# Integração Contínua no Processo de Desenvolvimento de Software

Alberto Guimarães Viana e José Marcelo Pereira de Araujo

***Palavras-chave*—Integração Contínua, SCAMPI, Qualidade de Software.**

## I. Resumo

A EVOLUÇÃO das tecnologias em ambientes de desenvolvimento de sistemas proporciona atualmente várias formas e metodologias para a construção de softwares. Neste sentido, as equipes de desenvolvimento são formadas proporcionalmente ao tamanho do software, o que infere grandes quantidades de atribuições e atividades a serem executadas. Deste modo, gerenciar o desenvolvimento do software torna-se uma tarefa difícil e dispendiosa, o que requer processo contínuo de integração de atividades, visando obter melhor qualidade e resultados com o desenvolvimento do software.

Partindo desta premissa, de que a integração contínua é uma prática de desenvolvimento de software onde os membros de um time integram seu trabalho frequentemente, geralmente cada pessoa integra pelo menos diariamente – podendo haver múltiplas integrações por dia. Cada integração é verificada por um *build* automatizado (incluindo testes) para detectar erros de integração o mais rápido possível. Muitos times acham que essa abordagem leva a uma significante redução nos problemas de integração, além de permitir que um time desenvolva um software coeso mais rapidamente [1].

Neste sentido, o presente trabalho propõe a utilização das seguintes tecnologias para a implantação e utilização da integração contínua em uma empresa:

1. *Jenkins* servidor de integração contínua;
2. PHP_*CodeSniffer* verifica se o padrão de codificação está sendo seguido;
3. PHP *Unit Testing Framework* provê um framework para a escrita e execução de testes, além de gerar a estatística da cobertura do código.
4. PHP *Copy/Paste Detector* (PHPCPD) examina o código e identifica trechos de código duplicados.
5. PHP *Line of Code* (PHPLoc) ferramenta que mede o tamanho do *software*.
6. PHP *Mess Detector* (PHPMD) verifica a existência de possíveis *bugs*, expressões complexas e parâmetros, métodos e propriedades não usadas.
7. PHP *Depend* é uma ferramenta que analisa o código e gera sua métrica.

Vários testes foram realizados nos códigos dos sistemas referências utilizados na validação do ambiente de integração contínua proposto neste trabalho, dentre eles, e como exemplo, seguem alguns resultados.

A Fig. 1 indica o resultado positivo quanto à realização de todos os teste unitários, ou seja, a ferramenta não atribuiu nenhuma falha nos códigos testados, o que é demonstrado por todo o gráfico está em cor verde.

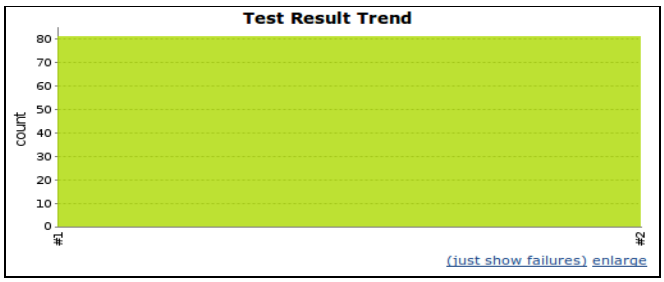


Fig. 1. Teste Unitário.

A Fig 2, indica que código está coberto 100% pelos testes, ou seja, todos os métodos e classes foram testados e a ferramenta não atribuiu nenhuma falha nos códigos testados.

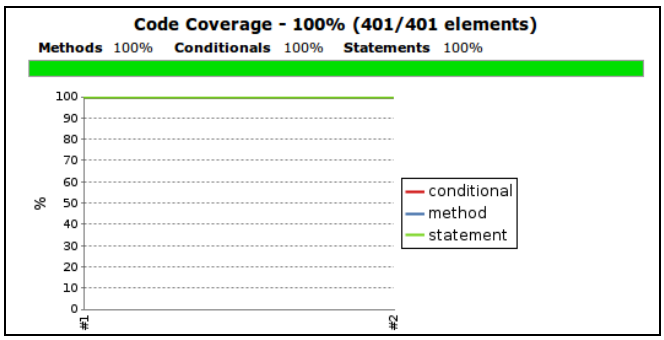


Fig. 2. Code Coverage.

A análise do resultado ilustrado na Fig. 3 nos permite admitir algumas conclusões, como por exemplo, avaliar as informações das linhas destacadas em cor amarela.

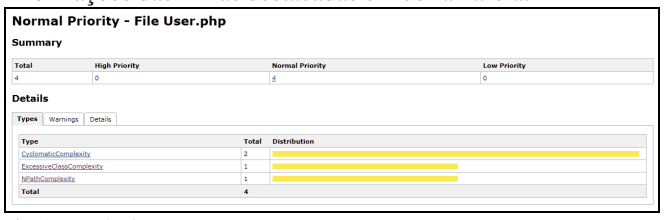


Fig. 3. Resultado PHPMD.

Na primeira linha, intitulada *Cyclomatic Complexity*, indica a

existência de uma grande quantidade de condicionais, sugerindo a refatoração do código. A segunda linha intitulada *Excessive Class Complexity* informa que a classe testada possui uma grande quantidade de métodos, gerando um alto grau de dependência das classes em relação a mesma, o que sugere um maior esforço durante a manutenção do código. Para finalizar a terceira linha intitulada *NPathComplexity* mensura o número de caminhos de execução acíclicos, ou seja, a quantidade de caminhos lógicos executados, exceto para as interações de um loop. Neste teste, somente um método apresentou deficiência na quantidade de caminhos lógicos em um programa. No entanto, é bom enfatizar que estes testes apenas alertam, quanto a possíveis perdas de performance e manutenbilidade no código.

O teste ilustrado na Fig. 4 acima indica a quantidade de repetições de código, o que neste caso não ocorreu, ou seja, no sistema por completo não houve a repetição desnecessária de códigos.

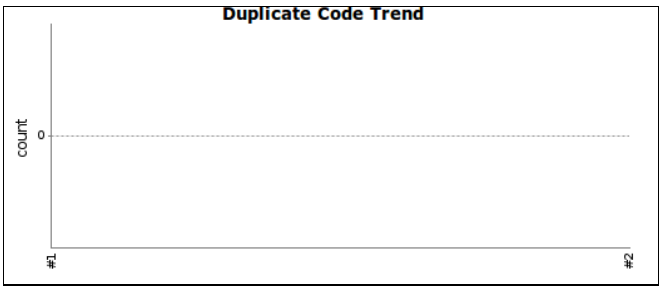


Fig. 4. Teste de repetição de código.

Por último a sintaxe dos códigos foram testados, e como nos resultados ilustrados na Fig. 5 acima, a ferramenta indica uma advertência quanto a presença de números na nomenclatura de algumas variáveis, o que foge ao padrão de codificação estabelecido.

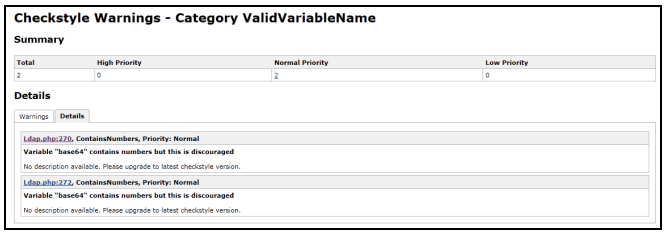


Fig. 5. Detalhes do PHPCS.

Além dos testes aplicados no sistema, o presente trabalho teve a preocupação de avaliar o processo de integração contínua, no intuito de verificar a adequação do mesmo, aos padrões de mercado e no provimento de qualidade ao software. Neste sentido, o instrumento de avaliação utilizado foi o SCAMPI, definido como método para avaliar modelos de processos [2], e atrelado aos princípios de qualidade do produto de software proferidos pela norma ISO 9126. Os níveis de adequação do artefato avaliado obedeceram aos seguis níveis avaliação ilustrados na Fig. 6.

| Nível de Implementação da Prática | Caracterização |
|---|---|
| FI (Fully Implemented) | • O indicador direto está presente e julgado adequado<br>• Existe pelo menos um indicador indireto e/ou afirmação para confirmar a implementação<br>• Não foi notada nenhuma fraqueza substancial |
| LI (Largelly Implemented) | • O indicador direto está presente e julgado adequado<br>• Existe pelo menos um indicador indireto e/ou afirmação para confirmar a implementação<br>• Foi notada uma ou mais fraquezas |
| PI (Partially Implemented) | • O indicador direto não está presente ou é julgado inadequado<br>• Artefatos ou afirmações sugerem que alguns aspectos da prática estão implementados<br>• Foi notada uma ou mais fraquezas |
| NI (Not Implemented) | • Qualquer situação diferente das acima |

Fig. 6. Níveis de Avaliação do SCAMPI

A Tabela I ilustra parte do relatório de avaliação do processo de integração contínua, em específico a avaliação da qualidade interna e externa denominada confiabilidade, a qual indica aproximadamente o percentual de 84%, em termos de completitude da confiabilidade do processo de integração contínua descrito neste trabalho. Neste sentido, infere dizer que as tecnologias adotadas estão funcionais, validam os códigos dos sistemas, além de, estarem em conformidade com os princípios de qualidade de produto de software estabelecidos pela ISO 9126.

Tabela I. Instrumento de Avaliação.

| Item de Avaliação | Qualidade Interna e Externa – Confiabilidade | Integração | Resultado |
|---|---|---|---|
| I- Maturidade | Instrumento de documentação e detecção das falhas ocorridas na execução do processo de integração contínua. | Existe | FI |
| I- Maturidade | Relatório indicando o nível de falhas ocorridas durante a execução do processo de integração contínua. | Existe | FI |
| I- Maturidade | Relatório indicando os erros acometidos pelas tecnologias utilizadas no processo de integração contínua. | Existe | FI |
| I- Maturidade | Os resultados do processo são transparentes aos usuários? | Existe | FI |
| II- Tolerância a falhas | O processo emite alertar contra falhas ocorridas durante a execução do mesmo. | Existe | FI |
| II- Tolerância a falhas | O processo determina quais os códigos estão sem testes unitários ou não foram testados? | Existe | FI |
| II- Tolerância a falhas | O processo emite um relatório indicando o motivo da interrupção durante a sua execução? | Não existe | NI |
| II- Tolerância a falhas | O processo emite um relatório indicando as possíveis falhas ocorridas durante a sua execução? | Existe | FI |
| III – Recuperação | Existe um procedimento para reinicialização do processo? | Não existe | NI |
| III – Recuperação | Existe um procedimento para guardar os testes unitários? | Existe | FI |
| III – Recuperação | Existe um procedimento para recuperações os relatórios com testes anteriores? | Existe | FI |
| III – Recuperação | Existe um relatório indicando as diferenças entre as execuções dos testes unitários? | Existe | PI |

## REFERÊNCIAS

[1] CMMI, Standard CMMISM Appraisal Method for Process Improvement (SCAMPISM), Version 1.1: 2001.
[2] DUVALL, Paul M. Continuous Integration. Improving Software Quality and Reducing Risk. Ed. Addison Wesley. 2007.