

Second System Syndrome

楽天技術研究所
ネットワーク応用通信研究所
*Ruby*アソシエーション
Heroku

まつもとゆきひろ
Yukihiro "Matz" Matsumoto
@yukihiro_matz



ソフトウェア開発

Software development




ソフトウェア開発は難しい

Software development is hard



誤解もある
It's often misunderstood



設計してコーディング

Design then code

間違い
Wrong!




ソフトウェア開発は設計である

Software development is designing



間違った前提による困難さ

Difficult to work on false assumption



設計=デザイン



デザインは難しい
Design is hard



何を作るべきか知らない

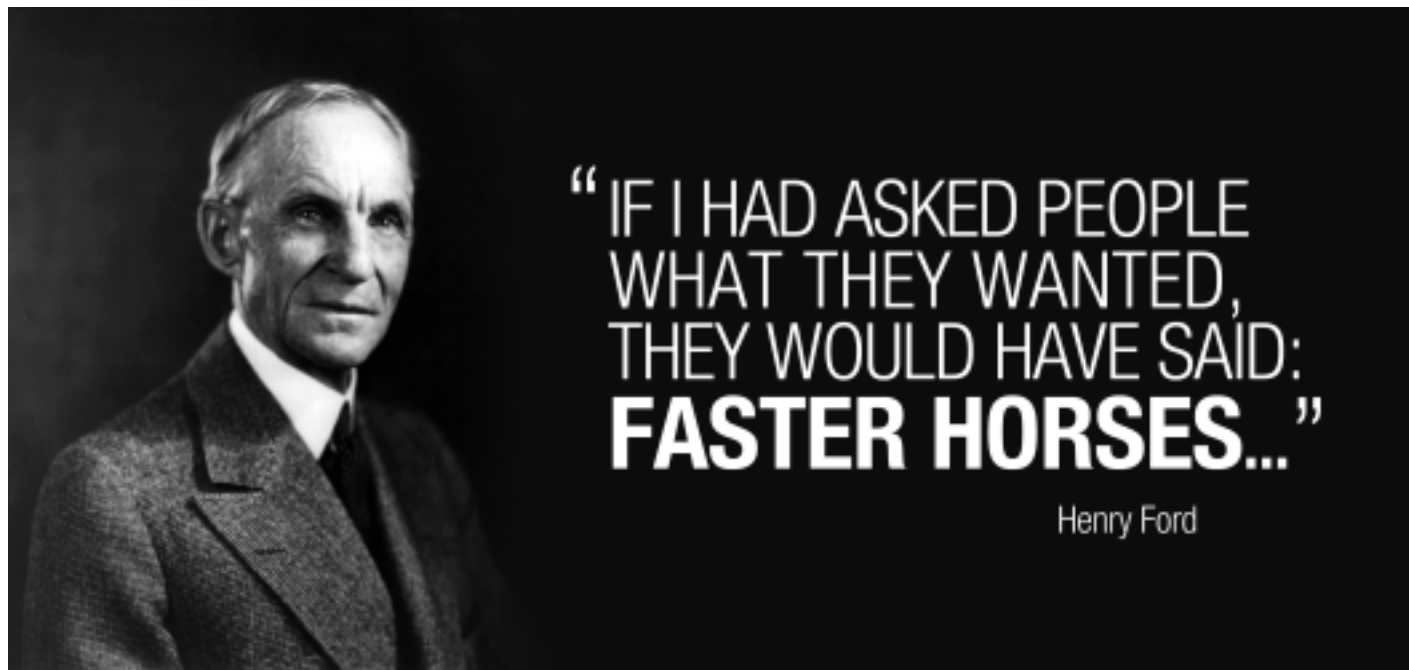
We don't know what we should make





欲しがるものを作ってはいけない

We shouldn't make what they want



顧客に何が欲しいか尋ねたら「速い馬」と答えたろう

ヘンリー・フォード



生活を変えるものを作る

Make what changes their lives



存在しないものを創造する

Create something that have never existed



デザインは決断である

Design is decision



決断は難しい
Decision is hard



未来はわからないから
Because the future is unknown



状況が変化するから
Because the situation changes



昔は良かった
Good old days



目的は明快だった
The target was clear



数值計算

Numerical calculation



事務処理

Business processing



良き昔は過ぎ去った

Good old days have over



今は解を求めてさまよう

Now we struggle to find the solution



解はないかもしれない


The solution may be illusion

Ruby



Rubyを公開した時(1995年)

When I released Ruby in 1995



「スクリプト言語に オブジェクト指向は要らない」

Some said we don't need OO for scripting



間違ってた
But they were wrong



何が欲しいか聞かなかった

I didn't ask them what they want



私が未来で使いたいものを作った

I created what I wanted to see in the future



わからない人もいた

Some didn't understand what they saw



気にしなかった

I didn't care



10年続けた
I kept working for 10 years



「当たり前」になった

Ruby became new normal



未来の「当たり前」を作る

Create new normal



ソフトウェア開発の本質

The essence of software development



良いソフトウェアを作った

Suppose you have created the great software



そこで終わりじゃない
It's not the end of your development



未来はわからないから
Because the future is unknown



状況が変化するから
Because the situation changes



生まれた瞬間から古びるソフトウェア

Software gets older soon after created



変化する状況への対応

We have to adapt to changing situation



時間経つにつれて

As time goes by



システムはより大きく複雑に

Systems will go bigger, more complex



保守も難しくなる

More difficult to maintain



複雑なシステムがイヤになった時

When we become sick of complex systems



セカンド・システム症候群が起きる

Second system syndrome will come



セカンド・システム症候群

Second system syndrome (SSS)



廢棄再創造希求症候群

Scrap and build syndrome



症状

Symptoms



すべて捨てたい誘惑

Temptation to throw away everything



ゼロから作り治したい誘惑

Temptation to create everything from scratch



もっと綺麗なシステムが作れる という幻想

Illution that we could create cleaner systems



もっと高性能なシステムが作れる という幻想

Illution that we could create systems that performs better



意思決定者を説得する熱意

Enthusiasm to persuade the boss



良いデザインへの強い熱意

Strong enthusiasm for better design



予想以上に困難なデザイン

Difficult design problems beyond expectation



予想以上にかかる時間

Delayed schedule beyond forecast



予算を越える費用

Development cost beyond budget




怒る顧客

Angry clients



破綻するプロジェクト

Project failure



あ痛た、た
Ouch!!



しょっちゅう起きる

SSS happens all the time



程度の違いこそあれ

with different severity



私の専門はプログラミング言語

My expertise is programming languages



言語も例外ではない

Languages are no exception



むしろ言語では頻発する

Rather many languages suffer SSS



言語は長生きだから
Because languages live longer



普通のアプリよりもはるかに
Far more than usual applications

Case 1



Perl5 vs Perl6



Perl6は2000年に開発開始

Perl6 project started 2001



Perlの思想を引継ぐ

Perl6 inherit Perl philosophy



ゼロから実装

Perl6 implementation from scratch



新文法

Totally new syntax



新VM

Totally new virtual machine



15年後(2015年)
15 years later (2015)



Perl6はまだない

We don't have Perl6 yet



今年のクリスマス(予定)

Finally this christmas (hopefully)



普及には恐らくさらに何年も

Probably need years to become widely used



誤解しないで欲しい
Don't get me wrong



Perlコミュニティを尊敬している

I respect Perl community



Perlコミュニティは賢い

Perl people are smart



そんな彼らも苦しむ
Even they suffer



セカンドシステム症候群は恐ろしい

Second system syndrome is scary

Case 2



Python2 vs Python3

Python3000



Python3000設計方針

Python3000 design policy



古いやり方を捨て機能重複を減らす

"reduce feature duplication by removing old ways of doing things"



西暦3000年までに出版

Wish we could release it before A.D.3000



何年も何年も議論

Discussed for years



2006年、Python3000開発開始

Python3000 project started in 2006




2008年、Python3.0公開

Python3.0 was relased in 2006



互換性問題

Compatibility problems



2015年になっても
Python2がまだ使われている
In 2015, Python2 is still widely used



Python3を捨てようという人まで

Some even claimed to give up Python3



最近Python3率が増えたような

Recently Python3 has adapted more widely

ようやくか
Finally



誤解しないで欲しい
Don't get me wrong




Pythonコミュニティを尊敬している

I respect Python community



Pythonコミュニティは賢い

Python people are smart



そんな彼らも苦しむ

Even they suffer



セカンドシステム症候群は恐ろしい

Second system syndrome is scary



Rubyも例外ではない

Ruby is no exception



Ruby1.8 vs Ruby1.9



パフォーマンス

Performance



多言語化

Multilingualization (M17N)



2000年構想開始

The idea was born in 2000



2004年プロジェクト開始

The project started in 2004



2007年1.9.0リリース

1.9.0 was released in 2007



互換性問題

Compatibility problems



普及に5年以上
Took 5 or more years



Pythonよりマシ

We've done better than Python3



「あきらめよう」という人はいなかった

No one suggested to give up at least



なにがよかったのか
But how?



どうやってセカンドシステム症候群を 克服するか

How can we overcome second system syndrome?



1. 「全部捨てる」ことを避けた

we have never thrown away everything



ひとつずつ置き換えた

We have replaced one at a time



文字列クラス

Replaced string class



仮想マシン

Replaced Virtual machine



オブジェクト表現

Replaced object representation



ガーベージコレクター

Replaced garbage collector



できるだけご完成を維持しつつ

Keep compatibility as much as possible



移行パスを用意しつつ

Prepare migration path



劇的な変化を試みない

Never tried too drastic changes



少しずつ変化する

Changed step by step



バージョンの幻想

2. versioning illusion



2.0 vs 3.0



5.0 vs 6.0



1.8 vs 1.9



3. 移行のご褒美

migration bait



1.9以降の大きなメリット

Moving 1.9 had huge benefit



パフォーマンス

Performance



動機付け

Motivation



セカンドシステム症候群対策の大原則

Rules of thumb of SSS



「全部捨て」をしない
Don't throw away everything



一気にやりすぎない
Don't push too hard



やさしく
Push softly



着実に
Push steady



互換性

Compatibility



変化を止めない
Keep moving forward



2.0は(ほぼ)完全な互換性を維持

2.0 had (almost) perfect compatibility



しかし時が経つにつれ
But as time goes by



セカンドシステム症候群は またやってくる

Second system syndrome comes again

Ruby 3.0



我々は原則を忘れない
But we don't forget the rules



1. 「全部捨て」をしない

Don't throw away everything

2. 一気にやりすぎない

Don't push too hard

3. やさしく

Push softly

4. 着実に

Push steady



Ruby3構想中

We started working on Ruby3.0




状況が変化するから
Because the situation changes



マルチコア

Multi cores



データスケーラビリティ

Data scalability



コードスケーラビリティ

Code scalability



実驗中

By experimenting ideas



なにも約束しない

We don't promise anything



セカンドシステム症候群対策の大原則

Rules of thumb of SSS



1. 「全部捨て」をしない

Don't throw away everything

2. 一気にやりすぎない

Don't push too hard

3. やさしく

Push softly

4. 着実に

Push steady

Ruby3



1. 人とコンピューターの協同 Man-machine collaboration

2. パフォーマンス Performance

3. コンカレンシー Concurrency



より広い領域へ Toward broader domain



より高い生産性More productive



かなり高い互換性 Yet keeping compatibility



Rubyが未来を作る

Ruby will create the future



Rubyコミュニティと一緒に

Along with Ruby community



あなたと一緒に
With you



みなさんと一緒に
With all of you



Happy Hacking!



Thank you