Technical Excellence in Software Development

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

Série de Seminários do INF + CEI-Talks
Software Development

Task A  Task B  Task C


Practices
- Practice X
- Practice Y
- Practice Z

04/04/2021  Ingrid Nunes <ingridnunes@inf.ufrgs.br>
Software Development

• But…

Ah 2018.

Scrum means "Waterfall but we don't have time for analysis".

Kanban means "Scrum, but we don't have time for sprint planning".

Agile means "We have no process, but we do use extensively"
Software Development

• And…

@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.

Software Development

• What does not work?

1. To adopt an approach without knowing it or adapt it, when it is required

2. To adopt an approach ignoring what is key for it to work

3. To consider something as does with no guarantees that it has been done as it should be
class UserController {
    createUser() {
        // check data valid
        new User()
    }
    resetPassword() {
        // check token valid
        // check pw valid
        user.setPassword(newPW)
    }
}

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

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
}

Duplicated code!
High coupling!
class UserController {
    createUser() {
        checkUser()
        new User()
    }

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

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

class UserValidator {
    checkUser() {
    }
}
Principles and Best Practices

• Effective use of object orientation

```javascript
class UserController {
    createUser() {
        new User()
    }

    resetPassword() {
        user.resetPassword(newPW)
    }
}
```

```javascript
class User {
    login
    password

    User () {
        ...  
        setPassword()
    }

    setPassword() {
        //check pw valid
    }

    resetPassword() {
        user.resetPassword(newPW)
        isTokenValid()
    }

    isTokenValid() {
        ... 
    }

    setPassword()
}
```
Principles and Best Practices

• 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
Principles and Best Practices

• GRASP Patterns
  • E.g. Specialist

• Code Smells / Refactoring Catalog
  • E.g.
    • Feature Envy
    • Data Class
High cohesion + Low coupling = Good modularity
A well-defined segmentation of the project effort ensures system modularity. Each task forms a separate, distinct program module. At implementation time each module and its inputs and outputs are well-defined, there is no confusion in the intended interface with other system modules. At checkout time the integrity of the module is tested independently; there are few scheduling problems in synchronizing the completion of several tasks before checkout can begin. Finally, the system is maintained in modular fashion; system errors and deficiencies can be traced to specific system modules, thus limiting the scope of detailed error searching.

Gouthier and Pont [1970]
Software Architecture

• Fundamental for an organised software evolution

Name

1. Modules
2. Dependencies
3. Module roles

Consistency between the conceptual and implemented architecture!
Software Architecture

- Fundamental for an organised software evolution

View

Business

Data

Business Rule

Duplicated code!
Software Architecture

• Architecture Recovery
Software Architecture

- WGB Method
- Dependencies
- MDS Metric
- Optimisation Problem

• Advantage
  • Avoid problem 3: To consider something as does with no guarantees that it has been done as it should be
Code Review

• Some facts (most derived from OSS)
  • Average of 1-2 reviewers and 2-3 comments per request
  • Newcomers tend to receive more attention
  • Small code changes and long descriptions facilitate the review
  • Most frequent discussion topics
    • Code improvement
    • Understanding
    • Social interactions

• MCR and pair programming are interchangeable in terms of cost
  • When pair programming is adopted within test-driven development, MCR has lower cost
  • Unit testing: finds more failures
  • MCR: less time in the detection and isolation of the underlying sources of the defects

• Most common type of support
  • Reviewer recommenders and visualizations of code changes

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.
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

Reasons for TD introduction at the code level

• From the self perspective
  • Tight schedule
  • Work overload
  • Pressure from the management

• From the perspective of other developers
  • Also development and technology inexperience

Practices that should be adopted to avoid TD

• 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
    • Dependencies, code smell detections
  • 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

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

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

• Be precise, concise and consistent in logging statements
• Specify (in advance) and follow logging conventions
### Software Logging and Monitoring

<table>
<thead>
<tr>
<th>Goal Group</th>
<th>Efficiency</th>
<th>Maintainability</th>
<th>Reliability</th>
<th>Security</th>
<th>Testability</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Goal Examples</strong></td>
<td>performance, energy saving, caching, improving resource consumption</td>
<td>bug finding, understanding, reuse, documentation, troubleshooting</td>
<td>health checking, fault tolerance, disaster recovery, adaptation, configuration fix</td>
<td>anomaly detection, data protection, malicious attack detection</td>
<td>testing (generation, validation, selection), reporting, verification</td>
</tr>
<tr>
<td>Frequency</td>
<td>Number of occurrences in a period</td>
<td>Number of references and dependencies</td>
<td>Inter-arrival times</td>
<td>Changes in occurrence history</td>
<td>Number of occurrences in test case</td>
</tr>
<tr>
<td>Maintainability</td>
<td>Number of operations involved</td>
<td>Static source code metrics</td>
<td>—</td>
<td>Contextual information of objects/classes</td>
<td>Fail test coverage</td>
</tr>
<tr>
<td>Expensiveness</td>
<td>Execution time</td>
<td>Source code locations of expensive methods</td>
<td>CPU and heap utilization, processing times</td>
<td>Transaction duration</td>
<td>Depth of call stack</td>
</tr>
<tr>
<td>Changeability</td>
<td>Number of repeated computations</td>
<td>Similarity between call graphs</td>
<td>Number of operations with cached results</td>
<td>Changes in contextual information</td>
<td>—</td>
</tr>
<tr>
<td>Error-proneness</td>
<td>Number of failures of a component</td>
<td>Number of handled exceptions</td>
<td>Number of failures perceived by users</td>
<td>Increase of failures in a specific component</td>
<td>Number of failure assertions or exception-throwing statements</td>
</tr>
<tr>
<td>Usage pattern</td>
<td>Changes in user navigational activity</td>
<td>—</td>
<td>Number of active users and idle/active intervals</td>
<td>Variations in the request payload for same operations</td>
<td>—</td>
</tr>
<tr>
<td>State variation</td>
<td>I/O consumption per operation</td>
<td>Changes in the system state</td>
<td>Number of write operations performed</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>Concurrency</td>
<td>Number of active users and threads</td>
<td>Number of references and dependencies</td>
<td>Number of race conditions</td>
<td>—</td>
<td>Number of locks per test case</td>
</tr>
<tr>
<td>Latency</td>
<td>Processing and bandwidth consumption</td>
<td>Throughput</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

04/04/2021

Ingrid Nunes <ingridnunes@inf.ufrgs.br>
Software Logging and Monitoring

• Tigris: a two-phase framework for software tracing

Summary

• Make an effective use of object orientation
• Have an architectural model
  • Software organisation and rules
• Adopt code review
• Manage technical debt
  • Make payments
• Use metrics and static analysis tools
• Collect runtime data
  • Consistent use of software logging
  • Know what data to collect with low performance impact
Challenges

• Quality
  • Conformance with requirements
    • Organisational requirements
    • Project requirements
      • Functional
      • Non-functional

Performance Bugs

Security Bugs
Challenges

- Performance Bugs
  - Finding and fixing performance issues
    - Search for equal or similar objects
    - Overuse of temporary structures
    - Containers used too little or too much
    - Data unnecessarily copied
    - Etc.
  - A software engineer’s responsibility!

Cloud computing: high costs
The end of Moore’s Law

Challenges

• Performance Bugs
  • Use of application-level caching

```java
public class C1() {
    public Object process() {
        // creating the cache component
        Cache cache = Cache.getInstance();

        // looking up for cache content (steps in b)
        Object content = cache.get("c1:c2-computation");
        if (content == null) {
            // cache miss (steps in c)
            content = C2.compute();

            // caching the content for future requests
            cache.set("c1:c2-computation", content);
        }

        // doing some business logic...
        return content;
    }
}
```

Podcast: Fronteiras em Engenharia de Software: https://anchor.fm/fronteirases Episode #8

Challenges

- Performance Bugs
- Use of application-level caching

```java
public class C1() {
    public Object process() {
        // creating the cache component
        Cache cache = Cache.getInstance();
        // looking up for cache content (steps in b)
        Object content = cache.get("c1:c2-computation");
        if (content == null) {
            // cache miss (steps in c)
            content = C2.compute();
            // caching the content for future requests
            cache.set("c1:c2-computation", content);
        }
        // doing some business logic...
        return content;
    }
}
```

Challenges

• Security Bugs

<table>
<thead>
<tr>
<th>ID: UC_0001</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nome: 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>
<tr>
<td><strong>Fluxo Alternativo 2 – Alternativa ao passo 3a.4.</strong></td>
</tr>
<tr>
<td>3b.1. O sistema verifica que a regra de negócio RNS não foi satisfeita.</td>
</tr>
<tr>
<td>3b.2. O sistema emite o alerta “O e-mail fornecido não é um endereço de e-mail válido.”</td>
</tr>
<tr>
<td>3b.3. Retorna ao passo 3a.2.</td>
</tr>
</tbody>
</table>
### Challenges

**Security Bugs**

- Ingrid Nunes <ingridnunes@inf.ufrgs.br>

---

### 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.”

4d.4. Retorna ao passo 2.

### Fluxo Alternativo 7 – Alternativa ao passo 4.

4e.1. O sistema verifica que a regra de negócio RN6 não foi satisfeita.

4e.2. O sistema emite o alerta “Usuário bloqueado: contate a nossa equipe de suporte.”

4e.3. Retorna ao passo 2.

---

### Regras de Negócio

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

**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.
Challenges

From the ICSE presentation by @margaretstorey
Challenges

Web Developer in 2021

Choose your path

- Front-end
- Back-end
- DevOps

Required for any path

- Git - Version Control
- Basic Terminal Usage
- Data Structures & Algorithms
- GitHub
- Licenses
- Semantic Versioning
- SSH
- HTTP/HTTPS and APIs
- Design Patterns
- Character Encodings

Find the detailed version of this roadmap along with resources and other roadmaps

http://roadmap.sh

https://github.com/kamranahmedse/developer-roadmap
Challenges

• Heterogeneous world of technologies and programming languages
  • OO and functional languages
  • Compiled and interpreted languages
  • Strongly and dynamically typed languages
  • General purpose and domain-specific languages

Our central finding is that both static type systems find an important percentage of public bugs: both Flow 0.30 and TypeScript 2.0 successfully detect 15%!

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.
Final Considerations

• Most of the content of this talk are covered in undergraduate courses
  • Problem: theory vs. practice
  • Role of internship (at least, in Brazil)?

Programmers vs. Software Engineers

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

“being a developer is not stressing at all”
John - 26 yrs old
Final Considerations

- Most of the content of this talk are covered in undergraduate courses
  - Problem: theory vs. practice
  - Role of internship (at least, in Brazil)?

Programmers vs. Software Engineers

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

Despite all advances, why does go horse still exist?
Final Considerations

• How to help us?

• Access to projects and developers for research
  • Project mining (with access to the source code or issue trackers)
  • Surveys with developers
  • It is possible to sign NDAs (Non-Disclosure Agreement)
    • Publications may or may not include acknowledgements
    • Anonymised data (after the company’s approval)
Thanks!

• Prof. Ingrid Nunes (UFRGS)
  • Homepage http://inf.ufrgs.br/~ingridnunes/
  • Twitter https://twitter.com/ingridnunesIN
  • Facebook https://www.facebook.com/ingridnunesIN
  • LinkedIn https://www.linkedin.com/in/ingrid-nunes/

Why does go horse still exist?