Faculdade de Tecnologia de Americana
Curso de Processamento de dados

TESTES E AUTOMAÇÃO EM APLICAÇÕES WEB

GUSTAVO LABBATE GODOY

Americana, SP
2010

Faculdade de Tecnologia de Americana
Curso de Processamento de dados

TESTES E AUTOMAÇÃO EM APLICAÇÕES WEB

GUSTAVO LABBATE GODOY

gustavolabbate@gmail.com

Estágio Supervisionado – ESTANAL, desenvolvido em cumprimento à exigência curricular do Curso de Processamento de dadosda FATEC Americana, sob orientação do Prof. João Sebastião de Oliveira Bueno

Americana, SP
2010

FICHA CATALOGRÁFICA elaborada pela
BIBLIOTECA – FATEC Americana – CEETPS

G533t	Godoy, Gustavo Labbate Testes e automação em aplicações WEB / Gustavo Labbate Godoy – Americana: 2010. 44f. Monografia (Graduação em Processamento de dados). - Faculdade de Tecnologia de Americana – Centro Estadual de Educação Tecnológica Paula Souza. Orientador: Prof. João Sebastião de Oliveira Bueno 1. WEB – rede de computadores 2. Informática I. Bueno, João Sebastião de Oliveira II. Centro Estadual de Educação Tecnológica Paula Souza – Faculdade de Tecnologia de Americana. CDU: 681.519

Bibliotecária responsável Ana Valquiria Niaradi – CRB-8 região 6203

BANCA EXAMINADORA

Prof. João Sebastião de Oliveira Bueno (Orientador)

Prof. Wagner Siqueira Cavalcante (Convidado)

Prof. Antonio Lacerda (Presidente da banca)

AGRADECIMENTOS

Agradeço a meu orientador pelas dicas e correções que fizeram com que meu trabalho tivesse continuidade. Agradeço à meus colegas de trabalho, que me mostraram caminhos para que minhas pesquisas fossem traduzidas em palavras e terminassem nesse trabalho. Agradeço às dificuldades do dia-a-dia que me propiciaram material e conteúdo para que essa monografia fosse construída. Agradeço à minha família, pelo apoio, incentivo e cooperação e pelo surgimento das oportunidades que fizeram este trabalho de pesquisa profissional se tornar um trabalho de pesquisa acadêmico.

DEDICATÓRIA

Dedico este trabalho principalmente à minha família, Ana Paula, Talita e Guilherme, pois é com a compreensão deles que posso estar avançando nesta etapa em minha carreira, e é por eles que tudo isso está sendo feito.

Dedico também aos professores que fazem e fizeram parte da minha carreira acadêmica dentro da Fatec, pois com suas palavras, parte de meu perfil profissional e acadêmico foi moldado.

RESUMO

O presente trabalho procura mostrar as técnicas de aplicação de teste de *software* em aplicações, em especial as aplicações para ambiente *web*.

O teste de *software*, integrado ao processos de qualidade de *software*, vêm, aos poucos, ganhando força e se tornando uma carreira promissora e essencial nas empresas de desenvolvimento e manutenção de *software*.

O trabalho apresenta também conceitos e ferramentas para automação de testes, sempre levando em consideração os conceitos e teorias dos processos de teste.

Esse trabalho apresenta de forma conceitual os processos de testes, baseados em pesquisa bibliográfica e boa parte teórica, principalmente em automação de testes, que se baseia em coleta de informações em ambiente profissional.

As principais fontes são materiais conceituados, base para certificações e de grande conhecimento entre profissionais na área. Periódicos estrangeiros também foram consultados no intuito de obter informações atualizadas e em um ambiente diferente do mercado tecnológico brasileiro.

Palavras Chave: testes, automação, aplicações web.

ABSTRACT

This paper shows the techniques of software testing in applications, particularly applications for web environment.

The test software, integrated into the processes of software quality, are gradually gaining strength and becoming a vital and rewarding career in business development and software maintenance.

The paper also presents concepts and tools for test automation, always taking into consideration the concepts and theories of testing procedures.

This paper presents concepts of testing processes, based on a literature and much theoretical, especially in test automation, which is based on collecting information in a professional environment.

The main sources are material reputable, basis for certification and of great knowledge among professionals in the area. Foreign periodicals were also consulted in order to obtain updated information in a different environment of brazilian technological market.

Keywords: tests, automation, web applications.

SUMÁRIO

LISTA DE FIGURAS

LISTAGENS

INTRODUÇÃO

No cenário de tecnologia de informação atual, manter a integridade dos *softwares* tem se tornado uma busca constante das empresas.

Manter profissionais preparados e adaptados aos processos de qualidade de *software* é um objetivo desejado entre essas empresas, que investem boa parte dos recursos dos projetos nessas atividades.

O objetivo geral do trabalho foi apresentar as técnicas e processos de teste que atendem às principais normas de certificações de qualidade em teste de *software* atuais, além de apresentar as técnicas e ferramentas para obter a qualidade constante através de automação. Esse conceito, apesar de não muito recente, vem sendo pouco empregado nas maiorias das empresas, e será uma tendência.

O método de pesquisa utilizado foi pesquisa em referências literárias, periódicos e artigos, em obras de grande conceituação na área de testes, tendo como duas fontes materiais preparatórios para certificação em qualidade de software, de âmbito internacional, além de coleta de informações em ambiente profissional.

Outras fontes de pesquisa foram periódicos e referências bibliográficas internacionais, onde é possível traçar um paralelo entre as tecnologias e técnicas empregadas no mercado brasileiro e no mercado internacional.

O trabalho foi estruturado em dois capítulos, sendo que o primeiro, subdivido em sub-capítulos, mostra uma breve história e os conceitos e técnicas de teste. Apresenta também conceitos de testes em aplicações *web*, com suas particularidades dos testes em ambiente *desktop*.

O segundo capítulo trata das técnicas de automação, apresentando também ferramentas para automação de processos e de testes. Procura enfatizar a utilização paralela dessas ferramentas, para que diversas técnicas de testes apresentadas no capítulo 1 sejam atendidas e executadas.

Com base nestas informações, o terceiro capítulo procura mostrar as considerações finais, tanto para as técnicas de teste quanto para a automação de processos e testes.

1 TESTE DE *SOFTWARE*

O pessimista vê dificuldade em cada oportunidade; o otimista vê oportunidade em cada dificuldade.

(Winston Churchill)

Há alguns anos, entre os anos 70 a 90, o desenvolvimento de aplicações era algo restrito a um seleto grupo de profissionais. As aplicações eram bem direcionadas, devido à tecnologia empregada na época pois, literalmente, eram construídas para o que seriam usadas. Esse seleto grupo de profissionais, pouco encontrados no mercado de trabalho, eram responsáveis por toda a etapa do desenvolvimento do *software*, desde a concepção até a entrega final. A qualidade também era de sua responsabilidade. Os desenvolvedores deveriam garantir que o que estava sendo construído estava de acordo com o especificado, através de testes diretos ao *software*.

Emerson Rios (2007:11), conceitualiza:

> (…) os testes eram efetuados pelos próprios desenvolvedores de software, cobrindo aquilo que hoje chamamos de testes unitários e testes de integração. (…) Muitas vezes, os próprios usuários eram também envolvidos para aprovar os resultados desses testes ou mesmo participar da criação dos dados dos testes. Isso gerava uma incidência muito grande de defeitos, que apareciam quando os sistemas já estavam em produção (...)

Dessa forma, os erros encontrados em produção, muitas vezes eram aceitos, pois o custo para se recontratar o profissional, interromper produções, eram maiores do que se adaptar ao erro. Em muitos casos, continuar usando o *software* com erro, se adequando ao problema, era mais prático.

Porém, com o advento da tecnologia para Internet, a *web*, o conceito começa a mudar. A convergência de aplicações *desktop* para a Internet aumentou, devido à facilidade para o uso dessas aplicações pelos usuários da Internet. As aplicações começaram a ter muitas funcionalidades incluídas, que antes, sem a Internet, não era possível.

Sistemas de comércio eletrônico, consulta on-line, redes sociais, entre outros, começaram a surgir e a expandir a necessidade de mais profissionais, e em consequência disso, uma redução no prazo da entrega desses produtos começou a ser exigida.

Novas linguagens começam a ser utilizadas por uma quantidade cada vez maior de profissionais, pois o conhecimento passa a ser mais difundido. Novas aplicações começam a ser escritas por profissionais preparados e outros nem tanto. As aplicações passam a ser mais baratas, de fácil aquisição pelas empresas. Passam a ser mais complexas, exigir mais recursos humanos para a construção, em um prazo para entrega muito menor que os antecessores. A concorrência entre linguagens e profissionais alavanca essa queda no prazo. O que acontece é o aumento de defeitos, devido à ineficácia na validação da qualidade, nem sempre prezada pelos desenvolvedores, com o pouco tempo que é proporcionado.

Emerson Rios (2007:11) define, em seu livro Base de Conhecimento em teste de *Software:*

> (..) Isso levou as organizações a procurar novos caminhos para melhorar a qualidade do software desenvolvido por seus técnicos. Um desses caminhos foi o aprimoramento da atividade de teste.

Como medida de contorno e evitar que essas situações ocorram, é necessário que se estabeleça o papel de uma equipe de testes. Estes profissionais devem ser independentes da equipe de desenvolvimento, porém devem ser integrados às suas atividades. O papel da equipe de teste é garantir que o *software* que está sendo desenvolvido está atendendo aos requisitos iniciais, mantendo a integridade do começo ao fim.

Processos de qualidade de software, englobando não somente a área de testes, são reguladas por alguns institutos, como a SEI (*Software Engineering Institute*) (http://www.sei.cmu.edu/cmmi/) que regula os processos do CMMI (*Capability Maturity Model Integration*) e pelo projeto MPS.br (Melhoria dos Processos do Software Brasileiro), que é apoiado pelo Ministério da Ciência e Tecnologia, Finep (Financiadora de Estudos e Projetos) e do Banco Interamericano de Desenvolvimento, sendo desenvolvido, no Brasil, pela Softex

(http://www.softex.br/mpsbr/_home/default.asp). Esses processos pretendem regular e certificar as empresas que se adequem a seus processos.

O processo de teste, normalmente incluído dentro de um processo de qualidade de *software* (como CMMI ou o MPS.br), procura abranger desde os testes funcionais (ou “o que o sistema deve fazer”) até testes como desempenho, confiabilidade, etc.

Ao se tratar os erros encontrados durante a fase de desenvolvimento, garante que a correção possa ser feita durante essa mesma etapa, reduzindo os custos de manutenção que porventura apareceriam. Para ilustrar essa situação, convém citar a regra 10 de Myers, que diz “quanto mais cedo os defeitos nos sistemas forem corrigidos, menores são os custos de correção para o Projeto” (Myers. 1979. 8 – The art of Software Testing). Esta regra visa definir a atividade principal da equipe de teste: encontrar erros durante o desenvolvimento, reduzindo os custos de manutenção e satisfazendo as necessidades iniciais do projeto.

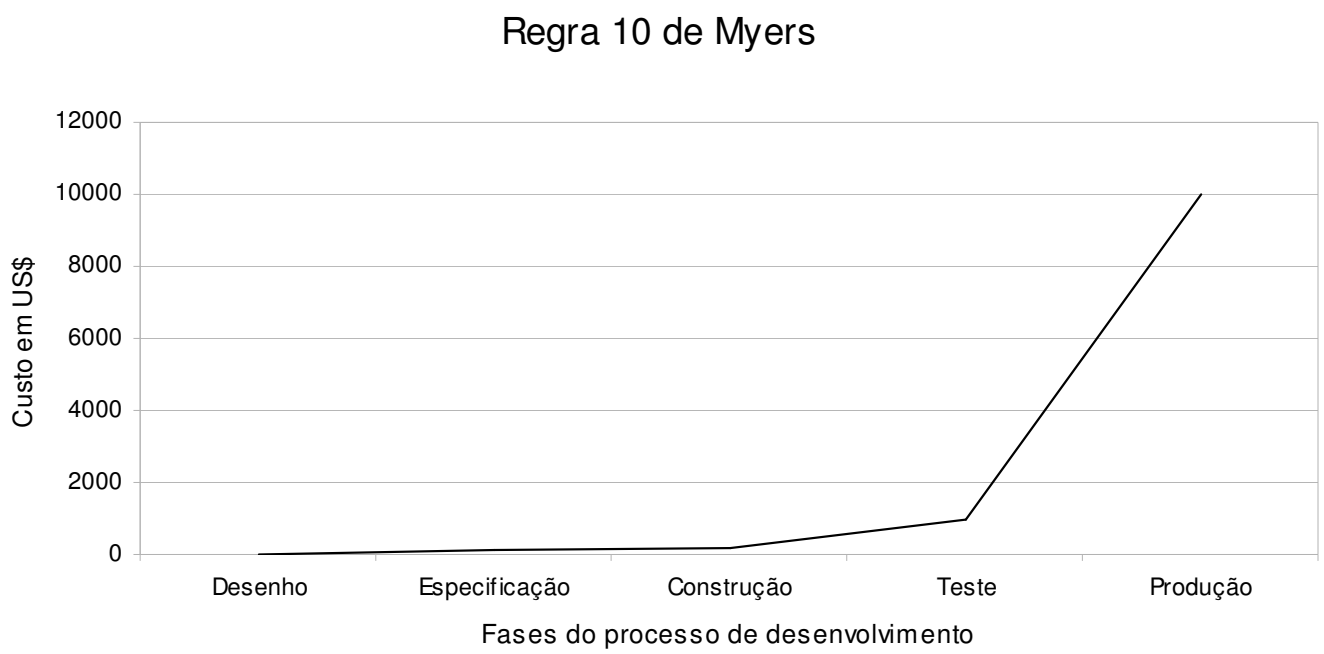


Figura 1: Regra 10 de Myers – Custo da correção de erro
Fonte: Souza, A. B. & RIOS, E. & Cristalli, R. S. & Moreira, T. *Base de Conhecimento em Teste de Software.* São Paulo : Martins, 2007.

Baseado nesse conceito, supõe-se que as atividades de validação e verificação do sistema sejam durante a etapa de construção e testes. Durante as fases de desenho e especificação, os testes são baseados em documentos. Durante a fase de construção, os testes já podem ser executados diretamente no sistema ou no código que está sendo construído para o sistema. Cabe à equipe de implementação e à equipe de testes esse trabalho em conjunto para validação e verificação do que está sendo construído, para eventuais correções em tempo de construção (Emerson Rios, 2007:19).

A atividade de teste analisa os sintomas apresentados durante a execução de um programa, e caracterizar perante suas principais causas. O teste pode identificar erros e também falhas. Erros são caracterizados por ações criadas por seres humanos, normalmente como engano. Os erros podem ser em documentação, em implementação ou em implantação. O erro ocasiona um defeito e esse defeito, na execução do sistema, torna-se uma falha. A falha é quando o sistema não conseguiu executar o que deveria. Quando o teste encontra a falha, deve-se encontrar a causa raiz, para identificação do erro e correção. Algumas falhas também podem ser ocasionadas por eventos externos à ação humana (recursos da máquina, incompatibilidade de outro *software*, etc). Nesses casos, a identificação da causa raiz pode gerar uma melhoria no *software* ou simplesmente uma correção no ambiente. (BSTQB, 2007:10)

De acordo com o Syllabus, do *Brazilian Software Training Qualification Board* (BSTQB):

> Os defeitos ocorrem porque os seres humanos são passíveis de falha e porque existe pressão no prazo, códigos complexos, complexidade na infra-estrutura, mudanças na tecnologia e/ou muitas interações do sistema.
> Falhas também podem ser causadas por condições do ambiente tais como: radiação, magnetismo, campos eletrônicos e poluição, que podem causar danos em software embarcado (*firmware*) ou influenciar a execução do *software* pelas mudanças de características de *hardware*.

Os testes devem seguir um critério, um padrão para que seja possível identificar e tomar as devidas ações. Toda atividade de teste deve seguir procedimentos de execução e de documentação. Essa padronização tem por objetivo prover resultados que possam medir a qualidade do *software*. Baseado no escopo dos requisitos funcionais e não-funcionais e nas características do *software*, é possível determinar qual a sua maturidade.

Segundo o Syllabus (BSTQB, 2007:12:

> (…) Uma visão comum do processo de teste é de que ele consiste apenas na fase de execução, como executar o programa. Esta, na verdade, é uma parte do teste, mas não contempla todas as atividades do teste. (…)
>
> No processo de teste, diferentes pontos de vista levam a diferentes objetivos. (…) No teste feito em desenvolvimento, o principal objetivo pode ser causar o maior número de falhas possíveis. (…) No teste de aceite, o objetivo pode ser confirmar se o sistema está funcionando conforme o esperado.

1.1 PROCEDIMENTOS DE TESTE DE *SOFTWARE*

Hoje, as atividades de teste podem ser regulamentadas pelos órgãos certificadores ou por processos de qualidade de *software* (como o CMMI ou MPS.Br). O procedimento adequado cabe à equipe definir, pois muitos conceitos podem variar em termos de metodologia de projeto aplicada. Porém, os conceitos básicos e genéricos de teste de *software* mantém-se os mesmos.

As atividades de teste em um projeto devem ser definidas da mesma forma que as outras atividades. Convém-se criar um projeto de teste de *software* em paralelo, pois é usual que a equipe de teste também trabalhe em outros projetos. Isso evitaria a alocação indevida de recursos.

Um projeto de teste, em paralelo ao projeto do sistema, deve ser definido nas fases iniciais do projeto de *software*. Essa abordagem garante que lições aprendidas em projetos anteriores economizem tempo no design do projeto. Pode garantir que documentos não serão escritos com erros, fazendo com que estes erros sejam implementados e, posteriormente, gerem falhas no sistema.

Em uma fase inicial, cria-se o Plano de Teste, que contém todas as informações necessárias para as etapas de teste. Nele, encontram-se informações como ambientes de teste, recursos disponíveis e utilizados, responsabilidades e prazos, tipos de testes a serem executados, resultados esperados, quando os testes devem iniciar e quando devem ser interrompidos, além de outras informações que se façam necessárias. Define-se também quais documentos contendo os casos de testes serão escritos.

Esse procedimento é integrante do ciclo de desenvolvimento de *software*. Criou-se o conceito “V” de teste, que determina procedimentos de execução e checagem, do inicio ao fim do ciclo. O conceito preza que os executores, que no caso são os implementadores do sistema, trabalhem em conjunto com os validadores, que são os testadores. Com isso, os riscos do projeto se reduzem, devido à interação entre as partes, que resolvem de imediato os problemas. Segundo os autores do livro **Base de conhecimento em teste de software:**

> (..) O grupo que FAZ trabalha com o objetivo de implementar o sistema, e a equipe que CONFERE, simultaneamente, executa procedimentos de teste visando minimizar ou eliminar riscos. Com esse enfoque, se os grupos trabalharem juntos e de maneira integrada, o alto nível de riscos que caracteriza os projetos de desenvolvimento de software irá decrescer a um patamar aceitável que permita a conclusão bem-sucedida do projeto.

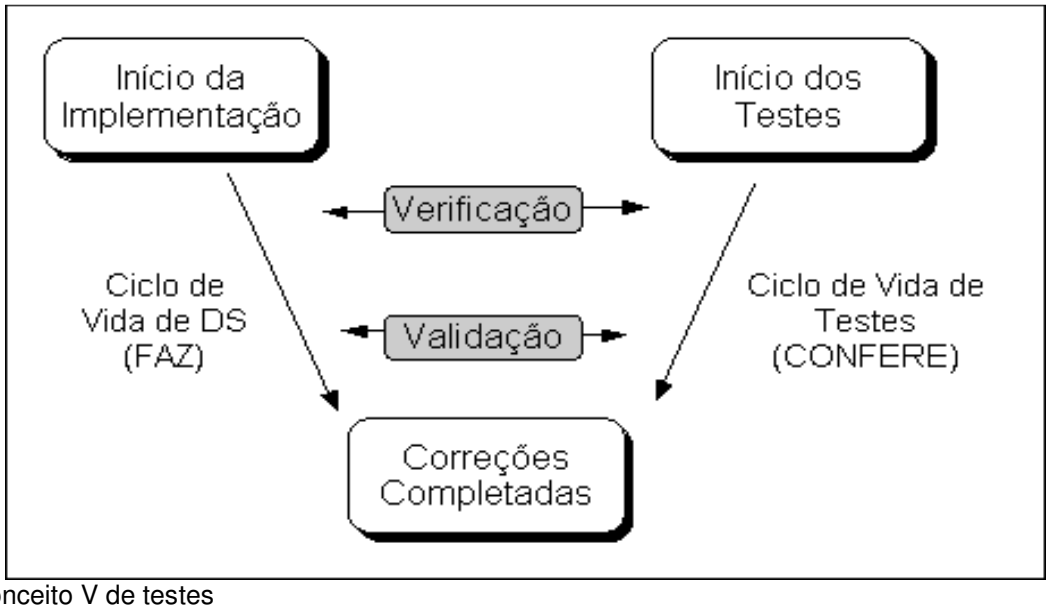


Figura 2: Conceito V de testes
Fonte: Souza, A. B. & RIOS, E. & Cristalli, R. S. & Moreira, T. *Base de Conhecimento em Teste de Software.* São Paulo : Martins, 2007.

Segundo os mesmos autores, o processo de teste deve-se basear no conceito “V” e definem:

> Os primeiros cinco passos usam a técnica de verificação como o principal meio para avaliar a correção dos produtos de desenvolvimento de *software*. Por outro lado, a técnica de validação serve para testar o *software* durante as atividades que vão desde a construção até a implantação. Os resultados de ambos devem ser registrados na documentação de testes. A validação e a verificação devem ser usadas para o desenvolvimento e a manutenção de *softwares*.

Abaixo, o modelo proposto, com os onze passos do processo:

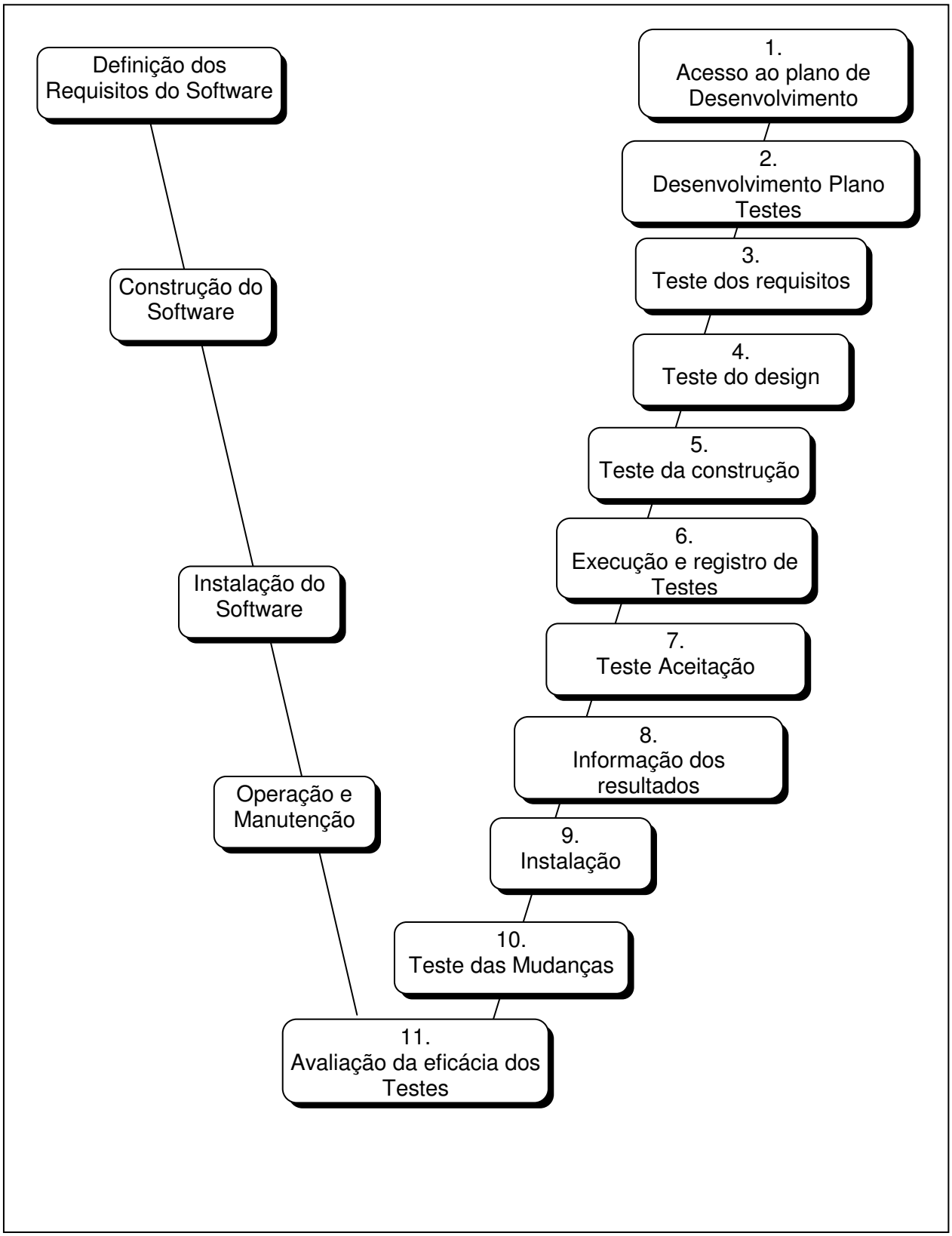


Figura 3: Processo de Teste
Fonte: Souza, A. B. & RIOS, E. & Cristalli, R. S. & Moreira, T. *Base de Conhecimento em Teste de Software.* São Paulo : Martins, 2007.

Os passos são:

1 **Acesso ao plano de desenvolvimento:** O documento de plano de desenvolvimento é validado e estando completamente correto, cria-se o plano de teste e define-se recursos de testes para o projeto.

2 **Desenvolvimento do plano de teste:** Etapa para a criação de todas as atividades de testes a serem executadas, bem como os recursos a serem utilizados. Nessa etapa o documento é definido.

3 **Teste dos requisitos de *software*:** Verificação dos requisitos. Requisitos inconsistentes podem levar a testes inconsistentes, portanto os erros devem ser apontados já nessa fase.

4 **Teste do desenho do software:** Verificação do desenho do *software*, através dos requisitos definidos e da arquitetura empregada.

5 **Teste da construção do software:** Verificação da eficácia da construção (*build*) do *software*.

6 **Execução dos testes:** Parte principal de todo o processo, pois é nesta etapa que os testes são executados e os erros ou falhas encontrados.

7 **Teste de aceitação:** Teste por parte dos clientes ou usuários, onde os requisitos devem ser verificados se estão atendidos, e se o escopo mínimo e aceitável de erros está coerente.

8 **Informação dos resultados do teste:** Etapa da divulgação dos resultados. Nesta etapa, a documentação dos erros encontrados e do sucesso obtido deve ser bem especificada. Costuma-se utilizar ferramentas específicas para esse acompanhamento, como ferramentas de gerenciamento de erros.

9 **Teste da Instalação do software:** Verifica-se se a instalação em um ambiente de produção está coerente com o esperado. Nesta fase, é usual que sejam executados testes de regressão e integração.

10 **Teste das Mudanças do *software*:** Verifica alterações ou correções aplicadas, após a implantação do *software*. Costuma-se aplicar testes de regressão nessa fase também.

11 **Avaliação das eficácias do *software*:** Processo gradual que pretende tomar como lições aprendidas as dificuldades encontradas, e melhorar as fases do processo. Nessa análise, é comum retornar aos passos anteriores e verificar as falhas e dificuldades encontradas.

Para esses processos, um ciclo de vida é definido com diversas etapas. Cada etapa corresponde a uma atividade, produtos ou documentos. Na primeira etapa, a de procedimentos iniciais, são definidos as linhas gerais do processo do teste e definido o plano de teste. Duas etapas acontecem em paralelo, até o fim do ciclo: planejamento e preparação.

Essas etapas suportam o processo, na definição das atividades e na elaboração das mesmas. As outras três etapas consomem cerca de 80% a 85% de todo o processo, que são a Especificação, Execução e Entrega, que, são responsáveis por elaboração e revisão de casos de teste e roteiros (Especificação), execução das técnicas de testes (Execução) e armazenamento dos resultados, documentação, divulgação de resultados e finalização do ciclo (Entrega). Esse ciclo de vida é chamado de 3P x 3E (Emerson Rios: 44 a 48).

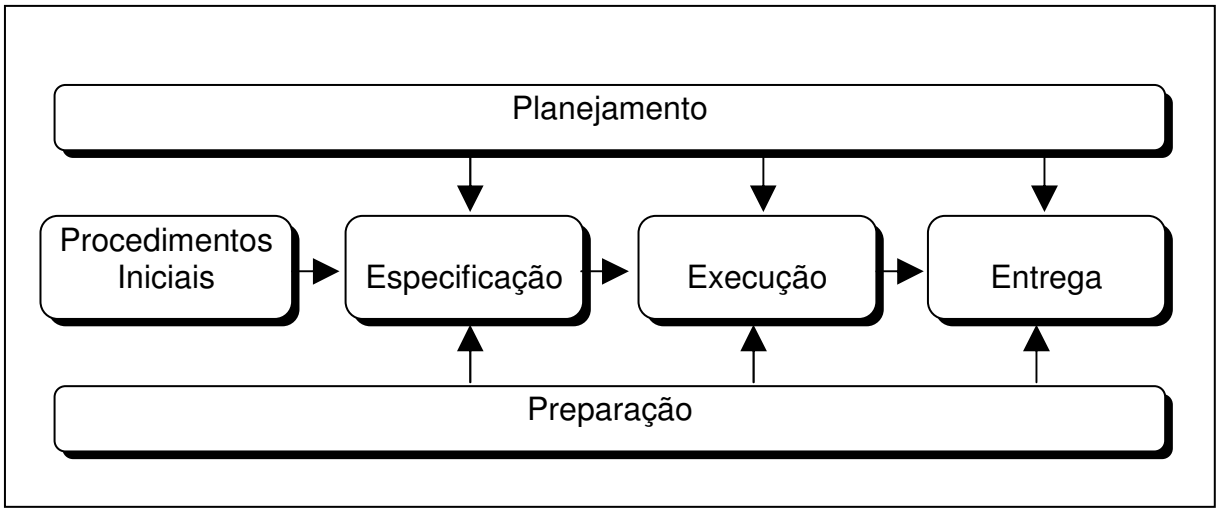


Figura 4: Ciclo de vida de Testes
Fonte: Souza, A. B. & RIOS, E. & Cristalli, R. S. & Moreira, T. *Base de Conhecimento em Teste de Software.* São Paulo : Martins, 2007.

1.2 TIPOS DE TESTES

Existem muitas publicações que abordam diferentes tipos de testes. Porém, os tipos de testes mais conhecidos, de acordo com Emerson Rios em seu livro Base de Conhecimento em teste de software, são:

1 **Unitário:** teste executado pelos desenvolvedores e implementadores do sistema. Este teste tem como foco a validação dos resultados das classes e métodos de um projeto. Por isso é executado pelo implementador, pois atua diretamente no código.

2 **Funcional:** normalmente executado pelo testador, mas também pode ser executado pelo implementador, o que não é comum. Tem por objetivo verificar se a funcionalidade está atuando da forma esperada. Pode ser executado desde a fase de requisitos, testando a documentação.

3 **Integração:** Este tipo de teste procura verificar a integração dos módulos. O teste funcional verifica a funcionalidade isoladamente, este teste pretende verificar e garantir que as funcionalidades funcionam de acordo, quando estão trabalhando juntas.

4 **Usabilidade**: Pode ser executada durante os testes funcionais ou de integração, em conjunto. Visa verificar se o sistema está prático quanto a interfaces e exibição de resultados.

5 **Estresse e desempenho:** Os testes de estresse e desempenho pretendem demonstrar até que ponto o sistema suporta uma certa quantidade de transações, por exemplo. Esse teste define se o requisito de desempenho definido está sendo atendido.

6 **Segurança:** Os testes de segurança tem por objetivo identificar quaisquer brechas que o sistema pode permitir no tratamento de informações. Dados sigilosos, privados, não podem ser de fácil acesso, tendo que ter um controle rígido e criterioso.

7 **Aceitação**: Os testes de aceitação são normalmente executados pelo cliente ou usuário final. É através dessa avaliação final que o produto poderá ser considerado apto para ser implantado em produção.

No momento da criação de um plano de testes, devem ser definidos quais os tipos de testes serão executados. Dessa forma, mantém-se o planejamento inicial previsto para o projeto e surpresas desagradáveis não são encontradas em momentos inoportunos.

1.3 DADOS DE ENTRADA PARA TESTES

Testes de *softwares* devem ser especificados baseados em critérios definidos na concepção do sistema, como já foi visto nos capítulos anteriores. Entretanto, os dados de entrada para os testes não costumam ser definidos neste momento. Esses dados, ou domínio de entrada, podem ser definidos no início dos testes. Um domínio de entrada pretende fornecer valores mínimos para testes, porém sem testar à exaustão um sistema, o que é impossível. José Carlos Maldonado, cita (Introdução ao teste de software, 269):

> Testar um programa com todos os seus possíveis valores de entrada ou executar todos os seus caminhos é idealmente desejável, mas impraticável. A maioria dos critérios de teste relaciona valores do domínio de entrada de um programa, agrupando-os em partições não necessariamente disjuntas. Em geral, os critérios de teste dividem o domínio de entrada do programa em subdomínios e requerem que pelo menos um ponto de cada subdomínio seja executado.

Dessa forma, deve-se definir os domínios de entrada para os testes, que garantirão a confiabilidade do sistema. Não é possível determinar que um sistema está isento de erros, pois é enorme a quantidade de situações que podem ocorrer no funcionamento, porém é possível determinar um nível de confiabilidade ao mesmo.

Existem erros conhecidos para essa teoria, pois restrições para os dados gerados são inerentes ao processo de testes, como a correção coincidente, caminho ausente, caminhos não executáveis e mutantes equivalentes.

Segundo José Carlos Maldonado (Introdução ao teste de software, 269):

> Correção coincidente: ocorre quando o programa em teste possui um defeito que é alcançado por um dado de teste, um estado de erro é produzido, mas coincidentemente um resultado correto é obtido. (...)
> Caminho ausente: corresponde a uma determinada funcionalidade requerida para o programa, mas que por engano não foi implementada, isto é, o caminho correspondente não existe no programa. Critérios estruturais de teste raramente auxiliam a determinação de caminhos ausentes, isto porque eles selecionam dados de teste baseados no código de programa e a especificação não é considerada. (...)
> Caminhos não-executáveis: Um elemento requerido por um dado critério estrutural é não-executável se não existir caminho executável que o cubra. Heurísticas tem sido utilizadas para determiná-los e evitar que esforço e tempo sejam gastos tentando gerar dados de teste para esses caminhos. (...)
> Mutantes equivalentes: A modificação feita no programa original não altera a função implementada, ou seja, o programa mutante implementa a mesma função que o programa original. Nesse caso, o mutante é dito equivalente. O problema de mutantes equivalentes é análogo ao de caminhos não-executáveis.

Assim, o autor pretende demonstrar que não somente os processos são imprescindíveis, mas conhecer os dados de entrada, resultam em teste mais eficazes e que encontrarão mais pontos falhos nos sistemas.

1.4 TESTES EM APLICAÇÕES *WEB*

Originalmente, as aplicações *web* baseavam-se em dados estáticos e eram simples de serem desenvolvidas e testadas. A arquitetura era básica: um servidor que armazenava as páginas estáticas em linguagem *HTML* e clientes com navegadores *web* que recebiam essas páginas *HTML* e exibiam ao usuário.

Entretanto, atualmente, as aplicações se tornaram complexas aplicações comerciais com arquiteturas distintas, que exigem mais do que um cliente-servidor. Aplicações *web* estão presentes em *smartphones* e outros dispositivos móveis.

Por se tornarem tão ágeis, compactas e móveis, as aplicações começaram a ser implementadas para diversos ramos, como instituições financeiras, comércio eletrônico, etc. Essas aplicações exigem um alto nível de validações pois devem ser seguras e estáveis. Silvia Regina, Maria Cláudia e Mario Jino (Introdução ao teste de software, 209), definem:

> A arquitetura e as tecnologias envolvidas em uma aplicação *web* mudaram drasticamente nos últimos anos. A configuração foi estendida do modelo cliente-servidor duas camadas para várias camadas e, embora muito semelhante às aplicações cliente-servidor tradicionais, o teste de aplicações *web* é um pouco mais complicado, pois existem diversos outros aspectos a considerar. O principal deles é que as aplicações *web* são dinâmicas e heterogêneas.
>
> A palavra heterogênea é comumente utilizada para designar as diversas maneiras utilizadas pelos componentes de *software* para se comunicarem e as diferentes e recentes tecnologias envolvidas. Além disso, deve-se considerar que os componentes encontram-se, na maioria das vezes, geograficamente distribuídos. Eles incluem *software* tradicional, programas em linguagem de *scripts*, *HTML*, base de dados, imagens gráficas e interfaces com o usuário complexas.

O teste de aplicações *web*, estruturadas das formas citadas acima, modelos são propostos para revelar os defeitos gerados pela manipulação de usuários distintos. Isso deve-se ao fato da interação de cada usuário ser diferente, pois as solicitações são distintas e aleatórias.

As aplicações *web* são multi camadas, como descrito pelos autores. Requisições *http* podem ser realizadas por navegadores *web*, *smartphones*, *hardwares* e devem receber as respostas em linguagens específicas. Testar essas aplicações é mais do que simplesmente navegar por páginas *HTML*. Deve-se levar em conta os diversos meios que essas chamadas podem ser executadas e tratar suas entrada e saídas.

Basicamente, as aplicações *web* baseadas em tecnologia Java utilizam o modelo MVC (*Model – View – Controler*), que é uma arquitetura que distingue as camadas de apresentação, objetos e processos. Dessa forma, a implementação se torna ágil, pois alterações na camada de visualização não influem nos dados ou nas regras que regem esses dados, como exemplo. Da mesma forma que os implementadores

devem ter o conhecimento nessa arquitetura, o testador também deve ter. Com isso, os testes em camada de apresentação podem ter resultados diferentes de testes realizados em camadas de objetos (gravação e recuperação em base de dados, por exemplo).

Assim, testes realizados em camada de apresentação podem ter resultados distintos de outras camadas, e as especificações de testes devem prever isso.

Contudo, os testes também tendem a ser mais ágeis, pois podem ser direcionados à camada implementada ou alterada. Em processos de desenvolvimento, é comum que a camada de apresentação seja designada à uma equipe de desenvolvimento e a camada de regras de negócio seja designada para outra equipe. Quando uma equipe termina uma camada, a equipe de testes não necessita que as outras camadas estejam prontas, pois os testes podem ser direcionados. Essa arquitetura também facilita os testes de regressão e testes de integração, pois podem ser tratados em separado.

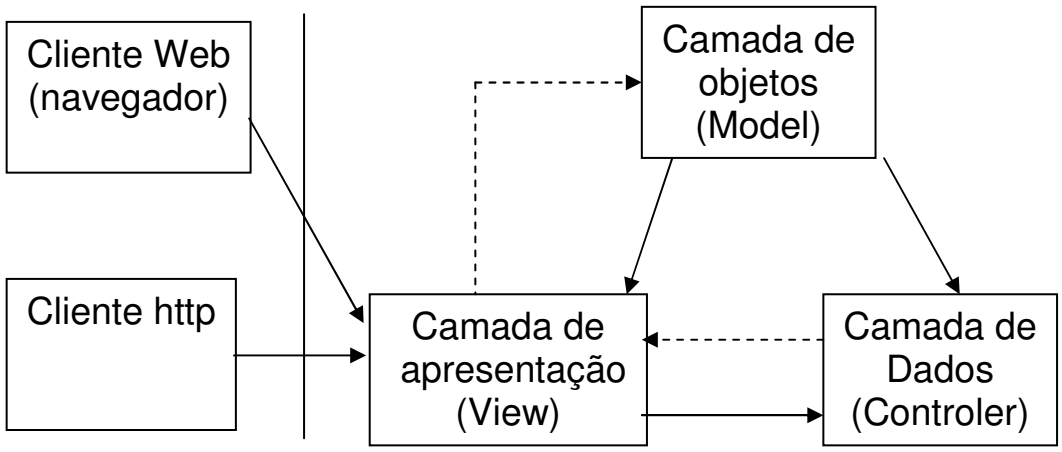


Figura 5: Requisições ao modelo MVC
Fonte: Gustavo Labbate Godoy, 2010

A figura 5 mostra o exemplo de acessos *web* e as camadas *Model – View – Controler*. A linha tracejada indica acesso indireto e a linha contínua indica acesso direto.

Conforme Silvia Regina, Maria Cláudia e Mario Jino (Introdução ao teste de software, 230):

> O teste funcional, estrutural e baseado em defeitos podem ser empregados com sucesso em aplicações *web* para revelar diversos tipos de defeitos, tanto na especificação como no código. Basicamente, os trabalhos descritos introduzem modelos para capturar diferentes aspectos de uma aplicação *web* visando a execução do teste baseado em fluxos de controle e de dados do teste de mutação.

Os testes de aplicações *web* são mais complexos do que se pode imaginar. Como visto, camadas intercaladas se alternam, sejam em execuções *web* propriamente ditas (requisições e respostas em navegadores) como por serviços ou processos automáticos.

2 AUTOMAÇÃO DE TESTES

Automação é uma palavra do latim que significa agir por si. Traduzindo, é algo que faz algo sozinho, sem interferência externa. Segundo o dicionário Houaiss, significa:

> "Sistema em que os processos operacionais em fábricas, estabelecimentos comerciais, hospitais, telecomunicações etc. são controlados e executados por meio de dispositivos mecânicos ou eletrônicos, substituindo o trabalho humano; automatização."

Automação é o conceito de utilizar dispositivos mecânicos ou eletrônicos, para a redução de recursos, como humanos, financeiros e equipamentos. As atividades desgastantes, repetitivas, podem ser continuamente executadas por um processo automático. Principalmente, visa substituir a mão-de-obra na atividade, não tornar a mão-de-obra ineficaz. Isso quer dizer que a automação não pretende tomar o lugar do ser humano e sim facilitar suas atividades.

Esse conceito, há muito tempo aplicado em indústrias, foi também aplicado no comércio. Hoje temos caixas eletrônicos, leitoras de cartões, impressoras fiscais, que agilizam os processos comerciais. Esse conceito pode ser usado também em atividades de informática.

Segundo Cem Kaner, um dos autores do livro **Lessons Learned in Software Testing:**

> (...)O propósito da automação de testes pode ser resumidamente descrito como a aplicação de estratégias e ferramentas tendo em vista a redução do envolvimento humano em atividades manuais repetitivas.

Como visto no capítulo anterior, as atividades de teste de *software* devem garantir que o produto que está sendo gerado é confiável e atende à todas as regras especificadas. Porém, as definições de um *software* tem grande mutabilidade, as atividades de testes devem ser repetidas exaustivamente. Além disso, as mudanças em códigos podem impactar drasticamente códigos de outros módulos, fazendo com que ocorra instabilidade em algo que anteriormente se encontrava estável.

Para se evitar esse tipo de problema, deve-se executar sempre os testes de regressão, que são desgastantes por serem repetitivos. Com isso, eleva-se o custo do recurso humano alocado nesta atividade e corre-se o risco do fator humano envolvido nisto. Por se tratar de atividade repetitiva, é da natureza humana que ocorra desatenção.

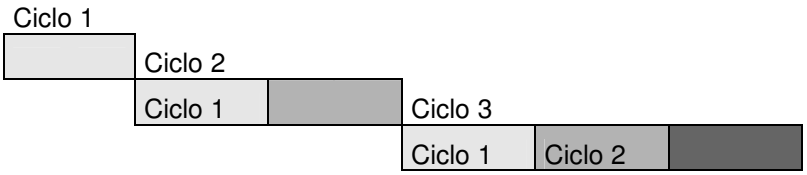
A automação dos testes atua nestas situações. Testes de regressão ou testes repetitivos podem ser automatizados, garantindo melhores resultados. Um teste de regressão pode ser programado para ser executado toda vez que uma nova versão de uma aplicação for liberada, após novas implementações. Com isso, verifica-se se o que estava funcionando anteriormente continua inalterado.

2.1 QUANDO AUTOMATIZAR

Um dos principais erros que se comete em um processo de implantação de automação é querer substituir todos os processos por processos automatizados. Como dito anteriormente, a automação não pretende substituir definitivamente o recurso humano, mas facilitar nos processos mais repetitivos. Um processo de automação não propõe melhorias, não resolve os problemas encontrados, não descobre novos erros além dos já esperados. Literalmente, faz o que lhe foi pedido e só. A automação não deve ser implantada para substituir as atividades de um mal testador, mas para que um bom testador tenha mais tempo livre para outras atividades.

Em conjunto de uma boa equipe, o processo de automação garante resultados rápidos e satisfatórios. As expectativas sobre o processo de automação devem ser em garantir um grande número de casos de testes executados continuamente, facilidade em repetição de testes, alocação adequada de recursos humanos, eliminação de erros humanos durante a execução dos testes, entre outros.

Um dos principais motivos para se criar a automação de testes são os testes de regressão em ciclos de teste.



Ciclo 1: Testes Funcionais

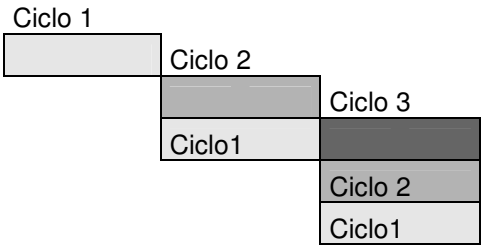
Ciclo 2: Testes Funcionais, Regressão Ciclo 1

Ciclo 3: Testes Funcionais, Regressão Ciclo 1, Regressão Ciclo 2

Figura 6: Ciclo de Desenvolvimento sem automação
Fonte: Gustavo Labbate Godoy, 2010.

Esse gráfico ilustra três ciclos de teste, onde os ciclos anteriores são repetidos no ciclo posterior. Os testes de regressão são realizados para garantir que o que já foi validado continua eficaz.

O mesmo gráfico, sob a perspectiva da automação, mostra um ganho considerável na execução dos testes, já que os testes de regressão podem ser aplicados em conjunto com os testes funcionais.



Ciclo 1: Testes Funcionais

Ciclo 2: Testes Funcionais, Automação

Ciclo 3: Testes Funcionais, Automação

Figura 7: Ciclo de Desenvolvimento com automação
Fonte: Gustavo Labbate Godoy. 2010.

2.1.1 TESTES AUTOMATIZADOS EM APLICAÇÕES *WEB*

Como visto no primeiro capítulo, o ambiente de aplicações *web* é complexo. Com a constante manutenção nessas aplicações, o teste das camadas se torna um esforço constante e dispendioso. Testes unitários, regressão, integração e outros

acabam se tornando gastos excessivos e em alguns casos, acabam não sendo aplicados.

As técnicas de automação de teste pretendem fornecer respostas ágeis e garantir que a aplicação mantenha um nível de qualidade dentro dos padrões e que essa etapa não transforme o processo de teste em um fator para atrasos para o processo de desenvolvimento.

Durante o desenvolvimento, após a conclusão de módulos, os testes unitários são executados e é garantido que o módulo está pronto para ser liberado. Quando há essa liberação, uma ferramenta de integração contínua constrói a aplicação, com as novas alterações nesse módulo e executa testes unitários (para o módulo alterado e para os demais módulos). Testes de integração e regressão são executados para verificar a aplicação como um todo e se o sistema não regrediu, passando a executar erros previamente corrigidos ou alterando a execução de funcionalidades já testadas e validadas. Novos dados de entrada são gerados para garantir uma diversidade de dados. Testes de desempenho também são executados neste momento, pois podem haver alterações em ambiente desde a última construção.

Com isso, a aplicação é testada não somente na camada de apresentação, mas nas camadas de regras de negócio e de tratamento dos dados, de forma ágil e que permita garantir que todo o conjunto da aplicação esteja aderente aos requisitos.

Outro ponto favorável à automação em aplicações *web*, é a facilidade com que é possível executar os testes em ambientes variados. É possível verificar o funcionamento de uma aplicação em diversos *browsers,* como *Internet Explorer* ou *Firefox*. Esses testes podem ser executados em paralelo, inclusive, para que em um mesmo ciclo, seja garantido a suportabilidade em diversas plataformas.

2.2 TIPOS DE TESTES AUTOMATIZADOS

Existem diferentes tipos de abordagem para automação. Podemos pensar em automatizar processos, seja ele executado manualmente ou pelo próprio sistema.

Eventualmente, esse processo de automação necessita de ferramentas específicas.

A maioria dessas ferramentas utiliza uma linguagem de *script* para que sua execução seja programada. É necessário conhecimento na geração desses *scripts* para que a ferramenta possa executar seus processos automatizados.

2.2.1 RECORD & PLAYBACK

Algumas ferramentas facilitam a criação dos *scripts* de testes através de um recurso conhecido como *record & playback* (algo como gravar e executar). A ferramenta grava as ações executadas pelo usuário e cria o script baseado nessa interação. Porém, como nem tudo é simples, exige manutenções por parte dos usuários. Esse tipo de automação costuma funcionar bem, porém costuma apresentar erros quando uma nova versão da aplicação com alterações é gerada ou quando erros são executados. Normalmente, estas ferramentas não incluem algum tipo de validação, ou seja, não é possível verificar se o teste foi satisfatório ou não.

Koen Wellens, em seu artigo para a revista *Testing Experience,* cita a diferença entre o teste *record & playback* sem manutenção e outro com validações e alterações do usuário. Basicamente, os testes são para um sistema de reserva de passagens aéreas. Pelas duas imagens, é possível notar a diferença e complexidade entre os testes. No primeiro caso, a execução de mais de um teste é repetitiva, sendo necessário muitas horas para manutenção (o que tornaria a automação inviável). O segundo caso mostra que, com um *script* bem criado, é necessário apenas alterar a massa de dados que comporá a execução dos testes.

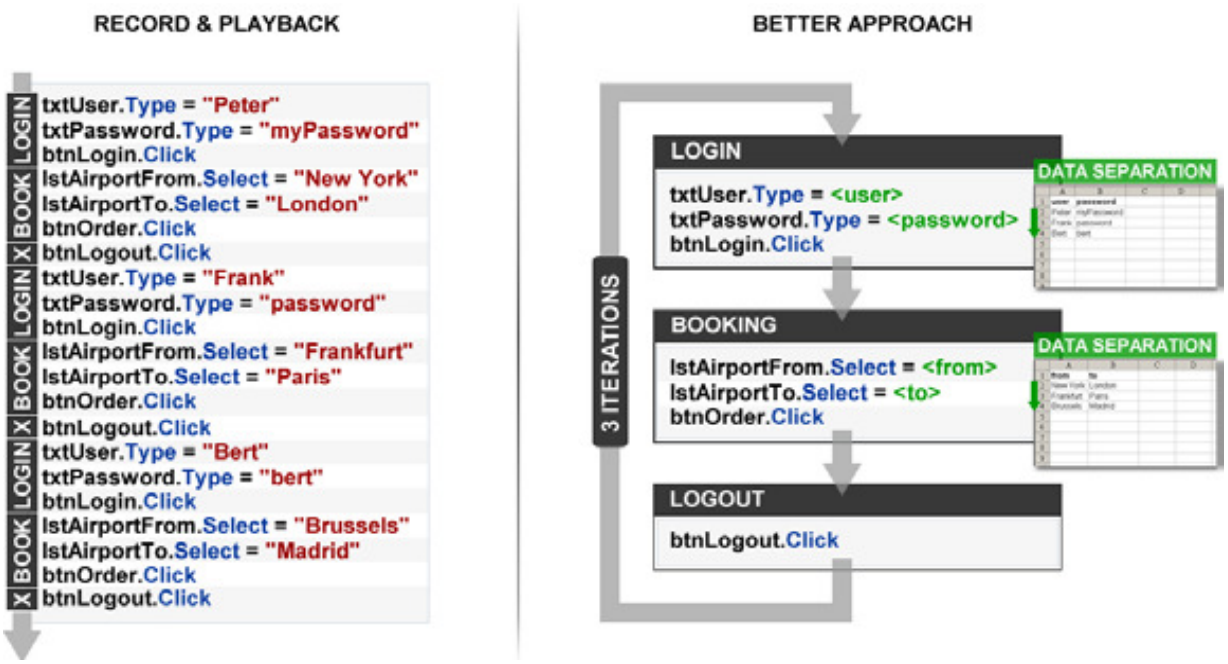


Figura 8: Record & Playback
Fonte: Wellens, K; The Record And Playback Fairytale. **Testing Experience Magazine.** P22, dez 2008.

Dessa forma, é necessário que seja analisado a correta utilização da ferramenta para captura e execução dos testes, para que sua manutenção não seja superior ao tempo gasto em testes manuais, o que fará com que a automação se torne desnecessária.

Duas ferramentas muito conhecidas para testes de *Record & Playback* em ambiente de aplicação web são o Selenium IDE (http://seleniumhq.org/) e o Watir (http://watir.com/).

O Selenium IDE é uma ferramenta criada basicamente em *HTML* e *Javascript*, que permite que qualquer aplicação sendo executada em *browsers* que suportem essas linguagens possam ser automatizadas. O Selenium IDE permite que qualquer *script* gravado pelo recurso *Record & Playback* possa ser exportado para várias linguagens (Java, C, Python, entre outras), para que extensões sejam criadas. Todo projeto Selenium usa a licença de uso do projeto Apache 2.0 (http://www.apache.org/licenses/LICENSE-2.0) e pode ser utilizado em *Unix/Linux* e *Windows*.

O Watir (pronuncia-se '*water*'), que usa licença BSD, é uma ferramenta criada utilizando a linguagem *Ruby*, e pode ser utilizado em *Windows* e *Unix/Linux*. Os

scripts criados em seu método de *Record & Playback* podem ser extendidos utilizando essa linguagem, o Ruby.

Uma ferramenta para *Record & Playback* para aplicações *desktop*, sob licença *freeware* (http://www.autoitscript.com/autoit3/docs/license.htm), é o *Autoit* (http://www.autoitscript.com/autoit3/index.shtml). Com ela, é possível automatizar testes para aplicações *desktop*. Sua linguagem própria é simples, baseada em *BASIC*, e permite que qualquer aplicação executada em um ambiente gráfico, em sistemas operacionais *Windows,* possa ser automatizada.

2.2.2 TESTES UNITÁRIOS

Durante o desenvolvimento, como visto anteriormente, é necessário que os desenvolvedores garantam sua qualidade executando os testes unitários. É comum que utilizem ferramentas para automatizar esses testes, pois o tempo dispendido com testes não pode ser superior ao tempo dispendido em desenvolvimento. Dessa forma, criam-se código específicos para os testes, que podem ser executados quando necessário. Normalmente, esses tipos de teste, por serem criados e executados pelo desenvolvedor, utilizam a linguagem utilizada no desenvolvimento da aplicação.

Este tipo de teste consiste em validar métodos ou funções, passando os valores necessários para que se obtenham resultados satisfatórios e insatisfatórios. Consiste basicamente em executar o método ou função com uma grande variedade de dados, verificando então se o método ou função retorna o que deveria retornar ou se trata corretamente os valores incorretos.

Com o intuito de automatizar estes testes, a ferramenta Junit (http://www.junit.org), podendo ser utilizado em *Windows* e *Unix/Linux*, sob a licença CPL (*Common Public License* - http://junit.sourceforge.net/cpl-v10.html) para a linguagem JAVA, é largamente utilizada. Seu conceito baseia-se em passar os dados aos métodos e validar, com exibição dos resultados. Com isso, pode-se garantir que, com qualquer alteração no código, as unidades mantiveram-se estáveis.

```
public class Math

{
    static public int add(int a, int b)
    {
     return a + b;
    }
}
```

Listagem 1: método de cálculo
Fonte: Gustavo Labbate Godoy, 2010

```
public class TestMath extends TestCase {

  public void testAdd()
  {
        int num1 = 3;
        int num2 = 2;
        int total = 5;
        int sum = Math.add(num1, num2);
        assertEquals(sum, total);
  }
}
```

Listagem 2: Método de teste
Fonte: Gustavo Labbate godoy, 2010

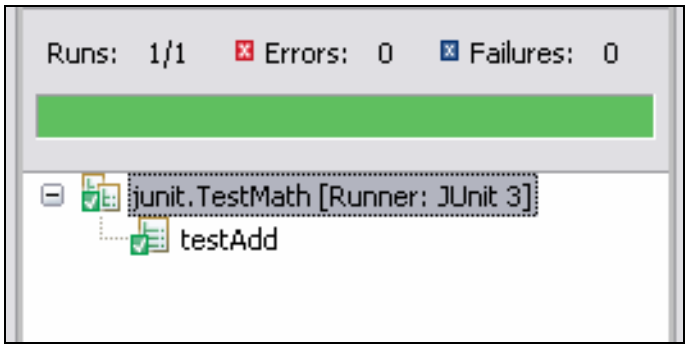


Figura 9: Resultado do teste executado
Fonte: Gustavo Labbate godoy, 2010

2.2.3 GERAÇÃO DE MASSAS DE DADOS

Para a criação de dados para testes, é necessário uma ferramenta que possua critérios específicos e seja de fácil uso. Criar uma base de dados e popular valores diversos, para que os profissionais de desenvolvimento e testes tenham consistência em suas atividades é uma tarefa árdua. Uma ferramenta que atende à esse requisito é o *Project R* (http://www.r-project.org/), sob licença GNU (http://www.gnu.org/philosophy/free-sw.html). Pode ser executado em *Unix/Linux*, *Windows* e *MAC OS*.

Basicamente, a ferramenta se propõe a criar dados, baseados em modelos estatísticos e gerar valores. O resultado de seu processamento é a criação de arquivos de texto para alimentação de bases de dados. A ferramenta também possui funcionalidade própria para geração de gráficos e relatórios dos dados gerados ou processados.

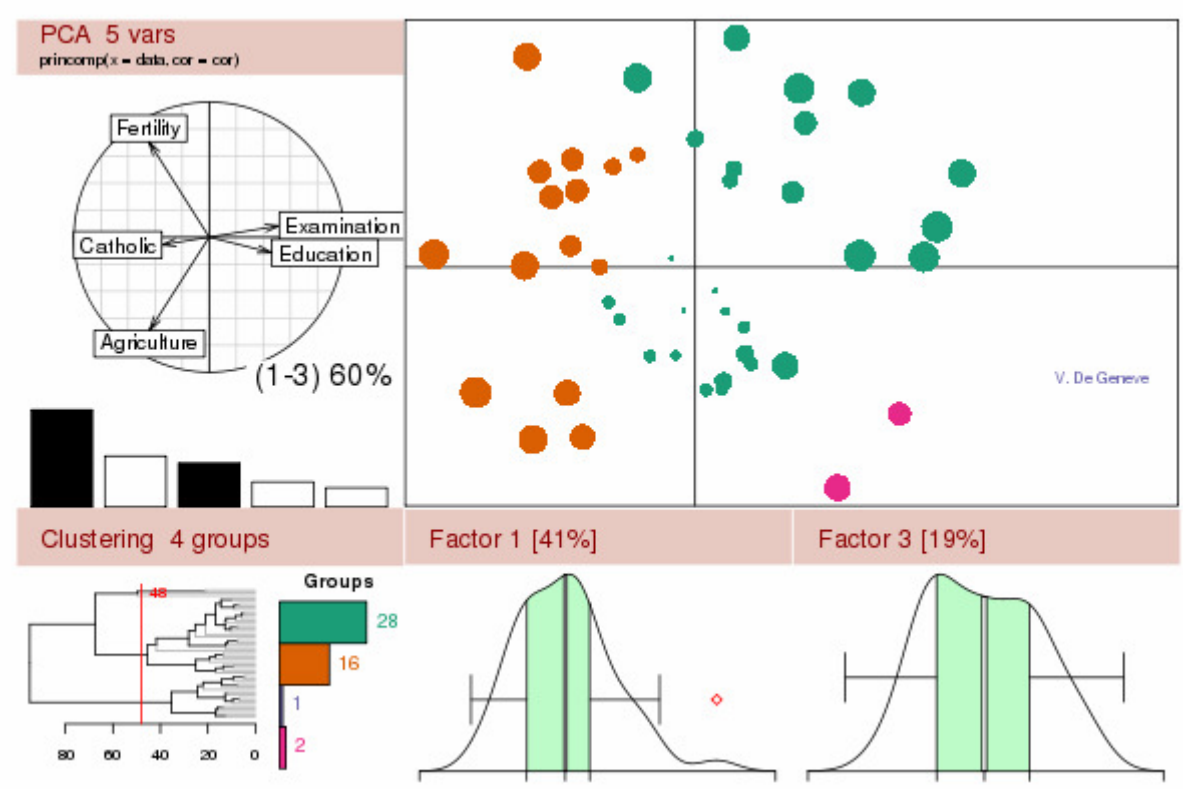


Figura 10: Tela de resultados da ferramenta Project R.
Fonte: http://www.r-project.org/

2.2.4 TESTES DE DESEMPENHO

Um ambiente de testes é diferente de um ambiente de produção. No ambiente de produção, as transações em uma aplicação se multiplicam de acordo com o número de usuários que a acessam. Em um ambiente de testes, há somente o acessos das pessoas envolvidas no projeto. Com isso, não se pode garantir a suficiência da aplicação em atender à todas as solicitações que se façam acima de uma determinada quantidade de acessos.

Além disso, recursos da máquina não podem ser garantidas, pois em aplicações com baixo acesso, como nos ambientes de testes, não é possível forçar situações em que a aplicação trabalhe exaustivamente.

Uma ferramenta gratuita, que auxilia nessas análises de desempenho é o *Jmeter* (http://jakarta.apache.org/jmeter/) , que usa a licença de uso do projeto Apache 2.0 (http://www.apache.org/licenses/LICENSE-2.0) e pode ser utilizado em *Unix/Linux* e *Windows*. Essa ferramenta cria situações com variados acessos e requisições a aplicações, podendo mensurar como está o desempenho da aplicação.

Pode ser utilizada para simular carga extremas em servidores, redes ou testar objetos ou analisar o desempenho como um todo, entre diferentes tipos de cargas, e também pode fazer teste de cargas concorrentes, que seriam vários processos sendo executados ao mesmo momento. A ferramenta disponibiliza relatórios gráficos com a análise do desempenho.

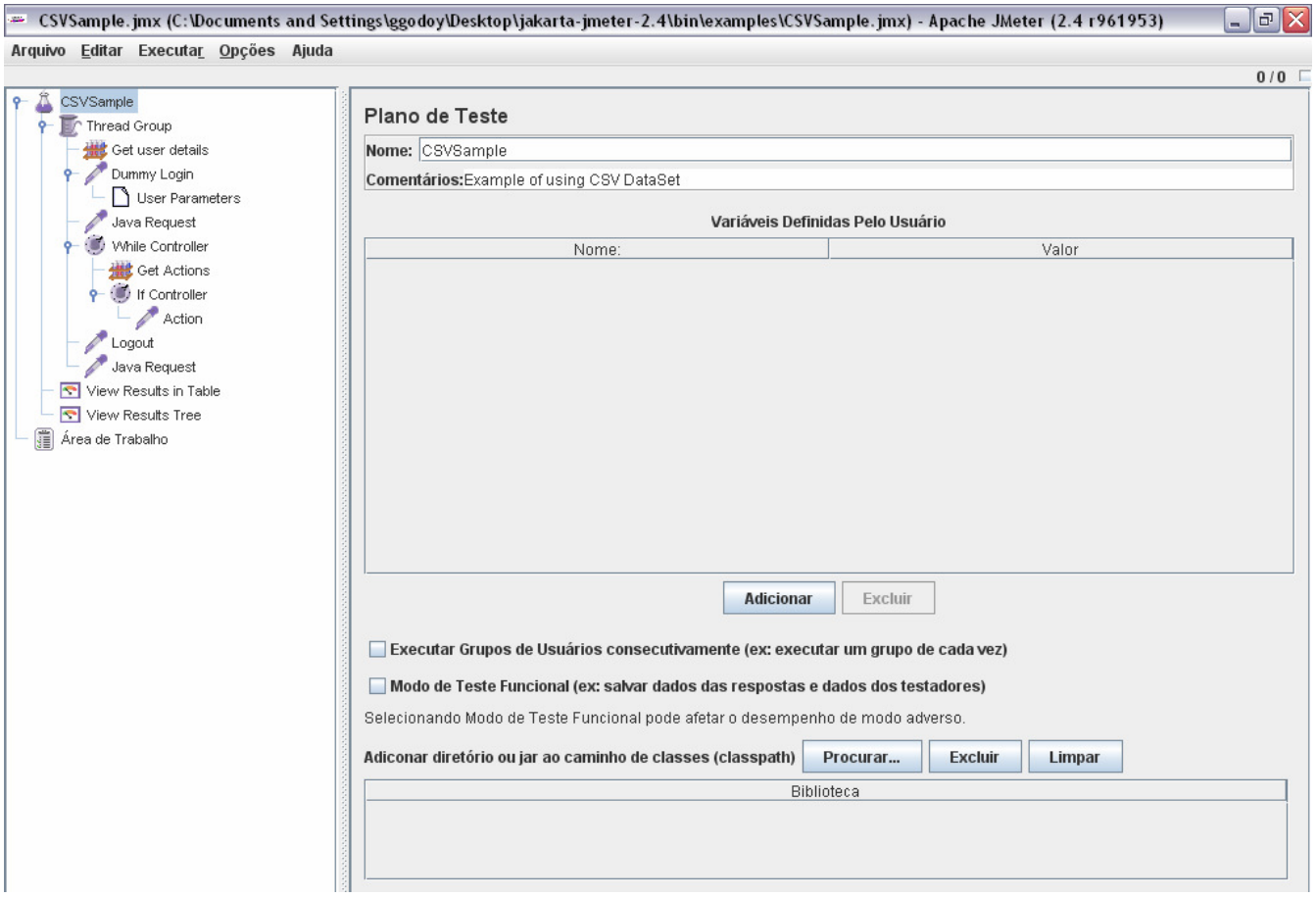


Figura 11: Testes de desempenho com Jmeter
Fonte: Gustavo Labbate godoy, 2010

2.2.5 INTEGRAÇÃO CONTÍNUA

A fase de construção do *software*, previsto no plano de desenvolvimento, costuma se integrar à atividades de teste. Nessa etapa, todo o código é compilado e o *software* se torna um programa executável. Nesse momento, os testes de integração e regressão devem ser aplicados, pois tem por objetivo garantir que o sistema esteja fazendo o que era esperado, e que novas implementações não causaram a regressão, fazendo com que erros conhecidos e tratados voltem a aparecer.

O processo de integração contínua tem por objetivo realizar os *builds* dos *softwares* e, em seguida, realizar processos de testes e verificações. A grande vantagem é que, por ser tudo automatizado, esse processo pode ser executado a qualquer momento, sem a necessidade de interação humana, pois o projeto pode ser configurado para ser executado em determinados momentos.

Existem várias ferramentas para a integração contínua no mercado. Uma delas, muito utilizada e com várias informações em fóruns específicos e que conta com a vantagem de ser gratuita, é o Hudson CI (http://www.hudson-ci.org/), que pode ser utilizado em *Unix/Linux* e *Windows*. Sua licença é baseada na licença MIT (*Massachusetts Institute of Technology*) para *softwares* livres.

O Hudson CI tem como principais características a compilação automática após arquivos serem armazenados em um programa *SCM (Source Code Management*), facilidade em construção (automática, manual), publicação de resultados de testes e notificação dos resultados por email.

Figura 12: Tela do Hudson CI
Fonte: Gustavo Labbate Godoy, 2010

O Hudson permite que, após a construção de uma aplicação, comandos sejam executados. Esses comandos, sejam em ambiente *Linux* ou *Windows*, podem conter os *scripts* de testes em Selenium, os testes unitários em Junit, os testes de desempenho em Jmeter, além de uma infinidade de outras opções, disponíveis em forma de *plugins*.

Todo o projeto pode ser reduzido quando uma ferramenta de integração contínua é bem aplicada.

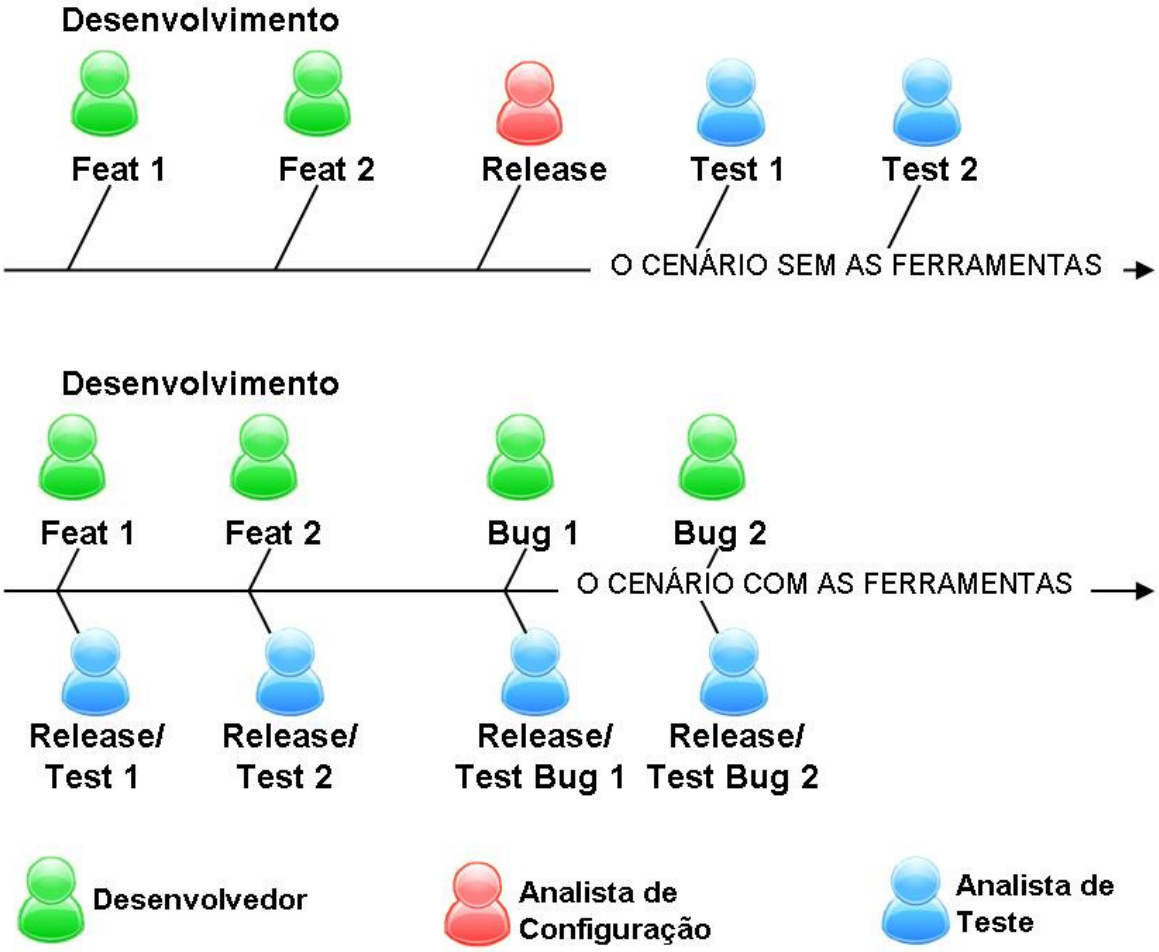


Figura 13: Processo de Desenvolvimento sem e com Integração contínua.
Fonte: Gustavo Labbate Godoy, 2010

Pela figura 13, temos o exemplo de como é um processo de desenvolvimento antes e depois da aplicação de uma ferramenta de integração contínua. No primeiro ciclo de vida, sem a aplicação da ferramenta, notamos os processos de desenvolvimento e testes bem separados. Primeiro, os desenvolvedores escrevem o código e realizam seus testes unitários. Quando terminam, há a ação de um profissional responsável por construir a aplicação (área de Gerenciamento de Configuração), que disponibiliza a aplicação para os testes. Todo o processo de teste é executado após essa etapa. Nesse processo todo, uma equipe depende da outra para realizar suas atividades, e ficam ociosas enquanto isso.

O segundo ciclo de vida, com a aplicação da integração contínua, os processos de teste se integram com o desenvolvimento. Após o desenvolvimento de

cada desenvolvedor e seus testes unitários, a aplicação pode ser construída sem a necessidade de um profissional de gestão de configuração, o que torna o processo mais independente. Enquanto os testes estão sendo executados, o desenvolvedor já está trabalhando em outro módulo do sistema. No mesmo espaço de tempo do ciclo, já foi possível desenvolver, testar e realizar as correções e testes de erros encontrados.

3 CONSIDERAÇÕES FINAIS

> “Uma paixão forte por qualquer objeto assegurará o sucesso, porque o desejo pelo objetivo mostrará os meios.”
>
> (William Hazlitt)

Todos esses conceitos, teóricos e práticos, influem em um ambiente profissional constantemente reciclável e mutável. A tecnologia, que progride diariamente, força os profissionais a se manterem atualizados e o aprendizado ou obtenção de novos conhecimentos se torna uma tarefa constante.

As técnicas aqui apresentadas, servem para que uma base histórica desse conhecimento seja preservada, tanto para os implementadores quanto para os testadores. Conhecimento dos erros e acertos influencia nos erros e acertos futuros, quando são conhecidos e aprendidos. As ferramentas atendem à esses propósitos.

Com o Hudson, é possível integrar o uso delas e armazenar as informações, fornecendo dados para novos projetos. Notificações também são enviadas por ele, o que torna o acompanhamento e tomada de ações mais práticas e pontuais.

Tornar a equipe de teste mais íntegra e forte fará com que os projetos sejam concluídos sem esforços e recursos desnecessários, resultando em projetos eficazes.

REFERÊNCIAS BIBLIOGRÁFICAS

BASE DE CONHECIMENTO PARA CERTIFICAÇÃO EM TESTE. Foundation Level Syllabus. Versão 2007 br. BSTQB.

Bastos, A.; RIOS, E; Cristalli, R.; Moreira T. **BASE DE CONHECIMENTO EM TESTE DE SOFTWARE**. Ed. Martins Fontes. 2.ed.

CERTIFICAÇÃO BRASILEIRA DE TESTE DE SOFTWARE. Referência Complementar – CBTS V1.1 – 18/04/2007 – Associação Latino Americana de Teste de Software (http://www.alats.org.br/LinkClick.aspx?link=RC-CBTS+maio2009.pdf&mid=1055)

Delamaro, M. E.; Maldonado, J. C.; Jino, M. **INTRODUÇÃO AO TESTE DE SOFTWARE** - Rio de Janeiro : Elsevier, 2007

DUSTIN, Elfriede; RASHKA, Jeff; PAUL, John. **Automated Software Testing: Introduction, Management, and Performance.** Canadá. Addison-Wesley Professional. 1999

Feitosa, C. Araújo, V. **Teste de Software – De tendência à realidade no Mercado de TI**. Disponível em:
http://www.rafrom.com.br/portal/index.php?option=com_content&view=article&id=270:teste-de-software--de-tendencia-a-realidade-no-mercado-de-ti&catid=55:informatica&Itemid=62
Acesso em: 25 de agosto de 2010. 13h46

KANER, Cem; BACH, James; PETTICHORD, Bret. **Lessons Learned in Software Testing**. United States Of America. John Wiley & Sons, Inc, new York, 2002

Neto, A. C. Dias. **Engenharia de Software - Introdução a Teste de Software**. Disponível em: http://www.devmedia.com.br/articles/viewcomp.asp?comp=8035 Acessado em 14/11/2010 23:53

Wellens, K; The Record And Playback Fairytale. **Testing Experience Magazine.** P22, dez 2008.