GMOペパボ

エスパー力を高める

ソフトウェアの問題解決の技術

柴田博志 / GMO PEPABO inc.

2019.05.18 はこだて未来大学 春のLT大会

GMOペパボ



Hiroshi SHIBATA @hsbt

Executive Officer VP of Engineering
Technical Director
at GMO Pepabo, Inc. @pepabo

https://www.hsbt.org

self.introduce

```
=> {
  name: “SHIBATA Hiroshi”,
  nickname: “hsbt”,
  organizations: [“ruby”, “rubygems”, “asakusarb”, “pepabo”, …],
  commit_bits: [“ruby”, “rake”, “rubygems”, “bundler”, “rdoc”,
“psych”,  “ruby-build”, “railsgirls”, “railsgirls-jp”, …],
  sites: [“hsbt.org”, “ruby-lang.org”, “rubyci.org”, “railsgirls.com”,
“railsgirls.jp”],
}
```



“これだけは伝えたい技術”

会社ではエンジニアじゃないんだよな…

エスパー力を高める



すごいエンジニアはまるで超能力者(エスパー)のよう

・エラーメッセージを見ただけでソースコードのどこを直せばいいのかわかる

・触ったことがないソフトウェアの使い方がわかる

・デバッグツールを使って「カタカタカタ…わかった」と不具合を見つける

この人たちエスパーでは???



これはエスパーではなくて“技術”
技術なら誰でも習得できる

ポイント1: ソフトウェアの特性を知る

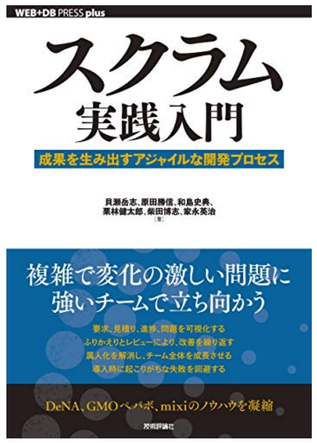
ソフトウェアってなんですか?
(自由記述: 2分)

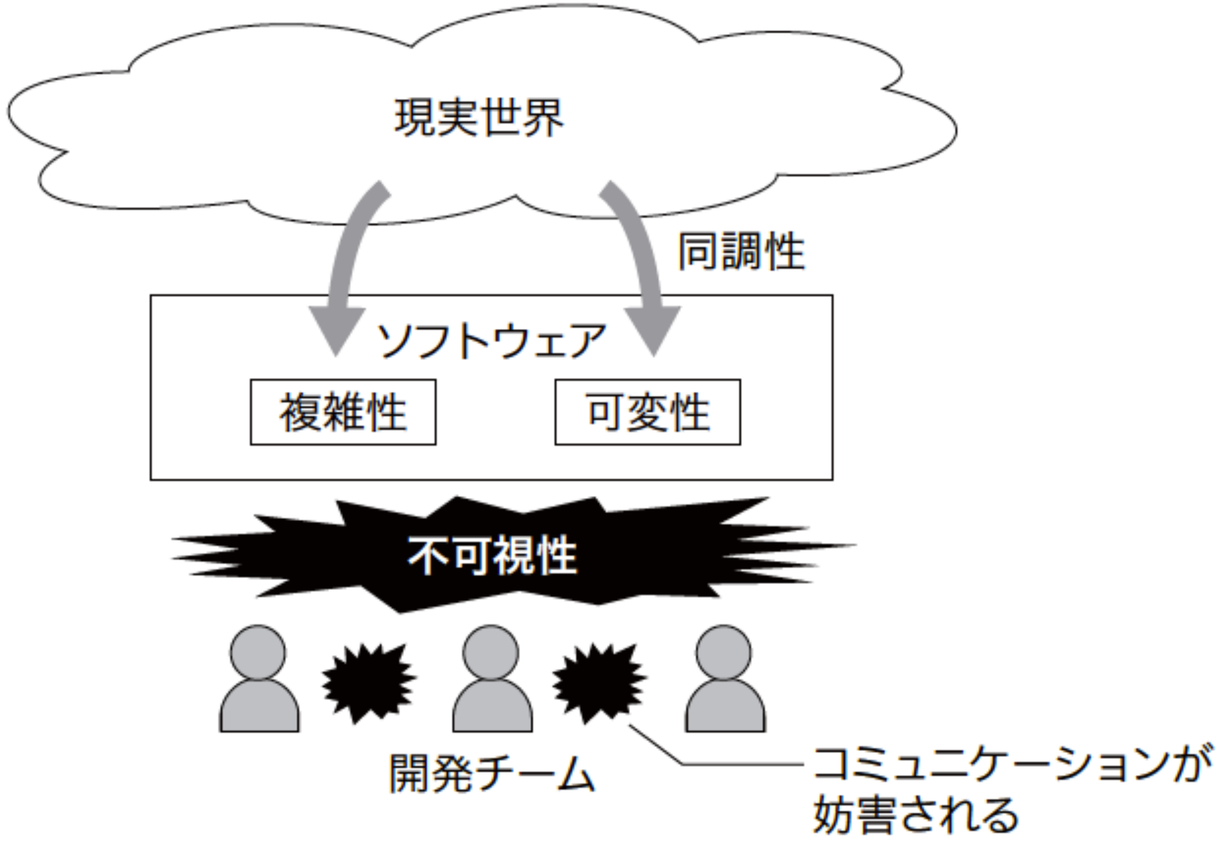
ソフトウェアはどこにあるのか

・ソフトウェアは人々の心の中にある

・ソフトウェアは人々のコミュニケーションの中にある

『スクラム実践入門』技術評論社、2016年、p.10





あらゆるソフトウェアが人と人との対話を持つ

- 特定の組織からの要求を解決するソフトウェア → 開発者: 開発チームまたは自分個人 利用者: 特定の組織に所属している人々
- OSSなど不特定多数の利用が想定されるソフトウェア → 開発者: 開発チームまたは自分個人 利用者: 不特定多数
- 自分向けのソフトウェア → 開発者: 自分 利用者:自分



ソフトウェアの動きを予測するには人を知ることが最短経路の場合がある

人を知るとソフトウェアを(高速に)理解できる

- Rails を作った DHH が作ったソフトウェアなので、この Active* は〜
- ○○さんが作ったソフトウェアなのでこういう時も対処されているはず
- Microsoft 所属の人が書いたコードなので Windows でも大丈夫

…



ポイント2: 期待している振る舞いを知る

良いソフトウェアって何だと思いますか?
(自由記述: 2分)

システム工学

・事象をインプット、系(システム)、アウトプットの三つでモデル化する

・インプットと系から生まれたアウトプットが期待値通りであるかどうかの差分を品質と呼ぶ



ソフトウェアの品質の基本的な考え方

- 製造業での品質工学では、開発設計段階で品質を作り込む

  → 製造段階で作り込むから機能のばらつきが発生する

- よくある誤解

「テスト・試験で品質向上」

- テストでわかることは作成物の品質がどの程度かであり、作成物そのものの品質が変化するわけではない

- これらはソフトウェアでも同じ



企画段階の品質管理

“品質”はユーザーの期待する機能と実際に提供している機能との乖離である

“高”品質

・機能と理想機能との乖離のばらつきが小さいこと

“低"品質

・機能と理想機能との乖離のばらつきが大きいこと

期待した振る舞い通りに動くこと

- 期待: インプットAを入れたらアウトプットBが出るはず
- 現実1: アウトプットCが出てきた、使い物にならない!
- 現実2: アウトプットBが出てきた、最高!

ソフトウェアは人と人との概念に存在する以上、同じ人や組織から作られたソフトウェアはあるインプットに対して同じアウトプットを出すことが多い

それをあらかじめ予測することで、期待通りにならなかった時の対処をする



期待通りに動かない場合にどうするか

- **自分の入力を疑う** → 自分のインプットは本当に A だったのか?
- **自分の期待を疑う** → 自分は B が出てくると思っていたが、世の中の多くの人や作者はそもそも C を出すものと思っているのではないか?
- **自分の選択を疑う** → A を入れたら B を出すシステムだと思っていたがそもそも C を出すシステムだったのでは?

自分は案外信用できない



ポイント3: 中身を知る

良いソフトウェアをどのように作りますか?
(自由記述: 2分)

ソフトウェアは組み合わせの時代

- Linux をはじめとするOSSの利用がソフトウェア開発の当たり前の時代
- あらゆるソフトウェアが相互に依存することで構成される
- 依存関係と依存しているソフトウェアの作者、振る舞いを知ることで対象とするソフトウェアそのものの動きが推測可能となる

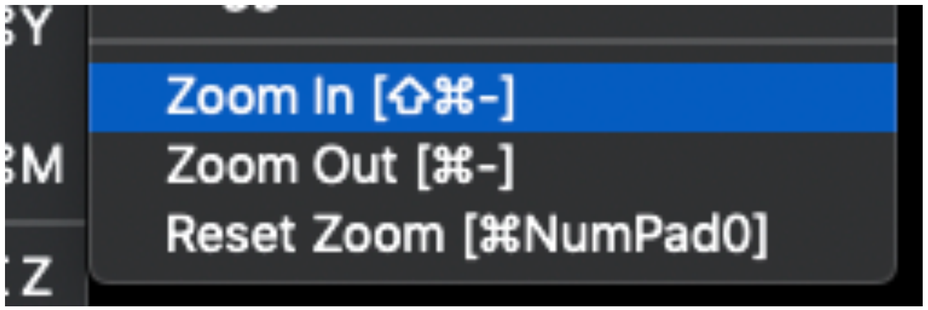


依存しているソフトウェアの振る舞いを知る

・CLI ライブラリの Thor を使っているソフトのコマンド体系は似ている

```
module Bundler
  class CLI < Thor
    require_relative "cli/common"
```

・VScode や Chrome は同じ Webkit を使っているならキーバインドは同じ



依存しているソフトウェアのデバッグ方法を知る

- Linux の syscall を見るなら strace の使い方

```
dtrace: error on enabled probe ID 2174 (ID 159: syscall::read:return): invalid kernel access in action #12 at DIF offset 68
read(0xB, "\0", 0xC8E)          = -1 Err#35
sigprocmask(0x1, 0x0, 0x7FFEE84519B0)           = 0x0 0
sigaltstack(0x0, 0x7FFEE84519A0, 0x0)           = 0 0
```

- Ruby なら binding.irb や dump の解読方法

```
-- C level backtrace information -------------------------------------------
                    0   libruby.2.4.3.dylib                 0x000000010c012da8 rb_vm_bugreport + 136
                    1   libruby.2.4.3.dylib                 0x000000010be9c7d3 rb_bug_context + 467
                    2   libruby.2.4.3.dylib                 0x000000010bf888a8 sigsegv + 72
```



依存しているソフトウェアの作りを知る

- 一般的に Rails なら Rails アプリケーション全てに知識を流用できる

```
class Changeset < ActiveRecord::Base
  belongs_to :repository
  belongs_to :user
```

- RSpec や Sidekiq など Rails と共に使われることが多いソフトウェアも読みやすい

```
RSpec.configure do |config|
  config.include Spec::Builders
  config.include Spec::Helpers
```



まとめ

・誰がソフトウェアを作っているかを知る

・ソフトウェアにどう動いて欲しいのかを知る

・ソフトウェアがどのように動くのかを知る

・ソフトウェアを知ることで動きを推測できるようにする

・エスパー力が高まって捗る





一緒にもっと面白くしませんか?

最新の採用情報をチェック→ @pb_recruit

新卒採用ページをチェック→ https://recruit.pepabo.com/info/graduate/