Code Quality: 
*Raiz vs. Nutella*

Prof. Ingrid Nunes
Universidade Federal do Rio Grande do Sul (UFRGS)

Promob Dev Day – May 28, 2021
@rla4, the tech lead on Stack Overflow for Teams, explains why we ignored several best practices when building Stack Overflow's public site 12 years ago, and how we're modernizing our codebase to be approachable and powerful today.

Why?
Software development is still a highly human-based activity.
Two Messages

Process and practices are helpful for the ability of repetition and likelihood of success.

We must make sure we invest time to think through all artefacts that we produce.
FIRST
THINGS
FIRST
Let's talk about requirements

<table>
<thead>
<tr>
<th>ID:</th>
<th>UC_0001</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Nome:</strong></td>
<td>Fazer login</td>
</tr>
<tr>
<td><strong>Descrição:</strong></td>
<td>Usuário solicita fazer login no sistema que são validados.</td>
</tr>
<tr>
<td><strong>Pré-condições:</strong></td>
<td>nenhum usuário está logado no sistema.</td>
</tr>
<tr>
<td><strong>Pós-condições:</strong></td>
<td>usuário está registrado como logado no sistema.</td>
</tr>
</tbody>
</table>

**Fluxo Básico**

1. Este caso inicia quando o usuário seleciona a opção para fazer login no sistema.
2. O sistema solicita ao usuário o seu nome de usuário e a senha.
3. O usuário fornece os dados solicitados.
4. O sistema valida os dados fornecidos.
5. O sistema registra o usuário como usuário logado no sistema.
6. O sistema exibe a tela inicial do sistema.
7. O caso de uso termina.

**Fluxos Alternativos**

**Fluxo Alternativo 1 – Alternativa ao passo 4**
4a.1. O sistema verifica que os dados estão incorreto.
4a.2. O sistema notifica ao usuário.
4a.3. Retorna ao passo 2.
Let’s talk about requirements

As a registered user<br>
I want to login<br>
So that I can access all system features

Acceptance criteria:

?
The point is not the notation but the content!
Let’s talk about requirements

**ID: UC 0001**

**Nome:** Fazer login

**Descrição:** Usuário solicita fazer login no sistema, fornecendo usuário e senha, que são validados. Caso sejam válidos, usuário é logado no sistema. Caso contrário, o acesso é bloqueado. Mecanismos de segurança são adotados para impedir o acesso de usuários maliciosos.

**Pré-condição:** usuários não estão logados no sistema.

**Pós-condição:** usuários estão logados no sistema.

**Fluxo Básico**

1. O sistema recebe um pedido de login de usuário.
2. O sistema solicita ao usuário o seu nome de usuário e a senha.
3. O usuário fornece o nome de usuário e a senha.
4. O sistema valida os dados de login.
5. O sistema registra o usuário como logado no sistema.
6. O sistema exibe a tela inicial do sistema.
7. O caso de uso termina.

**Fluxos Alternativos**

**Fluxo Alternativo 1 – Alternativa ao passo 3**

3a.1. O usuário seleciona a opção de lembrete de nome de usuário ou senha.
3a.2. O sistema solicita e-mail cadastrado no sistema.
3a.3. O usuário fornece e-mail.
3a.4. O sistema valida a regra de negócio RN5.
3a.5. O sistema envia e-mail com o nome de usuário e opção para redefinir a senha.
3a.6. O caso de uso termina.

**Fluxo Alternativo 2 – Alternativa ao passo 4**

4a.1. O sistema verifica que a regra de negócio RN1 não foi satisfeita.
4a.2. O sistema emite o alerta “Nome de usuário ou senha são obrigatórios.”
4a.3. O caso de uso termina.

**Fluxo Alternativo 3 – Alternativa ao passo 4**

4b.1. O sistema verifica que a regra de negócio RN2 não foi satisfeita.
4b.2. O sistema emite o alerta “Nome de usuário ou senha incorretos.”
4b.3. O caso de uso termina.

**Fluxo Alternativo 4 – Alternativa ao passo 4**

4c.1. O sistema verifica que a regra de negócio RN3 não foi satisfeita.
4c.2. O sistema contabiliza uma tentativa de login para o usuário.
4c.3. O sistema verifica que a regra de negócio RN6 não foi satisfeita.
4c.4. O sistema emite o alerta “Nome de usuário ou senha incorretos.”
4c.5. O caso de uso termina.

**Fluxo Alternativo 5 – Alternativa ao passo 4**

4d.1. O sistema verifica que a regra de negócio RN4 não foi satisfeita.
4d.2. O sistema emite o alerta “Usuário bloqueado: contate a nossa equipe de suporte.”
4d.3. O caso de uso termina.

**Regras de Negócio**

**RN1.** Um nome de usuário e uma senha foram fornecidos.

**RN2.** O nome de usuário fornecido corresponde a um nome de usuário cadastrado no sistema.

**RN3.** O hash da senha fornecida corresponde ao hash da senha cadastrada no sistema.

**RN6.** O usuário não se encontra bloqueado.

**Requisitos Não-funcionais**

**Hashing de Senha:** o algoritmo de hashing da senha deve ser SHA-3.

**Tempo de resposta:** o tempo de resposta deve ser inferior a 3 segundos.
Let’s talk about requirements

<table>
<thead>
<tr>
<th>ID: UC_0001</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Nome:</strong> Fazer login</td>
</tr>
<tr>
<td><strong>Descrição:</strong> Usuário solicita fazer login no sistema, fornecendo usuário e senha, que são validados. Caso sejam válidos, usuário é logado no sistema. Caso contrário, o acesso é bloqueado. Mecanismos de segurança são adotados para impedir o acesso de usuários maliciosos.</td>
</tr>
<tr>
<td><strong>Pré-condições:</strong> nenhum usuário está logado no sistema.</td>
</tr>
<tr>
<td><strong>Pós-condições:</strong> usuário está registrado como logado no sistema.</td>
</tr>
<tr>
<td><strong>Fluxo Básico</strong></td>
</tr>
<tr>
<td>1. Este caso inicia quando o usuário seleciona a opção para fazer login no sistema.</td>
</tr>
<tr>
<td>2. O sistema solicita ao usuário o seu nome de usuário e a senha.</td>
</tr>
<tr>
<td>3. O usuário fornece os dados solicitados.</td>
</tr>
<tr>
<td>4. O sistema valida as regras de negócio RN1, RN2, RN3 e RN6.</td>
</tr>
<tr>
<td>5. O sistema registra o usuário como usuário logado no sistema.</td>
</tr>
<tr>
<td>6. O sistema exibe a tela inicial do sistema.</td>
</tr>
<tr>
<td>7. O caso de uso termina.</td>
</tr>
<tr>
<td><strong>Fluxos Alternativos</strong></td>
</tr>
<tr>
<td><strong>Fluxo Alternativo 1 – Alternativa ao passo 3</strong></td>
</tr>
<tr>
<td>3a.1. O usuário seleciona a opção de lembrete de nome de usuário ou senha.</td>
</tr>
<tr>
<td>3a.2. O sistema solicita e-mail cadastrado no sistema.</td>
</tr>
<tr>
<td>3a.3. O usuário fornece e-mail.</td>
</tr>
<tr>
<td>3a.4. O sistema valida a regra de negócio RN5.</td>
</tr>
<tr>
<td>3a.5. O sistema envia e-mail com o nome do usuário e opção para redefinir a senha.</td>
</tr>
<tr>
<td>3a.6. Retorna ao passo 2.</td>
</tr>
</tbody>
</table>
Let's talk about requirements

28/05/2021

Ingrid Nunes <ingridnunes@inf.ufrgs.br>

Departamento de Informática Aplicada
Tel.: 3308 3333 / Fax: 3308 7308
Email: dep.ina@inf.ufrgs.br
WEB: http://www.inf.ufrgs.br/ina

Login – Descrição do Caso de Uso

ID: UC_000

Nome: Fazer login

Descrição: Usuário solicita fazer login no sistema, fornecendo usuário e senha, que são validados.
Caso sejam válidos, usuário é logado no sistema. Caso contrário, o acesso é bloqueado.

Mecanismos de segurança são adotados para impedir o acesso de usuários maliciosos.

Pré-condições: nenhum usuário está logado no sistema.
Pós-condições: usuário está registrado como logado no sistema.

Fluxo Básico

1. Este caso inicia quando o usuário seleciona a opção para fazer login no sistema.
2. O sistema solicita ao usuário o seu nome de usuário e a senha.
3. O usuário fornece os dados solicitados.
4. O sistema valida as regras de negócio RN1, RN2, RN3 e RN6.
5. O sistema registra o usuário como usuário logado no sistema.
6. O sistema exibe a tela inicial do sistema.
7. O caso de uso termina.

Fluxos Alternativos

Fluxo Alternativo 1 – Alternativa ao passo 3
3a.1. O usuário seleciona a opção de lembrete de nome de usuário ou senha.
3a.2. O sistema solicita e-mail cadastrado no sistema.
3a.3. O usuário fornece e-mail.
3a.4. O sistema valida a regra de negócio RN5.
3a.5. O sistema envia e-mail com o nome do usuário e opção para redefinir a senha.
3a.6. Retorna ao passo 2.

Fluxo Alternativo 2 – Alternativa ao passo 3a.4.
3b.1. O sistema verifica que a regra de negócio RN5 não foi satisfeita.
3b.2. O sistema emite o alerta “O e-mail fornecido não é um endereço de e-mail válido.”
3b.3. Retorna ao passo 3a.2.

Fluxo Alternativo 3 – Alternativa ao passo 4
4a.1. O sistema verifica que a regra de negócio RN1 não foi satisfeita.
4a.2. O sistema emite o alerta “Nome de usuário e senha são obrigatórios.”
4a.3. Retorna ao passo 2.

Fluxo Alternativo 4 – Alternativa ao passo 4
4b.1. O sistema verifica que a regra de negócio RN2 não foi satisfeita.
4b.2. O sistema emite o alerta “Nome de usuário ou senha incorretos.”
4b.3. Retorna ao passo 2.

Fluxo Alternativo 5 – Alternativa ao passo 4
4c.1. O sistema verifica que a regra de negócio RN3 não foi satisfeita.
4c.2. O sistema contabiliza uma tentativa de login para o usuário com nome de usuário fornecido.
4c.3. O sistema valida a regra de negócio RN4.
4c.4. O sistema emite o alerta “Nome de usuário ou senha incorretos.”
4c.5. Retorna ao passo 2.

Fluxo Alternativo 6 – Alternativa ao passo 4c.3.
4d.1. O sistema verifica que a regra de negócio RN4 não foi satisfeita.
4d.2. O sistema bloqueia o usuário.
4d.3. O sistema emite o alerta “Usuário bloqueado: contate a nossa equipe de suporte.”

Regras de Negócio

RN1. Um nome de usuário e uma senha foram fornecidos.
RN2. O nome de usuário fornecido corresponde a um nome de usuário cadastrado no sistema.
RN3. O hash da senha fornecida corresponde ao hash da senha cadastrada no sistema associada ao nome de usuário fornecido.
RN4. Podem ser feitas no máximo 3 tentativas de login de um usuário com senha incorreta.
RN5. O string fornecido é composto por mais de um caractere e possui @ no meio do string, e termina com .com
RN6. O usuário não se encontra bloqueado.

Requisitos Não-funcionais

Hashing da Senha: o algoritmo de hashing da senha deve ser SHA-3.
Tempo de resposta: o tempo de resposta de retorno quando as regras de negócio RN2 ou RN3 não forem válidas deve ser o mesmo.
In agile methods, all this information must be detailed in acceptance criteria and test cases!
But... wasn’t the talk about code quality?
Importance of (well-understood) requirements

• Prevent unpredicted challenges while designing and coding

• Enables a proper design of the solution
  • Rather than implementing bit by bit

• Prevent reworking the code

• Crucial for designing test cases

Software without well-specified requirements cannot be right or wrong, it can only surprise you.
Key Points

- Have an architectural model
- Do not neglect modularisation and separation of concerns
- Adopt code review (and check the points above!)
- Manage technical debt (and make payments!)
- Use metrics and static analysis tools (automation!)
- Software logging with guidelines and standards
Software Architecture

"The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them."

- Bass et al. 2003

• Modules
  • Including design-time, test-time, and run-time hardware and software parts

• Externally visible properties
  • Modules with interfaces, hardware units, objects

• Relationships and constraints
  • Dependencies

Consistency between the conceptual and implemented architecture!
Software Architecture

- Fundamental for an organised software evolution

Diagram:
- View
- Business
- Data
- Business Rule
- Duplicated code!
Software Architecture

• Key design decisions
  • Adopted technologies
    • spring Framework
    • python + django
  • Non-functional requirements

Scalability

7/9/21

ingridnunes@inf.ufrgs.br
Monoliths are fine if you are committed to them.

(Micro)services are fine if you are committed to them.

Microliths are what happens when an organization isn’t brave enough to pick a lane. The worst of both worlds without the advantages of any of them.
Software Architecture

Raiz

Nutella

Legend

Coarse-grained rule

Fine-grained rule

Implementation detail
Key Points

Have an architectural model

Do not neglect modularisation and separation of concerns
Modularity and SoC

High cohesion + Low coupling = Good modularity
Modularity and SoC

Is there any problem in this code?

class UserController {
    createUser() {
        // check data valid
        new User()
    }

    resetPassword() {
        // check token valid
        // check pw valid
        user.setPassword(newPW)
    }
}

class User {
    login
    password
    //getters
    //setters
}
Modularity and SoC

class UserController {

   createUser() {
      // check data valid
      new User()
   }

   resetPassword() {
      // check token valid
      // check pw valid
      user.setPassword(newPW)
   }
}

class User {

   login
   password

   // getters
   // setters
}

Duplicated code!
High coupling!
Modularity and SoC

class UserController {
    createUser () {
        checkUser()
        new User()
    }

    resetPassword () {
        // check token valid
        checkUser()
    }
}

class User {
    login
    password
    //getters
    //setters
}

class UserValidator {
    checkUser () {
        ...
    }
}
Modularity and SoC

• Effective use of object orientation

```java
class UserController {
    createUser() {
        new User()
    }
    resetPassword() {
        user.resetPassword(newPW)
    }
}
```

```java
class User {
    login
    password

    User () {
        ...
        setPassword()
    }
    setPassword() {
        //check pw valid
    }
    resetPassword() {
        isTokenValid()
        setPassword()
    }
    isTokenValid() { … }
}
```
Modularity and SoC

• Rules and Principles
  • Bertrand Meyer. 1988. Object-Oriented Software Construction.

  • Modularity Rules
    • Direct Mapping
    • Few Interfaces
    • Small interfaces (weak coupling)
    • Explicit Interfaces
    • Information Hiding

  • Modularity Principles
    • Linguistic Modular Units
    • Self-Documentation
    • Uniform Access
    • Open-Closed
    • Single Choice

• Principles S.O.L.I.D.

  • Single Responsibility Principles
  • Open-Closed Principle
  • Liskov Substitution Principle
  • Interface Segregation Principle
  • Dependency Inversion Principle
Modularity and SoC

- GRASP Patterns
  - E.g. Specialist

- Code Smells / Refactoring Catalog
  - E.g.
    - Feature Envy
    - Data Class
Key Points

Have an architectural model

Do not neglect modularisation and separation of concerns

Adopt code review (and check the points above!)
Code Review

- Effective verification technique
  - Code quality
  - Knowledge sharing
  - Code ownership

Code Review

• Code readability and understandability ≠ good modularity

“To do this, we use 444 human evaluations from 63 developers and we obtained a bold negative result: none of the 121 experimented metrics is able to capture code understandability, not even the ones assumed to assess quality attributes apparently related, such as code readability and complexity.”

Key Points

Have an architectural model

Do not neglect modularisation and separation of concerns

Adopt code review (and check the points above!)

Manage technical debt (and make payments!)
Technical Debt

WE FIND THE BUG
WE FIX THE BUG
NOW WE HAVE TWO BUGS
NOW WE HAVE THREE BUGS

© www.SoftwareTestingHelp.com
Technical Debt

- Metaphor created by Ward Cunningham to justify for non-technical stakeholders the need for refactoring.
- Some problems in the code are like financial debt. It is ok to make a loan, as long as it is paid.

**Technical debt management is crucial! It must be paid.**
Key Points

- Have an architectural model
- Do not neglect modularisation and separation of concerns
- Adopt code review (and check the points above!)
- Manage technical debt (and make payments!)
- Use metrics and static analysis tools (automation!)
• **Code Metrics**
  • Traditional: LOC, Fan-in, Fan-out, Cyclomatic Complexity, ...
  • CK Metrics: DIT, WMC, RFC, CBO, LCOM, NOC
Metrics and Static Analysis Tools

- **Code Metrics**
  - Traditional: LOC, Fan-in, Fan-out, Cyclomatic Complexity, ...
  - CK Metrics: DIT, WMC, RFC, CBO, LCOM, NOC

- **Static Analysis Tools**
  - **Automatically checking** of common problems
  - Use of **rules**, e.g. dependencies, code smell detections
  - Identification of **anomalies**
  - Part of **automated reviewers**
Metrics and Static Analysis Tools

“You can’t manage what you can’t measure”,
Tom DeMarco

• Experimental Software Engineering
• GQM
  • Framework for systematic measurement, data collection, and analysis
  • GOAL
    • Measurement objects can be products, processes and resources
  • QUESTION
    • Characterisation of the questions aligned with the objectives
  • METRIC
    • Measurements to answer the specified questions

Key Points

- Have an architectural model
- Do not neglect modularisation and separation of concerns
- Adopt code review (and check the points above!)
- Manage technical debt (and make payments!)
- Use metrics and static analysis tools (automation!)
- Software logging (with guidelines and standards!)
Software Logging and Monitoring

• What is logging?
  • It is the practice of recording relevant information about a running system

```java
log.debug("writing file to: {}", file);
```

• Be precise, concise and consistent in logging statements
• Specify (in advance) and follow logging conventions
Logging Best Practices: The 13 You Should Know

1. Don’t Write Logs by Yourself (AKA Don’t Reinvent the Wheel)
2. Log at the Proper Level
3. Employ the Proper Log Category
4. Write Meaningful Log Messages
5. Write Log Messages in English
6. Add Context to Your Log Messages
7. Log in Machine Parseable Format
8. But Make the Logs Human-Readable as Well
9. Don’t Log Too Much or Too Little
10. Think of Your Audience
11. Don’t Log for Troubleshooting Purposes Only
12. Avoid Vendor Lock-In
13. Don’t Log Sensitive Information

Final Considerations

• Most of the content of this talk are covered in undergraduate courses
  • Problem: theory vs. practice

• To give importance to code maintainability and legibility
  • Cost reduction (less bugs, easier evolution)
  • Happy programmers 😊😊😊

• How to convince management and those responsible for business decisions?
Software Development Effort Estimation

```c
string title = "Botão para Cancelar";
string sender = "Mateus";
```
Mental Health of Professionals of 
Software Development

Do you think that working on software development is stressful? Do you want to know how jobs related to SOFTWARE DEVELOPMENT affect a person's MOOD and MENTAL HEALTH?

We would like to find this out but, for that, we need the help of people from the software industry. If you are one, happy or not with your job, please complete our survey! We also would REALLY appreciate it if you share our invitation with your peers so we can reach a high number of participants :)

Link: https://forms.gle/NBTVuc5svbGfEBfz6
(estimated time: 20 min)
Podcasts

Emílias - Armação em Bits
@Emilias_UTFPR

Saíu o novo episódio do Emílias Podcast em que entrevistamos @ingridnunesIN, nova co-host do @FronteirasES:
open.spotify.com/episode/4gCd8c...
#MulheresPodcasters
#MulheresNaComputacao

Se gostar, dá RT!
cc @rla4

Translate Tweet
Fronteiras da Engenharia de Software
@FronteirasES

Episódio #8 no ar!
Arquitetura de Software com Ingrid Nunes (UFRGS)

Translate Tweet

#8. Arquitetura de Software com Ingrid Nunes (UFRGS) - Fronteiras da Engenharia de Software
🔗 open.spotify.com
Key Points

- Have an architectural model
- Do not neglect modularisation and separation of concerns
- Adopt code review (and check the points above!)
- Manage technical debt (and make payments!)
- Use metrics and static analysis tools (automation!)
- Software logging (with guidelines and standards!)

Prof. Ingrid Nunes
UFRGS

Homepage
ingridnunes.github.io

Twitter
twitter.com/ingridnunesIN

Facebook
facebook.com/ingridnunesIN

LinkedIn
linkedin.com/in/in/ingrid-nunes