

UNIVERSITY OF PASSO FUNDO
INSTITUTE OF EXACT SCIENCES AND GEOSCIENCES
GRADUATE PROGRAM IN APPLIED
COMPUTING

**CROPTEST: Data-driven test
automation for crop modeling
systems**

Marcio Nicolau

Passo Fundo

2018

**UNIVERSITY OF PASSO FUNDO
INSTITUTE OF EXACT SCIENCES AND GEOSCIENCES
GRADUATE PROGRAM IN APPLIED COMPUTING**

**CROPTEST: DATA-DRIVEN
TEST AUTOMATION FOR CROP
MODELING SYSTEMS**

Marcio Nicolau

Thesis submitted to the University of
Passo Fundo in partial fulfillment of the
requirements for the degree of Master in
Applied Computing.

**Advisor: Prof. Dr. Willingthon Pavan
Coorientador: Prof. Dr. José Maurício Cunha Fernandes**

Passo Fundo
2018

CIP – Cataloging in Publication

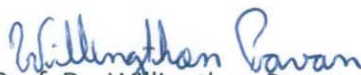
- N639c Nicolau, Marcio
CROPTEST: data-driven test automation for crop modeling systems / Marcio Nicolau. – 2018.
64 f. : il. color. ; 30 cm.
- Advisor: Prof. Dr. Willingthon Pavan.
Co-advisor: Prof. Dr. José Maurício Cunha Fernandes.
Thesis (Master in Applied Computing) – University of Passo Fundo, 2018.
1. Crop modeling. 2. Simulation methods 3. Data-driven test. 4. Computer simulation. 5. Agricultural informatics. I. Pavan, Willingthon, advisor. II. Fernandes, José Maurício Cunha, co-advisor. III. Title.

CDU: 631:004
004:631

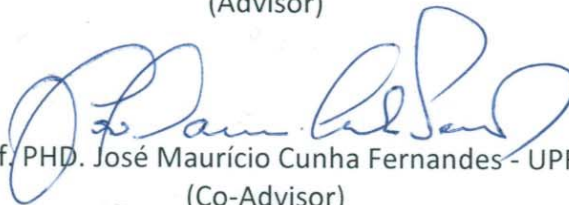
**MINUTE OF
ACADEMIC COURSE CONCLUSION WORK**

MÁRCIO NICOLAU

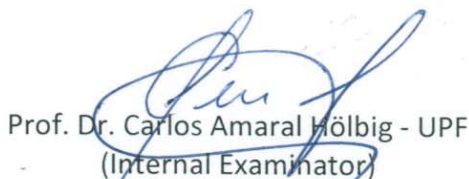
On the twenty-eighth day of March two thousand and eighteen, at 3:00pm BRT, on the Institute of Exact Sciences and Geosciences, building B5 of the University of Passo Fundo, was hold the public defense session of the Course Final Work "**CROPTEST: Data-driven test automation for crop modeling systems**" authored by Márcio Nicolau, academic of the Graduate Program in Applied Computing - PPGCA / UPF. According to information provided by the Postgraduate Council and listed in the archives of the PPGCA Secretariat, the student fulfilled the necessary requirements to submit his work to be evaluated. The examination board was composed by Dr. Willington Pavan, PhD. José Maurício Cunha Fernandes, Dr. Carlos Amaral Hölbig and PhD. Gerrit Hoogenboom. After the presentation and arguing, the examining board considered the candidate APPROVED. A period of up to forty-five (45) days, according to the Rules of the PPGCA, was granted for the academic to submit the final writing to the Postgraduate Council, in order to make the necessary referrals for the issuance of Master in Applied Computing Diploma. To record, this minute was drawn up, which are signed by the members of the examination board and by the PPGCA Coordinator.



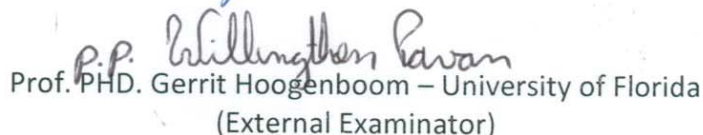
Prof. Dr. Willington Pavan - UPF
Examining Committee President
(Advisor)



Prof. PHD. José Maurício Cunha Fernandes - UPF
(Co-Advisor)



Prof. Dr. Carlos Amaral Hölbig - UPF
(Internal Examiner)


P.P. Willington Pavan
Prof. PHD. Gerrit Hoogenboom - University of Florida
(External Examiner)



Prof. Dr. Rafael Rieder
PPGCA Coordinator

ACKNOWLEDGMENTS

I would first like to thank my thesis advisor Prof. Dr. Willingthon Pavan of the Graduate Program in Applied Computing at the University of Passo Fundo. The communication to Prof. Pavan was always open whenever I ran into a trouble spot or had a question about my research or writing. He consistently allowed this thesis to be my own work but steered me in the right the direction whenever he thought I needed it.

I also would like to thank my thesis co-advisor Prof. Dr. José Maurício Cunha Fernandes of the Graduate Program in Applied Computing at the University of Passo Fundo and co-worker at Brazilian Agricultural Research Company (Embrapa) for all help, support, suggestions, and discussions about DSSAT and Crop System Models.

In the same way I would like to acknowledge the main developers of DSSAT Foundation for all their support, suggestions and ideas exchanged to improve the usability and functionalities to the future users of CROPTTEST. During all development time and at our regular meetings, I could better understand and realize the importance of crop model systems for agriculture.

Many thanks to DSSAT Foundation for all financial support of my master at the University of Passo Fundo and by sponsoring my participation on DSSAT official training workshop at University of Georgia – Griffin Campus (USA).

Finally, I must express my very profound gratitude to my parents Osvaldo and Diva, to my spouse Claudia and to my daughter Alice for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them.

Thank you.

Marcio Nicolau

CROPTEST: AUTOMAÇÃO DE TESTES COM BASE EM DADOS PARA SISTEMAS DE MODELOS DE CULTURAS

RESUMO

O presente trabalho descreve o desenvolvimento de uma nova versão do CROPTEST que aborda a problemática de comparar diferentes versões do DSSAT-CSM usando procedimentos automatizados. A nova versão expande a anterior pelo uso de mais medidas estatísticas aplicadas aos arquivos de saída, definido como o procedimento principal para comparação com base em arquivos de entrada predefinidos - arquivos de execução em lote e configurações de definição de experimentos - que geram arquivos com estrutura conhecidas denominada metodologia de teste por dados (DDT), área da engenharia de software. O *Agile scrum* foi adotado como metodologia de desenvolvimento. A nova interface do usuário e todas as funcionalidades foram projetadas para maximizar a experiência e a produtividade do usuário. O código final foi escrito usando tecnologias modernas para desenvolvimento Web as quais se mostraram úteis e responsivas ao desenvolvimento desktop usando bibliotecas e frameworks combinados especificamente para este objetivo. Uma Interface de Programação de Aplicação (API - *Application Programming Interface*) foi projetada e implementada para ser a fonte principal de recursos para extração de dados, análise e cálculos, é descrita em detalhes e, além disto, é apresentado como configurar seu uso e anexá-la a uma Plataforma de Serviços (PaaS - *Platform as a Service*) como Travis-CI ou Circle-CI, executando um subconjunto de todos os testes de culturas existentes no DSSAT para verificar a consistência e a melhoria do código proposto em relação à última versão estável armazenada no GitHub. Esta versão *Desktop* possui procedimentos de instalação automáticos definidos para os três principais sistemas operacionais - Windows, Linux e MacOS - o que torna o CROPTEST uma ferramenta multi-plataforma para teste de modelos CSM.

Palavras-Chave: API, CROPTEST, DSSAT, Integração Contínua, Modelos de Crescimento de Cultura, Testes Automatizados, Testes de Software baseado em Dados.

CROPTEST: DATA-DRIVEN TEST AUTOMATION FOR CROP MODELING SYSTEMS

ABSTRACT

The present work describes the development of a new version of CROPTEST that address the problem to compare different versions of DSSAT-CSM using automated procedures. The new version expands the current one adding more statistical analysis applied to output files and, using a methodology called data-driven test (DDT) from software engineering that use a fixed set of input files and compare the output results – in this case, the batch files and experiments settings. Agile scrum was adopted as the development methodology. The new user interface and all operation pattern were designed to maximize the user experience and improve their productivity. The final code was written using modern JavaScript web technologies which evidence useful and responsive to desktop development, using libraries and frameworks combined for this goal. The CROPTEST API, implemented and design to be the core source for parsing and calculations procedures, is described in detail and it showed how to set up a configuration to attached it to a PaaS (Platform as a Service) like Travis-CI or Circle-CI. In this setup is possible to run a subset of the all crop tests to check the consistency and improvement of the proposed code in relation with last stable version stored on Github master branch. The desktop version was tested and have automatic installation procedures already defined for the three major OS – Windows, Linux, and MacOS – making the CROPTEST a multi-platform tool for test CSM models.

Keywords: API, Automated Tests, Continuous Integration, Crop System Models, CROPTEST, Data-Driven Test, DSSAT.

LIST OF FIGURES

1	Snapshot of Analysis Form for experimental mode from the first CROPTEST version [10].	25
2	DDT approach. Schematic representation of test flow, figure adapted from [3].	26
3	DDT input file data example. This one was designed using a spreadsheet program and provide a file in rectangular form, figure adapted from [3] . . .	27
4	Comparison between a development work-flow with and without automated continuous integration. Steps highlighted (Automation) are candidates to automation using local or on-line tools.	28
5	Main steps to build, test and deliver a new software version using GitHub and Travis-CI. The interaction between PaaS services are based on programmable webhooks.	29
6	Summary type spreadsheet with data showing the most distinct variable and statistics for HWAH, MDAT, SWXM.	30
7	Comparison type spreadsheet with data showing differences between variables (based on selected threshold) for each CSM model.	30
8	The CROPTEST work-flow showing the relationship between DSSAT-CSM, User Interface an API to run, parse, calculate and show the statistics for the user.	42
9	Snapshot of main window operations for the new CROPTEST. In detail, the <u>Run</u> step, that permits the user to run CSM models from User Interface and observe the results and running status.	44
10	Snapshot of main window operations for the new CROPTEST. In detail, the <u>Run</u> step, showing the parse and load content of Batch file inside “Batch File Content” panel.	45
11	Snapshot of main window operations for the new CROPTEST. In detail, the <u>Run</u> step, showing the parse and load content of Batch file inside “Batch File Content” panel.	45
12	CROPTEST in detail, the <u>Run</u> step, showing the parse and load content of <u>Warningfile</u> inside “Warning/Error Output” panel.	46
13	Snapshot of main window operations for the new CROPTEST. In detail, the <u>Analysis</u> step, that permits the analyze the CSM models output for summary and evaluate files. Image details for summary output file.	47

14	In detail, the <i>Analysis</i> step, that permits the analyze the CSM evaluate output files. Each column shows the statistics for both models side-by-side to make the process of decision easier.	48
15	Workflow and related key steps necessary to test a new CSM version hosted on GitHub with environment allocated on Travis-CI.	55
16	The CROPTEST API work-flow showing the relationship between CSM output files, parser procedure, statistics calculations and results output to the user and for shell interaction (CI Platform).	57

LIST OF TABLES

1	Summary of the key Web technologies and their inter-relationship used to develop the new CROPTTEST Desktop Application.	41
2	CROPTTEST Desktop Application basic folder structure. Some settings could be adjusted and persisted between executions, others are fixed and defined on first run. The main folder is defined CTWork from last DSSAT version installed on the user computer.	43
3	CROPTTEST partial output (min , avg and max) of relative differences between CSM versions 4.6 and 4.7. A subset of all summary output variables are randomly selected for crops Cassava, Maize, Peanut, Potato, Rice, Soybean and Wheat.	49

LIST OF ABBREVIATIONS

DDT. – Data-driven Tests

CSM. – Crop System Models

DSSAT. – Decision Support System for Agrotechnology Transfer

APSIM. – Agricultural Production Systems Simulator

FORTTRAN. – Formula Translation

PaaS. – Platform as a Service

API. – Application Programming Interface

IBSNAT. – International Benchmark Sites Network for Agrotechnology Transfer

HWAH. – Harvest weight at harvest (kg/ha)

MDAT. – Maturity date (days or date)

SWXM. – Extractable water at maturity (mm)

CI. – Continuous Integration

OSS. – Open Source Software

SRAD. – Solar Radiation (MJ/m²/d)

AI. – Agreement Index

ME. – Model Efficiency

MAE. – Mean Absolute Error

RMSE. – Root Mean Squared Error

nRMSE. – Normalized Root Mean Squared Error

JSON. – JavaScript Object Notation

TDD. – Test-Driven Development

CONTENTS

1	INTRODUCTION	19
2	BACKGROUND	23
2.1	DSSAT	23
2.2	CROPTEST	23
2.3	QUALITY IN SOFTWARE DEVELOPMENT	24
2.3.1	Data-driven test	26
2.3.2	Agile Scrum	27
2.4	CONTINUOUS INTEGRATION	27
2.5	STATISTICAL METHODOLOGIES	29
2.6	TOOLS AND TECHNOLOGIES	32
3	CROPTEST DESKTOP	37
3.0.1	Introduction	37
3.0.2	Background	39
3.0.3	Material and Methods	40
3.0.4	Results and Discussion	43
3.0.5	Conclusions	50
4	CROPTEST API	51
4.0.1	Introduction	51
4.0.2	Background	52
4.0.3	Material and Methods	53
4.0.4	Results and Discussions	55
4.0.5	Conclusions	58
5	CONCLUSIONS AND FUTURE WORKS	59
	REFERENCES	61

1. INTRODUCTION

The task to create and maintain a software requires time, resources and effort from a development team and, based on all investment it is necessary to have some way to assure the quality and usability of the resultant software that is delivered to customers.

Use of methods and procedures during software development that guarantees the minimum set of quality for the user is a desired requirement for all company or development team. From software engineering, the most common method to do test avoiding wrong operations results and calculations, and improving the product quality is known as a unit test [1, 2].

Unit test implements localized checkout at the smaller part of the systems – functions or procedures – and, in some cases, could stack them together. As a result, if the smaller parts passed in the test and the whole system shows some problems or wrong behavior, the problems could be delegated to an aggregation process.

Even though this method has been working well in a variety of software and systems, some of them do not obtain all benefits of a unit test as main methodology, especially ones that were developed for legacy systems.

An alternative solution is a use of Data-driven tests (DDT), a software engineering procedure that use external data to compare output results and measure the quality and performance of software code [3, 4].

A case of complex and legacy system development could be found in Crop System Models (CSM) that is used from a long time to simulate the dynamic and interaction between plant growing components (soil-atmosphere-water-nutrients).

Those dynamic simulation models are used to evaluate and predict yield from crops, especially ones with economic and food security influence - like maize, soybean, and wheat – just as few examples. In this context, CSM software like DSSAT [5, 6] and, APSIM [7] is an example of complexity levels that this type of software could archive.

The DDT evidently provided a good solution for the problem of evaluating CSM software quality in the scenario that already has multiple sets of data input, simulation configurations, and field experiments data for each crop. Indeed, this rich test set generates output files from simulation, and these could be the main part of the comparison procedure.

Besides all complexity that is common and intrinsic to CSM, another important point to consider is the computer language or set of them, used to develop the whole system. Some languages, like FORTRAN, shown more difficult to implement software test, when this one only centered on the unit test.

The choice and use of FORTRAN as main development language earlier could now cause some problem to adopt modern techniques for testing procedures and quality

software assurance. On DSSAT-CSM case, the necessary effort and time to implement this for all modules and procedures could be prohibitive, the actual complexity and inter-relationship between core modules and crop systems could induce a low convergence and overall quality [8], and there was still a need for full-scale acceptance testing [9].

As a reference, the CSM version 4.7 from DSSAT provides 42 different crop models with the possibility to create scenarios and crop rotation simulation for several years in a row. In this context, the development, maintenance, and test become even more crucial and time-consuming task for the DSSAT development team.

There are two main problems to face at this point, how to proceed with test and quality software assurance in a suite of models that was not planned to accommodate this kind of procedure without changing the main code. Another is which will be the best practice that could permit the design of an automated testing system which could cover most of the core modules and integrated crop systems with low cost and minor time effort to implement?

As an attempt to solve those problems, DSSAT development team create a software with a predefined protocol to test a reduced and fixed number of model output files and observe the relative differences archived between two CSM versions, a stable and next release one. This tool, called CROPTTEST, was developed using a language which now does not have vendor support (since 2016) and, in the other hand, offers a minimal set of options to the developer test the quality of the software between versions.

This first CROPTTEST version helps to solve an immediate problem related to quality software test, but the evolution and growing complexity of CSM demanded the development of a new version with improved functionalities and desired performance, like one related to the possibility to run on multiplatform and work with more files and work variables.

The main objective of this work is to create a new CROPTTEST version, running both in desktop and in online infrastructures nominated Platform as a Service (PaaS), using DDT methodologies and modern web development technologies, libraries and frameworks for best user experience and usability.

This CROPTTEST is being utilized as a testing system by DSSAT development team to improve the CSM functionalities, comparing the results between releases versions and for development and incorporation of new crop models. Other efforts are related to prototyping and implementation of a generic parser to be used as an API for both Desktop and PaaS versions, that cover all types of tests and comparison for more model output files.

To present all steps used to develop the present work, this document is divided in chapters and each one detail the main phases as follow, in Chapter 2, a background of methods, technologies, tools, and literature review is presented. In Chapter 3, the toolset used for development and a sample of the Desktop interface usability are presented. In addition, the improvements and new functionalities of the first CROPTTEST version are detailed.

Chapter 4, details the development of the API used for parsing and comparison procedure for both Desktop and PaaS versions. Also, a minimal PaaS setup is presented to the user. Chapter 5, present the summarization of the steps and lessons learned during the development of this new CROPTTEST version, also presents some future works for the project.

2. BACKGROUND

This chapter defines the concepts, methods, and technologies investigated during the development of the new version of CROPTEST. A literature review of related works and necessary knowledge used as a base foundation for this propose developments are detailed in the following sections.

2.1 DSSAT

The DSSAT started as an international network of cooperating between scientists to facilitate the application and use of crop models in agronomic research [5]. In the early development, the main goal was integrated knowledge soil, climate, crops, and management for a better decision; this network was called IBSNAT [5].

In 2003, DSSAT incorporated 16 different crops [6] and a common layer of information shared by all crops, composed by a soil module, a crop template, a weather module. As well, it was developed as a module to evaluate the water and light competition among trilogy soil, plant and atmosphere.

In that same year, a structural change occurred to facilitate maintenance and allow easy replacement or addition of modules and crops, adjusting a culture model module and a species input file; this change generated the CSM for the new DSSAT, which is now called DSSAT-CSM [6].

The DSSAT-CSM is all writing in FORTRAN language that works mainly with files to transfer information between CSM modules. It is a fact that FORTRAN is the *lingua franca* for scientific codification in works that extensively use mathematical formulation, like crop model system.

In 2017, the CSM version 4.7 from DSSAT offer to the users 42 crop models. In this context, the development, maintenance, and test become even more crucial and time-consuming task for the main team responsible for main DSSAT code.

2.2 CROPTEST

The DSSAT Model Comparison Utility (CropTest) was developed with the purpose to compare the output of two versions of a DSSAT crop model, using data from the summary files. The first version was designed for users with DSSAT models familiarity and knowledge to run them from batch mode simulations [10].

To develop this version, the Visual Basic 6.0 from Microsoft Corporation was used as main language and development environment. This CropTest has been able to make calls to the DSSAT crop model binaries. The main idea was to measure the relative difference inside summary output files from a typical DSSAT generate by two sets of source codes (stable and development).

The main characteristics of this version are related to the use of a fixed structure to enable to run; the comparison is conducted using a templated spreadsheet file and the composed batch files used to get the output from models, with a possibility to be generated manually or using the proper DSSAT interface.

The comparison could be carried out selecting from two analysis mode: (a) Experiment and (b) Sensitivity Analysis. Each one with different meanings and specific options [10]. In experiment mode, four simulation options are available: (i) Model 1 only, (ii) Model 2 only, (iii) Import results and compare and, (iv) Run both models, import and analyze) are available.

Sensitivity mode allows an additional option when the sensitivity analysis was setup on analysis mode. All simulation results and analysis from experimental mode are presented inside one spreadsheet, containing summary and comparison information alongside with data from models (1 and 2) output.

The user could define a threshold value (2% default) used as a cut point for highlighting differences between models. Any differences between the two models more than this threshold are highlighted on comparison and summary. All the dates in comparison have a fixed threshold of zero. Any change in the threshold to evaluate the models in a different context or situation, did not automatically update the resulting spreadsheet, compelling the user to run the comparison procedure again.

The user interface is the same used by main DSSAT-CSM (Figure 1), and it was designed to run as an add-on or plug-in for main DSSAT interface. It is necessary to both versions of codes installed as a binary file and, proceed with setting for maximum threshold accepted and source path for binary input and output from models. Both models run with the same input files [10].

For the comparison purpose between the models, this first version uses a small set of statistics to support the comparison. The average and maximum changes in HWAH, MDAT and SWXM; also it reports the maximum deviation between models, describing the variable, maximum value, observed values from model 1 and 2, simulation run an experiment (file X) among other informations [10].

2.3 QUALITY IN SOFTWARE DEVELOPMENT

Developing a software with quality means to define a set of procedures for use by organizations or community group to ensure that a software will meet its quality goals at the

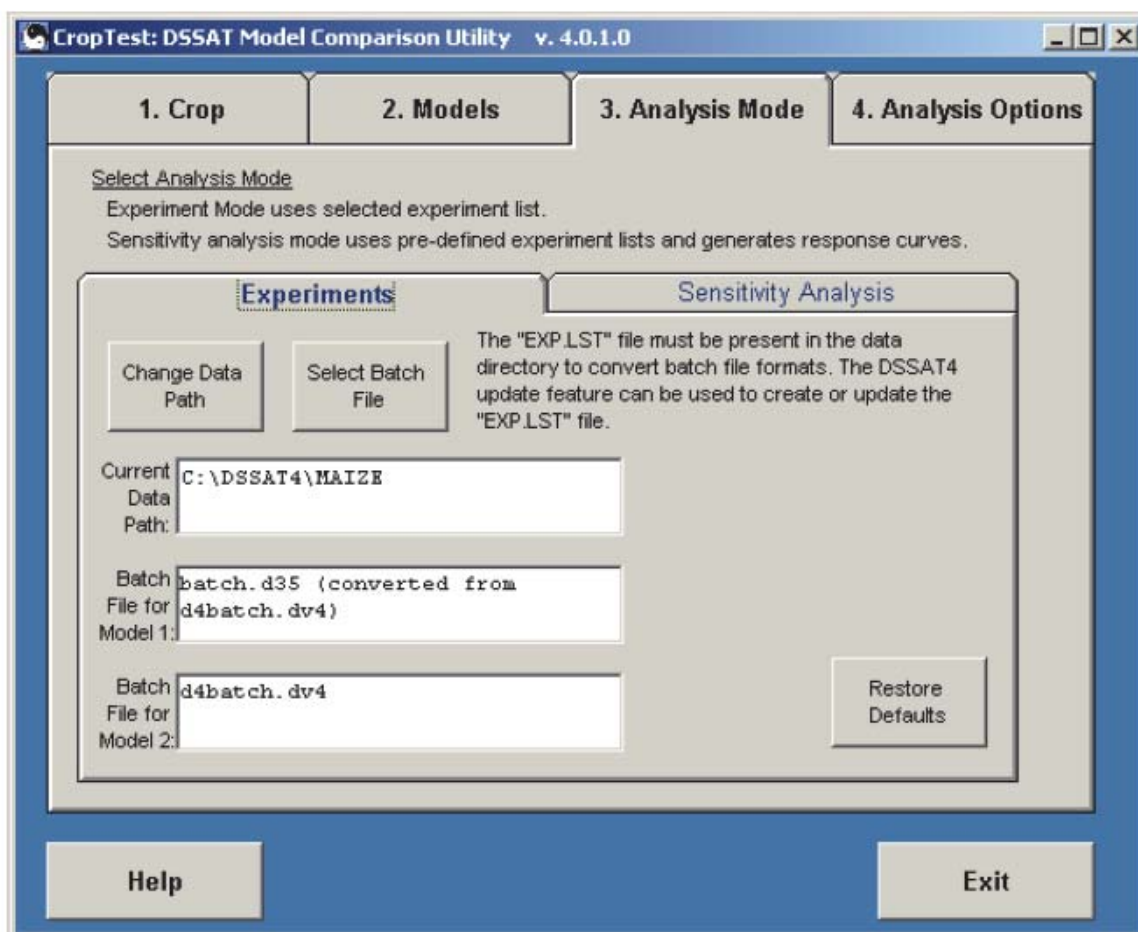


Figure 1: Snapshot of Analysis Form for experimental mode from the first CROPTTEST version [10].

best value to the customer, and to continually improve the organization's ability to produce software products in the future.

Tsai et al [1] advocate that when software testing is performed at the end of the coding phase, the quality could be compromised because of the due date, and so, verification and validation (i.e. *software test*) should be performed during all development phases.

In a certain way, the development and test should be inseparable and interconnected steps to support a better software delivery. However, in some cases, programmers without computer science background carry out the development and the most common tests applied in this context are related to the correctness of mathematical formulation and output results, based on some reference dataset recorded from the scientific or observational experiment.

Wilson et al [11] present an interesting view about software development by scientists and how they should improve the quality of the software used in science. While most scientists are careful to validate their laboratory and field equipment, most do not know how reliable their software is.

Unfortunately, the use of software engineering methods for quality software tests applied specifically to CSM is not well described and to the best of author's knowledge and thorough search of the relevant literature yielded no related work describing this approach.

2.3.1 Data-driven test

One way for testing software could be the use of scripts, the most common use of this approach is to put the input data together with code, and this approach could generate problems when those data need changes.

On the other side, when the software pass through structural changes a new version of all scripts (code and data) must be produced, and this scenario create a difficult generalization of test process structure (recycling code).

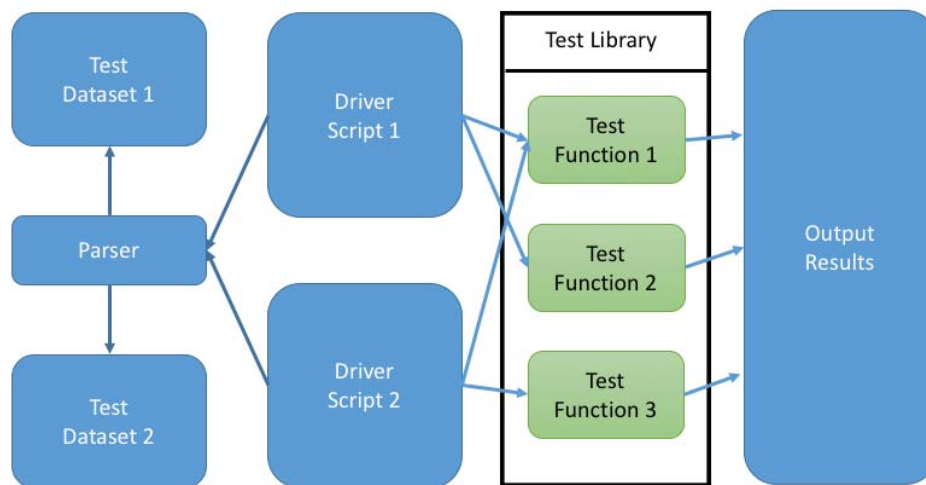


Figure 2: DDT approach. Schematic representation of test flow, figure adapted from [3]

A possibility to solve this kind of problem is to use external data sources as a source for test execution [3]. This is the core of DDT as illustrated in Figure 2. An advantage to using this form of software testing is that the data could change easily and can evolve as the software increases its functionalities.

Another advantage is that any person with knowledge from crop could generate a new dataset for the test, without any programming skills. It is often in tabular format and edited in spreadsheet programs, as seen in Figure 3. In resume, the data-driven approach is the key for ease-of-use in large-scale test automation.

	A	B	C	D	E
1	Test Case	Number1	Operator	Number2	ExpectedOutput
2	Add 01	1	+	2	3
3	Add 02	2	+	-2	0
4	Sub 01	3	-	1	2
5	Sub 02	1	-	-2	3
6	Mult 01	2	*	2	4
7	Mult 02	3	*	-2	-6
8	Div 01	4	/	1	4
9	Div 02	6	/	-2	-3
10					

Figure 3: DDT input file data example. This one was designed using a spreadsheet program and provide a file in rectangular form, figure adapted from [3]

2.3.2 Agile Scrum

The Agile Scrum [12] methodology advises that all requests for resources, user requirements, bug reports and other issues be captured and kept in a prioritized backlog. Each scrum team has a backlog product owner responsible for managing and prioritizing deliveries for each cycle. The project manager works closely with backlogs owners to provide management and prioritization of global products.

The development performs in time-box rounds, typically two to three weeks in duration. Within each round, a management team prioritizes and agree to the items of the backlog of interest for that round.

Once scoped and agreed, the team is free to complete the work in the best way. At the end of each round, the scrum team shows progress in the use cases and requirements articulated by stakeholders.

This way of development generally delivers the most important features early to the client, and this maximizes the returned value each development round and could be affordable inclusive for small development team [13].

2.4 CONTINUOUS INTEGRATION

Currently the software development, in most cases, goes across similar steps related to design, implementation, build and test. Fortunately, some of those steps could be automated using tools and online services (PaaS) to help the developer (individual or team) to have more time to do what is important – write better software.

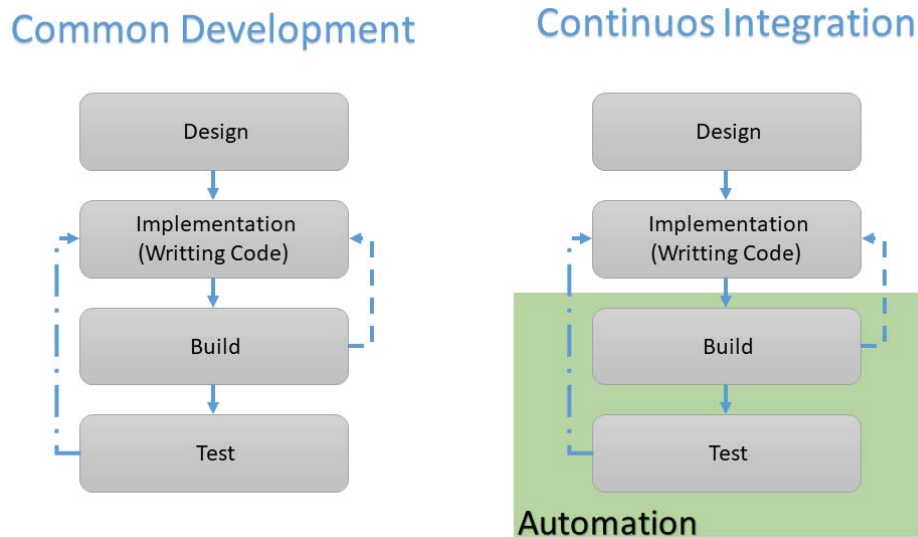


Figure 4: Comparison between a development work-flow with and without automated continuous integration. Steps highlighted (Automation) are candidates to automation using local or on-line tools.

Similarly, this new work-flow development using continuous integration associated with continuous delivery could fill the gap between software development and the access for this software by those users as soon as possible and with more quality [14].

The automation could happen on the highlighted steps from Figure 4. Those ones could be setup to run using *script files* for local settings or with *configuration files* and *web-hooks* for online.

The Continuous Integration (CI) is the most common step of the software life-cycle is the build release. In fact, CI tools improve the management of the process substantially and accelerate the time between development and test process.

Fitzgerald et al [14], describe that using CI/CD the developer could be closer from costumers without delays from development and test to deliver the solution, this affirmative is valid for all types of development and delivery, including web-pages, Internet portals, Desktop and Mobile Applications.

Travis CI [15] is a distributed continuous integration hosted system service used to build and test software, especially ones hosted on GitHub [16] platform. In general, open source projects may run with no charge fee.

Along with the existence of other vendors offering the same type of integration and services like Visual Studio Team Service¹, Circle-CI², GitLab³ and AppVeyor⁴; the author choice for Travis-CI and GitHub are related to the use of a student development pack⁵ offered

¹<https://www.visualstudio.com/pt-br/team-services/>

²<https://circleci.com>

³<https://about.gitlab.com>

⁴<https://www.appveyor.com>

⁵<https://education.github.com/pack>

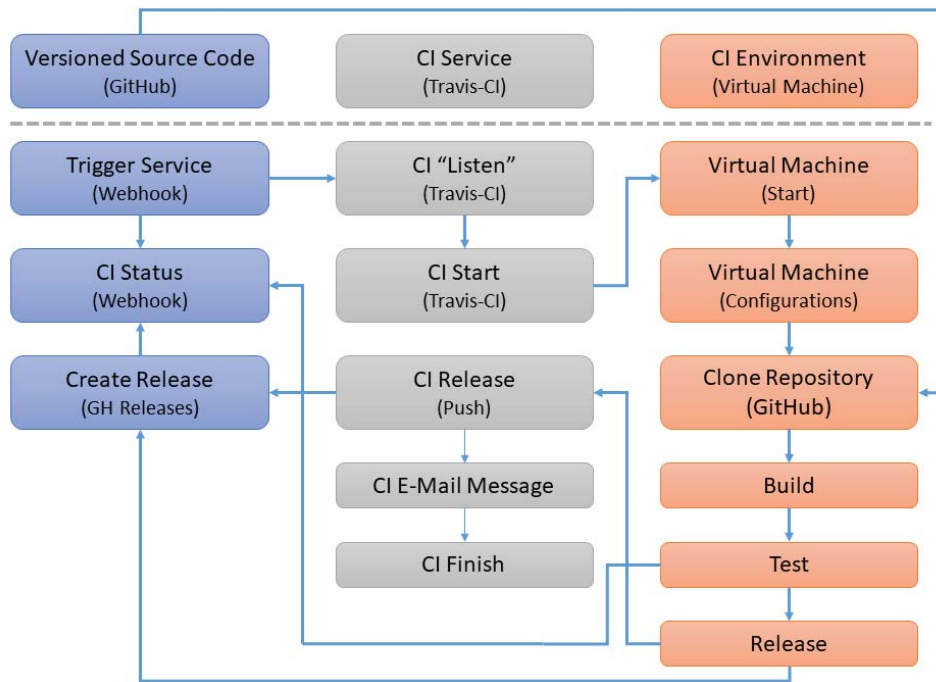


Figure 5: Main steps to build, test and deliver a new software version using GitHub and Travis-CI. The interaction between PaaS services are based on programmable webhooks.

by GitHub that includes full access to private repositories and private build services from Travis-CI valid during this project development.

The key steps related to the overall process integration is presented on Figure 5. The procedure is cycled and is triggered on specific check-points from development on developer own computer with updates carried out by webhooks and scripts from both GitHub and Travis-CI.

Deshpande and Riehle [17] evaluates the adoption of CI of agile software development on Open Source Software (OSS), and as a result, they present that open source software community still does not change their CI practices and neither influenced OSS developers at all.

2.5 STATISTICAL METHODOLOGIES

An important part of DDT method for the new version is to find mathematical formulations that endorse the gain in performance when comparing models. One common way to prove scientific evidence based on data is related to the application of statistical methodology.

The intent of this section is to expose the statistical formulation used by both versions of CROPTTEST to compare the results between CSM versions. Alongside with the main uses for each statistics, the formulation and references are provided when necessary, and this decision is based on common sense knowledge in the scientific community.

includes graphical reports for temperature offset, fixed temperature, fixed SRAD, fixed CO₂, rainfall multiplier, daylength offset and planting date.

Sensitivity analyses are performed to reveal the response to yield of various environmental, management or other stimuli. A pre-defined set of input data files have been prepared for each crop for use in sensitivity analysis without possibility to expand or modify these data sets [10].

Analysis on Version 2

The new version (v2) extend the number of statistics used to measure the relative performance between CSM models. Also, it works now on two types of output files – summary and evaluate – from simulations to archive this goal.

For the summary output file, the statistics used for summarization and comparison between crop models outputs are the detailed below [18]:

- **Number of Outliers:** Calculate the number of experiments that have abnormal or extreme value. Correspond to approximately 95% IC for the median. Formula: $1.5 \times IQR + [-Q1, +Q3]^6$.
- **Mean:** Calculate the average value for numeric column data. Formula: $\bar{x} = \sum x/n$.
- **Maximum:** Return the maximum value for numeric column data. Formula: $max(x)$.
- **Minimum:** Return the minimum value for numeric column data. Formula: $min(x)$.

For the evaluate output file, another set of statistics are used for comparison between models which are related to model performance and related measures [19]. A brief explanation alongside with the mathematical formulation are presented below:

- **Agreement Index (AI):** Is a measure of model accuracy, evaluating a cross-comparisons between simulated and observed data. Formula: $1 - \sum (y_i - x_i)^2 / \sum (|y_i - \bar{x}| + |x_i - \bar{y}|)^2$.
- **Model Efficiency (ME):** Measure of the predictive power of a simulated model. Formula: $1 - \sum (y_i - x_i)^2 / \sum (x_i - \bar{x})^2$.
- **Mean Absolute Error (MAE):** Measure of difference between two continuous variables. Formula: $\sum |y_i - x_i| / n$.
- **Root MSE (RMSE):** Measure of how spread out the residuals are. It tells you how concentrated the data is around the line of best fit. Represent the standard deviation of the prediction errors. Formula: $\sqrt{\sum (y_i - x_i)^2 / n}$.

⁶IQR: Inter Quartile Range / Q1, Q3: Quantiles 1 (25%) and 3 (75%)

- **Normalized Root MSE (nRMSE):** Express percent of the variation of the standard deviation of the prediction errors around the mean. Formula: $RMSE/\bar{x} \times 100$.
- **R Squared (R^2):** “Goodness-of-fit” for linear models. Measure the linear association (adherence) between x and y. Formula: $\left(\frac{\sum[(x_i - \bar{x})(y_i - \bar{y})]}{\sqrt{\sum(x_i - \bar{x})\sum(y_i - \bar{y})}} \right)^2$.

2.6 TOOLS AND TECHNOLOGIES

This section lists the key tools and technologies used to develop the Desktop version and the API for the new CROPTTEST. The selection of technologies is guided by three principles: (a) facility provided by the tools related which each one them to develop a software with high quality standard and that permits the setup for test the software during all phases of the development; (b) the development environment and setup make it easy to use with CI/CD services, and (c) the software maintenance could archive long term support, preferably via OSS developer community.

The developer community established around web technologies is one of the most engaged and supportive among others. Also, the applications of this kind of development expanded from web environment and now permit development for growing number of scenarios like web, mobile, desktop and edge devices. Based on former experience from the author and the rich set of tools, methods, and technologies, the natural choice is Javascript.

For the API tools, and related with the context to read, treat and analyze numerical data, the most popular choices could be Python⁷ or R⁸. Consequently, using the same principles related to desktop choice and, once more, based on previous author expertise, the natural direction for this type of development is the R software.

CROPTTEST Desktop

Based on arguments already exposed and, associated with the necessity to run this new Desktop version on multiple platforms, the use of modern web technologies presents a potential increase in productivity, usability and long-term maintenance.

The information related to each tool or technology and their inter-relationship, when this occurs, is presented as follows:

- **Electron:** Is an open source library developed by GitHub for building cross-platform desktop applications with HTML, CSS, and JavaScript. Electron accomplishes this by combining Chromium and Node.js into a single runtime and apps can be packaged for

⁷<https://www.python.org>

⁸<https://www.r-project.org>

Mac, Windows, and Linux. It began in 2013 as the framework on which Atom, GitHub's hackable text editor, would be built. The two were open sourced in the Spring of 2014.

Electron permits the development of Desktop Application using modern web technologies incapsulating it inside a Chromium environment but with access to computer resources like disk and memory. [20]. Source: <https://electronjs.org/>.

- **Webpack**: At core it is a *static module bundler* for modern JavaScript applications. When processes an application, it internally builds a *dependency graph* which maps every module your project needs and generates one or more *bundles*. It permits the use of loaders and plugins to extend the core functionalities, like the addition of the hot module replacement, transpiling (plugin) and splitting generated output. It is also capable to bundle and minify application for distribution. Source: <https://webpack.github.io/>.
- **BabelJS**: It is a JavaScript compiler that enable developers to use and implement Javascript code using next generation standard (with new functionalities) and deploy them to use with standard that actual browsers could understand and execute. As example, it could be used to covert from *draft* ECMAScript version to the one that all modern Web Browser could execute. Source: <https://babeljs.io/>.
- **Node.js**: A JavaScript runtime built on Chrome's V8 JavaScript engine. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. As an asynchronous event driven JavaScript runtime, Node is designed to build scalable network applications. It is similar in design to, and influenced by, systems like Ruby's Event Machine or Python's Twisted. It takes the event model a bit further. It presents an event loop as a runtime construct instead of as a library. In other systems there is always a blocking call to start the event-loop [21]. Source: <https://nodejs.org/en/>.
- **Semantic-UI**: Semantic UI React provides the JavaScript for your components. Also includes a stylesheet to provide the styling for your components. This is the typical pattern for component frameworks, such as Semantic UI or Bootstrap. It is used by companies like Amazon, Netflix and Microsoft to create and deliver their products. It is Javascript User Interface Framework based on Semantic-UI CSS library. Source: <https://react.semantic-ui.com/>.
- **ReactJS**: Is a declarative, component-based JavaScript library used for building user interfaces [22]. Declarative views make your code more predictable and easier to debug. Build encapsulated components that manage their own state, then compose them to make complex UIs. Since component logic is written in JavaScript instead of templates, you can easily pass rich data through your app and keep state out of the DOM. React makes it painless to create interactive UIs. Design simple views for each state in your application, and React will efficiently update and render just the right components when your data changes. Source: <https://reactjs.org/>.

- **Jest:** Is a JavaScript Testing Framework with the main philosophy related to offer an integrated “zero-configuration” experience, providing ready-to-use tools, to the developer end up writing more tests, which in turn results in more stable and healthy code bases. Fast interactive watch mode runs only test files related to changed files and is optimized to give signal quickly. A complete and ready to set-up JavaScript testing solution that works out of the box for any React project. At the end, Jest is a Unit Test framework for Javascript code. Source: <https://facebook.github.io/jest/>.
- **electron-builder:** A complete solution to package and build a ready for distribution Electron app for macOS, Windows and Linux with “auto update” support out of the box. It provides build version management, possibility to publishing artifacts to GitHub Releases, Amazon S3, DigitalOcean Spaces and Bintray. Support (compile for release-time on the fly on build) and Docker images to build Electron app for Linux or Windows on any platform. It is proficient to package and build an Electron application ready for distribution. Source: <https://github.com/electron-userland/electron-builder>.

CROPTEST API

In addition of the arguments presented earlier, another key point that contributed to the choice of the R software as the main language for API development is since the main DSSAT deployment already offers to the user an R installation. This eliminates the necessity to install additional software or language to use by Desktop API.

The information related to each package is presented as follows:

- **Readr** [23]: Read Rectangular Text Data. The goal is to provide a fast and friendly way to read rectangular data (like 'csv', 'tsv', and 'fwf'). It is designed to flexibly parse many types of data found in the wild, while still cleanly failing when data unexpectedly changes. Used for parsing summary and evaluate output files with support of flexible rules and meta-heuristics.
- **Dplyr** [24]: A Grammar of Data Manipulation. A fast, consistent tool for working with data frame like objects, both in memory and out of memory. This grammar provides a consistent set of verbs that help to solve the most common data manipulation challenges. It is designed to abstract over how the data is stored, meaning as well as working with local data frames or with remote database tables, using exactly the same R code. Used for pre-processing, cleaning and calculations of all statistics and comparison procedures between CSM models.
- **Jsonlite** [25]: A Robust, High Performance JSON Parser and Generator for R. A fast JSON parser and generator optimized for statistical data and the web. This package

offers flexible, robust, high performance tools for working with JSON in R and is particularly powerful for building pipelines and interacting with a web API. Used for export data from R API to the Desktop application via API functions.

- **Jug** [26]: A Simple Web Framework for R. It is a web framework aimed at easily building APIs. It is mostly aimed at exposing R functions, models and visualizations to third-parties by way of http requests. Used for exposing R API functions responsible by get input parameters from Desktop application interface and send results from calculation back.

3. CROPTEST DESKTOP

Abstract

A new version of CROPTEST that address the problem of compare different versions of DSSAT-CSM suite using automated procedures. In the first version, a restricted number of files are used for comparison, and this approach decreases the chance to detect a real change in results from all system based on changes in FORTRAN source code. The new version expand, the current number of output files used to proceed with comparison and are based on data-driven test (DDT) methodology; uses agile scrum development technics to improve the quality and the return value for DSSAT foundation every new release. A light version of this testing procedure could be setup to run directly on a online PaaS providing a broader response to the development team working on proposed model which would maintains the required quality expected for DSSAT-CSM code. In addition to a generic parser for DSSAT output files, a desktop application using Web modern technologies was designed and developed for a better user experience and productivity.

Keywords

crop modeling, data-driven test, automated tests, continuous integration

3.0.1 Introduction

A software development represents an important rule and carry out a big effort, especially to guarantee that all promises and functionalities will be delivered as expected. A software development could and, generally evolves, a few number of peoples called developer. But, besides the context where a specialist could write a computer program – like a scientist developing a novel tool to help others to use his new scientific or live improvement – one area of computer science called software engineering is responsible for the study, implementation, and dissemination of good standards for software development and delivery.

The most common tools, methodologies, and books are designed for help and improvement of commercial software. Scientists have been developing tools and software with methodologies for quality based on adaptations and adjust to those used in commercial development.

The scientific software presents special challenges for testing, most because cultural differences between scientist developers and software engineers, along with the characteristics of the scientific software make testing more difficult [27]. This context is now

changing, in part from the growing number of the scientific software that is coming to solve problems in the age of data and information [28].

That scenario is valid to crop modeling systems that is an intersection area between mathematics, biology, and agronomy. Most of the development is based on the training of scientists that have a basic to good knowledge of computer languages like FORTRAN and C.

FORTRAN is the *lingua franca* of modeling biological, life and engineering systems, must because the close relationship with the mathematical formulas but, from a testing software point of view, the language presents some limitations.

Specific tools have been developed for this approach, based specifically on FORTRAN like FRUIT [29], pFunit [30] and other adaptations, all them using unit test methodology.

A unit test is designed to test a small part of the code and verify if that portion produces the expected result based on assertions. Certainly, this is the most known and used method for testing software. Although, sometimes this approach does not fit in one project structure, for example, if the project already have many years of development and now it is necessary to implement some kind of software test.

Other methods and procedures present the developer with alternatives to the unit test pattern, for those we could cite test-driven development and DDT. The TDD⁹ proposes that first and foremost step must be to write the use tests case and, after that, develop a piece of software that satisfy or pass this test.

The DDT, on the other hand, uses a predefined set of input data that runs through the software and generate a known and valid output. The big change here is thinking the improvement of the software without the necessity of changing the code or write tests for each individual piece of the process. Unlike from unit test, that works from small parts and, hierarchically test larger parts, or believes that if the small parts work, the problem or failed output is due to joint or aggregations inability.

With DDT, the runs at the interaction parts of the software or system: the user input and the results output. This approach considers all parts of the entire system, and the measures of improvements are related to archive a better output – for the crop model systems, this is observed an output that fit well the experiment output. For crop model systems, we believe that this is the best options.

The CROPTTEST is a tool designed for quality software test of CSM, the main idea is to measure the relative differences from the summary and evaluates output files from DSSAT generate by two sets of source codes (stable and development).

The first version of CROPTTEST was created by DSSAT development team works with a predefined protocol to test a reduced and fixed number of model output files and ob-

⁹TDD: Test-Driven Development

serve the relative differences archived between two CSM versions. It is called CROPTEST and was developed using a programming language (Microsoft Visual Basic 6.0) without vendor support since 2016 [10].

This first version helps to solve an immediate problem related to quality software test, but the evolution and growing complexity of CSM demanded the development of a new version with improved functionalities and desired performance, like one related to the possibility to run on multiplatform and work with more files and work variables.

The main objective of this work is to create a new CROPTEST version to run in both desktop and online infrastructures nominated Platform as a Service (PaaS), using DDT methodologies and modern web development technologies, libraries and frameworks for best user experience and usability.

The model comparison and efficiency are carried out using evaluation measures and statistics common to model fitting context. Yang et al [19] review several statistical measures that could be used, for crop models evaluation. From this work we selected some indexes based on applicability and type of information used for classification of a better model.

Unfortunately, the use of software engineering methods for quality software tests applied specifically to CSM is not well described and to the best of author's knowledge and thorough search of the relevant literature yielded no related work describing this approach.

Following this introduction, Section 3.0.2 presents a brief background summary of the CROPTEST. Section 3.0.3 explain relationship between all those tools, methods and technologies. Section 3.0.4 results and discussion with include the new interfaces, the step procedures to training a new user for operation and some results from this operation. The final remarks and future works are presented in Section 3.0.5.

3.0.2 Background

The CROPTEST v1 was developed using Microsoft Visual Basic 6.0 by DSSAT development team. The main idea is to measure the difference between the summary and evaluates output files generate by two sets of CSM source codes (stable and development). All comparison result was stored is a spreadsheet file that shows the difference above a threshold using a semaphore system color (green-yellow-red) [10].

This version used the same user interface from DSSAT (Figure 1) and they were designed to run as an add-on or plugin for the main interface. It is necessary for both versions of codes installed as a binary file and, proceed with setting for maximum threshold accepted and source path for binary input and output from models. Both models run with the same input files.

Based on this description and further on the requirements for the new version, a plan with a review of technologies, tools, and methods for development of this new version was started. Ahead this period, a setup based on agile scrum development, using modern web technologies to design and develop the new desktop version using continuous integration tools and settings was the start point for present development.

3.0.3 Material and Methods

After Agile Manifesto [12] come to live and advocate about twelve principles that should coordinate the software development to deliver a piece of codes that is really dynamic in their deep essence. Other methods for software was been aggregated to the daily development routine.

One of them is called Agile Scrum [31] that was chooses for this project context, based on the necessity of development in strong interaction with the DSSAT Foundation (**Product Owner**) necessities for this tool and, to delivery intermediate versions with possibility to use and test, collect new directives and make those ones in functionalities for the CROPTEST. The development circle was defined in two weeks (**Sprint**) and the advisors (**Scrum Master**) suggestions are designed to deliver what was proposed.

All new suggestions and a bug report were filled on Github to make possible to evaluate the development coverage related to final project goals. The Github Project [16] was used to the management of the backlog and the issues during the sprint.

The interface requirements of the new CROPTEST are related to the necessity to run natively on the three major O.S. with the same interface, usability, and functionalities. Based on the resources restriction – only one developer, the native development for each O.S. was automatically rejected.

Among other options, the web development certainly could archive these requirements, but it must run from a web browser, and, the tool needed to run in own self-contained environment because of the heavy access to O.S. filesystem.

After analyzing how to solve this kind of problem, it was found a project started in 2013 internally at Github and now known as Electron [20]. It is a combination of Chromium¹⁰ - the engine of Google Chrome Web Browser - and Node.js [21], designed to build a cross-platform desktop application using modern Web technologies.

Today, companies like Microsoft (Skype¹¹), Github (Github Desktop¹²), Facebook (WhatsApp¹³) and Slack (Slack Client¹⁴) are already using Electron to build their own desk-

¹⁰Chromium - <http://www.chromium.org/Home>

¹¹<https://electronjs.org/apps/skype>

¹²<https://electronjs.org/apps/github-desktop>

¹³<https://electronjs.org/apps/whatsapp>

¹⁴<https://www.cyberscoop.com/electron-vulnerability-skype-slack/>

top application. Any Javascript library or framework could be used to compose the user interface application, the most common one is a Facebook framework called ReactJS [22] that uses reactive programming paradigm.

To develop the new version of CROPTEST Desktop, some Web technologies were used, among them Electron, Webpack, BabelJS, Node.JS, Semantic-UI, ReactJS, Jest and other React packages or frameworks with the recommended case of use besides the source address (Table 1).

The selection of these technologies was guided by the provision that related tools provide to software development with high-quality standard, that permits the setup for test the software during all phases of the development and, make it easy to use with CI/CD¹⁵ services. Also important is the long time support for software maintenance that could be archived via OSS¹⁶ developer community.

The developer community established around web technologies is one of the most engaged and supportive among others and, also based on former experience from the author and the rich set of tools, methods, and technologies shared and developed by Javascript community make this technologies a natural choice.

Table 1: Summary of the key Web technologies and their inter-relationship used to develop the new CROPTEST Desktop Application.

Component	Used for	Source ^a
Electron	Create Desktop Application	https://electronjs.org/
Webpack	Bundle and minify application for distribution	https://webpack.github.io/
BabelJS	Transpiler ^b	https://babeljs.io/
Node.js	Javascript engine	https://nodejs.org/en/
Semantic-UI	Javascript User Interface Framework	https://react.semantic-ui.com/
ReactJS	JavaScript library for building user interfaces	https://reactjs.org/
Jest	Unit Test	https://facebook.github.io/jest/
create-react-app	Application template	GH /facebook/create-react-app
react-redux	Application state container for JavaScript	https://redux.js.org/
react-router	Declarative routing for React	GH /ReactTraining/react-router
electron-builder	Package and build an Electron application ready for distribution	GH /electron-userland/electron-builder

^a **GH**: <https://github.com>

^b Conversion from *draft* ECMAScript version to default one that all modern Web Browser is able to run.

¹⁵CI: Continuous Integration / CD: Continuous Delivery

¹⁶OSS: Open Source Software

The project repository on Github was configured to check the source code automatically for each pull request or commit from a local machine. This setup uses the Continuous Integration configuration between a Github repository and Travis-CI [15].

Based on this configuration, each new tag added to the master branch of the repository start the build of the compiled version of CROPTEST to deliver using Github Releases for each platform (Windows, Linux, and MacOS).

The new work-flow and interaction between user steps, to run CSM models (based on batch files), parse the output files, calculate all statistics for both summary and evaluate output, and show the outputs for the user on new user interface is presented in Figure 8.

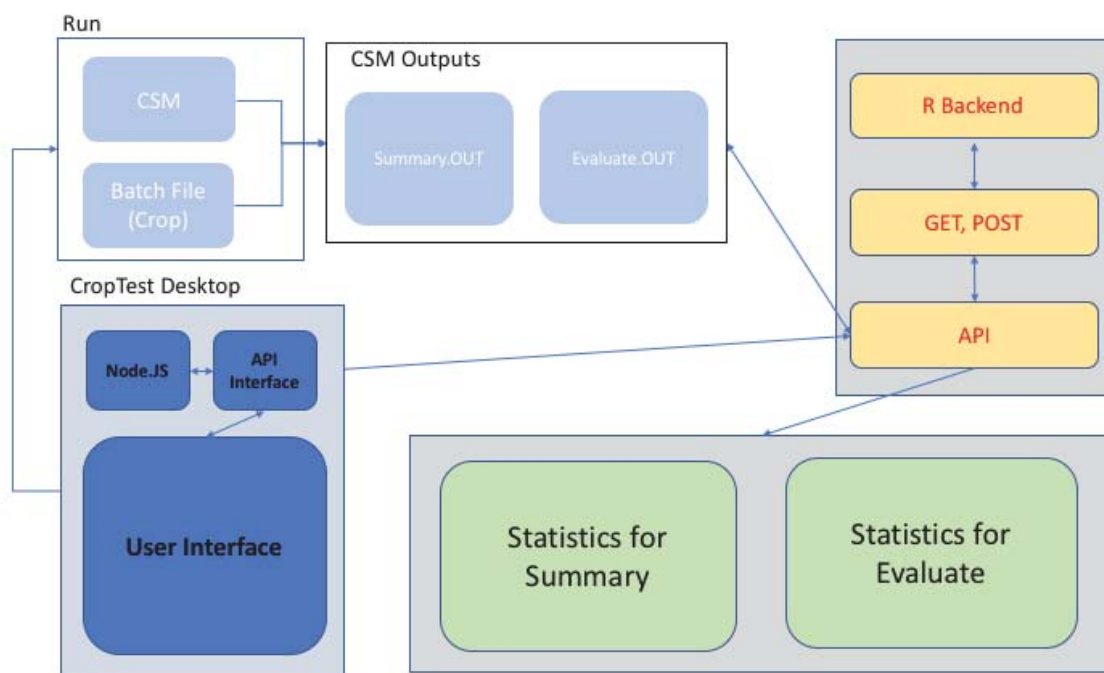


Figure 8: The CROPTEST work-flow showing the relationship between DSSAT-CSM, User Interface an API to run, parse, calculate and show the statistics for the user.

In short, Figure 8 clarify how this version is executed and which way it is responsible for coordinating the operation between the user interface and the CropTest API (see more in Chapter 4). The indicated sequence is related to the execution of CMS models, using batch files as configuration.

The previous operation generates as output, among others, the two files used for comparison (Summary and Evaluate) that are allocated within the correspondent folder model. After, an API call is made from the CropTest interface and the required data are shared as parameters – output files path and user defined threshold values.

The final step corresponds to the collecting processed data from API (summaries and statistics) for exhibition to the user, directly on the main interface. The entire process occurs dynamically and interactively, which offers greater agility to the model validation process when compared to the earlier CropTest version.

3.0.4 Results and Discussion

In this section, some aspects of CROPTTEST use through the steps of compare results between models. The results presented here are related to test; and data analysis realized before the release of version 4.7 compared with the last stable version 4.6 from CSM.

Run Step

At the Run step, the user must select both models for comparison procedure, remembering that all comparison will use Model A as the reference. Before running the models properly, it is necessary to build the binary version of CSM to use in comparison.

The CROPTTEST use a folder hierarchy to run models and store the output files for further analysis. The basic structure is composed of work folders for each model and a batch files folder. At the first run, CROPTTEST create two model folders to allocate the output files during the CSM run step.

Beyond this behavior, also it is allowed to user select the CSM binary from any filesystem folder. Table 2 describes the folders, contents use, default values and information about persistence and user settings enabled.

Table 2: CROPTTEST Desktop Application basic folder structure. Some settings could be adjusted and persisted between executions, others are fixed and defined on first run. The main folder is defined **CTWork** from last DSSAT version installed on the user computer.

Folder	Contents	Default Value	User Change?	Persistent?
ModelA	Binary CSM file from reference model.	DSSAT root folder	Yes, each run.	Yes
ModelB	Binary CSM file from proposed model.	CTWork\Model2	Yes, each run.	Yes
WorkA	Base directory for Model A output files.	CTWork\WorkA	No	Yes
WorkB	Base directory for Model B output files.	CTWork\WorkB	No	Yes
BatchFiles	Base directory for Batch Files used for crop model execution.	CTWork\BatchFiles	No	Yes

After selecting a crop model, the user could enable the use of “CTR”¹⁷ files during the model run, this permit overwrites the DSSAT and experiments files¹⁸ configuration related to output files. The next step is the choice of the Batch file to execute the models.

¹⁷A CTR file allows the user to set parameters that change the default behavior defined on each X file, it is like a global setting that overhead the outputs and run parameters for CSM crop model.

¹⁸X files - Set options for each crop model, p.ex. WHX for setting Wheat experiments.

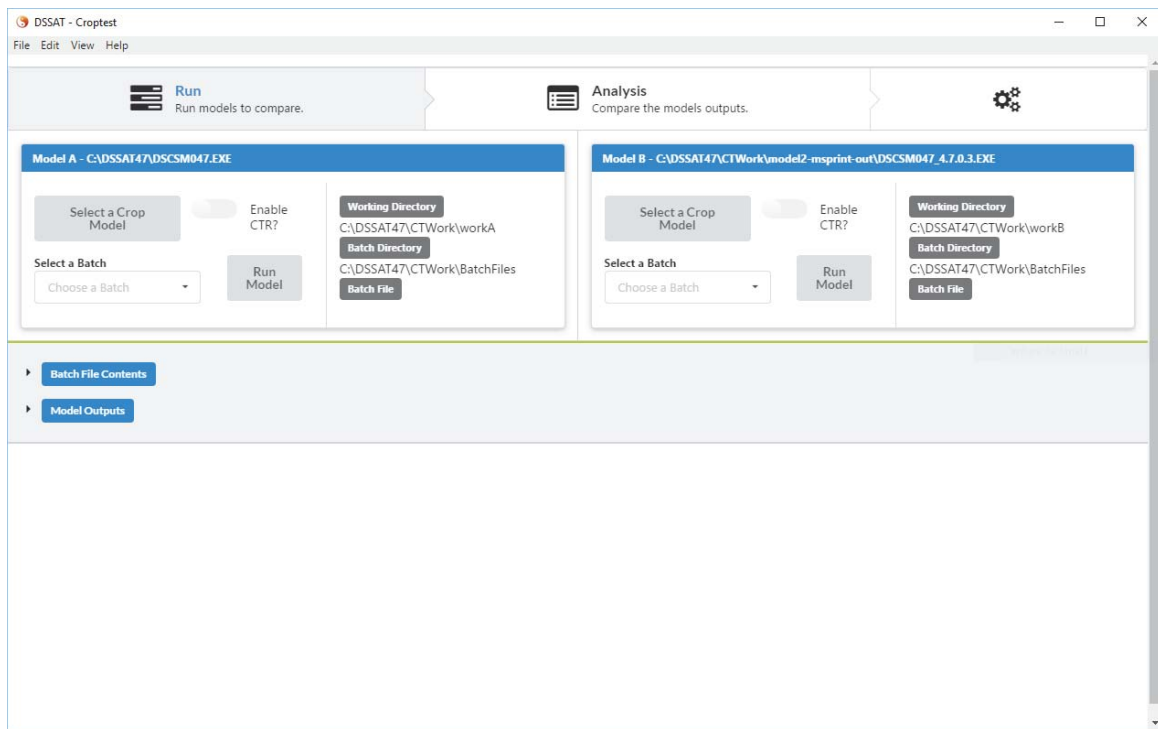


Figure 9: Snapshot of main window operations for the new CROPTEST. In detail, the *Run* step, that permits the user to run CSM models from User Interface and observe the results and running status.

All batch files located in the defined folder structure (Table 2) are parsed and shown to the user. For each selection, the file content is exhibited inside batch files content panel (Figure 10). As a default, each time the user changes the batch for Model A, it automatically will be set to Model B. Otherwise, changes in Model B only reflects located change.

The CROPTEST show, the CSM selected at the top of model's setting area, and the selected batch file name is also shown (Figure 9). As a final step, the user needs to click the "Run Model" button for each model – the CROPTEST could execute models in parallel.

When models finish their own run, the model output panel header change the color to alert the user that the run is done. For the cases where the model could not run or generate a warning output file, the button changes the color to reflect this behavior.

Figure 11 shows this behavior to alert the user of the current run state using semaphore scheme colors (Ok: **Model Output**, Warning: **Model Output**, Error: **Model Output**) alongside with the model output run.

Another functionality, designed to improve the interactivity with this process, is enabled when a warning or error files are detected during the model run to show the content of these files in a separated container (Figure 12).

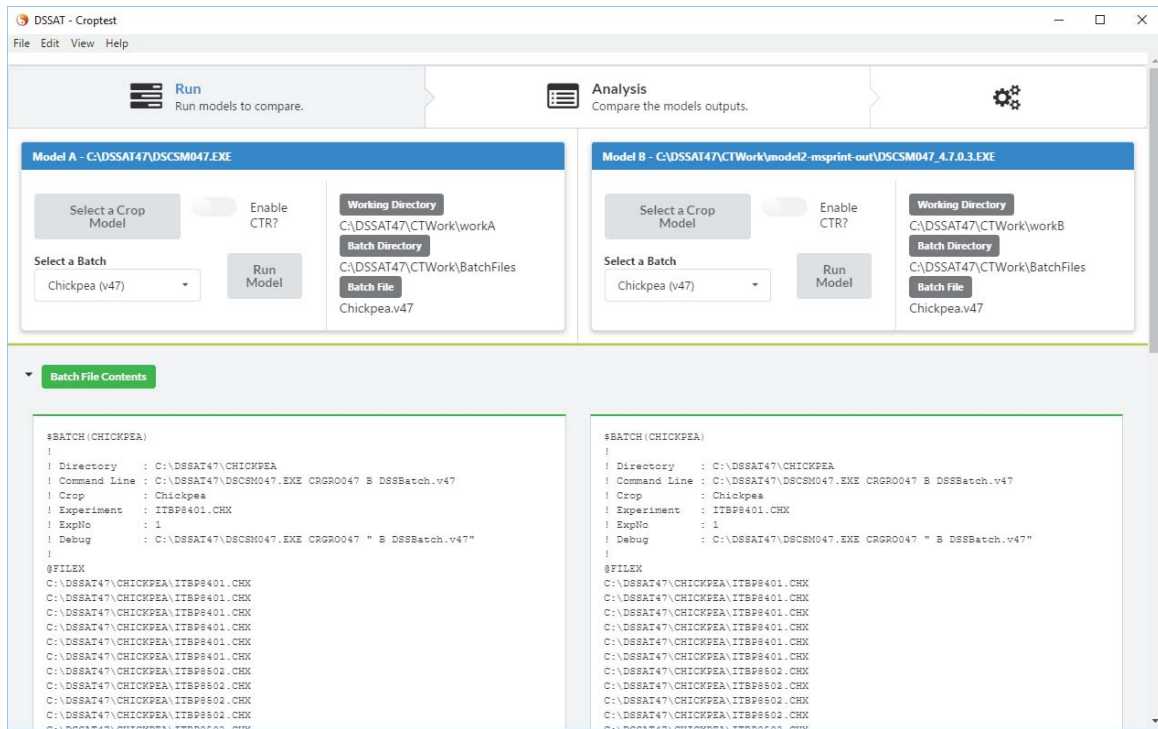


Figure 10: Snapshot of main window operations for the new CROPTEST. In detail, the *Run* step, showing the parse and load content of Batch file inside “Batch File Content” panel.

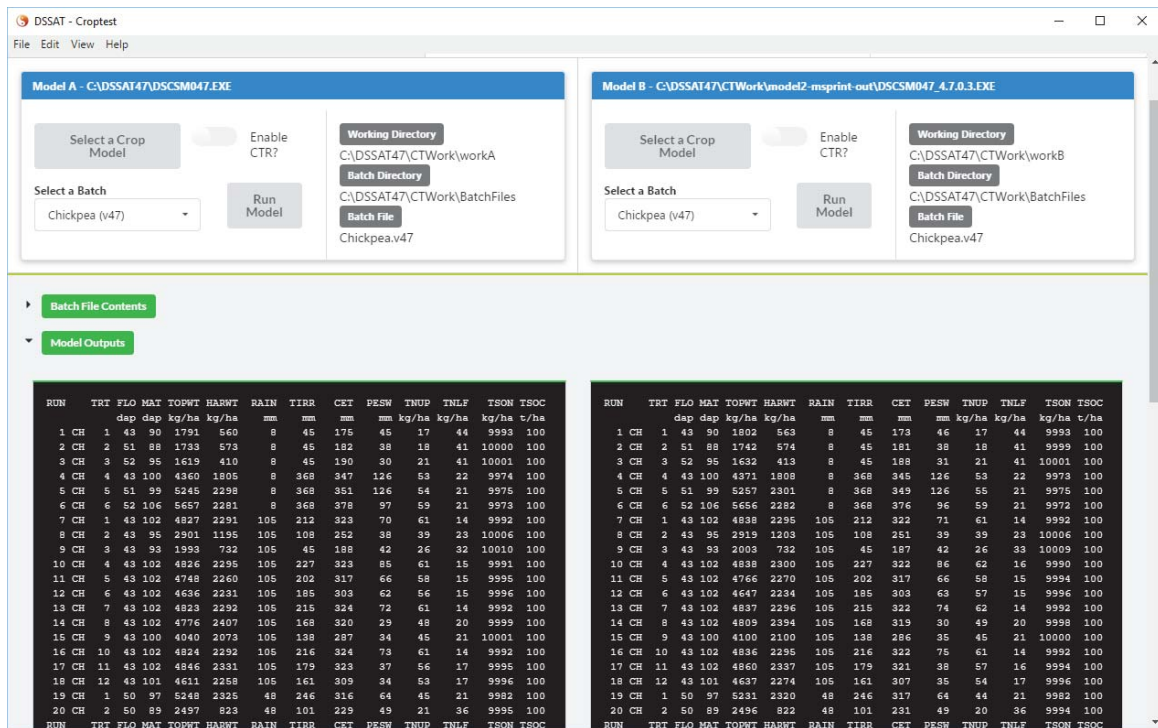


Figure 11: Snapshot of main window operations for the new CROPTEST. In detail, the *Run* step, showing the parse and load content of Batch file inside “Batch File Content” panel.

Analysis Step

Subsequently, the user must select the Analysis step to get all statistics and data summary from model outputs. At this point, the analysis and comparison of the models

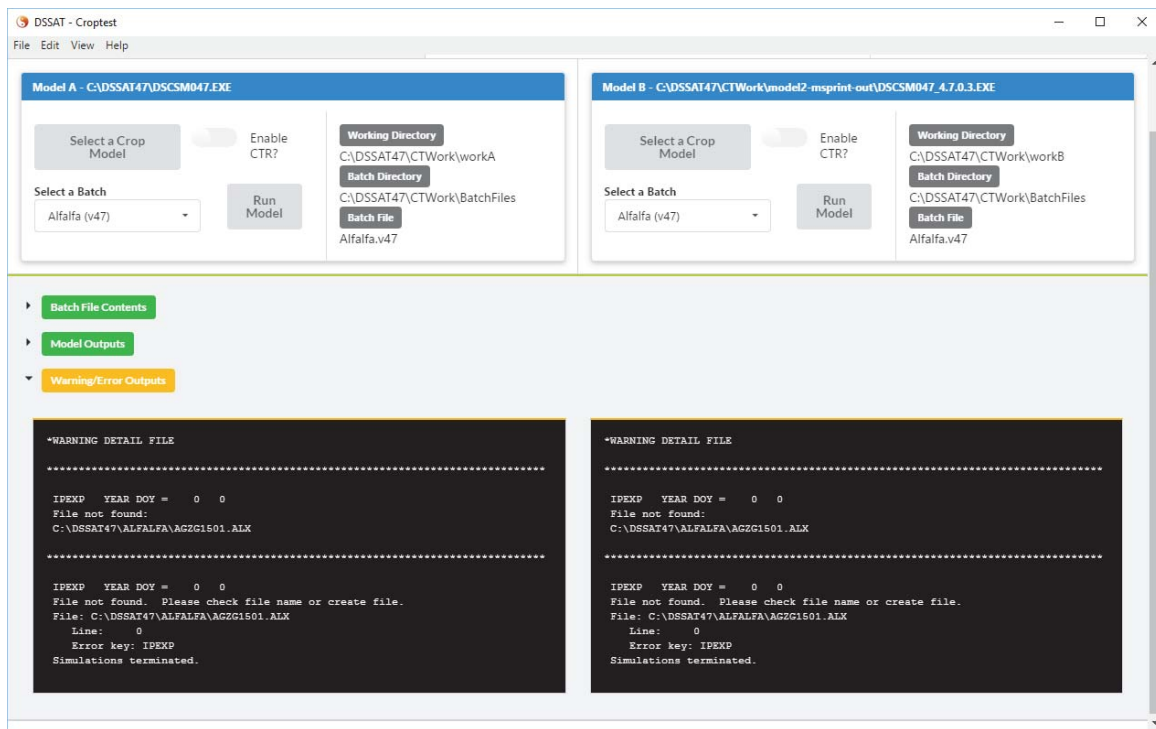


Figure 12: CROPTEST in detail, the *Run* step, showing the parse and load content of Warning file inside “Warning/Error Output” panel.

are made using sub-steps: statistics and calculations from the summary and evaluate the output.

During the parser of the files, the procedure checks if the files are valid DSSAT output files and if they are compatible and comparable between them. In the worst case, the user receives a message (Figure 13) saying that occur an error caused by a wrong file format or because the files are incompatible.

If the procedure detects a subset compatible, though the analysis and calculation proceed with this valid subset of both files. Specifically for evaluating output file, all comparison is carried out by model crop, because each one has a specific conjoint of output data (variables).

Figure 14 shows the statistical summary of comparison between models for evaluate output file. On this data table, the calculations from observed and simulated values are presented for each model output variable. Results are arranged side-by-side to make easier compare the models and find if the proposed new model is better than the last release version.

After showing some snapshots from CROPTEST user interface and explain some basic uses, it is important to describe the use of this tool during the evaluation of the new version of DSSAT tagged as 4.7 before it became a public release.

It is important to note that the results presented here is only a sneak-peak of all evaluation, and is based only on some crops that are available for use with DSSAT.

The screenshot displays the DSSAT - CROPTEST application window. At the top, there are two panels for configuring Model A and Model B. Both panels include a 'Select a Crop Model' dropdown (set to Chickpea), an 'Enable CTR?' toggle, and a 'Run Model' button. They also specify working and batch directories and batch files.

Below the configuration panels, there is a 'Summary.OUT' tab and an 'Evaluate.OUT' button. A 'Warning' message is displayed, indicating that certain treatments and columns are being dropped for comparison between the two models.

The 'Summary' tab is active, showing a table of statistics for various models. The table includes columns for SDAT, PDAT, EDAT, ADAT, IMDAT, HDAT, DWAP (%), CWAM (%), HWAM (%), HWAH (%), BWAH (%), PWAM (%), and HW. The statistics include Min. Value, Mean Value, Max. Value, Std. Deviation, Above Threshold (%), Outliers (%), and # Values.

Statistics	SDAT	PDAT	EDAT	ADAT	IMDAT	HDAT	DWAP (%)	CWAM (%)	HWAM (%)	HWAH (%)	BWAH (%)	PWAM (%)	HW
Min. Value	0	0	0	0	0	0	0.0	-1.5	-2.3	-2.3	0.0	-1.8	-4
Mean Value	0	0	0	0	0	0	0.0	-0.2	-0.1	-0.1	0.0	-0.1	0
Max. Value	0	0	0	0	0	0	0.0	0.5	0.7	0.7	0.0	0.7	0
Std. Deviation	0	0	0	0	0	0	0.0	0.4	0.5	0.5	0.0	0.4	0
Above Threshold (%)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.4	2.4	0.0	0.0	0
Outliers (%)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.4	7.1	7.1	0.0	9.5	2
# Values	42	42	42	42	42	42	42	42	42	42	42	42	4

Figure 13: Snapshot of main window operations for the new CROPTEST. In detail, the *Analysis* step, that permits the analyze the CSM models output for summary and evaluate files. Image details for summary output file.

The screenshot displays the DSSAT - Croptest software interface. The top menu includes File, Edit, View, and Help. The main interface is divided into three sections: Run, Analysis, and a settings gear icon. The Run section is currently inactive, showing 'Run models to compare.' The Analysis section is active, showing 'Compare the models outputs.'

Two model configuration panels are visible:

- Model A (Ver. 4.7.0.000 -release) - C:\DSSAT47\DSCSM047.EXE**: Includes a 'Select a Crop Model' button, an 'Enable CTR?' toggle, a 'Working Directory' field (C:\DSSAT47\CTWork\workA), a 'Batch Directory' field (C:\DSSAT47\CTWork\BatchFiles), a 'Batch File' field (Barley.v47), and a 'Run Model' button.
- Model B (Ver. 4.7.0.003 -develop) - C:\DSSAT47\CTWork\model2-msprint-out\DSCSM047_4.7.0.3.EXE**: Includes a 'Select a Crop Model' button, an 'Enable CTR?' toggle, a 'Working Directory' field (C:\DSSAT47\CTWork\workB), a 'Batch Directory' field (C:\DSSAT47\CTWork\BatchFiles), a 'Batch File' field (Barley.v47), and a 'Run Model' button.

Below the model panels, there are tabs for 'Summary.OUT' and 'Evaluate.OUT'. The 'Evaluate.OUT' tab is active, showing a 'General Statistics' table. The table has columns for Model A and Model B, and rows for various statistical metrics. A green 'Export to CSV' button is located above the table.

Statistics	ADAP		CNAM		CWAM		DRAP		EDAP		H#AM		Mc
	Model A	Model B	Model A	Model B	Model A	Model B	Model A	Model B	Model A	Model B	Model A	Model B	
Agreement Index	0.998	0.998			0.821	0.822					0.952	0.952	
D-Statistic	0.998	0.998			0.818	0.819					0.952	0.952	
Model Efficiency	0.990	0.990			-0.094	-0.084					0.851	0.852	
Mean Absolute Error	3.5	3.5			2127.4	2117.9					837.2	836.6	
Mean Difference	-3.5	-3.5			1950.2	1937.0					-103.2	-110.9	
Mean Obs. Values	95.6	95.6			6042.3	6042.3					6517.2	6517.2	
Mean Sim. Values	92.2	92.2	74.2	74.1	7992.5	7979.3	30.2	30.2	4.3	4.3	6414.1	6406.3	
Num. Obs. Values	13.0	13.0			13.0	13.0					13.0	13.0	

Figure 14: In detail, the *Analysis* step, that permits the analyze the CSM evaluate output files. Each column shows the statistics for both models side-by-side to make the process of decision easier.

Table 3: CROPTEST partial output (**min**, **avg** and **max**) of relative differences between CSM versions 4.6 and 4.7. A subset of all summary output variables are randomly selected for crops Cassava, Maize, Peanut, Rice, Soybean and Wheat.

Crop	Statistics	GNAM	HWAH	CWAM	DPNAM	HUM	EPCM	IRCM	DMPIM	PWAM	YPIM	ESCM	HWAM	NUCM
Cassava	Average		6.0%	-3.6%		-7.2%	-0.1%	-0.1%	-3.2%		6.3%	0.4%	6.0%	
	Maximum		11.1%	-2.9%		0.0%	0.8%	0.0%	-1.1%		9.3%	1.1%	11.1%	
	Minimum		3.5%	-4.3%		-12.5%	-0.4%	-2.0%	-4.3%		3.6%	-0.5%	3.5%	
Maize	Average	0.1%	0.2%	0.1%	0.1%	0.3%	0.1%	0.1%	0.1%	0.2%	0.1%	-0.0%	0.2%	0.2%
	Maximum	4.5%	3.8%	2.7%	2.7%	5.1%	1.1%	3.2%	0.3%	4.2%	1.3%	0.0%	3.8%	3.0%
	Minimum	-6.2%	-0.3%	-0.4%	-0.3%	-0.2%	0.0%	0.0%	0.0%	-0.4%	0.0%	-1.1%	-0.3%	-0.5%
Peanut	Average	0.1%	0.0%	0.0%	0.1%	-0.3%	0.0%	0.0%	0.0%	0.0%	0.0%	-0.0%	0.0%	1.3%
	Maximum	2.7%	0.9%	0.9%	0.5%	5.9%	0.6%	0.0%	0.9%	0.8%	0.9%	0.7%	0.9%	11.1%
	Minimum	0.0%	-0.1%	-0.2%	-0.1%	-6.2%	0.0%	0.0%	-0.2%	-0.1%	-0.1%	-0.8%	-0.1%	0.0%
Potato	Average	-22.9%	-21.6%	-11.2%	-11.2%	0.0%	12.3%	4.7%	-16.5%	0.0%	-27.4%	-11.7%	-21.6%	0.8%
	Maximum	1.4%	2.7%	8.2%	8.1%	0.0%	25.8%	12.3%	2.0%	0.0%	0.0%	-1.1%	2.7%	6.8%
	Minimum	-58.9%	-56.7%	-39.1%	-39.3%	0.0%	0.9%	-1.3%	-39.3%	0.0%	-65.6%	-21.9%	-56.7%	-17.5%
Rice	Average	0.1%	-4.0%	-3.6%	-2.7%	0.2%	-0.1%	0.6%	-4.4%	-3.9%	-4.8%	-0.3%	-4.0%	0.6%
	Maximum	1.8%	12.1%	10.6%	10.6%	0.6%	0.9%	3.0%	10.5%	11.0%	12.2%	3.1%	12.1%	1.9%
	Minimum	-1.0%	-14.2%	-18.6%	-16.8%	0.0%	-2.6%	0.0%	-22.0%	-15.3%	-16.8%	-4.2%	-14.2%	0.0%
Soybean	Average	0.1%	0.0%	0.1%	0.0%	0.9%	0.0%	0.0%	0.0%	0.0%	-0.1%	0.0%	0.0%	1.1%
	Maximum	1.3%	0.4%	0.8%	0.0%	4.8%	1.5%	0.0%	0.7%	0.4%	1.2%	0.2%	0.4%	3.2%
	Minimum	-0.9%	-0.7%	-0.8%	0.0%	-5.0%	-1.1%	0.0%	-1.8%	-1.1%	-7.1%	0.0%	-0.7%	0.0%
Wheat	Average	0.5%	0.6%	0.4%	0.4%	-0.1%	0.2%	0.0%	0.7%	0.6%	0.9%	-0.3%	0.6%	0.7%
	Maximum	1.5%	1.3%	1.1%	1.0%	0.0%	0.8%	0.0%	1.1%	1.2%	1.2%	0.0%	1.3%	1.8%
	Minimum	0.0%	0.0%	0.0%	0.0%	-0.4%	0.0%	0.0%	0.2%	0.0%	0.6%	-1.5%	0.0%	0.0%

From Table 3 is possible to see that some crops models show differences between versions 4.6 and 4.7 of CSM models. Among them, it is important to note the statistics for cassava, potato, and rice. As a reference to this process, from all 933 treatments tested, 264 shows differences between versions. Those treatments represent all data and experimental context available to the DSSAT user as default.

3.0.5 Conclusions

The new CROPTTEST interface have been designed to improve and make easier for the DSSAT development community members to validate and analyze results between CSM models. The step flow naturally carries out the user through the process with interactivity and minimal interference.

Compared with the previous version the whole interface, usability, and consistency to execute among diverse S.O. open the opportunity for developers to work with their favorite systems and deliver a better CSM for all designed systems and platforms.

The use of this new version expands the earlier version with more statistics and comparison possibilities, making the evaluation of a new CSM version more effective and flexible. But this new version steel have some missing functionalities like the use of sensitivity analysis.

The use of the CROPTTEST to analyze and compare the results before the 4.7 release was a big result. Although in development, the experience to test in real conditions and get feedback from the users was a great opportunity and provides a full list of lessons learned during this process, especially because the reinforce of the choice to use Agile Scrum methodology, while using Continuous Integration tools and systems.

A final word about the use of modern web technologies for desktop development proved higher satisfactory and with a minimal effort to learn a new library and development framework. All earlier knowledge from web development with Javascript was, and still is treasured for developer that would like to offer high quality alternatives to their customers.

As next steps, this new interface needs to accommodate the possibility to generate Batch files from the interface and the design and implementation of sensitivity analysis to test the new models with aggregate graphical functionalities.

4. CROPTEST API

Abstract

This API offers the same competency test and routines available for both Desktop and CI environment. The results observed for during the test setup was the same for both executions. A CI setup was revealed using GitHub and Travis-CI as choose PaaS services platform, but the same steps could be implemented in other platforms and vendors with a minimal adjusts. Using R software as the main language associated with the deliver the content as an API package proved to be a fast way to archive results based on statistics calculation and data pre-processing because of the large ecosystem sustained by an open-source community.

keyword

API, Continuous Integration, Data-Driven Test, Parser, R Package, Statistics

4.0.1 Introduction

An API is designed to encapsulate services and functionalities that need to be shared between tools and other software as a common resource. Formally, it presents a well defined and known access point that could receive inputs and parameters and return a processed or calculated result.

The CROPTEST API control the core operations related to parsing, pre-processing, statistical calculations, quantification of differences between models and pre-formatting data for visualization on a desktop application.

In general, compare models is about summarizes the difference between observed and simulated values. For this intent, one can choose statistical procedures to measure the goodness-of-fit or, on the other hand, observe the lack-of-fit for those two set of values.

Model evaluation and comparison for DSSAT [6, 32] crop model outputs are performed by parsing and analyzing the content of summary and evaluate files, these contain the basic and most important information for crop model simulation.

The summary output file shows information about variables related to dry weight, yield and yield components, water, nitrogen, phosphorus and organic matter, just to elucidate some of them. These components show the whole picture from which treatment as result of model output for crop simulation.

The evaluate output file shows information about the simulated and observed values for the same treatments as a summary file. Here, one must know how good is the crop model performance in relation to the fields observation.

But, both files only exhibits the numeric information, and any type of statistical or mathematical performance evaluation needs to be done in an external software like spreadsheets or statistical software.

This is the most valuable contribution of the proposed API, it is responsible for automatically performs all necessary calculations and statistical summarization for comparing individual input models – like the difference between the simulated and observed value for each variable for the evaluate output file – and for calculations for comparison between input models.

All API procedures and calculations are encapsulated into an R package, to make easier to distribute and use with both the desktop application and a continuous integration setup, and this approach makes the maintenance consistent and affordable.

The parsers automatically extract the information from inputs based on meta-heuristic procedure that use a group of positional and table rules to define the content to be extracted during the parse.

To develop this API was chosen an analogous approach used in other scientific contexts, along with the procedures already in use for the DSSAT statistical routines. Unfortunately, with the author's best knowledge and exhaustive research of relevant literature, there has been no related work describing this approach.

In Section 4.0.2 we describe shortly the technical background that makes the API runs. Section 4.0.3 show the materials and computational methods used to develop the API. Section 4.0.4 describe the results gathered from an API run from Travis-CI. Section 4.0.5 discuss the conclusions and presents the next steps and further works.

4.0.2 Background

API language development can affect the service performance and the long-term maintenance, based on these points the developer is responsible by the choice of one that could deliver the best experience for the client that needs this service.

For jobs related to parsing and processing data, computer languages that are designed for this intent offer an advantage when compared with general purposes ones like NodeJS¹⁹, C++²⁰, and Python²¹.

¹⁹<https://nodejs.org>

²⁰<https://isocpp.org>

²¹<https://www.python.org>

The R software [33] is a computer language and environment created and designed for data analysis, also it posses a large open source community²² and a repository²³ to share functionalities written and delivered in packages format.

Because the primary attributions of this API are related to parsing, preprocessing, and calculating comparison statistics based on input data, using community maintained packages could certainly reduce development time. Another method could be to develop a generic parser using the minimum rule setting.

For this method, the process of parsing a file must first define a minimum number of rules that describe in the list the types and amount of information that will be required to read and store for the subsequent procedure. For general purpose analysis, you need to write this minimum content setting, and the most common implementation uses a grammar specification [34].

Apart from the grammar specification is a powerful tool, sometimes it is only necessary the implementation of a smaller subset or a tokenized parser version based on table rules or meta-heuristics. This kind of simplification intended to make maintenance easier and still achieve good results.

Fortunately, it is possible to use a grammar specification from a R package, this is the intent of the “readr” package [23] a powerful tool to read text file types, like ones related to summary and evaluate outputs files from DSSAT. Other R packages was used to provide all functionalities necessary to this API, and all them will be described at Section 4.0.3.

It is important to note that all procedures implemented in API were designed to run from an R environment and supply the results to the CropTest Desktop Application and to be used as a command line tool from a Continuous Integration (CI) platform.

Currently the CI is the most common step in the software release life-cycle. In fact, this tools improve the management of the process substantially and accelerate the time between development and test process [35, 36, 37].

Jamil et al [38] review the practices for testing software and describe this step with fundamental importance, especially because of the growing complexity of software. Kanewala et al [27] explore the same for scientific software context and, again, cite this as an important role in overall software quality.

4.0.3 Material and Methods

The R package API for CROPTTEST was developed using the official manual to write R extensions [39], and the materials of best practices to develop a new R package

²²GitHub: <https://github.com/search?l=R&q=R&type=Repositories>

²³CRAN: <https://cran.r-project.org>

[40, 41]. Eddelbuettel et al [42, 43, 44] describes all steps necessary to integrate and extend R software using C++ programming language.

The parser functionalities are generic to accommodate and check a broad range of situations of resulting summary and evaluate output files, including functions to smooth response to errors and were developed using functionalities from package “readr” [23] to read data from output files and “dplyr” [24] to pre-processing and calculations for the file’s contents.

For the comparison procedure, the algorithm compares the first model (called Model A), as default it is the last stable version installed in user machine or the master branch for online execution, with the proposed one. The calculations for summary output utilizes the “Model A” as the reference to calculate the relative difference between numeric variables, and for date ones, the procedure calculates the absolute difference.

Other statistics used for comparison in summary file context, they are minimum, average and maximum observed value; standard deviation, the number and percent of treatments higher than the threshold, the number and percent of treatments tagged as an outlier (using box-and-whisker procedure [18]) and the number of treatments used for calculation.

In the evaluate file context, they are: agreement index, D-statistic, model efficiency, mean absolute error (MAE), the mean for difference, observed and simulated values, the number values for observed and simulated, the root mean squared error (RMSE), the normalized RMSE, the R squared (R^2) and the standard deviation for observed and simulated values [19].

After all calculations, the data results are exported back to the CropTest Desktop Application as JSON²⁴ [25] or as raw text to be used in CI process as a back-end API service [26]. The benefit to providing this API as a package is related to the possibility to use the same code in both environments, but before the use with CI, some additional configuration need to be done at the planned platform, for this work the choice was made by GitHub.

The detailed workflow process is illustrated in Figure 15, these steps occur between online services (PaaS) and is responsible for producing a test routine alongside with the build and binary release.

The automation setup is based on a configuration file with all necessary steps needed to complete the test using API functions. The main communication between services are done using “listeners” and “webhooks”²⁵.

When the developer makes a change in the code on the local machine and submits to the GitHub, a webhook is triggered to start the CI process on the service server. At this

²⁴JSON (JavaScript Object Notation): <https://www.json.org>

²⁵Webhooks provide a way to deliver notifications to an external web server when certain actions occur. For more information, see <https://developer.github.com/webhooks/>

service, a new virtual machine starts the process of building and test this new proposed source code.

After the build step finished, the test procedure starts and validates this proposed version comparing the results from this run with one from a stable version. If all procedure is successful, the release content is sent back to the GitHub as a new binary release.

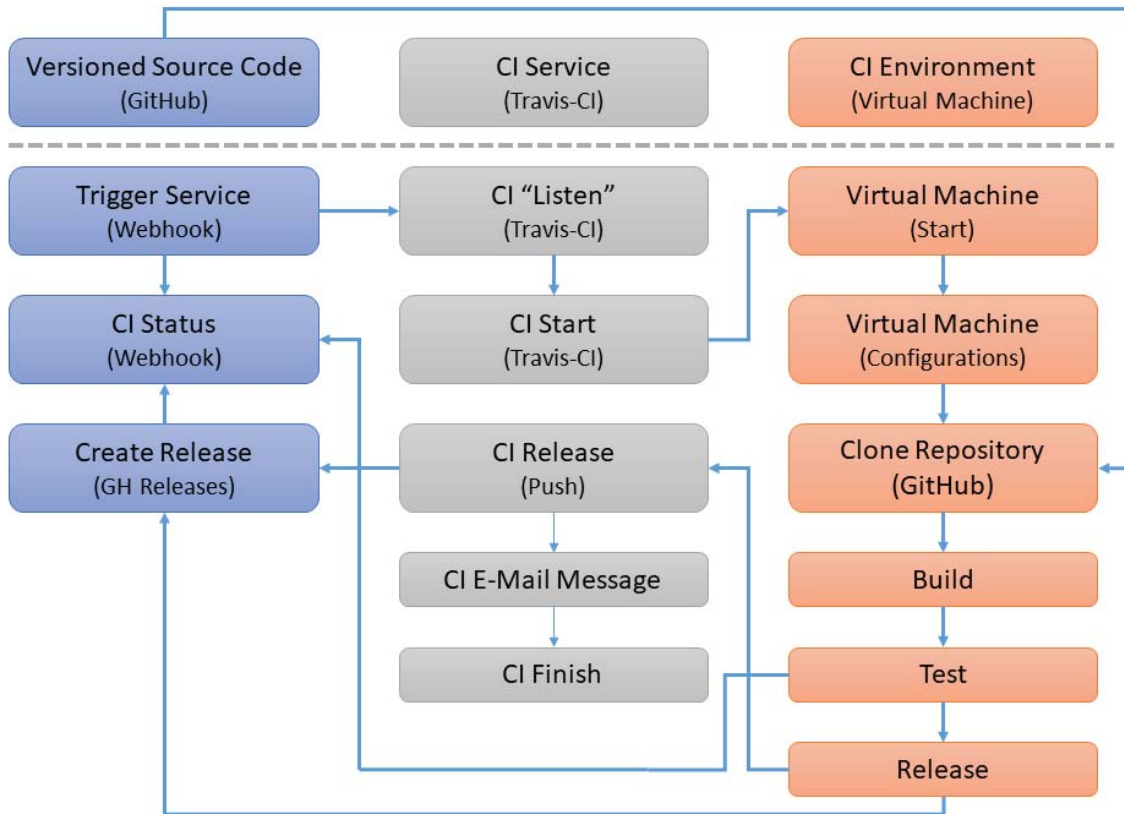


Figure 15: Workflow and related key steps necessary to test a new CSM version hosted on GitHub with environment allocated on Travis-CI.

To use the API in CI environment it is necessary to define a configuration file which elucidates the dependencies, procedures and tests that the overall process needs to conclude. As a sample for this configuration content, a pseudo-algorithm (4.1) details the minimal requirements and procedures necessary to build and test a new version of the DSSAT-CSM.

4.0.4 Results and Discussions

The API workflow is illustrated in Figure 16 where, in the main procedure, it received as parameters the path from input files (Summary and Evaluate) and the thresholds for numeric and date comparison (Summary) and the crop (Evaluate).

These parameters are received by the API in two ways: using the command line (R script) from a CI setup or through HTTP call. After, the parser procedures execute their


```

1: Enable configuration at GitHub
2: Enable access from CI platform to the repository
3: Add tasks to the CI configuration file at repository
4: while Build using Linux machine on CI platform do
5:   Install Cmake for compiling DSSAT
6:   Install R and dependency packages for API
7:   Clone DSSAT data structure
8:   Clone DSSAT source structure
9:   Clone API source structure
10:  Compile DSSAT using CMake
11:  if Compile DSSAT successfully then
12:    Copy final DSCSM0XX.EXE file to data structure
13:    Build and install API
14:    Run check script with input Batch Files
15:    if API Check is successfully then
16:      Report comparison table output
17:    else
18:      Report comparison error
19:    end if
20:  else
21:    Report compilation error
22:  end if
23: end while

```

Algorithm 4.1: Pseudocode with steps necessary to configure an automation for test new versions of CSM using Travis-CI or Circle-CI.

code using helper functions from “readr” package, followed by the pre-processing routines (“dplyr” package).

The core step occurs during the statistics calculations for both summary and evaluate inputs. In the end, these outputs are sent back as a JSON file using HTTP response (“jsonlite” package) or as a text file using “readr” package.

To check the API operation and results, a test was conducted comparing the results obtained from a local execution (CropTest Desktop) and from a PaaS execution using GitHub and Travis-CI configuration setup.

For the pair GitHub/Travis-CI, the procedure was enabled in a fork from the development branch from DSSAT GitHub repository which was used for setup and configurations test. Subsequent to all settings, some minimal changes in the code was committed to test and evaluate this new setup.

The first step was checking if the DSSAT-CSM binary was built correctly. After, the next test was related to the installation of the API, and the final was the batch file execution using the DSSAT data structure. For this setup, the check will be successful if the sum of variables (columns) with the relative or absolute difference between the models is equal to zero (no differences higher than the base threshold).

As default setting for CI platforms, if one step fails, all check procedure stops, and the last error are reported from CI platform log file, and an e-mail with a link to the log file is sent to the user registered as the owner of the repository.

During the execution of this configuration setup, the overall process from the commit through the finish and test report records the same results from one with changes made only locally and obtained using the CropTest Desktop version.

It is important to note that this result validate and confirm that the configuration setup, using only the API, could figure an alternative way to offer all DDT test routines to a larger number of developers inside DSSAT community, based on the possibility to run tests online at PaaS service like Travis-CI and archive the same results as run locally, using desktop version of the CROPTEST.

Overall, this represents a better use of time by the development team to manages the pull request and proposed changes because, before this configuration, one of the DSSAT developer members must download the proposed change in your own machine, build, and checks the errors and improvements by itself using Desktop version.

This procedure could be enabled to run from once a day to more intensive test, where each commit to the develop branch will start the whole process. All these must be set at CI panel settings for the enabled repository, in addition to batch files that will be run and the config file with the threshold default settings.

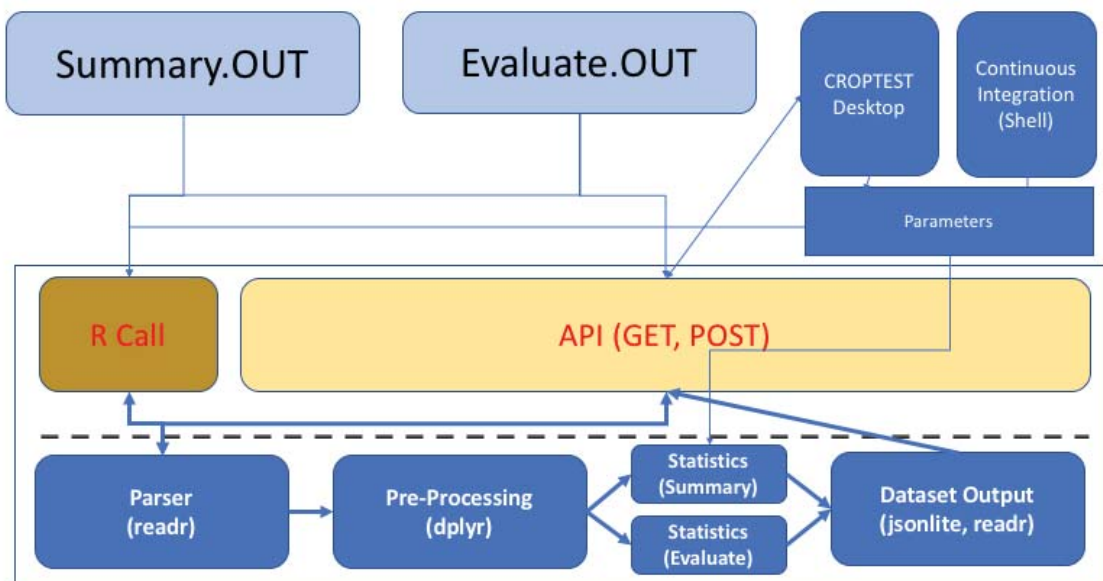


Figure 16: The CROPTEST API work-flow showing the relationship between CSM output files, parser procedure, statistics calculations and results output to the user and for shell interaction (CI Platform).

Upcoming, this setup enables the DSSAT developer team to deliver software [45] with less effort related to the testing process and with a possibility to have improvements historical data over time and, also, other types of analysis [46, 47].

Beller et al [35] describe that (CI) has become a best practice of modern software development and show the importance of software test as the reason why builds fail (10% more failures being caught at build time).

4.0.5 Conclusions

Using the same code in both Desktop and CI validation improve the efficiency and makes the test procedure transparent to the contributor and to maintain. The potential restriction for use in CI online validation is related to the time to get the result but as compensation, the maintainer receives an e-mail with the output of test without extra effort.

All statistics and intermediate calculations are developed using R software as the base programming language, which makes it easy to extend and add new calculations and statistics using any of the community package contributions without the obligation to write the all the code from scratch.

The results obtained during the tests are the same for Desktop and CI version, which validate the accuracy from the API procedures results independent of environment setup.

Moreover the setup implemented was too simplistic, it is possible to make tests improvements related with the expanding the covering settings which intent to receive more information between available models' differences.

As final remarks, this work shows how to implement a setup for using the CropTest API for testing in both scenarios, local and remote. Also confirms that the results were the same and, lastly, this type of setup could be used in other scientific projects to help and improve the software quality.

5. CONCLUSIONS AND FUTURE WORKS

As a final remarks from this work, the implementation of a test procedure that results in no change in the CSM source code and still could delivery with quality feedback to developers with intents to generate a final software with fewer bugs and improving the already known high quality of DSSAT suite.

Both Desktop and API design and implementation proved to be a good challenge in the sense of looking for alternatives always with the goal to bring the methods and techniques that are well established in Computer Science professionals and community, but still far from ideal on Scientific Developer community. In the end, this present work is one more step to fill this gap between areas that certainly could generate better user experience with higher software quality for use inside scientific community.

For general conclusions, comparing with the previews version the whole interface, usability, and consistency to execute among diverse Operating Systems (OS) are improved. This opens the opportunity for developers to work with their favorite systems and deliver a better CSM for all designed systems and platforms.

With this new interface, the steps flow naturally and carry out the user through the process with interactivity, minimal interference, and improved performance. This achievement is related to the use of modern web technologies for desktop development provides higher satisfactory software. The further maintenance could be easier archived, in part, because of the growing number of Web developers with knowledge of Javascript and related technologies.

The API supports the development both in Desktop and in Continuous Integration (CI) platforms. The parser procedure use table rules and meta-heuristics to extract the content from the summary and evaluate output files in a flexible and robust way. All statistics and calculations are developed using R software, which makes it easy to extend and add new calculations and statistics using well-known language with community support without the obligation to write the all the code from scratch. The R API Package used in both Desktop and CI validation improve the efficiency and makes the test procedure transparent to the contributor and to maintain.

As future works for each component that was being developed during this effort, the author could elect these follows:

- **Desktop:** As next steps, this new interface needs to accommodate the possibility to design and implementation of sensitivity analysis to test the new models with aggregate graphical functionalities.

- **API:** As next steps, will be necessary to define a better validation procedure for CI tests with the possibility to have a configuration files defining the procedure. Add a procedure to generate and update the references files for comparison (CSM stable version). Add calculations and procedures to run sensitivity analysis.
- **Continuous Integration:** The frequency of code validation on CI need to be set to accommodate the actual reality for DSSAT developer community, and this could be revised at specific times in the future; Discuss with DSSAT Core Team the better approach to deliver this automation procedure from main DSSAT Repository; and write and deliver a better documentation of the steps and type of settings.

REFERENCES

- [1] Tsai, B.-Y.; Stobart, S.; Parrington, N.; Thompson, J. B. "Iterative design and testing within the software development life cycle", *Software Quality Journal*, vol. 6, 1997, pp. 295–310.
- [2] Jan, S. R.; Shah, S. T. U.; Johar, Z. U.; Shah, Y.; Khan, F. "An innovative approach to investigate various software testing techniques and strategies", *International Journal of Scientific Research in Science, Engineering and Technology (IJSRSET)*, Print ISSN, 2016, pp. 2395–1990.
- [3] Laukkanen, P. "Data-Driven and Keyword-Driven Test Automation Frameworks", Ph.D. Thesis, 2006.
- [4] Sommerville, I. "Software Engineering". USA: Addison-Wesley Publishing Company, 2010, 9th ed..
- [5] Tsuji, G. Y. "Network management and information dissemination for agrotechnology transfer". In: *Understanding Options for Agricultural Production*, Tsuji, G. Y.; ; Hoogenboom, G.; ; Thornton, P. K. (Editors), Springer, Dordrecht, 1998, chap. 18, pp. 367–381.
- [6] Jones, J.; Hoogenboom, G.; Porter, C.; Boote, K.; Batchelor, W.; Hunt, L.; Wilkens, P.; Singh, U.; Gijsman, A.; Ritchie, J. "The DSSAT cropping system model", *European Journal of Agronomy*, vol. 18–3-4, jan 2003, pp. 235–265.
- [7] McCown, R.; Hammer, G.; Hargreaves, J.; Holzworth, D.; Freebairn, D. "Apsim: a novel software system for model development, model testing and simulation in agricultural systems research", *Agricultural systems*, vol. 50–3, 1996, pp. 255–271.
- [8] Feldt, R. "Do system test cases grow old?" In: *Software Testing, Verification and Validation (ICST)*, 2014 IEEE Seventh International Conference on, 2014, pp. 343–352.
- [9] Lindvall, M.; Ganesan, D.; Ardal, R.; Wiegand, R. E. "Metamorphic Model-Based Testing Applied on NASA DAT – An Experience Report". In: *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, 2015, pp. 129–138.
- [10] Porter, C. H.; Jones, J. W.; Hoogenboom, G. "DSSAT model comparison utility (CropTest)". In: *DSSAT v4. A Decision Support System for Agrotechnology Transfer*, Jones, J. W.; Hoogenboom, G.; Wilkens, P. W.; Porter, C. H.; Tsuji, G. Y. (Editors), University of Hawaii: ICASA Tools. International Consortium for Agricultural Systems Applications, 2004, vol. 4, chap. 3, pp. 1–29.

- [11] Wilson, G.; Aruliah, D. A.; Brown, C. T.; Chue Hong, N. P.; Davis, M.; Guy, R. T.; Haddock, S. H. D.; Huff, K. D.; Mitchell, I. M.; Plumbley, M. D.; Waugh, B.; White, E. P.; Wilson, P. “Best practices for scientific computing.”, *PLoS biology*, vol. 12–1, jan 2014, pp. e1001745, arXiv:1210.0530v4.
- [12] Beck, K.; Beedle, M.; Van Bennekum, A.; Cockburn, A.; Cunningham, W.; Fowler, M.; Grenning, J.; Highsmith, J.; Hunt, A.; Jeffries, R.; Others. “Manifesto for agile software development”, 2001.
- [13] Rising, L.; Janoff, N. “The Scrum software development process for small teams”, *IEEE Software*, vol. 17–4, 2000, pp. 26–32.
- [14] Fitzgerald, B.; Stol, K.-J. “Continuous software engineering: A roadmap and agenda”, *Journal of Systems and Software*, vol. 123, 2017, pp. 176 – 189.
- [15] Travis CI. “Travis CI”. Source: <https://travis-ci.org/>, 2018.
- [16] Github Core Team. “Introducing Projects for Organizations”. Source: <https://github.com/blog/2272-introducing-projects-for-organizations>, 2017.
- [17] Deshpande, A.; Riehle, D. “Continuous integration in open source software development”. In: IFIP International Conference on Open Source Systems, 2008, pp. 273–280.
- [18] Krzywinski, M.; Altman, N. “Visualizing samples with box plots.”, *Nature methods*, vol. 11–2, feb 2014, pp. 119–20.
- [19] Yang, J.; Yang, J.; Liu, S.; Hoogenboom, G. “An evaluation of the statistical methods for testing the performance of crop models with observed data”, *Agricultural Systems*, vol. 127, may 2014, pp. 81–89.
- [20] Electron Core Team. “Electron”. Source: <https://electronjs.org/>, 2018.
- [21] Node.js Team. “Node.js”. Source: <https://nodejs.org/en/>, 2017.
- [22] Facebook Inc. “A JavaScript library for building user interfaces | React”. Source: <https://facebook.github.io/react/>, 2014.
- [23] Wickham, H.; Hester, J.; Francois, R. “Read Rectangular Text Data Description”. Source: <https://cran.r-project.org/package=readr>, 2017.
- [24] Wickham, H.; Francois, R. “A Grammar of Data Manipulation”. Source: <https://cran.r-project.org/package=dplyr>, 2015.
- [25] Ooms, J. “The jsonlite Package: A Practical and Consistent Mapping Between JSON Data and R Objects”, *arXiv:1403.2805 [stat.CO]*, mar 2014, 1403.2805.

- [26] Smeets, B. "A Simple Web Framework for R". Source: <https://cran.r-project.org/package=jug>, 2017.
- [27] Kanewala, U.; Bieman, J. M. "Testing Scientific Software: A Systematic Literature Review.", *Information and software technology*, vol. 56–10, oct 2014, pp. 1219–1232.
- [28] Kelly, D.; Sanders, R. "The Challenge of Testing Scientific Software". In: *CAST 2008: Beyond the Boundaries*, 2008.
- [29] Chen, A. H.; David, P. "FORTRAN Unit Test Framework (FRUIT)". Source: <https://sourceforge.net/projects/fortranxunit/>, 2008.
- [30] Oloso, A.; Cruz, C.; Rilee, M. L.; David, A.; Clune, T. "pFUnit". Source: <https://sourceforge.net/projects/pfunit/>, 2005.
- [31] Abrahamsson, P.; Salo, O.; Ronkainen, J.; Warsta, J. "Agile Software Development Methods: Review and Analysis", *Computer Science Education*, vol. 12–3, sep 2017, pp. 167–168, 1709.08439.
- [32] Hoogenboom, G.; Porter, C.; Shelia, V.; Boote, K.; Singh, U.; White, J.; Hunt, L.; Ogoshi, R.; Lizaso, J.; Koo, J.; Asseng, S.; Singels, A.; Moreno, L.; Jones, J. "Decision Support System for Agrotechnology Transfer (DSSAT)". Source: <https://dssat.net>, 2017.
- [33] R Core Team. "R: A Language and Environment for Statistical Computing". Source: <https://www.r-project.org/>, 2017.
- [34] de Guzman, J.; Kaiser, H. "Boost-spirit, c++ libraries for parsing and output generation". Source: <http://boost-spirit.com>, 2015.
- [35] Beller, M.; Gousios, G.; Zaidman, A. "TravisTorrent: Synthesizing Travis CI and GitHub for Full-Stack Research on Continuous Integration". In: *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, 2017, pp. 447–450.
- [36] Tim Heckel. "Meet Travis CI: Open Source Continuous Integration". Source: <https://www.infoq.com/news/2013/02/travis-ci>, 2013.
- [37] Beller, M.; Gousios, G.; Zaidman, A. "Oops, My Tests Broke the Build: An Explorative Analysis of Travis CI with GitHub". In: *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, 2017, pp. 356–367.
- [38] Jamil, M. A.; Arif, M.; Abubakar, N. S. A.; Ahmad, A. "Software Testing Techniques: A Literature Review". In: *2016 6th International Conference on Information and Communication Technology for The Muslim World (ICT4M)*, 2016, pp. 177–182.
- [39] R Core Team. "Writing R extensions", 2017.

- [40] Sugawara, E.; Nikaido, H. “Properties of AdeABC and AdeIJK efflux systems of *Acinetobacter baumannii* compared with those of the AcrAB-TolC system of *Escherichia coli*.”, *Antimicrobial agents and chemotherapy*, vol. 58–12, dec 2014, pp. 7250–7, 1011.1669.
- [41] Wickham, H. “Advanced R”. Source: <http://adv-r.had.co.nz/>, 2016.
- [42] Eddebuettel, D.; François, R. “Rcpp : Seamless R and C++ Integration”, *Journal of Statistical Software*, vol. 40–8, 2011, pp. 1–18.
- [43] Eddebuettel, D. “Seamless R and C++ Integration with Rcpp”. New York, NY: Springer New York, 2013.
- [44] Eddebuettel, D.; Balamuta, J. J. “Extending R with C++: A Brief Introduction to Rcpp”, *PeerJ Preprints*, vol. 5, 2017, pp. 1—27.
- [45] Catelani, M.; Ciani, L.; Scarano, V. L.; Bacioccola, A. “Software automated testing: A solution to maximize the test plan coverage and to increase software reliability and quality in use”, *Computer Standards & Interfaces*, vol. 33–2, feb 2011, pp. 152–158.
- [46] Santos, E. A.; Hindle, A. “Judging a commit by its cover”. In: Proceedings of the 13th International Workshop on Mining Software Repositories - MSR '16, 2016, pp. 504–507.
- [47] Souza, R.; Silva, B. “Sentiment Analysis of Travis CI Builds”. In: 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR), 2017, pp. 459–462.