ところてんさん曰く: 大きく分けて二つのDXが存在する

レガシーシステムのモダナイズ
既存アナログ業務をデジタルで再設計
開発の内製化ノウハウの蓄積
デジタルな組織への変革
デジタルなプロダクト
市場競争力の確立

モード1のDX
モード2のDX

- モードI、守りのIT、System of Record（SoR）
  - 既存の業務の再設計・合理化、コスト削減、正しく記録を行うシステム
  - 作らなくてはいけないものをちゃんと作る、OAやFA、RPAはこちら
  - 現状はこれができていない企業が大半なので、これができるだけで競争優位
- モード2、攻めのIT、System of Engagement（SoE）
  - 新規事業を作り出すためのプロセス、デザイン思考
  - アジャイル開発、答えがないものを模索して作り上げていく

https://www.evangelism.jp/resources/business-with-it/

PMBOK第7版の大改訂

体験したことのない製品やサービスを作る方法論

体験したことの無い製品・サービスを作るには？

- 要件定義の存在しない世界
  - 「顧客に言われたとおり作る」ことが不可能
  - 顧客も「欲しいものが分からない」から
- 単純な聞き方では絶対に分からない
  - （例）自動運転車の中で何がしたい？
    - ユーザーはそもそも自動運転車の車中を創造できない
  - （例） AIをどうやって活用しますか？
    - AIを使ったことが無い人が分かるはずがない

ITpro http://itpro.nikkeibp.co.jp/

体験したことの無い体験を作るには？

- プロトタイプをとにかく早く作る
- それを顧客にテストしてもらう
- 顧客のフィードバックを得る
  - 実物を体験すれば、さすがに何かは言える
- 良くない点を製品を改善する
- 失敗を改善するサイクルを繰り返す
- つまり「リーンスタートアップ」
  - 実現手法が、アジャイル開発やデザイン思考

https://www.slideshare.net/atsnakada/ss-80520040

https://www.slideshare.net/atsnakada/ss-80520040

デザイン思考＋アジャイル開発

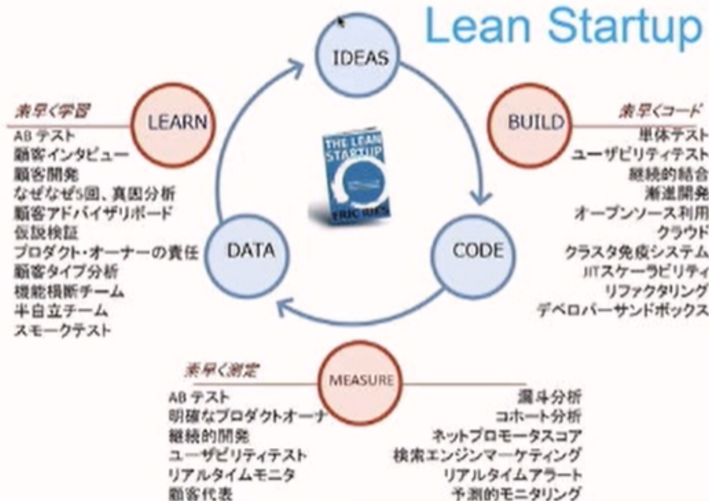
シリコンバレーの強みは技術＋デザイン

- シリコンバレー企業の最大の特徴は？
  - リリースした製品は「必ず間違っている」
  - 「リリース後の改善力」こそが強み
    - ユーザーの「痛み」を引き出すためには、質問力を備えたデザイナーが必要＝**デザイン思考**
    - ユーザーの要望に素早く応えられるソフトウエア開発者が必要＝**アジャイル開発**
    - 質問に的確に答えられる「優れたユーザー」
      - シリコンバレー住人は質問慣れしている
- だからシリコンバレーで作りたい

http://itpro.nikkeibp.co.jp/

https://www.slideshare.net/atsnakada/ss-80520040

リーンスタートアップ



http://blogs.itmedia.co.jp/hiranabe/2011/05/lean-startup-introduction-in-japanese-1.html

予測型から適応型へ

決められたものを
決められたときまでに
破綻なく作る

だれも答えがわからないものを
模索しながら作り続ける
しかも答えが刻々と変わっていく

だれも答えがわからないものを作るには、「アジャイル」な開発が有効

ウォーターフォールとアジャイルの品質の違い

鈴木雄介/Yusuke SUZUKI
@yusuke_arclamp

いわゆるWF開発は外部品質（機能仕様）をゴールとし、フェーズ&ゲートによるプロセスを重視する。アジャイル開発では利用時の品質（使って価値があるか）の向上を目的とするので、適応力を維持するために内部品質（仕組み化や自動化）を重視する。

Translate Tweet

11:58 PM · Jul 17, 2021 · Twitter for iPhone

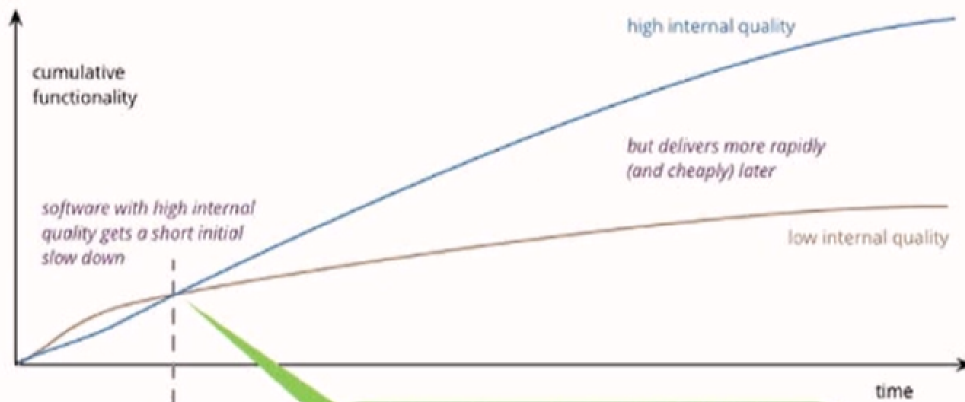
https://twitter.com/yusuke_arclamp/status/1416411959403040771

ソフトウェアの品質を外部指標で特徴づける人は多い。正しいことをする、バグがない、速い、などだ。だが、それらはより深い原因の症状にすぎない。

本書で説明するソフトウェアの品質は内部品質である。内部品質を作り込んだ結果として、外部品質として定義される特性の実現に近づくことができる。内部品質は結果ではなく原因であり、良いソフトウェアが備えているべきものだ。

『レガシーコードからの脱却』p.58

内部品質への投資の損益分岐点は1ヶ月以内に現れる

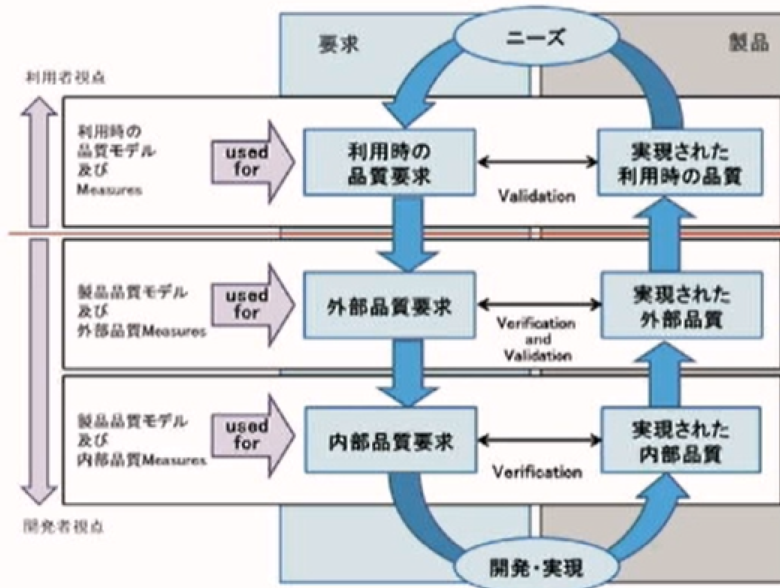


内部品質への投資の損益分岐点は
3年後とかではなく
1ヶ月以内に現れる

https://martinfowler.com/articles/is-quality-worth-cost.html

Takuto Wada が話しています...

SQuaRE 品質モデル: 利用時の品質と製品品質



変更容易性の高いソフトウェアによるアジリティの獲得

ところてん
@tokoroten

なんとなく、DXという言葉に対しての引っかかりが理解できてきた気がする
「デジタル」という言葉に「アジリティ」という意味が含まれていないので、
「変更容易性の高いソフトウェアによるアジリティの獲得」というDXの本質がソフトウェアエンジニア以外には伝わってない感じなんだな

- デジタルを活用したダブルループ学習によるアジリティの獲得
  - デジタルネイティブな組織は、その中に学習し続け、自らを軌道修正していく仕組みを構築（DevOps、スクラム、リーンスタートアップ、）
  - コンピュータシステムを市場環境の変化に合わせて柔軟に変化させていく
  - ITシステムの改善速度を上げていくことで、市場競争力を確保

https://twitter.com/tokoroten/status/1385826623451111424

アジリティを支える品質副特性

- Modularity
- Reusability
- Analysability ≒ Understandability
- Modifiability
- Testability

自動テストの観点で Testability をさらに分解すると

- 実行円滑性（Operability）
  - 自動実行が容易でかつ高速
- 観測容易性（Observability）
  - テスト結果の取得・期待値比較を自動化しやすい
- 制御容易性（Controllability）
  - 事前状態を制御し、テスト対象を自動操作しやすい
- 分解可能性（Decomposability）
  - テスト対象の切り出し、テスト環境への部分置換が容易

https://speakerdeck.com/goyoki/testability-in-embedded-software

Testability を向上させる戦術

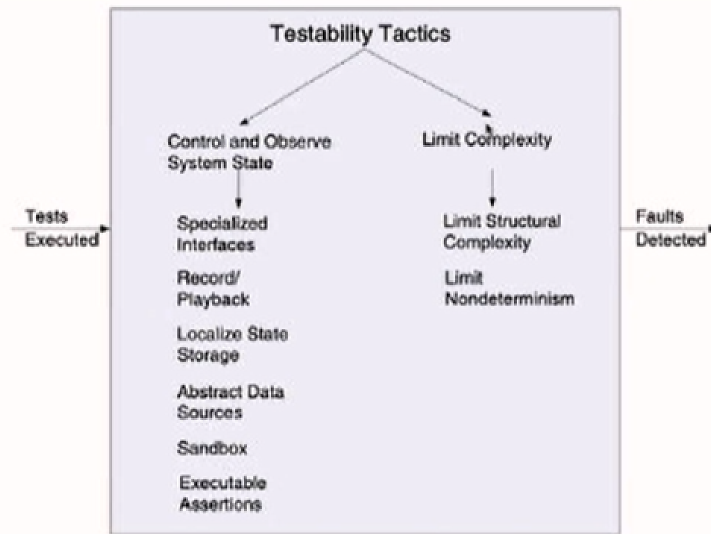
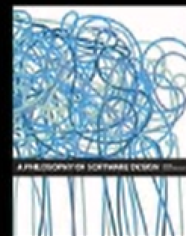


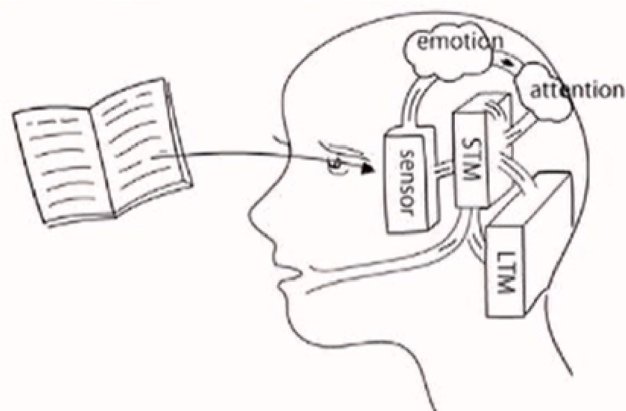
FIGURE 10.4 Testability tactics

Complexity の兆候

- High cognitive load: 高い認知負荷
- Change amplification: 変更の発散
- Unknown unknown: 未知の未知



High cognitive load: 高い認知負荷



感覚記憶
0.5 ~ 2sec

短期記憶
15 ~ 30 sec

長期記憶
死ぬまで?

認知負荷とは、任意のタイミングに短期記憶にかかる心的活動の合計量のこと

Unknown unknown: 知らないこと自体を知らない

There are known knowns; there are things we know that we know.

There are known unknowns; that is to say, there are things that we now know we don't know.

But there are also unknown unknowns – there are things we do not know we don't know.

-Donald Rumsfeld

https://poststatus.com/known-knowns-known-unknowns-and-unknown-unknowns/

Naming も非常に重要

Moriyoshi Koizumi
@moriyoshit

この仕様のせいで8時間くらい溶けた... This is strange logic じゃない、今すぐ直してほしい
github.com/terraform-prov...

```go
func expandSubnetPrivateLinkNetworkPolicy(enabled bool) string {
	// This is strange logic, but to get the schema to make sense for the end user
	// I exposed it with the same name that the Azure CLI does to be consistent
	// between the tool sets, which means true == Disabled.
	if enabled {
		return "Disabled"
	}

	return "Enabled"
}
```

Modifiability を向上させる戦術: 結合度と凝集度

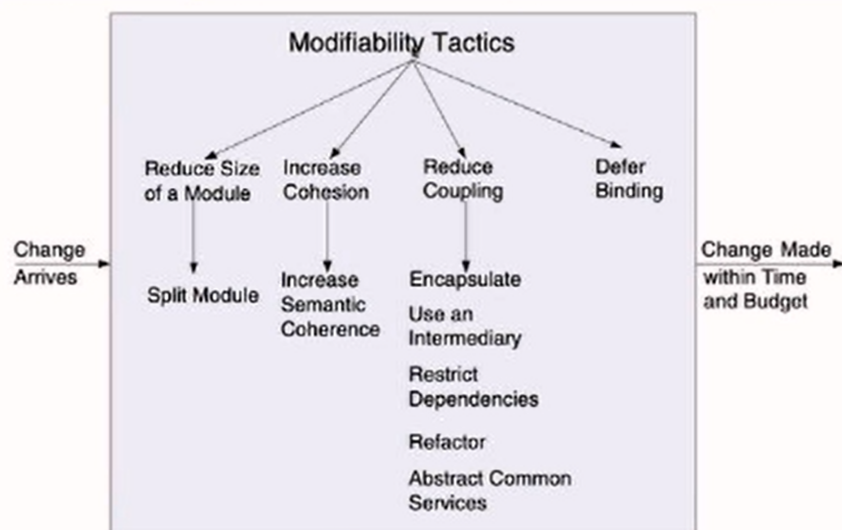


FIGURE 7.3 Modifiability tactics

20年経ってさらに洗練された達人プログラマーの教え

Takuto Wada
@t_wada

多くのプログラマの人生に影響を与えた名著『達人プログラマー』が20年ぶりに改訂され、待望の翻訳版が11/21に発売される。内容の1/3が新規追加で、既存部分もほとんどリライトされている / “達人プログラマー 熟達に向けたあなたの旅（第2版） | Ohmsha”

Translate Tweet

達人プログラマー 熟達に向けたあなたの旅（第2版） | Ohm...
本書は、David Thomas and Andrew Hunt, The Pragmatic Programmer 20th Anniversary Edition (Addison Wesley, ...
ohmsha.co.jp

· はてな / Hatena

Quote Tweets 1.1K Likes

達人プログラマー 第2版
熟達に向けたあなたの旅
David Thomas・Andrew Hunt 共著
村上雅章 訳
The Pragmatic Programmer
your journey to mastery

https://www.amazon.co.jp/dp/4274226298

Agile という言葉は名詞ではなく、物事の進め方を形容する形容詞

アジリティーというものは、現実の世界やソフトウェア開発の世界の双方において、変化に対応する、すなわち何らかの行動に出た後で遭遇した未知なるものごとに対処するということなのです。

「何をしたら良いか」というのは誰も教えてくれません。しかしわれわれは、どれをすべきかという精神について語ることができます。これは不確実性をどのように取り扱うかということに尽きるのです。

『達人プログラマー 第2版』p.334

Agile という言葉は名詞ではなく、物事の進め方を形容する形容詞

アジャイルソフトウェア開発宣言は、フィードバックを収集し、それに基づいて行動するということを示唆しています。このため、アジャイルな方法で仕事をするためのレシピは以下のようなものになります。

1. 自らの現在地点を見つけ出す
2. 目的地点に向けて、最も意味のある最小単位の1歩を踏み出す
3. 現在地点を評価し、問題があれば修正する

上記の手順を作業が完了するまで繰り返します。
そしてこれを、あなたが取り組んでいる
あらゆるレベルの作業に対して再帰的に適用します。

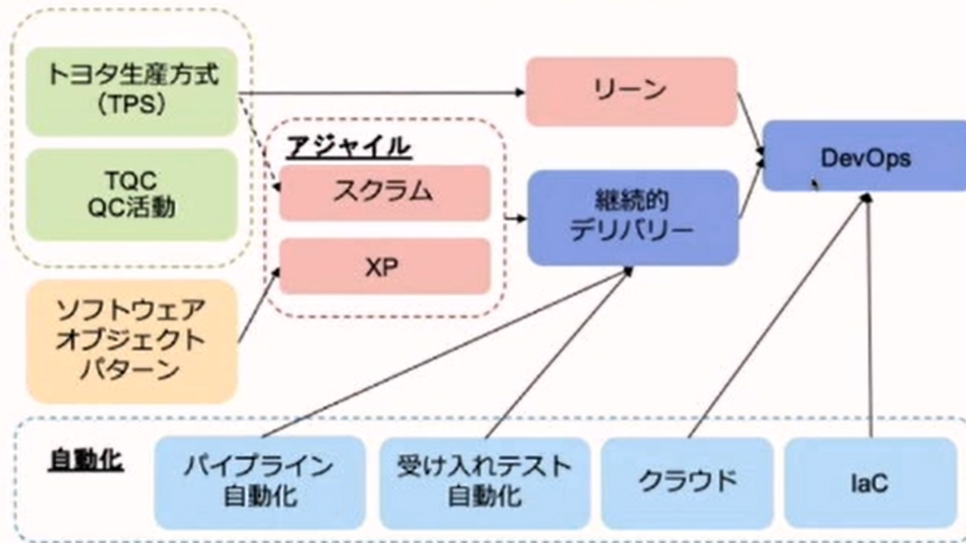
『達人プログラマー 第2版』p.334

Two-way Door: 決定を可逆にする

決定には、結果的に不可逆的なものやほぼ不可逆的なものがあります。これらは一方通行のドアのようなものです。こうした決定を行う際は、慎重に、綿密に、ゆっくりと、熟慮と協議を重ねて行われなければなりません。通り抜けて反対側に出た後で見えているものが気に入らなかったとしても、以前の場所には戻れないからです。これをタイプ 1 の決定としましょう。

しかし、ほとんどの決定はそのようなものではなく、変更可能で、可逆的で、双方向のドアです。もしあなたがこうしたタイプ 2 の決定で最適でない決定をしてしまったとしても、その結果に長く耐える必要はありません。扉を開けて、また戻ればいいのです。タイプ 2 の決定は、判断力の高い個人や少人数のグループが迅速に行えますし、そうすべきです。

ジェフ・ベゾス、Amazon 株主への手紙(2015)


アジャイルからDevOpsへ
トヨタ生産方式
（TPS）
TQC
QC活動
ソフトウェア
オブジェクト
パターン
アジャイル
スクラム
XP
リーン
継続的
デリバリー
DevOps
自動化
パイプライン
自動化
受け入れテスト
自動化
クラウド
IaC
図1 アジャイルからDevOpsへの流れ
https://kokotatata.hatenablog.com/entry/2020/06/01/163652

Parallel Change: 安全なリアーキテクティング戦略
https://www.isejingu.or.jp/about/naiku/ujibashi.html

アジリティを支える品質特性とは保守性と移植性

アジリティーの本質

すべてをアジャイルな形で機能させるには、優れた設計に向けたプラクティスを実践する必要があります。というのも、優れた設計によって変更が容易になるためです。そして変更が容易である場合、あらゆるレベルで躊躇なく調整が可能になるのです。

それこそがアジリティーというものなのです。

『達人プログラマー 第2版』p.336

よい設計の本質：Easier To Change (ETC)

『達人プログラマー 第2版』p.36

よい設計の本質：Easier To Change (ETC)

我々が知る限り、この世の中のあらゆる設計原則はETC原則を特殊化したものとなっています。

結合を最小化するのが望ましいのはなぜでしょうか? それは懸念を隔離することで、変更をしやすくする、つまりはETC原則です。

責務を単一化するという原則はなぜ有益なのでしょうか? それは要求の変更が単一のモジュールに対応づけられる、つまりはETC原則です。

名前の付け方がなぜ重要なのでしょうか？ それは優れた名前はコードの可読性を向上させ、変更時にはその名前を手掛かりにする、つまりはETC原則です

『達人プログラマー 第2版』p.36

ご清聴ありがとうございました

- アジリティが必要になったのは、だれも答えがわからないものを模索しながら作り続ける時代になるから
- アジリティとは複雑で不確実な状態下でフィードバックに基づいて常に意思決定していく力
- アジリティを支える品質特性は保守性と移植性
- DevOps の技術がアジリティを支える
- 究極的にはアジリティの本質はETC原則で補強できる