

Qualidade com Agilidade

para startups



2015-02-06

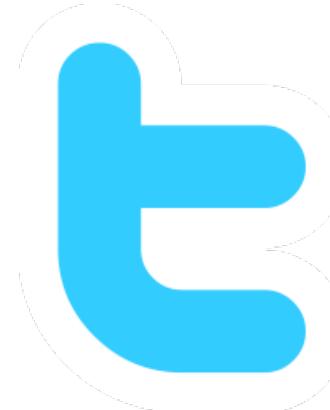


about.me/paulocheque

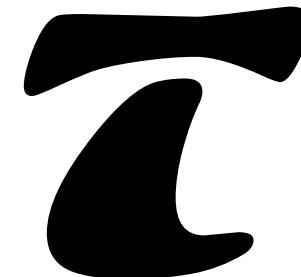
about.me/paulocheque



**oss,
bibliotecas,
bootstraps**



**links,
dicas,
comandos**



**CodeArt.io
(em dev)
contato,**



Concepção
Criação
Consolidação



Fases de desenvolvimento de uma startup

Ideia



Validação



Protótipos



Versões alfa (descartáveis)



KEEP
CALM
AND
GO
CRAZY

Concepção da startup

**Versão beta
em produção
(permanente)**



**Momento de
decisões
importantes**

Manutenção real



Criação da startup

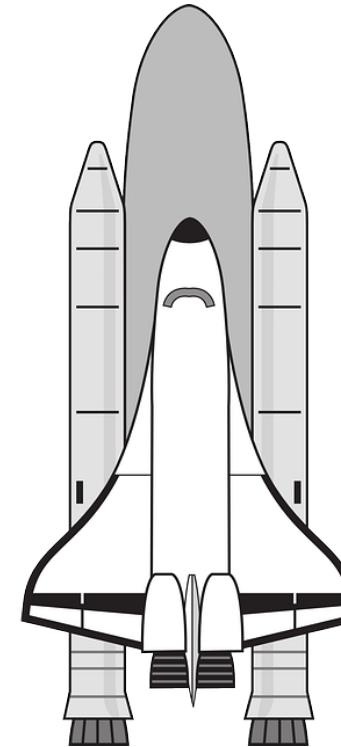
Aumento de clientes e de escala

Customizações

Mudanças de prioridades

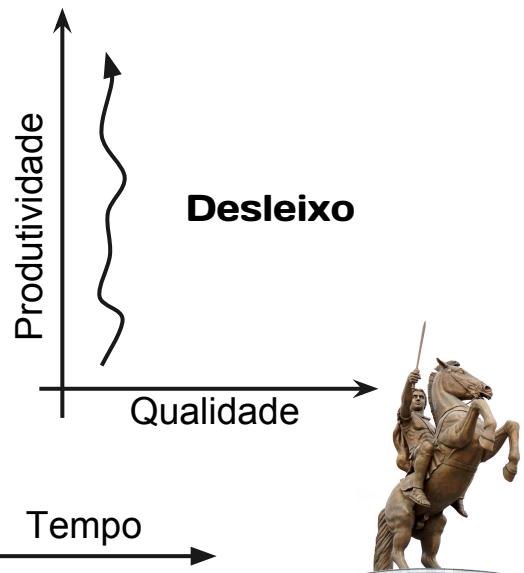
Sistemas em evidência

Consolidação da startup



Estratégia 1: go horse

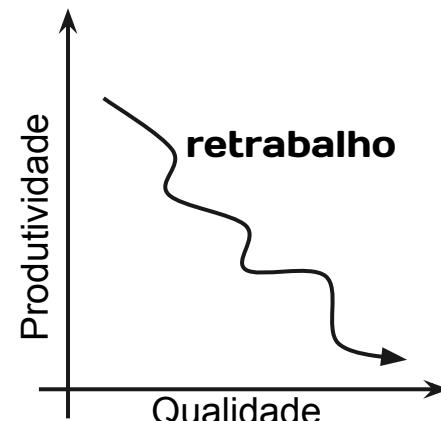
Criação



**manutenção
bugs**

**competitividade
progresso**

Consolidação



Estratégia 2: perfeição

Criação



Consolidação

↓ competitividade
↓ progresso

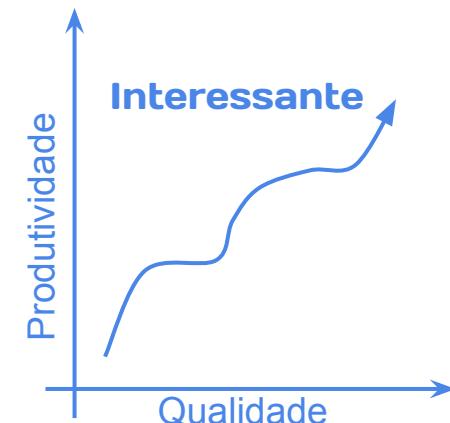
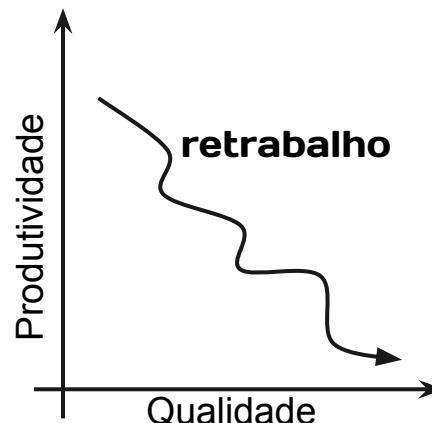


Estratégia 3: equilíbrio

Criação



Consolidação



O que é qualidade?

correção

manutibilidade

desempenho

usabilidade

escalabilidade

Flexibilidade

segurança

...



Qualidade com agilidade

boa priorização

equilíbrio

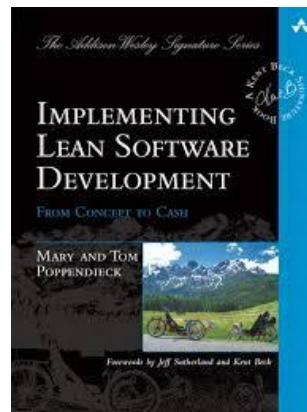
evitar desperdício

trabalhar
preventivamente

otimizar o todo

adiar
comprometimentos

entregar rápido



Implantação

Backend: deploy e rollback

Mobile: geração dos binários, pacotes etc

Script automático e rápido

- **1 comando ou 1 click**
- **demore segundos ou poucos minutos**

Implantação: exemplo

Heroku

```
git push heroku master  
heroku releases  
heroku rollback v46
```

Ferramentas:

- **Fabric (python)**
- **Rake (ruby)**
- **Gradle (java, scala)**
- **xcode-build (iOS)**
- **Vagrant**
- **Docker**
- ...

Implantação: exemplo

Fabric

```
fab staging deploy  
fab production deploy
```

```
@task  
def deploy(tag=None):  
    print(red("Deploying"))  
    update(tag)  
    with cd(env.app_path), prefix(venv()):  
        # in priority order  
        env.run('pip install -r requirements.txt')  
        env.run(manage('clean_pyc')) # safe (run before migrate)  
        env.run(manage('migrate'))  
        env.run(manage('makemessages'))  
        env.run(manage('compilemessages'))  
        env.run(manage('collectstatic --noinput'))  
        env.run(manage('compile_pyc')) # optimization  
    server_restart()  
    print(green("Deploy success"))
```

Implantação suave

Sem migrações críticas de dados

- sem risco de perda de dados

Deploys seguros

- sem derrubar a aplicação
- manter duas versões em execução simultaneamente
- evitar incompatibilidades de versões

Qualidade do código

Compare:

A

```
def generate_signature(key, serializable_data):
    return base64.urlsafe_b64encode(hmac.new(key, json.dumps(serializable_data), digestmod=hashlib.sha256).digest())
```

B

```
def generate(secret_key, serializable_data):
    string = json.dumps(serializable_data)
    sha256hash = hmac.new(secret_key, string, digestmod=hashlib.sha256).digest()
    value = base64.urlsafe_b64encode(sha256hash)
    return value
```

C

```
def generate_signature(secret_key, serializable_data):
    string = json.dumps(serializable_data)
    sha256hash = hmac.new(secret_key, string, digestmod=hashlib.sha256).digest()
    signature = base64.urlsafe_b64encode(sha256hash)
    return signature
```

Qualidade do código

Clareza e simplicidade do código evita desperdício de tempo

Mas o mais importante é a coesão e o design

Código coeso é fácil de refatorar

Dica: Testes de unidade

Testes automatizados ajudam na
criação do **design** das classes

Está difícil de testar?

Algo está errado
O código está **coesão**?

Testes de unidade

Testes não são para encontrar erros,
são para **prevenir erros**

O código de um teste deve ser
simples, curto, Fácil de implementar,
independente e rápido

Dicas sobre testes

Não desperdice tempo com testes de baixa qualidade

**Testes de código bagunçado devem ser simples e pouco específicos
(até o código ser refatorado)**

Testes: exemplos

```
def test_remove_file(self):
    filepath = self.create_temp_file()
    self.remove_temp_file(filepath)
    self.assertFileDoesNotExist(filepath)
```

```
def test_FileHandler_delete_singular3(self):
    handler = FileHandler
    type = 'delete'

    for id in range(1,6):
        file = File.objects.get(id=id)

        file.url = 'filestest/' + str(file.id)
        # Upload the file on amazon S3
        S3Helper.create('filestest', '/' + str(file.id), 'data')
        file.save()

        for attachment in file.attachments.iterator():
            attachment.delete()

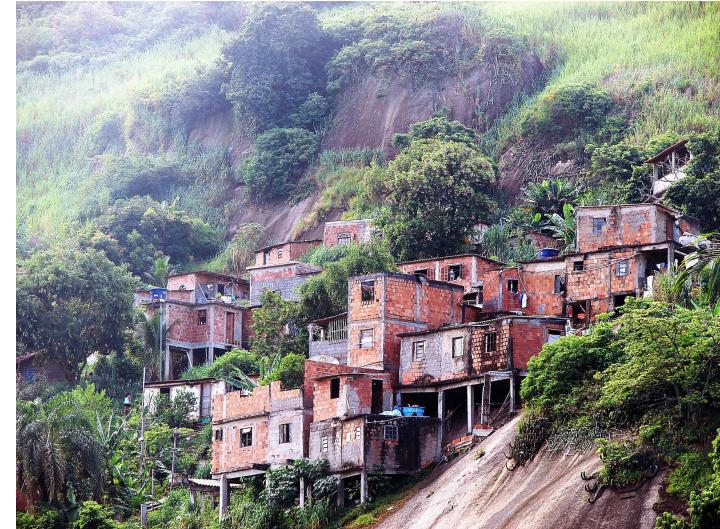
    for model in ModelX.objects.filter(id__lte=5):
        model.logo = None
        model.save()
    for model in ModelY.objects.filter(id__lte=5):
        model.delete()

    some_data = (
        # Exist and accessible
        ('1/', {}, 'populated_dict', 1),
        ('2/', {}, 'populated_dict', 1),
        # Exist and inaccessible
        ('6/', {}, 'gone', None),
        ('7/', {}, 'gone', None),
        # Don't exist
        ('100/', {}, 'gone', None),
        ('1000/', {}, 'gone', None),
    )
    self.execute(type, handler, some_data)
```

Arquitetura: crítico!

Arquitetura bagunçada:

- módulos e classes muito acopladas
- replicação de código
(copy and paste)
- gambiarra
- bugs
- difícil manutenção



Dicas de arquitetura

Não adie refatorações

**Uma refatoração
de cada vez**

**Faça rápido,
evite merges**



APIs

Siga padrões estabelecidos (ex: Rest)

- **Seja consistente com seus padrões**

**Sua API irá evoluir: versione
*/v1/resource/***

Mantenha a flexibilidade

APIs

Documentação simples

- no máximo, casos de teste

Documentação completa e detalhada
somente se ela for aberta para uso de
terceiros

Dados: crítico!

**Dados inválidos geram bugs em
lugares inesperados**

Opte por um significado por campo

Não procrastine correção de dados

- **Dados obsoletos se tornam uma incógnita**

Dicas de persistência

Sempre salve horários em UTC

Avalie salvar dinheiro em centavos: é mais fácil trabalhar com inteiros

**Explicite as unidades dos campos:
cents_of_dollar, area_m2, distance_cm**

**Investimos tempo em
prevenção e qualidade**

**Não Fomos
perfeccionistas nem
desleixados**

**Código está fácil de
refatorar**

**E o desempenho?
escalabilidade?
segurança?**



Retrospectiva: evitamos desperdício

Deploy ágil

Boa manutenibilidade

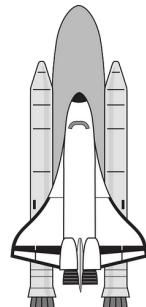
Dados coerentes

Aumento de clientes e de escala

Customizações

Mudanças de prioridades

Sistemas em evidência



Consolidação da startup



Retrospectiva: seu sistema está robusto!

Segurança

**Gerenciamento de senhas do sistema
e controle de acesso dos usuários**

Executar testes de segurança

- prioridade por nível de gravidade
- injeção de código, cross-site scripting,
clickjacking

Desempenho

**Processamento pesado? Grande uso
de memória? Use tarefas assíncronas**

**Muito uso da rede? Otimize e
compacte as mídias**

Tarefas lentas e repetidas? Use cache

Desempenho

Profilers e testes de carga local

Configure servidores Web e de bancos de dados

Otimize queries, denormalize dados

Escalabilidade

Com exceção de situações de emergência, não escale desempenho ruim causado por erros de código, os corrija

Mantenha o deploy simples



Meça e acompanhe

Tenha dashboards com tabelas e gráficos

Faça projeções

Identifique pontos de melhoria

Não desperdice tempo

**Desenvolver
software de
qualidade é criar
conhecimento**

**Invista em sua
equipe**

**Incentive seus
funcionários**

CLT Artigo 473 (IV): O empregado poderá deixar de comparecer ao serviço sem prejuízo do salário: por 1 (um) dia, em cada 12 (doze) meses de trabalho, em caso de doação voluntária de sangue devidamente comprovada;



Obrigado! paulocheque@gmail.com