TRILHA DEVTEST

Evitando bugs através do Desenvolvimento Colaborativo

Uma parceria QA e Dev

THE DEVELOPER'S CONFERENCE

TDC2021

Quem são Dani e Nat?

Dani Cavalcanti

- Menina de QA na Sensedia e na QA Ladies;
- Mãe da Malu Helena e da Megg Louise
- Mentora na comunidade Elas Programam;
- Apaixonada pela cultura ágil;
- Design Thinker;
- Amante de palestras e treinamentos;
- Movida por desafios;
- Em formação.

@qadanicavalcanti danielle-cavalcanti-b2a48275 dani-cavalcanti

Nat Miranda

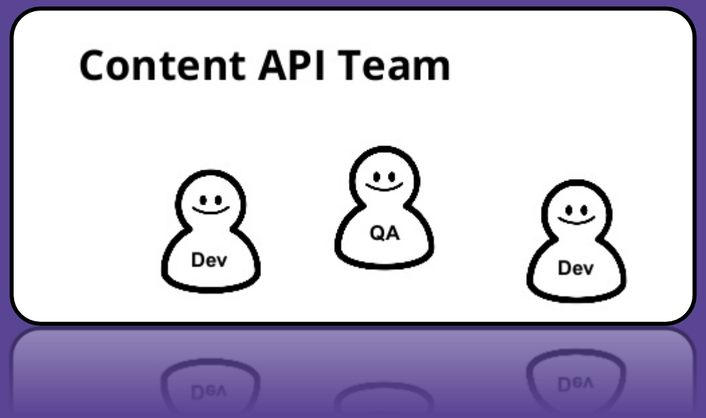
- Menina de QA na Sensedia;
- Apaixonada por processo de qualidade;
- Palestrante;
- Automação de testes;
- Amante de compartilhar conhecimento;
- Movida por desafios;
- Em formação.

nataliamirandademoura NataliaMirandadeMoura

TRILHA DEVTEST

TDC2021

O que nos motivou a falar desse tema?



01

Identificamos que a falta de documentação voltada para backend pode ocasionar problemas;

02

A importância de um processo de qualidade;

03

QA e Dev trabalhando juntos.

O PROBLEMA DE NÃO TER UMA DOCUMENTAÇÃO

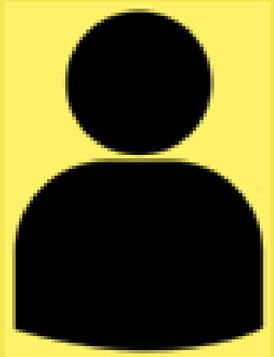
Alto índice de bugs

Muitas refatorações ao longo do desenvolvimento

Problemas no momento de integração (Backend e Frontend)

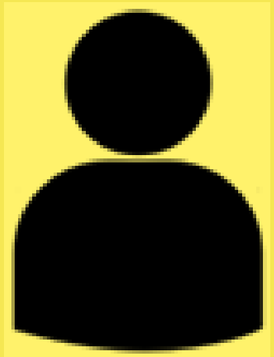
Não seguir os padrões de API Rest

Problema relacionado ao Status Code



Dev 1

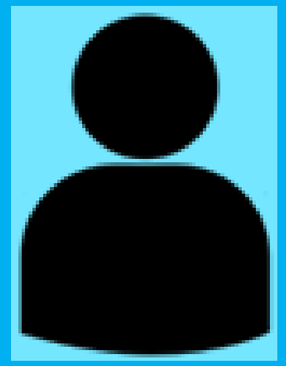
```
[HttpGet("id")]
[ProducesResponseType (StatusCodes.Status200OK)]
[ProducesResponseType (StatusCodes.Status400BadRequest)]
[ProducesResponseType (StatusCodes.Status404NotFound)]

public IActionResult Get (int id)
{
    var cliente = clienteBusiness.FindById(id);
    if (cliente == null) return NotFound (Resource.mensagemStatus404);
    return Ok(cliente);
}
```



Dev 2

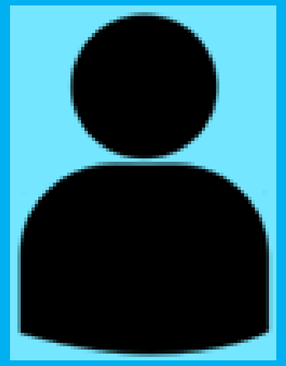
```
[HttpGet("id")]
[ProducesResponseType (StatusCodes.Status201Created)]
[ProducesResponseType (StatusCodes.Status400BadRequest)]

public IActionResult Get (int id)
{
        var cliente = clienteBusiness.FindById(id);
        return OK (cliente);
}
```

Problema relacionado as mensagens de retorno



Dev 1

```
[HttpPost]

[ProducesResponseType (StatusCodes.Status201Created)]

public IActionResult Post ()

{

  var cliente = clienteBusiness.FindById(id);

 return Created(cliente, “Inserido com sucesso”);

}
```



Dev 2

```
[HttpPost]
[ProducesResponseType (StatusCodes.Status200OK)]
[ProducesResponseType (StatusCodes.Status404NotFound)]

public ActionResult<PessoasCadastradasViewModel> ObterPessoasCadastradas (
     [FromBody] PessoasCadastradasConsultaViewModel
pessoasCadastradasConsultaViewModel)
{
        PessoasCadastradasViewModel pessoasCadastradasViewModel =
_pessoaCadastradaService.ObterPessoasCadastradas(pessoasCadastradasConsultaViewModel);
        if (pessoaCadastradasViewModel != null)
            return OK (pessoasCadastradasViewModel, "Pessoa cadastrada com sucesso.");
}
```

Solução 1 – Ter uma classe contendo as mensagens

```
jetoPilotoQA | ProjetoPilotoQA.Controllers.Base.Resource | mensagemStatus404

1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Threading.Tasks;
5
6   namespace ProjetoPilotoQA.Controllers.Base
7   {
        3 referências
8       public static class Resource
9       {
10          public static string mensagemStatus200GET = "Consultado com sucesso.";
11          public static string mensagemStatus200PUT = "Alterado com sucesso.";
12          public static string mensagemStatus200DELETE = "Excluído com sucesso.";
13          public static string mensagemStatus201 = "Criado com sucesso.";
14          public static string mensagemStatus404 = "Dados informados não encontrados.";
15      }
16  }
17
```

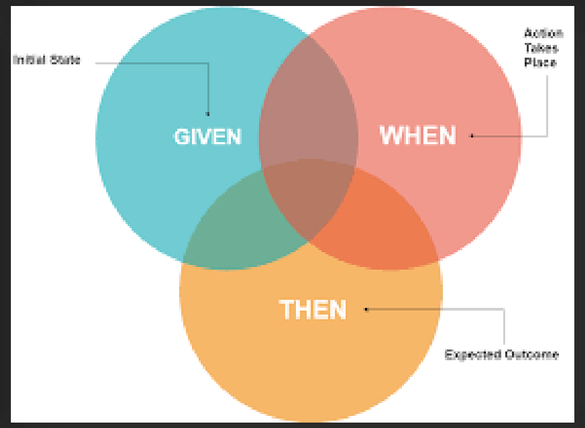
Solução 2 – Problem Details

- Para tratar das mensagens de erros em nossas aplicações, podemos utilizar o Problem Details.

- A especificação **RFC 7807 — Problem Details for HTTP APIs**, criada em março de 2016 pela IETF, foi criada com o intuito de padronizar os formatos de mensagens de erro em APIs HTTP, evitando assim que novos formatos sejam criados.

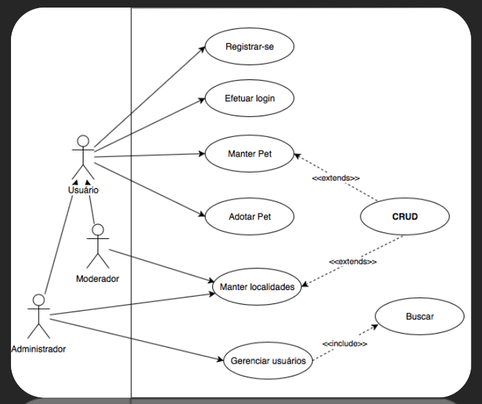
- A especificação basicamente trata das mensagens de erro dos status code entre os ranges 400 e 500. Além disso usa-se no header Content-Type o tipo **aplication/problem**, incluindo o formato da serialização da mensagem, json ou xml.

```
{
    "type": "Dados incorretos",
    "title": "Número de CPF inválido.",
    "detail": "O número de CPF digitado é um número de CPF inválido.",
    "status": 400,
    "instance": "/cliente/msgs/400"
}
```



BDD ou Caso de Uso?

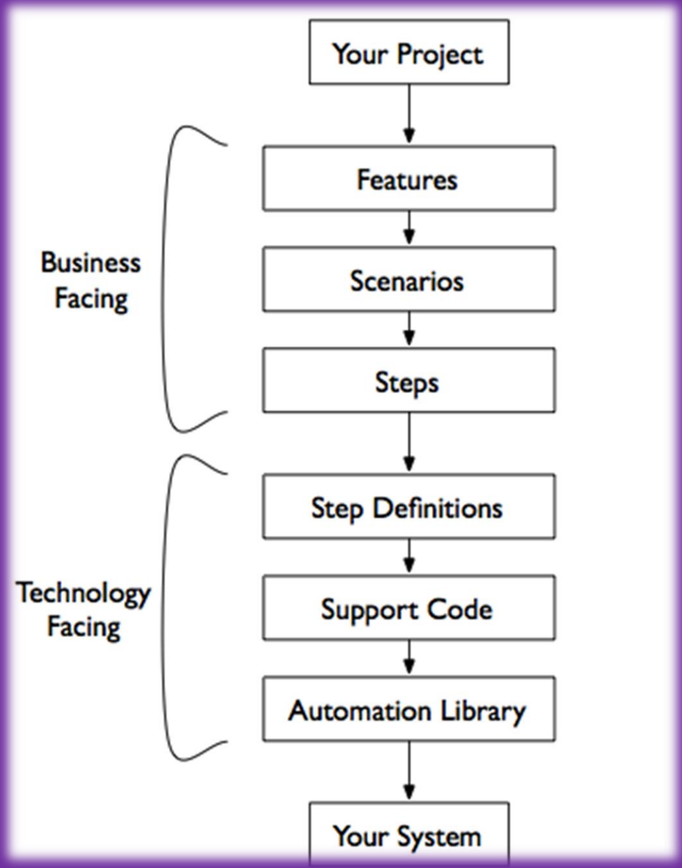
- O cliente e/ou time decide
- O importante são os insumos que serão levantados nas reuniões;
- QA vai documentar da forma acordada com o restante do time.



Documentação para Backend e APIs

Insumos para o desenvolvimento

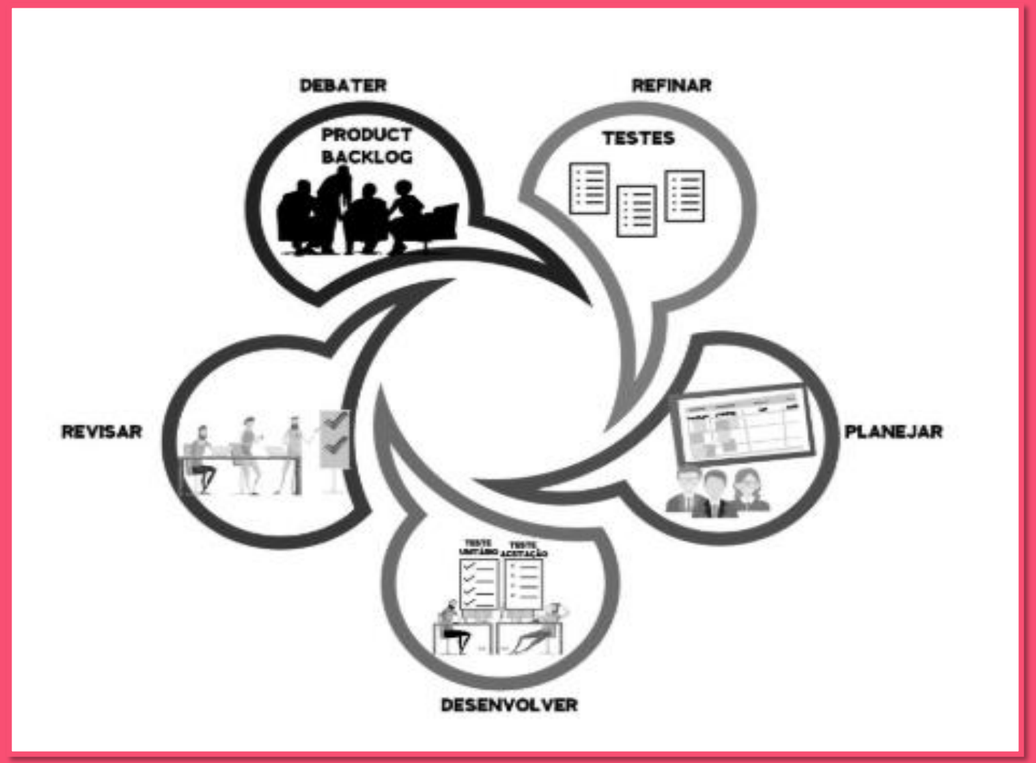
- Contratos definidos;
- Os tipos das propriedades do contrato;
- Qual endpoint e qual o padrão do endpoint;
- Métodos HTTP;
- Definição dos status code seguindo os padrões de API Rest;
- Mensagem de retorno para cada cenário.



CDD – Collaboration Driven Development

O objetivo desse framework, desenvolvido por Carol Vilas Boas e Rodrigo Vieira, seguindo a visão do framework Scrum, é diminuir o déficit de qualidade de software de cada squad, pensando em incluir o QA de ponta a ponta, aumentando a interação do time e ganhando agilidade. Através de pair com POs, QAs e Devs no desenvolvimento de tarefas e eventos, é possível implantar a qualidade de software dentro do processo de desenvolvimento como um todo.

QA e Dev atuando de forma colaborativa

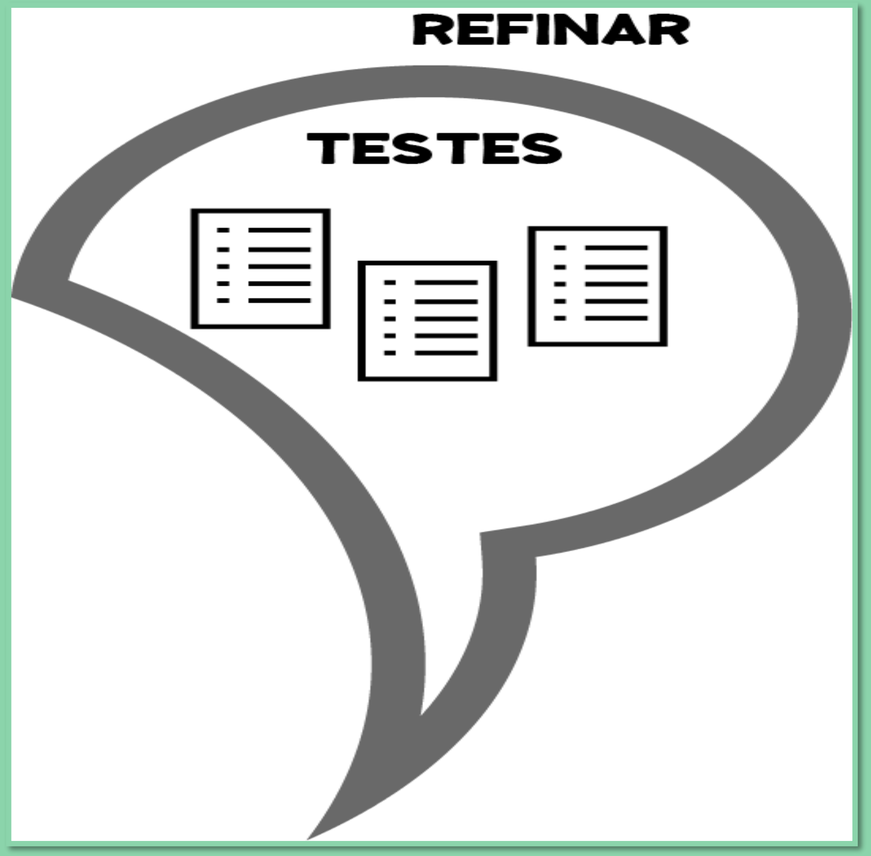


- Qualidade como parte da entrega, não do processo;
- Desenvolvimento com base na comunicação entre a área de negócios e tecnologia;
- Usa os testes como ferramenta de análise de requisitos;

Etapa 1 - Debater

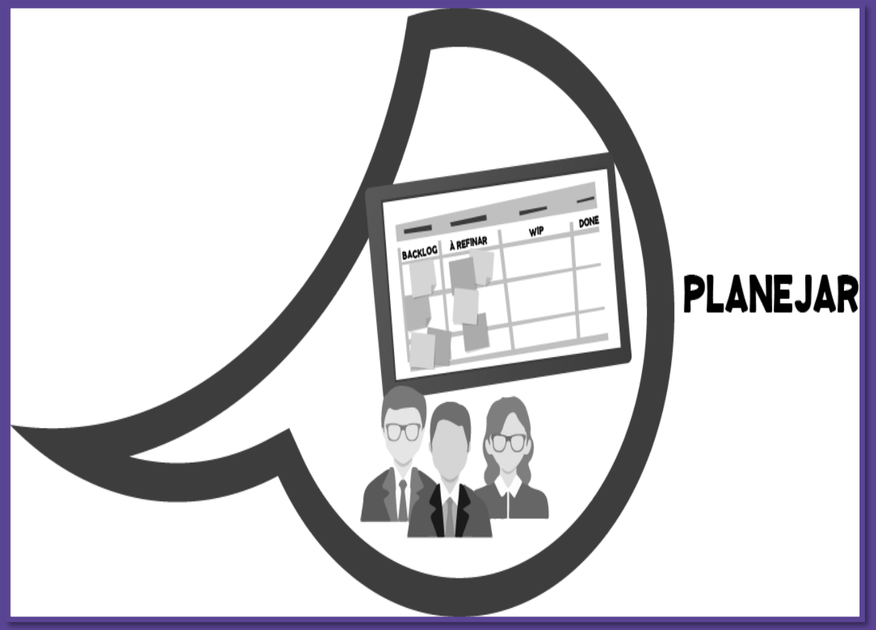
- Utilizar o conhecimento coletivo para construção da documentação. (P.O, Negócios, QA...)
- QA, nesse momento, representa as dores do Dev Team
- Mapeamento de fluxos e possíveis impactos (técnicos e de negócios)
- Documentação alto nível para definição do comportamento da funcionalidade

Etapa 2 - Refinar



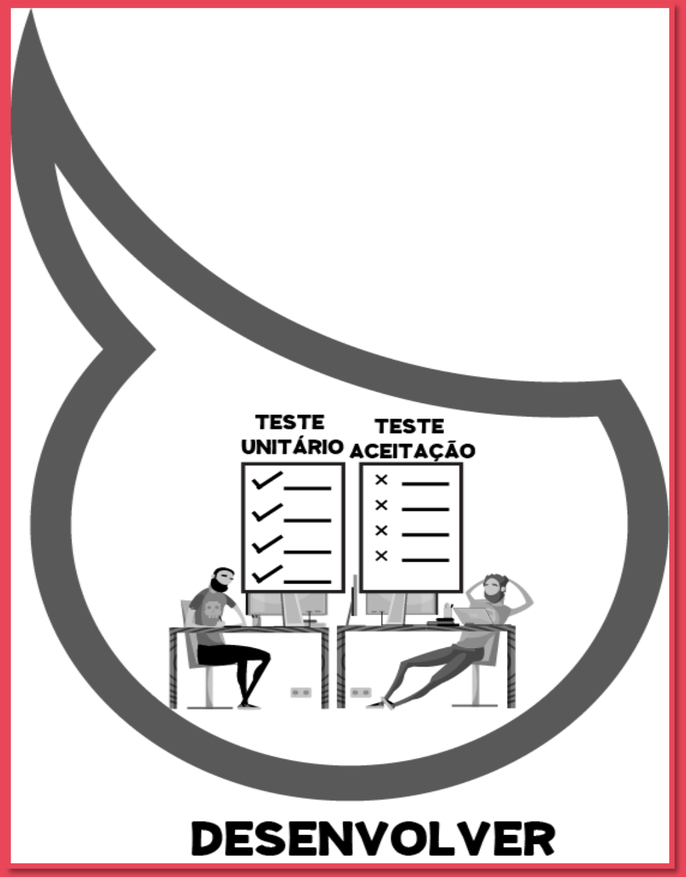
- Envolvimento do Dev Team para entendimento de COMO será desenvolvido.
- QA e P.O. defendendo o entendimento e o objetivo de negócios
- Documentação já está mais detalhada, entrando em um nível de histórias. Com objetivos específicos, mas ainda high level
- Convergindo o entendimento
- Mapeamento de dependências técnicas

Etapa 3 - Planejar



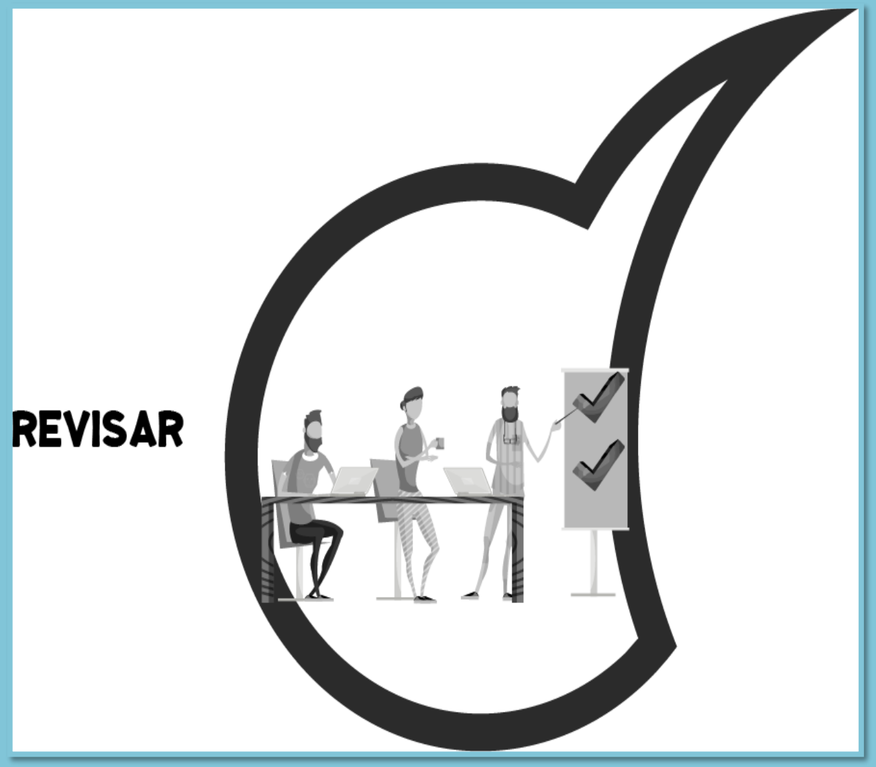
- Documentação contemplando os cenários de testes
- Decisão do tipo de teste:
  - Unitário;
  - Serviço;
  - Funcional.
- Inspeção e Adaptação das atividades para a próxima sprint
- Output – Sprint Backlog

Etapa 4 - Desenvolver



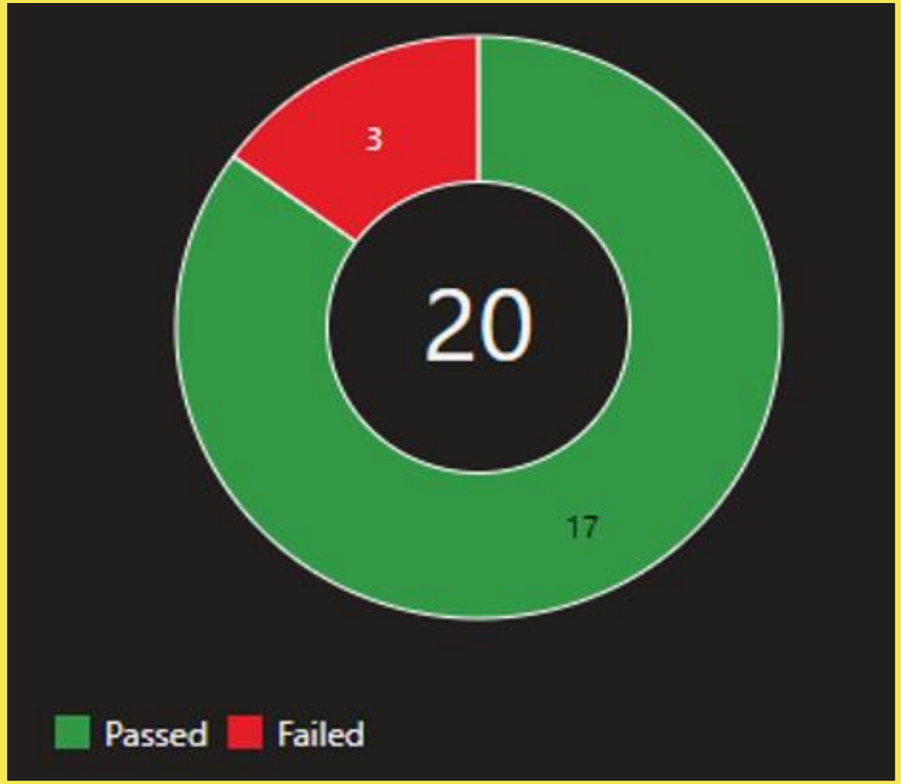
- Pair programming
- Testes Unitários padronizados e revisados pelo QA
- Trabalho colaborativo (Linguagem agnóstica)
- Escolha da linguagem adequada (sem sopa de letrinhas)
- Refactoring é bem vindo

Etapa 5 - Revisar



- Não precisa esperar a Sprint Review
- Entendimento de COMO foi desenvolvido
- Construção do conhecimento coletivo
- Documentação (na medida certa!)

Nossos resultados atuando de forma colaborativa



Test Plan no Azure DevOps

Postman com Newman

Agradecemos a sua participação!

Dúvidas?