

Guidelines de engenharia IT Teamlyzer

08.02.2023

Código

“Se achar que o código é mau, você mesmo o escreveria melhor, e quem o criou não sabia o que estava a fazer, deve ter cuidado para não cair você mesmo numa armadilha.”

Código

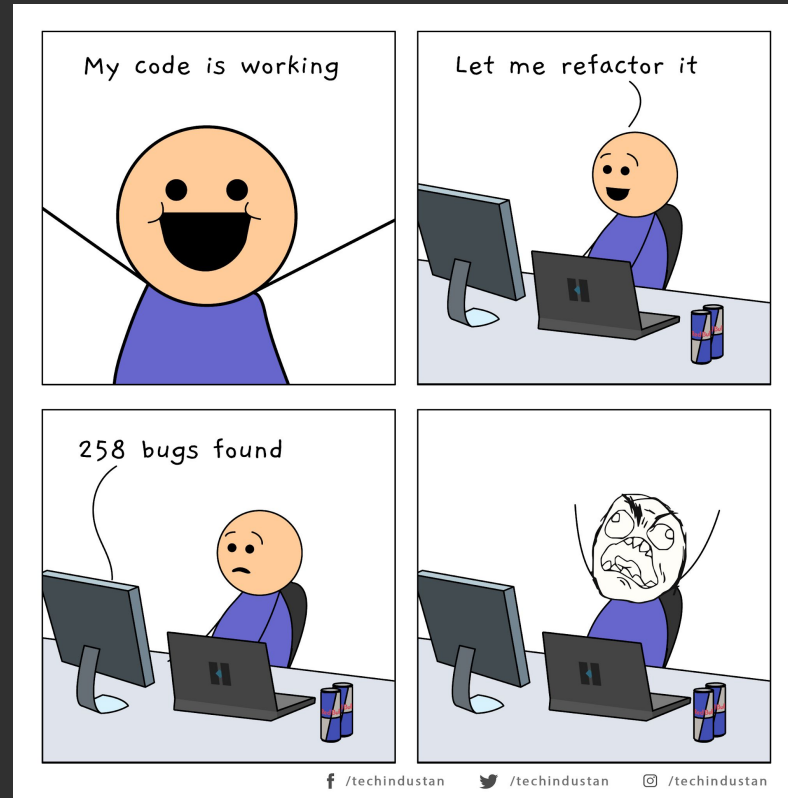
“Normalmente há uma razão pela qual o código tem o aspecto que tem, e se não o escreveu, pode simplesmente não o saber.”

Código

“Por conseguinte, ao melhorar a base de códigos existente, é preciso ter cuidado.

O fragmento que alteramos pode conter dependências das quais ainda não estamos cientes.”

Código



Código

**Conclui-se que é importante
saber quando manter e quando
alterar a atual codebase.**

Equipa de engenharia

4 pessoas

- Dinis - CTO
 - Vinícius - Software Developer (middle)
 - João - Software Developer (middle)
 - Paulo - Software Developer (júnior)
-
- Não temos QAs.
 - Não temos DevOps.
 - Não temos testes automatizados.
 - Temos uma audiência IT muito exigente.
 - Tech savvy.

Riscos

- Introduzir bugs e:
 - Colocar o site em baixo para comunidade (valores últimos 12 meses).
 - +3 milhões de pageviews.
 - +300.000 users.
 - Clientes pagarem e não poderem usar o site. Falamos de empresas como:
 - Mercedes.
 - Empresas do grupo VW.
 - NOS.

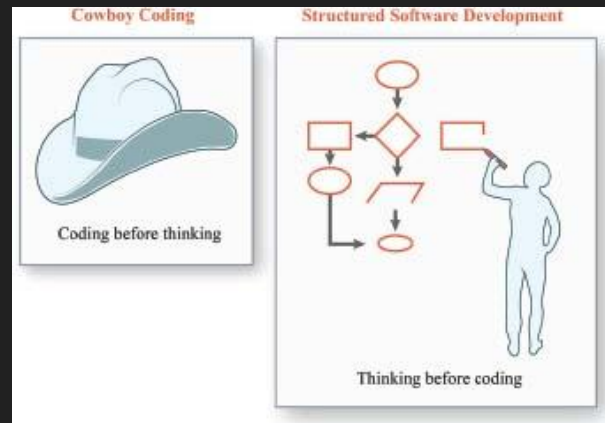
Regras gerais

- Pensar antes de começar a programar.
- Não programar sem perceber o problema.
 - Não há cowboy coding*.
- Evitar complexidade desnecessária sempre.

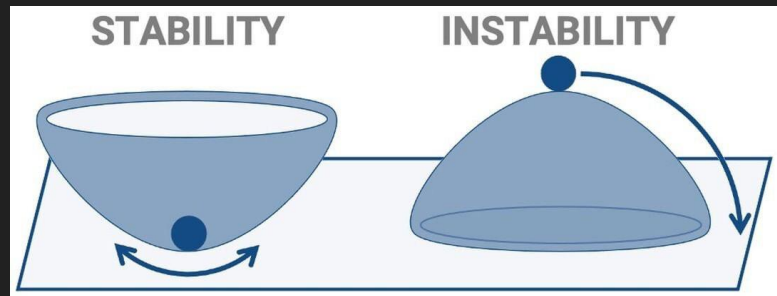
* Cowboy coding is software development where programmers have autonomy over the development process.

This includes control of the project's schedule, languages, algorithms, tools, frameworks and coding style.

Typically, little to no coordination exists with other developers or stakeholders.



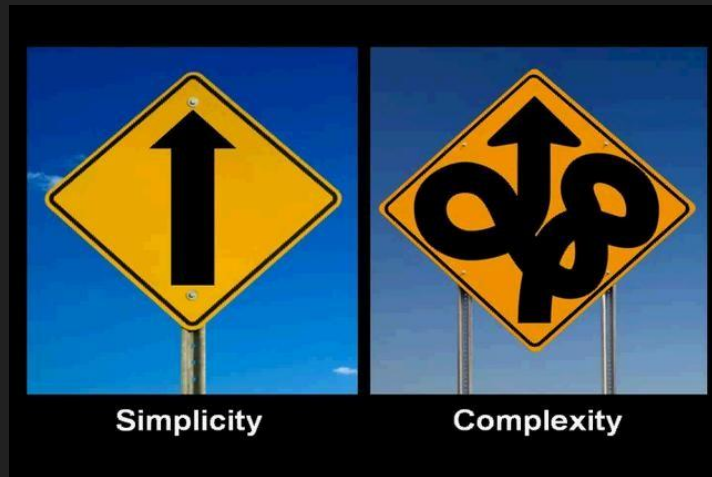
Regras gerais



- Só mexer no código que for preciso no contexto da issue e/ou projeto!
 - Simplificando: **mexer apenas no indispensável.**
- Se quiser resolver um bug fora do contexto do projeto, peço ao Dinis para criar a issue antes de o resolver.
- Evitar querer fazer mais do que pedido sem consultar.
 - Não queremos efeito surpresa no desenvolvimento.
 - Queremos previsibilidade/estabilidade.

Regras gerais

- Preferir sempre que possível **simplicidade** sobre complexidade.
- Evitar escrever muito código sem que haja necessidade evidente.
- Evitar programar muitas linhas sem consultar a direção técnica (aka Dinis).
 - O que pode acontecer, se não o fizerem, é programarem código que depois não vai passar nas code reviews.



Regras gerais

- Respeitar as instruções dadas.
 - Se é dito para fazer da forma X, não insistir na forma Y.
 - Se é dito para parar de mexer no código, não vai aparecer commits depois do aviso.
 - Quando são pedidas correções, confirmar que as mesmas são testadas e validadas por quem as fez.
 - Queremos evitar isto. →



Regras (Github)

- 1 branch por issue se for correcção.
- 1 branch por projeto, caso novo desenvolvimento.
 - Naming em inglês.
 - Exemplo: Branch chamado **issue-254** para lidar com a issue 254.
- Dinis atribui tarefas no projeto do Github.
 - Não há self assignment de issues a menos que essa instrução seja dada.



Regras (Github)

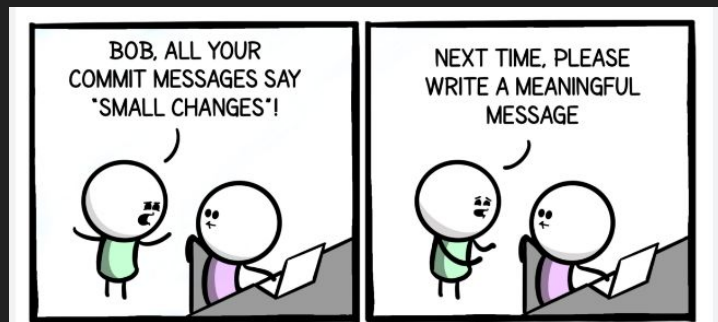
- Merges devem ser o mais frequentes possíveis (TBD) para evitar situações como a da imagem abaixo.
 - Mudanças menores, mais frequentes.
 - Menor risco por mudança.



Regras (Github)

- Manter o estado do projeto no Github sempre atualizado
- Se alguém vai trabalhar numa issue o status deve ser, antes de mais, **todo** e depois colocado **in progress** por essa pessoa.
- Se está para review deve ser colocado **"for review"**.
- Referenciar sempre issues nos commits.
 - Estrutura pretendida do texto do commit:

- "Issue #491:
 - Update README.md with xxxx."



Regras (Github)

- Não há push de conflitos para o repositório.
 - Obviamente devem ser resolvidos localmente.
- Branches utilizados para desenvolvimento são **rebased**, a partir do master, **frequentemente**, para evitar conflitos.
 - Idealmente sempre que houver um novo push para o master.

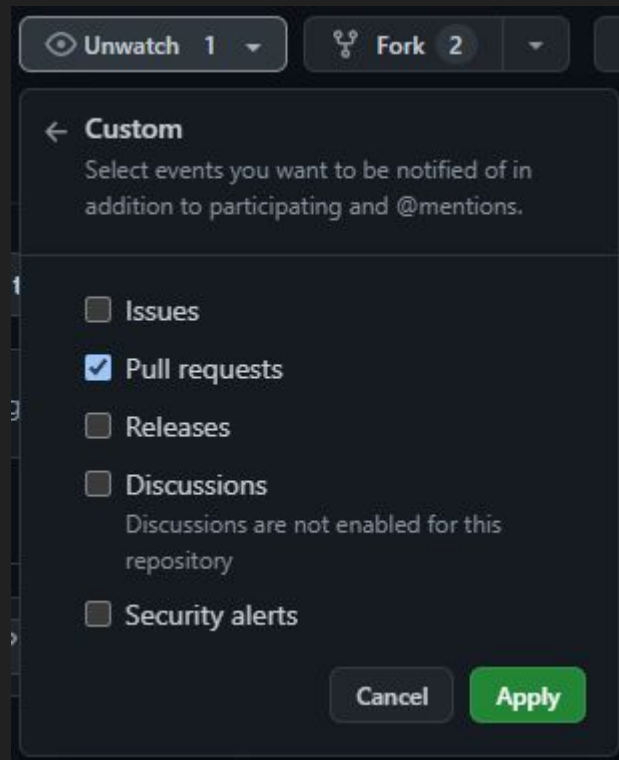


Regras (Github)

```
#atualizar master branch local
git checkout master
git pull --rebase origin master

#rebase do branch que estamos a trabalhar
git checkout issue-485
git rebase master
```

- Fazer watch no master do Teamlyzer e meter custom alerts relativamente aos PR.
- Assim já sabemos que temos de fazer rebase no branch em que estamos a desenvolver.

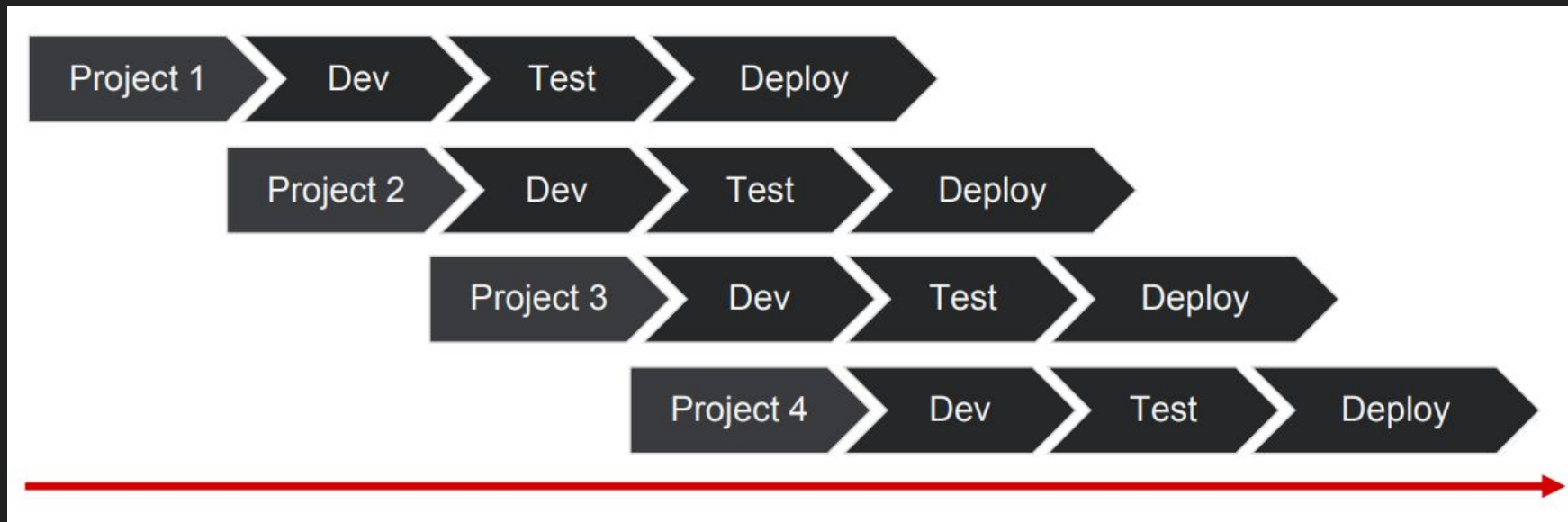


Regras (Github)

- Branches depois de merged são automaticamente apagados pelo Github.
- Deployment para produção é feito via **CI / CD - Github Actions**.
- Dúvidas são colocadas nas issues/PR e respondidas lá.
 - Eventualmente podemos usar o **Slack** para troca de ideias.



Ciclo de desenvolvimento actual




Regras (Coding standard)

- **Porque nos devemos preocupar em ter coding standards?**
 - Consistência entre programadores.
 - Facilidade de manutenção e desenvolvimento.
 - Legibilidade, usabilidade, segurança, entregas mais rápidas.
 - Se por algum motivo for necessário desviar do padrão, deve documentar a ocorrência.

Regras (Código)

- Evite misturar tabulações e espaços.
- Usar **4 espaços** para indentar Python, JS e CSS.
 - Configurar IDEs se necessário.
- Classes com iniciais maiúsculas são a convenção:
 - E.g. ParentSpecialization, FastSurvey, ...
- Em caso de dúvida seguir PEP 8.
- Existe na codebase um problema de uniformização de classes CSS entre _ (underscore) e - (traço).
 - Adotar, daqui em diante, letras minúsculas e usar - (traço):
 - E.g. **job-board** em vez de jobBoard ou job_board



```
1 if phrase == "hi":
2     print("hello")
3 else:
4     print("sorry I dont
4 spaces
```

Regras (Código)

- **Flask** for the win.
 - Se Flask resolve, resolvemos com Flask.
 - Se Flask tem forma automática de fazer, é dessa forma que o vamos fazer.
 - Se não sei, pergunto como se faz.
 - Temos bom know how interno de Flask.
 - Ou vejo como foi feito no código anteriormente.



Regras (Código)

- Se na codebase está a ser usada uma forma, eu vou fazer da mesma forma, ou pelo menos no push para o remote vai ter essa forma.

```
return render_template("companies/jobs.html", title='test',
```

Não é:

```
return render_template(  
    "companies/jobs.html",  
    title='test'
```

- Desta forma pelo menos mantemos consistência.



Regras (Código)

- Code reviews feitas no github e reversões/correcções da **responsabilidade** do autor do código
- **Lembrar que:**
 - Nunca vai entrar em produção mais alterações do que as indispensáveis ao projecto (PR vai ser rejeitado).
 - O motivo é que queremos estabilidade na codebase.



Responsabilidades

- Más implementações, reverter implementação desnecessariamente complexa, programar com dúvidas, pode causar:
 - Refazer trabalho.
 - Problemas de suporte.
 - Impacto noutras equipas.
 - Mais horas de trabalho.
 - Problemas de uptime.
 - Queixas de clientes.
 - Atrasos nas entregas.
 - Pressão superior.
 - Muito stress.



Os nossos valores na equipa

- Simplicidade.
- Estabilidade.
- Responsabilidade.
- Autonomia.

Esses 4 valores traduzem-se em:

- **Tranquilidade na equipa.**



