

40431: Modelação e Análise de Sistemas

Teste e garantia de qualidade no processo de software

Ilídio Oliveira

v2022-01-21 | TP12

Objectivos de aprendizagem

Identificar atividades de validação e verificação no SDLC

Descrever as camadas do teste da pirâmide

Descrever o âmbito dos teste unitários, de integração, de sistema e de aceitação

Explicar o ciclo de vida do *TDD*

Explique como as atividades do QA são inseridas no processo de desenvolvimento numa abordagem clássica e em métodos ágeis

Relacionar os critérios de aceitação das histórias (*user stories*) com testes ágeis

Explicar a dinâmica das equipas que adotam processos de CI

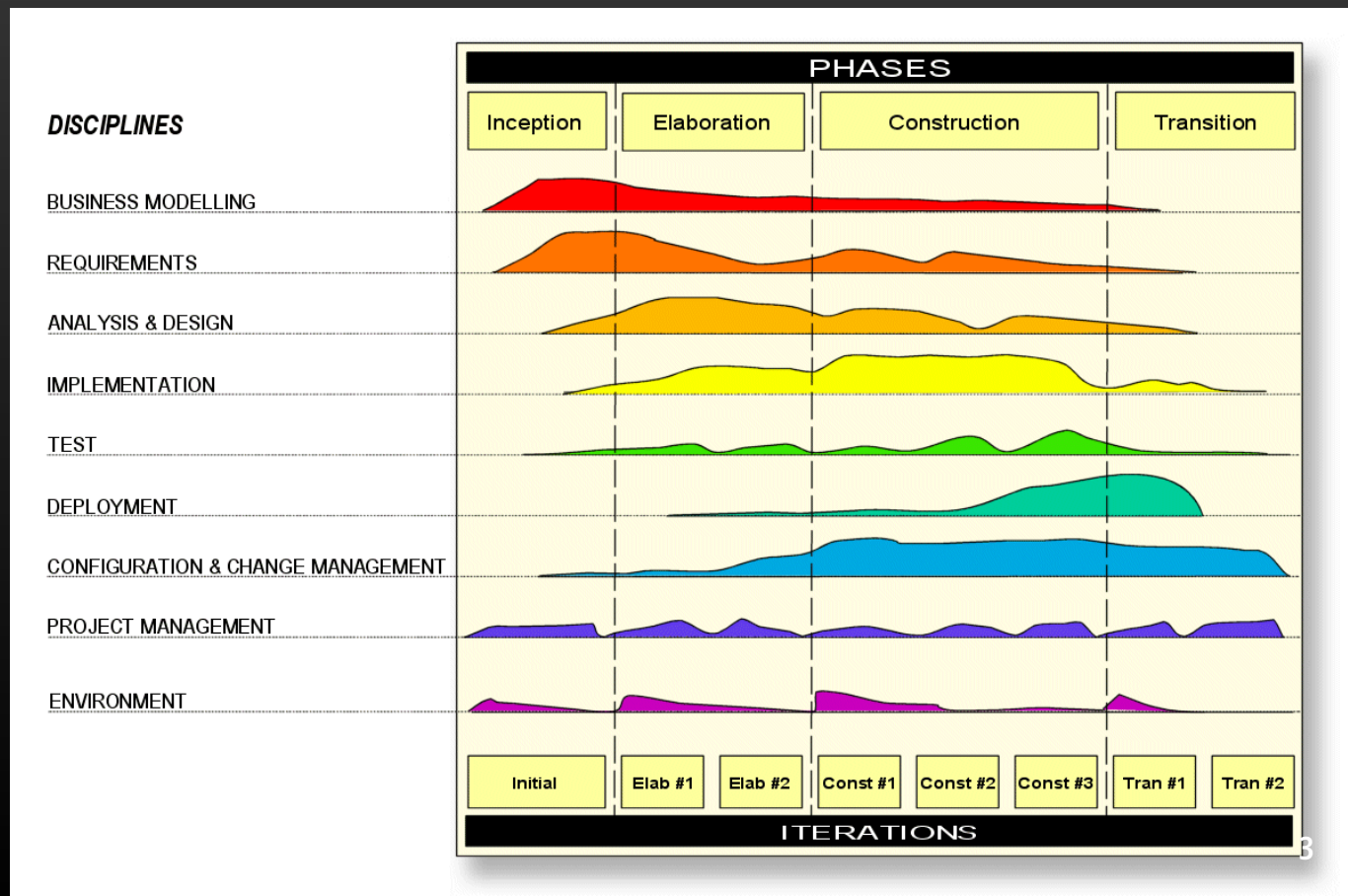
Relacionar as práticas de CI/CD com a abordagem ágil ao desenvolvimento

Explicar o sentido da expressão “continuous” em CI/CD

OpenUP/Unified Process activities

A abordagem do UP ao SDLC pode ser visualizada como uma matriz, cruzando diferentes disciplinas técnicas com iterações evolutivas no projecto. (Nota: fases UP ≠ fases SDLC)

A Construção é tipicamente a etapa maior, e inclui atividades de especificação, implementação e validação.

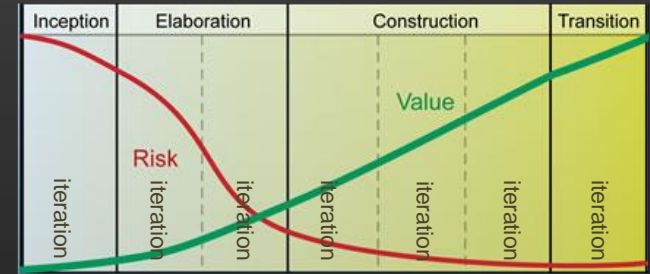


Construção: implementar o produto

**Definir, conceber, implementar e testar
cada vez mais cenários, de forma
incremental**

Evoluir progressivamente a arquitectura
executável para completar o sistema

Evoluir a arquitectura à medida que se progride



**Demonstrações frequentes e instalações
parciais**

A estratégia de implementação parcial depende
muito do sistema que se constrói

**Construção diária com processo de
construção automatizado**

**Poderá ter de ter uma equipa de teste
separada, se tiver**

Ambientes de teste complexos

Sistemas de segurança ou de missão crítica

Credit: Per Kroll (IBM)

Algumas ideias do desenvolvimento ágil

QUICK LOOK

What is it? Agile software engineering **combines a philosophy and a set of development guidelines.**

The philosophy encourages customer satisfaction and early incremental delivery of software; small, highly motivated project teams; informal methods; minimal software engineering work products; and overall development simplicity. The development guidelines stress **delivery over analysis and design** (although these activities are not discouraged), and active and continuous communication between developers and customers.

Who does it? Software engineers and other project stakeholders (managers, customers, end users) work together on an agile team—a team that is self-organizing and in control of its own destiny. An agile team fosters communication and collaboration among all who serve on it.

Why is it important? The modern business environment that spawns computer-based systems and software products is fast-paced and ever-changing. Agile software engineering represents a reasonable alternative to

conventional software engineering for certain classes of software and certain types of software projects. It has been demonstrated to deliver successful systems quickly.

What are the steps? Agile development might best be termed “software engineering lite.” The basic framework activities—communication, planning, modeling, construction, and deployment—remain. But they morph into a minimal task set that pushes the project team toward construction and delivery (some would argue that this is done at the expense of problem analysis and solution design).

What is the work product? Both the customer and the software engineer have the same view—the only really important work product is an operational “software increment” that is delivered to the customer on the appropriate commitment date.

How do I ensure that I’ve done it right? If the agile team agrees that the process works, and the team produces deliverable software increments that satisfy the customer, you’ve done it right.

O dinamismo do mercado obriga a igual dinamismo das TIC/desenvolvimento; especialmente quando os produtos do desenvolvimento passam a assumir um papel fundamental na criação das vantagens competitivas do negócio!

A transformação digital (competitiva) obriga a uma eng.a de software competitiva.

Velocidade “furiosa”?



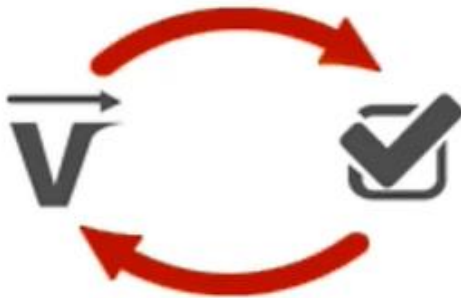
Greater speed may generate more risk and less quality...



Para avançar depressa e com segurança, é preciso preparar a “máquina”: mexer no próprio processo de engenharia de sw para produzir de forma ágil e com qualidade.

... but

Velocity = Direction + Speed



quick feedback
improves direction
which improves quality which improves
speed
which improves feedback

É indispensável considerar as práticas que podem fomentar e medir a qualidade do produto

GARANTIA DE QUALIDADE DE SOFTWARE

conjunto de atividades (práticas) para controlar e monitorizar o processo de desenvolvimento de software para atingir os objetivos do projeto com um certo nível de confiança em termos de qualidade

CONTROLO DE QUALIDADE DE SOFTWARE

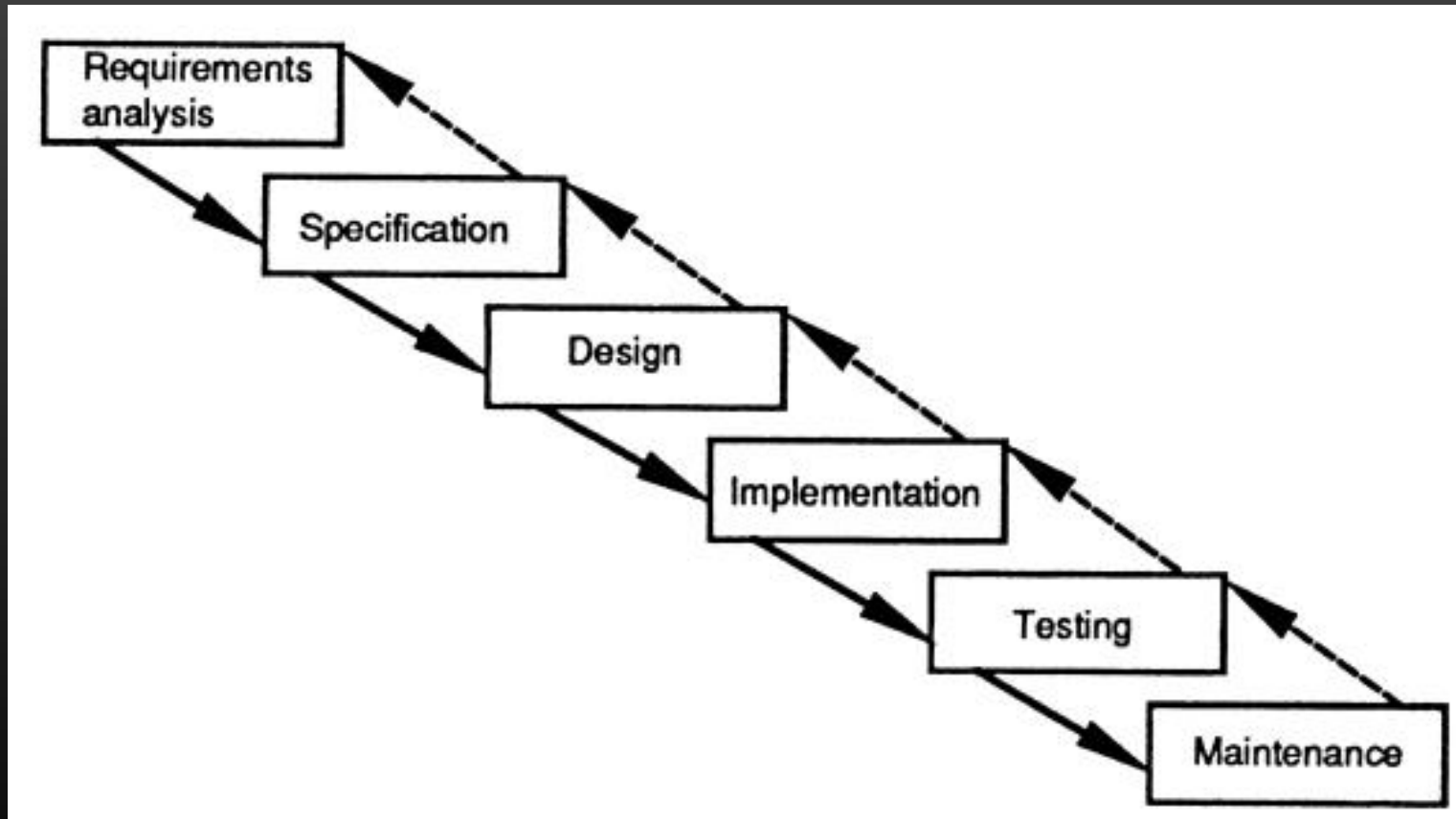
Avalia se os produtos de software estão dentro dos padrões de qualidade definidos recorrendo a inspeções e diferentes tipos de testes

Sw Quality Assurance != Sw Quality Control

O SQC visa detetar e corrigir defeitos. A SQA tem como objetivo preveni-los.

A garantia de qualidade é parte integrante de um processo de engenharia

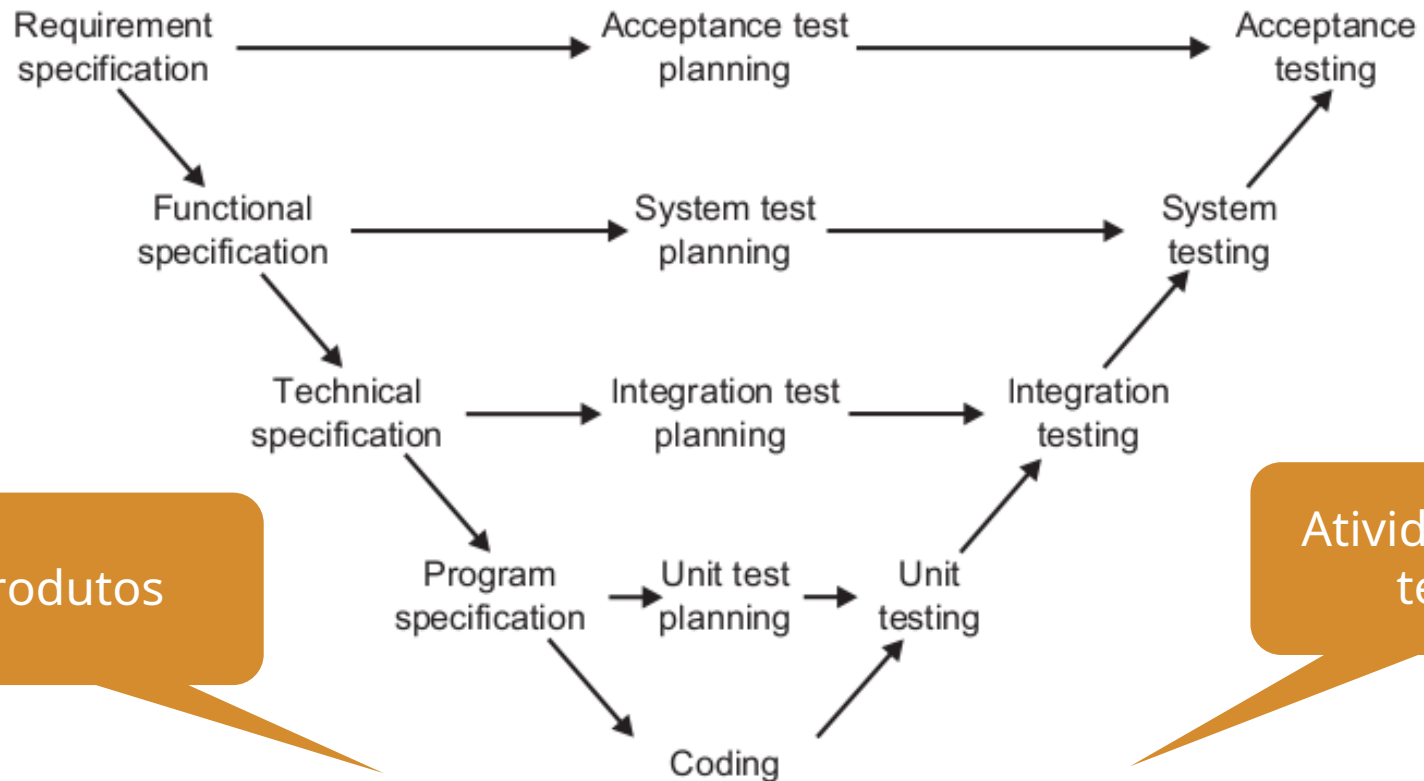
Abordagem de engenharia "clássica": Modelo waterfall



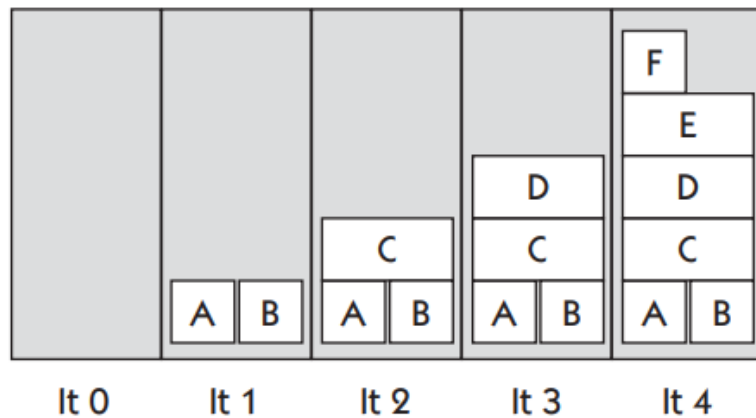
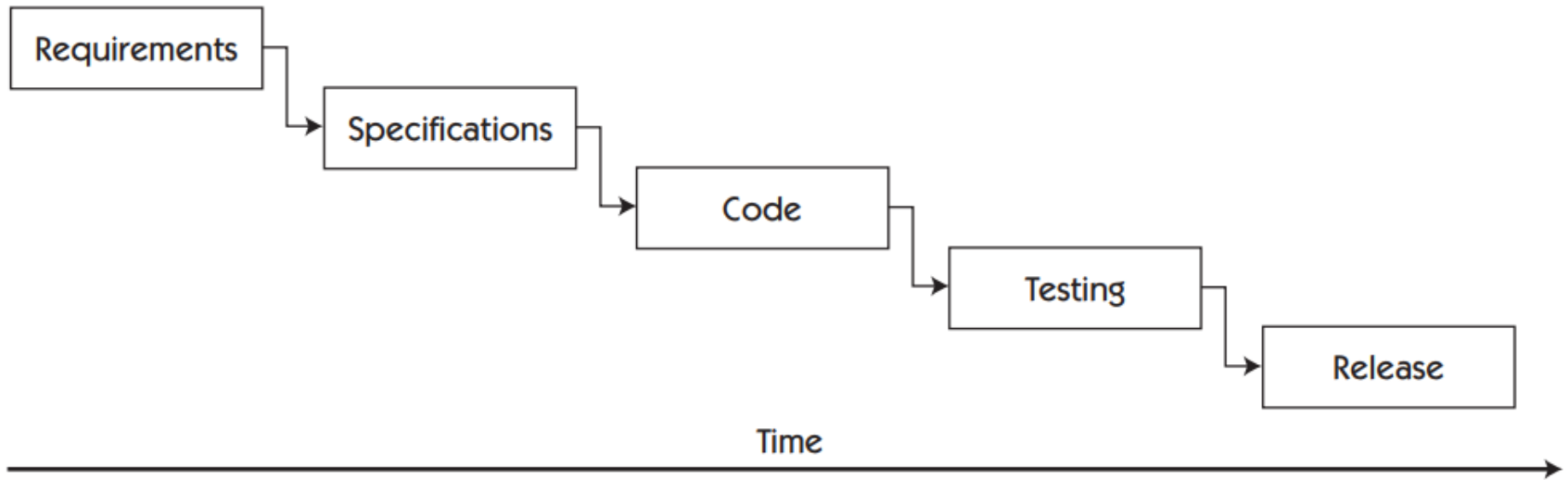
W. Royce, "Managing the Development of Large Software Systems," *Proc. Westcon*, IEEE CS Press, 1970, pp. 328-339.

Ciclo de vida dos testes e o ciclo de vida do desenvolvimento do sw na abordagem sequencial (V-Model)

Figure 2.2 V-model for software development



Phased or gated—for example, Waterfall



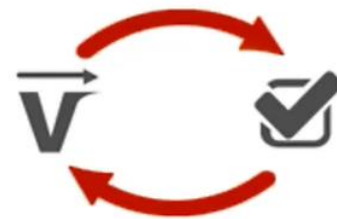
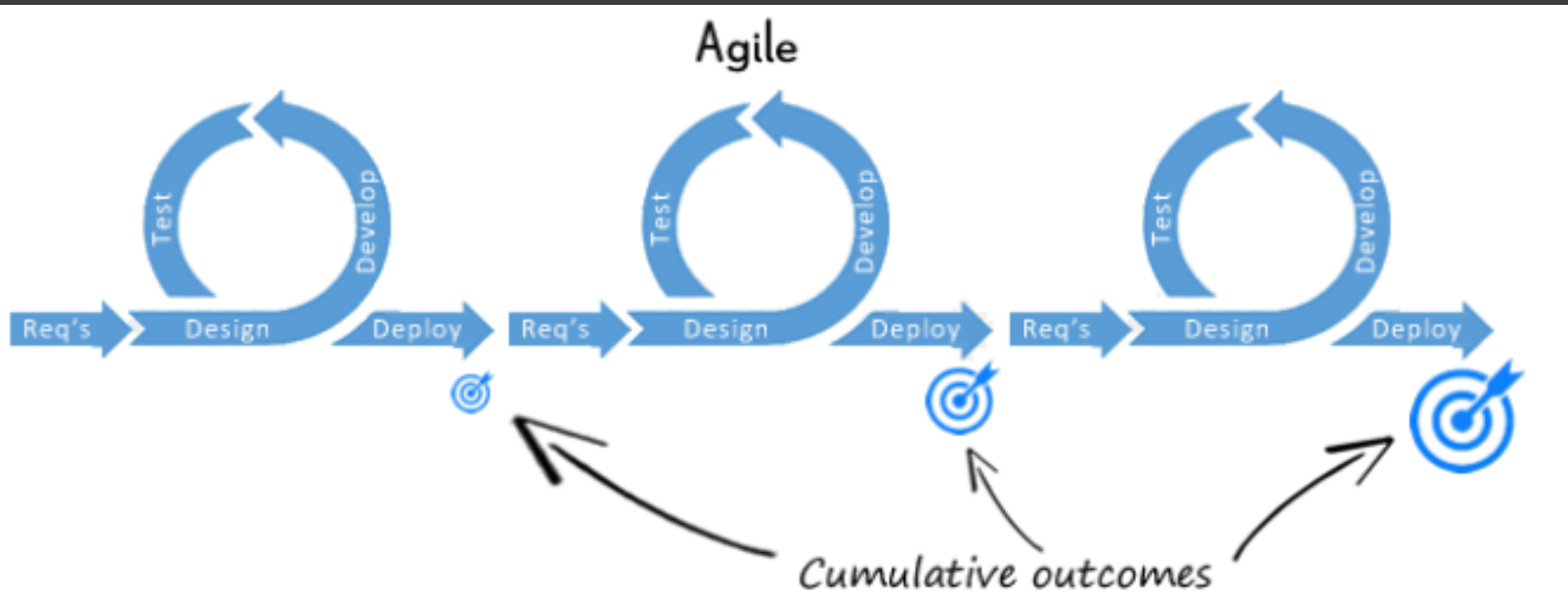
Agile:

Iterative & incremental

- Each story is expanded, coded, and tested
- Possible release after each iteration

Figure 1-4 Traditional testing vs. agile testing

Na abordagem ágil, a garantia de qualidade tem de ser aplicada em cada ciclo



quick feedback
improves direction
which improves quality which improve
speed
which improves feedback

O papel dos testes de software

Atividades de verificação e de validação

VERIFICAÇÃO: ESTAMOS A FAZER O SISTEMA DA MANEIRA CERTA?

Verificar os produtos contra as suas especificações

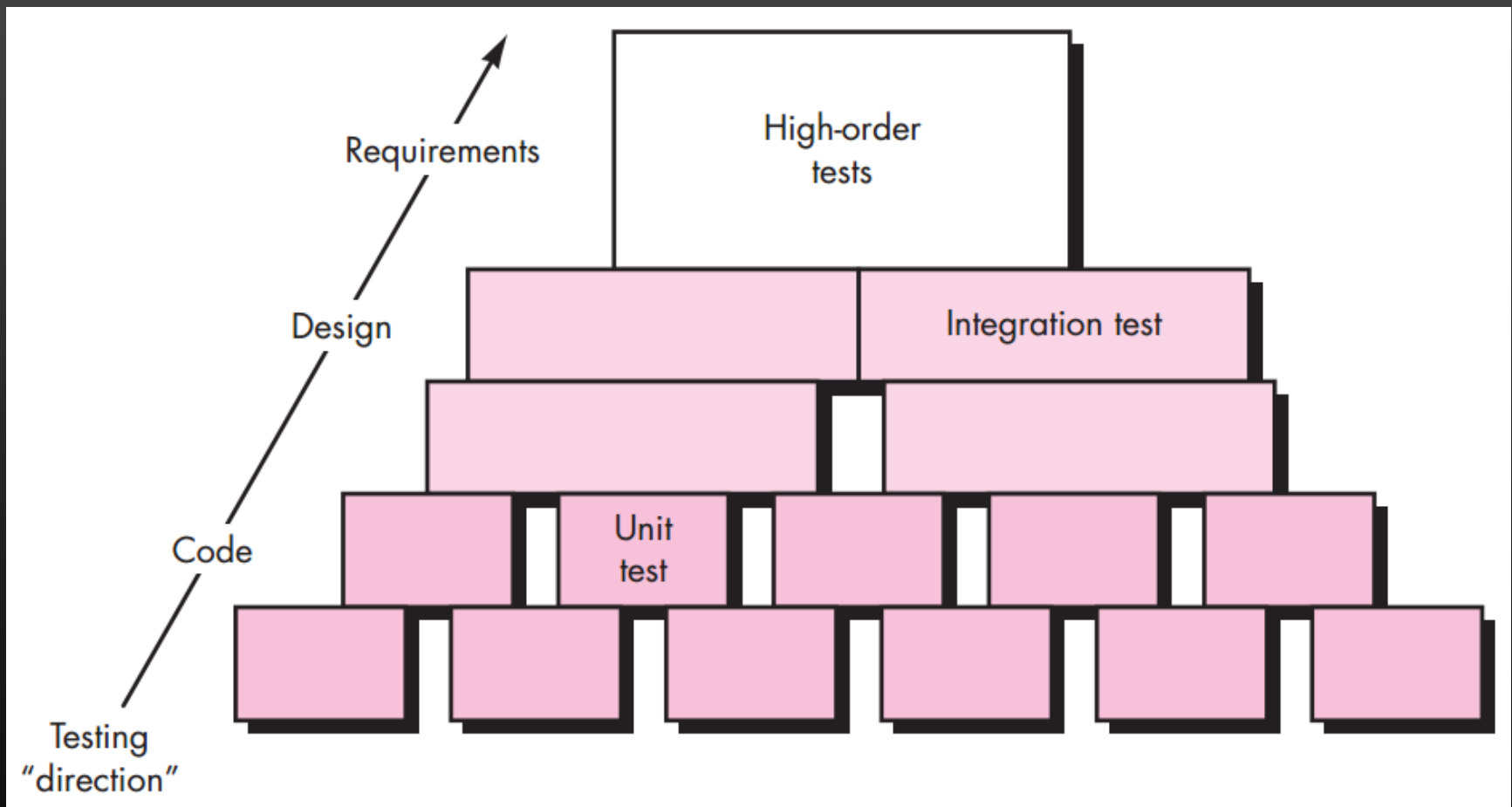
Verificar a coerência entre módulos

Verificar contra as melhores práticas da indústria

...

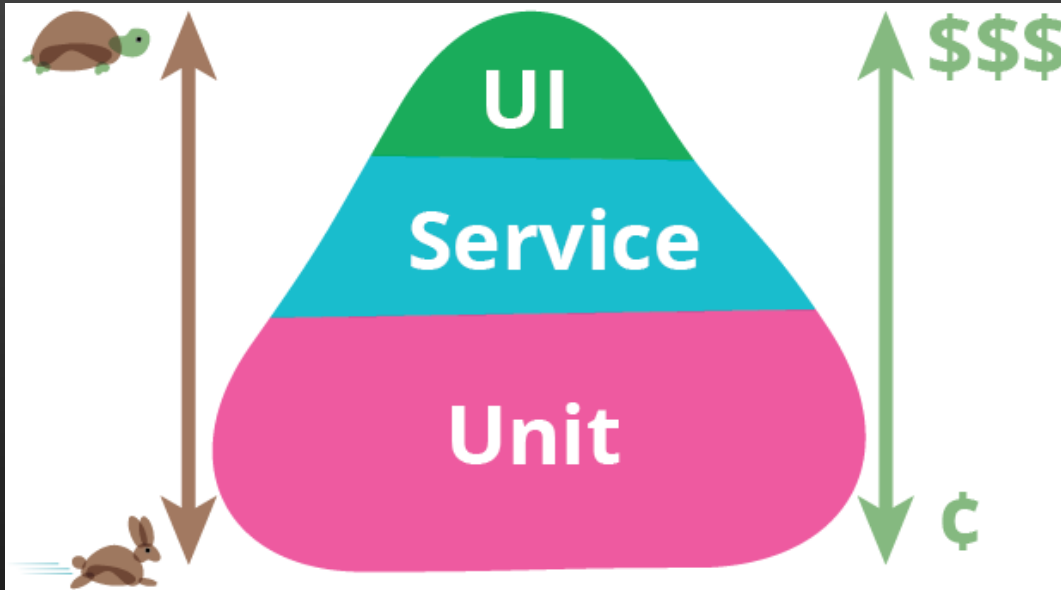
VALIDAÇÃO: ESTAMOS A FAZER O SISTEMA CERTO?

Verificar os produtos contra os requisitos e expectativas do utilizador



Os testes começam a nível de unidades e vão "alargando" o âmbito.

A pirâmide dos testes

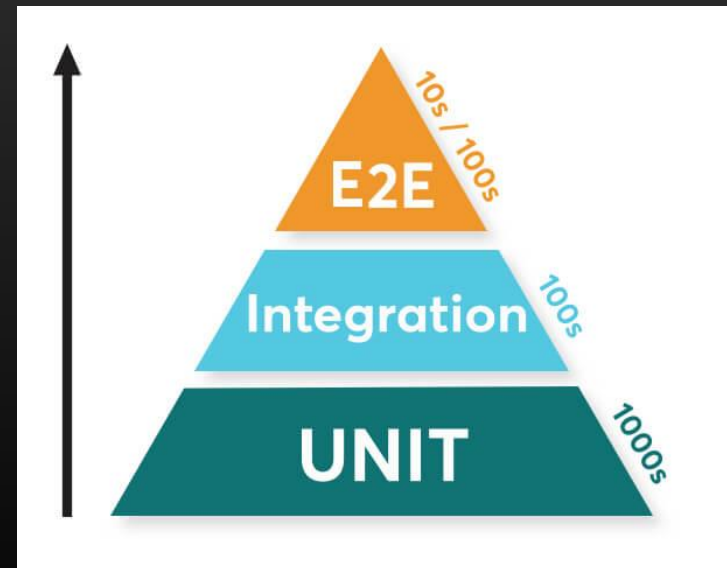


<https://martinfowler.com/bliki/TestPyramid.html>

Há diferentes níveis de teste; não é que uns sejam mais importantes que outros...

Os testes “de baixo” são mais granulares e específicos; os testes “de cima” são mais abrangentes (e menos finos na capacidade de deteção da *root cause*).

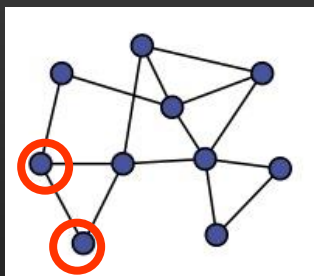
Uma estratégia de testes deve incluir diversos tipos de testes.



<https://www.blazemeter.com/blog/agile-development-and-testing-an-introduction>

Diferentes técnicas de teste são apropriadas em diferentes momentos/âmbitos

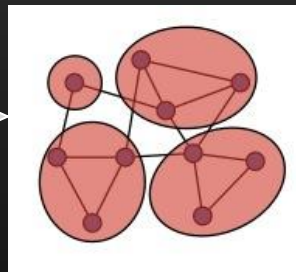
Teste de unidade



Cada módulo faz o que é suposto fazer?

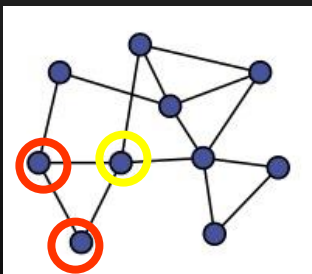
gestão da complexidade

Testes de integração



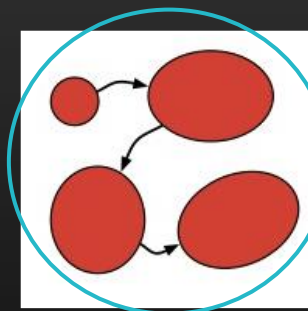
Obtém-se os resultados esperados quando as peças são montadas?

Testes de integração



O programa satisfaz os requisitos?

Aceitação / Testes Funcionais



Testes de sistema

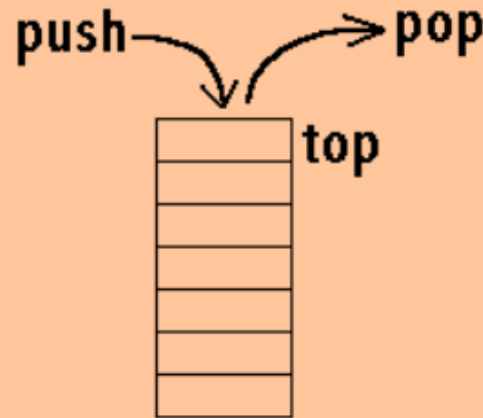
O sistema globalmente funciona como esperado, no ambiente alvo?

Programador vs cliente

Exemplo de um teste unitário: a estrutura de dados Pilha

Qual é o contrato “funcional” desta classe?

O que se pode/deve testar para confirmar que o contrato está bem implementado?



Operations

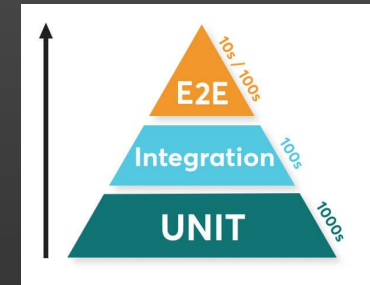
- `push(x)`: add an item on the top
- `pop`: remove the item at the top
- `peek`: return the item at the top (without removing it)
- `size`: return the number of items in the stack
- `isEmpty`: return whether the stack has no items

Exemplo de teste unidade: Verificar o contrato da Stack/Pilha

- Uma pilha está vazia na construção
- Uma pilha tem tamanho 0 na construção
- Depois de n inserções para uma pilha vazia, $n > 0$, a pilha não está vazia e seu tamanho é n
- Se alguém inserir x e a seguir retirar, o valor retornado é x , o tamanho da pilha é diminuído em um.
- Se alguém inserir x e de seguida espreitar (`peek`), o valor devolvido é x , mas o tamanho permanece o mesmo
- Se o tamanho é n , então depois de n *pops*, a pilha está vazia e tem um tamanho 0
- Tirar de uma pilha vazia lança uma `NoSuchElementException`
- Espreitar uma pilha vazia lança uma `NoSuchElementException`
- Apenas para pilhas com tamanho limitado: inserir para uma pilha cheia lança uma `IllegalStateException`

→ See also: Ray Toal's notes.

(demonstração)

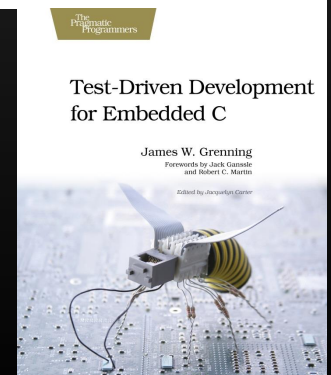


Prática ágil: *Test-driven development*

“Debug Later Programming”

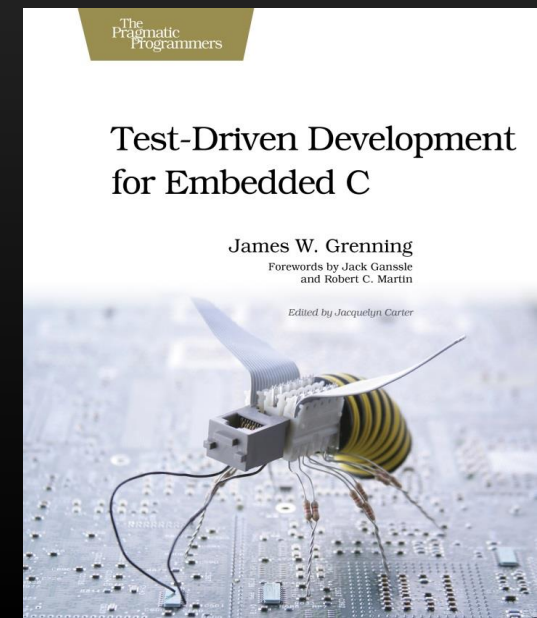
We’ve all done it—written a bunch of code and then toiled to make it work. **Build it and then fix it.** Testing was something we did after the code was done. It was always an afterthought, but it was the only way we knew.

We would spend about half our time in the unpredictable activity affectionately called *debugging*. Debugging would show up in our schedules under the guise of test and integration. It was always a source of risk and uncertainty. Fixing one bug might lead to another and sometimes to a cascade of other bugs. We’d keep statistics to help predict

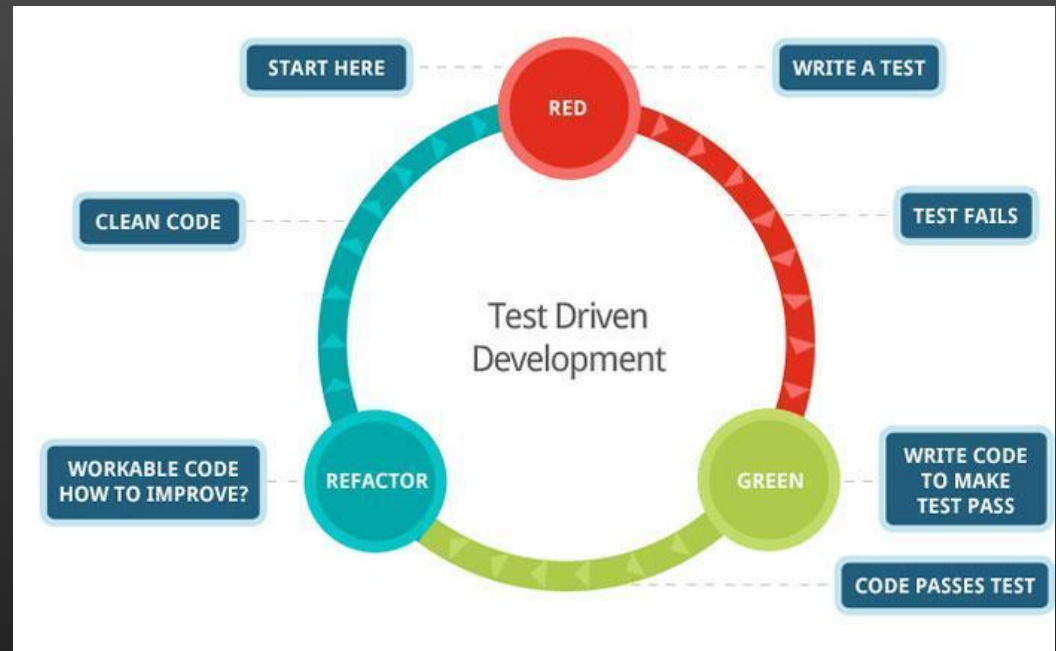


"Test-driven development"

Test-Driven Development is a technique for building software incrementally. Simply put, no production code is written without first writing a failing unit test. Tests are small. Tests are automated. Test-driving is logical. Instead of diving into the production code, leaving testing for later, the TDD practitioner expresses the desired behavior of the code in a test. The test fails. Only then do they write the code, making the test pass.



O ciclo do TDD



At the core of TDD is a repeating cycle of small steps known as the TDD microcycle. Each pass through the cycle provides feedback answering the question, does the new and old code behave as expected? The feedback feels good. Progress is concrete. Progress is measurable. Mistakes are obvious.

The steps of the TDD cycle in the following list are based on Kent Beck's description in his book *Test-Driven Development* [Bec02]:

1. Add a small test.
2. Run all the tests and see the new one fail, maybe not even compile.
3. Make the small changes needed to pass the test.
4. Run all the tests and see the new one pass.
5. Refactor to remove duplication and improve expressiveness.

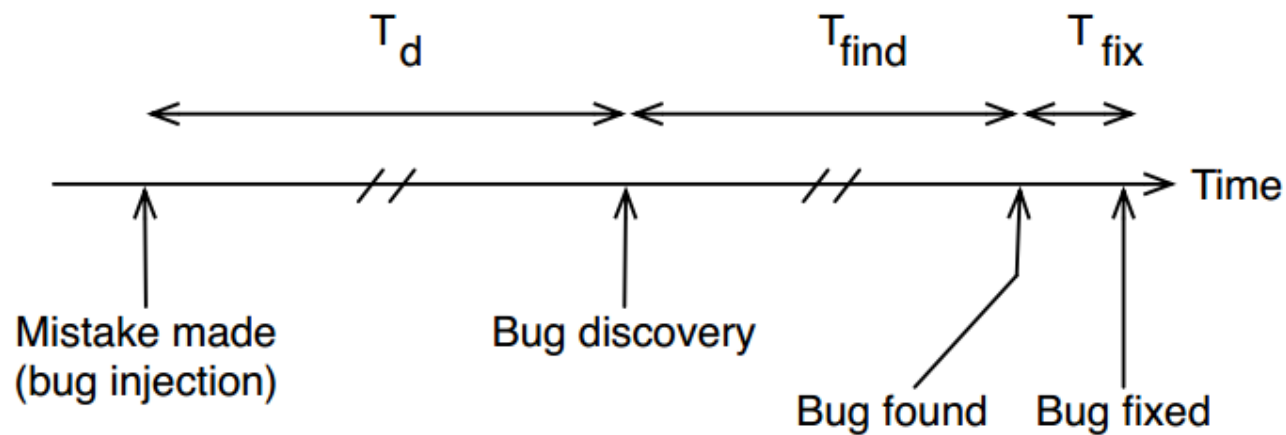


Figure 1.1: Physics of Debug-Later Programming

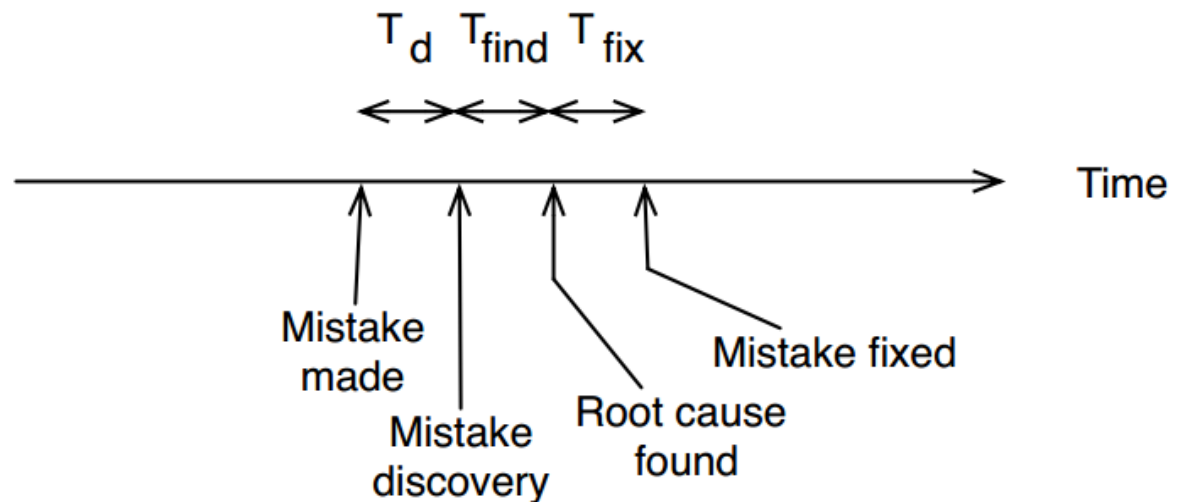


Figure 1.2: Physics of Test-Driven Development

The benefits of TDD: an investigation

A summary of selected empirical studies of test-driven development: industry participants*

Family of studies	Type	Development time analyzed	Legacy project?	Organization studied	Software built	Software size	No. of participants	Language	Productivity effect
Sanchez et al. ⁶	Case study	5 years	Yes	IBM	Point-of-sale device driver	Medium	9–17	Java	Increased effort 19%
Bhat and Nagappan ⁷	Case study	4 months	No	Microsoft	Windows networking common library	Small	6	C/C++	Increased effort 25–35%
	Case study	≈7 months	No	Microsoft	MSN Web services	Medium	5–8	C++/C#	Increased effort 15%
Canfora et al. ⁸	Controlled experiment	5 hours	No	Soluziona Software Factory	Text analyzer	Very small	28	Java	Increased effort by 65%
Damm and Lundberg ⁹	Multi-case study	1–1.5 years	Yes	Ericsson	Components for a mobile network operator application	Medium	100	C++/Java	Total project cost increased by 5–6%

Prática ágil: Gestão do trabalho (e testes) com “user stories”

Stories, use cases, scenarios



FIGURE 8:
THE RELATIONSHIP BETWEEN THE FLOWS AND THE STORIES

Figure 5. Capturing the properties of a use case and its slices using sticky notes.



a use case and its properties captured on a sticky note

7.1 select and buy
1 product

flows: BF
test: 1 product,
default payment,
valid details

5

7.3 support systems
unavailable

flows: BF, A9, A10,
A1, A12
test: select product,
provide information,
disconnect each
system in between¹³

7.2 select and buy
100 products

flows: BF
test: 100 products,
default payment,
valid details

5

some slices from the
use case captured on
their own sticky notes

"slices" corresponde ao
conceito de "user stories"



Clarificar os cenários & regras.

Arranjar exemplos representativos (para validar a implementação)

Por vezes: discutir sobre protótipos das interações.

Este trabalho é feito para as *stories* "em cima da mesa", não para todo o projeto.

Figure 1-5 A developer discusses an issue with customers

História e condições de aceitação

História: O visitante pesquisa o livro por nome de autor

Sendo um visitante do site da livraria

Quero pesquisar por nome de autor

De modo a ver as obras e novidades de um autor.

①

②

Cenário 1: Pesquisa com sucesso

Dado que estou na página de entrada da Fnac.pt

E insiro o nome do autor "Valério Romão" no campo de pesquisa

Quando seleciono o botão para iniciar pesquisa

Então a página de resultados inclui "Valério Romão" no título

E existe um livro chamado "Autismo" na lista

E existe um livro chamado "Cair Para Dentro" na lista.

③

Cenário 2: Pesquisa sem resultados

Dado que estou na página de entrada da Fnac.pt

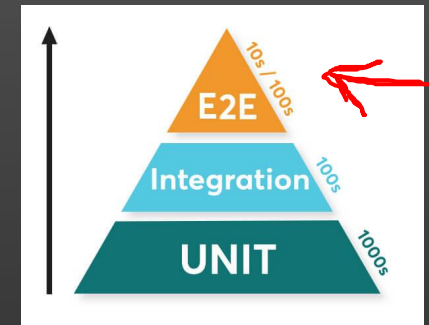
E insiro o nome do autor "askjfdenf kjewnjknkdsjn" no campo de pesquisa

Quando seleciono o botão para iniciar pesquisa

Então a página de resultados inclui "askjfdenf kjewnjknkdsjn" no título

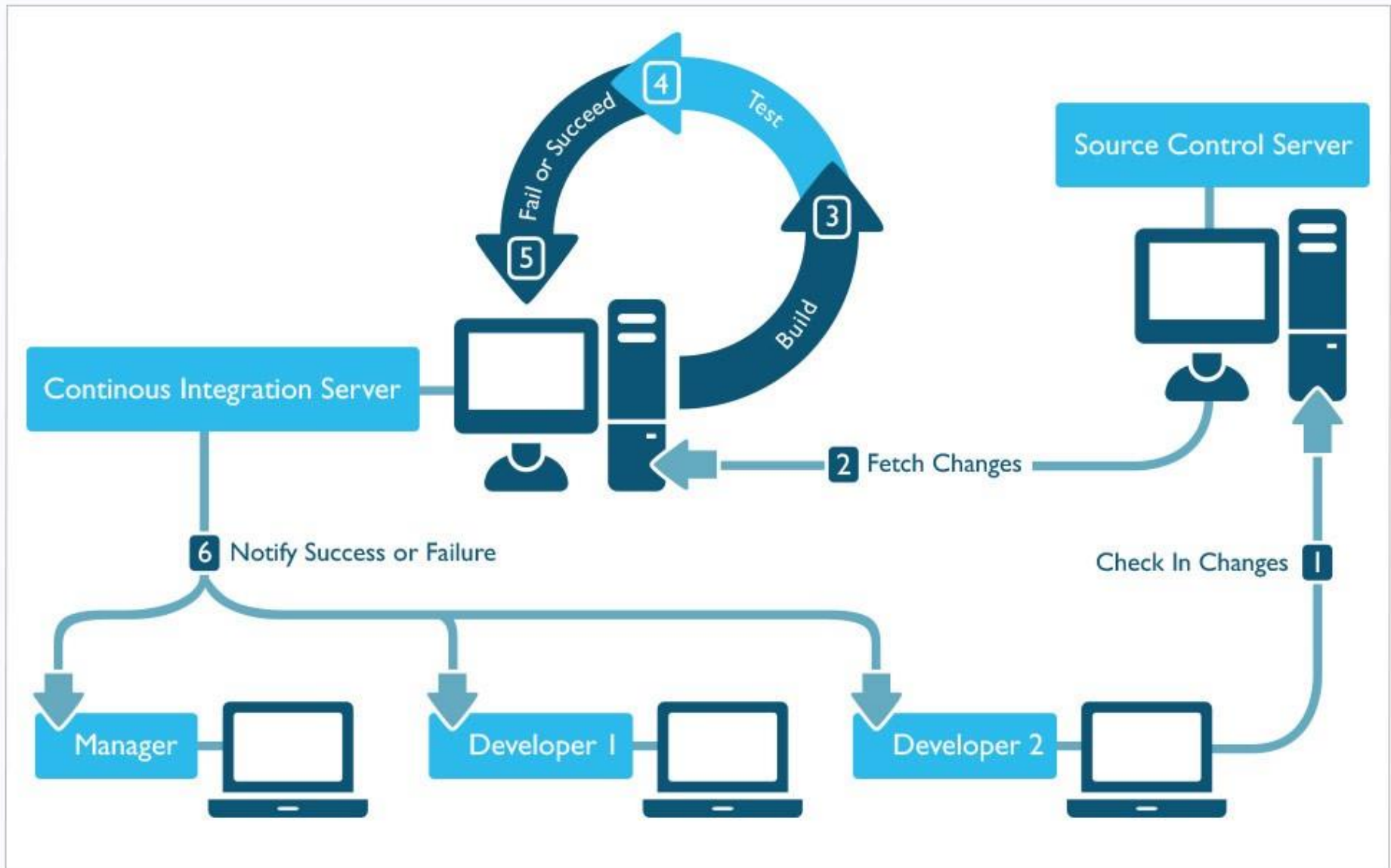
E existe a menção "Não há resultados para a tua pesquisa" na página

(Demonstração)



Test Suites			
Test Suites	Command	Target	Value
livaria-fnac-suite*	open	https://www.fnac.pt/	
• results-valerio-5 *	type	id=Fnac_Search	Valério Romão
Untitled Test Case *	click	//button[@type='submit']	
no-results *	click	//div[3]/div/div[2]	
	assertTitle	Valério Romão, uma pesquisa em Livros na F nac.pt	
	assertText	link=Autismo	Autismo
	assertText	link=Cair Para Dentro	Cair Para Dentro

Prática ágil: Integração contínua



<https://insights.sei.cmu.edu/devops/2015/01/continuous-integration-in-devops-1.html>

Integração contínua

A essência está na simples prática de todos na equipa integrarem frequentemente.

Cultura CI

Pronto para executar
Propriedade do código partilhada
Todos são corresponsáveis

Sem CI

Esforço grande e imprevisível para integrar
A aplicação, na maior parte do tempo, não está num estado executável



Integrar cedo e muitas vezes

Um “inferno” na integração

Práticas de integração contínua

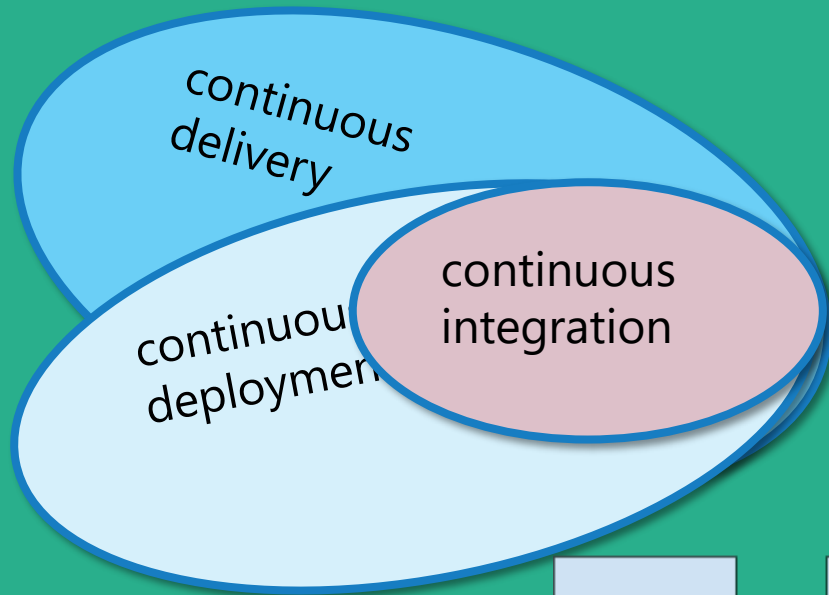
1. Os Programadores submetem regularmente para o repositório partilhado
2. As alterações no repositório de código são observadas e despoletam automaticamente processos de *build*
3. Feedback imediato sobre o estado das *builds* (resolver *builds* com problemas tem alta prioridade)
4. Opcional: distribuir os binários resultantes da *build* para uma repositório de referência.
5. Opcional: lançar os testes de integração/aceitação.

A CI torna o processo de desenvolvimento mais suave e menos arriscado

↓ "funciona no meu computador"

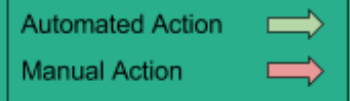
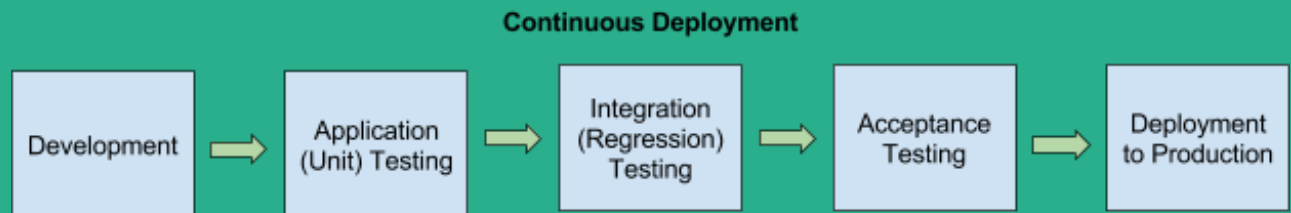
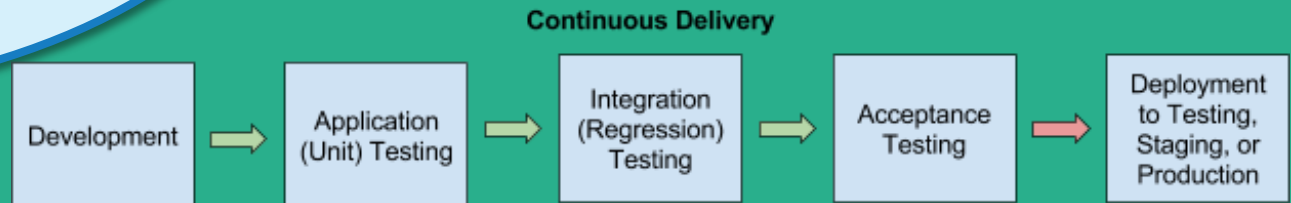
Deteção precoce de falhas

Termos relacionados (mas diferentes)

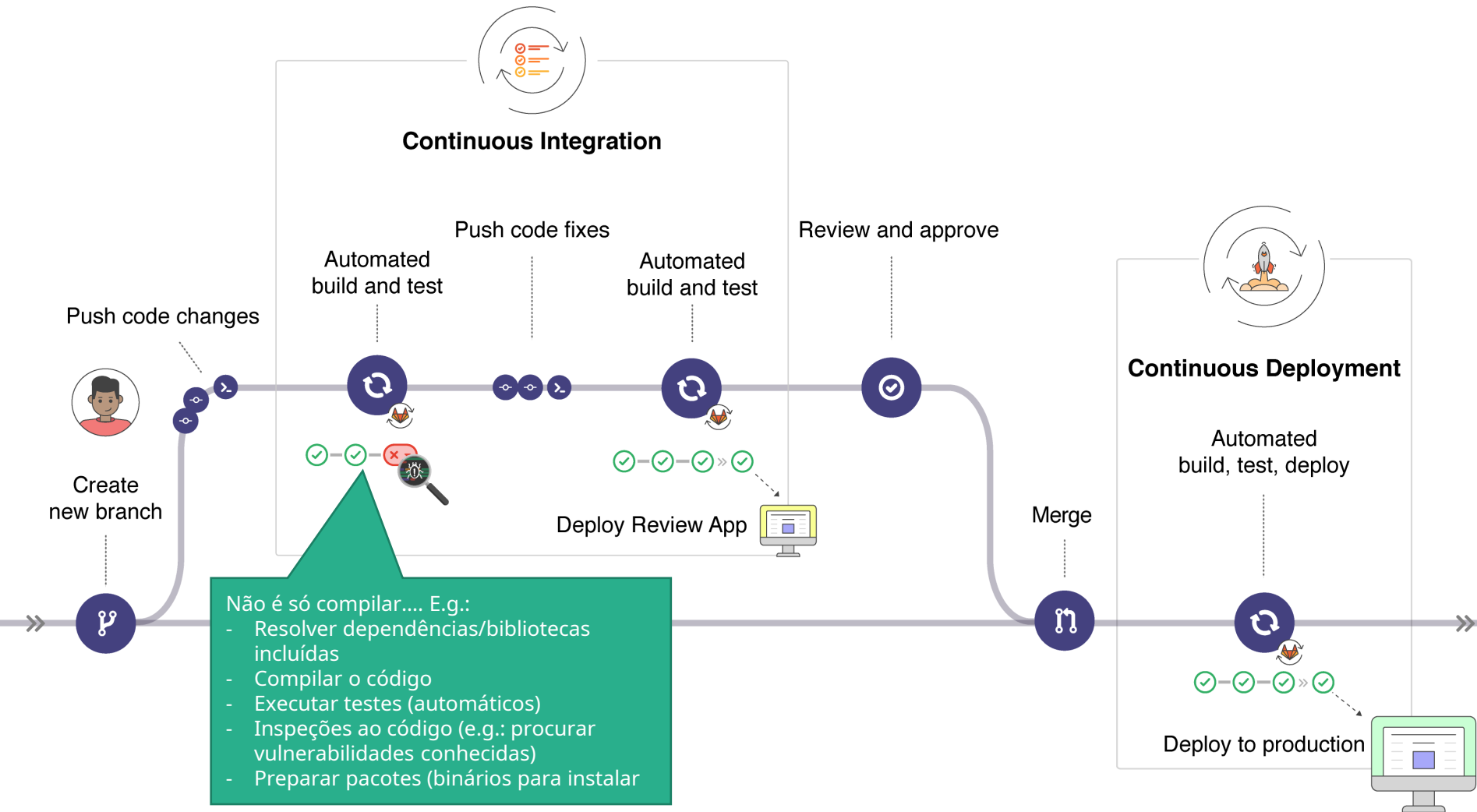


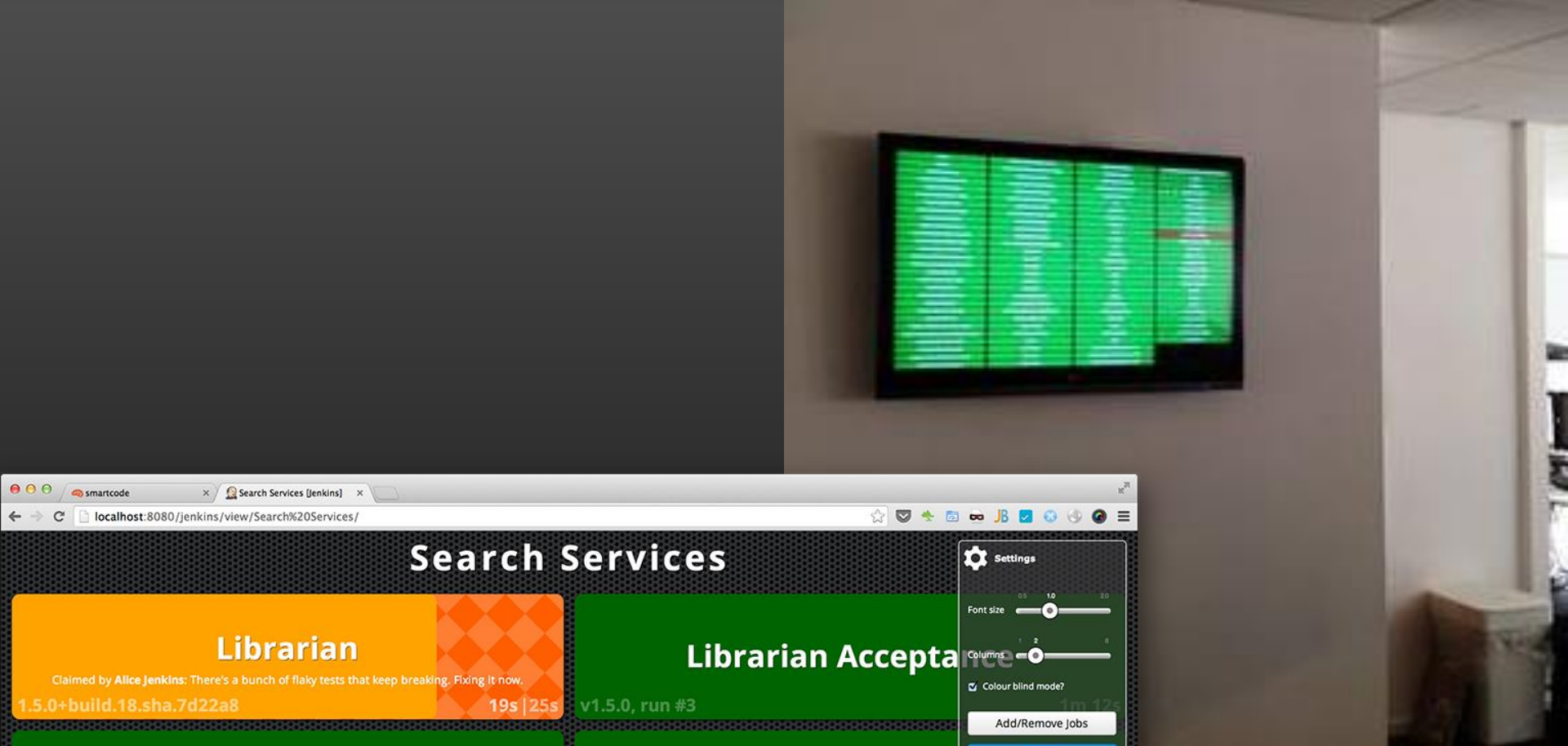
A equipa é capaz de fazer entregas frequentes para produção, mas pode optar por não colocar em produção de imediato (geralmente relacionado com a estratégia das empresas)

Sentido de “continuous” → pode ser feito a qualquer altura



Exemplo: Gitlab Ci/CD





smartcode

Search Services [Jenkins]

localhost:8080/jenkins/view/Search%20Services/

Search Services

Librarian

Claimed by Alice Jenkins: There's a bunch of flaky tests that keep breaking. Fixing it now.

1.5.0+build.18.sha.7d22a8

19s | 25s

Performance Benchmark

#1

1m 13s

Promote to UAT

Identified Problems: Invalid credentials

#12

22s | 2m 7s

Librarian Acceptance

v1.5.0, run #3

1m 13s

Promote to PROD

#2

1m 2s

Search API Contract Tests (PROD)

#1

17s

Search API Contract Tests (UAT)

#2

37s

Settings

Font Size 0.5 1.0 2.0

Columns 1 2

☒ Colour blind mode?

Add/Remove Jobs

Done

Brought to you by Jan Molak