Integração Contínua no Desenvolvimento de Software com a Linguagem ABAP

Leonardo Alexandre Pletsch[1], Josiane Brietzke Porto[1]

[1]Unidade Acadêmica de Pesquisa e Pós-Graduação – Universidade do Vale do Rio dos Sinos (UNISINOS) - Av. Unisinos, 950 – 93.022-750 – São Leopoldo – RS – Brasil

leonardoalexandrep@yahoo.com, josibrietzke@unisinos.br

Resumo. *O processo de Integração Contínua (IC) permite avanços nos produtos de software no que tange à qualidade, disponibilidade e entrega. A IC é normalmente associada às linguagens mais comuns de mercado como Java e C#, porém seus benefícios não estão necessariamente restritos à estas linguagens. Esta pesquisa-ação mostra a implantação de IC num contexto de desenvolvimento de software na linguagem de programação Advanced Business Application Program (ABAP), num produto desenvolvido para o mercado brasileiro por uma empresa de Enterprise Resource Planning (ERP). Os resultados mostraram que mesmo com a IC não implementada em sua totalidade, resultados positivos consistentes puderam ser alcançados, tais como sistema de feedback mais eficiente e uma redução no percentual de chamados de clientes causados por bugs.*

Abstract. *The Continuous Integration (CI) process allows advances to software products towards quality, availability and delivery. The CI is normally associated to the most commons programming languages in the market like Java and C#, however its benefits are not restricted to these languages. This research-action shows a CI process implementation in Advanced Business Application Program (ABAP) programming language, in a product developed for the Brazilian market by an Enterprise Resource Planning (ERP) company. Despite the fact of process could not be implemented completely, consistent benefits were achieved, such a most efficient feedback system and a decrease in rate of customer incidents caused by bugs.*

1. Introdução

Considerando que qualidade interna de um produto de software influencia a sua qualidade externa e a sua qualidade em uso, uma maneira de ajudar na qualidade interna de um software consiste na implementação de um processo de Integração Contínua (IC), dado que é possível acrescentar várias medições de qualidade interna do software neste processo, como mostrado em estudo anterior de Moreira (2010).

Segundo Fowler (2006), a IC é uma prática de desenvolvimento de software em que membros de uma equipe integram seu trabalho frequentemente ou pelo menos múltiplas integrações ao dia. Cada integração é verificada por um *build* automático, para detectar erros de integração o mais rápido possível. Esta abordagem pode levar a uma

significativa redução de problemas de integração e um desenvolvimento coeso de software mais rápido [Fowler 2006].

A automação e o f*eedback* rápido permitem que os problemas sejam identificados rapidamente. Nesse contexto, *Jenkins* [ISOTOPE11 2017] corresponde a uma reconhecida aplicação de IC e Entrega Contínua. Uma das vantagens na sua utilização consiste na quantidade de *plug-ins* disponíveis, que englobam várias linguagens de programação e tecnologias [Mota 2015].

A *Advanced Business Application Program* (ABAP), originalmente *Allgemeiner Berichts-Aufbereitungs-Prozessor*, em alemão ou em português, Processador Genérico de Criação de Relatórios é uma linguagem de programação de alto nível, criada pela companhia de software alemã SAP [SAP 2017].

Nesse contexto, o produto UEG (nome fictício) foi desenvolvido com a linguagem ABAP desde o seu início, em 2013. O uso de testes automáticos, especialmente testes unitários esteve sempre presente na rotina da equipe de desenvolvimento. Todavia, com o aumento do número de funcionalidades e consequentemente, o aumento no tamanho do código fonte, o escopo de monitoramento destes testes aumentou na mesma proporção. Dado que as ferramentas de *feedback* de sistema para a linguagem ABAP atuais não agregam o resultado de um *landscape*, o *feedback* do sistema é pouco eficiente e os desenvolvedores deixam de agir em função dos resultados dos testes.

Nesse cenário de desenvolvimento, com o advento de *plugins* do *Jenkins* para a linguagem ABAP, a centralização dos resultados dos testes automáticos se tornou possível. Dado que a IC é a melhor ferramenta para prover *feedback* automático, direcionado e em tempo real [Duvall, Matyas and Glover 2007], a utilização desta técnica se tornou evidente para o desenvolvimento do produto UEG. Uma vez endereçado o problema inicial de *feedback*, um grande número de possibilidades de melhorias no desenvolvimento do produto se abriu. Um exemplo disso é a coleta de métricas específicas, aliada ao processo de IC, garantindo a qualidade dos produtos técnicos e auxiliando em decisões táticas durante o projeto [Pressmann 2016].

Este trabalho propõe a implementação de um processo eficiente de IC em desenvolvimentos na linguagem ABAP, levando em consideração o contexto do produto UEG. Pode ser representado pela seguinte questão de pesquisa: Como implementar IC no desenvolvimento de software com a linguagem ABAP?

Cabe ressaltar que uma das práticas da IC é a coleta de métricas a cada *build*, entretanto, neste trabalho a única métrica coletada corresponde a de cobertura de linhas de comando por testes unitários. Outras métricas, tais como complexidade ciclomática [McCabe 1994], aderência à *naming convention* [Smit, Gergel, Hoover and Stroulia 2011] e estabilidade [Martin 2002] não são abordadas no escopo desse trabalho, dado que no momento o único *plugin* disponível para ABAP é o Sonar e seu custo por ano é de £9.500,00, por instância [ABAP Sonar 2017].

Esse trabalho pode contribuir para aumentar a qualidade do produto UEG da organização objeto desse estudo, permitindo um ciclo de *feedback*, baseado em testes automáticos aos desenvolvedores a cada entrega de código novo. Desta maneira,

entende-se que a utilização da IC pode diminuir o número de chamados de clientes relacionados aos *bugs* no produto.

A contribuição principal deste artigo pode servir de referência para a implementação de um processo de IC em outras aplicações e ambientes de desenvolvimento com a linguagem ABAP, tendo em vista que até o momento existe somente breves esboços sobre o tema [Held 2015][Krawczyk 2013]. Dado que este trabalho é o primeiro que traça um paralelo utilizando dados quantitativos sobre os ganhos na utilização de IC na linguagem ABAP, em um produto real pode-se derivar o caráter inovador da solução. Seus resultados tornam-se relevantes por verificar e demonstrar na prática, os benefícios teóricos propostos pela IC, num contexto prático e real, que adota essa abordagem de desenvolvimento de software.

Além dessa introdução, o artigo está organizado em mais quatro seções. Na seção 2, o referencial teórico aborda a linguagem ABAP, os principais termos da IC e ferramentas, que suportam esta metodologia. Na seção 3, os aspectos metodológicos adotados são apresentados e, na seção 4, o desenvolvimento da solução é descrito em detalhes, além dos resultados obtidos. Por fim, a seção 5 aponta considerações finais.

2. Referencial Teórico

2.1. Integração Contínua (IC)

Neste trabalho acredita-se que a IC é uma maneira de aumentar a qualidade do processo, que resulta imediatamente nos atributos de qualidade interna. Esta afirmação se baseia no fato da IC executar regularmente todos os testes automáticos do produto em desenvolvimento, evidenciando problemas de qualidade interna (via testes unitários), qualidade externa (via testes funcionais). Desta forma, eventuais problemas de qualidade em uso podem ser evitados, proporcionando um ganho na qualidade em uso do software produzido.

No que tange características de IC, cabe uma descrição de termos relacionados:

- *Build*: consiste em um ou múltiplos *scripts* usados, para compilar, testar, inspecionar e distribuir software [Duvall, Matyas and Glover 2007]. Gerar um *build* é um procedimento de construir uma aplicação;
- *Job*: refere-se as tarefas executáveis, monitoradas e/ou controladas pelo *Jenkins*, por exemplo. Um *build* pode ser o produto da execução de um ou vários *jobs* [Jenkins 2017];
- *Commit*: ação de gravar alterações em um repositório [Git-scm 2017];
- *Deploy*: obter software funcional, pronto para implantação (*deployment*) a qualquer momento [Sepalla 2010], inclusive, com instalação de software automaticamente, a partir da IC, por meio de alguns passos específicos [Duvall 2007];
- Máquina ou Servidor de IC: uma máquina separada, que reproduz o ambiente produtivo. Se possível, exatamente o sistema operacional, banco de dados e versões de bibliotecas como planejado em ambiente de produção [Sepalla 2010].

De acordo com Fowler (2006), as práticas que compõe a IC são: *(i)* manter uma fonte única de repositório de código fonte; *(ii)* Automatizar o *build*; *(iii)* Fazer com que

o *build* seja auto-testável; *(iv)* Todos da equipe devem fazer *commit* no repositório principal todos os dias; *(v)* Todo o *commit* deve iniciar um *build* repositório principal na máquina de integração; *(vi)* Corrigir *builds* quebrados imediatamente; *(vii)* Manter o *build* rápido; *(viii)* Testar em um clone de um ambiente produtivo; *(ix)* Fazer com que qualquer pessoa possa ter o último executável facilmente; *(x)* Todos possam ver o que está acontecendo; *(xi)* Automatizar o *deploy*.

Como se pode ver a partir dessas práticas, os benefícios da IC são consideráveis num ambiente de desenvolvimento de software. *Jenkins* é uma ferramenta de código aberto para IC e entrega contínua [Jenkins 2017], sendo considerada uma das principais ferramentas, que apoiam a execução deste processo. Baseia-se num servidor *Apache Tomcat* e pode rodar projetos do *Apache Maven*, assim como *Shell Scripts* e comandos de *Batch* do *Windows*. Além de possuir diversos *plug-ins*, que permitem integração com diversas ferramentas de gerenciamento de código fonte. Possui também a possibilidade de integração com várias linguagens de programação.

2.2. *NetWeaver Application Server* ABAP

Desde de 2004, a linguagem ABAP integra a plataforma *SAP NetWeaver* [NetWeaver 2004]. Como pode ser visto na Figura 1, adaptada de SAP Help (2017), uma aplicação ABAP roda em um servidor de aplicação específico, dentro desta plataforma.

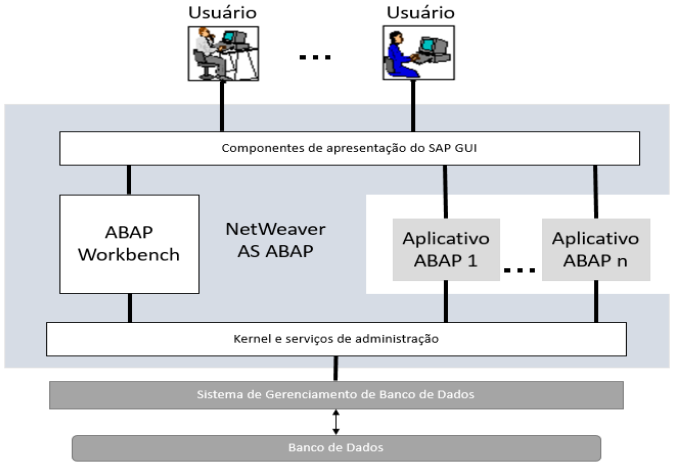


Figura 1. Visão lógica dos componentes da plataforma *NetWeaver*

Conforme a Figura 1, o *Kernel* e os *Administration Services* são o ambiente de *runtime* para todas as aplicações ABAP. Os mesmos são independentes de *hardware*, sistema operacional e banco de dados. O *runtime* ABAP é escrito, principalmente, em C e em C++. Entretanto, algumas partes de baixo nível também são escritas em ABAP.

Diferentemente de outras linguagens de programação, no ABAP o código fonte das aplicações é armazenado no banco de dados. Desta forma, o banco de dados de um sistema *NW AS ABAP* divide-se em dois grandes componentes lógicos primordiais, os dados gerados relativos à aplicação e o repositório [SAP Help 2017].

A parte central do repositório é o *ABAP Dictionary*, que contém descrições das estruturas e das relações entre os dados. Estas descrições são usadas pelo *NW AS ABAP* para interpretar e gerar objetos de aplicação no ambiente de *runtime*, tais como

programas e telas [SAP Help 2017]. Estes objetos são denominados objetos de repositório.

O componente *ABAP Workbench* é um ambiente de desenvolvimento completo para aplicações na linguagem ABAP. Com ela é possível criar, editar, testar e organizar aplicativos. O *ABAP Workbench* está totalmente integrado com o *NW AS ABAP* e, como as outras aplicações em ABAP, também é escrito nesta linguagem [ABAP 2017] [SAP Help 2017].

No que tange ao *landscape* dos sistemas *NW AS,* para proteger os sistemas produtivos de alterações não desejadas ou com falhas é recomendado que o sistema de desenvolvimento seja separado do sistema de produção. Em sistemas *NW AS*, é recomendado definir políticas e procedimentos para mudanças e transportes para o sistema de produção.

Neste contexto é recomendado a utilização do tipo *three-tier*. Isto significa ter um sistema separado para desenvolvimento, um sistema separado para qualidade (testes) e outro para a produção. Estes três sistemas compartilham um diretório comum de transporte. Com esta configuração de ambiente é possível fazer mudanças e testá-las, sem interferir nas operações produtivas [SAP Landscape 2017].

Nos sistemas ABAP, *change requests* e seus *tasks* constituintes provêm o mecanismo necessário para gravar os objetos que tenham sido alterados. Quando as mudanças de *customizing objects* ou *development objects* são realizadas, os objetos alterados são gravados em um *task*. Cada usuário do sistema é associado a uma *task*, que é uma simples lista de objetos alterados por aquele usuário. *Tasks* são agrupadas em *change requests* correspondentes a objetivos de um projeto específico.

Além de listar os objetos alterados, em cada *task* é documentado quem alterou cada customização ou objeto de desenvolvimento, bem como o propósito de cada alteração. *Change requests* e *tasks* proporcionam um histórico completo de todas as mudanças ocorridas durante um projeto na linguagem ABAP. Dado que as mudanças no software criadas durante uma implementação de um projeto na linguagem ABAP são resultado de configurações (*customizing*) ou desenvolvimento, havendo dois tipos de *change requests*, *customizing change requests* e *workbench change requests*. As alterações de código fonte são salvas em *workbench change requests*. Por fim, a liberação das *tasks*, e consequentemente da *request*, é o primeiro passo para o transporte destas alterações dentro do *landscape*.

2.3. Automação de Testes

Um software é tão confiável quanto a sua cobertura de testes, e os testes tão valiosos quanto a sua frequência de execução [Duvall, Matyas and Glover 2007]. Desta forma, a automação dos diferentes tipos de testes e sua execução frequente são o caminho para a elevação de qualidade de um software.

A seguir são listadas técnicas e tecnologias envolvidas em automação de testes e usadas na composição do ambiente de desenvolvimento deste estudo:

- Testes Unitários: adota-se a definição de Cunningham (2013), que se trata de um tipo de teste automatizado ou teste de desenvolvedor, para outros autores. Unitário, neste contexto refere-se aos testes de baixo nível, escritos na mesma linguagem do

código produtivo, que acessa diretamente seus objetos e membros. Para fins de qualidade, a falha de um teste unitário implica em somente uma unidade e desta forma, saber exatamente onde achar o *bug* [Cunningham 2013];

- *ABAP Unit*: uma ferramenta do tipo *xUnit* para a linguagem ABAP. É diretamente incluída dentro do ambiente de desenvolvimento ABAP e do ambiente *runtime* ABAP [ABAP Unit 2017];
- *ABAP Test Cockpit* (ATC): conjunto de ferramentas que permite executar testes estáticos em aplicações ABAP. É diretamente integrado aos ambientes de desenvolvimento ABAP, permitindo ao desenvolvedor verificar seu código dentro do contexto em que está acostumado [SAP Community 2016];
- *JUnit*: framework aberto para escrever e executar testes unitários na linguagem Java. Desta forma, consiste numa instância de uma arquitetura *xUnit* para *framework* de testes unitários [JUnit 2017];
- *Axis Apache WSDL2Code*: *plug-in* que através de um *WSDL* gera *stubs* de cliente e servidor para a chamada ou a implementação de *web service*, conforme *WSDL* utilizado [Apache 2016];
- *Apache Olingo*: biblioteca *Java* que implementa *Open Data Protocol* (OData). Atende aspectos de cliente e servidor do protocolo *OData* [Apache Olingo 2017], que é a melhor forma de consumir e prover *web services REST* [OData 2015];
- *Raspberry PI*: computador de tamanho reduzido, originalmente projetado para educação, inspirado no 1981 BBC Micro [CCH 2017] [OPENSOURCE 2017];
- *Bootstrap*: *framework* para *HTML*, *CSS* e *Java Script* para desenvolvimento de projetos *mobile first* responsivos. É *open source*, hospedado, desenvolvido e mantido no *GitHub* [Bootstrap 2017];
- Selenium: *framework* de teste para aplicações *web*, que provê uma ferramenta de gravação/*playback*, sem a necessidade de aprender linguagem de *script*. É provido de linguagem de domínio específico para desenvolvimento de testes em várias linguagens como *Java*, *C#*, *PHP,* entre outras [Selenium 2017].

3. Método de Pesquisa

Esta pesquisa caracteriza-se pela natureza aplicada, qualitativa e exploratória, com coleta de dados longitudinal. Adota-se pesquisa-ação como método de pesquisa, consistindo em modificações no ambiente foco do estudo através da ação do pesquisador, evidenciada pela implementação de diferentes ferramentas relacionadas ao processo de IC.

A pesquisa foi realizada no contexto de uma equipe de desenvolvimento de um setor, pertencente a uma multinacional do ramo de *Enterprise Resource Planning* (ERP), com atuação em mais de 190 países, tendo até o presente momento mais de 320.000 clientes. Conforme estrutura organizacional, essa equipe se encontra cinco níveis abaixo do diretor responsável, no Conselho de Administração.

O setor no qual essa equipe pertence é responsável pela adaptação do software padrão da empresa para especificidades de cada país, sendo essa equipe de

desenvolvimento responsável pelo produto UEG da organização. Este produto consiste num *framework* para o tratamento de dados transacionais provenientes de sistemas de ERP, com base em um banco de dados *in-memory*. O UEG foi lançado ao final de 2013 e desde então vem sendo atualizado e tem ganhado novas funcionalidades. Atualmente, o UEG já vendeu mais de 70 licenças, com aproximadamente 17 clientes em produção.

Quanto à coleta de dados, a técnica análise documental foi adotada, a partir de registros das ferramentas internas da empresa, por meio de dados do servidor de *IC Jenkins*, além de observação direta, tendo em vista que um dos pesquisadores atua na equipe do estudo [Azevedo e Machado 2011].

A principal métrica selecionada para verificação dos resultados e benefícios foi o número de chamados de clientes por *support package* (SP), porque permite inferir a eficiência do processo de teste do produto UEG como um todo, visto que em cada SP são realizadas várias etapas de testes, para fins de identificação e correção do máximo de defeitos possível, além de estar em uso há algum tempo na organização.

Para a análise dos dados quantitativos foram gerados gráficos da quantidade de chamados de clientes, após cada SP, de forma a aferir a eficiência no desenvolvimento. Idealmente, antes do lançamento de um SP todos defeitos deste software seriam detectados e nenhum chamado de cliente existiria. Estes chamados forma classificados de acordo com a suas causas. As causas que foram utilizadas neste trabalho foram: dúvida, erro de cliente, novo requisito e *bug*. Esta forma de analisar os incidentes foi definida numa reunião dos conceitos de *Deffect Removal Effectiveness*, citados por Kan (2002) com uma forma de classificação das ferramentas internas da empresa.

Entre as limitações, o processo de *Continuous Inspection* [Duvall, Matyas and Glover 2007] não foi considerado, dado as limitações técnicas existentes no ambiente de desenvolvimento objeto desse estudo, que não permitem a coleta unificada das métricas de análise estática de código. O foco deste estudo é o subprocesso de *Continuous Testing* [Duvall, Matyas and Glover 2007].

Outra limitação relaciona-se com o fato de alguns autores incluírem o *deploy* do código, como uma etapa de IC [Ståhl, Bosch 2014]. Neste trabalho, o *deploy* de código não está incorporado no processo de IC. Ainda, cabe ressaltar que os achados do estudo se limitam à análise dos dados quantitativos obtidos, a partir da métrica supracitada. Portanto, os autores reconhecem outras análises e considerações possíveis, através da implementação futura de mais métricas relacionadas ao processo, na organização.

4. Desenvolvimento da Solução Técnica

Nesta seção são descritas as ações implementadas de IC na linguagem ABAP, na unidade de análise, bem como os resultados da adoção deste processo no desenvolvimento do produto UEG. Antes, o processo de desenvolvimento do UEG é mostrado brevemente.

4.1. Processo de Desenvolvimento do Produto UEG

A seguir contextualize-se o processo de desenvolvimento do produto UEG. Esse processo é formado pelas seguintes etapas:

1. *Support Package Planning*: a partir de *support packages* (SPs), o produto UEG é entregue. Esta entrega contém correções de *bugs*, mudanças legais e novas funcionalidades. As mudanças legais e novas funcionalidades são planejadas e priorizadas, para serem desenvolvidas durantes os *sprints;*
2. *Sprints*: Um *sprint* no desenvolvimento do UEG possui duas semanas de duração. São desenvolvidas partes das novas funcionalidades e mudanças legais, sob forma de *backlog items*. Além de *designs* de desenvolvimentos complexos e testes funcionais dos *backlogs* finalizados, no *sprint* anterior. Testes unitários são considerados parte do desenvolvimento. O s*print* só será aceito se todos os testes unitários estiverem "OK";
3. *Solution Acceptance Test* (SAT): duas semanas antes do fechamento do sistema para o processo de *Assembly* é executada uma bateria de testes, envolvendo especialistas de negócio e desenvolvedores. Funcionalidades desenvolvidas são retestadas pelos especialistas de negócio e os desenvolvedores focam em correções de *bugs* encontrados e na criação e execução de testes adicionais. Para o SP ser aprovado para o *Assembly*, o SAT precisa ter os seguintes critérios de saída atendidos: *(i)* 100% dos testes executados; *(ii)* 90% dos testes com *status* OK; *(iii) t*odos bugs com prioridade 1 e 2 fechados; *(iv)* todos *bugs* com prioridade 3 e 4, acompanhados de definições de como serão tratados (assinalados em *backlogs* ou não serão corrigidos);
4. *Assembly*: consiste em consolidar todas as *requests* de desenvolvimento num formato em que os clientes possam fazer *download* do site da empresa e instalar em seus servidores. Porém, além desta consolidação são realizados testes de instalação e de segurança. O processo de *Assembly* é finalizado, quando os testes de instalação e segurança são finalizados e os eventuais erros encontrados corrigidos. Uma vez que esse processo é finalizado, passa-se para a etapa *Release to Customer*;
5. *Release to Customer*: consiste na publicação do *link* para *download* do SP no *website* da empresa e a notificação dos clientes, posteriormente.

4.2. Implementação de IC para a Linguagem ABAP

Esta pesquisa adapta os conceitos de IC para a linguagem ABAP. A primeira diferença substancial da linguagem ABAP em comparação com outras linguagens é que não há desenvolvimento na máquina local do desenvolvedor. Todo desenvolvimento é centralizado no sistema de desenvolvimento, instalado num servidor remoto. Desta maneira, o *commit* no repositório central é substituído por um *test transport* para o sistema de qualidade. Dada esta restrição, a seguinte abordagem de adaptação foi feita nesse contexto, baseada em Duvall, Matyas e Glover (2007) e conforme Tabela 1.

A partir desta adaptação foi montada uma estratégia de consolidação para a agregação de todos os resultados em uma fonte de dados única, utilizando o servidor de *IC Jenkins*. Como o UEG possui diferentes tecnologias integradas, uma infraestrutura de testes precisou ser criada, conforme mostrado na Figura 2.

Já, a infraestrutura de desenvolvimento do sistema UEG é composta por quatro sistemas diferentes organizadas da maneira ilustrada na Figura 3. Nessa figura, os dois *landscapes* são necessários devido ao mecanismo de entrega de SPs do *NetWeaver*. O

processo de *Assembly* [Merrian-Webster 2017] de um SP toma o tempo de três semanas e é executado por um departamento externo. Durante o processo de *Assembly*, o sistema de desenvolvimento precisa ficar fechado, fazendo com que a equipe fique bloqueada. Com este *setup*, a equipe pode continuar desenvolvendo no *landscape Infinity*, enquanto o *landscape* de entrega fica fechado para o *Assembly* do SP.

Tabela 1. Adaptação dos passos fundamentais de IC para a Linguagem ABAP

Prática	**Implementação na Linguagem ABAP**
1. Executar o *commit* de código frequentemente	Executar *test transports* frequentemente
2. Não executar o *commit* de código quebrado	Não transportar código que possuam testes com falha ou erros de ATC.
3. Corrigir builds quebrados imediatamente	Corrigir o mais rápido possível, erros de testes e ATCs nos sistemas de qualidade.
4. Escrever testes de desenvolvimento automatizados	Criar testes automatizados para tudo o que possa ajudar o time
5. Todos os testes e inspeções devem passar	O status dos sistemas de qualidade devem ser sempre OK
6. Rodar builds Privados	Rodas os builds nos sistemas de desenvolvimento frequentemente
7. Evitar ficar mantendo código quebrado	Corrigir o quanto antes erros de testes e ATCs nos sistemas de qualidade.

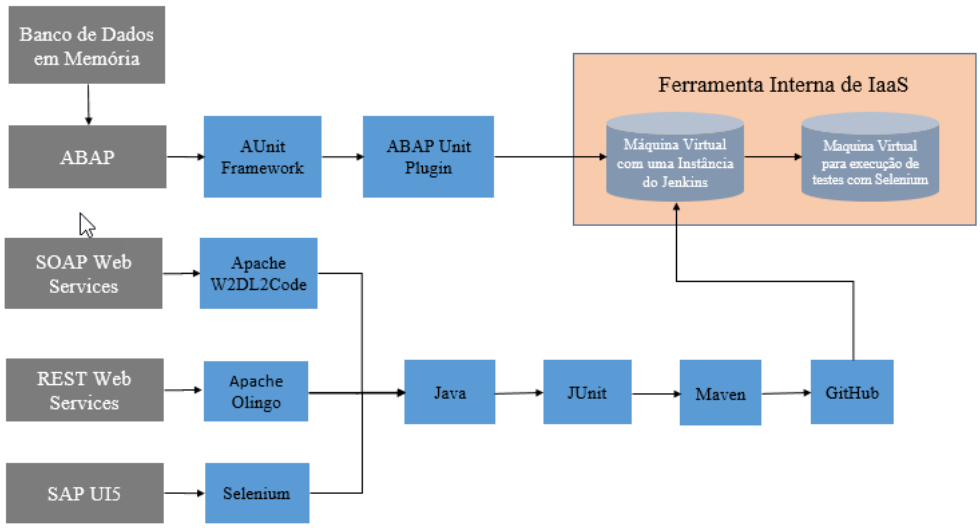


Figura 2. Infraestrutura de testes automáticos no desenvolvimento do UEG

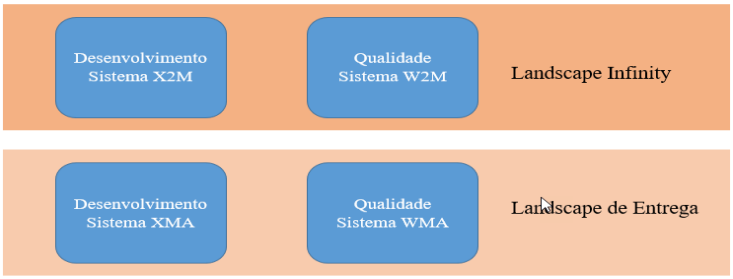
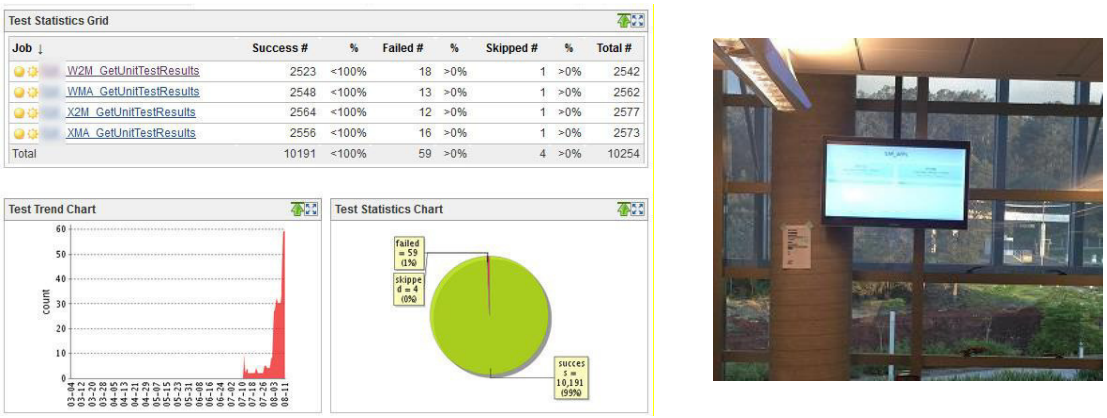


Figura 3. *Landscapes* de desenvolvimento do produto UEG

Outro motivo importante para a utilização deste *setup* é o fato do *landscape* de entrega ter como objetivo se assemelhar o mais próximo possível aos sistemas dos clientes. A partir deste *landscape* são geradas as *code patches* de correção de *bugs* reportados pelos clientes. Estes *code patches* são internamente chamados de notas. Já o *landscape Infinity* é dedicado ao desenvolvimento de funcionalidades maiores que só são entregues via SP.

Anteriormente, o *feedback* do sistema se dava somente através de oito *e-mails* diários, provenientes das ferramentas atuais do *framework* ABAP. Com o passar do tempo, estes *e-mails* foram sendo ignorados pelos desenvolvedores. Tendo isso em vista, os resultados foram integrados no *Jenkins*, sendo uma forma única de visualização para os testes unitários. Porém, devido à grande quantidade de testes automáticos, houve uma grande economia de tempo para se observar os resultados, conforme mostra a Figura 4(a), retirada do servidor *Jenkins* do time UEG.



(a) (b)

Figura 4. Monitoramento dos ambientes de desenvolvimento do produto UEG

Para melhorar ainda mais a visualização do *status* dos testes foi criado um monitor de IC, usando uma TV 42", junto de um mini *PC Raspberry PI*, conforme Figura 4(b). Inicialmente foi colocado uma *view* do próprio *Jenkins* para o monitoramento. Porém, o *feedback* inicial dos desenvolvedores foi que os resultados apresentados eram de difícil visualização. Diante disso foi desenvolvida uma nova *interface* para visualização, com o *framework web Bootstrap*. Dada a diretriz estabelecida na Tabela 1 foi colocado nessa TV somente os *status* dos sistemas de qualidade. O desenvolvimento da solução completa foi executado nessa pesquisa-ação, conforme linha do tempo da Tabela 2.

Tabela 2. Implantação de IC nos *support packages* do produto UEG

Support Package	Etapa do processo
SP04 (de 13/02/2015 até 24/08/2015)	*Setup* do *Jenkins*; Monitoramento dos testes unitários; Instalação de monitor de 24".
SP05 (de 25/08/2015 até 11/12/2015)	Desenvolvimento dos testes de *Web Services SOAP.*
SP06 (de 12/12/2015 até 04/04/2016)	Desenvolvimento dos testes em *Selenium*; Desenvolvimento dos testes de *Web Services REST.*
SP07 (de 05/04/2016 até 08/08/2016)	Instalação de TV de 42"; Nova *interface* em *Bootstrap;* Inclusão das medições de cobertura de comandos dos desenvolvimentos em ABAP.

4.3. Medição e Análise dos Resultados Obtidos

Ressalta-se que o processo de IC implantado até o fim deste trabalho não incorpora a totalidade das práticas que o compõe (por exemplo, a automação de testes de cenário, o processo de inspeção contínua, dentre outros). Na Figura 5, o número de chamados de clientes, com as respectivas causas após cada SP é apresentado. Retrata-se a quantidade de incidentes, após cada SP como uma forma de medir a eficiência da etapa de SAT, visto que idealmente, esta etapa deveria encontrar todos os *bugs* do produto.

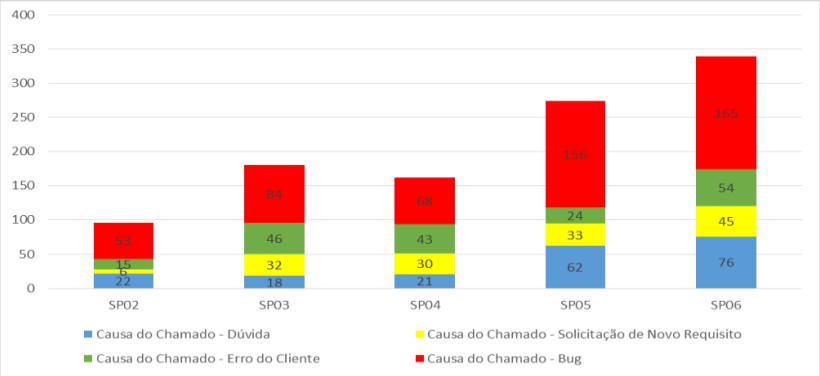


Figura 5. Evolução dos chamados em função dos *support packages* do UEG

Na Figura 5, avaliando-se os últimos dois SPs verifica-se um aumento de 23% na quantidade de chamados de clientes (de 275 para 340), bem como um aumento de aproximadamente 6% do número de chamados causados por *bugs* (de 156 para 165). Numa primeira análise pode-se imaginar que IC não proporcionou uma diminuição na quantidade de chamados e consequentemente, não adicionou qualidade ao produto UEG. Porém, há um fato que indica uma melhoria consistente: o fato do aumento do número de incidentes causados por *bugs* (6%) ser uma ordem de grandeza significativamente menor, do que o aumento do número de incidentes total (23%).

Na Figura 6 tem-se o percentual dos chamados de clientes gerados por *bugs* na quantidade geral de incidentes de cada SP. Esta análise é importante para compreender o impacto do desenvolvimento no total dos chamados. Como pode ser observado, há uma redução significativa no percentual dos *bugs* nos últimos SPs (de 56,97% para 48,68%), o que indica uma tendência de melhoria no processo de desenvolvimento. A expectativa era que esta tendência se confirmasse a cada SP, entretanto, é possível perceber que outros benefícios foram alcançados com a adoção de práticas de IC.

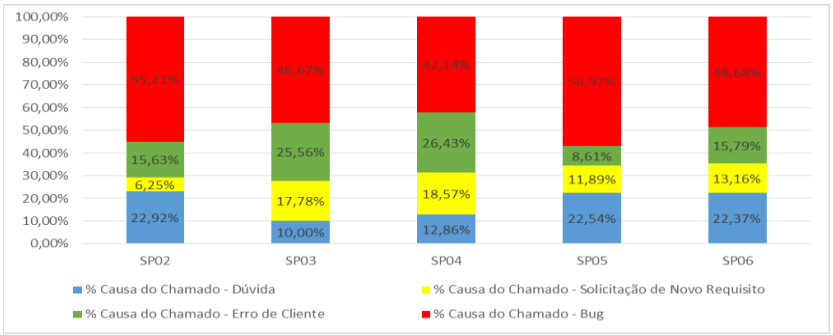


Figura 6. Distribuição das causas dos chamados dos *support packages* do UEG

Nota-se uma quantidade de chamados causados por *bugs* um tanto desproporcional no SP05. Um fato que pode justificar esta situação refere-se a entrada em produção de grandes clientes. Isto levou a um estresse da solução significativo, fazendo com que muitos *bugs* fossem identificados no mesmo período de tempo.

A partir do início do SP07 foi iniciado a medida da cobertura de linhas de comando em ABAP. Com base neste monitoramento, a equipe de desenvolvimento percebeu que haviam uma quantidade grande de código relativo às funcionalidades obsoletas ou ferramentas internas que continuavam sendo entregues para os clientes sem necessidade. Estas linhas de códigos eram um verdadeiro passivo tecnológico para o produto UEG, que eventualmente causavam problemas de instalação.

Conforme a Figura 7, extraída de ferramenta interna (*ABAP Unit Framework*) pode-se ver redução de mais de 60.000 linhas de código, que eram entregues sem necessidade para os clientes, tornando o produto muito mais enxuto e muito mais manutenível. Percebe-se uma clara tendência de crescimento nos chamados de clientes, o que pode ser interpretado como falta de qualidade no produto. Porém, verificando-se a Figura 6 e avaliando a tendência de crescimento do número total de chamados clientes, em comparação com a tendência de crescimento de chamados de clientes causados por *bugs* é possível considerar que estas duas tendências não tem o mesmo comportamento, principalmente, na comparação entre os dois últimos SPs.

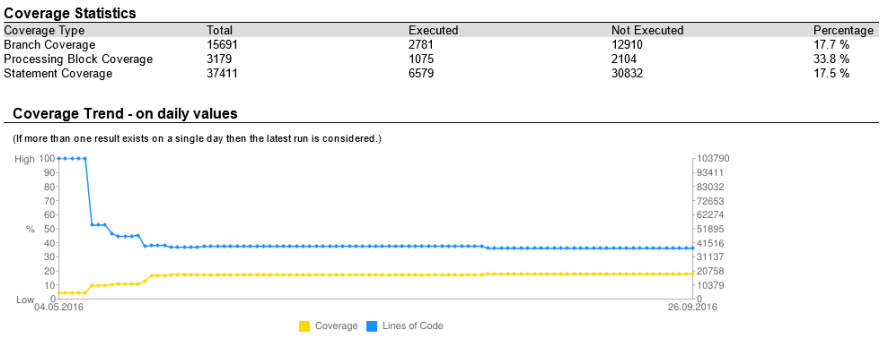


Figura 7. Evolução da cobertura de linhas de código no produto UEG

Neste caso, o número total de chamados aumentou 23% (de 275 para 340) e o número de chamados causados por *bugs* aumentou 6% (de 156 para 165). Vale ressaltar que o produto UEG é dedicado ao tratamento de dados tributários de empresas no Brasil. De acordo com Forbes (2013), o Brasil detém a maior complexidade tributária do mundo. Considerando este fato e somando-se à realidade de que cada cliente adquirente do produto UEG pode possuir as suas próprias especificidades tributárias, é esperado que o número de chamados aumente até atingir um ponto de equilíbrio.

O grande mote deste trabalho, a partir da orquestração dos testes automáticos via IC é evitar que regressões sejam incluídas no código, e consequentemente, *bugs* corrigidos voltarem. Se por um lado não foi percebida uma redução no número de incidentes, após esta implementação parcial do processo de IC é válido afirmar que sem o processo de IC, provavelmente, o número de chamados cresceria de uma forma muito mais agressiva do que a verificada neste trabalho.

5. Considerações Finais

Dado o objetivo geral estabelecido nesse estudo foi implantado um processo de IC no desenvolvimento ABAP, como forma de aumentar a qualidade do produto UEG e melhorar o *feedback* dos testes automáticos para os desenvolvedores. Como consequência desta implantação, esperava-se uma diminuição nos chamados de cliente. Mesmo que este benefício não tenha sido diretamente alcançado foi possível determinar uma melhora na qualidade do produto UEG, seja pela diminuição do percentual de *bugs* no último SP, seja pela percepção coletada da equipe envolvida, via reuniões de projeto.

Os motivos que limitaram o alcance dos benefícios podem estar associados ao fato de que o processo de IC não estar implementado em sua totalidade no contexto em estudo e ao produto ainda estar em fase de amadurecimento, tendo em vista muitos chamados estarem relacionados a dúvidas e erros feitos pelos clientes, bem como a adição de novos requisitos, dada a complexidade do domínio e uma variação de cliente para cliente. Cabe observar que somente melhorando o sistema de *feedback* dos testes automáticos foi possível notar uma melhora na qualidade do produto. Com isto, é possível atestar o atendimento dos objetivos de pesquisa.

Uma observação sobre o método de avaliação dos resultados se faz importante. Ele é fortemente baseado na métrica *Deffect Removal Effectiveness*, postulada por Kan (2002). Nos resultados foi verificado a eficiência de todo o processo de desenvolvimento da seguinte forma: idealmente, o processo de teste do produto UEG (incluindo a IC) detectaria todos os bugs do produto. Assim sendo, cada chamado de cliente pode ser interpretado como uma perda de eficiência no processo de teste. Esta visão garantiu uma maneira consistente de medir progresso, fato que permitiu atestar que a proposta deste artigo foi atingida de forma quantitativa.

Vale ressaltar que este trabalho não possui precedente na literatura. Sugestões de como implantar um processo de IC em linguagens de uso amplo (como Java, por exemplo) são relativamente comuns dentro e fora do meio acadêmico. Porém, ao tratar uma linguagem que não possui desenvolvimento local, tão específica como o ABAP foi preciso uma carga autoral considerável, para viabilizar a proposta de solução técnica e sua conseguente implantação, na organização objeto do estudo.

Como sugestão de futura pesquisa pode-se conduzir uma implementação técnica do processo, consistindo em um guia a ser utilizado para implementar IC em desenvolvimentos ABAP em âmbito global, tanto dentro da empresa SAP como em empresas parceiras. Um ganho extremamente interessante com a adoção deste processo, em outros contextos de desenvolvimento seria o *feedback* prático, que a solução proposta neste artigo teria, contribuindo para o aperfeiçoamento do mesmo. Além disso, sugere-se dar continuidade na implementação de todo o escopo teórico de IC (*continuous inspection*, *deploy,* etc.) para a linguagem ABAP.

References

ABAP. (2017) “ABAP SAP Documentation”, https://help.sap.com/saphelp_nw70/helpdata/en/fc/eb2e97358411d1829f0000e829fbfe/frameset.htm. Acesso em: 29 abr. 2017.

ABAP Sonar. (2017) “SonarABAP“, https://www.sonarsource.com/why-

us/products/codeanalyzers/sonarabap.html. Acesso em: 24 abr. 2017.

ABAP Unit. (2017) “ABAP Unit”, https://help.sap.com/saphelp_nw74/helpdata/en/49/18061c005338a1e10000000a421937/content.htm. Acesso em: 29 maio 2017.

Apache. (2016) “WSDL2Code”, https://axis.apache.org/axis2/java/core/tools/maven-plugins/axis2-wsdl2code-maven-plugin/. Acesso em: 16 abr. 2017.

Apache Olingo. (2017) “Apache Olingo™”, https://olingo.apache.org/. Acesso em: 16 abr. 2017.

Azevedo, D. and Machado, L. (2011) “Métodos e Procedimentos de Pesquisa: do Projeto ao Relatório Final”, Editora Unisinos.

Bootstrap. (2017) “Bootstrap”, http://getbootstrap.com. Acesso em: 16 abr. 2017.

Centre for Computing History (CCH). (2017) “Eben Upton - Life Before Raspberry Pi”, http://www.computinghistory.org.uk/det/42103/Eben%20Upton%20-%20Life%20Before%20Raspberry%20Pi. Acesso em: 07 abr. 2017.

Cunningham (2013). “Unit Test”, http://c2.com/cgi/wiki?UnitTest. Acesso em: 29 mar. 2017.

Duvall, P.; Matyas, S.; Glover, A. (2007) “Continuous Integration: Improving Software Quality and Reducing Risk“, Addison-Wesley.

Forbes. (2013) “Brazil Ranked Most Time-Consuming Tax Regime In The World”, http://www.forbes.com/sites/joeharpaz/2013/12/17/brazil-ranked-most-time-consuming-tax-regime-in-the-world/#5719a8622fdf. Acesso em: 14 mar. 2017.

Fowler, M. (2006) “Continuous Integration”, https://martinfowler.com/articles/continuousIntegration.html. Acesso em: 15 mar. 2017.

Git-scm. (2017) “Git-commit”, https://git-scm.com/docs/git-commit. Acesso em: 11 abr. 2017.

Held, M. (2015) “Journey to setup best possible Continuous Integration with ABAP Project”, goo.gl/TGjhnP. Acesso em: 26 jul. 2017.

ISOTOPE11. (2017) “Jenkins blog”, https://isotope11.com/blog/styling-your-jenkins-continuous-integration-server. Acesso em: 25 abr. 2017.

Jenkins. (2017) “Jenkins”, https://jenkins-ci.org/. Acesso em: 05 abr. 2017.

JUnit. (2017) “Frequently Asked Questions”, http://junit.org/junit4/faq.html#overview_1. Acesso em: 29 abr. 2017.

Kan, S. H. (2002) “Metrics and Models in Software Quality Engineering”, Addison-Wesley, 2th edition.

Krawczyk, A. (2013) “Continuous Integration – Automated ABAP Unit Tests in Hudson”, goo.gl/mdkPTy. Acesso em: 26 jul. 2017.

Martin, R. C. (2002) “Agile Software Development: Principles, Patterns, and Practices”, Prentice Hall.

McCabe, T. and Watson A. (1994) “Software Complexity”, *CrossTalk: The Journal of Defense Software Engineering*.

Merrian-Webster. (2017) “Assembly Language”, http://www.merriam-webster.com/dictionary/assembly%20language. Acesso em: 25 mar. 2017.

Moreira, G. de S. P. *et al*. (2010) “Software product measurement and analysis in a continuous integration environment”, Information Technology: New Generations, Third International Conference on, p. 1177–1182.

Mota, G. M. (2015) “Implantando uma Etapa de Análise Estática para Revisão de Código em uma Infra de Integração Contínua”, Trabalho de Conclusão do Curso de Especialização em Qualidade de Software – Unidade Acadêmica de Pesquisa e Pós-Graduação, Unisinos, São Leopoldo.

Netweaver. (2004) “Overview of NetWeaver AS ABAP”, https://help.sap.com/saphelp_nw70/helpdata/en/fc/eb2e97358411d1829f0000e829fbfe/frameset.htm. Acesso em: 07 maio 2017.

OData (2015). “OData - the best way to REST”, http://www.odata.org/. Acesso em: 16 abr. 2017.

Pressmann, R. S. and Maxim, B. R. (2016) “Engenharia de Software: uma abordagem professional”. 8. ed. AMGH. ISBN 978–85–8055–533-2.

OPENSOURCE. (2017) “What is a Raspberry Pi?”, https://opensource.com/resources/what-raspberry-pi. Acesso em: 16 abr. 2017.

SAP. (2017) “About SAP SE”, http://go.sap.com/corporate/en.html. Acesso em: 04 abr. 2017.

SAP Community. (2017) “ABAP Test Cockpit”, https://wiki.scn.sap.com/wiki/display/ABAP/ABAP+Test+Cockpit. Acesso em: 16 abr. 2017.

SAP Landscape. (2017) “The SAP System Landscape”, http://goo.gl/G79IQG. Acesso em: 27 mar. 2017.

SAP Help. (2017) “Overview of NetWeaver AS ABAP”, http://help.sap.com/saphelp_nw70/helpdata/en/fc/eb2e97358411d1829f0000e829fbfe/content.htm. Acesso em: 15 mar. 2017.

Sepalla, A. (2010) “Improving software quality with Continuous Integration”, Dissertação de Mestrado, Alto University, Esbo.

Ståhla, D. and Bosch, J. (2014) “Modeling continuous integration practice differences in industry software development”, The J. of Systems and Software, 87, pp 48-59.

Selenium. (2017) “What is Selenium?”, http://docs.seleniumhq.org/. Acesso em: 22 abr. 2017.

Smit, M., Gergel B., Hoover H. J. and Stroulia E. (2011) "Code convention adherence in evolving software", In: 27th IEEE International Conference on Software Maintenance (ICSM), Williamsburg, VI, 2011, pp. 504-507.