Software Quality and Measurement Lecture 25

Review to 2nd Exam

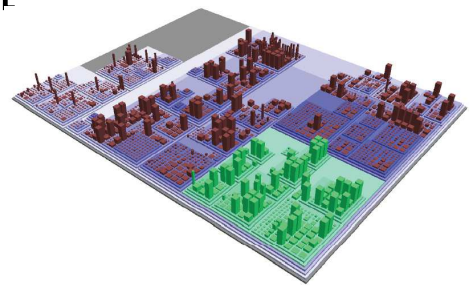
Eduardo Figueiredo
http://www.dcc.ufmg.br/~figueiredo
eseufmg@gmail.com

1 June 2015

14: Software Visualization

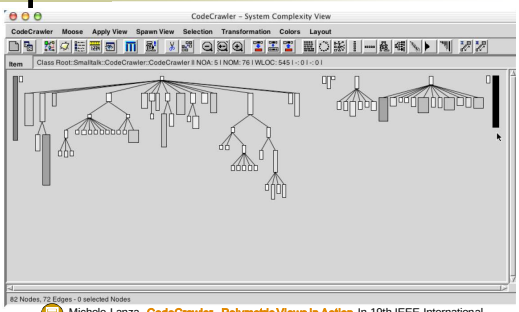
- Evaluating the design with metrics is a hard task
  - Metrics are too fine grained and generate a lot of data to analyze
- Visualization has long been used to break down the complexity of information

System as a City



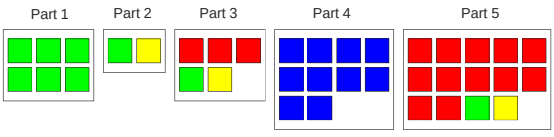
R. Wettel and M. Lanza. **Visualizing Software Systems as Cities.** In 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis, pp. 92 - 99, 2007

Polymetric View



Michele Lanza. **CodeCrawler - Polymetric Views In Action.** In 19th IEEE International Conference on Automated Software Engineering (ASE), pp. 394 - 395, 2004.

Distribution Map



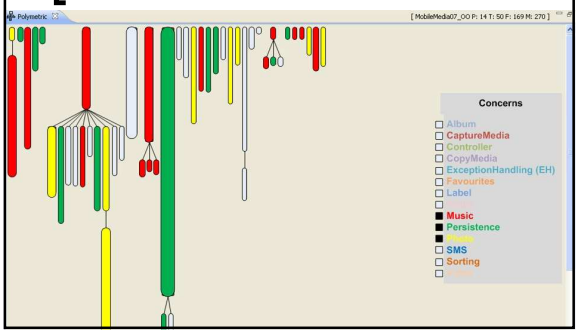
- Five parts with 6, 2, 5, 10, and 14 elements, respectively
- Four properties: Red, Blue, Green, and Yellow

View Crosscutting Concerns

- Crosscutting “shapes”
  - Black Sheep and Octopus
- Inheritance relationships
  - Climbing Plant and Hereditary Disease
- Concern-based coupling connections
  - Tree Root and Tsunami
- Structure of crosscutting code
  - Copy Cat and Behavioural Concern

E. Figueiredo, B. Silva, C. Sant'Anna, A. Garcia, J. Whittle, and D. Nunes. **Crosscutting Patterns and Design Stability: An Exploratory Analysis.** In proceedings of the 17th International Conference on Program Comprehension (ICPC), 2009.

Inheritance-wise Structure



Bibliography

- M. Lanza, R. Marinescu. **Object-Oriented Metrics in Practice**. Springer, 2006. (Section 3.2)
- S. Ducasse et al. **Distribution Map**. International Conference on Software Maintenance (ICSM), 2006.
- E. Figueiredo et al. Crosscutting Patterns and Design Stability: An Exploratory Analysis. International Conference on Program Comprehension (ICPC), 2009.
- G. Carneiro, et al. Identifying Code Smells with Multiple Concern Views. Brazilian Symposium on Software Engineering (SBES), pp. 128-137, 2010.

15: Paper Presentation

- A Validation of Object-Oriented Design Metrics as Quality Indicators
- Detection Strategies: Metrics-Based Rules for Detecting Design Flaws

16: Paper Presentation

- On the Effectiveness of Concern Metrics to Detect Code Smells: An Empirical Study

17: Design Patterns

- A design pattern is a general reusable solution to a common problem
- Patterns are known best practices
  - They allow reuse of knowledge from experts
- They do not describe a complete solution, since it is supposed to be reused in different applications

Structural Patterns

- **Adapter**
- Bridge
- **Composite**
- **Decorator**
- Facade
- Flyweight
- Proxy

Behavioral Patterns

- **Chain of Responsibility (CoR)**
- Command
- Interpreter
- Iterator
- **Mediator**
- Memento
- Observer
- **State**
- Strategy
- Template Method
- Visitor

Creational Patterns

- Abstract Factory
- Builder
- **Factory Method**
- Prototype
- **Singleton**

Bibliography

- E. Gamma, R. Helm, R. Johnson, J. Vlissides. **Design Patterns**, 1st. Edition. Addison Wesley, 1994.
  - Chapter 1
  - Adapter, Composite and Decorator
  - Chain of Responsibility, Mediator and State
  - Factory Method and Singleton

18: Programming Idioms

- Idioms are low-level patterns specific to a programming language
- An idiom describes how to solve a implementation-specific problem in a programming language
- Idioms ease communication among developers
  - They also speed up development and maintenance tasks

Code Clone

- Code Clone, also called Duplicated Code, is a well known code smell in software systems
  - Code clones are often the result of copy & paste practices
- Studies show that about 5% to 20% of a software code is duplicated

Code Clone Types I and II

- Type I: Identical code fragments except for variations in whitespace, layout, and comments
- Type II: Structurally identical fragments except for variations in identifiers, literals, types, layout, and comments

Code Clone Types III and IV

- Type III: Copied fragments with further modifications
  - Statements can be changed, added, or removed in addition to variations in identifiers, literals, types, and comments
- Type IV: Fragments that perform the same computation, but implemented through different syntactic variants

Bibliography

- F. Buschmann et al. **Pattern-Oriented Software Architecture: A System of Patterns**. John Wiley & Sons, 1996. (Chapter 4)
- A. von Staa. **Programação Modular**. Elsevier, 2000. (Appendix 3, 4 e 5)
- M. Balazinska et al. "Measuring Clone Based Reengineering Opportunities". International Software Metrics Symposium (METRICS), 1999.
- R. Koschke, R. Falke, P. Frenzel. "Clone Detection Using Abstract Syntax Suffix Trees". Working Conference on Reverse Engineering (WCRE), 2006.
- J. Krinke. "Identifying Similar Code with Program Dependence Graphs". Working Conference on Reverse Engineering (WCRE), pp. 301-309, 2001.

19: Software Architecture

- Software architecture is the high level structure of a software system
- It links the problem space (requirements) to the solution space (design)
  - Architecture can be considered the first step to design the software solution
  - Architecture design is a creative activity

Architectural Patterns

- An architectural pattern is a general, reusable solution to a recurring problem in software architecture
- It documents knowledge about a common problem
- It is supposed to be reused across applications

From Mud to Structure

- Layered Architecture
  - It helps to structure applications that can be decomposed into group of tasks
- Pipes and Filters
  - It provides a structure for systems that process a stream of data
- Blackboard
  - It is useful for problems for which no deterministic solution is known

Architectural Patterns

- Distributed Systems
  - Client-Server
  - **Broker**
- Interactive Systems
  - Model-View-Controller (MVC)
  - **Presentation-Abstraction-Control**
- Adaptable Systems
  - **Microkernel**
  - Reflection

Bibliography

- F. Buschmann et al. **Pattern-Oriented Software Architecture: A System of Patterns**. John Wiley & Sons, 1996.
  - Chap. 2 Architectural Patterns
- Ian Sommerville. **Software Engineering**, 9th Edition. Pearson Education, 2011.
  - Chapter 6 Architectural Design

20: Processo de Software

- Conjunto de atividades que leva ao desenvolvimento do produto software
- Um processo define
  - Quem faz, o que faz e quando fazer
  - Nem sempre diz como fazer
- Não existe um processo ideal
  - Organizações desenvolvem seus próprios processos

Modelos de Processo Gerais

- Modelo Cascata
- Desenvolvimento Incremental
- Baseado em Reuso

Processos Ágeis

- As regras de negócios mudam rapidamente
  - O software tem que ser adaptado para as novas regras
- Desenvolvimento e entrega rápida são importantes em mercados competitivos
  - A entrega rápida pode ser tão (ou mais) desejável que a qualidade

Manifesto Ágil

- Passamos a valorizar:
  - **Indivíduos e interações** mais que processos e ferramentas
  - **Software em funcionamento** mais que documentação abrangente
  - **Colaboração com o cliente** mais que negociação de contratos
  - **Responder a mudanças** mais que seguir um plano

http://agilemanifesto.org/

Programação Extrema (XP)

- Proposta a partir de boas práticas de desenvolvimento incremental
- Propõe o envolvimento do cliente ao extremo
  - O cliente (ou seu representante) deve estar disponível durante todo o desenvolvimento
- Programadores trabalham em pares

Bibliografia

- Ian Sommerville. **Engenharia de Software**, 9ª Edição. Pearson Education, 2011.
  - Cap. 2
  - Cap. 3
- Manifesto Ágil.
  - http://agilemanifesto.org/

21: Code Clone Exercise

Method 1

```
void printOwing(double amount) {
    printBanner();

    //print details
    System.out.println ("name:" + _name);
    System.out.println ("amount" + amount);
}
```

Method 2

```
void showBalance() {
    System.out.println ("name:" + customer);
    System.out.println ("amount" + balance);
    System.out.println ("Date: "+ new Date());
}
```

Analysis Result:

Extracted Method

```
void printDetails (double amount) {
    System.out.println ("name:" + _name);
    System.out.println ("amount" + amount);
}
```

Method 1 after refactoring

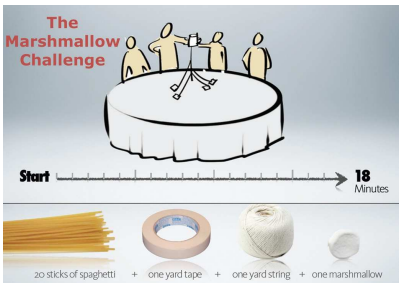
```
void printOwing(double amount) {
    printBanner();
    printDetails(_name, amount);
}
```

Method 2 after refactoring

```
void showBalance() {
    printDetails(customer, balance);
    System.out.println ("Date: "+ new Date());
}
```

Is code clone? Explain why.

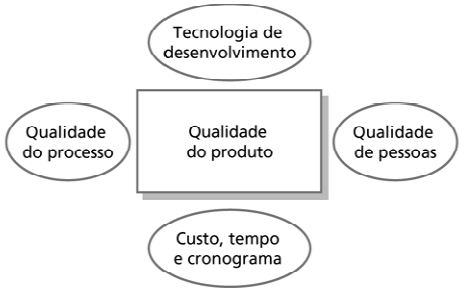
22: The Marshmallow Challenge



Fonte: www.marshmallowchallenge.com

33

Qualidade do Produto



34

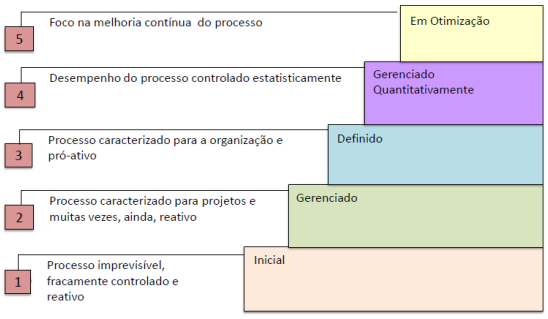
23: Melhoria de Processo

- Após uma avaliação da indústria de software, foi criado o Modelo de Maturidade da Capacitação (CMM)
  - Depois disso, vários modelos semelhantes e compatíveis com o CMM foram definidos
  - Um deles: CMM para Software
    - 1986: Início dos Trabalhos
    - 1991: Publicação da Primeira Versão

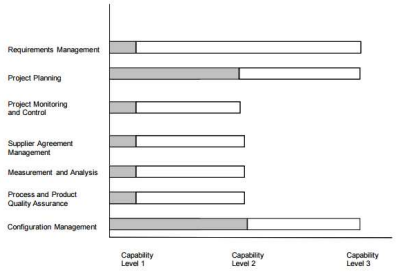
Representações do CMMI

- O CMMI tem duas representações
  - CMMI por Estágios
  - CMMI Contínuo
- CMMI por Estágios permite avaliação da maturidade do processo em 5 níveis
- CMMI Contínuo permite uma classificação mais fina
  - Classifica cada área de processo

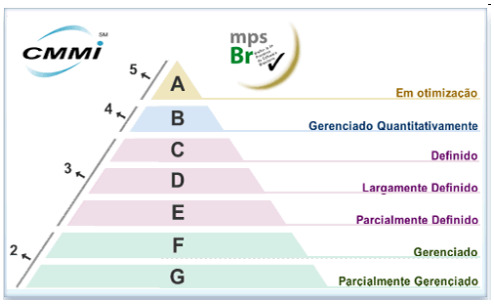
CMMI por Estágios



CMMI Contínuo



MPS.Br vs. CMMI



Níveis de Maturidade MPS.Br

- Nível A - Em Otimização
- Nível B - Gerenciado Quantitativamente
- Nível C - Definido
- Nível D - Largamente Definido
- Nível E - Parcialmente Definido
- Nível F - Gerenciado
- Nível G - Parcialmente Gerenciado

Bibliografia

- Ian Sommerville. **Engenharia de Software**, 9ª Edição. Pearson Education, 2011. (Capítulo 26)
- A. Koscianski e M. Soares. **Qualidade de Software**, 2ª Edição. Novatec, 2006. (Capítulo 7)
- CMMI for Development 1.3

24: Paper Presentation

- Are Students Representatives of Professionals in Software Engineering Experiments?
- Evaluating Pair Programming with Respect to System Complexity and Programmer Expertise

3rd Exam (optional)

- The 3rd exam is allowed to students with at least 40 points (undergrads)
  - It replaces 1st or 2nd exam
  - 20 pts (Graduate) or 40 pts (Undergrads)
- Which lectures to study?
  - If it replaces 1st exam: Lectures 1 to 13
  - If it replaces 2nd exam: Lectures 14 to 25

O que fazer?

- Se deseja fazer prova substitutiva
  - Você deve mandar um email solicitando-a até 08/06
  - Você deve dizer qual prova deseja substituir (Prova 1 ou Prova 2)
  - Você deve vir a aula do dia 08/06 (revisão para 3ª prova)