

# ソフトウェア エンジニアとしての **姿勢と心構え**

Apr 05, 2021 @ リクルート

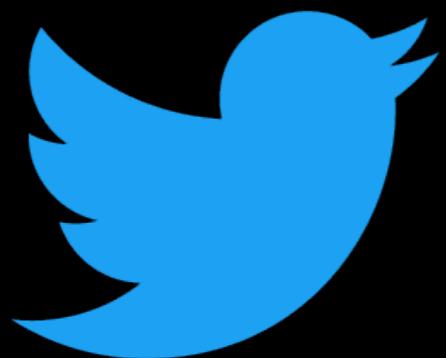
和田 卓人 (@t\_wada)



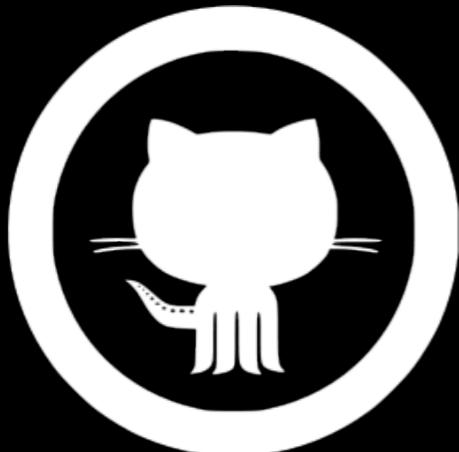
# 和田卓人



t-wada



t\_wada



twada

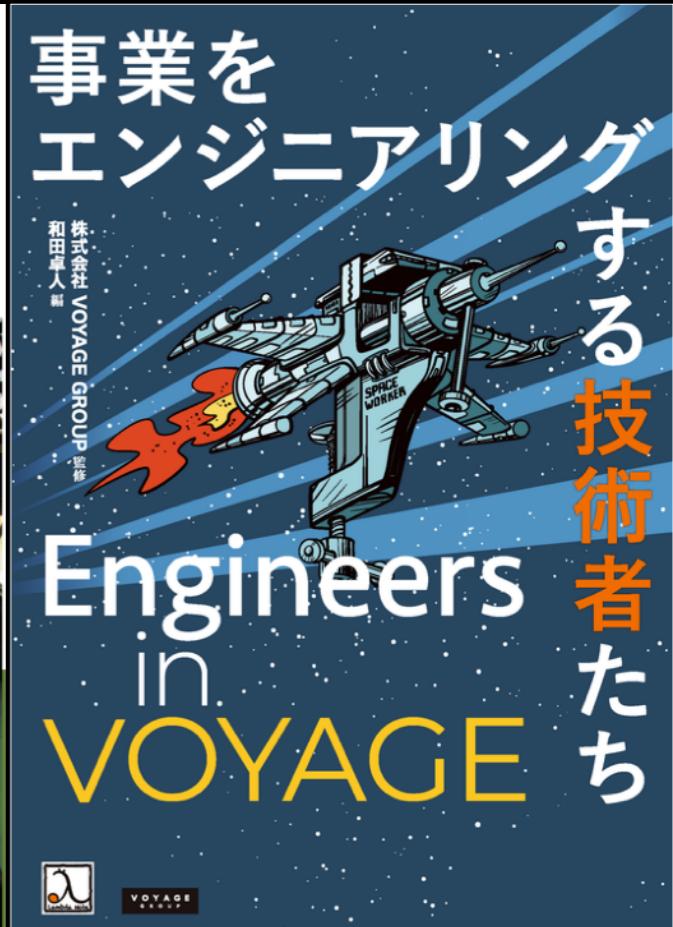
# 手掛けた本たち

97



プログラマが  
知るべき97のこと

97 Things Every Programmer Should Know



# キャリア的なもの

- 大学在学中から設計とプログラミングのアルバイトを始める
- 卒業後プログラマとしてのキャリアを開始
- 電子政府のサブプロジェクト(数千人規模)でリードプログラマ
- XP のコーチとして 4 人のアジャイルチームに参加
- 講演、執筆、OSS 活動を始める
- 現在は技術顧問業を行っている



Collective Wisdom  
from the Experts

# プログラマが 知るべき97のこと

97 Things Every Programmer Should Know

O'REILLY  
オライリー・ジャパン

Kevlin Henney ■  
和田 卓人 著  
夏目 大 訳





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](http://ohmsha.co.jp)

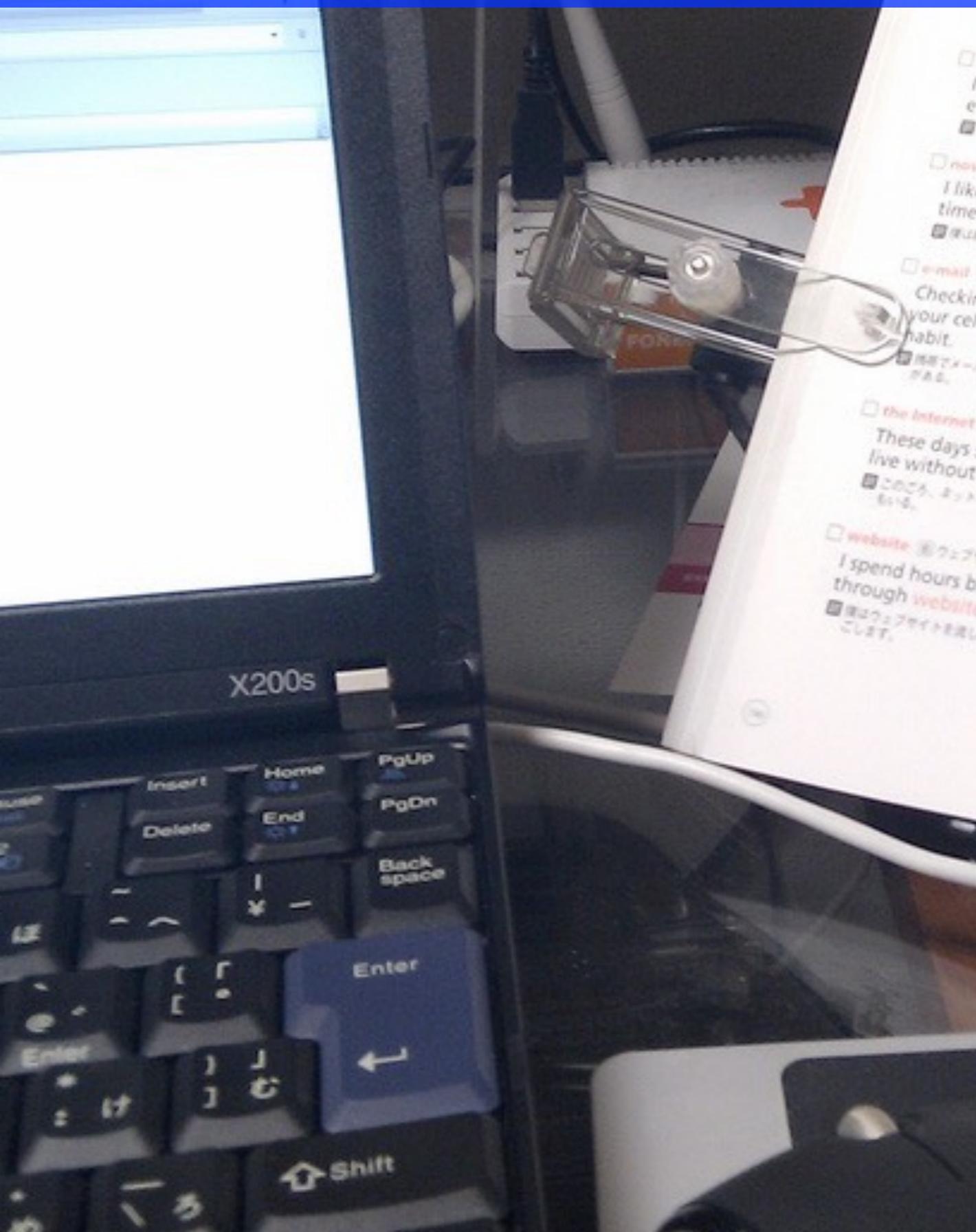
3:06 PM · Nov 6, 2020 · はてな / Hatena

[View Tweet activity](#)

**443** Retweets **65** Quote Tweets **1.1K** Likes



# 学び続ける姿勢



## プログラマが 知るべき97のこと

97 Things Every Programmer Should Know

O'REILLY®  
オライリー・ジャパン

Kevin Henney ■  
和田 卓人 ■  
夏目 大司 ■

## Dialogue

Tom: "Hi! I'm back!"  
Lisa: "Oh, hi! I just came home, t  
Tom: Did you *check* the answerin  
Lisa: Yeah, but there were no nev  
Tom: "Turn on the *TV*, will you? I  
weather forecast for tomorrow  
Lisa: There are no *news* programs  
don't we watch the *video* we  
"until then?  
Tom: Yeah, but I will *check* my e-ma  
tomorrow's *weather* on the Inter  
Lisa: OK, I'll watch TV for a while."  
finished.

トム: たまには、  
リサ: あら、おかえり。私もいま帰ったところよ。  
トム: 電子メール、チェックした?  
リサ: ええ、でも新しいメッセージが  
トム: テレビanton...

“あなたの  
知識ポートフォリオ  
に対して定期的な投  
資を行うこと”



技術を学ぶので  
はなく、技術の  
学び方を学ぶ

# Agenda

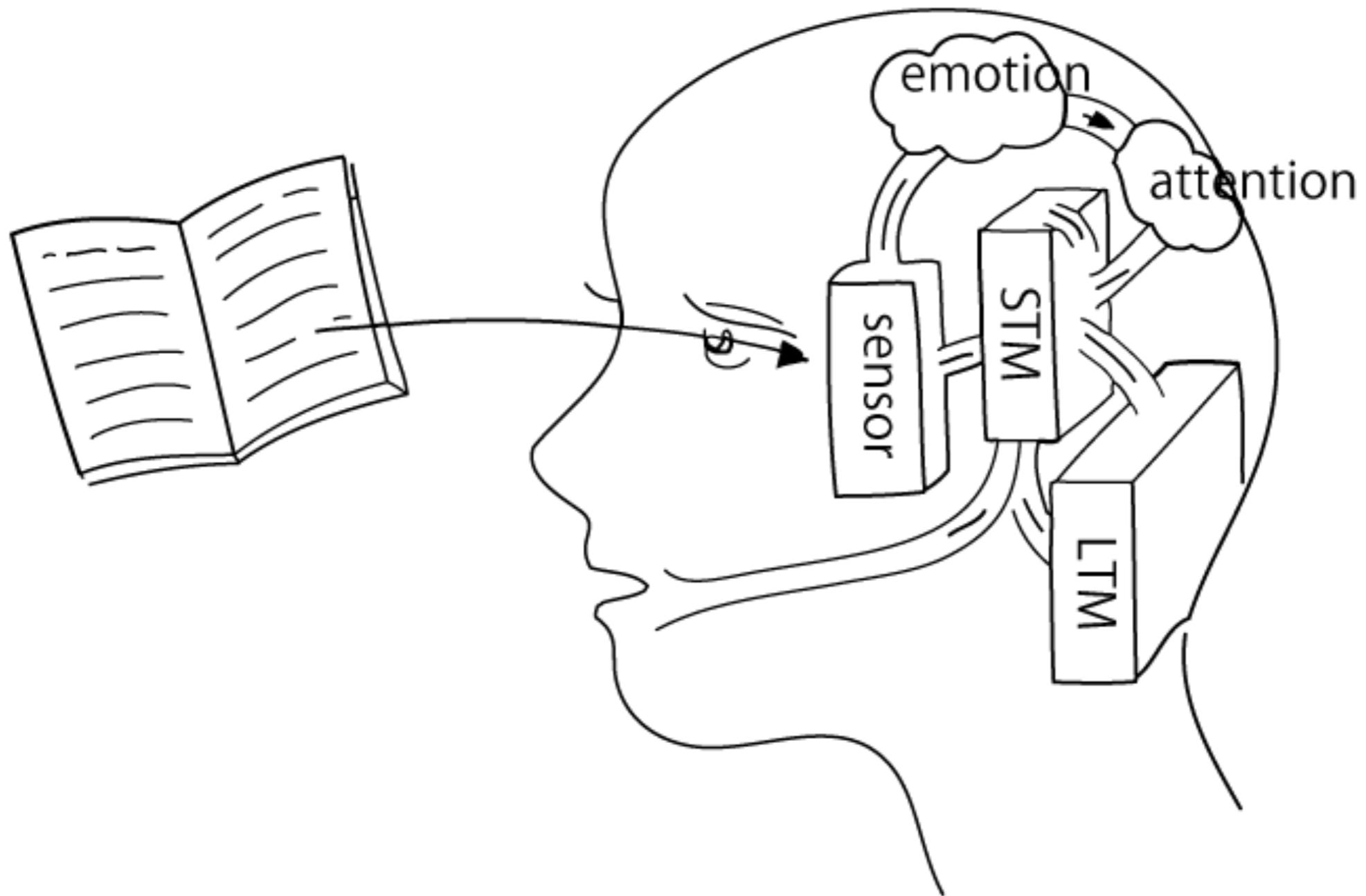


学び方を学ぶ

現役プログラマでいるために

おわりに

# 学びの仕組み



感覚記憶

0.5 ~ 2sec

短期記憶

15 ~ 30 sec

長期記憶

死ぬまで?

# 脳内インデックスを作る



ピッカーを育てる = 反復練習  
何度も長期記憶から出し入れする



荷物を他の荷物とくっつける  
連想記憶を育てる



月に1冊のペースで技術書を読む

手を動かして学ぶ

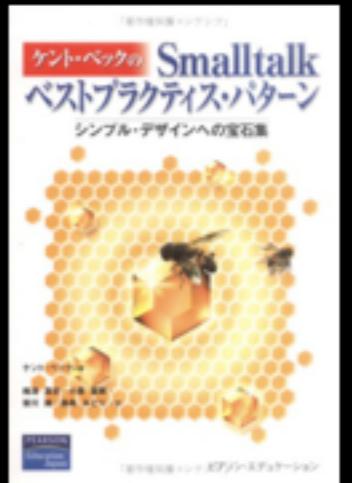
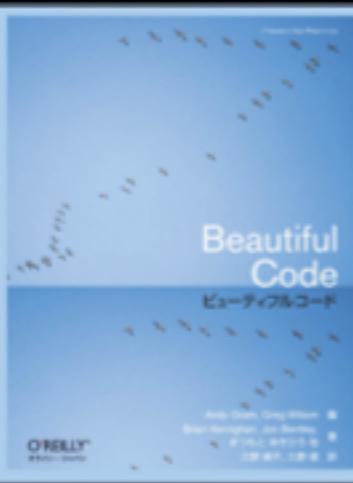
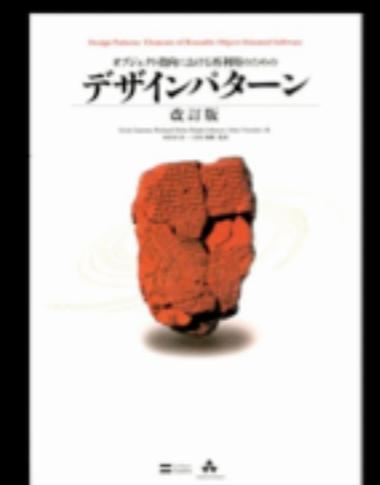
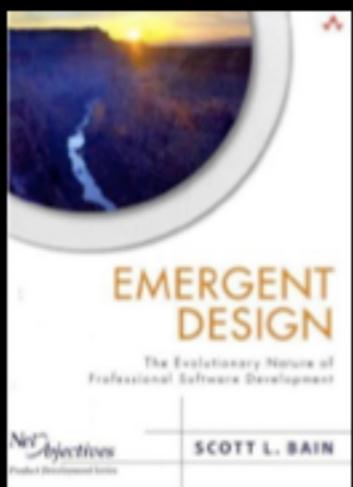
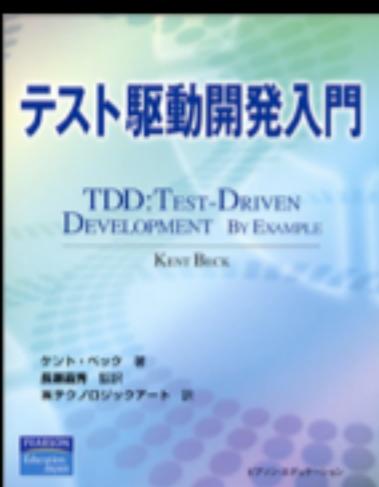
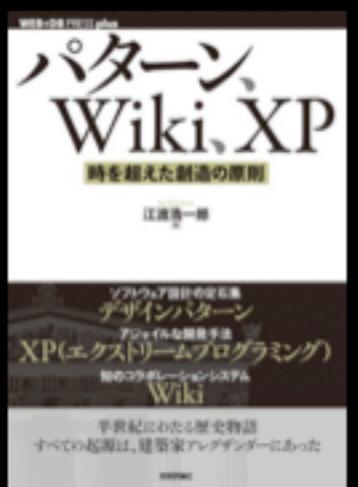
毎年少なくとも言語を1つ学習する

身の回りをプログラミング対象にする

アウトプットを行う

# 1. “月に1冊の ペースで技術書 を読む”





# たとえば、時系列に並べる



# 書籍の時代からテックブログの時代へ



Uber Engineering

## Uber's Journey Toward Better Data Culture From First Principles

Krishna Puttaswamy and Suresh Srinivas

March 16, 2021

Sign up for Uber Enginee...

Your email address

Subscribe

LinkedIn Engineering

Board Browse Pending Changes History

Details for User Defined Test

Request

Source Transport

⋮ 1	USER	http, https
⋮ 2	ML	*
⋮ 3	ML	http/2
⋮ 4	USER	grpc

Engineering Blog

[Rethinking site capacity projections with Capacity Analyzer](#)  
Deepanshu Mehndiratta March 16, 2021

While site outages are inevitable, it's our job to minimize both the duration of outages and the likelihood for an outage to occur. One of our preemptive m...

Topics: Performance, infrastructure, SRE

[Solving the data integration variety problem at scale, with Gobblin](#)  
Chris Li February 24, 2021

Co-authors: Chris Li, Kevin Lau, and Subbu Sanka Editor's Note: Recently, the Apache Software Foundation (ASF) announced Apache® Gobblin™ as a Top-Level Pr...

Topics: Gobblin, Big Data, Open Source

Popular Articles

- Meet Mich... Machine Le... September 2021
- Introducing Microservic... July 23, 2021
- Why Uber ... from Postg... July 26, 2021
- Uber's Big ... Petabytes October 1, 2021

月に1冊のペースで技術書を読む



手を動かして学ぶ

毎年少なくとも言語を1つ学習する

身の回りをプログラミング対象にする

アウトプットを行う

# 2. “手を動かして学ぶ”



プログラマが  
知るべき97のこと

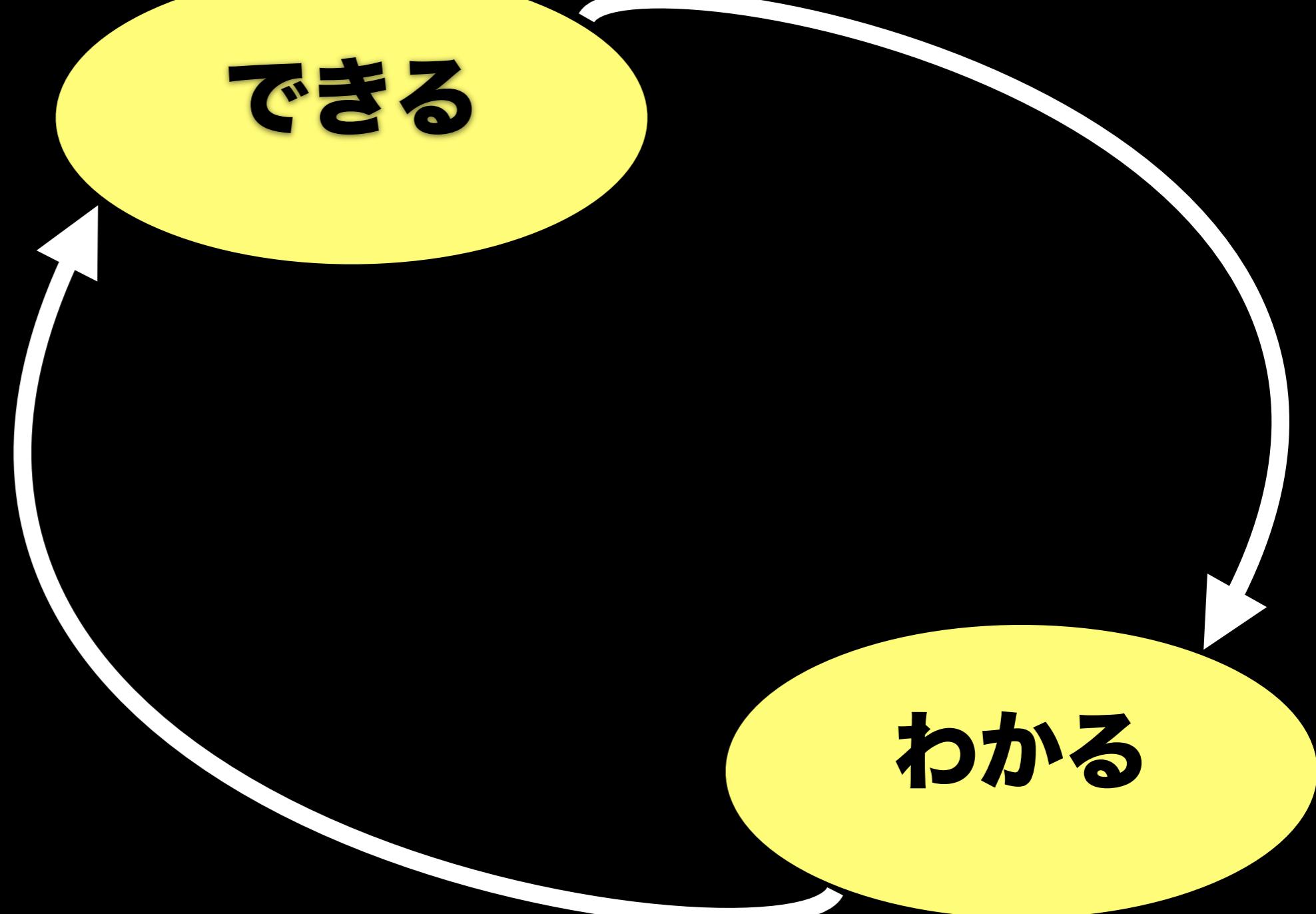
97 Things Every Programmer Should Know

やる

できる

わかる

矢印: 正の接続 =  
根元が増えれば先も増える。  
根元が減れば先も減る



# 写経

68

## Dialogue

Tom: "Hi! I'm back!"

Lisa: "Oh, hi! I just came home, t-

Tom: Did you *check* the answerin-

Lisa: Yeah, but there were no nev-

Tom: "Turn on the *TV*, will you? I

weather forecast for tomorrow

Lisa: There are no *news programs*

don't we watch the *video* we

"until then?

Tom: Yeah, but I will *check* my e-ma-

tomorrow's *weather* on the *Inte-*

Lisa: OK, I'll watch TV for a while."

finished.



トム: ただいま。  
リサ: あら、おかえり。私もいま帰ったところよ。

トム: 部屋電、チェックした?  
リサ:ええ、でも新しいメッセージはなかったわ。

トム: テレビつけてよ。明日の天気予報を見なくちゃいけないんだ。

リサ: 今朝までニュース番組はないわよ。それまで

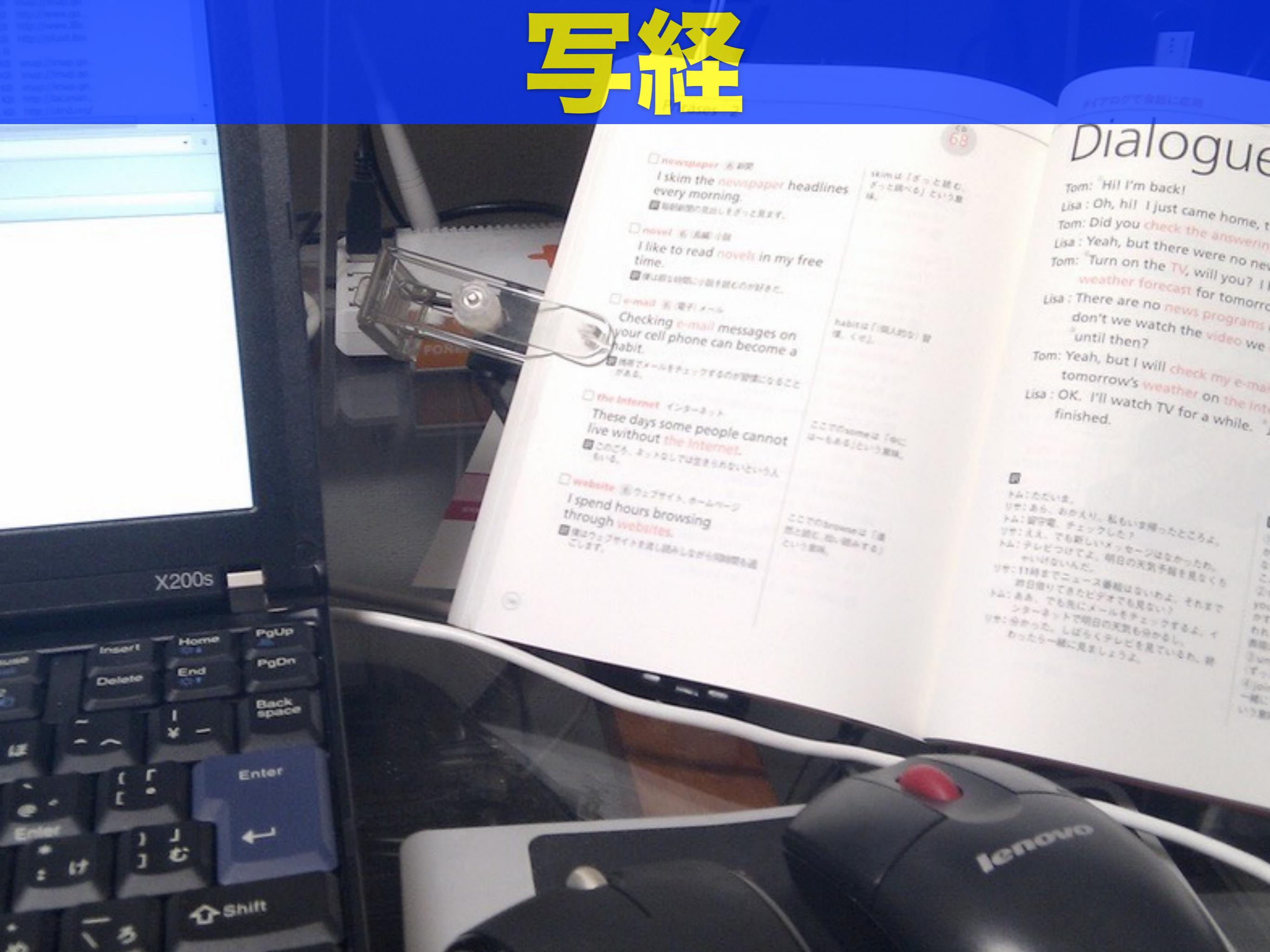
昨日借りてきたビデオでも見ない?

トム: ああ、でも先にメールをチェックするよ。イ

ンターネットで明日の天気も分かるし。

リサ: なかった。しばらくテレビを見ているわ。お

わったら一緒に見ましょよ。





Takuto Wada  
@t\_wada



技術書の「写経」の方法。 1. 口一カルで使える SCM を用意 2. 「ほんたつた」などで対象の本を固定 3. ひたすらサンプルコードを書いて実行 4. 実行するたびにコミット(コミットログにページ番号を含める) 5. 疑問点があったらコミットログや本に書き込む 6. 章ごとにタグを打つ

[Translate Tweet](#)

5:06 PM · Feb 12, 2010 · Twitter Web Client

月に1冊のペースで技術書を読む

手を動かして学ぶ

毎年少なくとも言語を1つ学習する



身の回りをプログラミング対象にする

アウトプットを行う

# 3. “毎年少なくとも 言語を1つ 学習する”



# プログラミング言語は複数習得すべき

第二の言語には、是非とも、最初の言語とはパラダイムの違う言語を選ぶべきです。それはなぜかというと、パラダイムの違う言語を学ぶと、アルゴリズム、イディオム、パターンの実装について嫌でも考えるようになるからです。

同様のアルゴリズムを実装するにしても、色々なやりかたがあり得ることに気づきます。この体験が、プログラマの技術を大きく向上させます。



プログラマが  
知るべき97のこと  
97 Things Every Programmer Should Know

# 言語だけでなく文化も学ぶ

Andy HuntとDave Thomasは、多くの人に影響を与えた著書『達人プログラマー』の中で、「毎年、新たなプログラミング言語を1つは学ぶこと」と勧めています。私はそのアドバイスに従い、過去何年かの間に実際に数多くの言語を学んできました。

そして、その中で「言語を学ぶというのは、ただ文法、構文を学ぶことではなく、その背景にある文化も学ぶこと」という重要な教訓を得ました。



# TECHNOLOGY RADAR

[Search](#)[About the Radar](#)[Build your Radar](#)[Subscribe](#)[Techniques](#)[Tools](#)[Platforms](#)[Languages & Frameworks](#)

## ● ADOPT ?

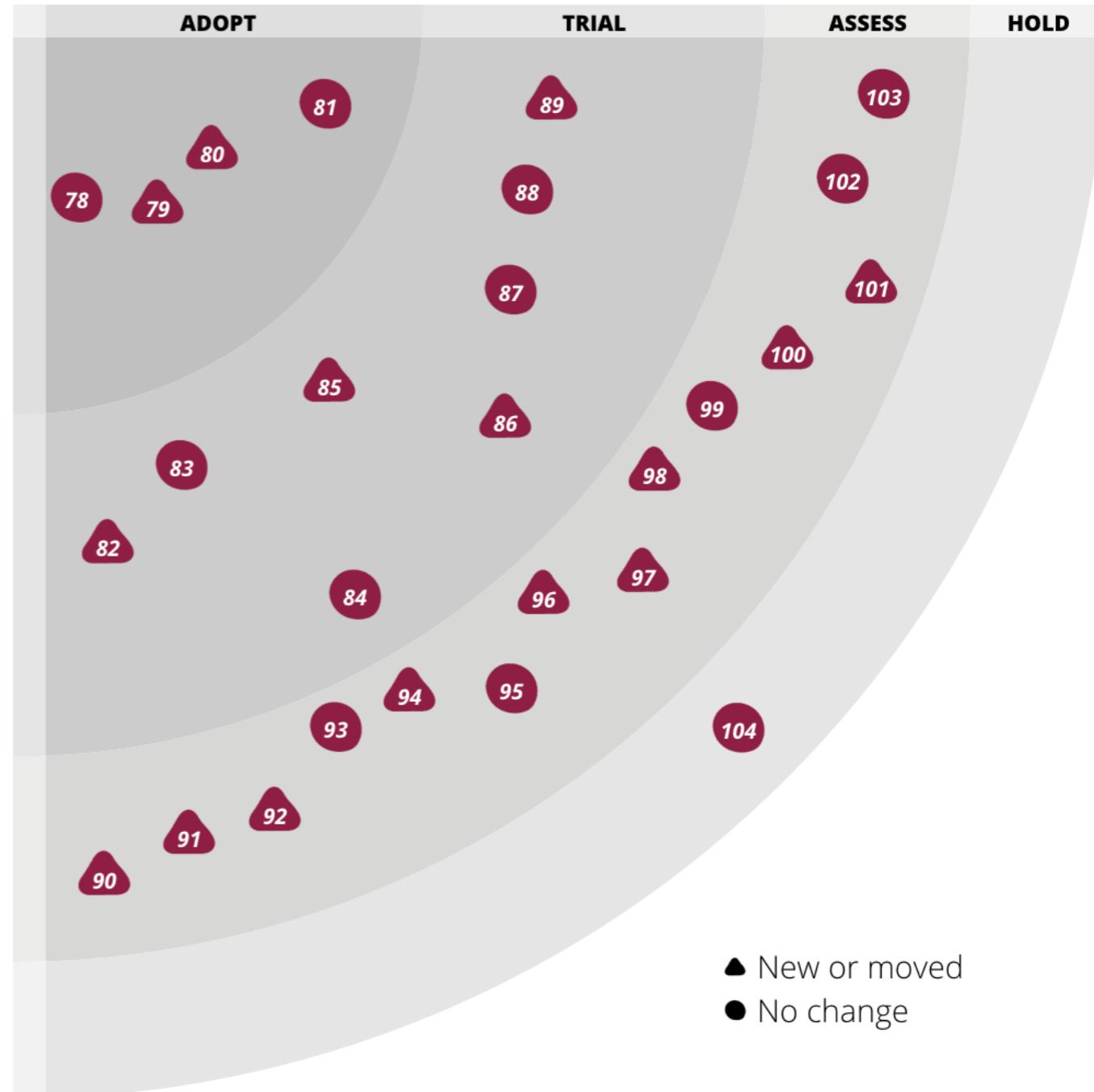
- 78. Ember.js
- 79. Python 3
- 80. ReactiveX
- 81. Redux

## ● TRIAL ?

- 82. Avro new
- 83. Elixir
- 84. Enzyme
- 85. Hangfire new
- 86. Nightwatch new
- 87. Phoenix
- 88. Quick and Nimble
- 89. Vue.js

## ● ASSESS ?

- 90. Angular 2 new
- 91. Caffe new
- 92. DeepLearning.scala new
- 93. ECMAScript 2017



Unable to find something you expected to see? Your item may

# TECHNOLOGY RADAR

Q Search   About the Radar   Build your Radar   Subscribe

Techniques

Tools

Platforms

Languages & Frameworks

**i** The information in our interactive Radar is currently only available in English. To get information in your native language, please download the PDF [here](#).

## ● ADOPT ?

77. Python 3

## ● TRIAL ?

78. Angular

79. AssertJ new

80. Avro

81. CSS Grid Layout new

82. CSS Modules new

83. Jest new

84. Kotlin

85. Spring Cloud

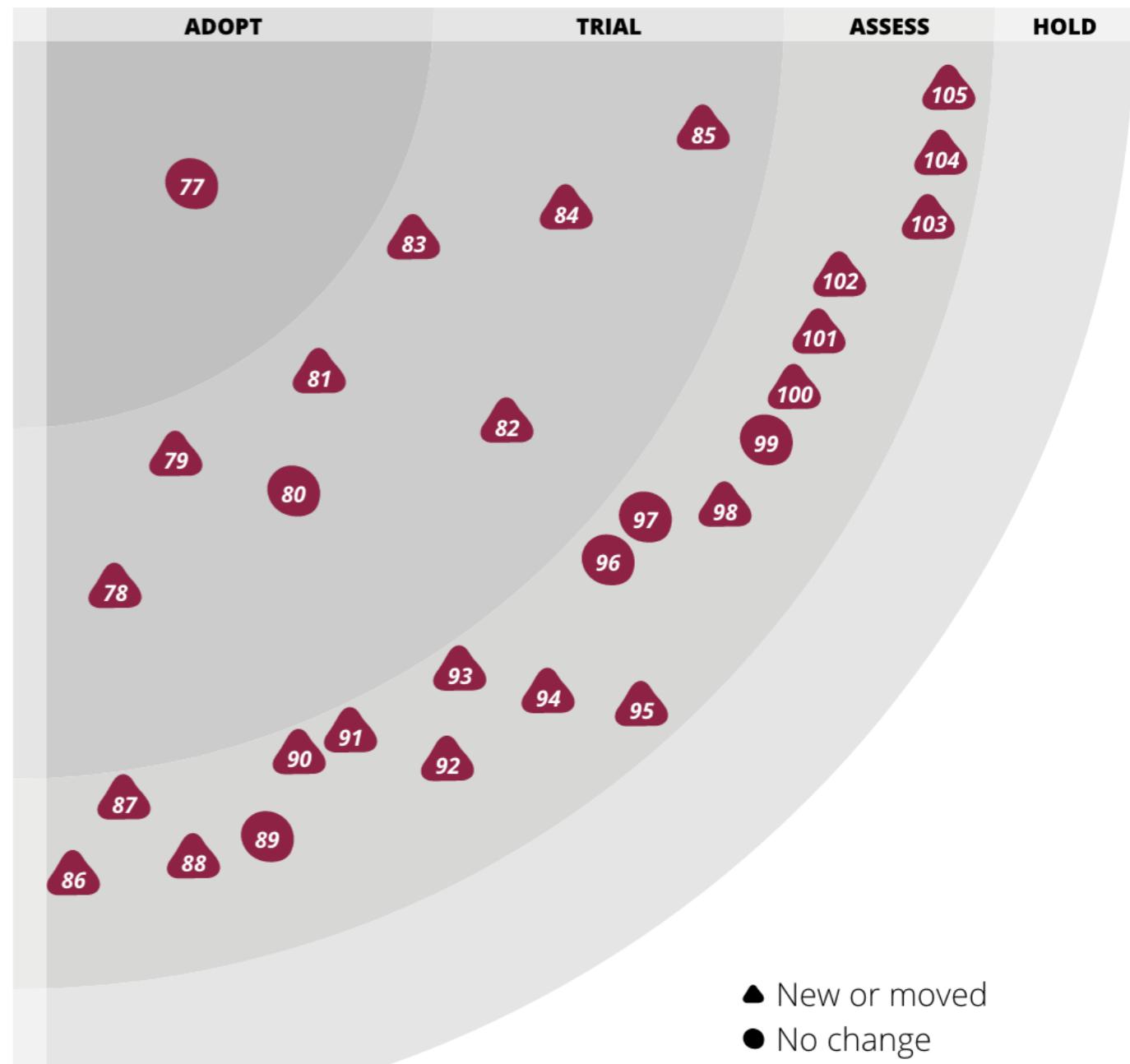
## ● ASSESS ?

86. Android Architecture Components new

87. ARKit/ARCore new

88. Atlas and BeeHive new

89. Caffe



# TECHNOLOGY RADAR

Q Search    About the Radar    Build your Radar    Subscribe

Techniques

Tools

Platforms

Languages & Frameworks

- i** The information in our interactive Radar is currently only available in English. To get information in your native language, please download the PDF [here](#).

## ● ADOPT ?

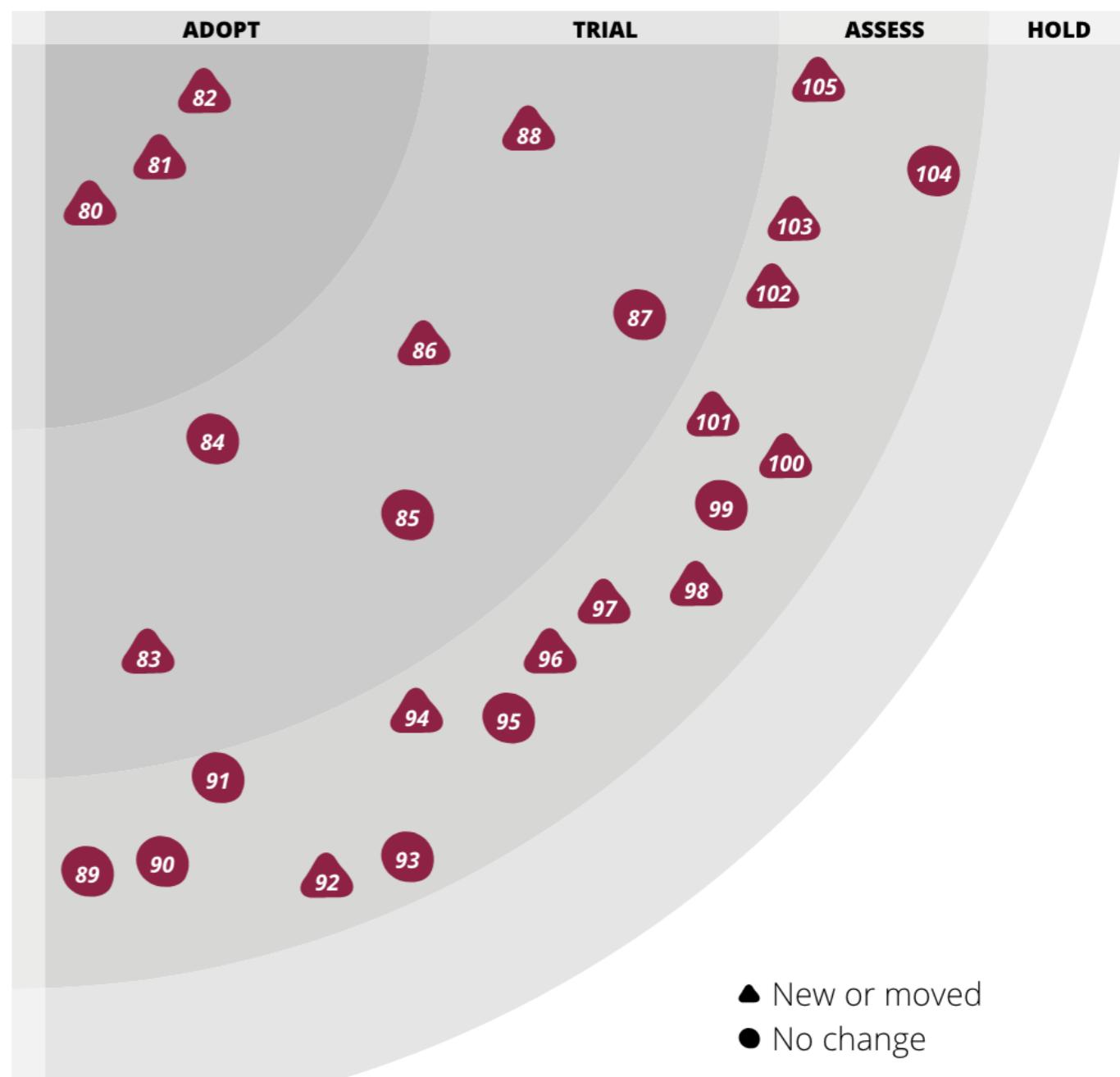
- 80. AssertJ
- 81. Enzyme
- 82. Kotlin

## ● TRIAL ?

- 83. Apollo New
- 84. CSS Grid Layout
- 85. CSS Modules
- 86. Hyperledger Composer New
- 87. Keras
- 88. OpenZeppelin New

## ● ASSESS ?

- 89. Android Architecture Components
- 90. Atlas and BeeHive
- 91. Clara rules
- 92. Flutter New
- 93. Gatsby



**i** The information in our interactive Radar is currently only available in English. To get information in your native language, please download the PDF [here](#).

## ● ADOPT ?

## ● TRIAL ?

76. Jepsen

77. MMKV New

78. MockK New

79. TypeScript

## ● ASSESS ?

80. Apache Beam New

81. Camunda New

82. Flutter

83. Ktor New

84. Nameko New

85. Polly.js New

86. PredictionIO New

87. Puppeteer New

88. Q# New

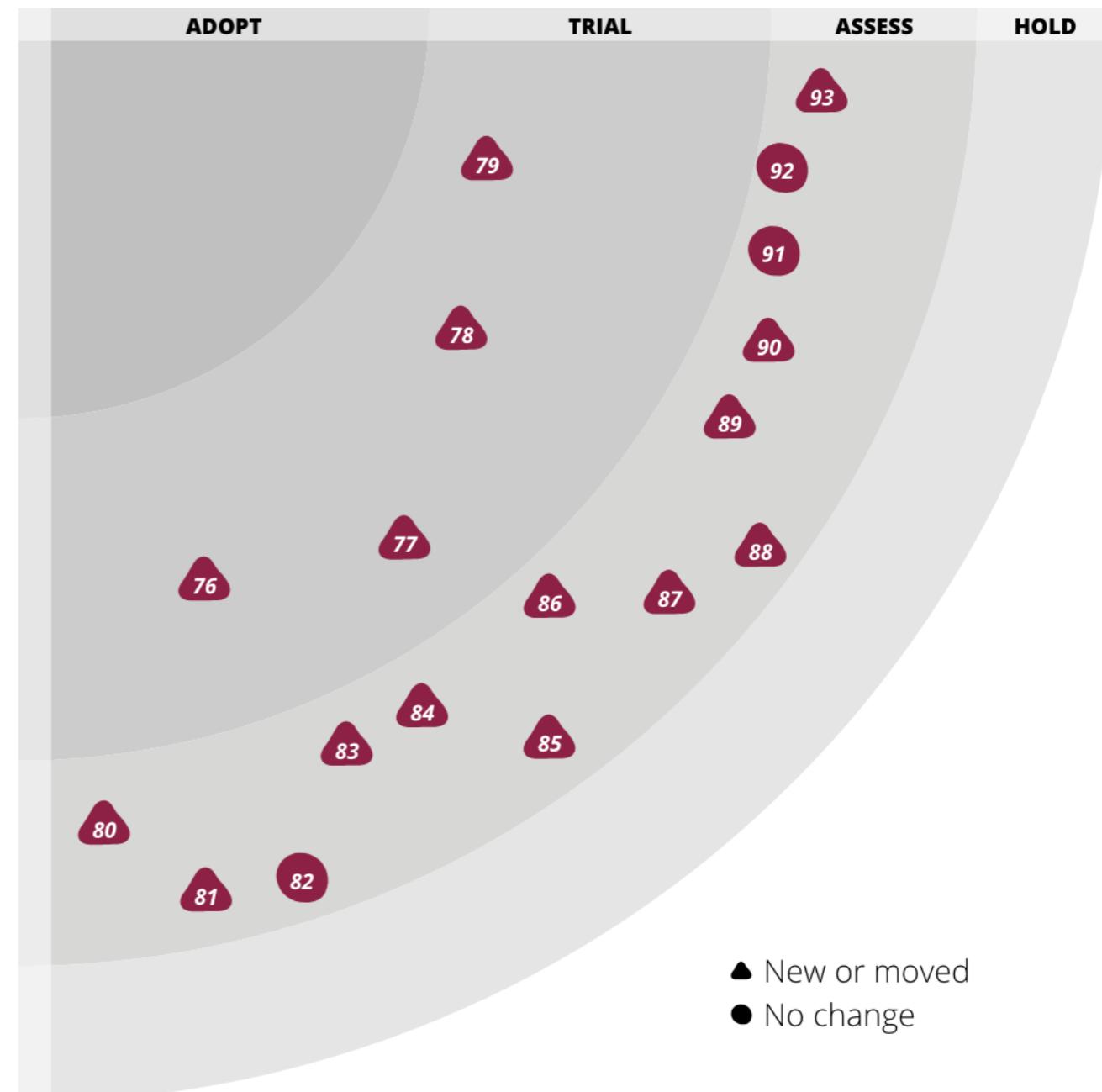
89. SAFE stack New

90. Spek New

91. troposphere

92. WebAssembly

93. WebFlux New



**i** Unable to find something you expected to see?

Each edition of the radar features blips reflecting what we came across during the previous six months. We might have covered what you are looking for on a [previous edition](#) already. We sometimes cull

## ● ADOPT ?

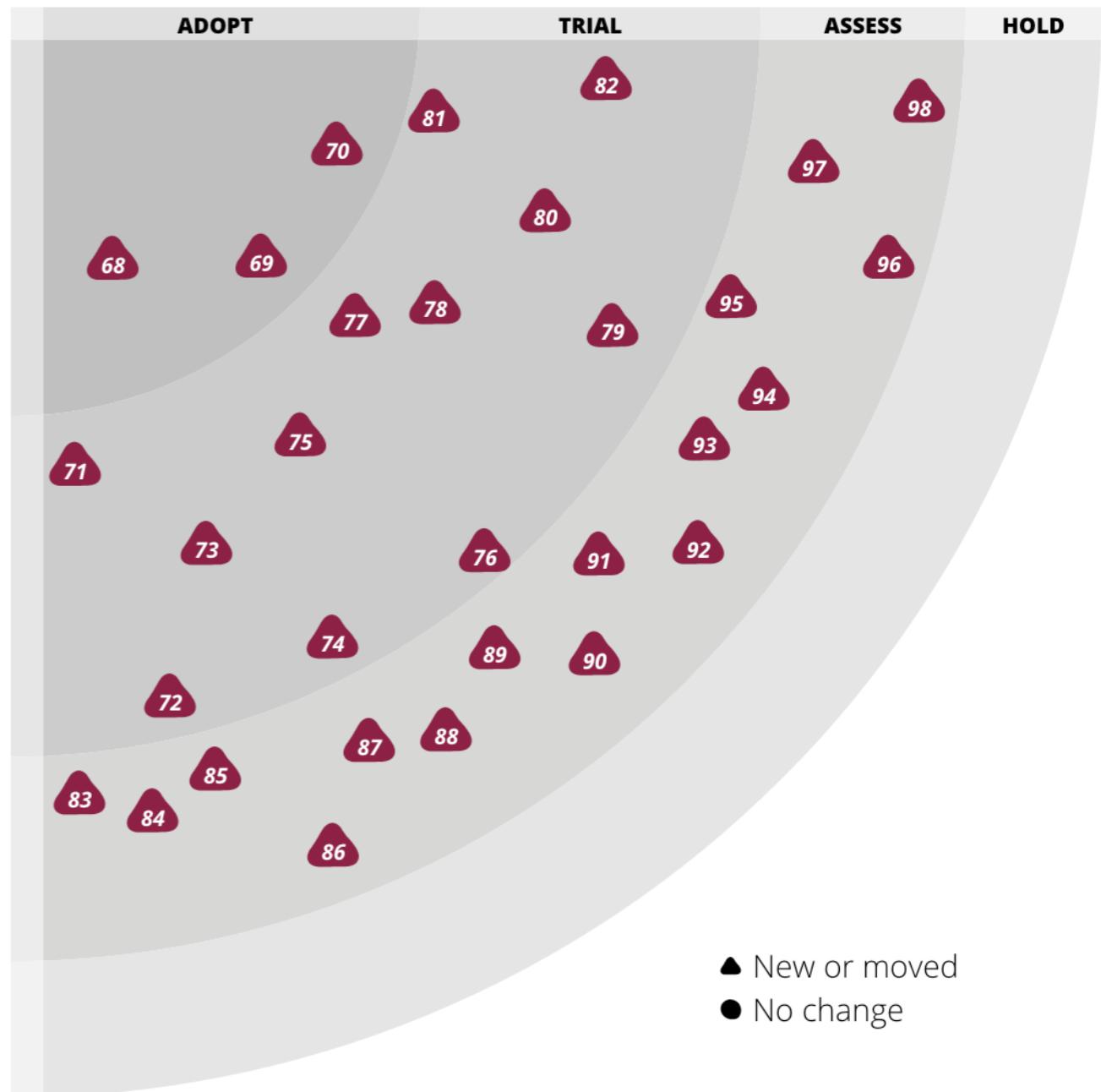
- 68. Apollo
- 69. MockK
- 70. TypeScript

## ● TRIAL ?

- 71. Apache Beam
- 72. Formik
- 73. HiveRunner
- 74. joi
- 75. Ktor
- 76. Laconia
- 77. Puppeteer
- 78. Reactor
- 79. Resilience4j
- 80. Room
- 81. Rust
- 82. WebFlux

## ● ASSESS ?

- 83. Aeron
- 84. Arrow
- 85. Chaos Toolkit
- 86. Dask
- 87. Embark
- 88. fastai
- 89. http4k
- 90. Immer
- 91. Karate



- ▲ New or moved
- No change

**i** Unable to find something you expected to see?

Each edition of the radar features blips reflecting what we came across during the previous six months. We might have covered what you are looking for on a [previous radar](#) already. We sometimes cull things just because there are too many to talk about. A blip might also be missing because the radar reflects our experience, it is not based on a comprehensive market analysis.

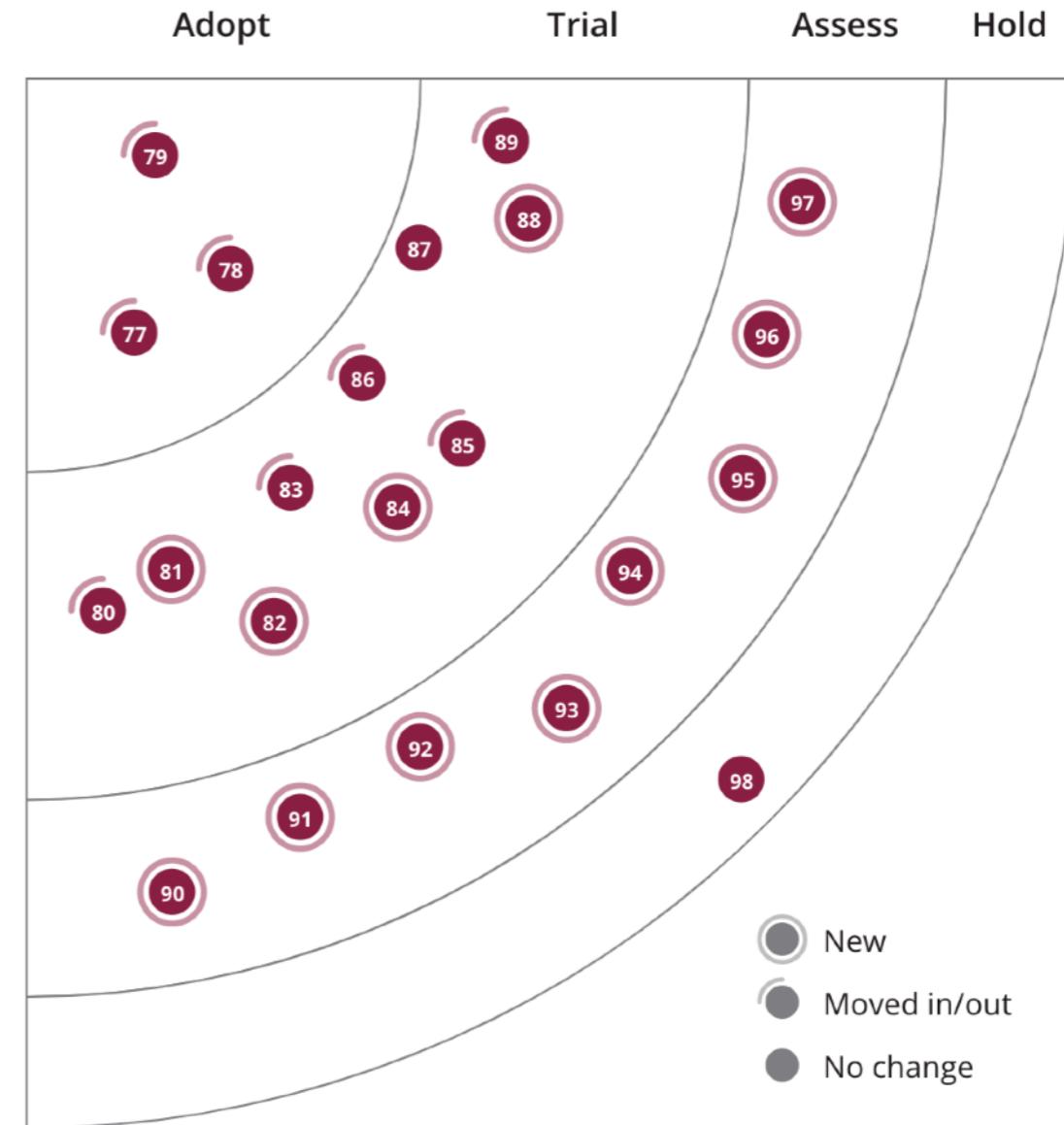
# Languages & Frameworks

## Adopt ?

- 77. React Hooks
- 78. React Testing Library
- 79. Vue.js

## Trial ?

- 80. CSS-in-JS
- 81. Exposed
- 82. GraphQL Inspector
- 83. Karate
- 84. Koin
- 85. NestJS
- 86. PyTorch
- 87. Rust
- 88. Sarama
- 89. SwiftUI



## Assess ?

- 90. Clinic.js Bubbleprof
- 91. Deequ
- 92. ERNIE
- 93. MediaPipe
- 94. Tailwind CSS
- 95. Tamer
- 96. Wire

Unable to find something you expected to see?

Each edition of the radar features blips reflecting what we came across during the previous six months. We might have covered what you are looking for on a [previous radar](#) already. We sometimes cull things just because there are too many to talk about. A blip might also be missing because the radar reflects our experience, it is not based on a comprehensive market analysis.

# Languages & Frameworks

## Adopt ?

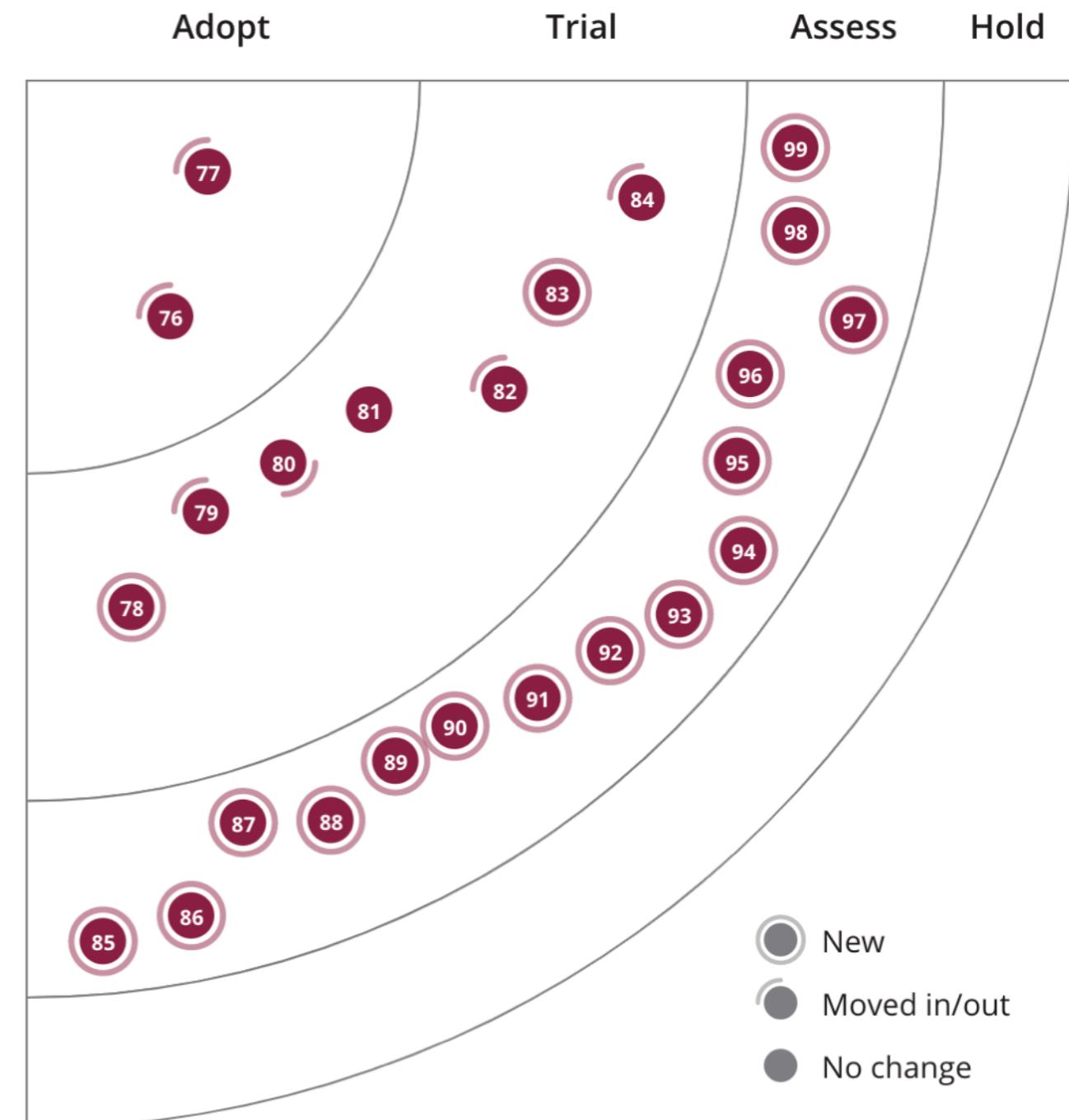
- 76. Arrow
- 77. jest-when

## Trial ?

- 78. Fastify
- 79. Immer
- 80. Redux
- 81. Rust
- 82. single-spa
- 83. Strikt
- 84. XState

## Assess ?

- 85. Babylon.js
- 86. Blazor
- 87. Flutter Driver
- 88. HashiCorp Sentinel
- 89. Hermes
- 90. io-ts
- 91. Kedro



Unable to find something you expected to see?

Each edition of the radar features blips reflecting what we came across during the previous six months. We might have covered what you are looking for on a [previous radar](#) already. We

# 技術者と英語

“英語ができるようになるというのは、  
「大きな図書館の鍵」を渡されるような  
ものです。一人ひとりの人生にいろんな  
可能性を与えてくれます”

—— 高松 珠子



月に1冊のペースで技術書を読む

手を動かして学ぶ

毎年少なくとも言語を1つ学習する



身の回りをプログラミング対象にする

アウトプットを行う

**4. 身の回りを  
プログラミング  
対象にする**



Collective Wisdom  
from the Experts

# プログラマ向けの本の監修者は どうあるべきか

# プログラマらしく

怠惰、傲慢、短気

プレーンテキストを好む

すべてをバージョン管理する

すべてを自動化する

変化を抱擁する

# プログラマらしく

原稿は markdown 形式

原文はスクレイピングして取得

git を使いバージョン管理

heroku に push してサイトに反映

監修差分は docdiff で表示

# 97 Things Every Programmer Should Know

## はじめに

エッセイ一覧

寄稿者一覽

寄稿者紹介

## ゲラ(zip)

EPUB  
Stanza

EPUB で確認

管理者	全原稿	監訳中	監訳済	査読中	査読済	編集中	編集済	
分別のある行動	Seb Rose	001	<a href="#">原文</a>	<a href="#">翻訳</a>	<a href="#">監訳</a>	<a href="#">差分</a>	N/A	<a href="#">編集済にする</a>
関数型プログラミングを学ぶことの重要性	Edward Garson	002	<a href="#">原文</a>	<a href="#">翻訳</a>	<a href="#">監訳</a>	<a href="#">差分</a>	<a href="#">編集後差分</a>	<a href="#">編集済にする</a>
ユーザが何をするかを観察する（あなたはユーザではない）	Giles Colborne	003	<a href="#">原文</a>	<a href="#">翻訳</a>	<a href="#">監訳</a>	<a href="#">差分</a>	<a href="#">編集後差分</a>	<a href="#">編集済にする</a>
コーディング規約を自動化する	Filip van Laenen	004	<a href="#">原文</a>	<a href="#">翻訳</a>	<a href="#">監訳</a>	<a href="#">差分</a>	<a href="#">編集後差分</a>	<a href="#">編集済にする</a>
美はシンプルさに宿る	Jørn Ølmheim	005	<a href="#">原文</a>	<a href="#">翻訳</a>	<a href="#">監訳</a>	<a href="#">差分</a>	N/A	<a href="#">編集済にする</a>
リファクタリングの際に注意すべきこと	Rajith Attapattu	006	<a href="#">原文</a>	<a href="#">翻訳</a>	<a href="#">監訳</a>	<a href="#">差分</a>	<a href="#">編集後差分</a>	<a href="#">編集済にする</a>
共有は慎重に <b>個々のエッセイのステータス管理</b>	Udi Dahan Robert C. Martin (アンクル・トーブ)	007	<a href="#">原文</a>	<a href="#">翻訳</a>	<a href="#">監訳</a>	<a href="#">差分</a>	<a href="#">編集後差分</a>	<a href="#">編集済にする</a>
ボイスアウト・ルール		008	<a href="#">原文</a>	<a href="#">翻訳</a>	<a href="#">監訳</a>	<a href="#">差分</a>	<a href="#">編集後差分</a>	<a href="#">編集済にする</a>
他人よりまず自分を疑う	<b>原文、翻訳原稿、監修原稿へのリンク</b>	Allan Kelly	009	<a href="#">原文</a>	<a href="#">翻訳</a>	<a href="#">監訳</a>	<a href="#">差分</a>	N/A
ツールの選択は慎重に	Giovanni Asproni	010	<a href="#">原文</a>	<a href="#">翻訳</a>	<a href="#">監訳</a>	<a href="#">差分</a>	N/A	<a href="#">編集済にする</a>
ドメインの言葉を使ったコード	Dan North	011	<a href="#">原文</a>	<a href="#">翻訳</a>	<a href="#">監訳</a>	<a href="#">差分</a>	<a href="#">編集後差分</a>	<a href="#">編集済にする</a>
コードは設計である	Ryan Brush	012	<a href="#">原文</a>	<a href="#">翻訳</a>	<a href="#">監訳</a>	<a href="#">差分</a>	<a href="#">編集後差分</a>	<a href="#">編集済にする</a>
コードレイアウトの重要性	Steve Freeman	013	<a href="#">原文</a>	<a href="#">翻訳</a>	<a href="#">監訳</a>	<a href="#">差分</a>	N/A	<a href="#">編集済にする</a>
コードレビュー	Mattias Karlsson	014	<a href="#">原文</a>	<a href="#">翻訳</a>	<a href="#">監訳</a>	<a href="#">差分</a>	<a href="#">編集後差分</a>	<a href="#">編集済にする</a>
コードの論理的検証	Yechiel Kimchi	015	<a href="#">原文</a>	<a href="#">翻訳</a>	<a href="#">監訳</a>	<a href="#">差分</a>	N/A	<a href="#">編集済にする</a>
コメントについてのコメント	Cal Evans	016	<a href="#">原文</a>	<a href="#">翻訳</a>	<a href="#">監訳</a>	<a href="#">差分</a>	N/A	<a href="#">編集済にする</a>
コードに書けないことをコメントにする	Kevlin Henney	017	<a href="#">原文</a>	<a href="#">翻訳</a>	<a href="#">監訳</a>	<a href="#">差分</a>	<a href="#">編集後差分</a>	<a href="#">編集済にする</a>
学び続ける姿勢	Clint Shank	018	<a href="#">原文</a>	<a href="#">翻訳</a>	<a href="#">監訳</a>	<a href="#">差分</a>	N/A	<a href="#">編集済にする</a>
「便利さ」は「利便性」ではない	Gregor Hohpe	019	<a href="#">原文</a>	<a href="#">翻訳</a>	<a href="#">監訳</a>	<a href="#">差分</a>	N/A	<a href="#">編集済にする</a>
すばやくデプロイ、こまめにデプロイ	Steve Berczuk	020	<a href="#">原文</a>	<a href="#">翻訳</a>	<a href="#">監訳</a>	<a href="#">差分</a>	N/A	<a href="#">編集済にする</a>
技術的例外とビジネス例外を明確に区別する	Dan Bergh Johnsson	021	<a href="#">原文</a>	<a href="#">翻訳</a>	<a href="#">監訳</a>	<a href="#">差分</a>	N/A	<a href="#">編集済にする</a>
1万時間の訓練	Jon Jagger	022	<a href="#">原文</a>	<a href="#">翻訳</a>	<a href="#">監訳</a>	<a href="#">差分</a>	<a href="#">編集後差分</a>	<a href="#">編集済にする</a>
ドメイン特化言語	Michael Hunger	023	<a href="#">原文</a>	<a href="#">翻訳</a>	<a href="#">監訳</a>	<a href="#">差分</a>	N/A	<a href="#">編集済にする</a>

Rod Begbie

夜遅くのことでした。その時、私はページレイアウトのテストのため、プレースホルダデータをサンプルデータを入力していました。

ユーザ名には、英國のパンクロックバンド、ザ・クラッシュのメンバーの名前を使いました。会社名には、同じく英國のパンクロックバンド、セックス・ピストルズの曲名を使いました。あとはティッカーシンボルをティッカーシンボル((監訳注: 株式市場で上場企業や商品を識別するため付けられる符丁 <http://ja.wikipedia.org/wiki/ティッカーシンボル>)を入れるだけです。そこで私は、卑猥な4文字の言葉を大文字で入れることにしました。

そう、皆さんご存知の、"F"で始まる言葉なんかを使ったわけです。

特に問題はないだろうと思っていました。自分や他のプログラマが面白がって見るだけのものだし、どうせ翌日には「本物」のデータソースに入れ替えることになっていたからです。ところが翌朝、プロジェクトマネージャが、件の4文字言葉の表示されている画面のスクリーンショットをとり、あるプレゼンに使ってしまったのです。

プログラミング作業中には中の、どうしても、この種の「いたずら」をしたくなるのですが、そうするとプログラミング履歴に妙なデータが入り込んでしまうことになりますや「武勇伝」は良くある話です。こういうのは戦争手記みたいなもので「まあ誰も見ないのだから」と油断していると、思いがけず多くの人の目に触れてしまうことがあります。

露呈の仕方は様々ですが、いずれにしろ珍しいことではありません。またしかし、関わったプログラマ個人や、開発チーム、あるいは会社全体にとって命取りになってしまふこともあります会社全体にとっては命取りになりかねません。どんなパターンがあり得るか、例をいくつかあげておきましょう。

- \* ステータスミーティング中進捗会議中、まだ機能が実装されていないボタンを顧客がクリックしてしまう。すると「二度とクリックすんじゃねーぞ、バーカ！」というメッセージが表示される。
- \* レガシーシステムの保守を担当するプログラマが、エラーダイアログの追加を指示される。そこで彼は、既存のログ機能の出力を利用しようと考える。しかし、元々は裏で動いていたログ機能で、動いていて出力がユーザの目に触れる事はなかったものである。しかし、その改造によって触れる事はなかったログ機能だったために、保守作業によって「おい、バットマン、データベースのクソ野郎がヘマをしやがったぜ！」といったメッセージが画面に表示され、ユーザから見えるようになってしまいます。
- \* 誰かが、テスト用の管理インターフェースと、製品版の管理インターフェースを混同してしまい、ふざけたデータを入力してしまう。その結果、オンラインストアの顧客が、「等身大ビル・ゲイツ型マッサージロボット - 価格100万ドル」といった商品が売られているのを目にすることになる。

「好事多を出せず、悪事千里を走る」というのは古くから言われることです。今の時代なら、なおさらそうでしょう。誰かの「あら」が見つかれば、その噂は、Digg、Twitter、Flib-flarbなどにより、あっという間に世界中に広まってしまいます。担当者が寝ている間に、タイムゾーンの違う国に広まってしまえば、何も手を打つことはできないでしょう。

ソースコードの中でさえ、まったく安心とは言えません。2004年にはWindows 2000のソースコードのTAR書庫がファイル共有ネットワークに流出しています。その時は、ソースコードに卑猥な言葉、ふざけた言葉が使われていないか、grepで嬉々として調べている人間が大勢いました\* (実を言えば、その時に発見された // TERRIBLE HORRIBLE NO GOOD VERY BAD HACK = 「実際にひどい、まったく良いところのない、最悪のハッキングだ」というコメントを私は気に入ってしまい、以来、何度か使っていっています・・・)。

要するにはソースコードの中でさえ、コードに何かテキストを入力するまったく安心とは言えません。2004年にはWindows 2000のソースコードのTAR書庫がファイル共有ネットワークに流出しています。その時は――コメントであれ、あるいはロダソースコードに卑猥な言葉、ダイアロダふざけた言葉が使われていないか、テストデータであれ――常にgrepで嬉々として調べている人間が大勢いました((<http://www.kuro5hin.org/story/2004/2/15/71552/7795>)) (実を言えば、その時に発見された // TERRIBLE HORRIBLE NO GOOD VERY BAD HACK = 「これがもし公になったとして問題にならないか」と自問せよ実際にひどい、ということです。そうすればまったく良いところのない、突然最悪のハッキングだ」というコメントを私は気に入ってしまい、卑猥な言葉が大写しになり以来、その場にいる全員が顔を赤らめるというような事態は防げます何度もか使ってしまっています・・・)。

[\* <http://www.kuro5hin.org/story/2004/2/15/71552/7795>] (<http://www.kuro5hin.org/story/2004/2/15/71552/7795>) 要するには、コードに何かテキストを入力する時は――コメントであれ、あるいはログ、ダイアログ、テストデータであれ――常に「これがもし公になったとして問題にならないか」と自問せよ、ということです。そうすれば、突然、卑猥な言葉が大写しになり、その場にいる全員が赤面するというような事態は防げます。

# 最近つくったもの

子育てや教育関係の

LINE Bot

Amazon Alexa Skill

など

月に1冊のペースで技術書を読む

手を動かして学ぶ

毎年少なくとも言語を1つ学習する

身の回りをプログラミング対象にする



アウトプットを行う

5. アウトプット  
トを行う



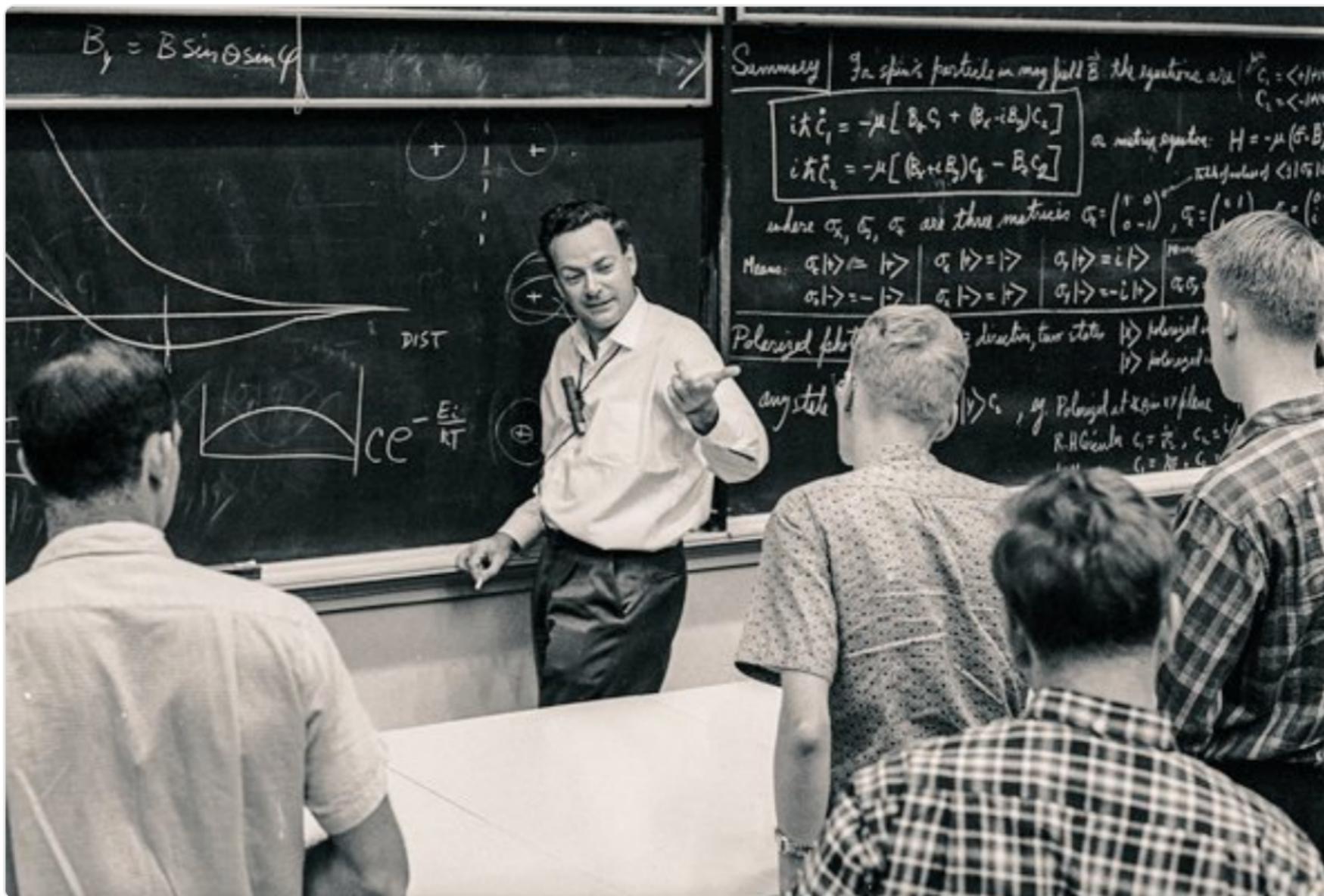
Richard Feynman

@ProfFeynman

Follow



If you want to master something, teach it.



2:15 AM - 6 Apr 2018

1,808 Retweets 4,299 Likes



35



1.8K

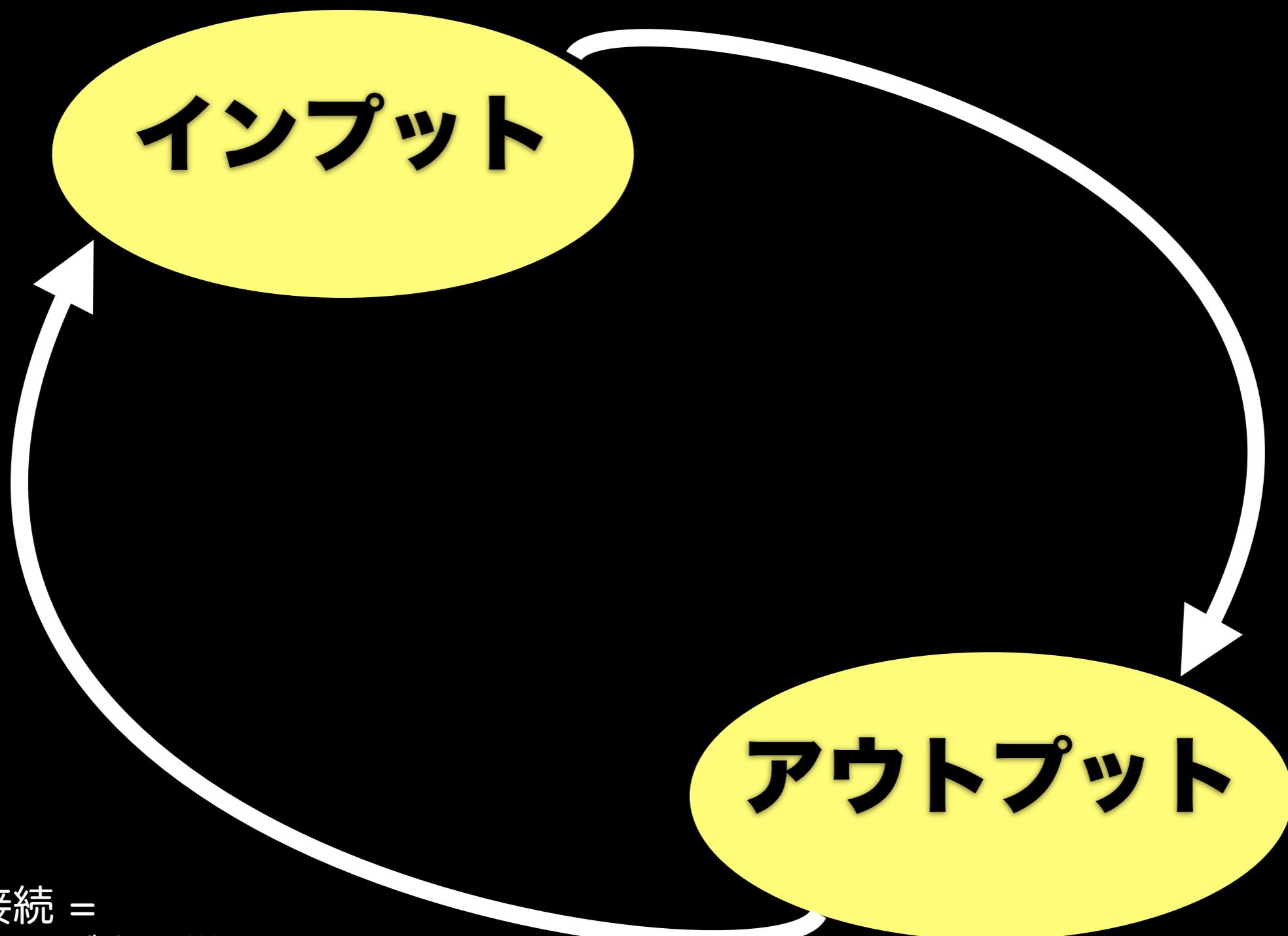


4.3K



<https://twitter.com/ProfFeynman/status/981943455508148225>

# 正のフィードバックループ



矢印: 正の接続 =  
根元が増えれば先も増える。  
根元が減れば先も減る

量は質に転化する



# blog を書く

2013-12-25 tdd

## ■ 不具合にテストを書いて立ち向かう

テストを行っている品質保証チームや、実際にシステムを使っているお客様から不具合が報告されたとき、あなたはどう思いますか？悲しんだり、恥ずかしいと思い、不具合修正にすぐに着手したいと気がはやるのが人情というものです。しかし、焦っているときに行う作業はしばしば視野が狭く、一つの不具合修正が三つの新たな不具合を生んでしまうようなことになりがちです。

テスト駆動開発（TDD : Test Driven Development）は、プログラマが自分の不安を克服し、自分が書くコードに自信を持ちながる一歩一歩進んでいくための手法です。不具合の発生は、端的に言えばこれまでの「自信」を揺らがせる事態です。テスト駆動開発者は不具合にどう立ち向かうのでしょうか？

やはりテストを書いて立ち向かってゆくのです。私はテスト駆動開発を数年間実践してきた中で、心がけているひとつの「捉」があります。それは「**不具合の修正時には必ず先に不具合を再現する自動テストを書いてから修正する**」というものです。これはもちろん私の発案ではなく、XP（eXtreme Programming）やTDDの先達から学び、それを実践するうちに私にも身についたものです。例えば『テスト駆動開発入門』では、次のように記されています。

欠陥が報告されたときに最初にすべきことは何か。→ 失敗する最小のテストを作成し、実行した後に修正する。

回帰テストは、完全な先見があれば、最初のコーディング時に作成していたテストである。回帰テストを作成するたびに、どうすれば、最初にそのテストを作成できたかを考えよう。

—— 『テスト駆動開発入門』 p.136 回帰テスト

不具合修正時のテストは、次のような手順で行います。

01. 手元で不具合を再現させる

“情報発信、blog、発表、公開などは、数学の(未解決問題の)証明ではなく、料理のようなもの”



# 執筆する（まずは雑誌から）



*Test-Driven Development  
By Example*

# テスト駆動開発

Kent Beck 著  
和田卓人 訳



OHM  
Ohmsha

# 技術同人誌市場の登場



Illustration by shatiko

技術書オンラインイベント 第10回 技術に出会えるオンラインイベントに参加しよう！

# コードを公開する



# Takuto Wada

twada

### Block or report user



© <http://tako3.com> / <http://github.com/tako3>

## Organizations



## Pinned repositories

[power-assert-js/power-assert](#)

Power Assert in JavaScript. Provides descriptive assertion messages through standard assert interface. No API is the best API.

JavaScript 1.7k 35

[unassert-js/unassert](#)

Encourages programming with assertions by providing tools to compile them away.

JavaScript ★ 118 🔍 3

licensify

Browserify plugin to prepend license header to your bundle

JavaScript ★ 142 🔍 9

## qunit-tap

A TAP Output Producer Plugin for QUnit

JavaScript 73 14

## type-name

## Just a reasonable `typeof`

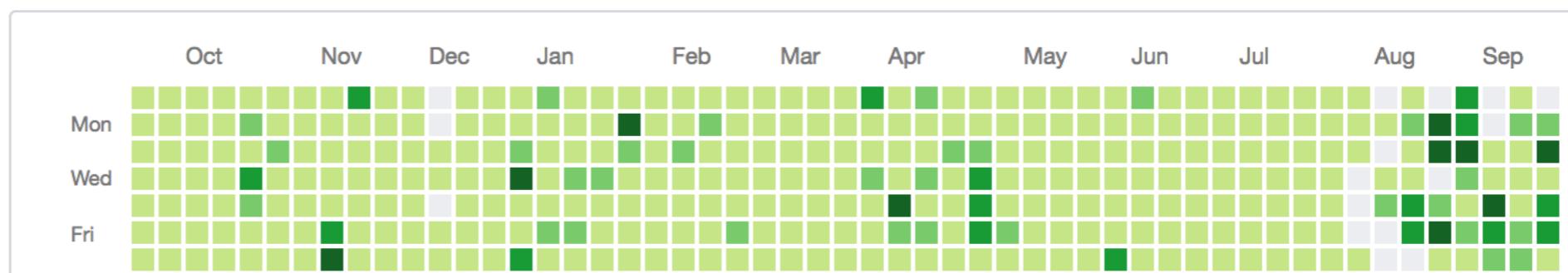
JavaScript 65 4

[coverlay.el](http://coverlay.el)

## Test coverage overlay for Emacs

● Emacs Lisp ★ 47 ⚡ 6

1,695 contributions in the last year



# 講演する

よろしくおねがい

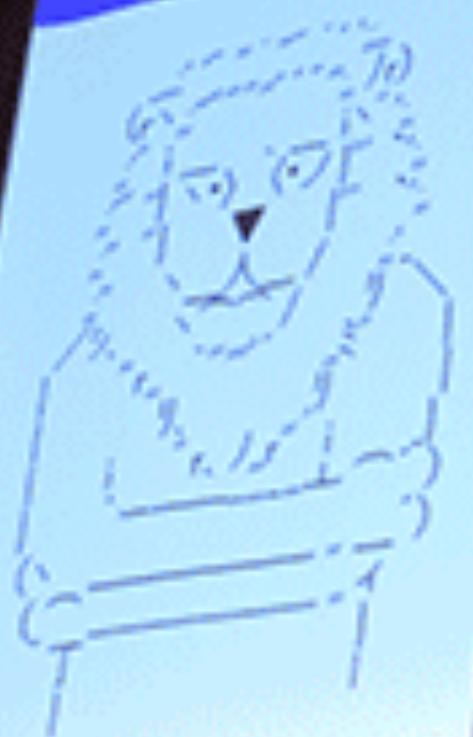
t-wada

t\_wada

twada



#devsumiB



Developers  
Summit

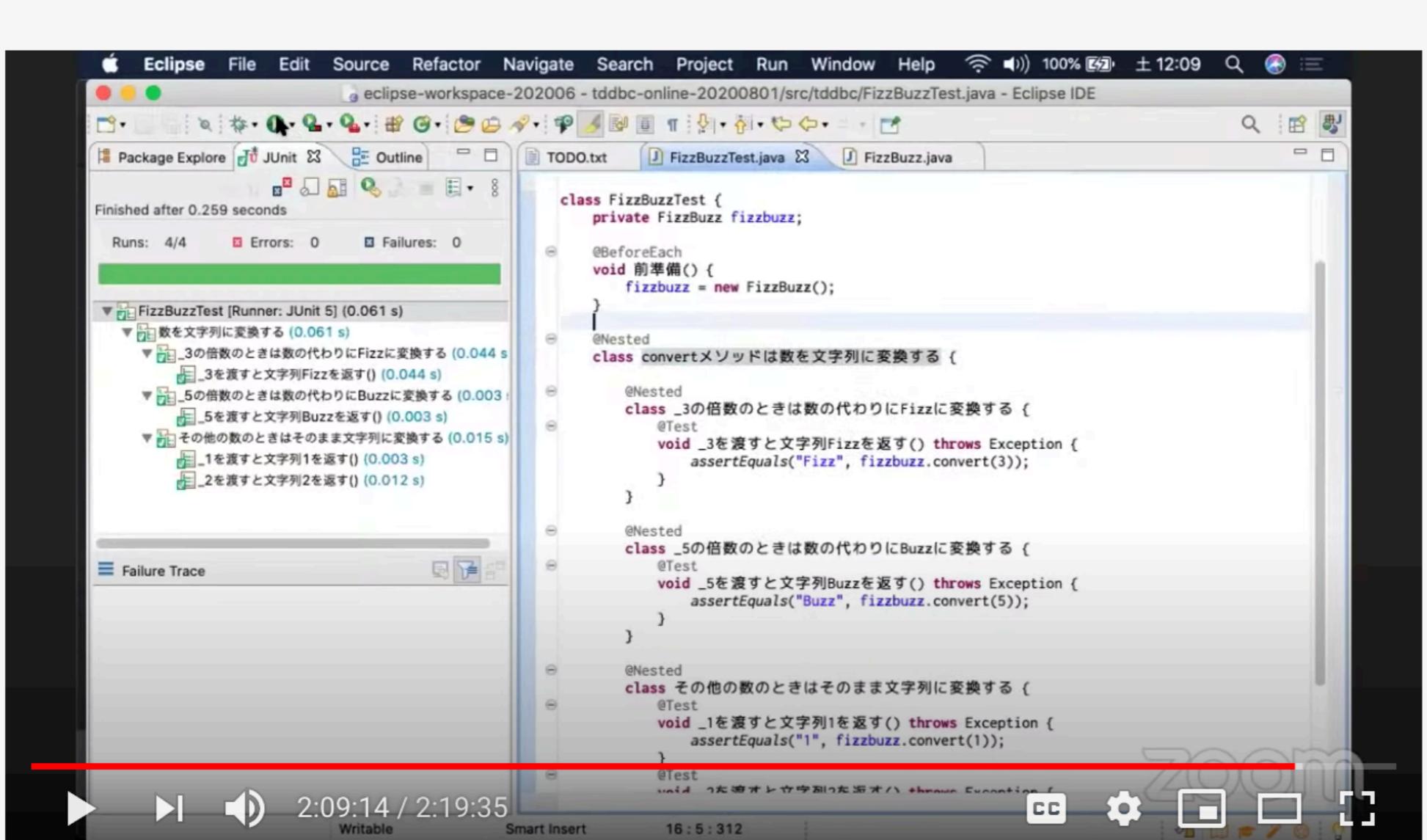
できればライブコーディングで



# 動画配信



Search



## TDD Boot Camp 2020 Online #1 基調講演/ライブコーディング

13,577 views • Streamed live on Jul 31, 2020

482

2

SHARE

SAVE

...



TDDBC-Online  
493 subscribers

SUBSCRIBE

<https://www.youtube.com/watch?v=Q-FJ3XmF1T8>

# アウトプットのチャネル

- Twitter
- blog, Qiita 等
- 雑誌記事(Web, 紙媒体, 電子媒体)
- 書籍(共著, 翻訳, 監訳, 単著)
- 講演 (社内勉強会, 社外LT, 社外講演)
- ライブコーディング
- GitHub



**interval**

**97 KINOCO**

**97 Things Every Programmer Should Know**

# Agenda

学び方を学ぶ



現役プログラマでいるために

おわりに



毎日コードを書く

年下から学ぶ

過去から未来を見る

人のつくる渦を見る

大事なことに集中する

1. 毎日コードを

書く

# あの John Resig でもうまくいかないこと

- jQuery 作者 John Resig は週末に自分のプロジェクト開発を頑張ろうとしたが、失敗。
  - 平日と同じ馬力では書けない
  - 全ての週末が空いているわけではない
  - 一週間（あるいは二週間）は長い。コードを忘れてしまう
- そこで John Resig が行ったことは……



# Write Code Every Day

Last fall, work on my [coding side projects](#) came to a head: I wasn't making adequate progress and I couldn't find a way to get more done without sacrificing my ability to do effective work at [Khan Academy](#).

There were a few major problems with how I was working on my side projects. I was primarily working on them during the weekends and sometimes in the evenings during the week. This is a strategy that does not work well for me, as it turns out. I was burdened with an incredible amount of stress to try and complete as much high quality work as possible during the weekend (and if I was unable to it felt like a failure). This was a problem as there's no guarantee that every weekend will be free – nor that I'll want to program all day for two days (removing any chance of relaxation or doing anything fun).

There's also the issue that a week between working on some code is a long time, it's very easy to forget what you were working on or what you left off on (even if you keep notes). Not to mention if you miss a weekend you end up with a two week gap as a result. That massive multi-week context switch can be deadly (I've had many side projects die due to attention starvation like that).

Inspired by the incredible work that [Jennifer Dewalt](#) completed last year, in which she taught herself programming by building 180 web sites in 180 days, I felt compelled to try a similar tactic: working on my side projects every single day.

# 四つのルール

- 1.毎日コードを書くこと。ブログ、ドキュメント、その他はコードを書いたらやってよい。
- 2.意味のあるコードを書くこと。インデントやフォーマットの修正、可能ならばリファクタリングもコード書きにはカウントしない。
- 3.深夜 24 時前に終わらせること。
- 4.書いたコードを github で全て OSS にすること。

# (当時の) @jeresig の github profile

[Overview](#)[Repositories 91](#)[Stars 377](#)[Followers 13.3k](#)[Following 29](#)

## Pinned repositories

[mongaku/mongaku](#)

An image similarity search engine and database.

JavaScript ★ 18 ⚡ 1

[node-stream-playground](#)

Explore Node.js streams with an interactive playground.

JavaScript ★ 407 ⚡ 34

[node-romaji-name](#)

Normalize and fix common issues with Romaji-based Japanese names.

JavaScript ★ 18 ⚡ 2

[node-yearrange](#)

Node module for converting year range strings into usable dates.

JavaScript ★ 18 ⚡ 1

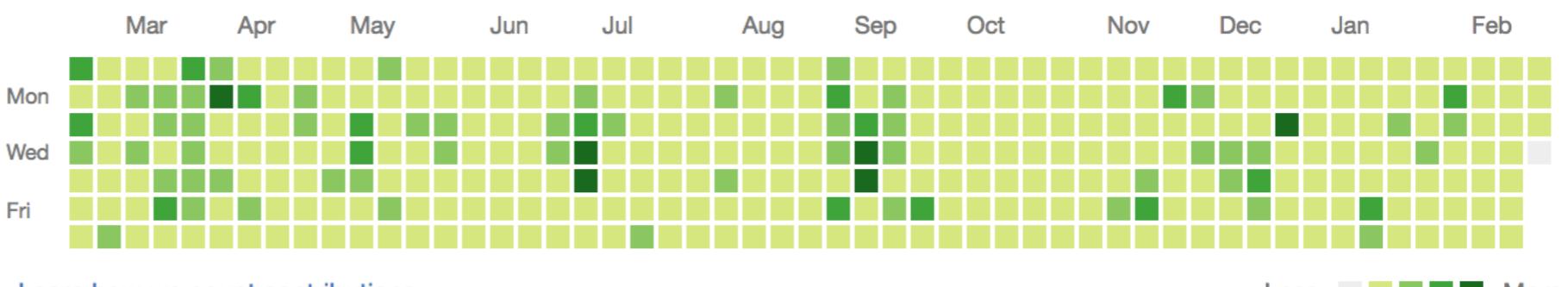
## John Resig jeresig

[Block or report user](#)[@Khan](#)[Brooklyn, NY](#)[jeresig@gmail.com](#)[http://ejohn.org/](#)

## Organizations



## 910 contributions in the last year



# John Resig に起こった変化 (1)

- ・ 必要最小限のコードへの集中: 一日30分～1時間程度で意味のあるコードを書くことが強いられる (休日にはもっとかけられる)
- ・ プログラミングの習慣化: github に草を生やすのが目的ではない。自分で自分自身のために生活習慣を変えるのが大事
- ・ 不安との戦い: 以前は「十分に」進んでいるか、「十分に」完成しているか、不安があった。毎日コードを書いてみて、進んでいるという実感は、実際の進捗と同じくらい重要だという気づきを得た

# John Resig に起こった変化 (2)

- ・ 週末の過ごしかた: 以前は開発の全てを週末に賭けて失敗していたが、いまや週末はそれほど重要でなくなり、リアルライフを充実できるようになった
- ・ バックグラウンド処理: 散歩中、シャワー中、常にコードのことをバックグラウンドで考えるようになり、良いアイデアが浮かぶようになった
- ・ コンテクストスイッチ: 以前は週に一回の開発だったのでコンテクストスイッチのコストがあったが、いまは毎日なのでそれがない

# John Resig に起こった変化 (3)

- ・ ワークライフバランス: 仕事/生活/自分のプロジェクトのバランスの取り方が分かったのが最大の収穫だった。毎日やるということは、バランスを取ること
- ・ まわりからの理解: 「毎日コードを書く」という習慣を公言することで、パートナーからの理解も得られるようになった
- ・ どれだけコードを書いたか: この習慣を続けると書くコードやアウトプットは自分でも覚えられないくらいの量になり、充実感を得られる

「いま、小さなことを多く積み上げることが、とんでもないところへ行くただひとつの道なんだなというふうに感じています」

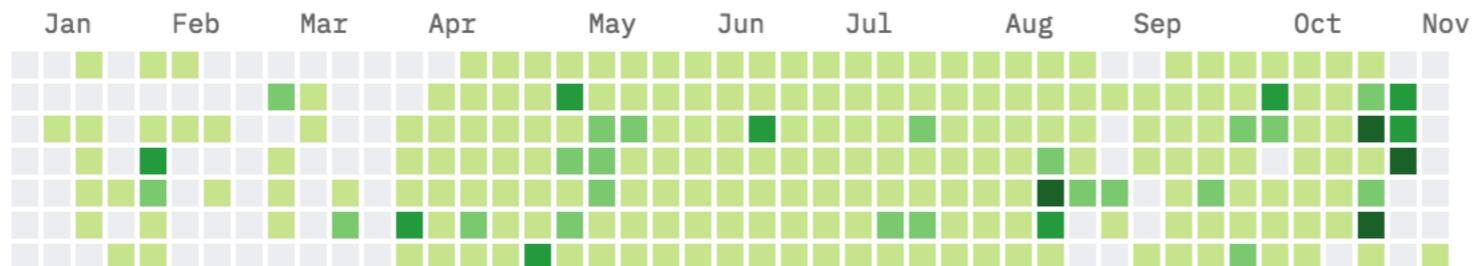
イチローが2004年にMLB年間最多安打を更新したときの言葉

# 私も結構続けました

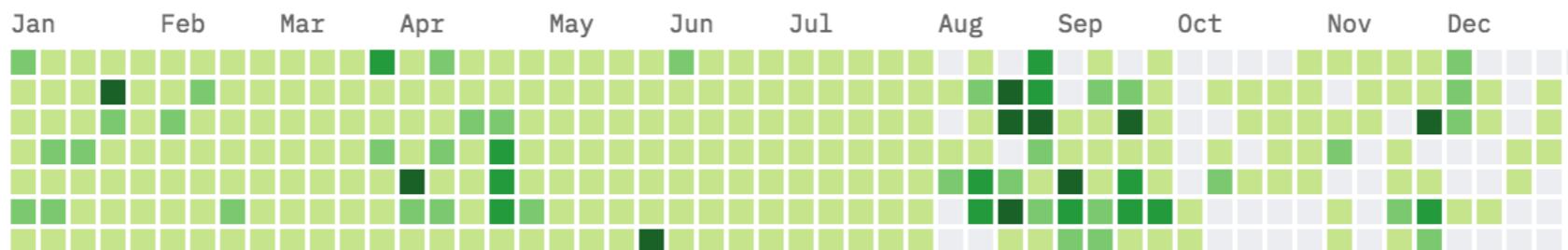
@twada on GitHub

Less  More

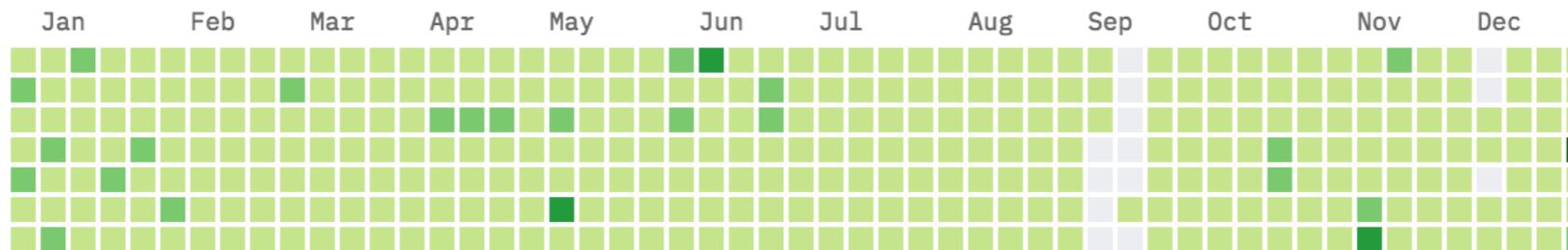
2018: 1,290 Contributions (so far)



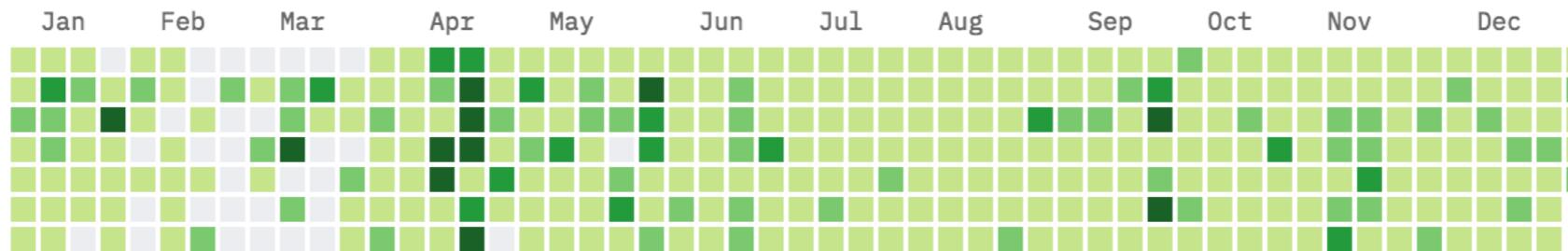
2017: 1,583 Contributions



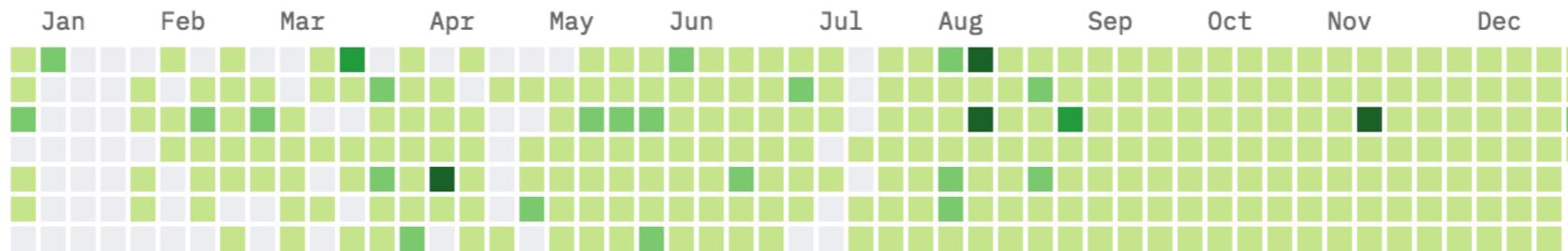
2016: 1,616 Contributions



2015: 2,529 Contributions



2014: 2,740 Contributions



# 事例



Overview

Repositories 101

Stars 345

Followers 44

Following 16

## Pinned repositories

### bqspec

SQL testing tool for Google BigQuery.

Python ★ 6

### embexpr

embedded python expression parser (for mainly easy DSL or config file)

Python

## 趣味

4月に@t\_wadaさんの講義を聞いてから write code every day を実践していくいろんなコードを趣味で書いてきたので紹介させてください。

Recruit Technologies Co.,Ltd.

Tokyo, Japan

owan.orisano@gmail.com

go-ad

unofficial Active Directory Authentication Library for golang

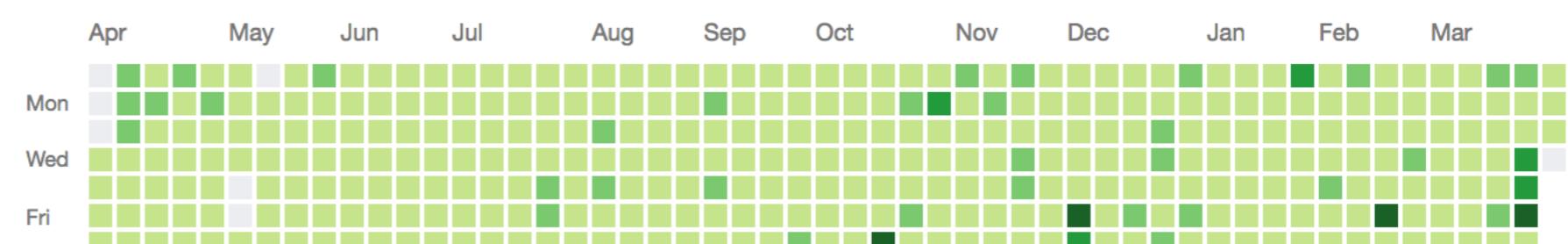
Go ★ 6

Wiener

A Python3 implementation of the Wiener attack on RSA

Python ★ 6 ⚡ 1

1,754 contributions in the last year



Learn how we count contributions.

Less ⚡ More

# 始発駅近くに住み、座れる可能性をコントロールする



毎日コードを書く



年下から学ぶ

過去から未来を見る

人のつくる渦を見る

大事なことに集中する

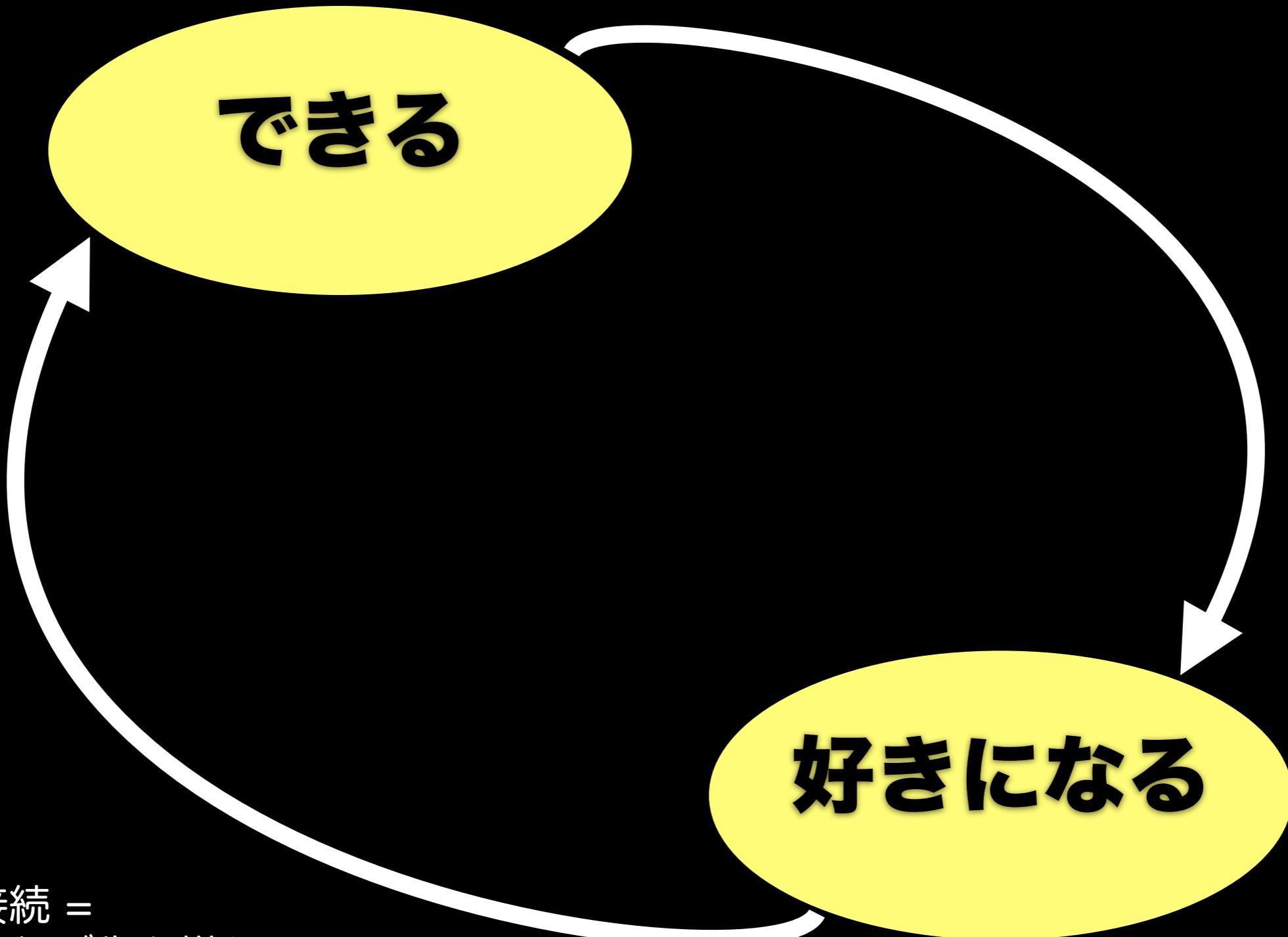
2. 年下から

学ぶ

“一生プログラマーでい  
れるかどうかは、言い  
換えれば年下から学べ  
るか否か。”



# 過剰適合とタコツボ化



矢印: 正の接続 =

根元が増えれば先も増える。  
根元が減れば先も減る

# ベンチマークとアンラーニング

- 定期的に自分のスキルを棚卸しする
- 積極的に外部に出て、自分のスキルを相対化する
- 使う道具を定期的に変える
- 未知のコミュニティに参加する
- 若者から学ぶ
- 若者と同じ土俵で競う

# ペアプログラミング



ベテランにはアンラーニングのチャンス

毎日コードを書く

年下から学ぶ



過去から未来を見る

人のつくる渦を見る

大事なことに集中する

# 3. 過去から 未来を知る

技術は「振り子」



技術は「らせん」





Takuto Wada  
12 Presentations

 Star this Talk 82 Stars

 Published in Programming

 Stats 16,507 Views

 Edit this presentation

## Share

 Twitter, Facebook

 Embed

 Direct Link

 Download PDF

# 技術選定の審美眼

和田卓人 @t\_wada  
2018/02/15 デブサミ2018

#devsumi #devsumiD

技術選定の審美眼 / Understanding the Spiral of Technologies  
by [Takuto Wada](#)

Published February 15, 2018 in Programming

初演: 2018/02/15 デブサミ2018 15-D-1

ハッシュタグ: #devsumi #devsumiD

<https://speakerdeck.com/twada/understanding-the-spiral-of-technologies>

# 技術選定の審美眼。時代を超えて生き続ける技術と、破壊的な変化をもたらす技術を見極める（前編）。デブサミ2018

2018年2月20日

IT分野の技術はつねに速いスピードで変化し続けています。そうしたなかで登場する新しい技術には、スルーしてもかまわないものと、スルーすべきでない重要な技術があります。

めまぐるしい変化の中で、どこに着目することで重要な技術を見極めるのか。一方で、長年にわたって変わらず現役で使われ続けている技術にはどのような特徴があるのでしょうか。

2月15日と16日の2日間、都内で開催されたイベント「Developers Summit 2018」では、こうした変化する技術、変化しない技術をテーマにした、和田卓人氏の講演「技術選定の審美眼」が行われました。

本記事では、その講演の内容をダイジェストで紹介します。

## 技術選定の審美眼

和田卓人といいます。



## カテゴリ

- ・ クラウド
- ・ Docker / コンテナ / 仮想化
- ・ 開発ツール / 言語 / プログラミング
- ・ DevOps / アジャイル開発
- ・ 運用ツール / システム運用
- ・ Web技術 / JavaScript
- ・ Windows / Linux / OS
- ・ サーバ / ストレージ / ネットワーク
- ・ RDB / NoSQL / ミドルウェア
- ・ 機械学習 / AI / ビッグデータ
- ・ 業務アプリケーション / Office
- ・ 働き方 / 給与 / 学び
- ・ 業界動向 / IoT / その他
- ・ 編集後記 / おもしろ
- ・ 殿堂入り / オススメ

## Amazon WorkSpaces

仮想デスクトップで業務の効率化  
無料利用枠内で、2カ月間体験



[詳しくはこちら »](#)



## Blogger in Chief

Junichi Niino (jniino)

IT系の雑誌編集者 オンラインメデ





技術などを深掘りして楽しむPodcastです。



## 29. 技術選定の審美眼(2) w/ twada

2020年03月22日

twadaさんをゲストに、前回に引き続き技術選定の審美眼、集中と分散の螺旋について語っていただいたエピソードです。



## 28. 技術選定の審美眼(1) w/ twada

2020年03月16日

twadaさんをゲストに、技術選定の審美眼およびUnix、REST/Web、RDMBS/SQLの共通点について語っていただいたエピソードです。



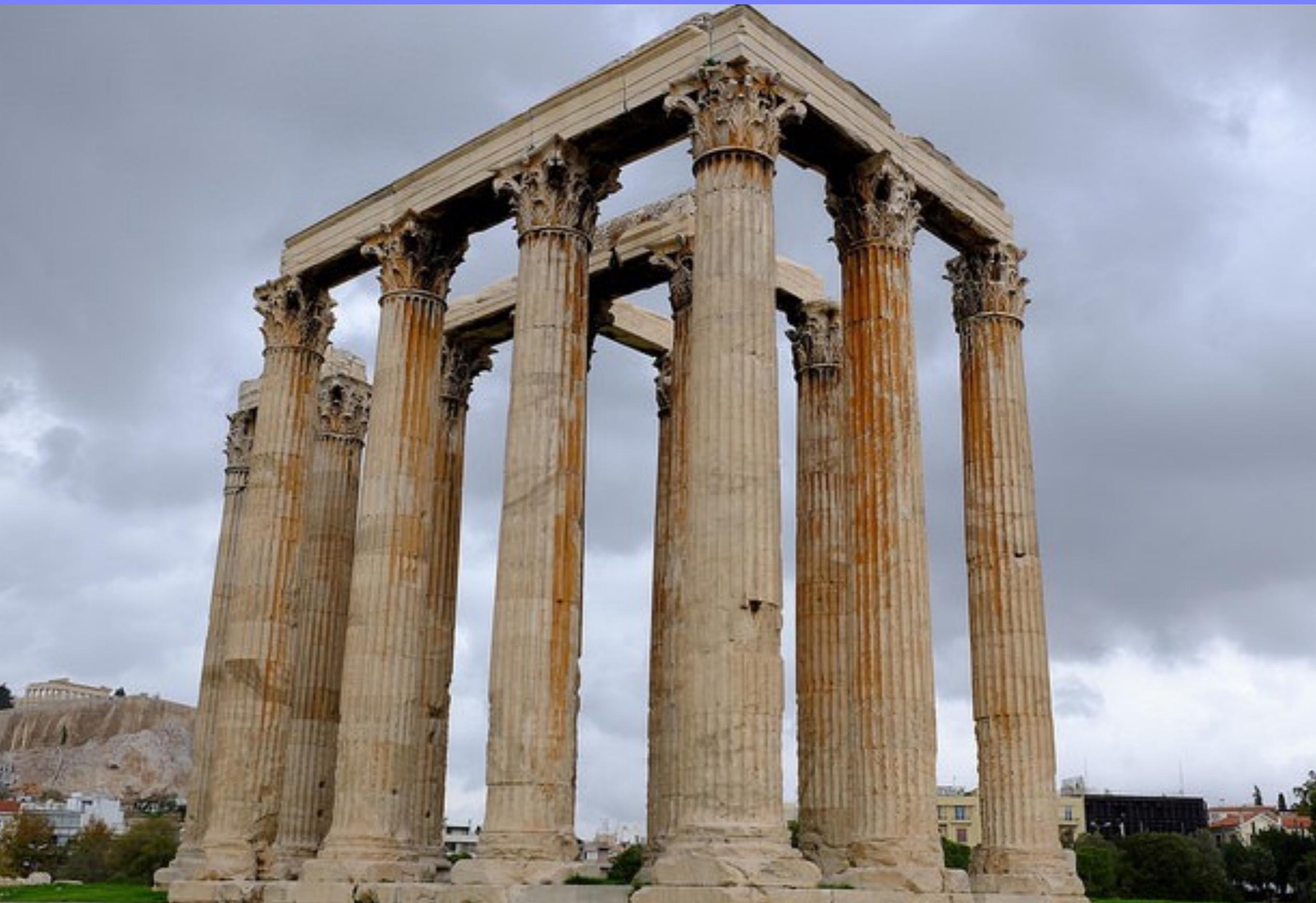
## 27. 論理削除とは何か？どのような解法があるのか？ w/ twada

2020年03月02日

twadaさんをゲストに、RDBにおける論理削除や、その解決方法について語っていただいたエピソードです。



「T字型」ではなく複数の柱を



毎日コードを書く

年下から学ぶ

過去から未来を見る



人のつくる渦を見る

大事なことに集中する

# 4. 人のつくる 洞を見る

# 組織の時代から個人の時代へ



Features   Business   Explore   Marketplace   Pricing

Search GitHub

Sign in or Sign up

## Built for developers

GitHub is a development platform inspired by the way you work. From **open source** to **business**, you can host and review code, manage projects, and build software alongside millions of other developers.

Username

Pick a username

Email

you@example.com

Password

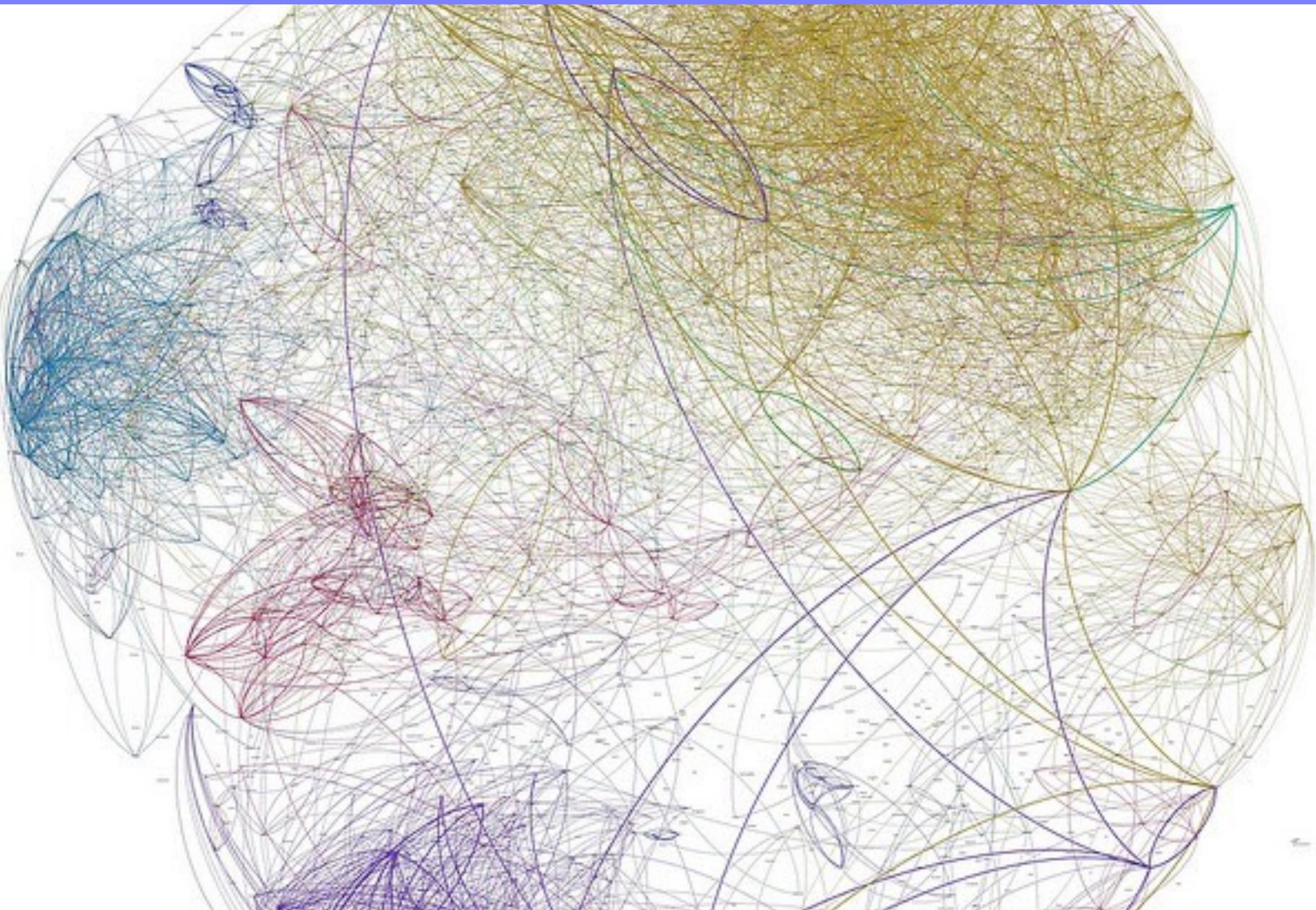
Create a password

Use at least one letter, one numeral, and seven characters.

**Sign up for GitHub**

By clicking "Sign up for GitHub", you agree to our [terms of service](#) and [privacy policy](#). We'll occasionally send you account related emails.

個が多く集まると何かが起こる



# ロードマップ指向からエコシステム指向へ





## My Profile



MY PROFILE  
POWERED BY **ddy**

## hot entries

- [文教族が専門知を評価しないで利権を失うの巻 - アンカテ](#)
- [ドロドロなIT - アンカテ](#)
- [世界を変えているのはコンピュータでなくて数学じゃないだろうか? - アンカテ](#)
- [片山被告の独善性の謎 - アンカテ](#)

## ロードマップ指向とエコシステム指向 ★★18★

IT業界の世代間ギャップを「ロードマップ指向 VS エコシステム指向」という図式でまとめるとうまく整理できるような気がしてきた。

他の業界でも、常に勉強していないと仕事にならない所では、似たような問題があるかもしれない。普通の人は「ロードマップ」の中では真ん中を進むべきで、「エコシステム」の中では真ん中を避けるべきだ、という話。

私は、80年代からずっとプログラマをしていて、今でも現場でコードを書く仕事をしているので、同世代の人から、彼らと現場の若い人との仲裁役というか通訳のようなことを期待されることが多い。

確かにそこには微妙なギャップがあって、自分はどちらの言い分にも共感する所があるので、なんとかそれを言葉にしたいのだが、なかなかうまく言えなかった。

プログラマという仕事は、今も昔も勉強をしてないと普通の仕事も成立しないのだが、その勉強の仕方というか意味づけが、違ってきてると思うのだ。単純に言えば、量の問題である。今は勉強すべきことがずっと多い。

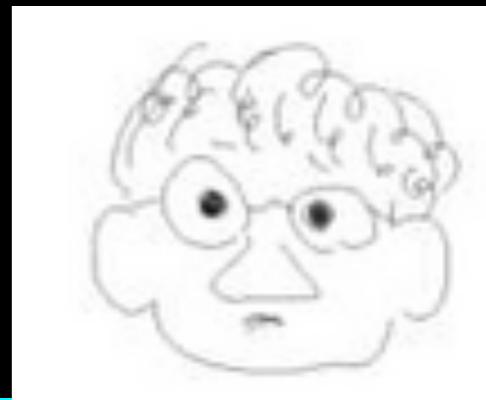
しかし、量の問題としては、何年も前に現場を離れて管理職になっ

“しかし、今の業界は、「エコシステム」の時代だ。熱帯雨林のように、食いあいつつ共生しあうさまざまなタイプのプレイヤーが、自分の為だけの個別の意思決定をして、その相互作用で技術が発展していく。「エコシステム」は矛盾だらけで、ある技術が発展するのと同時に、そのアンチテーゼとなる技術も伸びる”

“「ロードマップ」が指示する未来の方向と違う方向に進むことは致命的な間違いだが、「エコシステム」はむしろ中心部がレッドオーシャンで、周辺部に生き残りが容易なブルーオーシャンがある”



“普通の人は「ロードマップ」  
の中では真ん中を進むべきで、  
「エコシステム」の中では真ん  
中を避けるべきだ”



# The Rise of “Worse is Better”

[rpg@lucid.com](mailto:rpg@lucid.com)

日本語訳:

[daiti-m@is.aist-nara.ac.jp](mailto:daiti-m@is.aist-nara.ac.jp)

私や [Common Lisp](#) と [CLOS](#) のデザイナーのほとんどは、[MIT/Stanford](#) 方式の設計に親しんでいる。この方式の核心は、「正しい」やり方をせよ、ということにつきる。デザイナーにとっては、以下の点をすべて正しく満たすことが重要である。

- 簡潔性  
デザインは実装と使用法の両面において単純でなければならない。このとき、使用法が単純な方が、実装が単純なことより重要である。
- 正当性  
デザインはすべての点において正しいものでなければならない。誤りは許されない。
- 一貫性  
デザインは一貫性を欠いたものであってはならない。一貫性を保つためには完全性は少しだけ犠牲にしてもよい。一貫性は正当性と同じくらい重要である。
- 完全性  
デザインは実際に起こる重要な状況にはすべて対応できなければならない。起こることが予想される場合はすべて扱えなければならない。簡潔さのために完全さが大きく損なわれることがあってはならない。

ほとんどの人がこの条件に同意されることと思う。このデザイン哲学を、以下「MITアプローチ」と呼ぶことにしたい。Common Lisp (とCLOS) や [Scheme](#) はこのデザイン哲学を体現している。

"悪い方がよい"原則はこれと少しだけ異なる：

- 簡潔性  
デザインは実装と使用法の両面において単純でなければならない。このとき、実装が単純な方が、使用法が単純なことより重要である。単純さがデザインにおいて最も重視されるべき事柄である。
- 正当性  
デザインはすべての点において正しいものでなければならない。ただし、どちらかといえば、正しいことよりは単純なことが重要である。
- 一貫性  
デザインは一貫性を失くさないものであってはならない。単純さを保つために、一貫性は犠牲にされることがある。しかし、あまり起こら



Rui Ueyama  
@rui314

Follow

なんか違うんだよなあという方向にまとめている人も少なくなかったけど、これはとてもよいまとめ。

Takuto Wada @t\_wada

Worse Is Better に関するする（両立できない）とする抽象になったとしてもドルが下がり、進化的な

Show this thread

4:01 PM - 7 Apr 2018

28 Retweets 101 Likes



↑ 28



Takuto Wada

@t\_wada

Follow

Worse Is Better に関する自分の解釈は「設計の正しさ/美しさと実装の単純さが対立する（両立できない）ときは、実装の単純さを選択した方が、たとえそれが漏れのある抽象になったとしても現実の問題を解決し、実装の単純さによって開発参加のハードルが下がり、進化的な強さを獲得できる」というもの

12:17 AM - 6 Apr 2018

126 Retweets 304 Likes



↑ 126



304



# 過去を知り、未来に備える 技術選定の審美眼2019

和田卓人 @t\_wada

2019/05/17



Takuto Wada

May 17, 2019

Programming

37

3.3k



毎日コードを書く

年下から学ぶ

過去から未来を見る

人のつくる渦を見る



大事なことに集中する

5. 大事なことに  
集中する

# エッセンシャル思考

最少の時間で成果を最大にする

グレッグ・マキューン=著 高橋璃子=訳



ダニエル・ピンク（『モチベーション3.0』著者）

クリス・ギレボー（『1万円起業』著者）

アダム・グラント（『GIVE & TAKE』著者）

2014年

全米ベストセラーの邦訳！

apple、google、facebook、Twitterのアドバイザーを務める著者の

**1 99%の無駄を捨て  
1%に集中する方法！**

10万部  
突破！



# Agenda

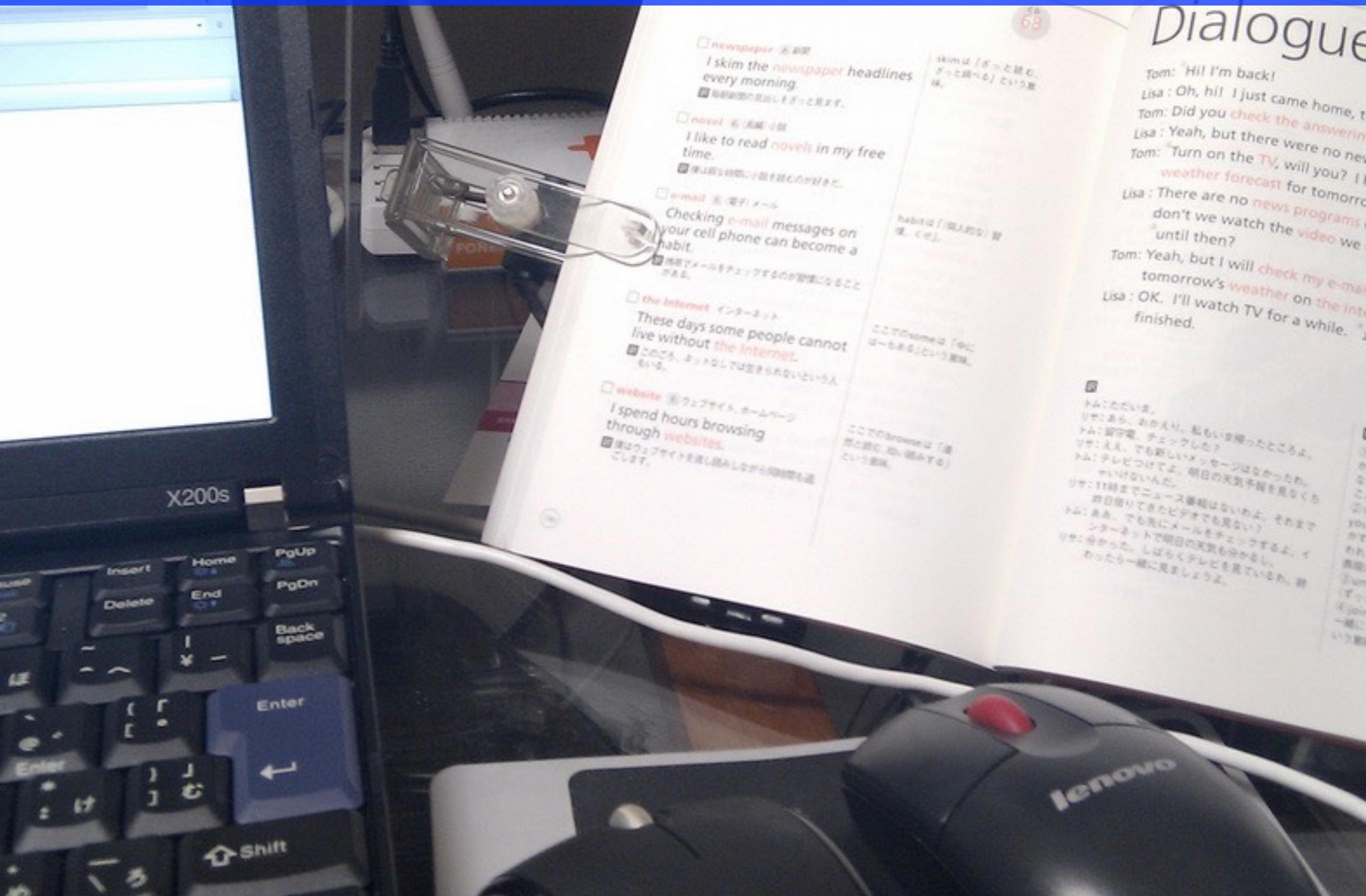
学び方を学ぶ

現役プログラマでいるために



おわりに

# 学び続ける姿勢



技術を学ぶので  
はなく、技術の  
学び方を学ぶ

誇りあるプロになってください



ご清聴ありがとうございました