エンジニアとして
この先生きのこるために

和田 卓人

Apr 03, 2020 @リクルートテクノロジーズ

和田卓人

Hatena

t-wada

t_wada

twada

監修、監訳、翻訳した本たち

テスト書いてますか！！

集中。。。 : 2011/12/06(火) 18:32:24.37 0

テスト書いてないとかお前それ
@t_wadaの前でも
同じこと言えんの?

スタンド名: ワイルド・サバンナ

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

よろしくお願いします

集中。。。 : 2011/12/06(火) 18:32:24.37 0

テスト書いてないとかお前それ
@t_wadaの前でも
同じこと言えんの?

97

プログラマが知るべき97のこと

97 Things Every Programmer Should Know

Kevlin Henney 編
和田 卓人 監修
夏目 大 訳

O'REILLY®
オライリー・ジャパン

The Pragmatic Programmer: From Journeyman to Master

新装版

達人プログラマー

職人から名匠への道

Andrew Hunt・David Thomas 共著
村上雅章 訳

The Pragmatic Programmer

Ohmsha

Takuto Wada
@t_wada

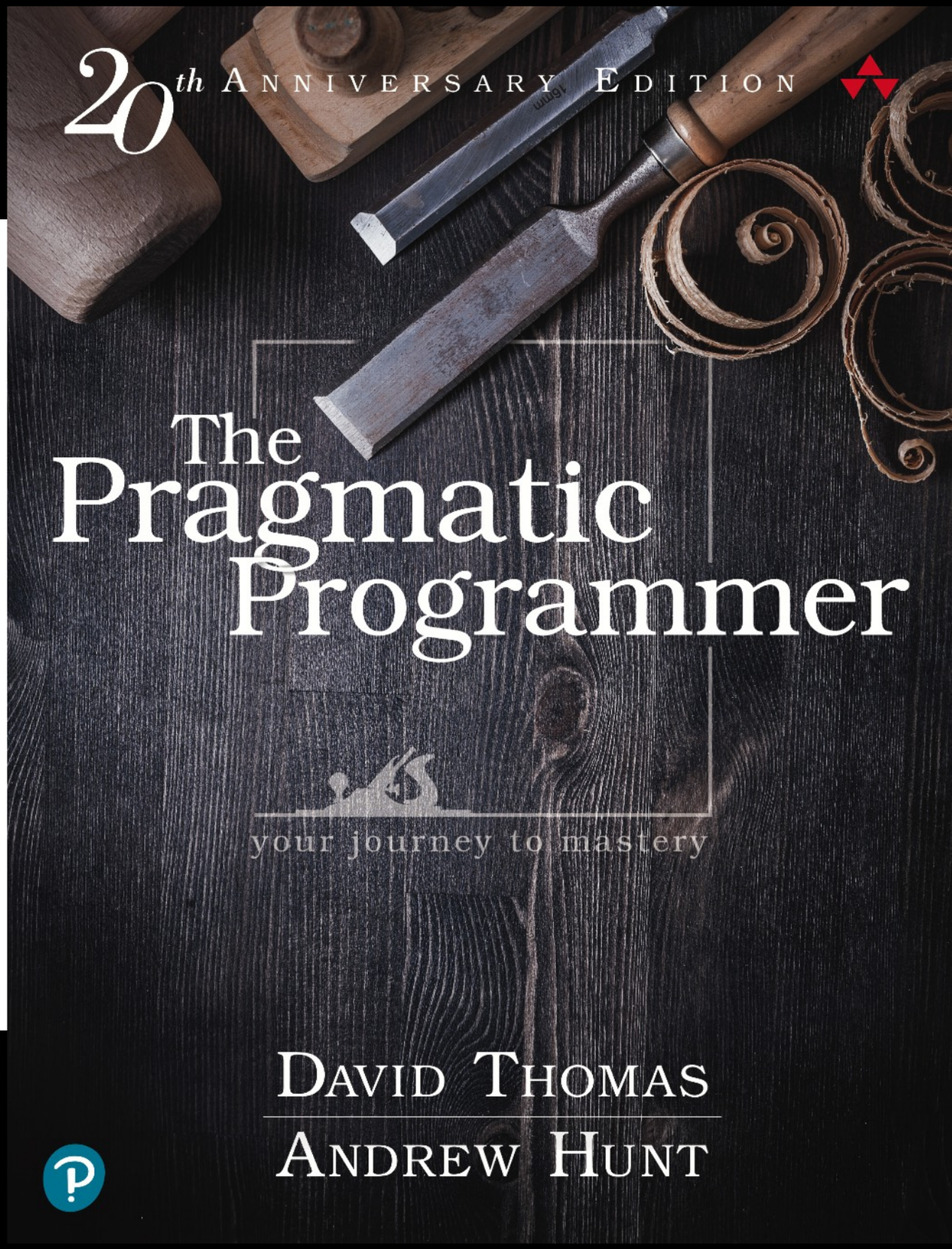
多くのプログラマの人生に影響を与えた『The Pragmatic Programmer』（邦訳『達人プログラマー』）が20年ぶりに改訂。1/3が新規追加の内容。既存部分もほとんどリライト。5月8日からbeta bookがPragmatic Bookshefで購入可 / “Coming Soon: The Pragmatic Programmer, 20th…” htn.to/XXURy7hXYF

Translate Tweet

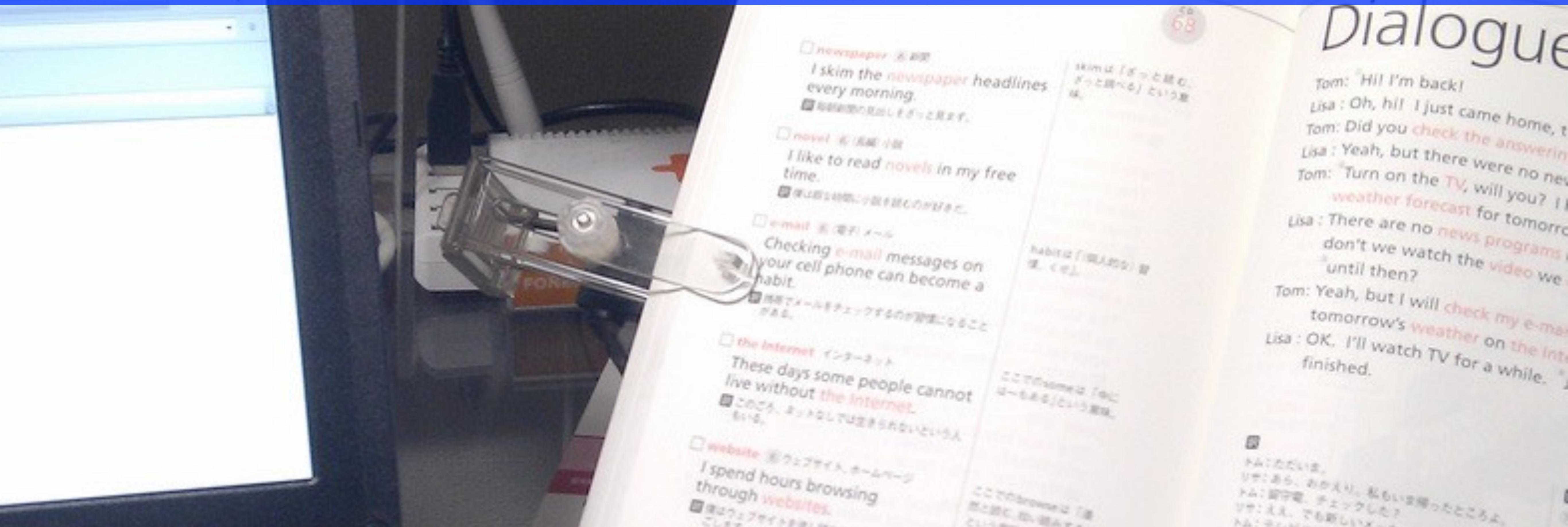
10:04 AM - 7 May 2019

292 Retweets **621** Likes

1 292 621



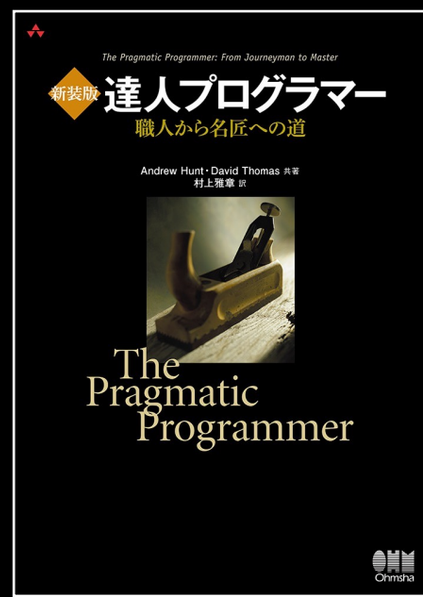
学び続ける姿勢




Dialogue
X200s
Home
PgUp
Insert
Delete
End
PgDn
Back space
Enter
Shift

97
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 編
和田 卓人 監修
夏目 大 訳

“常にあなたの知識ポートフォリオに投資すること”



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

👉 四半期毎に技術書を読む

手を動かして学ぶ

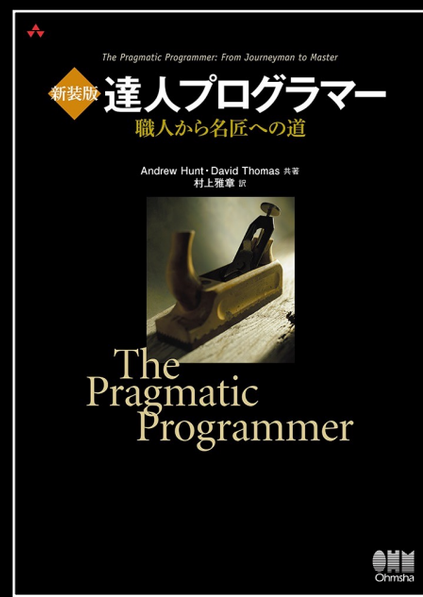
毎年少なくとも1つの言語を学習する

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

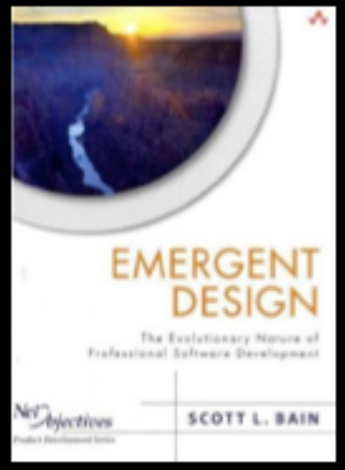
1. “四半期毎に

技術書を読む”




パターン、Wiki、XP
時を超えた創造の原則
デザインパターン
XP(エクストリームプログラミング)
Wiki
半世紀にわたる歴史物語
すべての起源は、建築家アレグザンダーにあった


テスト駆動開発入門
TDD: Test-Driven Development By Example
Kent Beck


EMERGENT DESIGN
The Evolutionary Nature of Professional Software Development
SCOTT L. BAIN


パターン指向リファクタリング入門
Refactoring to Patterns
ソフトウェア設計を改善する27の作法
デザインパターンを取り入れてプログラムを改善しよう


O'REILLY
プレファクタリング


ソフトウェアアーキテクチャ
PATTERN-ORIENTED SOFTWARE ARCHITECTURE
近代科学社


アジャイルソフトウェア開発の奥義
第2版
オブジェクト指向開発の神髄と匠の技


Java言語で学ぶ
リファクタリング入門
結城 浩
どうすれば、きれいで読みやすいコードになるのか?
Javaの実例を通して学ぶ
リファクタリング入門書の決定版


リファクタリング
プログラミングの体質改善テクニック
マーチン・ファウラー


デザインパターン
改訂版


プログラムデザインのための
パターン言語


Patterns of Enterprise Application Architecture
エンタープライズアプリケーションアーキテクチャパターン
エンタープライズアーキテクチャのレイヤ化とは?


アジャイルソフトウェア達人の技
Clean Code
クリーンコード
ASCII


達人プログラマー
システム開発の職人から名匠への道
The Pragmatic Programmer


The Pragmatic Programmers
達人プログラマー
ソフトウェア開発に不可欠な基礎知識
ASCII


プログラミング作法
A THE PRACTICE OF PROGRAMMING
PROGRAMMING
ASCII


Stephen P. Berczuk, Brad Appleton
Software Configuration Management Patterns:
パターンによるソフトウェア構成管理
いま、プロジェクトの現場は混乱に陥っている!!
プロジェクトの失敗を防ぐためには必見だ!!


新装版
AntiPatterns
アンチパターン
ソフトウェア危篤患者の救出


Code Reading
コードは小説より面白い!


Code Quality
コード・クオリティ

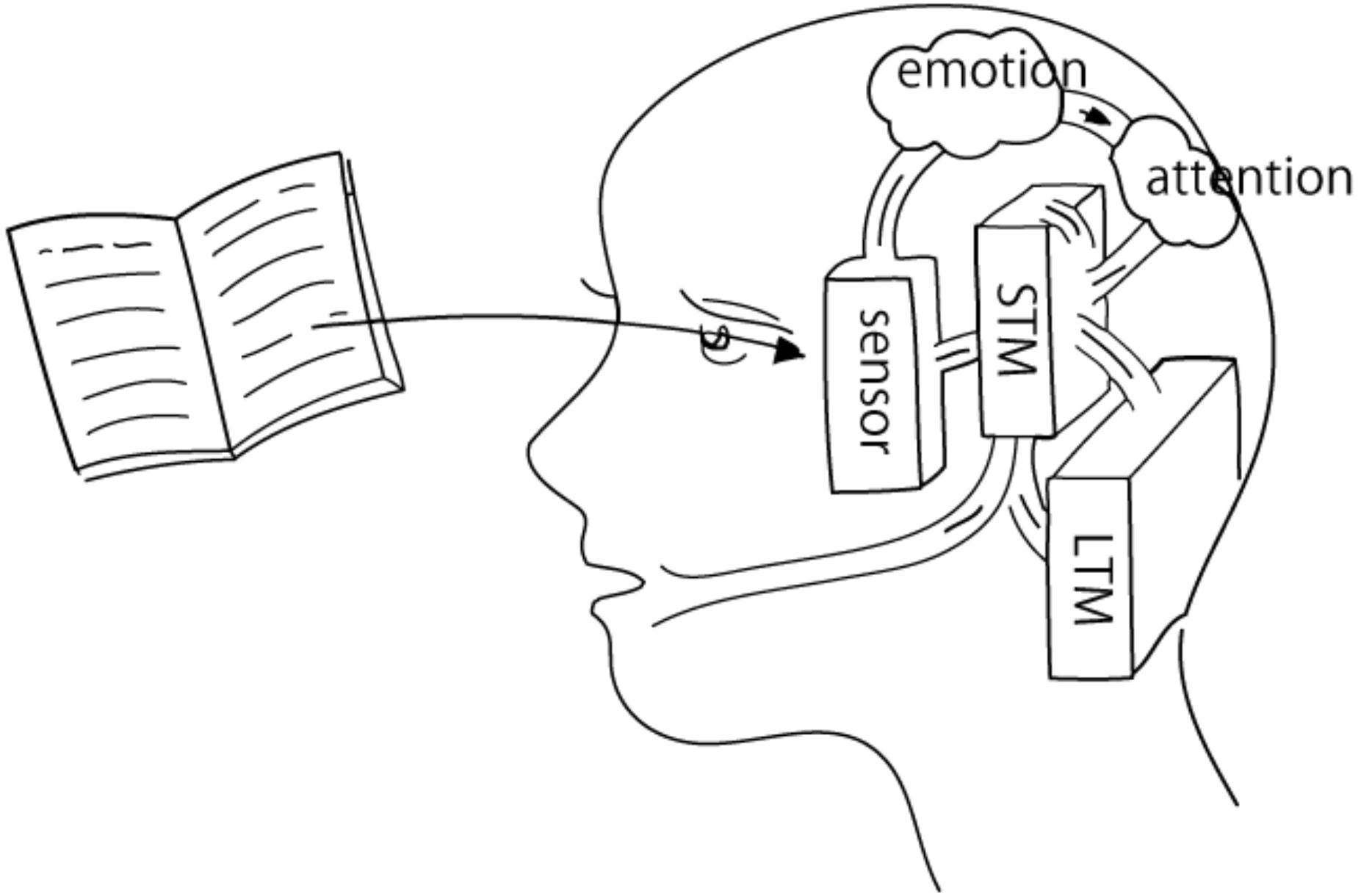

Beautiful Code
ビューティフルコード
O'REILLY


コードコンプリート
CODE COMPLETE
第2版
完全なプログラミングを目指して
Microsoft Press


ケント・ベックの Smalltalk
ベストプラクティス・パターン
シンプル・デザインへの宝石箱


実装パターン

学びの仕組み



感覚記憶
0.5 ～ 2sec

短期記憶
15 ～ 30 sec

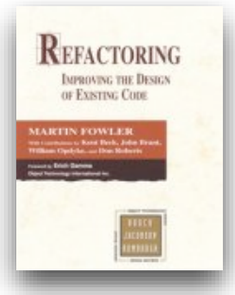
長期記憶
死ぬまで?

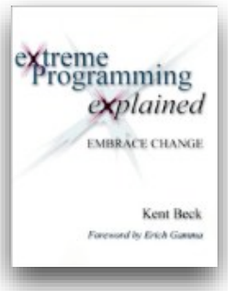
脳内インデックスを作る

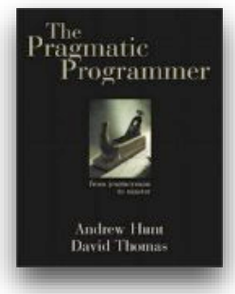
ピッカーを育てる ＝ 反復練習
何度も長期記憶から出し入れする

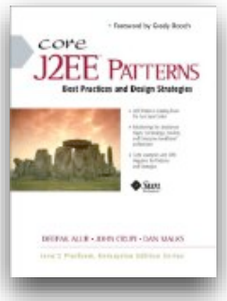
荷物を他の荷物とくっつける
連想記憶を育てる

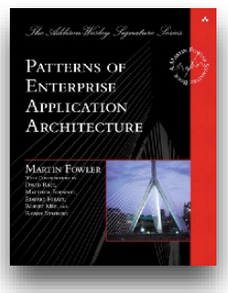
たとえば、時系列に並べる

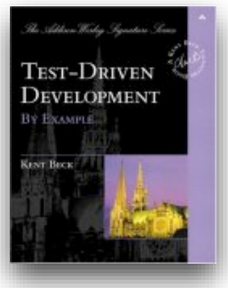


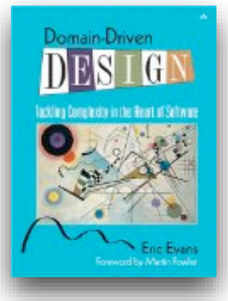


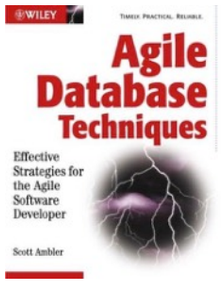


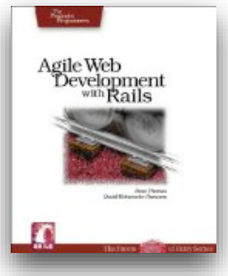


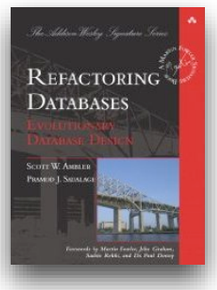


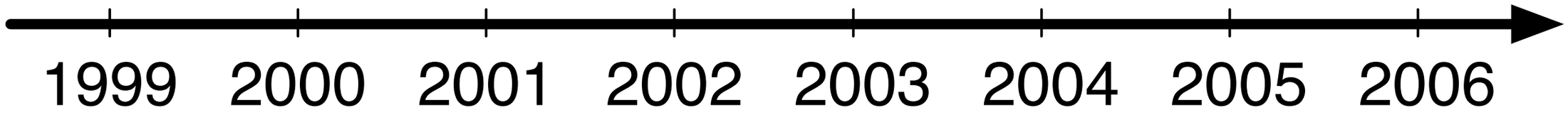












四半期毎に技術書を読む

👉 手を動かして学ぶ

毎年少なくとも1つの言語を学習する

身の回りをプログラミング対象にする

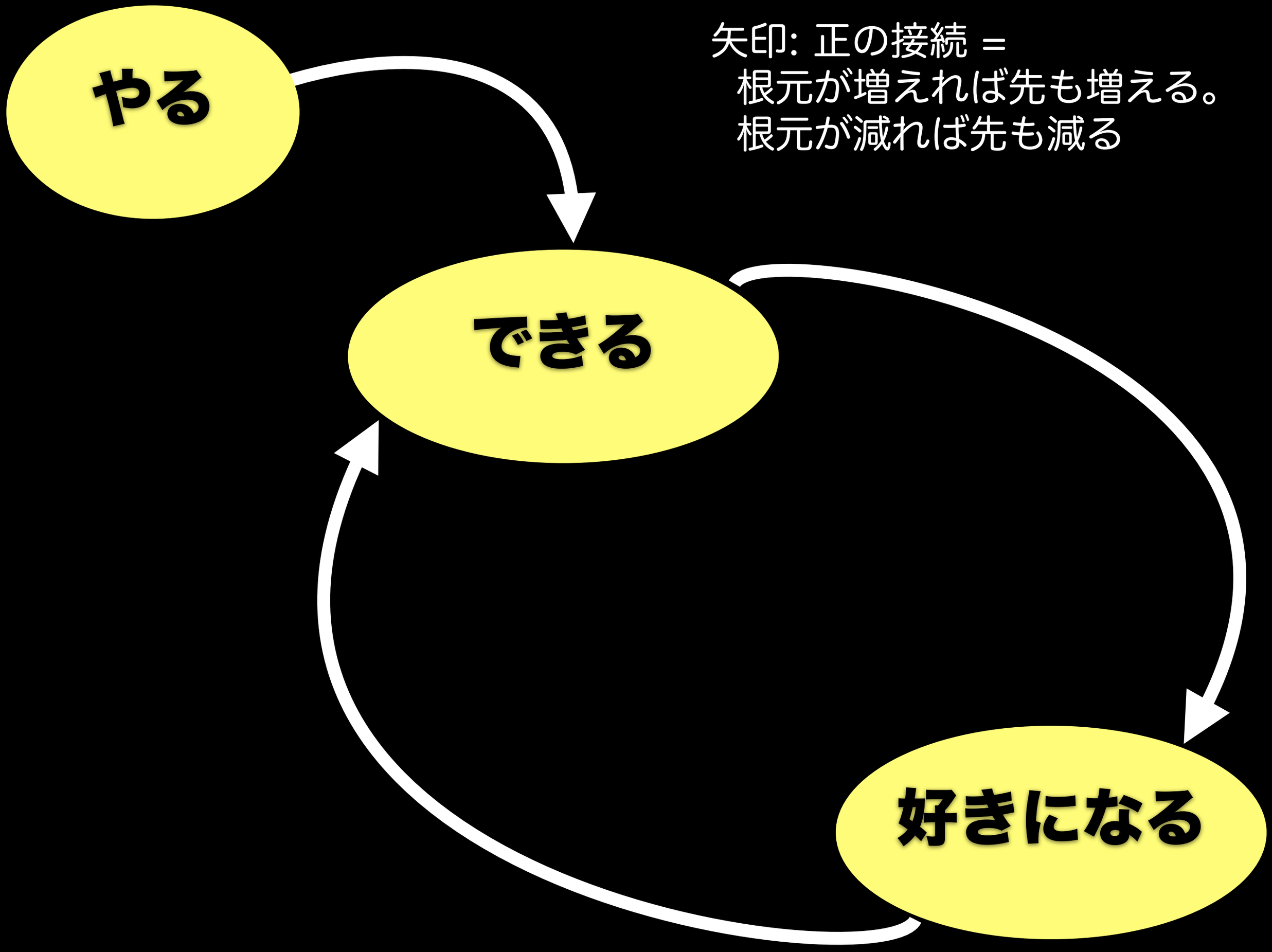
アウトプットを行う

2. “手を動かし

て学ぶ”


97
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 編
和田 卓人 監修
夏目 大 訳

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



デールの円錐

I see and I forget.
I hear and I remember.
I do and I understand.
— Confucius

After 2 weeks, we tend to remember ...

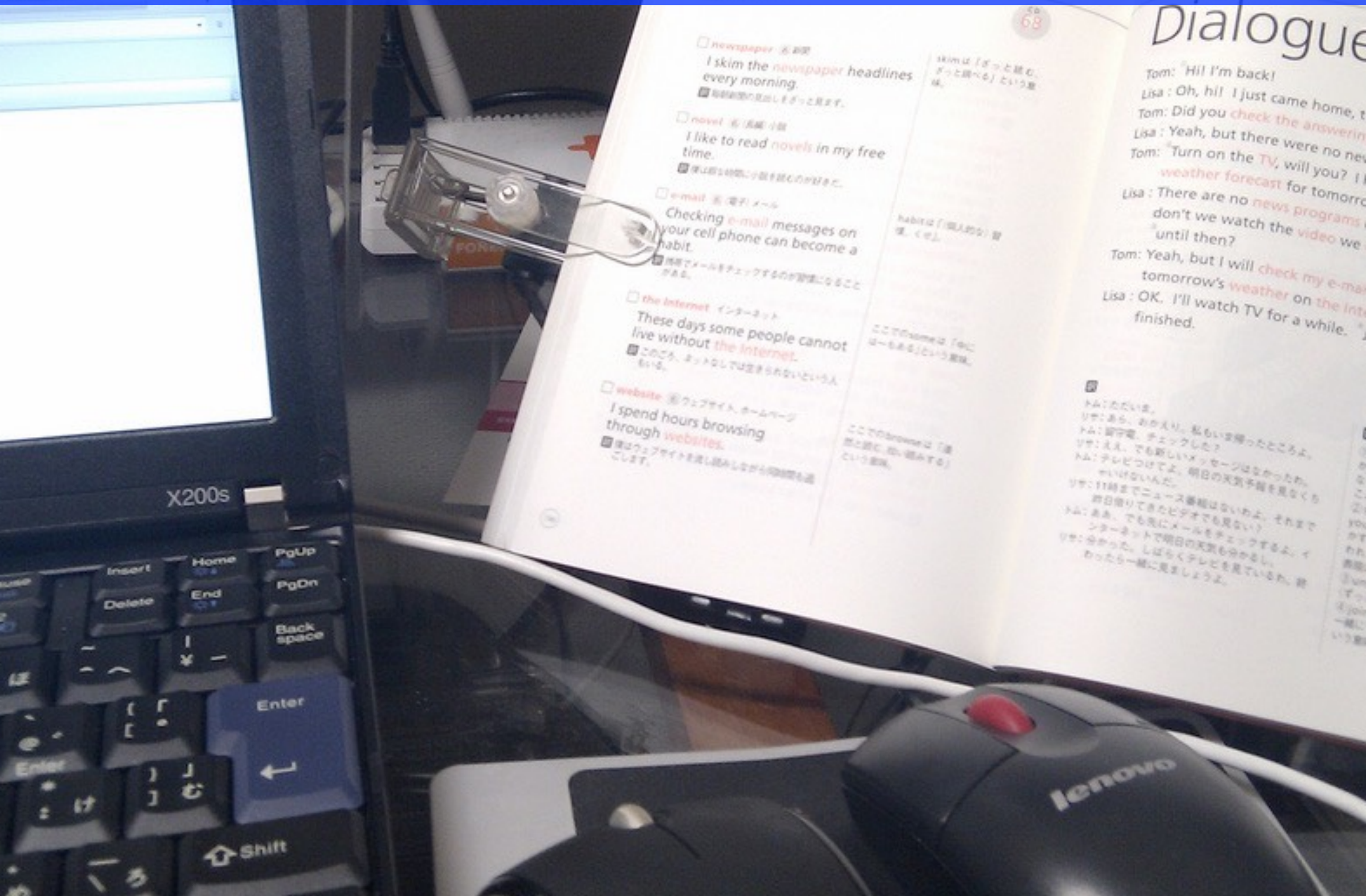
- 10% of what we READ
- 20% of what we HEAR
- 30% of what we SEE
- 50% of what we SEE & HEAR
- 70% of what we SAY
- 90% of what we SAY & DO

Passive

Active

Source: Edgar Dale (1969)

写経


newspaper 名 新聞
I skim the newspaper headlines every morning.
novel 名 長編小説
I like to read novels in my free time.
e-mail 名 電子メール
Checking e-mail messages on your cell phone can become a habit.
the Internet インターネット
These days some people cannot live without the Internet.
website 名 ウェブサイト、ホームページ
I spend hours browsing through websites.
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 ne
Tom: Turn on the TV, will you? I
weather forecast for tomorro
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 int
Lisa : OK. I'll watch TV for a while.
finished.
X200s
Insert
Home
PgUp
Delete
End
PgDn
Back space
Enter
Shift
lenovo

Takuto Wada
@t_wada

技術書の「写経」の方法。 1.ローカルで使えるSCMを用意 2.「ほんたった」などで対象の本を固定 3.ひたすらサンプルコードを写して実行 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

Reply Delete Favorite

50+ RETWEETS | 50+ FAVORITES

5:06 PM - 12 Feb 10 via Twitter · Embed this Tweet

http://twitter.com/t_wada/statuses/9000231741

四半期毎に技術書を読む

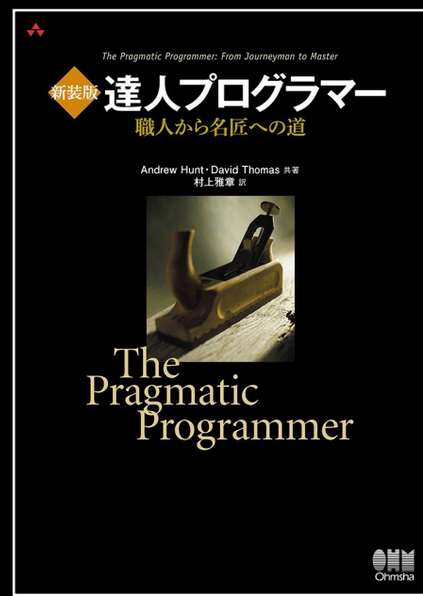
手を動かして学ぶ

👉 毎年少なくとも1つの言語を学習する

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. “毎年少なくとも一つの言語を学習する”



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。

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

Techniques | Tools | Platforms | Languages & Frameworks

Search | About the Radar | Build your Radar | Subscribe

ADOPT

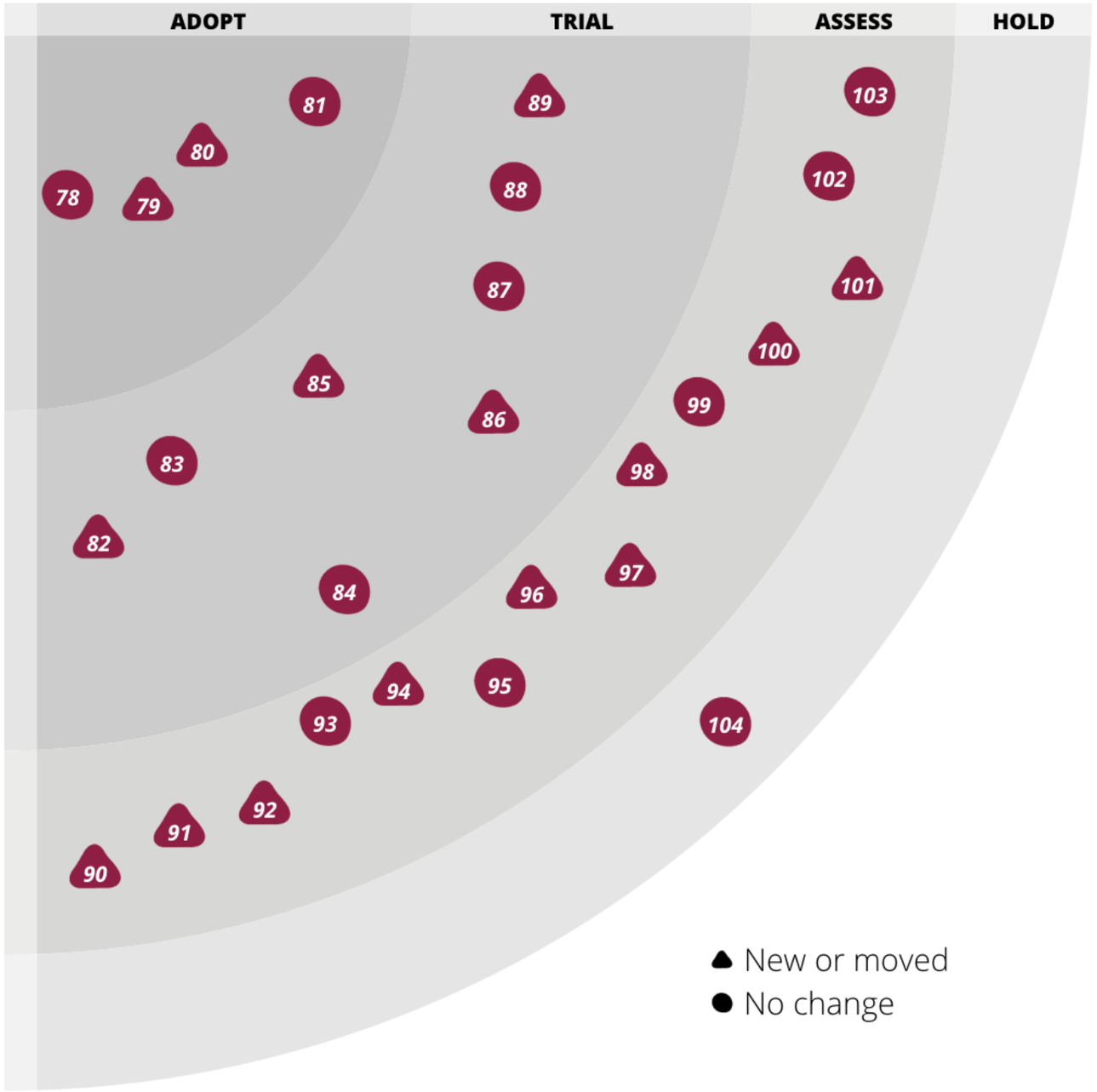
78. Ember.js
79. Python 3
80. ReactiveX
81. Redux

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

http://www.thoughtworks.com/radar/languages-and-frameworks

TECHNOLOGY RADAR

Techniques | Tools | Platforms | Languages & Frameworks

Search | About the Radar | Build your Radar | Subscribe

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

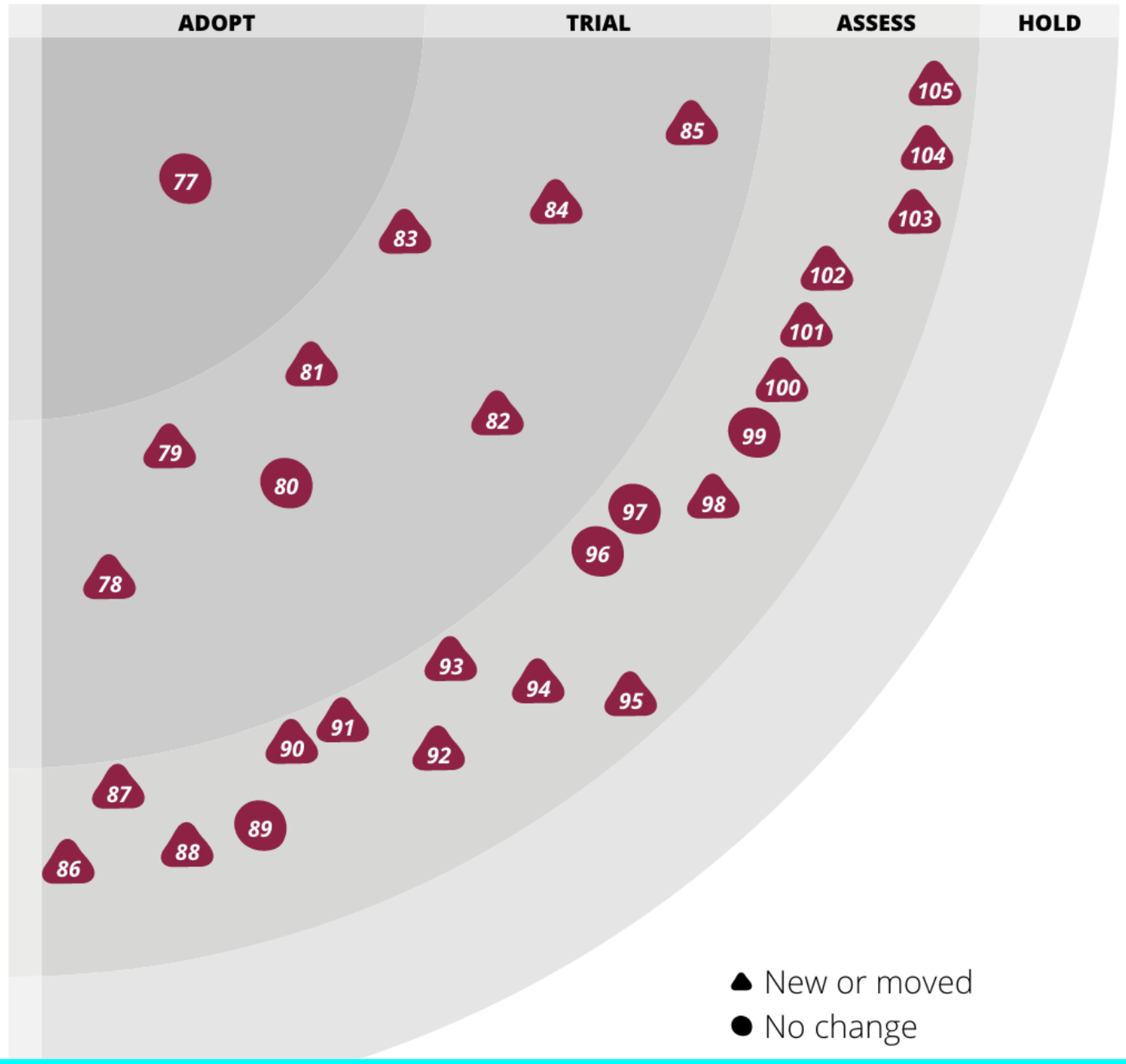
77. Python 3

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



http://www.thoughtworks.com/radar/languages-and-frameworks

TECHNOLOGY RADAR

Techniques | Tools | Platforms | Languages & Frameworks

Search | About the Radar | Build your Radar | Subscribe

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

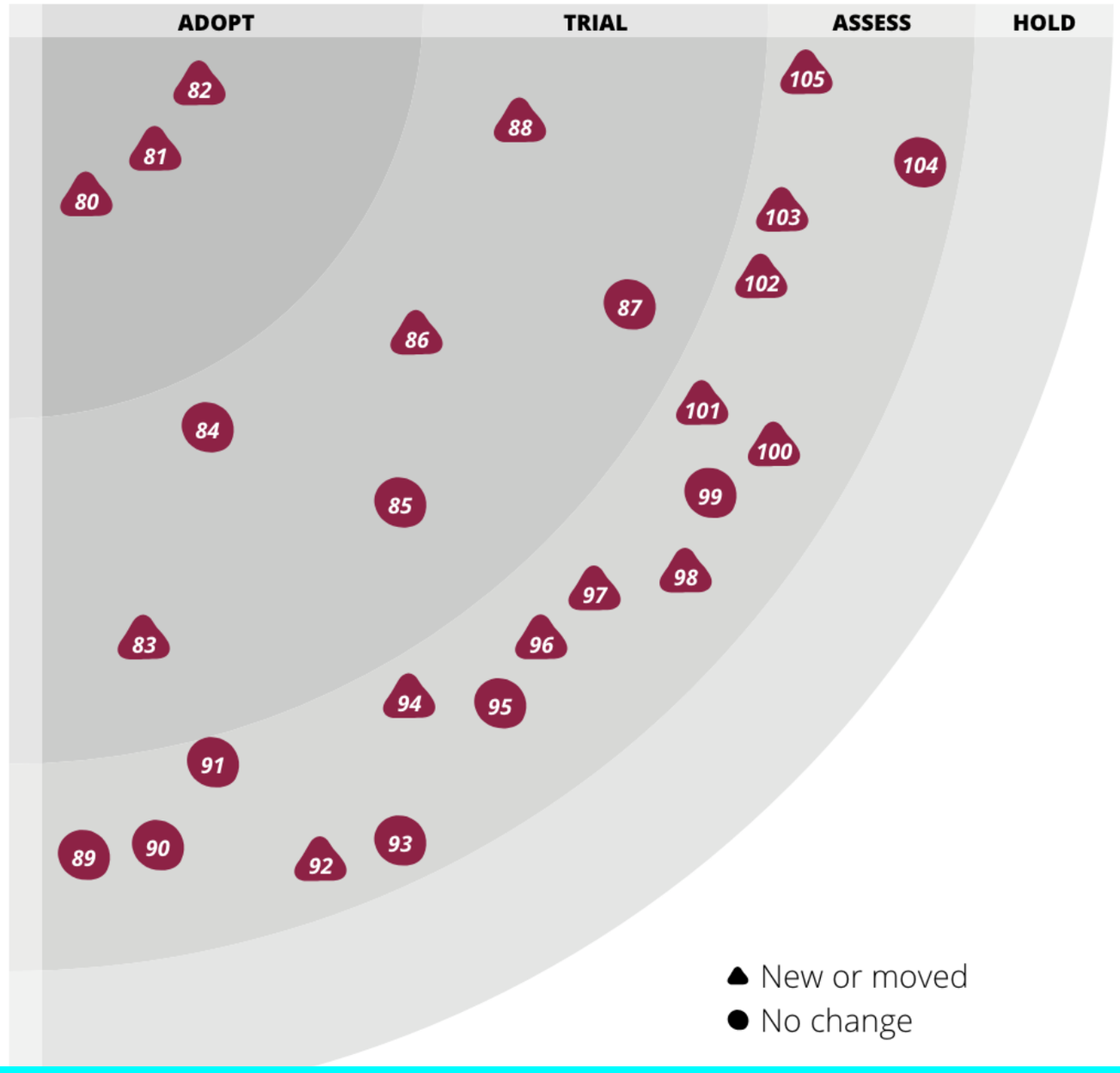
80. AssertJ
81. Enzyme
82. Kotlin

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



http://www.thoughtworks.com/radar/languages-and-frameworks

The information in our interactive Radar is currently only available in English. To get information in your native language, please download the PDF here.

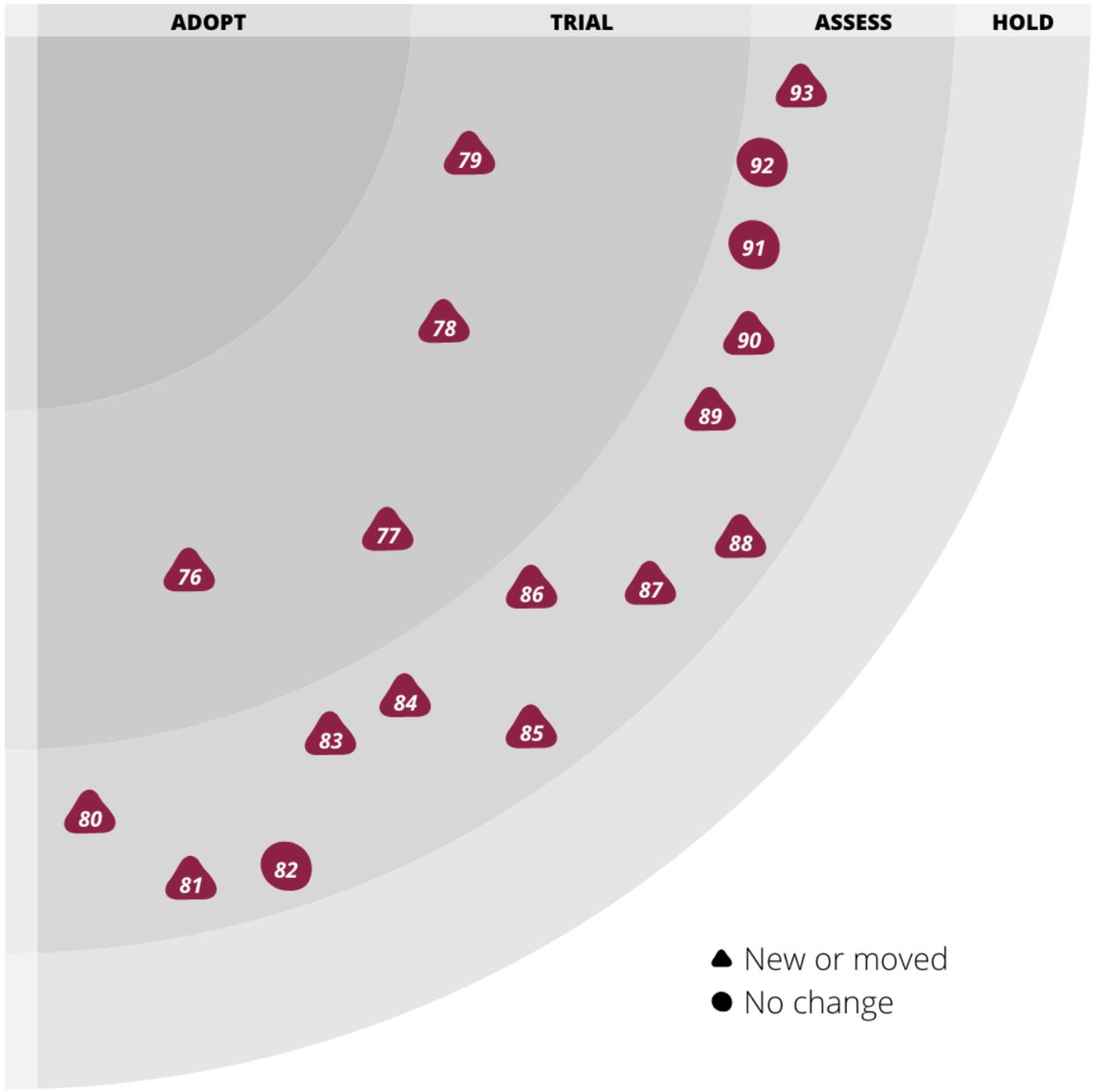
ADOPT

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



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

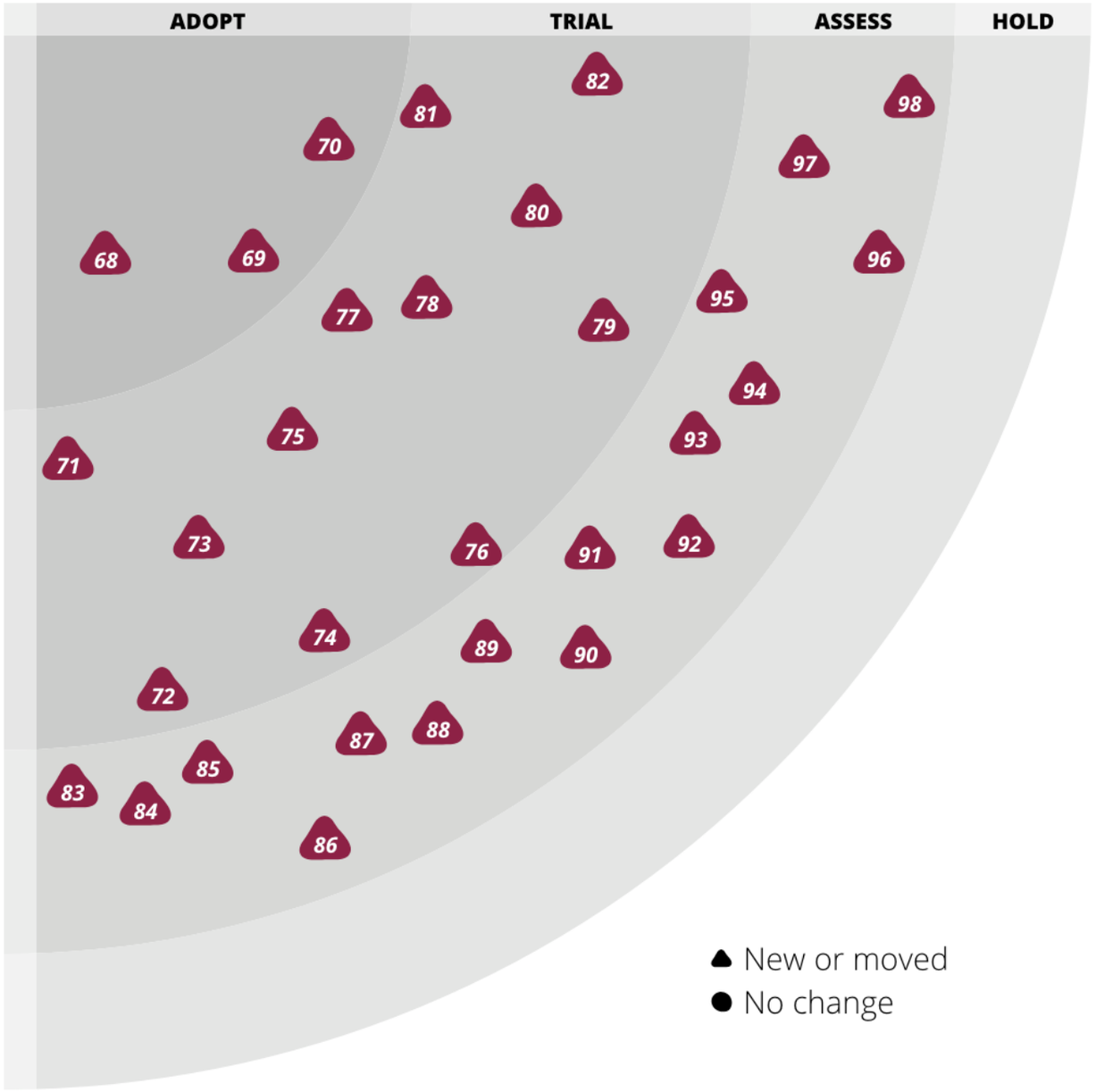
68. Apollo
69. MockK
70. TypeScript

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



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.

http://www.thoughtworks.com/radar/languages-and-frameworks

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

http://blogos.com/article/85541/

四半期毎に技術書を読む

手を動かして学ぶ

毎年少なくとも1つの言語を学習する

👉 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

プログラマ向けの本の監修者はどうあるべきか

Kevlin Henney 編
和田 卓人 監修
夏目 大 訳

O'REILLY®
オライリー・ジャパン

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

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

個々のエッセイのステータス管理

原文、翻訳原稿、監修原稿へのリンク

翻訳原稿と監修原稿の差分、レビュー後の差分

分別のある行動	Seb Rose	001	原文	翻訳	監訳	差分	N/A	編集済にする
関数型プログラミングを学ぶことの重要性	Edward Garson	002	原文	翻訳	監訳	差分	編集後差分	編集済にする
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	原文	翻訳	監訳	差分	編集後差分	編集済にする
コーディング規約を自動化する	Filip van Laenen	004	原文	翻訳	監訳	差分	編集後差分	編集済にする
美はシンプルさに宿る	Jørn Ølmheim	005	原文	翻訳	監訳	差分	N/A	編集済にする
リファクタリングの際に注意すべきこと	Rajith Attapattu	006	原文	翻訳	監訳	差分	編集後差分	編集済にする
共有は慎重に	Udi Dahan	007	原文	翻訳	監訳	差分	編集後差分	編集済にする
ボーイスカウト・ルール	Robert C. Martin（アンクル・ボブ）	008	原文	翻訳	監訳	差分	編集後差分	編集済にする
他人よりまず自分を疑う	Allan Kelly	009	原文	翻訳	監訳	差分	N/A	編集済にする
ツールの選択は慎重に	Giovanni Asproni	010	原文	翻訳	監訳	差分	N/A	編集済にする
ドメインの言葉を使ったコード	Dan North	011	原文	翻訳	監訳	差分	編集後差分	編集済にする
コードは設計である	Ryan Brush	012	原文	翻訳	監訳	差分	編集後差分	編集済にする
コードレイアウトの重要性	Steve Freeman	013	原文	翻訳	監訳	差分	N/A	編集済にする
コードレビュー	Mattias Karlsson	014	原文	翻訳	監訳	差分	編集後差分	編集済にする
コードの論理的検証	Yechiel Kimchi	015	原文	翻訳	監訳	差分	N/A	編集済にする
コメントについてのコメント	Cal Evans	016	原文	翻訳	監訳	差分	N/A	編集済にする
コードに書けないことのみをコメントにする	Kevlin Henney	017	原文	翻訳	監訳	差分	編集後差分	編集済にする
学び続ける姿勢	Clint Shank	018	原文	翻訳	監訳	差分	N/A	編集済にする
「便利さ」は「利便性」ではない	Gregor Hohpe	019	原文	翻訳	監訳	差分	N/A	編集済にする
すばやくデプロイ、こまめにデプロイ	Steve Berczuk	020	原文	翻訳	監訳	差分	N/A	編集済にする
技術的例外とビジネス例外を明確に区別する	Dan Bergh Johnsson	021	原文	翻訳	監訳	差分	N/A	編集済にする
1万時間の訓練	Jon Jagger	022	原文	翻訳	監訳	差分	編集後差分	編集済にする
ドメイン特化言語	Michael Hunger	023	原文	翻訳	監訳	差分	N/A	編集済にする

見られて恥ずかしいデータは~~入力しない~~ 使わないこと
==========

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、英国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、同じく英国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たのです。

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」~~をしたくなるものですが、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るのです。

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うこともあります~~ 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~~触れることはなかったものである。しかし、その改造によって~~ 触れること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う。
* 誰かが、テスト用の管理インタフェースと、製品版の管理インタ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。

「好事門を出でず、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 = 「実に実にひどい、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、何度か使ってしまっています・・・）。~~

~~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、~~ ~~あるいはログ~~ ソースコードに卑猥な言葉、~~ダイアログ~~ 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~~ 実に実にひどい、~~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

四半期毎に技術書を読む

手を動かして学ぶ

毎年少なくとも1つの言語を学習する

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. アウトプットを行う

デールの円錐

I see and I forget.
I hear and I remember.
I do and I understand.
— Confucius

After 2 weeks, we tend to remember ...

- 10% of what we READ
- 20% of what we HEAR
- 30% of what we SEE
- 50% of what we SEE & HEAR
- 70% of what we SAY
- 90% of what we SAY & DO

Passive

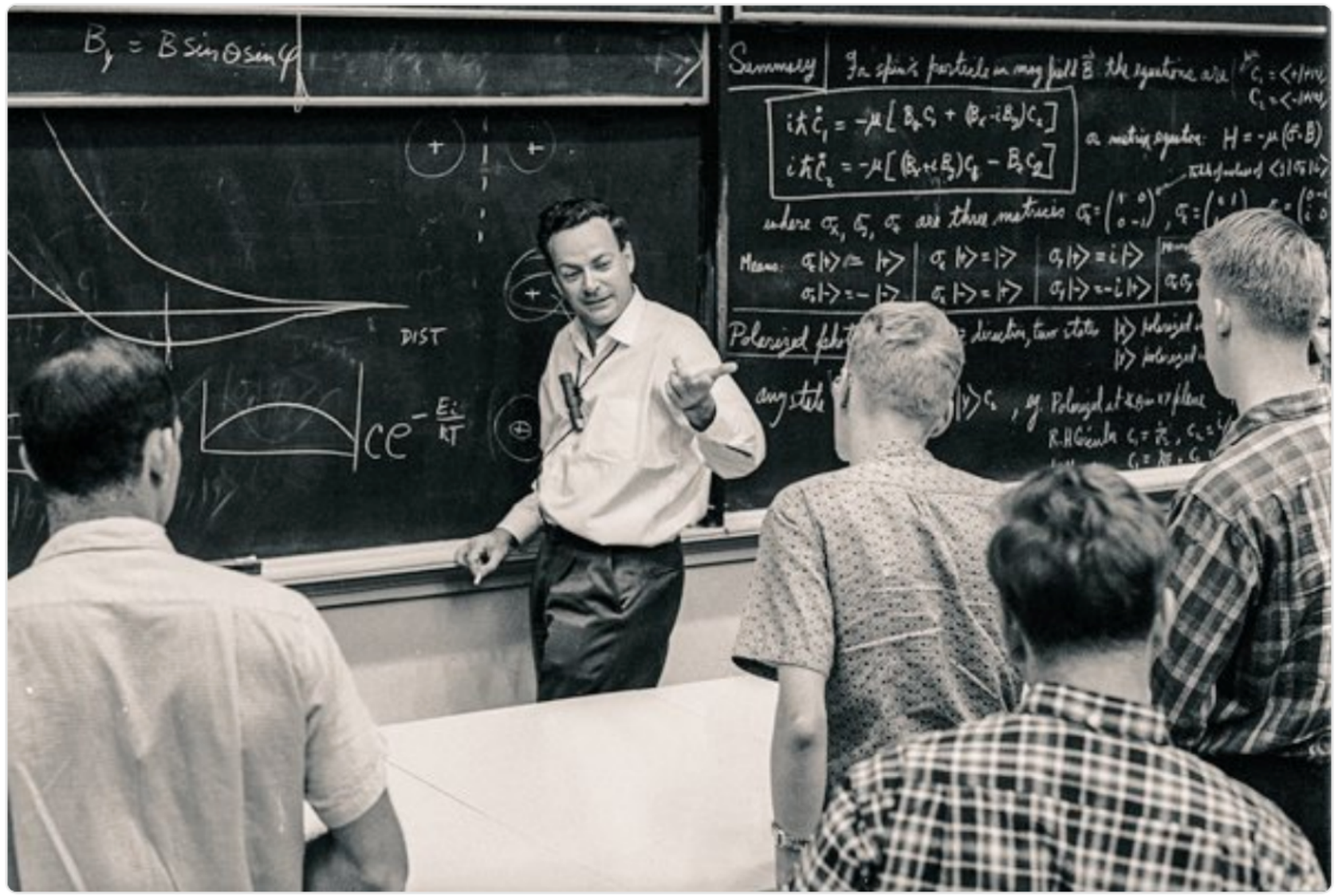
Active

Source: Edgar Dale (1969)

Richard Feynman
@ProfFeynman

Follow

If you want to master something, teach it.



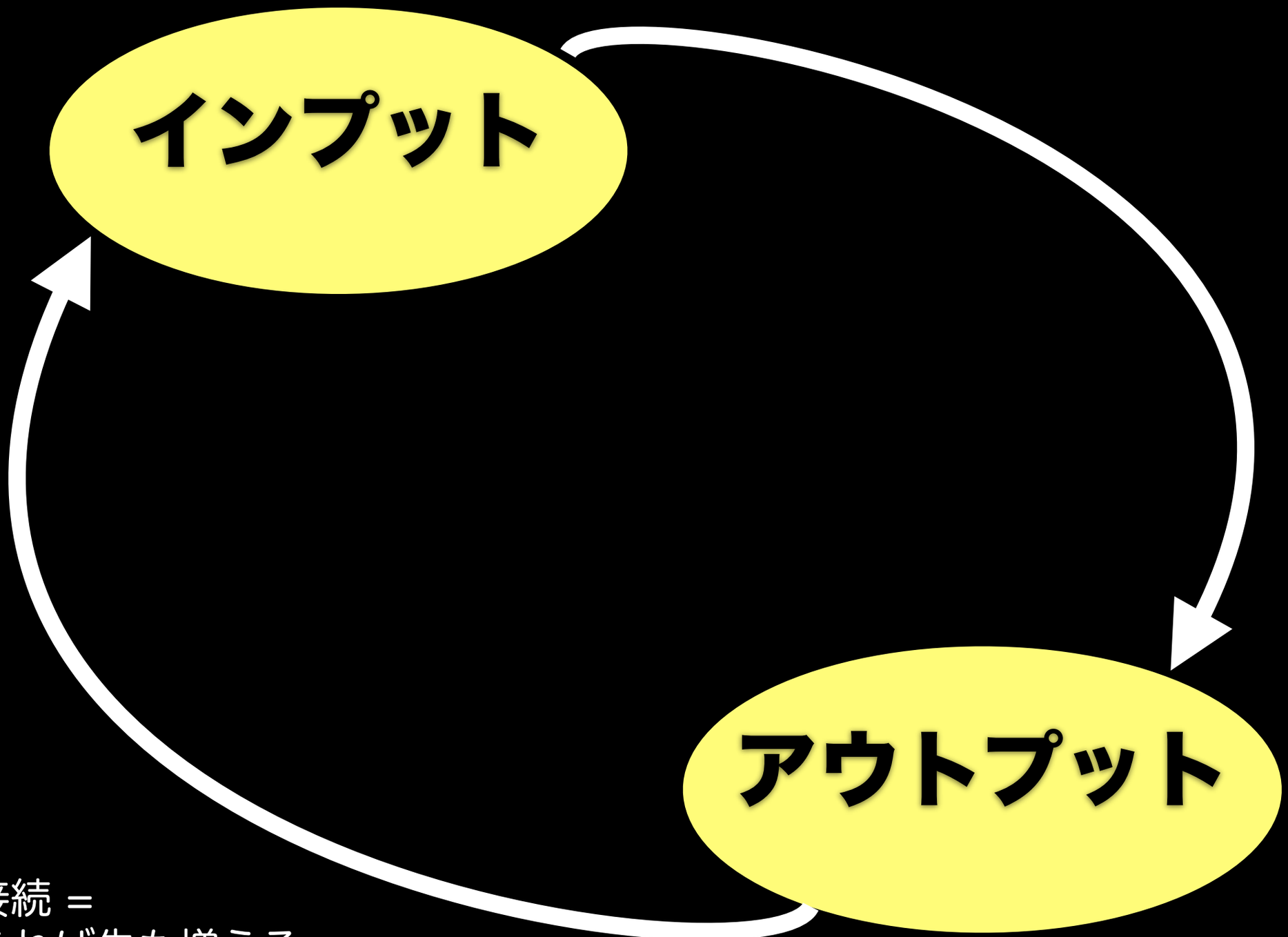
2:15 AM - 6 Apr 2018

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

http://d.hatena.ne.jp/KZR/20080808/p1

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ら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「掟」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てきたものです。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）

gihyo.jpの連載

『［動画で解説］和田卓人の“テスト駆動開発”講座』

http://gihyo.jp/dev/serial/01/tdd/

全20回すべて動画付き解説

ニコニコ動画でも見れます

WEB+DB過去記事の特設サイトと動画も

実演！ムービーもWeb配信 テスト駆動開発

WEB+DB PRESS Vol.37 特集 特設ページ 実演！リファクタリング

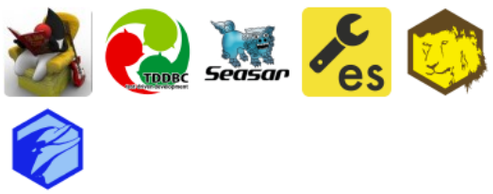
コードを公開する

Takuto Wada
twada

Block or report user

Towersquest Inc.
Tokyo, Japan
http://tako3.com/http://github…

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

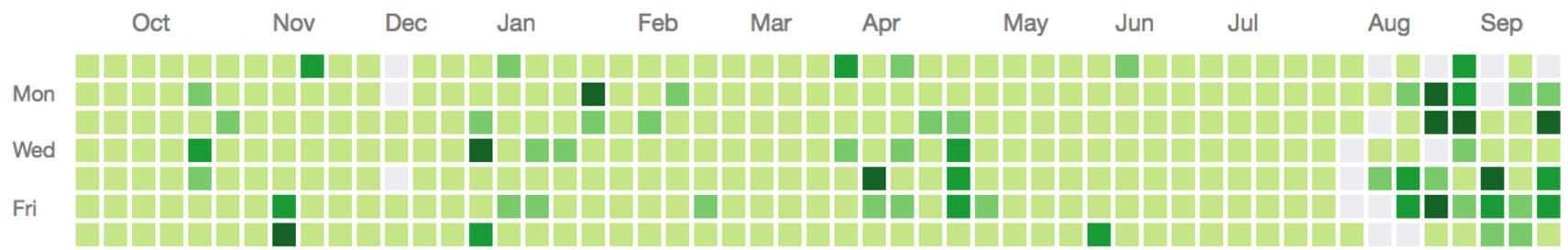
licensify
Browserify plugin to prepend license header to your bundle
JavaScript ★ 142 9

qunit-tap
A TAP Output Producer Plugin for QUnit
JavaScript ★ 73 14

type-name
Just a reasonable `typeof`
JavaScript ★ 65 4

coverlay.el
Test coverage overlay for Emacs
Emacs Lisp ★ 47 6

1,695 contributions in the last year



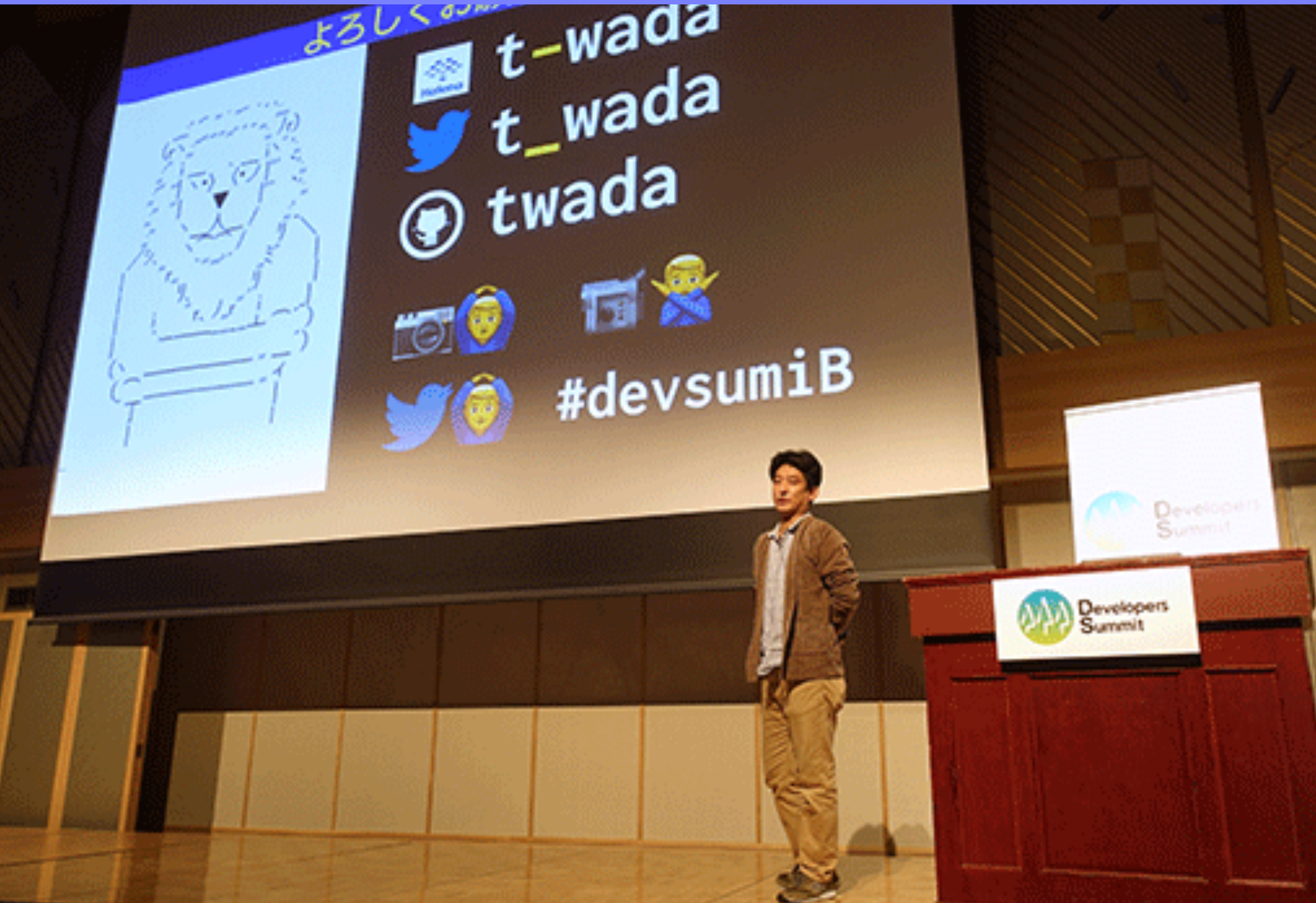
power-assert

```
assert(typeof item.id === 'strong')
       |      |    |  |
       |      |    |  false
       |      |    "foo"
       |      Item{id:"foo",name:"bar"}
       "string"
```

```
--- [string] 'strong'
+++ [string] typeof item.id
@@ -1,6 +1,6 @@
 str
-o
+i
 ng
```

https://github.com/power-assert-js/power-assert

講演する



ライブコーディングが最もハイリスクハイリターン

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

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ダ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……

John Resig
Contact, Subscribe

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。

http://ejohn.org/blog/write-code-every-day/

(当時の) @jeresig の github profile

John Resig
jeresig

Block or report user

@Khan
Brooklyn, NY
jeresig@gmail.com
http://ejohn.org/

Organizations

Overview | Repositories 91 | Stars 377 | Followers 13.3k | Following 29

Pinned repositories

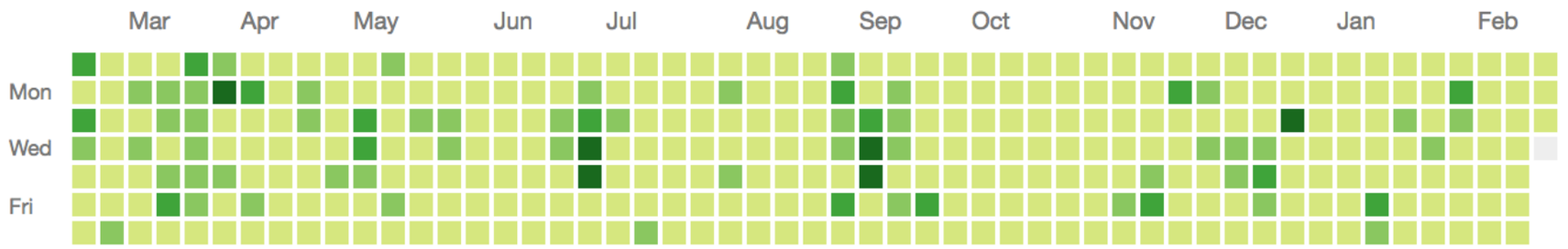
mongaku/mongaku
An image similarity search engine and database.
JavaScript ★ 18 2 1

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

910 contributions in the last year



Learn how we count contributions.
Less More

https://github.com/jeresig

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、バランスを取るということ

- まわりからの理解: 「毎日コードを書く」という習慣を公言したことで、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」

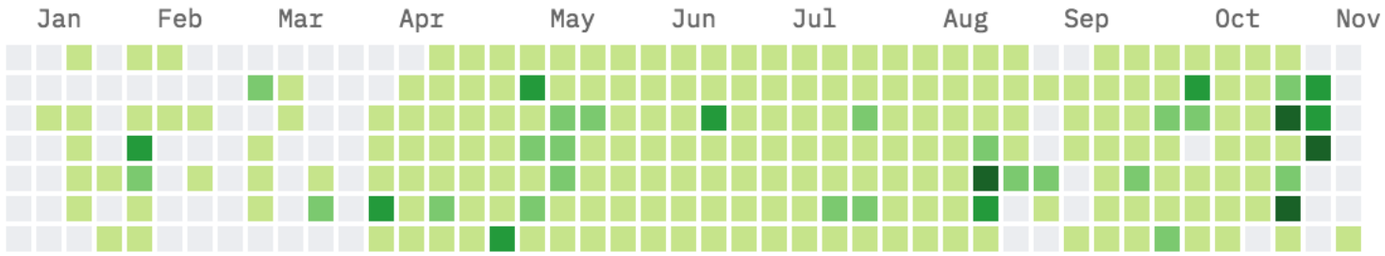
イチローが2004年にNBA年間最多安打を更新したときの言葉

私も結構続けました

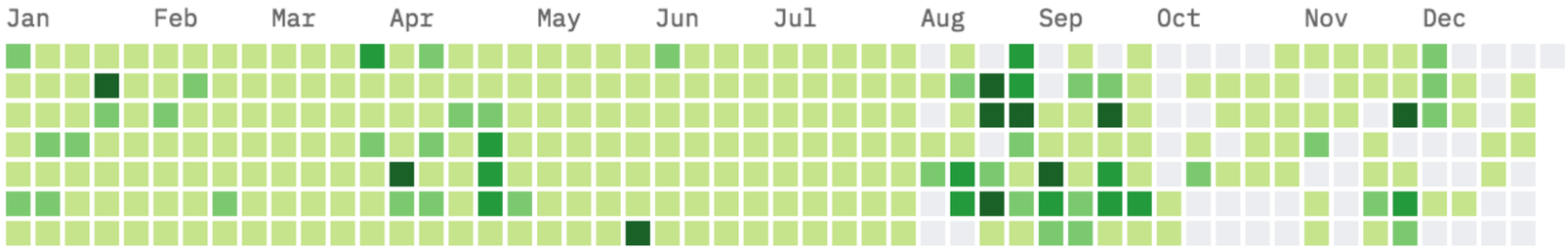
@twada on GitHub

Less More

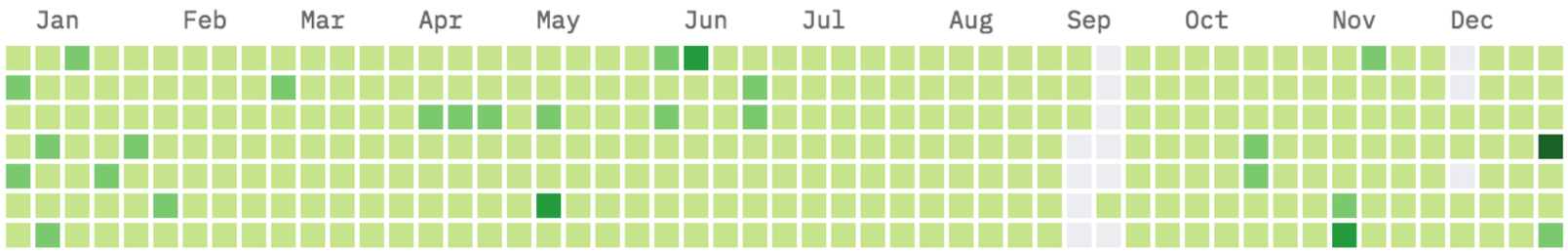
2018: 1,290 Contributions (so far)



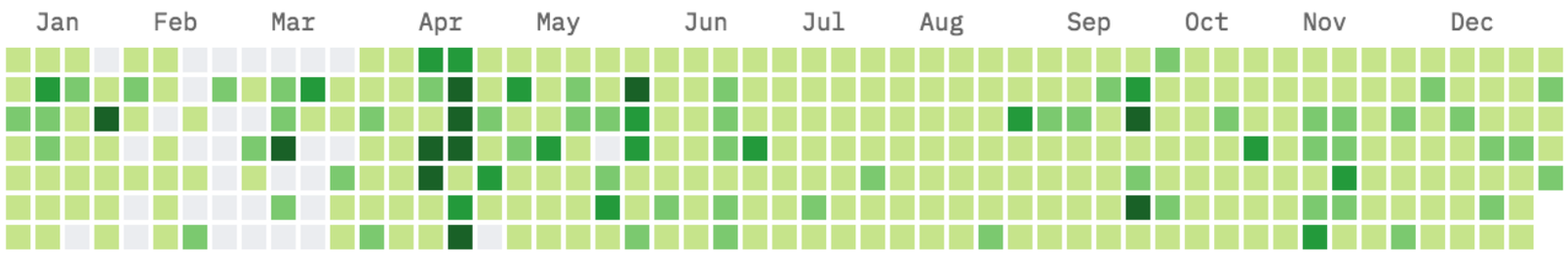
2017: 1,583 Contributions



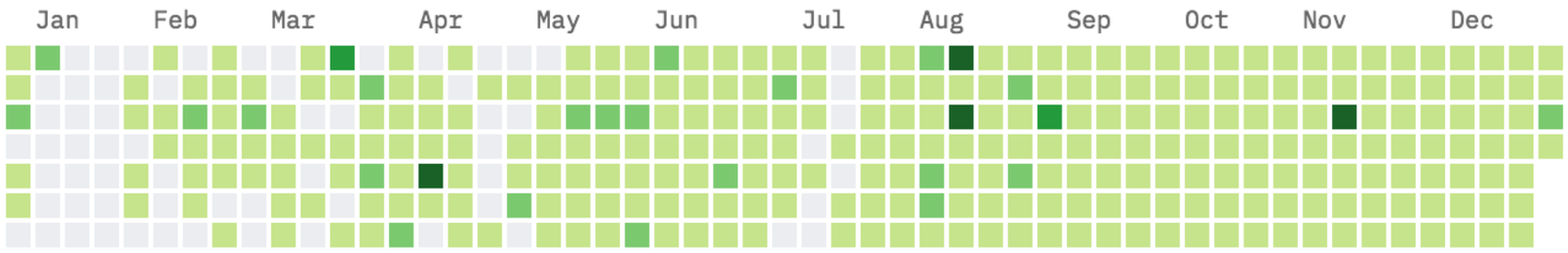
2016: 1,616 Contributions



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

N YONASHIRO

or

ne

Blo

趣味

4月に@t_wadaさんの講義を聞いてから`write code every day`を実践していて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.

unofficial Active Directory Authentication Library for golang

Go ★ 6

A Python3 implementation of the Wiener attack on RSA

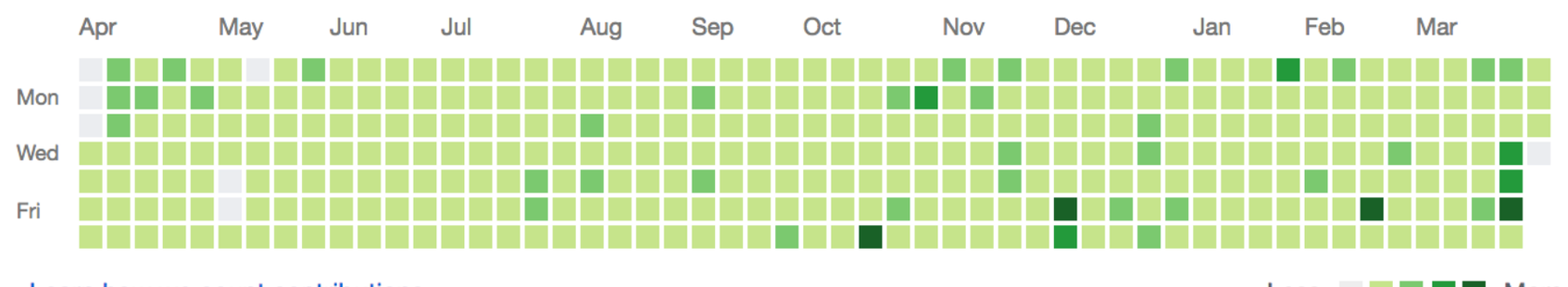
Python ★ 6 1

Recruit Technologies Co.,Ltd.

Tokyo, Japan

owan.orisano@gmail.com

1,754 contributions in the last year



Learn how we count contributions.

Less More

現在の @jeresig の github profile

John Resig
jeresig

Principal Architect at **@Khan**

@Khan
Brooklyn, NY
Sign in to view email
https://johnresig.com/

★ PRO

Block or report user

Organizations

Overview | Repositories 107 | Projects 0 | Stars 578 | Followers 15.7k | Following 29

Pinned

mongaku/mongaku
An image similarity search engine and database.
JavaScript ★ 55 9

node-stream-playground
Explore Node.js streams with an interactive playground.
JavaScript ★ 460 42

node-romaji-name
Normalize and fix common issues with Romaji-based Japanese names.
JavaScript ★ 22 3

node-yearrange
Node module for converting year range strings into usable dates.
JavaScript ★ 19 1

665 contributions in the last year

2020 2019 2018 2017 2016 2015 2014

Jun Jul Aug Sep Oct Nov Dec Jan Feb Mar

Learn how we count contributions. Less More

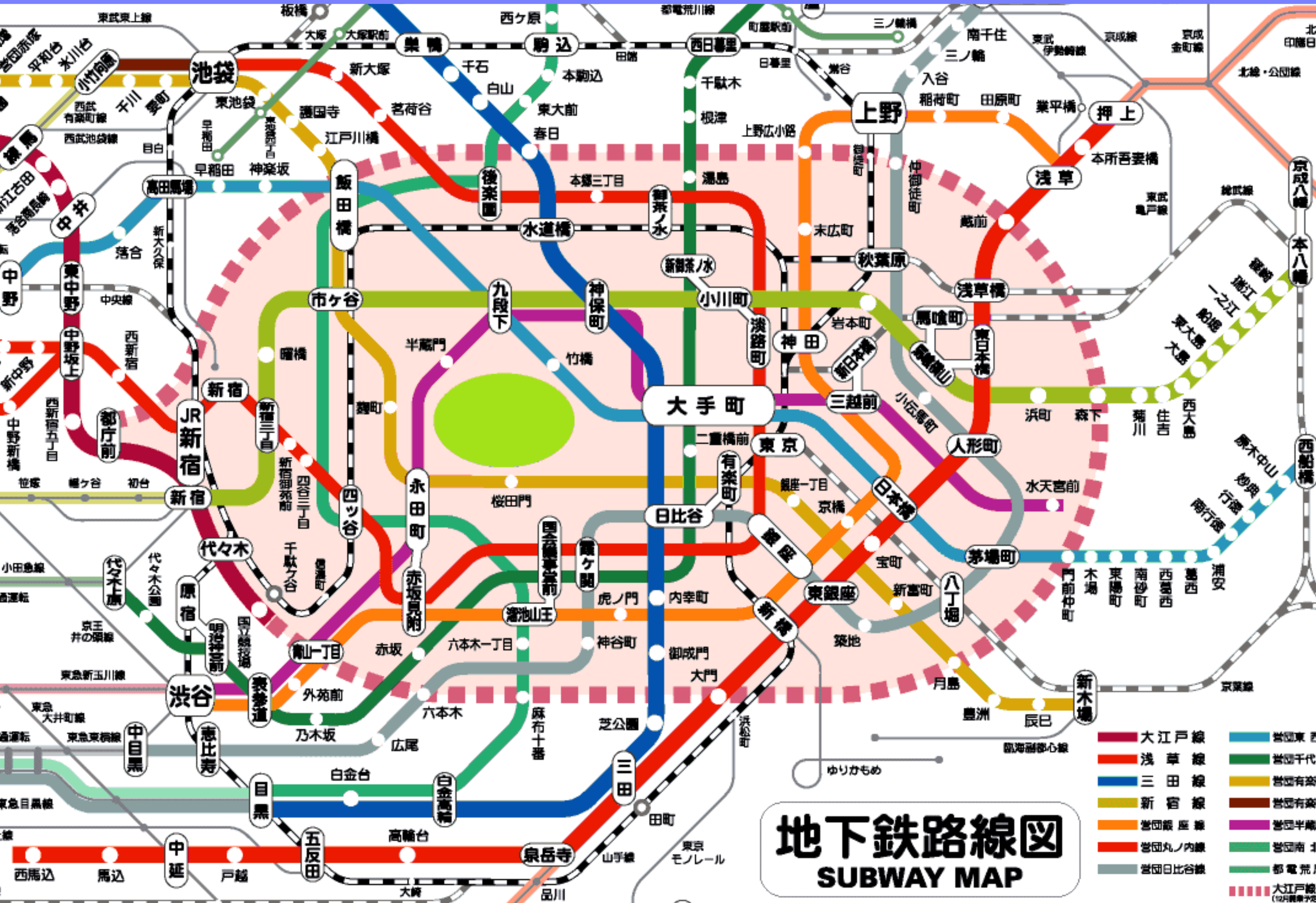
@Khan @mongaku @webpack

Activity overview

32%
Code review

https://github.com/jeresig

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

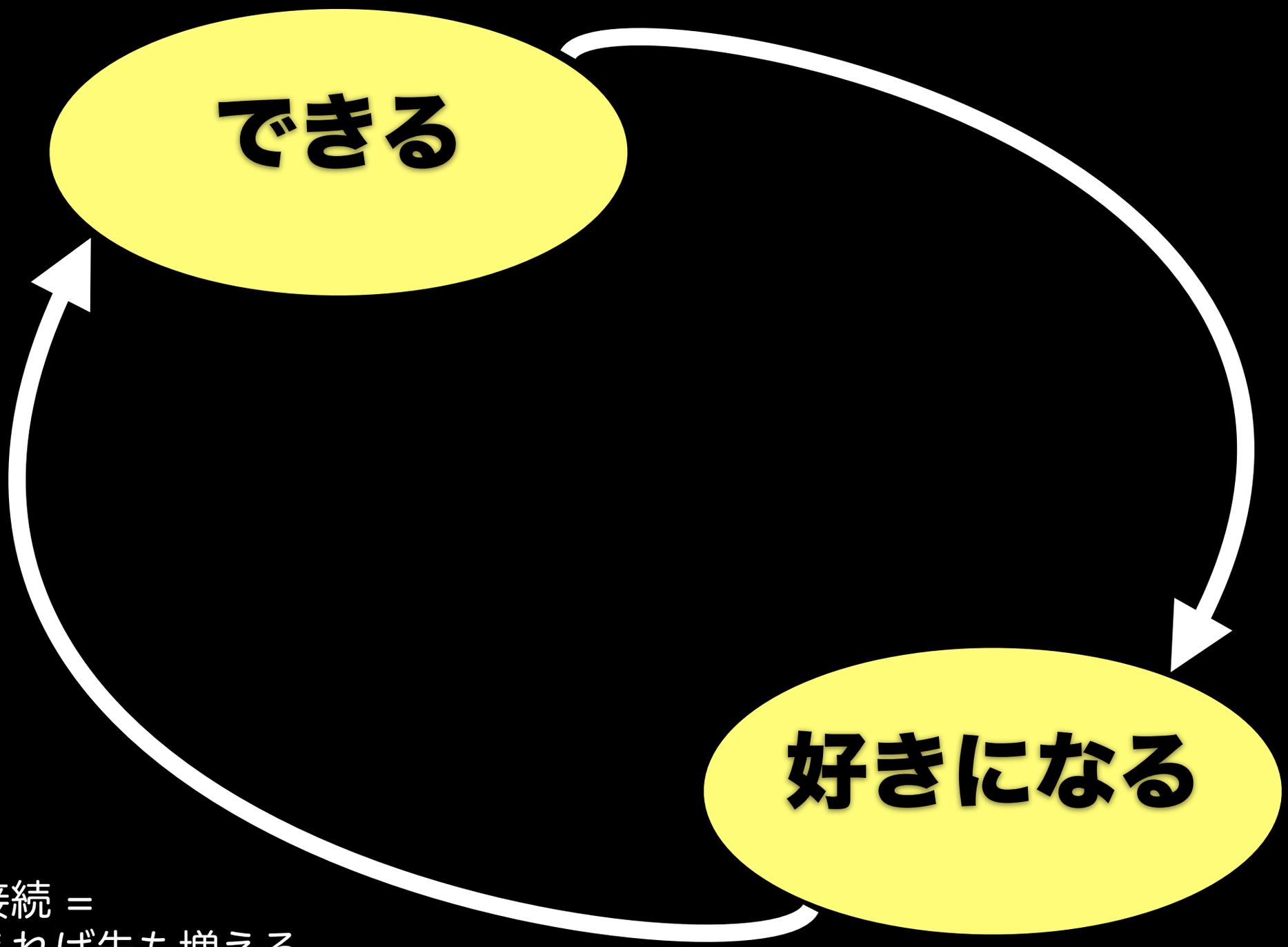
大事なことに集中する

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」

Speaker Deck
Published on Feb 15, 2018

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

Takuto Wada
12 Presentations

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

Blogger in Chief

Junichi Niino（jniino）
IT系の雑誌編集者、オンラインメデ

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

渦を見る

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

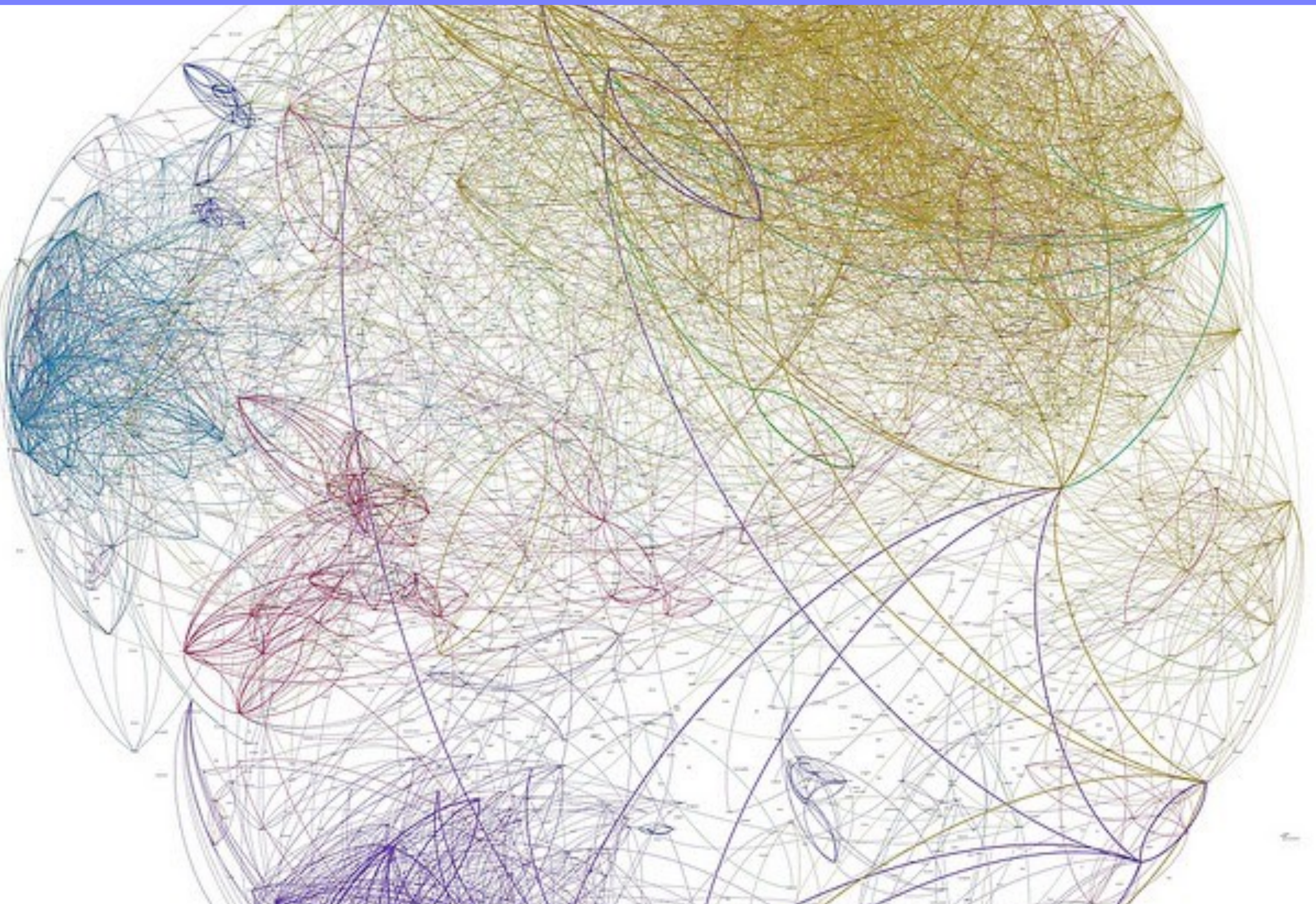
Create a password

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

Agile Media Network
パートナーブログ

My Profile

MY PROFILE
POWERED BY iddy

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

2014-03-30

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。

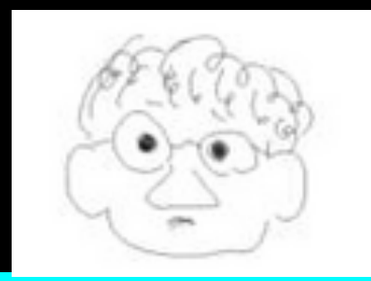
プログラマという仕事は、今も昔も勉強をしてないと普通の仕事も成立しないのだが、その勉強の仕方というか意味づけが、違ってきていると思うのだ。単純に言えば、量の問題である。今は勉強すべきことがずっと多い。

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

http://d.hatena.ne.jp/essa/20140330/p1

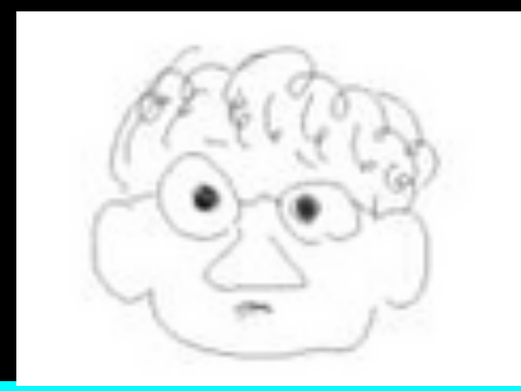
“しかし、今の業界は、「エコシステム」の時代だ。熱帯雨林のように、食いあいつつ共生しあうさまざなタイプのプレイヤーが、自分の為だけの個別の意思決定をして、その相互作用で技術が発展していく。「エコシステム」は矛盾だらけで、ある技術が発展するのと同時に、そのアンチテーゼとなる技術も伸びる”

“「ロードマップ」が指し示す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”



http://d.hatena.ne.jp/essa/20140330/p1

“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”



http://d.hatena.ne.jp/essa/20140330/p1

The Rise of “Worse is Better”

rpg@lucid.com
日本語訳:
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 正当性と同じぐらい重要である。
- 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の方が重要である。
- 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。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ぁ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
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

[redacted]

[redacted]

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

https://twitter.com/t_wada/status/982155252643250177

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

https://speakerdeck.com/twada/worse-is-better-understanding-the-spiral-of-technologies-2019-edition

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

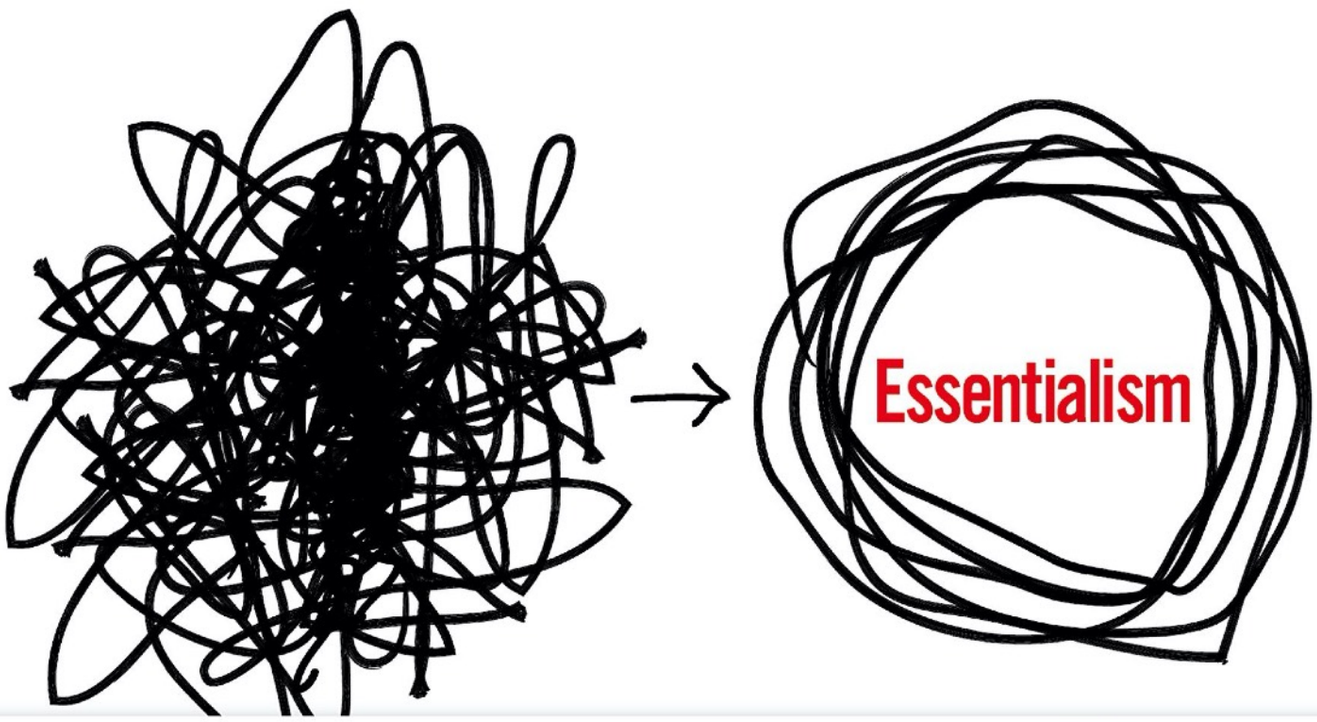
5. 大事なことに

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）
絶賛！

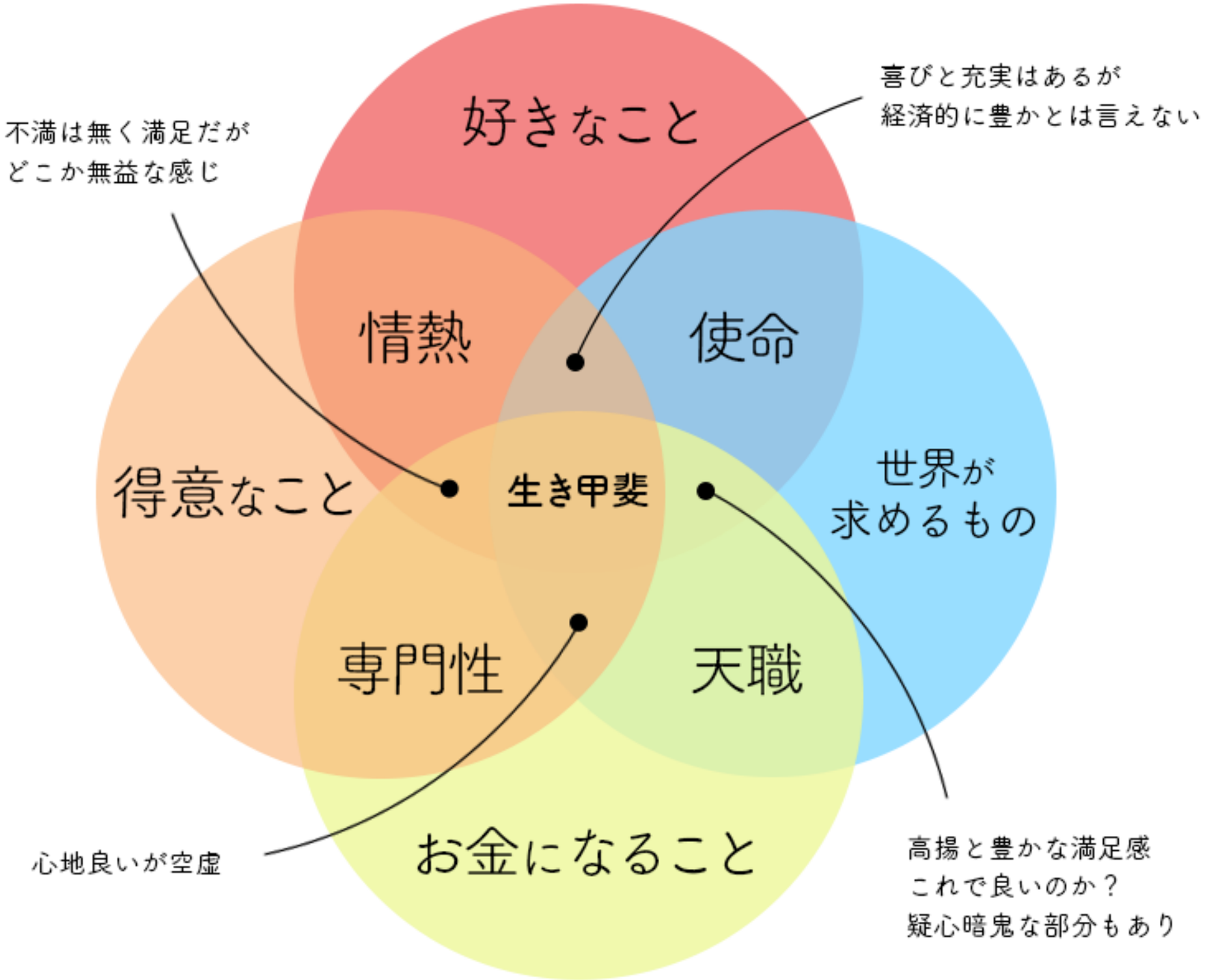
10万部突破！

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

99%の無駄を捨て
1%に集中する方法！

生き甲斐の図



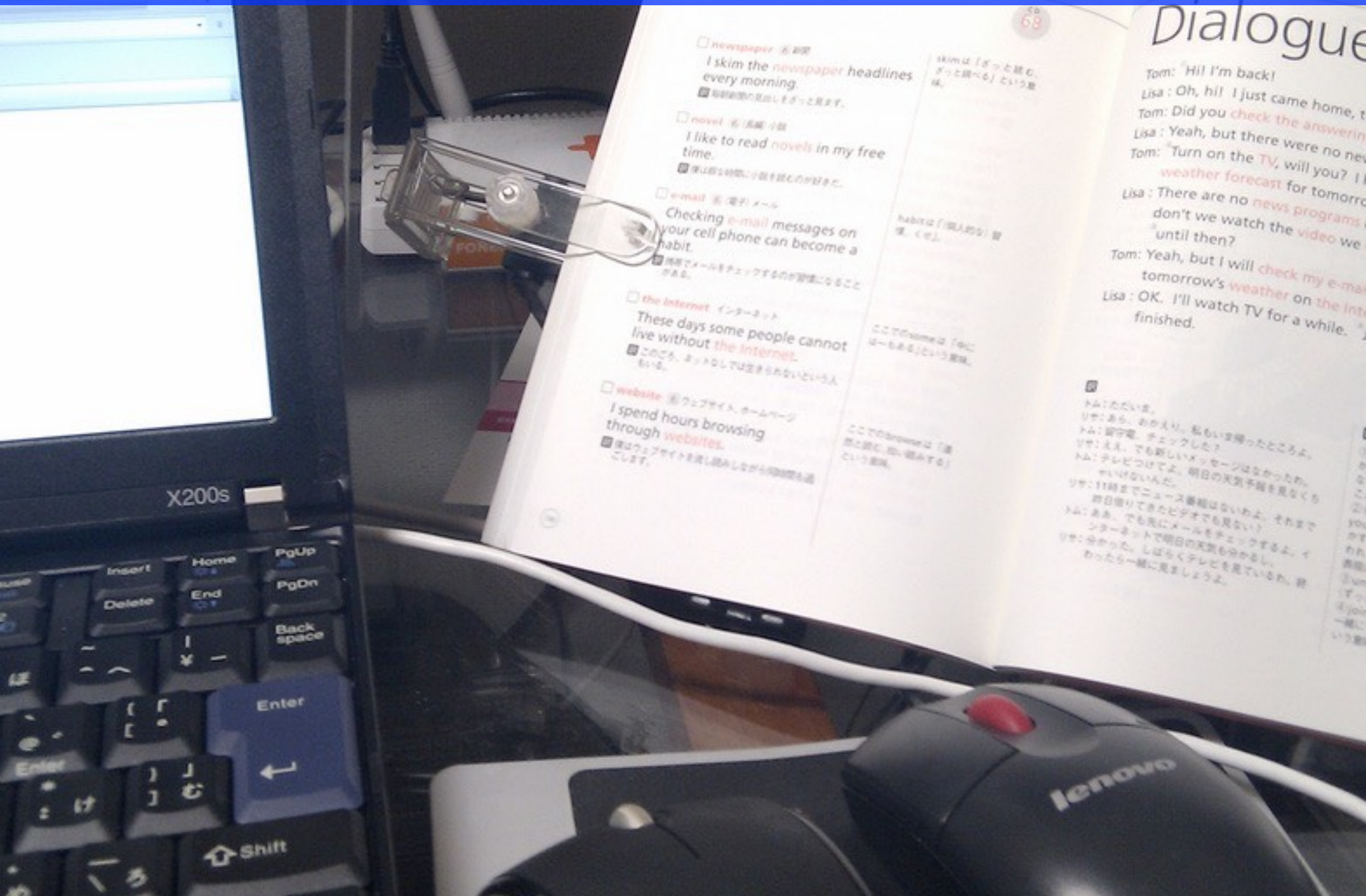
Agenda

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


X200s
Insert
Home
PgUp
Delete
End
PgDn
Back space
Enter
Shift
Lenovo
68
newspaper 名 新聞
I skim the newspaper headlines every morning.
novel 名 長編小説
I like to read novels in my free time.
e-mail 名 電子メール
Checking e-mail messages on your cell phone can become a habit.
the Internet インターネット
These days some people cannot live without the Internet.
website 名 ウェブサイト、ホームページ
I spend hours browsing through websites.
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 ne
Tom: Turn on the TV, will you? I
weather forecast for tomorro
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 int
Lisa : OK. I'll watch TV for a while.
finished.

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