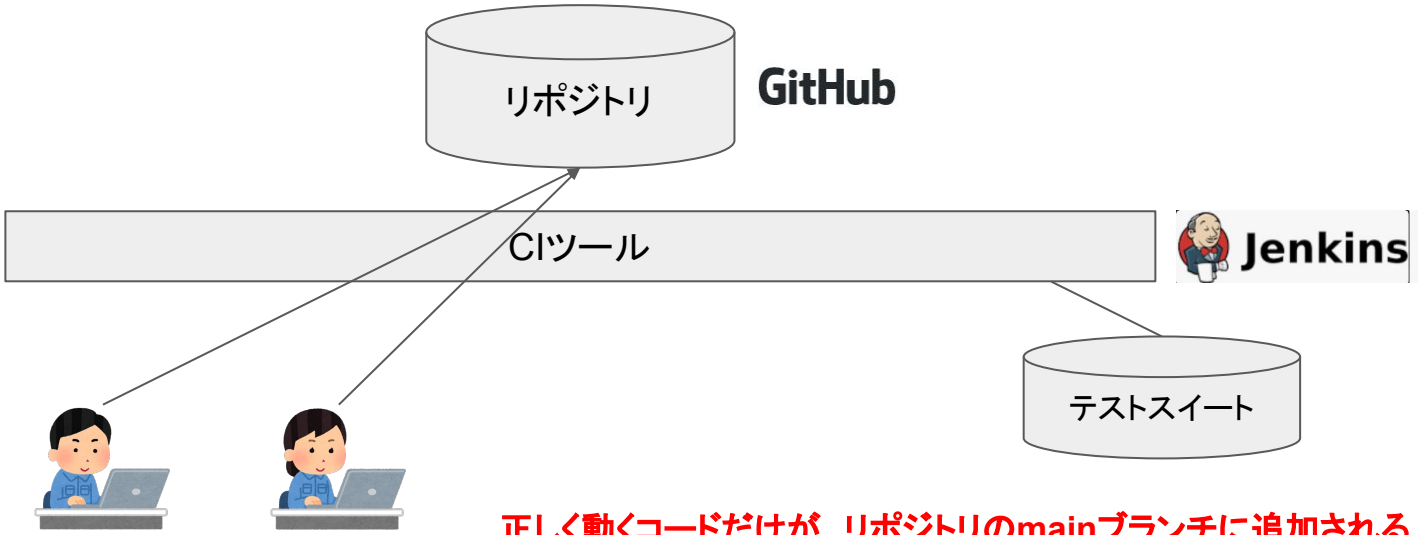
現在のテストスイートに対しては、常にすべてのテストが通る状態を保つ



ISBN-13 : 978-0321146533



継続的インテグレーション（CI）

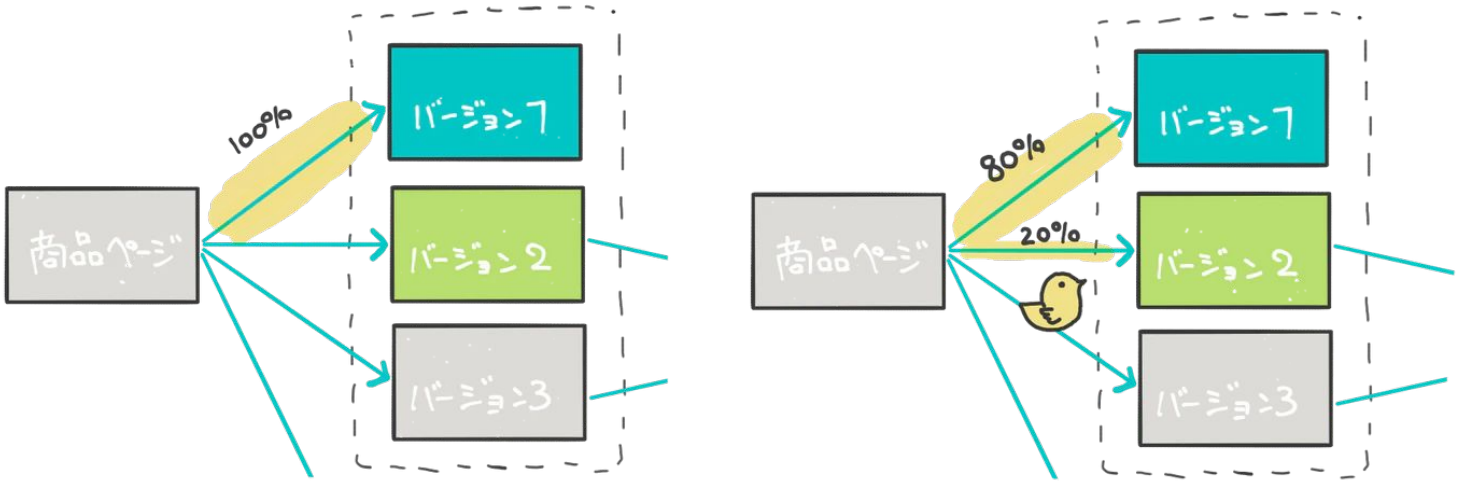


正しく動くコードだけが、リポジトリのmainブランチに追加される

広範に影響がある変更については、各自が他の作業を止めて対応する(cf.「アンドンを引く」、非難無しのポストモーテムを行う)

カナリア・リリース（ダーク・ローンチ）

新しいリリースを、限定されたユーザーに解放することで、影響を最小限にしながらフィードバックを得る



出典：https://qiita.com/Ladicle/items/5a68ff85e15e86781453

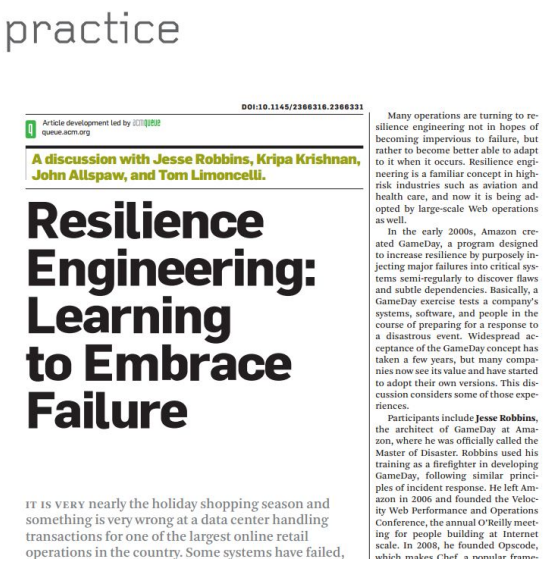
A/B テスティング

ランダム化比較試験をリアルタイムに行うことで、より性能の高いバージョンを探す

出典：https://www.assion.co.jp/blog/guide-of-abtesting/

意図的な障害の注入

“Game Day” – 本番システムでのテスト



practice

DOI:10.1145/2366316.2366331

Article development led by queue.acm.org

A discussion with Jesse Robbins, Kripa Krishnan, John Allspaw, and Tom Limoncelli.

Resilience Engineering: Learning to Embrace Failure

IT IS VERY nearly the holiday shopping season and something is very wrong at a data center handling transactions for one of the largest online retail operations in the country. Some systems have failed,

Many operations are turning to resilience engineering not in hopes of becoming impervious to failure, but rather to become better able to adapt to it when it occurs. Resilience engineering is a familiar concept in high-risk industries such as aviation and health care, and now it is being adopted by large-scale Web operations as well.

In the early 2000s, Amazon created GameDay, a program designed to increase resilience by purposely injecting major failures into critical systems semi-regularly to discover flaws and subtle dependencies. Basically, a GameDay exercise tests a company's systems, software, and people in the course of preparing for a response to a disastrous event. Widespread acceptance of the GameDay concept has taken a few years, but many companies now see its value and have started to adopt their own versions. This discussion considers some of those experiences.

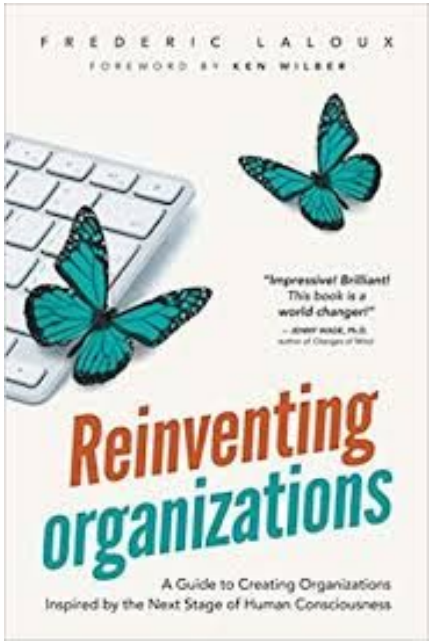
Participants include **Jesse Robbins**, the architect of GameDay at Amazon, where he was officially called the Master of Disaster. Robbins used his training as a firefighter in developing GameDay, following similar principles of incident response. He left Amazon in 2006 and founded the Velocity Web Performance and Operations Conference, the annual O'Reilly meeting for people building at Internet scale. In 2008, he founded Opscode, which makes Chef, a popular frame-

出典: CACM Vol. 55, NO. 11, 2012

Netflixでの “Chaos Monkey”

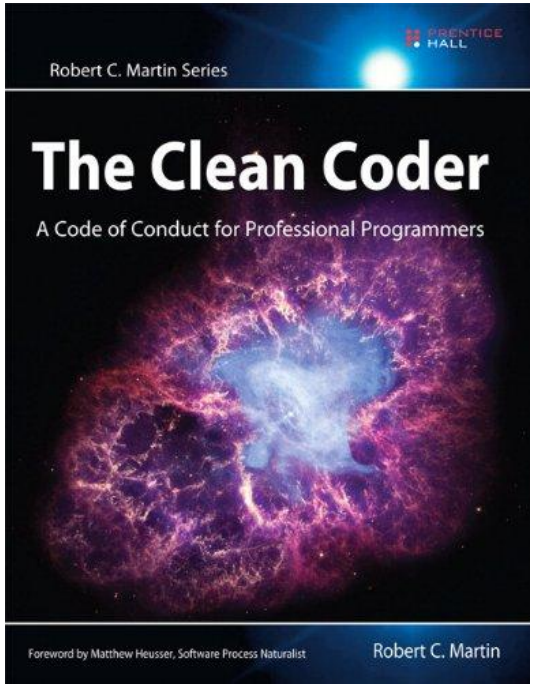
“The best way to avoid failure is to fail constantly”
http://techblog.netflix.com/2010/12/5-lessons-weve-learned-using-aws.html

学習する組織 = TEAL型組織



出典：Frédéric Laloux, *Reinventing Organizations*, 2014.

思想:「生き物」としてのソフトウェア



ISBN-13: 978-0137081073

- ソフトウェアには完成形はない
- プログラムを見たら、書き換えなさい
- 書き換えられないプログラムとは、死んだプログラム

Software 2.0

Andrej Karpathy Nov 12, 2017 · 8 min read

I sometimes see people re as just “another tool in your machine learning toolbox and cons, they work here or there, and sometimes y Kaggle competitions. Unfortunately, this interpreta completely misses the forest for the trees. Neural networks are not just another classifier, they represent the beginning of a fundamental shift in how we write software. They are Software 2.0.

深層学習は、ソフトウェア開発の方法を根本から変えるものだ

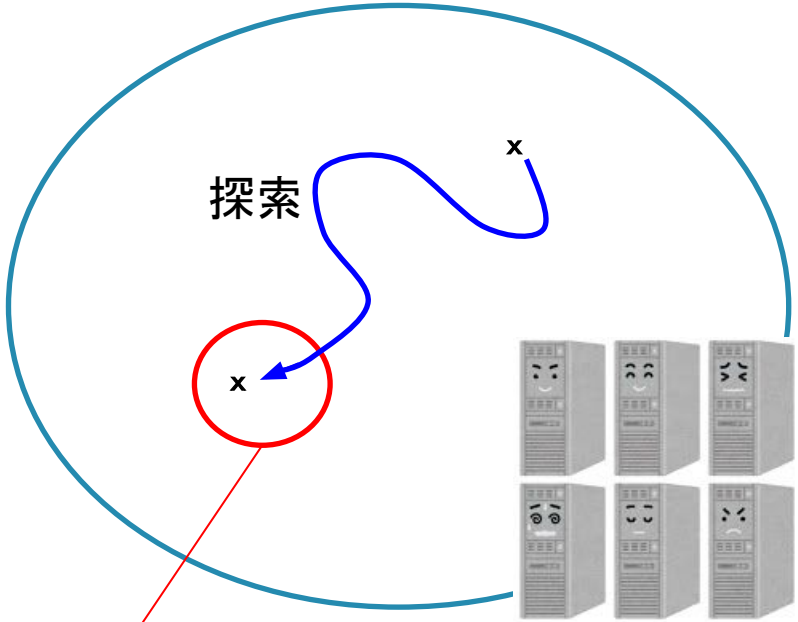
https://medium.com/@karpathy/software-2-0-a64152b37c35

作るプログラム、探すプログラム

作るプログラム

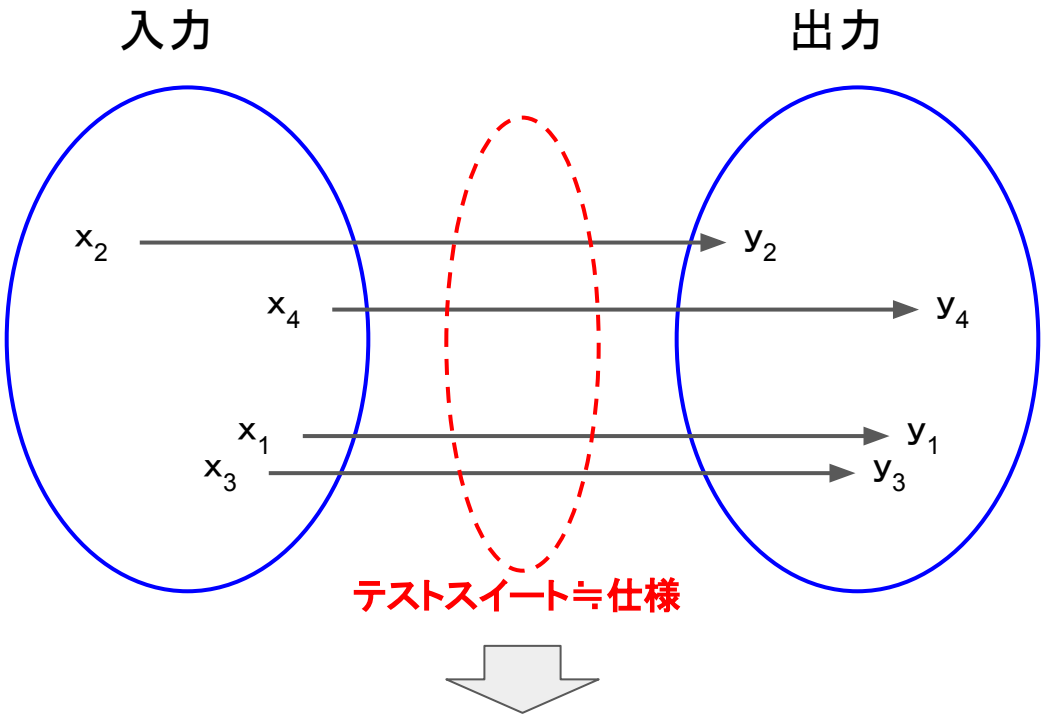
```
def read_hourly_data(pattern, d, h):
    index_cols = patterns[pattern]
    d_str = d.strftime("%Y%m%d")
    d_h = d + pd.Timedelta(' {} hours'.format
    fn = data_dir + "s3/realtime/{}/clipped_
      .format(d_str, d_str, h, pattern[0:5])
    print("Reading {}...".format(fn))
    if os.path.exists(fn):
        with open(fn, 'rb') as f:
            hourly = pd.read_csv(fn, usecols
            hourly = hourly.set_index(index_
            hourly.rename(columns={"populati
    else:
        hourly = pd.DataFrame(columns=index_
        hourly.set_index(index_cols,
        hourly.rename(colu
    return hourly
```

探すプログラム



仕様Sを満たすプログラムたち

テスト駆動開発とは仕様をテストケースで近似する営み



自動デバッグ（Automatic Program Repair）

COMMUNICATIONS OF THE ACM

HOME | CURRENT ISSUE | NEWS | BLOGS | OPINION | RESEARCH | PRA

Home / Magazine Archive / December 2019 (Vol. 62, No. 12) / Automated Program Repair / Full Text

REVIEW ARTICLES

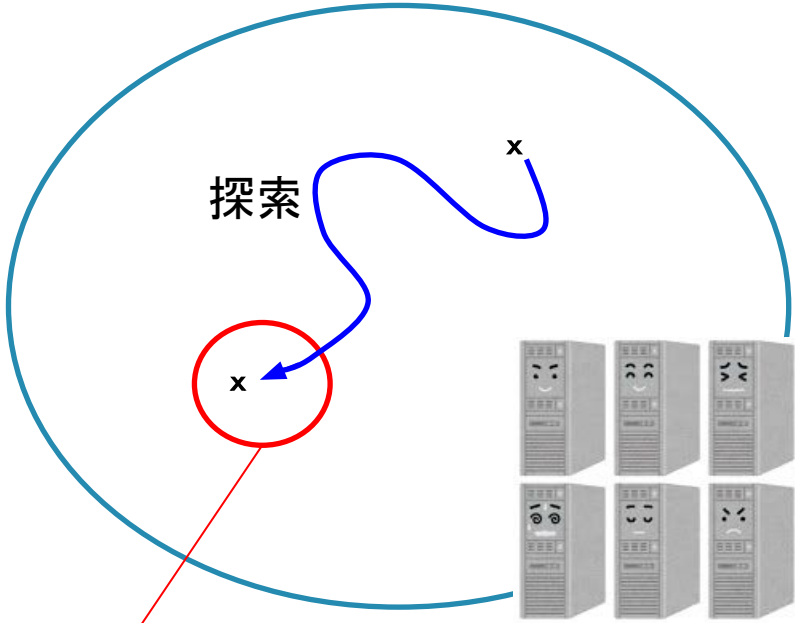
Automated Program Repair

By Claire Le Goues, Michael Pradel, Abhik Roychoudhury
Communications of the ACM, December 2019, Vol. 62 No. 12, Pages 56-65
10.1145/3318162
Comments

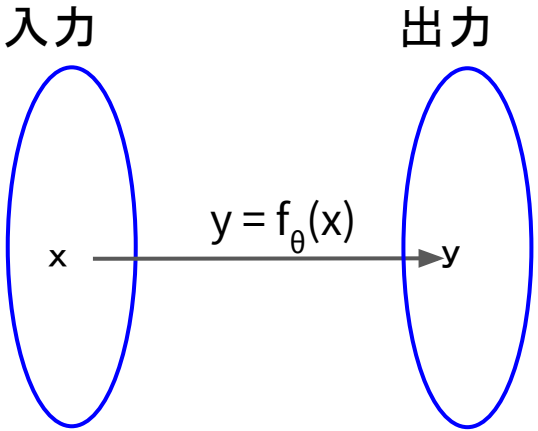
VIEW AS: SHARE:

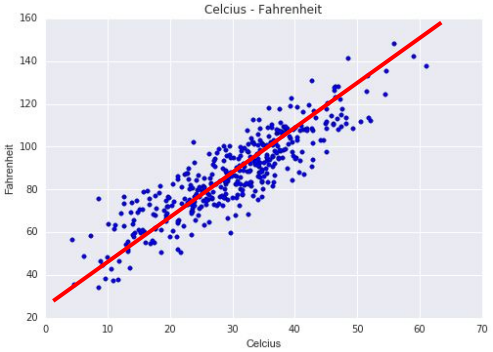
Alex is a software developer, a recent hire at the company of her dreams. She is finally ready to push a highly anticipated new feature to the shared code repository, an important milestone in her career as a developer. As is increasingly common in development practice, this kind of push triggers myriads of tests

https://cacm.acm.org/magazines/2019/12/241055-automated-program-repair/fulltext



線形回帰をプログラミングと見ると…





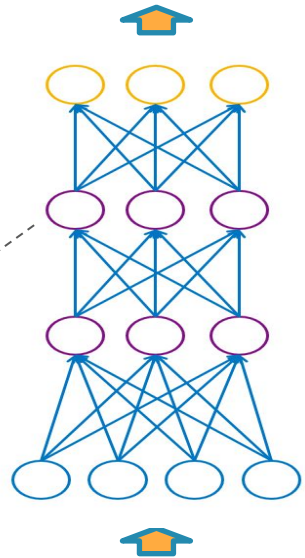
- f_θ は線形な関数族の元
  - $f_\theta(x) = a*x + b$
  - <a, b>がパラメタθ、これを求めることが、プログラミングに相当

- 入出力例の集合を訓練データTとする
- 仕様を命題Sの代わりに、誤差Lで与える
  - L(T, θ) -- テストデータに対する誤差

L(T,θ)を最小化するθを効率よく計算する方法（最小二乗法）がある！

汎用計算機構としての深層ニューラルネット

出力（超多次元）



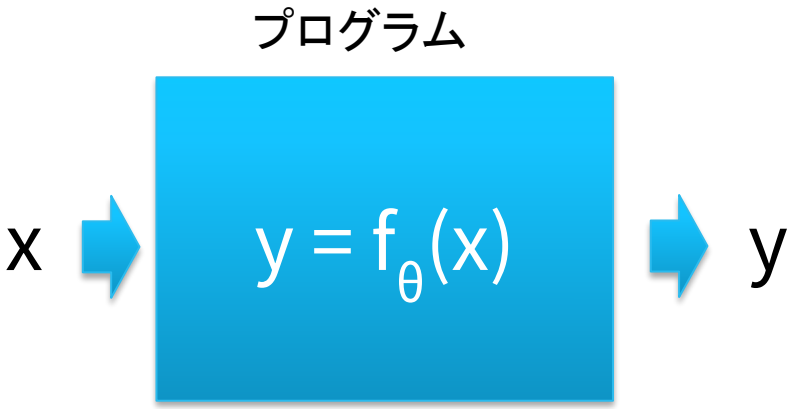
活性化関数が非線形性を表現

入力（超多次元）

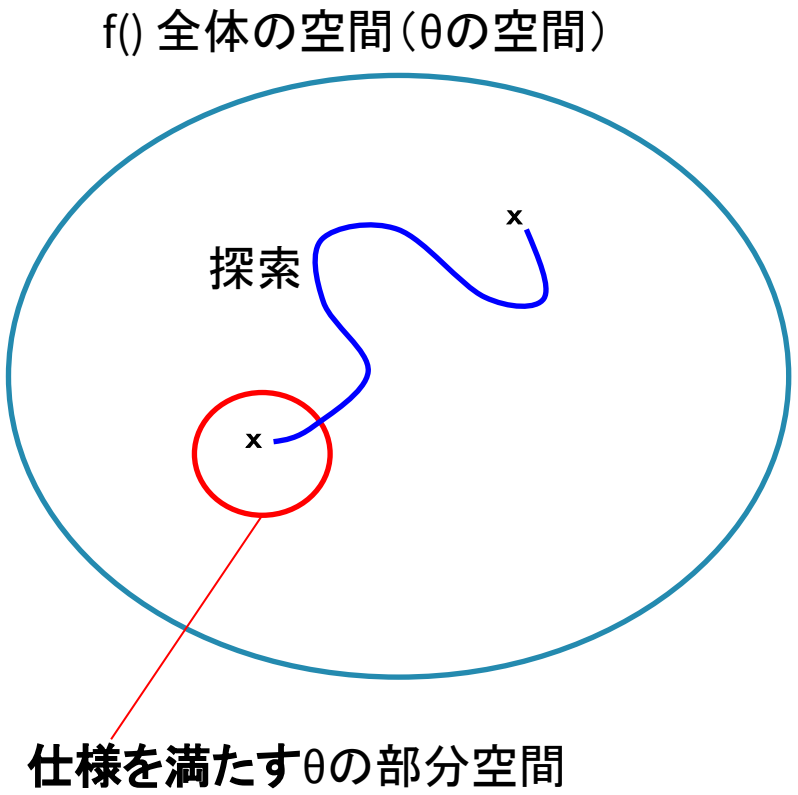
- 勾配法でパラメタを最適化できる
- 桁違いに多いパラメタ
  - 任意の多次元非線形関数を近似*
  - **疑似的にチューリング完全！**

* G. Cybenko. Approximations by superpositions of sigmoidal functions. *Mathematics of Control, Signals, and Systems*, 2(4):303–314, 1989.

探すプログラミングとは何か – パラメタの探索

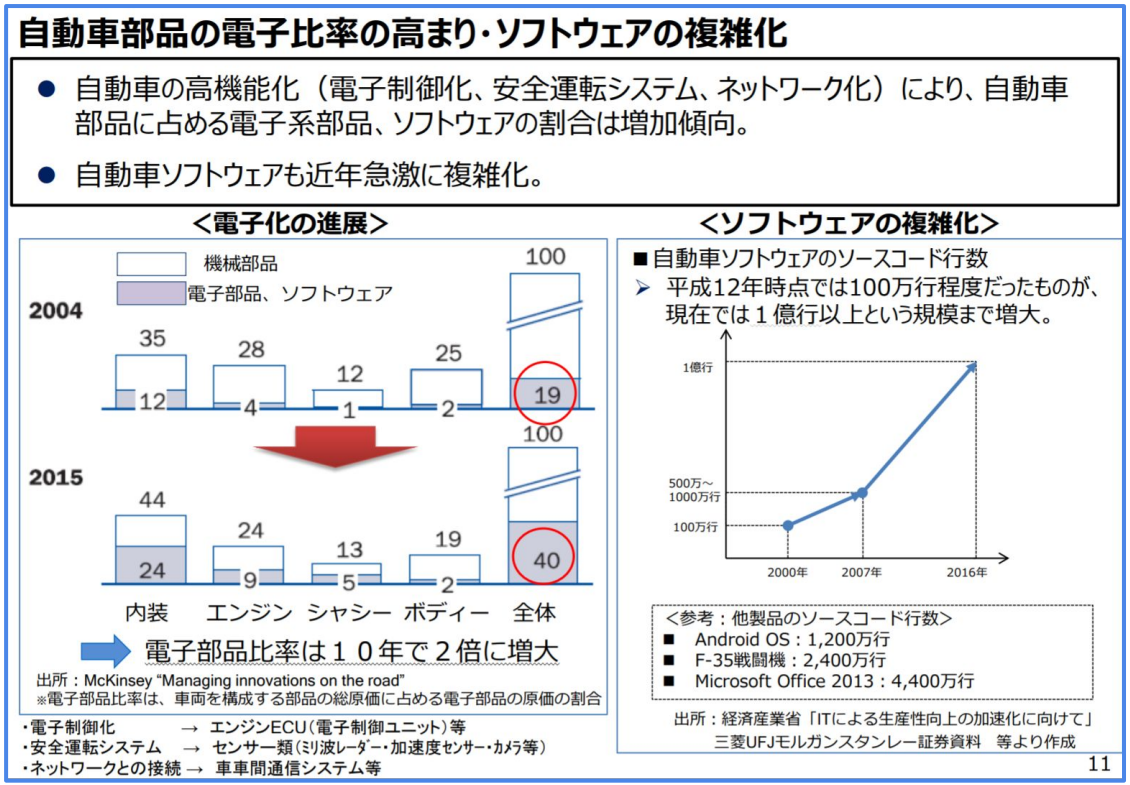


fは (パラメトリックな) 関数族
- f がCプログラムの全体
- f が線形 : Y = Ax + B
- fは深層ニューラルネット



３．ソフトウェア開発に学ぶものづくり

ものづくりへの回帰：ソフトウェアの比重が高まるものづくり



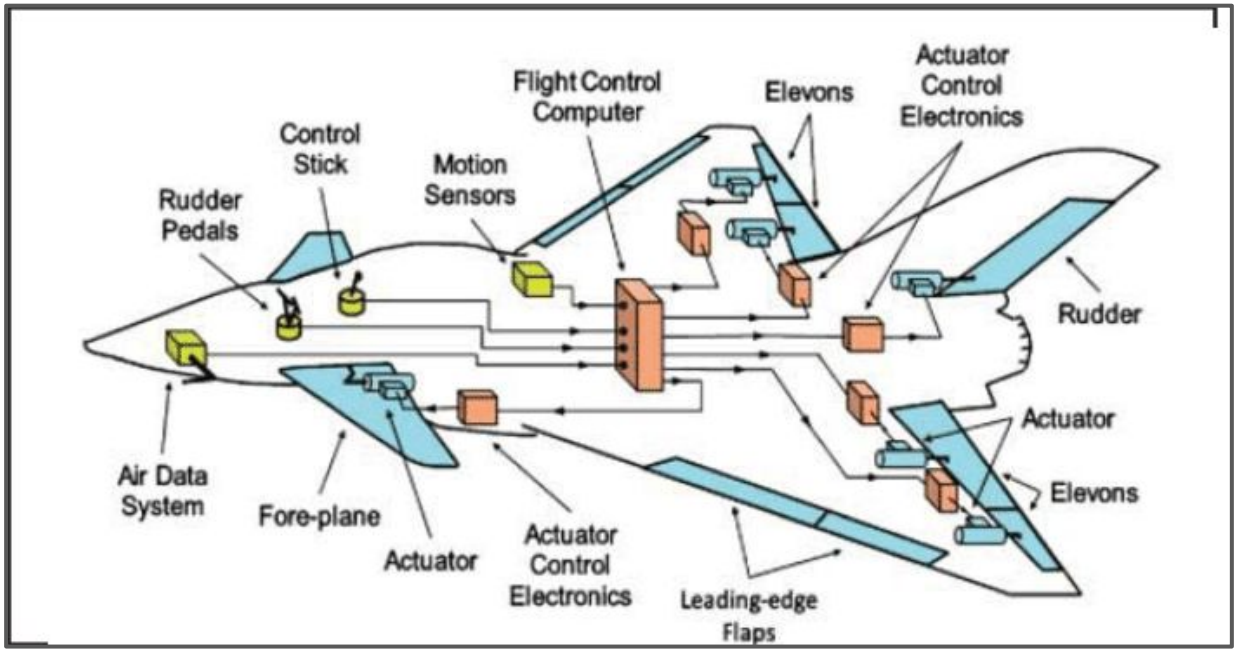
出典：経済産業省 自動車新時代戦略会議（第1回）-配布資料
https://www.meti.go.jp/committee/kenkyukai/seizou/jidousha_shinjidai/001_haifu.html

ソフトウェア更新による、出荷後の価値向上

iOS 14の画面

出典: CNET https://japan.cnet.com/article/35143361/

DevOpsを可能にするものづくり：ソフトウェアによるデカップリング

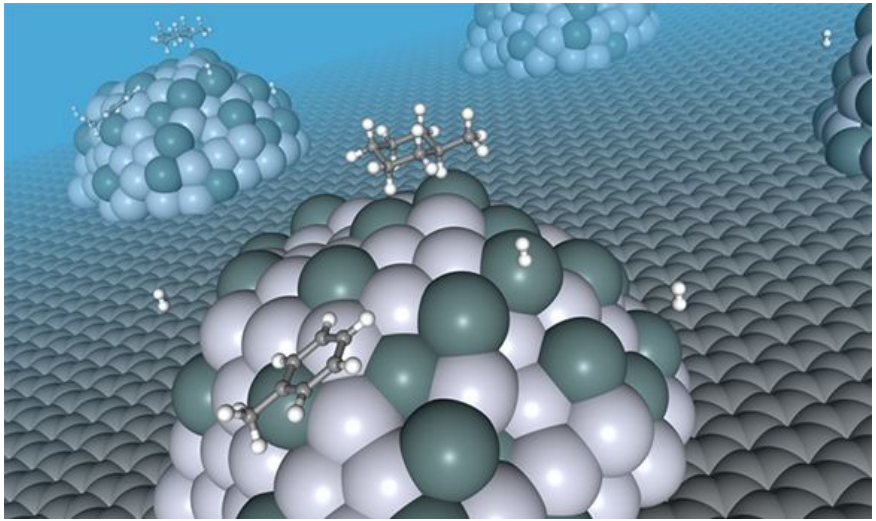


出典: Rezawana Islam, Evolution of aircraft flight control system and fly-by-light flight control system, 2013.

ソフトウェアの変更により、異なる性能の機体にすることができる

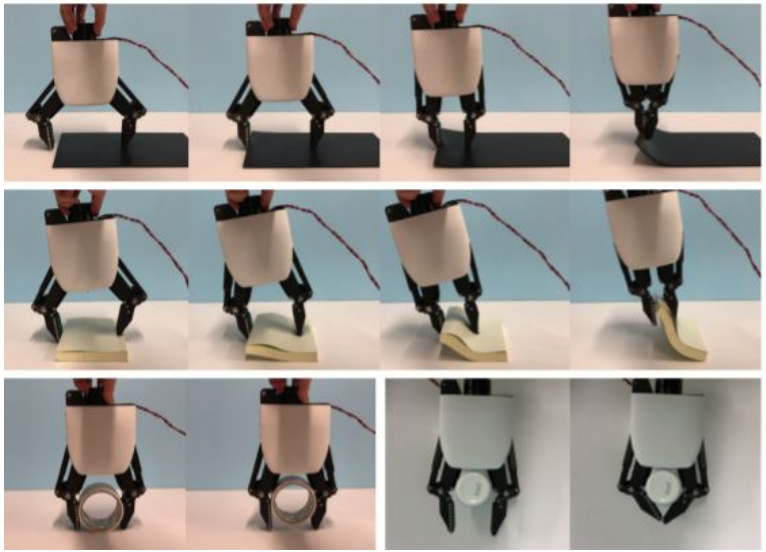
ものづくりにおける「探索による設計」

汎用原子レベルシミュレータによる材料探索



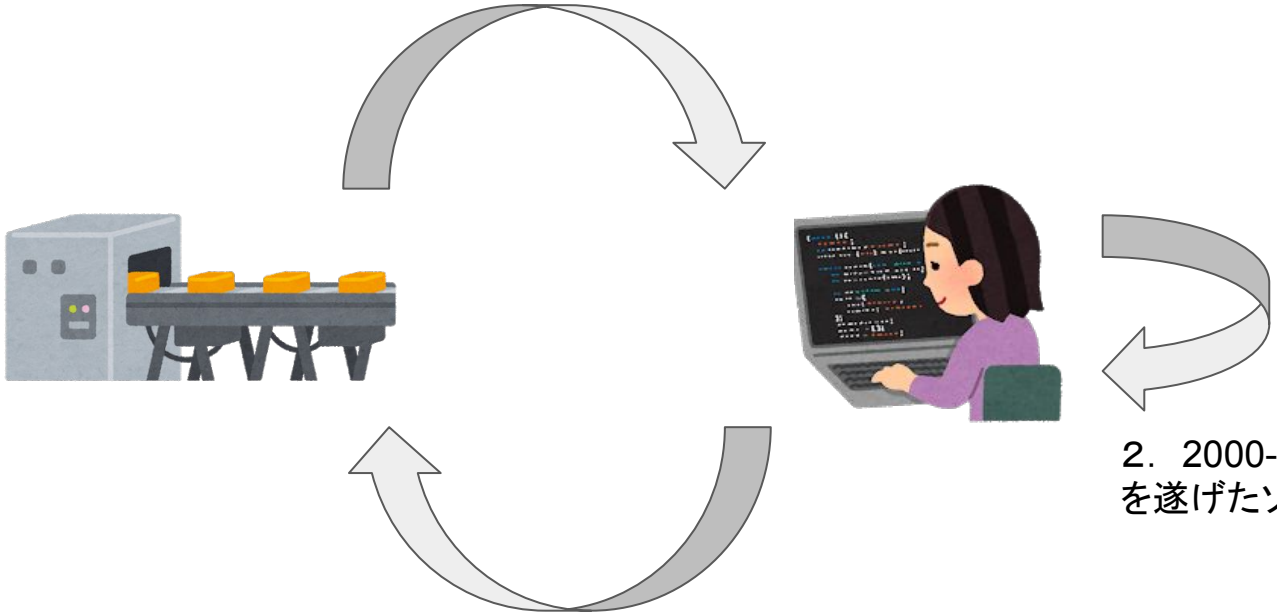
出典：https://www.preferred.jp/ja/news/pr20210427/

ロボットハンドの設計



出典：https://arxiv.org/pdf/2008.05777.pdf

1．20世紀後半：ものづくりに学んだソフトウェア開発



2．2000-現在：独自の進化を遂げたソフトウェア開発

3．これから：ソフトウェア開発に学ぶものづくり

Thank You