Engenharia de Software II

Reengenharia de Software

Prof. André Hora
DCC/UFMG
2019.1

DCC
DEPARTAMENTO DE
CIÊNCIA DA COMPUTAÇÃO

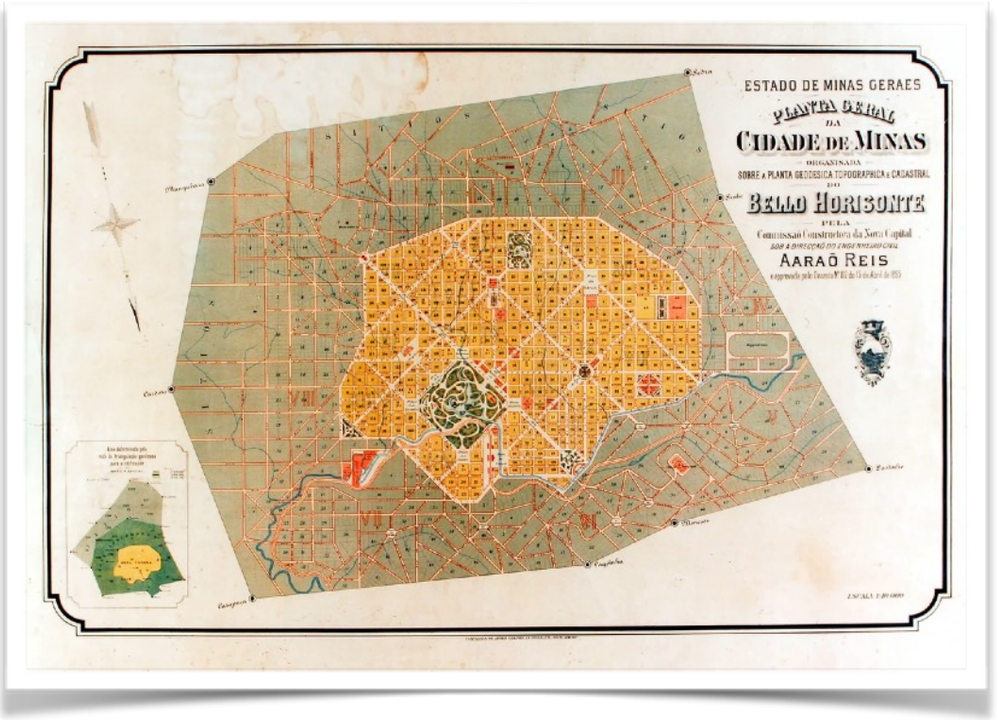
UFMG

Agenda

1. **Evolução de software**
2. Reengenharia de software
3. Processo de reengenharia
4. Ferramentas

Exemplo: **Em Teoria**

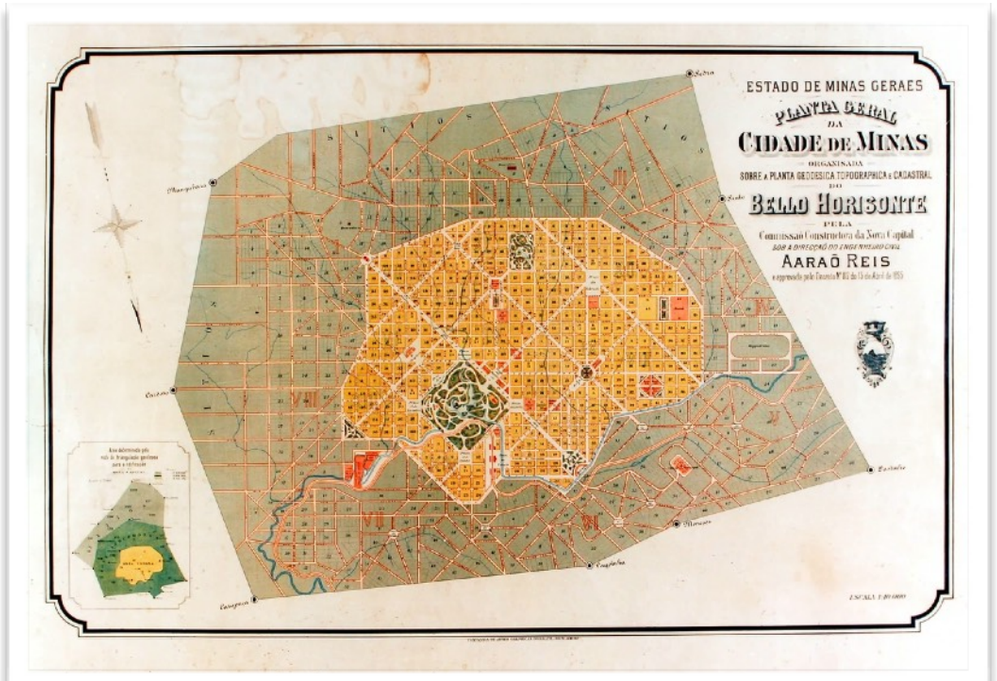
1900



População: 10K

Exemplo: **Realidade**

1900

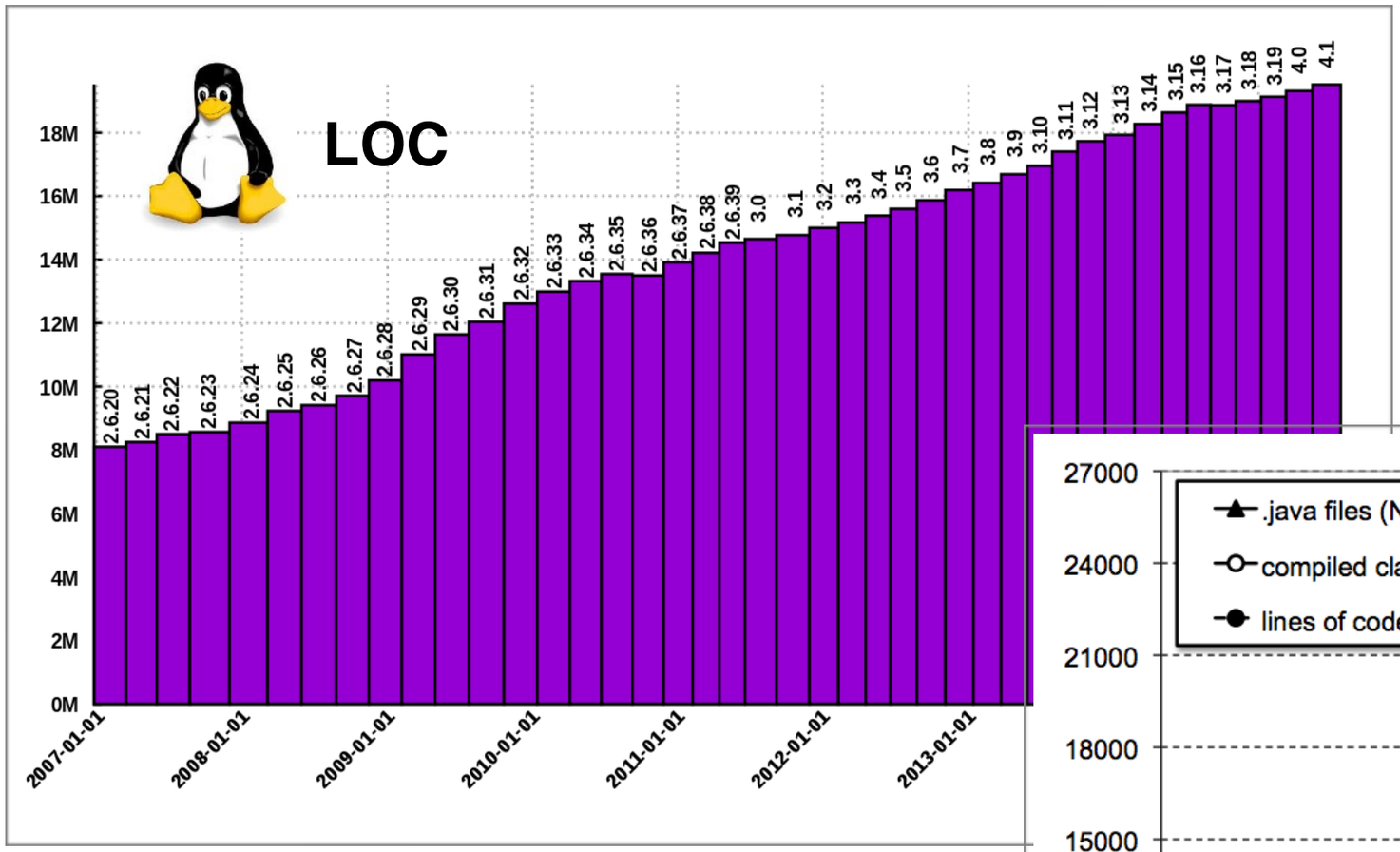


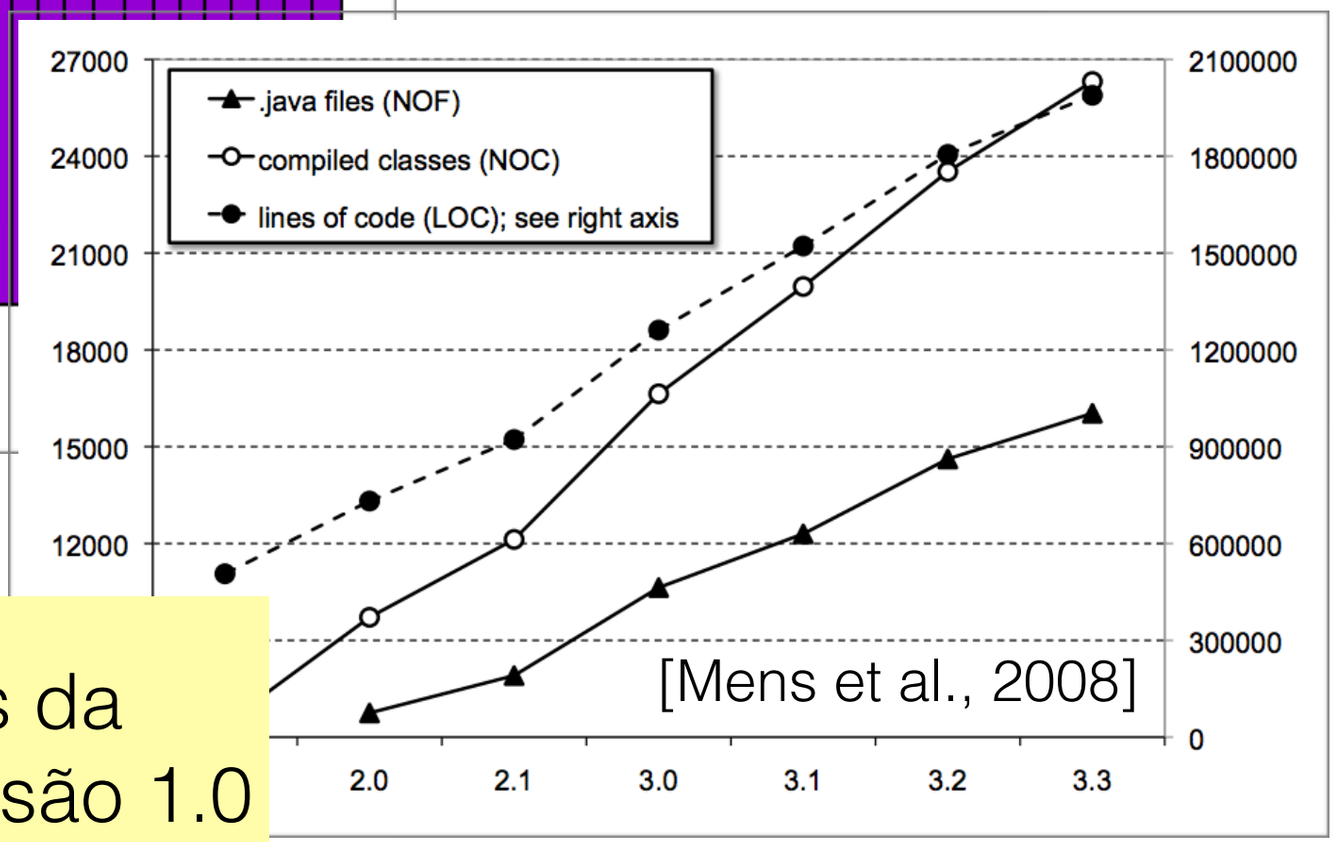
População: 10K

2017

População: 2.5M (5.8M)

Evolução de Software





No Eclipse, 58% dos métodos da versão 2.0 não existiam na versão 1.0

Leis de Lehman

Mudança contínua: software deve ser continuamente adaptado, senão torna-se menos satisfatório

Complexidade crescente: se não forem tomadas medidas para reduzir complexidade, ela irá aumentar progressivamente

Crescimento contínuo: software deve ter funcionalidades ampliadas para manter a satisfação dos seus usuários

Declínio da qualidade: software se deprecia se eles não receberem as mudanças necessárias

Manutenção de Software

- Processo de alterar o sistema após sua entrega
  - Correção de defeitos
  - Refatoração
  - Adição de funcionalidades
  - Mudanças no ambiente externo

Refatoração

- Refatoração: mudança realizada nas estruturas internas do programa para facilitar seu entendimento e baratear suas alterações sem modificar seu comportamento
- Refatoração vs otimização?

Refatoração

- Refatoração: mudança realizada nas estruturas internas do programa para facilitar seu entendimento e baratear suas alterações sem modificar seu comportamento

- Refatoração vs otimização?

- Otimização: assim como refatoração, otimização não altera comportamento, apenas altera estruturas internas

  - Otimização torna o código mais difícil de manter

Tipos de Manutenção

- Manutenção Corretiva
- Manutenção Preventiva
- Manutenção Evolutiva
- Manutenção Adaptativa

Tipos de Manutenção

- Manutenção Corretiva
- Manutenção Preventiva
- Manutenção Evolutiva
- Manutenção Adaptativa

Algumas vezes manutenção se torna **difícil**, **arriscada** e **custosa**

Reengenharia de Software

Agenda

1. Evolução de software
2. **Reengenharia de software**
3. Processo de reengenharia
4. Ferramentas

Reengenharia de Software

- **Reconstruir** sistema
- Melhorar desempenho, manutenabilidade, segurança, documentação, entre outros
- Melhorar estruturas e entendimento
- Sistemas legados

Software Legado

+ Valioso ($)

– Linguagem de programação antiga (ex: COBOL, PHP)

– Tecnologia obsoleta

– Mudou muito, mostra sinais de adaptações

– Não pode ser descartado nem atualizado (custoso)

Benefícios da Reengenharia

?

Benefícios da Reengenharia

- Reduzir a **complexidade** de sistemas legados
- Exemplos:
  - Portar para nova plataforma
  - Melhorar performance
  - Extrair modelos
  - Explorar novas tecnologias
  - Reduzir dependência humana

Benefícios da Reengenharia

- Reduzir **riscos:**
  - Risco de desenvolver o sistema novamente é alto
  - Podem ocorrer erros de especificação ou problemas no desenvolvimento
- Reduzir **custos:**
  - Custo de reengenharia pode ser muito menor que o custo de desenvolvimento ($12M x $50M)
  - Ferramentas podem automatizar uma parte

Quando aplicar reengenharia?

- Documentação obsoleta ou não documentação
- Desenvolvedores deixaram empresa
- Conhecimento limitado do sistema
- Muito tempo gasto para realizar pequenas mudanças
- Correção de defeitos frequente
- Problemas de manutenção (efeito cascata)

Engenharia x Reengenharia

?

Engenharia x Reengenharia

Engenharia Reversa x Reengenharia

Chikofsky e Cross definem:

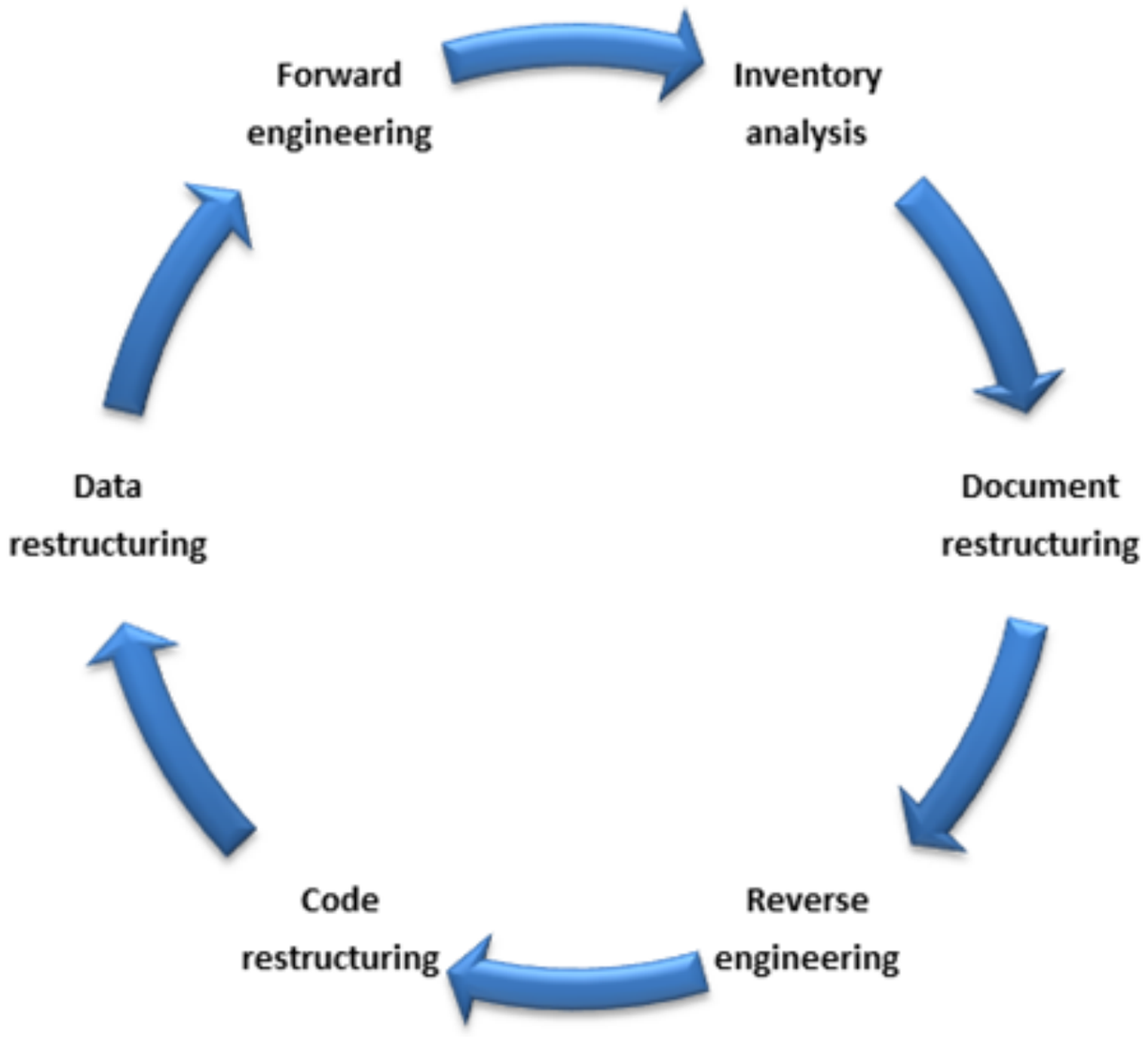
Reverse Engineering is the process of analysing a system to identify the components and their interrelationships, and create representations at a higher level of abstraction. (**entender**)

Reengineering is the examination and alteration of a system to reconstitute it in a new form and the subsequent implementation of the new form. (**reestruturar**)

Agenda

1. Evolução de software
2. Reengenharia de software
3. **Processo de reengenharia**
4. Ferramentas

Processo de Reengenharia



- Etapas bem definidas
- Cíclico

1. Análise de Inventário

- Toda organização tem um **inventário** de aplicações
- Organizado numa planilha (tamanho, idade, importância)
- Priorizar sistemas candidatos
- Alocar recursos

2. Reestruturação de Documentação

- Documentação **fraca** e **escassa** é a marca de sistemas legados
- Opções:
  - Não fazer nada
  - Documentar apenas mudanças
  - Documentar o mínimo necessário

3. Engenharia Reversa (1)

- Processo de **extrair informação** a partir do código
- Exemplo: extrair diagramas de classes
- Informação extraída pode ser provida para:
  - Engenheiro de software
  - Ferramenta

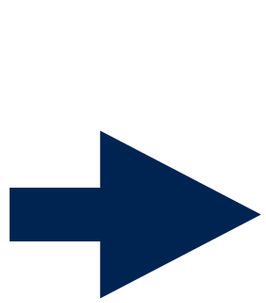
3. Engenharia Reversa (2)

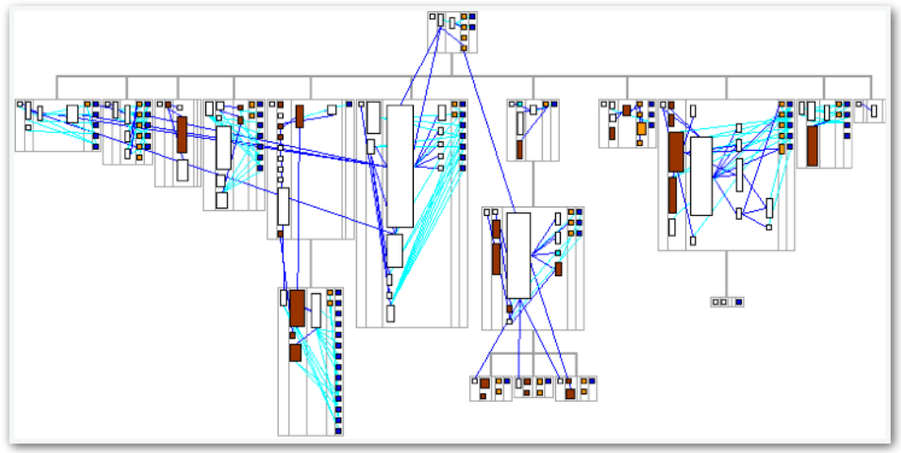
- Entender:
  - Dados: interno x global
  - Processamento: abstrações procedurais (com ferramentas)
  - Interface com o usuário: estrutura e comportamento da UI

```
class AllTermWeight extends PayloadTermWeight {

    AllTermWeight(AllTermQuery query, IndexSearcher searcher, boolean needsScores) throws IOException {
        super(query, searcher, needsScores);
    }

    @Override
    public AllTermSpanScorer scorer(LeafReaderContext context, Bits acceptDocs) throws IOException {
        if (this.stats == null) {
            return null;
        }
        // we have a custom weight class, we must check in case something is wrong with _all
        Terms terms = context.reader().terms(query.getField());
        if (terms != null && terms.hasPositions() == false) {
            throw new IllegalStateException("field \"" + term.field() + "\" was indexed without position
        }
        TermSpans spans = (TermSpans) query.getSpans(context, acceptDocs, termContexts);
        if (spans == null) {
            return null;
        }
```





4 e 5. Reestruturação de Código e Dados

- Modifica código ou dados
- Não altera arquitetura
- Código: mesma funcionalidades, mais qualidade
- Dados: projeto de dados mais efetivo (mudar BD)

6. Engenharia Avante

- Recupera o projeto de software
- Melhora a qualidade do sistema
- Adiciona novas funcionalidades
- Modifica arquitetura
- Resultado: nova configuração do sistema

Custo-benefício

- Determinado quantitativamente
- Manter estado atual:
  - Custo de suporte + manutenção ($$$)
- Aplicar reengenharia:
  - Custo da reengenharia + suporte + manutenção ($)
- **Compensa quando**: sistema tem vida longa e pobre manutenabilidade

Agenda

1. Evolução & Manutenção de software
2. Reengenharia de software
3. Processo de reengenharia
4. **Ferramentas**

Ferramentas

- Analisadores de código:
  - Análise estática
  - Análise dinâmica
- Entrada: código, traces de execução, etc

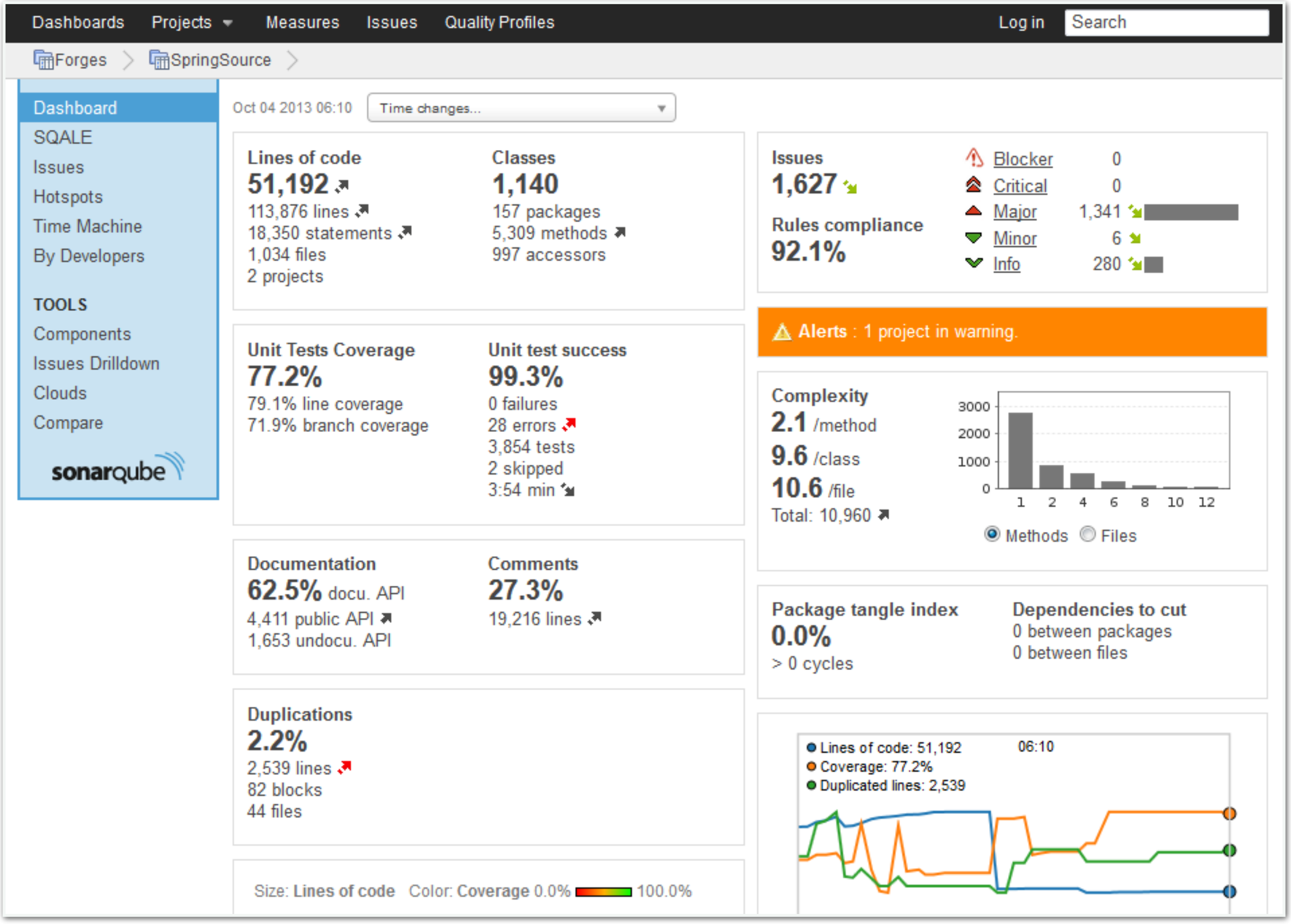
Análise Estática

- Extrai informação do **código**
- Diagramas: Classe, Pacote, Dependências, etc
- Vantagem: rápida, barata e não demanda execução
- Desvantagens:
  - Ordem dos eventos é perdida (por onde começo a ler o diagrama?)
  - Relacionamentos dinâmicos não são capturados (ex: polimorfismo, reflexão, etc)

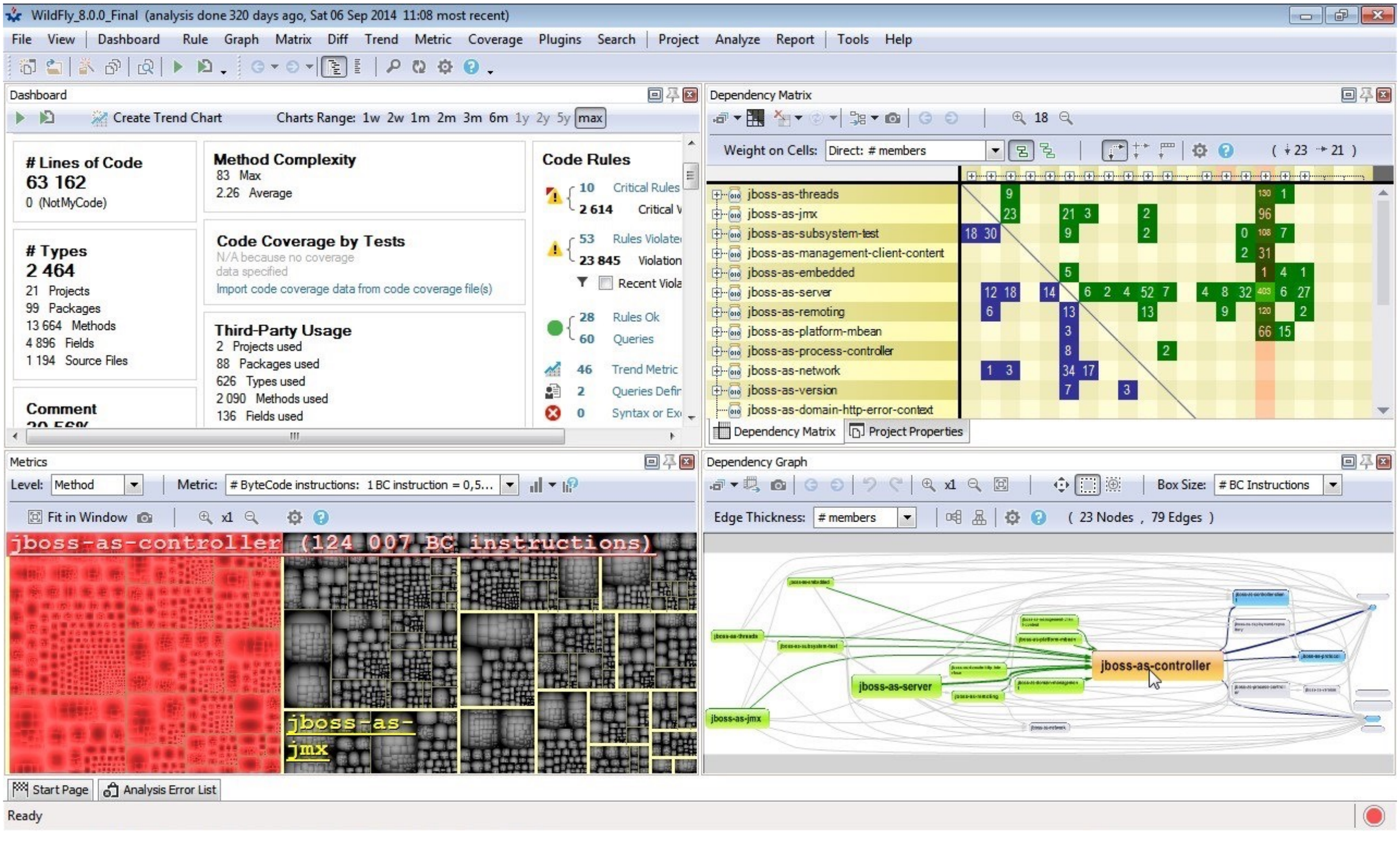
Análise Dinâmica

- Extrai informação da **execução**
- Diagramas: Objeto, Colaboração, Sequência, etc
- Vantagem: mais fácil de acompanhar ordem eventos
- Desvantagem:
  - Demanda execução do sistema
  - Incompleta
  - Escalabilidade

SonarQube

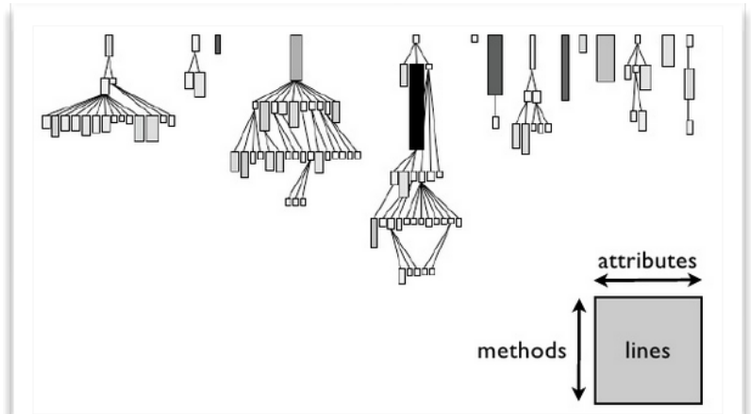

Dashboards
Projects
Measures
Issues
Quality Profiles
Log in
Search
Forges
SpringSource
Dashboard
SQALE
Issues
Hotspots
Time Machine
By Developers
TOOLS
Components
Issues Drilldown
Clouds
Compare
sonarqube
Oct 04 2013 06:10
Time changes...
Lines of code
51,192
113,876 lines
18,350 statements
1,034 files
2 projects
Classes
1,140
157 packages
5,309 methods
997 accessors
Issues
1,627
Rules compliance
92.1%
Blocker 0
Critical 0
Major 1,341
Minor 6
Info 280
Alerts : 1 project in warning.
Unit Tests Coverage
77.2%
79.1% line coverage
71.9% branch coverage
Unit test success
99.3%
0 failures
28 errors
3,854 tests
2 skipped
3:54 min
Complexity
2.1 /method
9.6 /class
10.6 /file
Total: 10,960
3000
2000
1000
0
1 2 4 6 8 10 12
Methods
Files
Documentation
62.5% docu. API
4,411 public API
1,653 undocu. API
Comments
27.3%
19,216 lines
Package tangle index
0.0%
> 0 cycles
Dependencies to cut
0 between packages
0 between files
Duplications
2.2%
2,539 lines
82 blocks
44 files
Lines of code: 51,192
Coverage: 77.2%
Duplicated lines: 2,539
06:10
Size: Lines of code
Color: Coverage 0.0%
100.0%

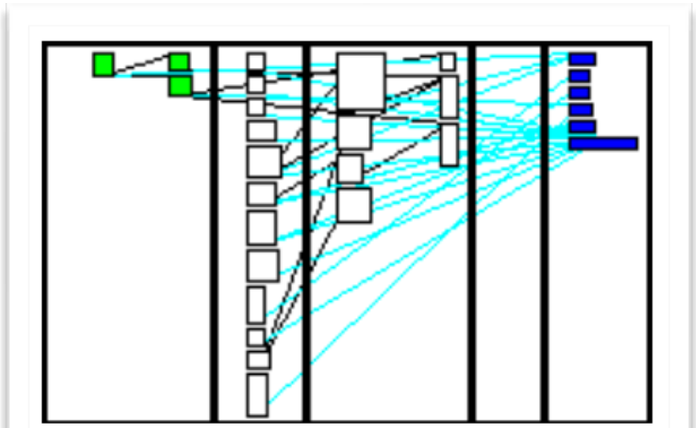
JArchitect

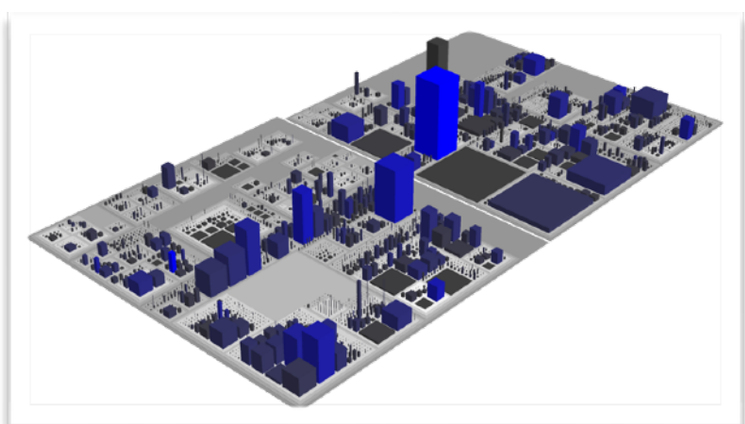

WildFly_8.0.0_Final (analysis done 320 days ago, Sat 06 Sep 2014 11:08 most recent)
File View Dashboard Rule Graph Matrix Diff Trend Metric Coverage Plugins Search Project Analyze Report Tools Help
Dashboard
Create Trend Chart
Charts Range: 1w 2w 1m 2m 3m 6m 1y 2y 5y max
Lines of Code
63 162
0 (Not My Code)
Types
2 464
21 Projects
99 Packages
13 664 Methods
4 896 Fields
1 194 Source Files
Comment
Method Complexity
83 Max
2.26 Average
Code Coverage by Tests
N/A because no coverage data specified
Import code coverage data from code coverage file(s)
Third-Party Usage
2 Projects used
88 Packages used
626 Types used
2 090 Methods used
136 Fields used
Code Rules
10 Critical Rules
2 614 Critical
53 Rules Violate
23 845 Violation
Recent Viola
28 Rules Ok
60 Queries
46 Trend Metric
2 Queries Defir
0 Syntax or Ex
Dependency Matrix
Weight on Cells: Direct: # members
jboss-as-threads
jboss-as-jmx
jboss-as-subsystem-test
jboss-as-management-client-content
jboss-as-embedded
jboss-as-server
jboss-as-remoting
jboss-as-platform-mbean
jboss-as-process-controller
jboss-as-network
jboss-as-version
jboss-as-domain-http-error-context
Dependency Matrix
Project Properties
Metrics
Level: Method
Metric: # ByteCode instructions: 1 BC instruction = 0,5...
Fit in Window
jboss-as-controller (124 007 BC instructions)
jboss-as-jmx
Dependency Graph
Edge Thickness: # members
Box Size: # BC Instructions
(23 Nodes , 79 Edges)
jboss-as-controller
jboss-as-server
jboss-as-jmx
Start Page
Analysis Error List
Ready

Ferramenta de reengenharia: Moose

- Independente de linguagem
- Extensível
- Análise estática
- Análise histórica
- Meta-modelos
- Visualizações
- Métricas







[eg, Ducasse e Lanza, 2005; Ducasse et al., 2006; Wettel e Lanza 2008, 2009; Hora et al., 2012]