iEnvironment: Perspectives on Metadata-Oriented Testing of Research Software

Doug Mulholland, Paulo Alencar, Donald Cowan
David R. Cheriton School of Computer Science
University of Waterloo, Waterloo, Ontario, Canada N2L3G1
{dwm, palencar, dcowan}@csg.uwaterloo.ca

Abstract

As a research software platform, iEnvironment has been proposed to support open and big data sharing and reuse for researchers working on surface water issues. Research software refers to software development tools that accelerate discovery and simplify access to digital infrastructures. Although research software platforms are becoming increasingly more innovative and powerful, this increasing complexity hides a greater risk of failure as unplanned and untested program scenarios arise. As systems age and are maintained by different programmers the risk of a change impacting the overall system increases. In contrast, systems that are built with less emphasis on program code and more emphasis on the metadata that describes the application can be more readily changed and maintained by individuals who are less technically skilled but are often more familiar with the application domain. Such systems can also be tested using automatically generated testing regimes.

Keywords—Metadata, research software, open science, big data, testing.

1. Introduction

Open science has emerged as a concept that refers to the need to ensure the availability and usability of research outcomes, including the resulting data and the code used to generate the data [1]. One of the key components of open science is the development of research software that accelerates discovery and simplifies access to digital infrastructures [1, 2-6]. These so-called research software platforms are an essential tool for interdisciplinary research, in which collaboration becomes a key factor, and many examples of such platforms have emerged recently in a wide variety of domains, including health, environment and astronomy.

Research software platforms can be built to be increasingly more innovative and powerful than ever before. However, with increasing complexity there is a greater risk of failure if unplanned and untested program scenarios arise [7-9]. As systems age and are changed by different programmers the risk of a change impacting the overall system tends to increase. In contrast, systems that are built with less emphasis on program code and more emphasis on metadata [10] that describes the application can be more readily changed and maintained by individuals who are less technically skilled but are often more familiar with the application domain. Such systems can also be tested using automatically generated testing regimes.

2. iEnvironment

iEnvironment is a distributed open and big data management platform and user gateway for integrated environmental monitoring and modelling (IEMM) related to surface water. The intent of the platform is to support new research groups and the new data and data management and processing capabilities these groups need [11]. The general goal is to: improve significantly environmental science and engineering research on surface water by providing researchers with the ability to access and share environmental data and presentation facilities easily, while also developing best practices through sharing of best-of-breed monitoring and modelling tools. The existing system is a sustainable collaborative research data platform for better access to surface water-related data and models, enabling researchers from multiple disciplines to collaborate and easily discover, access, combine and reuse data and models from multiple sources in order to perform novel forms of analyses. The extended platform will support new ways in which knowledge in surface water science and engineering is created and applied to understand better water availability, quality, and dynamics.

iEnvironment is a multi-tiered platform composed of three main layers, namely: (i) the user interface layer; (ii) the application layer; and (iii) the data layer. The current focus of the next stage of the platform is to leverage existing capabilities and technologies to add new data sources and tools related to surface water for new environmental research groups. These groups need to be supported in terms of new product features to cope with new data, data management and processing challenges. The extended platform will rely on the essential services provided by the Canadian Digital Infrastructure including Compute Canada store,

compute and cloud resources. The project is currently developing the following additional components required by the new research teams:

- Extended service components to support novel dashboards; new data, metadata, and algorithm storage; additional search, publication and access capabilities; new data ingestion and transformation; additional what-if scenario analysis; extra types of notifications; and additional access to Compute Canada resources to access and process new data and algorithms;

- New service application components to support easy access to the surface water monitoring and modelling applications and the relevant datasets, which include tools to model water quality, quantity and dynamics such as river flows and structures and biodiversity;

- New service components to support additional data management capabilities, including new data, metadata and algorithm sharing service; new authentication-based annotations/commentaries component; new data synchronization service; novel data quality control service; new data mapping service; and new application-database abstract interface component;

- New service components to support additional applications, including new process/workflow management and reproducibility services; new spatial variability and spatial pattern assessment services; new cumulative effects support components; new scenario testing and reporting; tool integration services; and new automated testing and test generation services.

3. Metadata-Driven Research Software Testing

In this subsection we discuss how the descriptions of metadata-driven research software systems are being transformed into automated testing regimes that exercise and stress the systems within a systematic and reproducible framework. In this way, we are discussing some aspects of this transformation towards the next-generation of metadata-oriented testing of research software. The question is: How can metadata, that is, data that describes the application, support the creation of automated testing regimes that exercise and stress the systems within a systematic and reproducible framework? In the next paragraphs we discuss some answers for this question

Towards metadata-driven systems. In traditional systems programmers write detailed programs that control user interactions and data accesses in great detail. While software development kits and various utility function packages can be parameterized, the values of parameters are usually defined within the calling functions of the program code. In a system with a greater emphasis on descriptive data (metadata), definitions are stored for at least two key aspects of how that data should be managed. In particular, how data access facilities should retrieve the actual application data such as database queries and file access methods as well as how that data should be presented and accessed by users through a user interface are all stored in a metadata storage facility such as additional database tables. A relatively simple and generalized "kernel" of data access and user interface code can retrieve configuration and context settings from the metadata storage on demand, incorporate additional data from other data sources as needed and perform the required operations.

In systems with more detailed code there is a greater requirement for highly skilled programmers, a scarce and expensive resource, to create and maintain the system. With a greater emphasis on data that describes the application, the need for programmers should be reduced. As well the likelihood of coding errors is also reduced.

Administration of these metadata-driven systems is performed by editing the metadata; no program code changes are required for most changes to the system. Because the metadata is stored in a structured form, such as tables in a relational database system, it can be edited using a simple form-editing facility and these changes can be made by someone without programming experience. They are often made by staff of partner organizations that have more knowledge of the application domain and much less programming capability.

Metadata-driven testing. We have also made significant use of automated test suites for testing and tuning system performance. As many of the systems evolved to a web-based architecture, web-based technologies for testing were adapted to a metadata-driven architecture. One version that is frequently used is based on a PHP WebDriver implementation. In the metadata-driven architecture, collections of WebDriver actions are stored in database tables along with the results of each execution of a suite and each action within the suite. The testing framework can also identify expected results for actions. A small and simple PHP codebase was created to retrieve test suite directives from the database, pass them to the PHP WebDriver interface, retrieve results, store those in the database and then repeat with the next test suite directive.

Test replay and repeatability. Test suites, expected outputs and actual outputs can all be preserved

indefinitely in database tables for future analysis. As a system is changed it is highly desirable to rerun every possible test scenario to ensure that old problems that were believed to have been resolved don't reappear and that new problems don't arise.

Distributed multi-site testing. In the current version of our application development framework, data entry form fields are described with several fields, including the following:

> Entity name: (identifier suitable as an SQL database column name)
>
> Data type: one of {integer, numeric, text, checkbox, select-list, …}
>
> Required: Yes/No
>
> Maximum width: (integer value)
>
> (several other optional fields are supported by the framework but have been omitted for this example)

The "entity name" is used in an HTML <input name="(identifier)"> data field definition; "Data type" defines the type of data to be permitted; "Required" specifies if the form can be entered successfully without any value in this field; "Maximum width" is an optional integer value that specifies the maximum number of characters that are permitted in the data field (usually specified as the "size" attribute of an <input> data entry field.

For testing purposes, the test suite facility supports several test directives, including the following:

```
//          text following "//" is ignored
            open      (url)
            clear     {name=element-name}
            type      (name=element-name, text)
            click     (name=element-name)
            displayScreen     // capture a screen snapshot
            of the application (browser) window
```

These directives enable web pages to be opened, allow form fields to be cleared of any existing value, text to be entered into a form field and an entity on a page to be clicked. Several other directives are supported as described in the Selenium Grid server framework [13].

Transformation-based generation. With both the application and test suites defined as metadata it's possible to define a variety of transformations that use the application definition to generate tests for a wide variety of aspects of the system. For example, if a field on a data entry form is defined as accepting a numeric value in the range zero to 250, tests can be generated to attempt entering data values for zero, 250, -1, 251 and a variety of other possible values. As well, non-numeric values can be attempted to verify that appropriate diagnostics are generated and system response is acceptable.

To test the data entry facility, including field value validation, for a form field named "variable1" of type integer that is a required value with up to three digits on a form with a "Submit" button named "actionSubmit", a test sequence similar to the following can be used:

```
// verify that an empty form field for "variable1" is not permitted…

open (url for the form, possibly including logon/password sequence)

clear "name=variable1"

click "name=actionSubmit"

displayScreen                                   // record the result screen

// verify that a value of "0" in "variable1" is acceptable…

open (url)

clear "name=variable1"

type "name=variable1","0"

click "name=actionSubmit"

displayScreen
```

Logging for diagnosis and additional testing. An additional component of metadata-driven systems, and indeed many traditional systems, can further aid in system testing, maintenance and analysis. Detailed application logging can help greatly to reproduce a request or series of requests that resulted in a program error. Logs should contain enough data to allow the request sequence to be completely reproduced. In multi-user scenarios, such as web-based systems, the timestamps for log entries can be invaluable for accurately reproducing the order and timing of a sequence of requests.

In the iEnvironment platform access logs are stored in database tables and periodically archived, depending on the application usage. Logs are very helpful in identifying when an entity was changed or accessed, what userid is associated with the access and providing the old and new value of the entity.

Logging data can also be used to generate many additional tests for several purposes. From a sustainability perspective, ensuring that problems don't recur is a high priority. As well, focused system performance improvements and tuning can be performed

by exercising a system with frequently used requests, as identified from an analysis of the operational logs. In addition, the test engine can, itself, be tested by using test engine log data.

Software agents for metadata-based testing. Just as a data management system can be created with a greater emphasis on metadata and a reduced emphasis on program code, so also software agents can be defined with more emphasis on metadata. We use the name "Declarative Software Agents" to describe such an entity [12]. By using descriptive data to define the agent's input, rules and actions where log data is recorded as the output of the agent, actions such as autonomous consistency checking between two sites become much simpler to describe and perform. In many data management scenarios repeated instances of the same data are viewed as a maintenance problem and a challenge to be avoided or overcome, but with a declarative agent together with results validation, these scenarios are transformed into opportunities to achieve improved data caching and overall system performance.

Self-testing components. An additional test suite directive, "checkpointDB" will be added to the test suite environment in the future to enable an entire database or some part of it to be saved in a place that can be accessed by a subsequent test operation. When the system either performs an action within a test suite or fails to do so, a comparison against the checkpointed version will be made to determine whether the desired action(s) and only the desired action(s) were performed correctly. Tests like these can be run either in response to a manual request or as the result of a timed or other autonomous decision criteria.

4. Conclusion

As research software systems such as iEnvironment are used, maintained and age, automated testing and detailed logging are two facilities that help to ensure that the system continues to perform to its expected standards. In conclusion, with metadata-driven testing, we believe that in the future, as more metadata is captured and used, it will be possible to automate the generation of more aspects of system testing, in particular: what needs to be tested, when it should be tested and how to test it.

References

[1] Maricial, L., Henninger, B., Scientific Data Repositories on the Web: An Initial Survey, *Journal of the American Society for Inf. Science and Technology*, vol. 61, pp. 2029– 2048, 2010.

[2] Woelfle, M., Olliaro, P., Todd, M., Open Science is a Research Accelerator, Nature Chemistry, vol. 3, no. 10, pp. 745–748, 2011.

[3] Vicente-Saez, R., Martinez-Fuentes, *C.*, Open Science Now: A Systematic Literature Review for an Integrated Definition, Journal of Business Research, vol. 88, pp. 428–436, 2018.

[4] Nosek B., Alter, G., Banks, *G.*, Borsboom, *D.*, Bowman, *S.*, Breckler, *S.*, et al. Promoting an Open Research Culture. Science, vol. 348, no. 6242, pp. 1422–1425, 2015.

[5] Gewin, V., Data Sharing: An Open Mind on Open Data, *Nature, vol.* 529, pp. 117–119, 2016.

[6] Baker, M., 1,500 Scientists Lift the Lid on Reproducibility, Nature, vol. 533, pp. 452–454, 2016.

[7] Jorgensen, P., Software Testing: A Craftsman's Approach, 4th edition, CRC Press, 2013.

[8] Meyers, G., The Art of Software Testing, John Wiley & Sons, 2004.

[9] Orso, A., Rothermel, G., Software Testing: A Research Travelogue, Proceedings of the Future of Software Engineering (FOSE), International Conference on Software Engineering, pp. 117-132, 2014.

[10] Tanenbaum, A., Metadata Solutions, Addison-Wesley, 2001.

[11] Alencar, P., Cowan, D., Mulholland, D., MacVicar, B., Courtenay, S., Murphy, S., McGarry, F., iEnvironment: A software platform for integrated environmental monitoring and modeling of surface water, IEEE International Conference on Big Data (BigData), 2017.

[12] Alencar P., Cowan D., Mulholland D., Oliveira T. (2003) Towards Monitored Data Consistency and Business Processing Based on Declarative Software Agents. In: Garcia A., Lucena C., Zambonelli F., Omicini A., Castro J. (eds) Software Engineering for Large-Scale Multi-Agent Systems. SELMAS 2002. Lecture Notes in Computer Science, vol 2603. Springer, Berlin, Heidelberg.

[13] Selenium Grid server framework https://selenium.dev/documentation/en/grid/