Sistema de auxílio à tomada de decisão da área agrícola DSSAT: automatizando o processo de integração de novas versões

Marina Dezordi Lopes[1], **Alexandre Lazaretti Zanatta**[1], **Willingthon Pavan**[1], **Carlos Amaral Holbig**[1]

[1]Programa de Pós-Graduação em Computação Aplicada (PPGCA)
Universidade de Passo Fundo (UPF) 99.052-900 – Passo Fundo – RS – Brasil

marinadezordi@gmail.com, {zanatta,pavan,holbig}@upf.br

Abstract. *This paper presents a software development process in the automation of testing, code integration and verification of the new version of the software, through the technique of Continuous Integration in a Support System for Decision and Transfer in Agrotechnology (DSSAT). For this, several tools and frameworks were used. The results shows the effective automation of the DSSAT versioning process and, consequently, a reduction in the work performed by its administrators were noticed.*

Resumo. *Este trabalho apresenta um processo de desenvolvimento de software na automatização de testes, integração de códigos e verificação da nova versão do software, por meio da técnica da Integração Contínua em um Sistema de Suporte para Decisão e Transferência em Agrotecnologia (DSSAT). Para isso, foram utilizadas várias ferramentas e frameworks. Percebeu-se a efetiva automatização do processo de versionamento do DSSAT e, por consequência, uma redução no trabalho realizado pelos seus administradores.*

1. Introdução

A entrega de uma nova funcionalidade ou alteração de um software em produção para utilização do usuário deve ser confiável e passar por todas as rotinas de testes necessárias para garantir a integridade do sistema. Dependendo do tamanho do software, esses testes podem ser demorados, demandando mais tempo dos desenvolvedores para chegar a um resultado satisfatório.

Um processo de organização de desenvolvimento de sistema, com verificação automática de dependências e de configurações, de testes e de integração pode tornar o trabalho do desenvolvedor mais rápido e mais confiável. Além disso, acredita-se que é possível a obtenção de uma maior agilidade na entrega de novas versões de sistemas, com mais qualidade e segurança.

O *Decision Suport System for Agrotechnology Transfer* (DSSAT) ou Sistema de Suporte para Decisão e Transferência em Agrotecnologia, realiza a simulação de modelos de mais de 42 culturas como soja, milho, trigo e sorgo, entre outras. O objetivo é simular o crescimento, o desenvolvimento e o rendimento em função da dinâmica solo-planta-atmosfera conforme destacam Jones et al. [Jones et al. 2003].

O DSSAT incorpora ferramentas que facilitam o uso desses modelos, entre eles programas de gerenciamento de banco de dados para solo, clima, culturas e dados experimentais, utilitários e aplicativos.

A falta de uma automatização e organização do processo de integração de novas versões da suíte DSSAT, aliada a quantidade de 300 mil linhas de código, incentivou a realização deste trabalho. Atualmente os desenvolvedores e administradores do DSSAT fazem manualmente as atualizações, as integrações, as verificações de resultados, os testes e a implantação das novas versões da suíte.

Este trabalho apresenta o processo de automatização de versionamento da suíte DSSAT, por meio da utilização da técnica da Integração Contínua [Hilton et al. 2016]. A seção 2 apresenta o referencial teórico, e na seção 3, a suíte DSSAT e o seu módulo Croptest são detalhados. Na seção 4 discute-se, brevemente, os trabalhos relacionados. A metodologia é apresentada na seção 5. Os resultados e as análises são discutidos na seção 6. Por fim, nas seções 7 e 8 as conclusões e as referências são apresentadas respectivamente.

2. Referencial Teórico

Integração Contínua (IC) é um processo realizado durante o desenvolvimento de software em que a integração dos trabalhos de uma equipe ocorre com muita frequência, ao menos uma vez ao dia. O princípio de IC tem como objetivo automatizar as integrações com uma nova funcionalidade executável a cada atualização em um software. Martin Fowler em [Hilton et al. 2016] foi o responsável por disseminar a IC. Em cada integração é executada uma nova *build* automatizada do software, assim, essa abordagem possibilita que a equipe identifique possíveis problemas e que possa resolvê-los rapidamente. *Build* é o resultado da conversão do código-fonte em um software que será executado em um computador.

A prática da IC acontece quando uma nova alteração é adicionada ao sistema e enviada ao repositório remoto. O passo seguinte é executar a *build* e realizar os testes automáticos para cobrir todo o sistema, incluindo as partes recém-adicionadas. O desenvolvedor recebe um retorno sobre os códigos integrados e, em caso de falha, deve priorizar os ajustes [Hamdan and Alramouni 2015].

Algumas opções de ferramentas para IC estão disponíveis. O Travis CI, é uma plataforma de IC gratuita para projetos de código fonte aberto, oferecendo suporte ao processo de desenvolvimento, construindo e testando automaticamente alterações de código, fornecendo retorno imediato sobre as alterações. O CircleCI, é uma ferramenta baseada em nuvem, possibilita a automatização dos testes em máquinas virtuais ou em unidades padronizadas chamadas de contêineres que possuem tudo o que o software precisa para executar, como o código fonte, as bibliotecas e as ferramentas de sistema. O Jenkins, é um servidor de automação independente e de código aberto usado para automatizar todos

os tipos de tarefas relacionadas à criação, teste e distribuição ou implementação de software. Estas três ferramentas possuem integração com o Github e com o Docker, e podem ser utilizadas gratuitamente quando o repositório é privado.

A construção de um container é a base do processo de automatização. O Docker é um container que possibilita a criação de ambientes específicos e customizados de acordo com a necessidade.

3. A suíte DSSAT e o seu módulo Croptest

O uso do DSSAT é bastante amplo, mais de 14 mil pesquisadores, educadores, consultores, agentes de extensão, produtores, políticos e tomadores de decisão distribuídos em mais de 150 países. O software é utilizado nas fazendas, em gerenciamento de precisão, avaliações regionais do impacto da variabilidade climática e das mudanças climáticas, modelagem baseada em gene e em seleção de reprodução, uso de água, emissão de gases de efeito estufa e sustentabilidade a longo prazo através dos equilíbrios de carbono orgânico e nitrogênio no solo, entre outras várias aplicações importantes.

As opções de cultura, de solo, de dados climáticos e de gerenciamento permitem ao usuário fazer diversas combinações que geram uma simulação de experimento virtual sem ser necessário experimentos reais. O DSSAT permite aos usuários a comparação entre simulação de resultados com dados observados. A partir da simulação de prováveis resultados de estratégias de manejo de culturas, oferece aos usuários dados para avaliar novas culturas, produtos e práticas a serem adotadas.

O DSSAT é a combinação de modelos de simulação de culturas (CSM do inglês *Cropping System Model*), sistemas de banco de dados e sistemas de suporte à decisão. Os CSMs são os softwares que simulam o crescimento e o rendimento das culturas, com base em cálculos previamente estabelecidos e dados de entrada das práticas de manejo do solo, clima e cultura. O DSSAT-CSM simula o rendimento de uma variedade de culturas.

O CropTest (*Model Comparison Utility* ou Utilitário de Comparação de Modelos) do DSSAT tem objetivo de comparar as saídas de modelos de culturas de duas versões diferentes do DSSAT. Ele é utilizado pelos administradores do sistema para realizar os testes e concluir se as alterações submetidas pelo desenvolvedor podem ser integradas ao *branch release*. Na sua versão atual, o CropTest recebe os arquivos de entrada, que são processados pelos DSSAT, e gera os seguintes arquivos de saídas: `plantgrow.out`, `summary.out` e `evaluate.out`. Gera, também, uma planilha de diferenças percentuais apontando aquilo que se manteve, foi melhorado ou piorado, para que, a partir disso, possa ser tomada a decisão. Se os testes alcançaram bons resultados, as alterações podem ser incluídas ao *branch release*.

Quando o desenvolvedor faz uma alteração, testa e aprova o resultado, inicia-se um processo de testes e aprovação dessas alterações para que possam ser incluídas nas novas versões do DSSAT. Observa-se que o desenvolvedor precisa requisitar aos administradores do repositório a inclusão da alteração no ramo *develop*. Os administradores do DSSAT utilizam a ferramenta de testes Croptest para realizar a avaliação do que foi desenvolvido.

A partir dessa avaliação, os administradores decidem se o código está, ou não, apto a ser integrado, como pode ser visualizado na Figura 1. Toda vez que um *pull*

request (1), ou seja, um pedido de integração de novos códigos é feito, inicia-se um longo processo. O administrador recebe a notificação do GitHub (2), faz o download do código completo do desenvolvedor (3) e executa a compilação (4), que pode demorar alguns minutos. Quando a *build* é executada e apresenta erros (5) o administrador rejeita o *pull request* (7). Caso contrário, quando a *build* não apresenta erros (8) o administrador parte para o processo de testes utilizando o Croptest (9). Neste momento são comparadas as diferenças dos resultados do DSSAT atualizado pelo desenvolvedor e o DSSAT em versão estável. Se o Croptest apresentar diferenças (6) o administrador rejeita o *pull request* (7), caso contrário, o *pull request* pode ser aceito (10) e, finalmente, integrado ao DSSAT no *branch develop*.

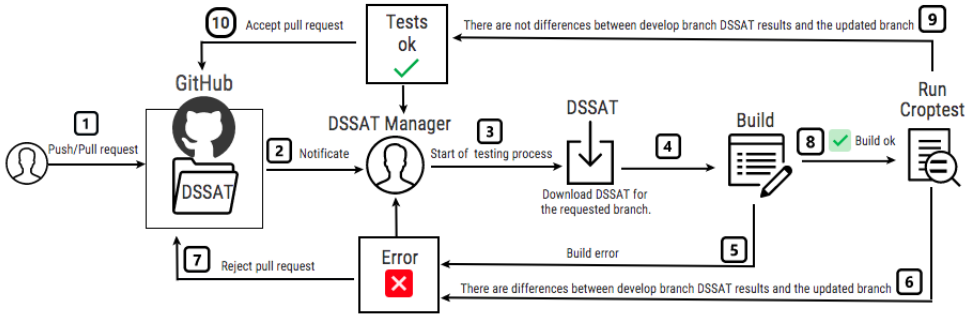

Figura 1. Fluxo das integrações atuais no DSSAT

4. Trabalhos Relacionados

A aplicação da IC no DSSAT se assemelha na automatização dos testes demonstrados no trabalho de Mossige, Gotlieb e Meling [Mossige et al. 2015], que pode facilitar os testes e garantir uma maior segurança, como é o cenário do DSSAT. Islam e Zibran [Islam and Zibran 2017] atentam ao fato de que os resultados da *build* são afetados tanto pelo número de linhas alterados como pela quantidade de arquivos modificados, fato que provavelmente será observado também na IC aplicada ao DSSAT. A tendência observada por Zhao et al. [Zhao et al. 2017] de uma distribuição maior dos projetos e em mais *branches* pode se tornar uma característica do DSSAT, após a real aplicação da IC.

Alguns estudos relatam desafios na utilização da IC em diferentes ambientes. No estudo de Kauss et al. [Knauss et al. 2016], por exemplo, a fabricante de automóvel, não tem uma estrutura colaborativa interfuncional, de forma que as integrações entre ferramentas e modelos se tornam mais complexas. Outras pesquisas como a de Laukkanen et al. [Laukkanen et al. 2015], Stahl et al. [Ståhl et al. 2017], Hilton [Hilton 2016] relatam a influência da automatização dos testes na confiabilidade dos softwares e o comportamento do desenvolvedor relativo a frequência de *commits*. No caso do DSSAT é diferente, a IC automatiza um processo já realizado, tanto pelos desenvolvedores quanto pelos administradores do DSSAT, além disso, muitas execuções de testes mais intensivos são realizados até que o DSSAT chegue em nova versão oficial.

5. Metodologia

Para a aplicação da IC no DSSAT, inicicalmente, fez-se um estudo do processo de integração de novos códigos no DSSAT. Após, realizou-se uma análise comparativa de

várias ferramentas, entre elas, a Travis CI, a Circle CI e Jenkins. A seguir, os testes automatizados foram escritos utilizando o RCroptest, que é a API responsável pelos testes do DSSAT. A automatização dos testes aconteceu a partir da utilização de ferramentas de CI, também responsável por realizar a execução de *builds* privadas de cada momento que o desenvolvedor realiza um *push*, essa característica colabora também com que o desenvolvedor evite requisitar integração de códigos falhos.

Por fim, foram realizados os testes utilizando-se do Croptest (que é responsável por comparar versões do DSSAT apontando diferenças de resultados das simulações dos modelos). Para a comparação, o algoritmo compara o primeiro modelo, chamado Modelo A, e a última versão proveniente das alterações do desenvolvedor, chamada de Modelo B. Calculando a diferença relativa entre variáveis numéricas e de data considerando a porcentagem de diferença aceitável passadas como parâmetro (`thNumeric`) e (`thDate`), respectivamente [Nicolau 2018]. O procedimento calcula a diferença absoluta e o resultado dessa comparação é testado. Quando há diferenças, a execução é interrompida, e, uma mensagem indica que diferenças entre as versões dos modelos do DSSAT foram encontradas. O Github foi utilizado como repositório remoto e controle de versionamento pelo DSSAT. Um Docker privado foi construído a partir da escrita de um `Dockerfile`.

6. Resultados e Análises

A automatização da *build* e dos testes do DSSAT antes das atualizações do *branch develop* foi possível utilizando as três ferramentas de IC selecionadas e a estrutura base criada por meio do Docker. O container Docker com todas as dependências instaladas e configuradas para compilar e testar o DSSAT otimiza o processo de realizar a *build*. Se o Docker não fosse utilizado, tudo precisaria ser feito a cada *build*, com isso, demandando mais tempo.

Os desenvolvedores e administradores do DSSAT encontram no novo fluxo, que pode ser analisado na Figura 2, uma maneira diferente de obter os resultados das alterações do sistema. Neste fluxo observa-se que o início do processo é sempre quando o desenvolvedor executa uma das ações *push* ou *pull request* (1). Dentro do repositório do DSSAT existe o arquivo responsável por realizar a *build* e os testes (2), o qual aciona o funcionamento da Integração Contínua via a ferramenta (3). O Docker (4) executa todos os comandos do arquivo, e gera um resultado da compilação e dos testes (5) que pode ser visualizado na ferramenta (6). Dessa forma, o desenvolvedor acompanha a evolução das suas alterações a cada nova versão gerada na sua *branch* de trabalho. Além disso, o administrador do repositório tem menos trabalho para avaliar as alterações por conta da automatização.

Com esse cenário, os desenvolvedores e os administradores do DSSAT conseguem testá-lo no mesmo ambiente, evitando que as diferenças de instalações e configurações de um sistema ou de outro influenciem no resultado dos testes. Além disso, toda vez que o desenvolvedor faz um *push*, recebe o resultado da *build*, e esse processo gera alguns benefícios como: é realizado automaticamente, economiza tempo do desenvolvedor; o desenvolvedor provavelmente não fará um *pull request* enquanto a *build* apresentar erros, mitigando a possiblidade de suas aterações sejam negadas; e, por fim, saberá antes dos problemas antecipando as soluções.

As ferramentas de IC mostram o resultado da *build* de forma semelhante entre elas. Elas possuem uma lista com o histórico de todas as *builds* já realizadas. As ferramentas

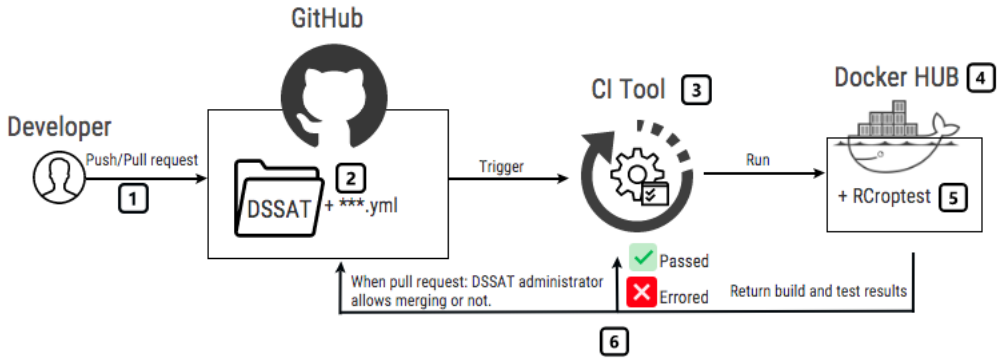

Figura 2. Fluxo da aplicação da Integração Contínua para automatização do processo de *build* e de testes de alterações do DSSAT.

fornecem o console, com todos os comandos executados e os retornos de cada um deles - importante para que o desenvolvedor possa encontrar erros e saber exatamente como a *build* se comportou. Além disso, elas apresentam um resumo de cada *build*, mostrando o nome da *branch*, se passou ou não, quanto tempo demorou para executar, entre outras informações.

Cada ferramenta utilizada possui algumas diferenças. O Travis CI e o CircleCI se assemelham na forma como são configurados, pois são baseados em nuvem e utilizam o arquivo YAML para definir os comandos da *build*. Ainda assim, configurar o CircleCI foi mais prático e rápido, porque possui uma documentação mais assertiva. Além disso, basta informar as credenciais do Docker, e indicar que ele será utilizado como serviço no arquivo YAML que o container definido é instanciado automaticamente.

O Travis CI possui uma particularidade interessante, pois mostra no GitHub se os testes tiveram sucesso ou não. O uso do Jenkins é trabalhoso por ser necessário realizar todas as configurações localmente ou em um servidor dedicado. Além disso, para todas as ferramentas necessárias para o funcionamento da Integração Contínua, um *plugin* precisa ser instalado e configurado - foi o caso do GitHub, do Docker e até mesmo do Pipeline. Esse processo deixa mais complexo o início do funcionamento da automatização. A vantagem é que o Jenkins possui mais possibilidades de personalização, porém para o propósito do DSSAT as outras ferramentas também mostraram-se suficientes. Em relação ao tempo de execução da *build*, o CircleCI apresenta o menor tempo, a ferramenta completa a *build* em torno de três a cinco minutos, o Travis CI leva por volta de seis minutos e o Jenkins onze minutos.

O objetivo de automatizar a *build* e realizar os testes do DSSAT, enquanto os desenvolvedores modificam o código em suas *branches* locais, e, também quando integram ao *branch develop* foi atendido plenamente pelas ferramentas de IC selecionadas por este trabalho. Para os administradores decidirem se aceitam ou não o *pull request* recebido, uma parte do trabalho já está pronta e, para a maioria dos casos, uma análise nas diferenças dos arquivos utilizando o GitHub será suficiente para a tomada de decisão, se aceita ou rejeita o novo código.

7. Conclusões

A aplicação de ferramentas de Integração Contínua no processo de versionamento do DSSAT traz uma mudança significativa no fluxo de trabalho dos desenvolvedores e dos administradores do sistema. Um processo que era manual, hoje está automatizado, agregando, com isso, uma série de benefícios e contribuindo com a evolução do DSSAT.

Para os desenvolvedores a mudança está principalmente no controle da *build* de cada versão enviada ao repositório remoto de trabalho. É possível obter o resultado de cada uma das *builds* utilizando a ferramenta de IC enquanto desenvolve e realiza um *push* a ferramenta compila e testa o que é essencial ao DSSAT. No momento em que algum erro é encontrado, o desenvolvedor sabe que precisa corrigir e conta com a ferramenta para saber qual problema ocorreu, consultando o console com o histórico de cada comando executado e os resultados.

Sem a automatização, para se obter o mesmo resultado, o desenvolvedor teria que compilar o DSSAT na versão que está alterando, fazer o download do DSSAT do *branch develop* atualizado e compilar também, para, somente após utilizar o Croptest ou o RCroptest para realizar a comparação dos modelos. Uma rotina longa e demorada que faz o trabalho render pouco, ou faz o desenvolvedor optar por não realizar todos esses passos a cada envio de código.

Atualmente, os administradores do DSSAT necessitam apenas compilar as duas versões do sistema - a que o desenvolvedor quer integrar e a do *branch develop* - e depois testar, com a automatização recebem esse resultado pronto. Também realizam algumas outras avaliações que consideram pertinentes, como olhar as diferenças entre os arquivos.

Outro ponto importante é que o desenvolvedor realiza os testes no mesmo ambiente que o administrador do DSSAT utiliza quando recebe um *pull request*. Esse fato evita que o desenvolvedor considere que suas alterações estão adequadas, mas, por conta de diferenças de configurações ou de dependências, os testes do administrador podem indicar algum problema. Este ambiente favorece que o administrador do DSSAT receba menos alterações com problema.

Entre as ferramentas, o TravisCI se destacou pela integração com o GitHub, rápido de começar a ser utilizado, além disso com o acesso via GitHub Education foi possível utilizá-lo sem restrições em um repositório privado. O CircleCI foi o mais rápido e fácil de começar, apresentou um tempo menor na execução da *build* e tem uma documentação completa e fácil de entender, porém possui uma restrição na versão gratuita referente ao tempo de execução de *build* por semana que pode ser um impeditivo para a sua utilização. O Jenkins precisa de muitas configurações e é o mais demorado para iniciar a utilização, possui documentações dispersas por conta dos *plugins* desenvolvidos por terceiros mas, todavia, é uma ferramenta com muita possibilidade de customização e é totalmente gratuita.

Como trabalhos futuros considera-se a implantação da IC em um recente repositório público do DSSAT. Também, esta técnica será aplicada em outros projetos, especialmente naqueles que contam com parcerias entre várias instituições.

Referências

Hamdan, S. and Alramouni, S. (2015). A Quality Framework for Software Continuous Integration. *Procedia Manufacturing*, 3:2019–2025.

Hilton, M. (2016). Understanding and improving continuous integration. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering - FSE 2016*, pages 1066–1067, New York, New York, USA. ACM Press.

Hilton, M., Tunnell, T., Huang, K., Marinov, D., and Dig, D. (2016). Usage, costs, and benefits of continuous integration in open-source projects. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering - ASE 2016*, pages 426–437, New York, New York, USA. ACM Press.

Islam, M. R. and Zibran, M. F. (2017). Insights into continuous integration build failures. In *Proceedings of the 14th International Conference on Mining Software Repositories*, MSR '17, page 467–470. IEEE Press.

Jones, J., Hoogenboom, G., Porter, C., Boote, K., Batchelor, W., Hunt, L., Wilkens, P., Singh, U., Gijsman, A., and Ritchie, J. (2003). The dssat cropping system model. *European Journal of Agronomy*, 18(3):235 – 265. Modelling Cropping Systems: Science, Software and Applications.

Knauss, E., Pelliccione, P., Heldal, R., Agren, M., Hellman, S., and Maniette, D. (2016). Continuous Integration Beyond the Team. In *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement - ESEM '16*, pages 1–6, New York, New York, USA. ACM Press.

Laukkanen, E., Paasivaara, M., and Arvonen, T. (2015). Stakeholder Perceptions of the Adoption of Continuous Integration – A Case Study. In *2015 Agile Conference*, pages 11–20. IEEE.

Mossige, M., Gotlieb, A., and Meling, H. (2015). Testing robot controllers using constraint programming and continuous integration. *Information and Software Technology*, 57:169–185.

Nicolau, M. (2018). CROPTEST : data-drive test automation for crop modeling systems.

Ståhl, D., Mårtensson, T., and Bosch, J. (2017). The continuity of continuous integration: Correlations and consequences. *Journal of Systems and Software*, 127:150–167.

Zhao, Y., Serebrenik, A., Zhou, Y., Filkov, V., and Vasilescu, B. (2017). The impact of continuous integration on other software development practices: A large-scale empirical study. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 60–71.