Análise Estática e Métricas
Ênfase em Qualidade de Produto

Auri Marcelo Rizzo Vincenzi[1], Márcio Eduardo Delamaro[2] e José Carlos Maldonado[2]

[1]Instituto de Informática
Universidade Federal de Goiás

[2]Instituto de Ciências Matemáticas e de Computação
Universidade de São Paulo

Organização

Introdução

Introdução (1)

- Qualidade de Software é uma área de grande importância na Engenharia de Software, e intimamente ligada à etapa de Verificação e Validação (Anderson, 2008).
- Atividades de análise estática e dinâmica podem ser empregadas conjuntamente para maximizar a detecção de defeitos em produtos de software.

Análise Estática (ISO/IEC/IEEE, 2010)

"Processo de avaliar um sistema ou componente com base em sua forma, estrutura, conteúdo ou documentação".

Análise Dinâmica (ISO/IEC/IEEE, 2010)

"Processo de avaliar um sistema ou componente com base em seu comportamento durante a execução".

Análise Estática (1)

- É realizada sem a execução do programa;
- Utilizada na avaliação de artefatos como documento de requisitos, diagramas, código fonte e outros.
- Dentre as principais atividades de análise estática estão (ISO/IEC/IEEE, 2010):
  - Inspeção;
  - Revisão Informal;
  - Leitura Baseada em Perspectiva (Basili et al., 1996)
  - *walk-through*;
  - *desk checking*.
- Em geral, também não é possível provar a correção do artefato por meio de técnicas de análise estática.

Análise Estática (2)

- Principais Vantagens:
  - Pode ser utilizada durante todas as fases de desenvolvimento.
  - Fornece para os desenvolvedores informações sobre artefatos ambíguos, incompletos, ou sem finalidade.
- Principais Desvantagens:
  - Em geral, não é adequada para a detecção de defeitos comportamentais.
  - Exige muitos recursos (humanos) e esforço para ser conduzida com qualidade.
    - 2000 LOC requer em torno de 10 reuniões de inspeção para revisar completamente o código
  - Para Inspeção, necessário um bom *checklist* para permitir uma boa inspeção do artefato.

Análise Estática (3)

- A atividade de Análise Estática pode ser manual ou automática.
- Principalmente aquela realizada em código ou artefatos executáveis são, em geral, realizadas por ferramentas automatizadas.

Análise Estática Automatizada (1)

- Analisadores Estáticos (*bug finding tools*)
  - Vasculham o código fonte e detectam possíveis defeitos;
  - Constroem uma representação abstrata do comportamento do programa analisado e examina seus dados;
  - Por utilizar aproximações com a análise, defeitos que não existem podem ser relatados, devido à imprecisão da abstração.
- A emissão de avisos (*warnings*) indicando defeitos que não existem é considerada a principal desvantagem da análise estática automatizada.

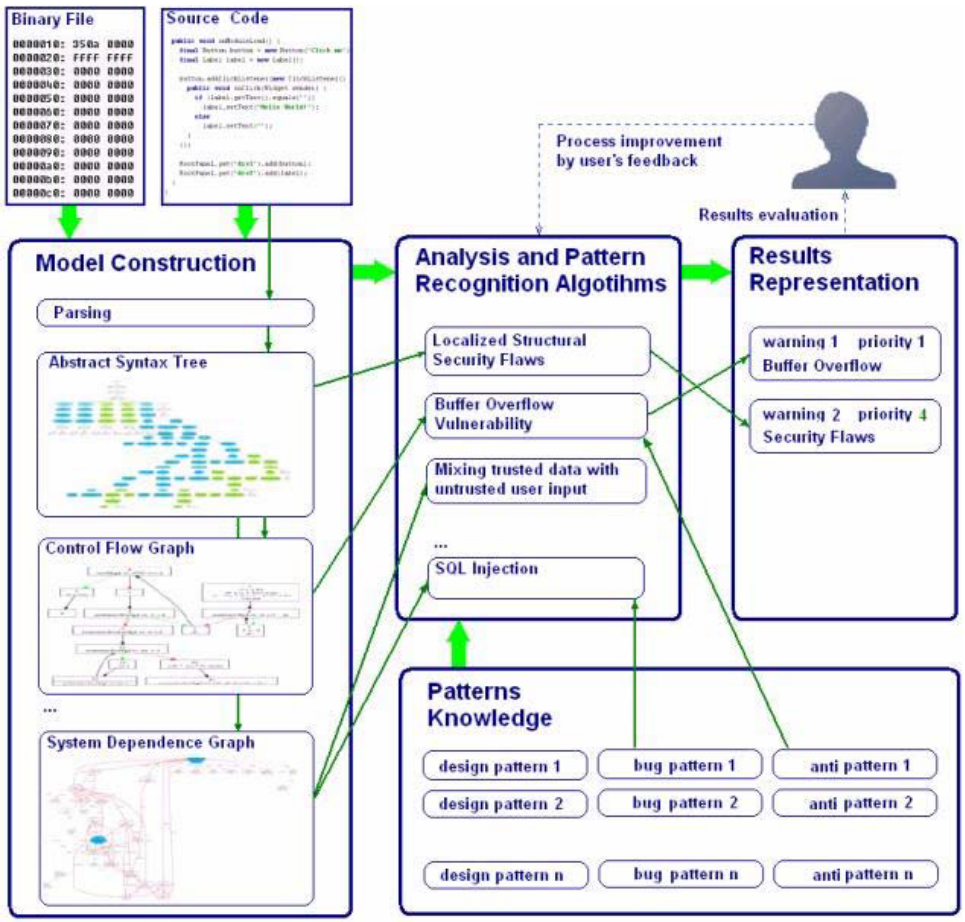
Análise Estática Automatizada (2)

- Um aspecto que diferencia os analisadores estáticos de outras técnicas estáticas é a presença da classificação dos avisos de acordo com níveis de prioridade/severidade;
- Inexiste uma classificação padrão, deste modo cada ferramenta define sua forma de classificação dos possíveis defeitos identificados;
- Em teoria, quanto maior o nível de prioridade do aviso, maior a chance de se tratar de um defeito real;
- Entretanto, mesmo dentre os avisos de alta prioridade, existem os chamados falsos positivos.

Análise Estática Automatizada (3)

Tipo Aviso	**Significado**
Falso Positivo	Avisos que indicam defeitos inexistentes no código fonte.
Falso Negativo	Nenhum aviso é reportado, mas existem defeitos no código fonte.
Verdadeiro Positivo	Avisos que realmente correspondem a defeitos no código fonte.

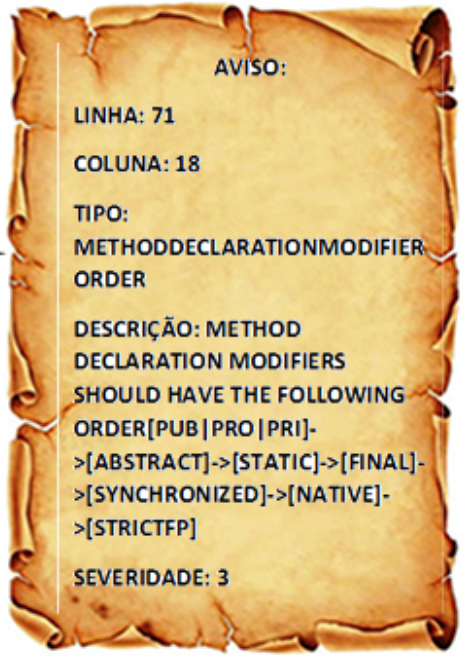
Análise Estática Automatizada (4)



Estrutura geral de uma ferramenta para análise estática de código (Kirkov e Gagre, 2010)

Análise Estática Automatizada (5)

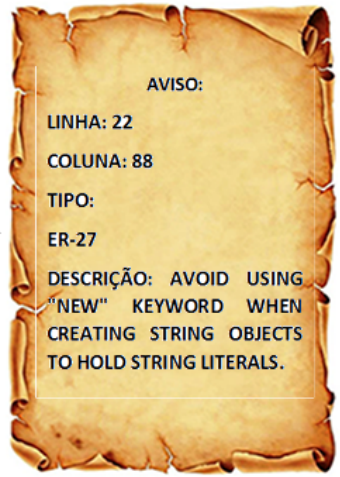
```
65    // other codes through 15 reserved for system use
66    /** reserved for local use */
67    final static public int LOG_LOCAL0 = 16<<3;
68    /** reserved for local use */
69    final static public int LOG_LOCAL1 = 17<<3;
70    /** reserved for local use */
71    final static public int LOG_LOCAL2 = 18<<3;
72    /** reserved for local use */
73    final static public int LOG_LOCAL3 = 19<<3;
74    /** reserved for local use */
75    final static public int LOG_LOCAL4 = 20<<3;
76    /** reserved for local use */
77    final static public int LOG_LOCAL5 = 21<<3;
78    /** reserved for local use */
79    final static public int LOG_LOCAL6 = 22<<3;
80    /** reserved for local use*/
81    final static public int LOG LOCAL7 = 23<<3;
```



Exemplo de aviso emitido pela ferramenta JCSC.

Análise Estática Automatizada (6)

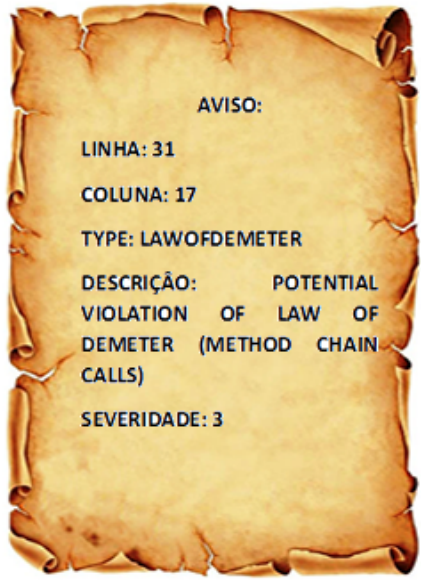
```
16  public static void main(String[] args) {
17
18      try {
19          ManageAccount.num = 5;
20          ManageAccount[] bank=new ManageAccount[ManageAccount.num];
21          String[] accountName= {new String("A"),new String("B"),new String("C"),new String("D"),new String("E"),
22                  new String("F"),new String("G"),new String("H"),new String("I"),new String("J"),
23          };
24          for (int j=0; j<ManageAccount.num; j++) {
25              bank[j]=new ManageAccount(accountName[j],100);
26              ManageAccount.accounts[j].print();;//print it
27          }
28
```



Exemplo de aviso emitido pela ferramenta Hammurapi.

Análise Estática Automatizada (7)

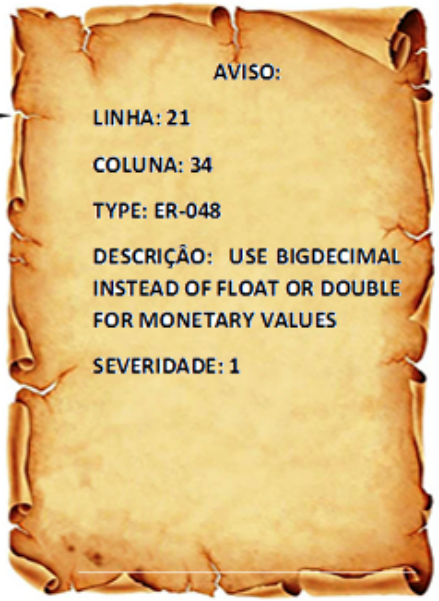
```
23
24      // Create accouns with initial balance of 100
25      Bank bank = new Bank(numBusinessAccounts, numPersonalAccounts, 100);
26      bank.work();
27      bank.printAllAccounts();
28
29      // Check to see that all of the balances are stable
30      for (int i=0; i < numBusinessAccounts+numPersonalAccounts; i++) {
31          if (bank.getAccount(i).getBalance() != 300) {
32              throw new RuntimeException("bug found");
33          }
```



Exemplo de aviso emitido pela ferramenta PMD.

Análise Estática Automatizada (8)

```
21    synchronized  void transfer(Account ac,double mn) {
22        amount-=mn;
23        //System.out.println("ac.amount is $"+ac.amount);
24        if (name.equals("D")) {
25            System.out.println("unprotected");
26            ac.amount+=mn;//no aquire for the other lock!!
27            //+= might cause problem --it is not atomic.
28        } else {
29            //System.out.println("protected");
30            synchronized (ac) {
31                ac.amount+=mn;
32            }
33        }
34    }
```



Exemplo de aviso emitido pela ferramenta Hammurapi.

Análise Estática Automatizada (9)

- Analisadores Estáticos são, em geral, classificados como:
  - Verificadores de defeitos: orientados à detecção automática de defeitos baseada em padrões, erros esses comumente realizados pelos programadores e que não são necessariamente detectados pelo compilador.
  - Verificadores de estilo: direcionados à detecção de itens em desacordo com o estilo adotado de codificação.

Verificadores de Defeitos

- Exemplos de defeitos identificados:
  - Conexão com Banco de Dados não finalizada.
  - Perda de Referência (*null deference*).
  - Uma exceção não esperada, portanto não tratada pelos desenvolvedores.
  - A atribuição de valor a uma variável local, que não é utilizada em nenhum trecho subsequente.
  - A atribuição de `null` a uma variável (*null assignment*).
  - Uma possível tentativa de alocar *array* de tamanho negativo.

Verificadores de Estilo

- Exemplos de problemas de estilo identificados:
  - Bloco `catch` vazio.
  - Acrescentar o modificador final a uma variável. A utilização generosa do modificador final é uma boa prática de programação, permitindo que o compilador e a máquina virtual realizem pequenas otimizações.
  - Nome de uma variável não está de acordo com o padrão "^[a-z][a-zA-Z0-9]*$".
  - A linha do código muito longa.
  - A falta de comentário do JavaDoc.

Métricas de Software (1)

Métrica de Produto (ISO/IEC/IEEE, 2010)

"Métrica usada para medir uma característica de qualquer produto intermediário ou final resultante do processo de desenvolvimento de software".

Métrica de Processo (ISO/IEC/IEEE, 2010)

"Métrica usada para medir características de métodos, técnicas e ferramentas empregadas no desenvolvimento, implementação e manutenção de sistemas de software".

Terminologia e Definições (1)

Medição (*Measurement* (ISO/IEC/IEEE, 2010)

1. "Ato ou processo de atribuir um número ou categoria a uma entidade que descreve um atributo dessa entidade".
2. "Uso de uma métrica para atribuir um valor (o qual pode ser um número ou categoria) a partir de uma escala para um atributo de uma entidade".

Medida (*Measure*) (ISO/IEC/IEEE, 2010)

1. "Variável na qual um valor é atribuído como resultado de uma medição".
2. "Fazer uma medição".
3. "Aplicar uma métrica".

Métrica (*Metric*) (ISO/IEC/IEEE, 2010)

1. "Método e escala de medição".
2. "Medida quantitativa do grau em que um sistema, um componente, ou processo possui um determinado atributo".

Benefícios das Métricas (1)

- Alguns dos benefícios do uso de métricas são (Craig e Jaskiel, 2002):
  - Identificar áreas de riscos que demandem mais teste;
  - Identificar necessidades de treinamento;
  - Identificar oportunidades de melhoria no processo;
  - Controlar e rastrear a situação do projeto;
  - Fornecer a base para estimativa de esforço de teste.

Benefícios das Métricas (2)

- Identificar áreas de risco que exigem mais teste
  - Análise ou Lei de Pareto (80/20)

- Muitos fenômenos de software seguem a distribuição de Pareto: 80% dos resultados são decorrentes de 20% do todo.

- Coletar e analisar a **densidade de defeitos** por módulo permite identificar áreas de risco em potencial.
- Complexidade também é uma métrica importante na identificação de áreas de risco em potencial.

Benefícios das Métricas (2)

- Identificar áreas de risco que exigem mais teste
  - Análise ou Lei de Pareto (80/20)

- Muitos fenômenos de software seguem a distribuição de Pareto: 80% dos resultados são decorrentes de 20% do todo.
- Por exemplo: 80% dos defeitos (não necessariamente os mesmos) vem de 20% dos módulos.

- Coletar e analisar a **densidade de defeitos** por módulo permite identificar áreas de risco em potencial.
- Complexidade também é uma métrica importante na identificação de áreas de risco em potencial.

Benefícios das Métricas (2)

- Identificar áreas de risco que exigem mais teste
  - Análise ou Lei de Pareto (80/20)

- Muitos fenômenos de software seguem a distribuição de Pareto: 80% dos resultados são decorrentes de 20% do todo.
- Por exemplo: 80% dos defeitos (não necessariamente os mesmos) vem de 20% dos módulos.
- Isso geralmente ocorre pois existem partes do sistema que são usualmente mais complexas ou escritas a partir de uma especificação mais pobre, incompleta ou inconsistente.

- Coletar e analisar a **densidade de defeitos** por módulo permite identificar áreas de risco em potencial.
- Complexidade também é uma métrica importante na identificação de áreas de risco em potencial.

Benefícios das Métricas (3)

- Identificar necessidade de treinamento
  - Métricas que medem informações sobre tipo e concentração de defeitos no software, artefatos de teste ou processo, permitem identificar necessidades de treinamento.
  - Por exemplo, uma alta taxa de defeitos relacionados ao "vazamento de memória" pode demandar treinamento para evitar a ocorrência desse tipo de defeito.

Benefícios das Métricas (4)

- Identificar oportunidades de melhoria de processo
  - A mesma análise feita para identificar necessidade de treinamento pode ser utilizada para identificar oportunidade de melhoria de processo (de teste e de desenvolvimento)
  - Ao invés de treinamento, alterações nos processos podem evitar a ocorrência de determinados tipos de defeitos.

Benefícios das Métricas (5)

- Controlar e rastrear a situação do projeto
  - Gerentes de teste e os demais interessados precisam usar métricas para controlar o esforço de teste e rastrear seu progresso.
  - Medições do número, tipo, severidade e distribuição dos defeitos e o número de casos de teste executados são formas de monitorar o progresso da execução dos testes.

Benefícios das Métricas (6)

- Fornecer a base para estimativa de esforço de teste
  - Sem métricas adequadas, gerentes e profissionais tornam-se impotentes para realizar qualquer estimativa.
  - Estimativas como:
    - Quanto tempo o teste irá demorar?
    - Quantos defeitos devem ser identificados?
    - Quantos testadores são necessários?
    - Dentre outras. Só podem ser definidas com base em experiências anteriores (métricas).

Benefícios das Métricas (7)

- Justificar orçamento, ferramenta e treinamento
  - Frequentemente os gerentes de teste percebe falta de pessoal para a execução dos testes.
  - Entretanto, sem métricas para apoiar seu sentimento qualquer demanda de aumento na equipe dificilmente é aceita.

Benefícios das Métricas (8)

- Fornecer limites para tomada de ação
  - Um sinal de maturidade no uso de métricas é o uso de limites (*meters*) para sinalizar o disparo de ações.
  - Considerado um sinal de maturidade pois para ser efetivo precisam ser planejados e estabelecidos no início do projeto ou durante um projeto anterior.

Regras de Ouro (1)

- Pergunte à Equipe
  - Pergunte aos desenvolvedores e testadores quais métricas vão ajudar a equipe a desempenhar suas atividades.
  - Tem que se acreditar na métrica para que ela seja coletada e passe a exercer o seu papel.

Regras de Ouro (2)

- Use uma métrica para validar a métrica
  - Raramente se tem confiança na tomada de decisão baseada em uma única métrica.
  - Gerentes são aconselhados a validar uma métrica com outra métrica.
  - Por exemplo, a eficácia de teste pode ser medida por:
    - Medida de cobertura;
    - Eficiência na remoção de defeitos.

Regras de Ouro (3)

- Normalize o valor da métrica
  - Como cada sistema, projeto, versão, pessoa, etc. é único, métricas precisam ser normalizadas.
  - Para reduzir a normalização é importante tentar comparar objetos similares, tais como dois projetos similares em tamanho, escopo e complexidade.

Regras de Ouro (4)

- Meça o valor de coletar a métrica
  - Coletar uma métrica exige tempo e esforço.
  - É importante identificar qual a utilidade da métrica frente o esforço exigido na sua coleta e análise.
  - Coletar dados que ninguém usa é inútil.

Regras de Ouro (5)

- Reveja periodicamente a necessidade de cada métrica
  - Métricas úteis para um projeto podem não ter valor para outro.
  - Por exemplo, tempo gasto para escrever casos de teste pode ser útil para teste sistemático mas inútil para o teste exploratório.

Regras de Ouro (7)

- Facilite a coleta e análise das métricas
  - Idealmente, métricas deveriam ser coletadas automaticamente.
  - Coleta de métrica como produto de outra atividade também é uma boa alternativa.
  - Por exemplo, informações sobre defeitos devem ser coletadas para isolar e corrigir defeitos, mas essas mesmas informações podem se usadas para acompanhar o resultado dos testes.
  - Inevitavelmente, algumas terão que ser coletadas manualmente.

Regras de Ouro (8)

- Respeite a confidencialidade dos dados
  - Gerentes de teste devem estar cientes de que determinadas informações são restritas a alguns grupos ou indivíduos.
  - É importante identificar quais métricas devem estar disponíveis para quais grupos.

Regras de Ouro (9)

- Procure por interpretações alternativas
  - Podem existir formas diferentes de interpretar o mesmo dado.
  - Às vezes vale a pena apresentar os dados brutos, processados e interpretados.
  - É importante perguntar aos envolvidos o que os dados brutos representam para eles.
  - A interpretação deles pode ser diferente.

Sobre estatística...

"Existem três tipos de mentira: mentira, mentiras malditas e estatística." (Benjamin Disraeli)

Regras de Ouro (10)

- Formate o dado para a audiência
  - Existem diferentes grupos de interessados em determinadas informações.
  - É importante apresentar os dados formatados conforme o conhecimento prévio e necessidade de cada grupo.

Regras de Ouro (11)

- Oferecer treinamento
  - Para muitos engenheiro de software métricas são misteriosas.
  - É importante treiná-los para explicar:
    - Porque a métrica é coletada;
    - Como ela é coletada;
    - Com que frequência ela é coletada;
    - Quem tem acesso e usa a métrica;
    - Como alterar o parâmetro da métrica.

Regras de Ouro (12)

- Criar uma planilha de métricas

Item	**Descrição**
Nome	Nome da métrica
ID	Acrônimo para a métrica
Área	Tamanho/Planejamento/Recurso/Qualidade/Retrabalho/...
Descrição	Breve descrição do que está sendo medido e porque.
Observação	Como a medição é realizada.
Frequência	Com qual frequência a métrica é coletada e atualizada.
Escala	Qual unidade de medida é usada.
Intervalo	Qual intervalo de valores é possível.
Passado	Quais valores foram coletados no passado.
Atual	Qual o valor atual ou do último resultado.
Esperado	É esperada a ocorrência de mudanças? Se sim quais?
Limite (*meters*)	Ação que deve ser realizada caso o resultado atinja um valor limite.

Planilha de documentação de métrica adaptada de Hetzel (1993).

Exemplos de Métricas (1)

Métrica	**Exemplo Desenvolvimento**	**Exemplo Teste**
Tamanho	Número de módulos ou linhas de código	Número de módulos, linhas de código ou casos de teste
Planejamento	Número de módulos completados por unidade de tempo	Número de casos de teste escritos ou executados por unidade de tempo
Recurso	Dinheiro gasto, horas trabalhadas	Dinheiro gasto, horas trabalhadas
Qualidade	Número de defeitos por linha de código	Eficiência na remoção de defeitos, cobertura
Retrabalho	Linhas de código escrita para corrigir defeito	Número de ciclos de teste para testar defeito corrigido

Exemplos de Métricas (2)

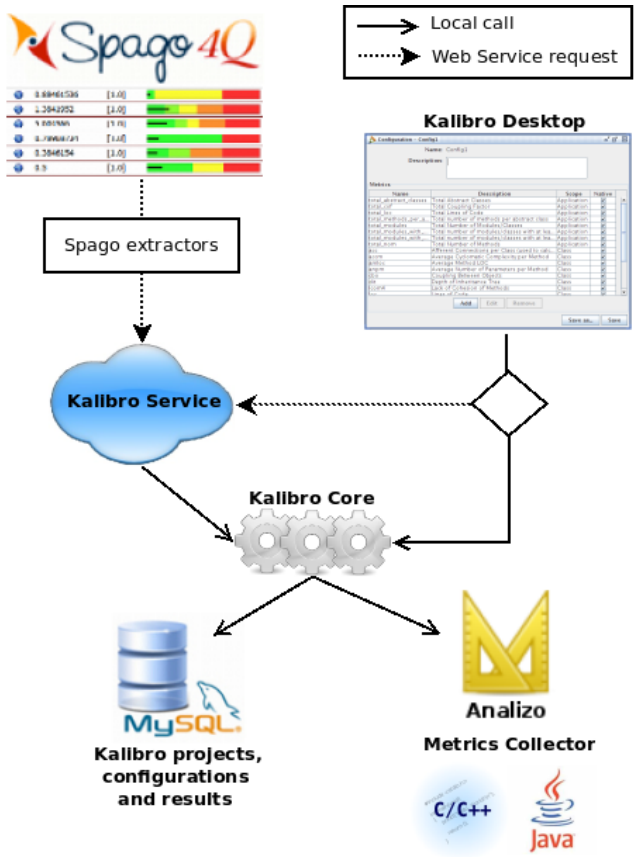
- Um estudo realizado por Rick Craig e Bill Hetzel, comentado em (Craig e Jaskiel, 2002), sintetiza um conjunto de métricas utilizado em bons projetos. São elas:

- Defeitos nos testes
- Defeitos após liberação
- Problemas abertos
- Questões abertas
- Desempenho do cronograma
- Mudanças nos planos e cronograma
- Resultados de teste
- Confiabilidade
- Tempo gasto corrigindo defeitos
- Defeitos incluídos após correção
- Linhas de código
- Aderência ao processo
- Cobertura de código
- Complexidade
- Custo de retrabalho
- Custo de qualidade

Automatização (1)

- Existem iniciativas de automatização da coleta de diferentes tipos de métrica.
- A maioria é destinada à coleta de métricas de produto a partir do código fonte.
- Uma dessas iniciativas é o projeto Kalibro (http://kalibro.org/) (Meirelles, 2013)
  - Visa a facilitar o entendimento de métricas de produto.
  - Permite a um especialista em métricas definir os limites de interesse para cada métrica.

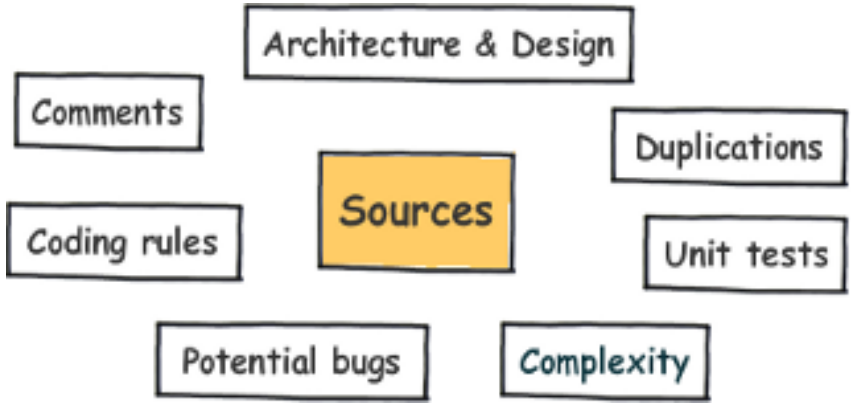
Automatização (2)



Fonte: http://ccsl.ime.usp.br/kalibro.

Automatização (3)

- Outra ferramenta bastante conhecida e que integra diferentes métricas estáticas e dinâmicas em um ambiente de fácil extensão é a SonarQube (http://www.sonarqube.org/).
- Uma versão on-line do SonarQube está disponível em http://nemo.sonarsource.org/.
- São coletadas métricas em sete eixos de qualidade, conforme ilustrado abaixo:



Eixos de qualidade do SonarQube. Fonte http://www.sonarqube.org/.

Conclusão (1)

- Técnicas de Análise Estática e Dinâmica são complementares.
- Análise Estática permite a avaliação de diversos artefatos gerados durante todo o ciclo de vida de desenvolvimento.
- A automatização da análise estática é mais frequente para a coleta de dados a partir do código fonte.
- Existem diferentes métricas de produto que podem ser coletadas a partir de código-fonte.
- O importante é coletar métricas que serão úteis para a análise de risco e tomada de decisão.
- Ferramentas como Mezuro (Meirelles, 2013) e SonarQube contribuem para a coleta automática e visualização de métricas de código-fonte.

Referências I

Anderson, P. The use and limitations of static-analysis tools to improve software quality. *CROSSTALK The Journal of Defense Software Engineering*, v. 21, n. 6, p. 18–21, 2008.

Basili, V.; Green, S.; Laitenberger, O.; Lanubile, F.; Shull, F.; Sørumgård, S.; Zelkowitz, M. V. The empirical investigation of perspective-based reading. *Empirical Software Engineering*, v. 1, n. 2, p. 133–164, 1996. Disponível em http://dx.doi.org/10.1007/BF00368702

Craig, R. D.; Jaskiel, S. P. *Systematic software testing*. Artech House Publishers, 2002.

Hetzel, B. *Making software measurement work: Building an effective measurement program*. Wiley-QED, 1993.

ISO/IEC/IEEE Systems and software engineering – vocabulary. 2010.

Kirkov, R.; Gagre, E. Source code analysis – an overview. *CYBERNETICS AND INFORMATION TECHNOLOGIES*, v. 10, n. 2, p. 60–77, 2010.

Meirelles, P. *Monitoramento de métricas de código-fonte em projetos de software livre*. Tese de doutoramento, Instituto de Matemática e Estatística – Universidade de São Paulo, São Paulo, SP, disponível em: http://www.teses.usp.br/teses/disponiveis/45/45134/tde-27082013-090242/pt-br.php. Acesso em: 02/02/2014, 2013.