Towards Agile and Test-Driven Development in NLP Applications

Jana Z. Sukkarieh
Educational Testing Service
Rosedale Road
Princeton, NJ 08541, USA
Jsukkarieh@ets.org

Jyoti Kamal
Educational Testing Service
Rosedale Road
Princeton, NJ 08541, USA
Jkamal@ets.org

Abstract

c-rater® is the Educational Testing Service technology for automatic content scoring for short free-text responses. In this paper, we contend that an Agile and test-driven development environment optimizes the development of an NLP-based technology.

1 Introduction

c-rater (Leacock and Chodorow, 2003) is the Educational Testing Service technology for the automatic content scoring of short free-text responses for items whose rubrics are concept-based. This means that a set of concepts or main points are prespecified in the rubric (see the example in Table 1). We view c-rater's task as a textual entailment problem that involves the detection of whether a student's answer entails a particular concept (with the additional challenge that the students' data contains misspellings and grammatical errors). Our solution depends on a combination of rule-based and statistically-based NLP modules (Sukkarieh and Blackmore, 2009). In addition to databases, a JBOSS server (www.jboss.org), and two user interfaces, c-rater consists of 10 modules–eight of which are Natural Language Processing (NLP) modules. Figure 1 depicts the system's architecture. The c-rater engine is where all the linguistic processing and concept detection takes place. Section 2 lists some of the major problems we face while developing such a complex NLP-based application and how our adoption of Agile and test-driven development is helping us.

Example Item (Full Credit 2)	***Concepts or main/key points:***
Figures are given ***Prompt:*** The figures show three polygons. Is the polygon in Figure 1 an octagon, hexagon, or parallelogram? Explain your answer.	**C1:** *The polygon/it is a quadrilateral with two sets of parallel sides OR the opposite sides are of equal length OR opposite angles are equal* **C2:** *The polygon/it has four/4 sides*
Scoring rules: 2 points for C1 (only if C2 is not present) 1 point for C1 and C2 Otherwise 0	

Table 1. Example item for c-rater scoring

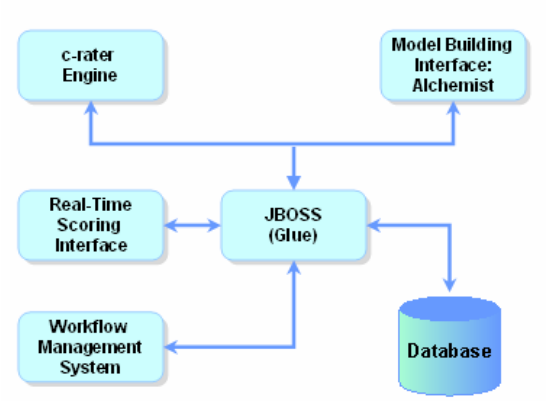

Figure 1. c-rater's System Architecture

2 Major Concerns and Solutions

2.1 Communication

In the past, the implementation of each module was done in isolation and communication among team members was lacking. When a team member

Proceedings of the NAACL HLT Workshop on Software Engineering, Testing, and Quality Assurance for Natural Language Processing, pages 42–44,
Boulder, Colorado, June 2009.

encountered a problem, it was only then that s/he would be aware of some logic or data structure changes by another member. This is not necessarily an NLP-specific problem, however due to the particularly frequent modifications in NLP-based applications (see Section 2.2), communication is more challenging and updates are even more crucial. The adoption of Scrum within Agile (Augustine, 2005) has improved communication tremendously. Although both the task backlog and the choice of tasks within each sprint is done by the product owner, throughout the sprint the planning, requirement analysis, design, coding, and testing is performed by all of the team members. This has been effecting in decreasing the number of logic design errors.

2.2 Planning and Frequent Modification

Very frequent modifications and re-prioritizing are, to a great extent, due to the nature of NL input and constant re-specification, extension, and customization of NLP modules. This could also be due to changes in business requirements, e.g. to tailor the needs of the application to a particular client's needs. Further, this could be a response to emerging research, following a sudden intuition or performing a heuristic approach. Agile takes care of all these issues. It allows the development to adapt to changes more quickly and retract/replace the last feature-based enhancement(s) when the need arises. It allows for incorporating research time and experimental studies into the task backlog; hence the various sprints. The nature of the Agile environment allows us also to add tasks driven by the business needs and consider them highest in value.

2.3 Metrics for Functionality and Progress

Metrics for functionality includes measuring progress, comparing one version to another and monitoring the effect of frequent modifications. This particularly proves challenging due to the nature of c-rater's tasks and the NLP modules. In most software, the business value is a working product. In c-rater, it is not only about producing a score but producing one for the "right" reasons and not due to errors in the linguistic features obtained.

Until recently, comparing versions meant comparing holistic scores without a sense of the effect of particular changes. Evaluating the effect of a change often meant hand-checking hundreds and hundreds of cases. To improve monitoring, we have designed an engine test suite (each is a pair <model-sentence, answer> where model-sentence is a variant of a concept) and introduced automated testing. The suite is categorized according to the linguistic phenomenon of interest (e.g., passive, ergative, negation, appositive, parser output, co-reference output). Some categories follow the phenomena in Vanderwende and Dolan (2006). Some RTE data was transformed for engine tests. This produced a finer-grained view of the NLP modules performance, decreased the amount of hand-checking, and increased our confidence about the "correctness" of our scores.

2.4 Maintenance and Debugging

Until very recently maintaining and debugging the system was very challenging. We faced many issues including the unsystematic scattering of common data structures, making it hard to manage dependencies; long functions making it difficult to track bugs; and late integration or lack of regular updates causing, at times, the system to crash or not compile. Although this may not be deemed NLP-specific, the need to modify NLP modules more frequently than anticipated has made this particularly challenging. To face this challenge, we introduced unit tests (UT) and continuous integration. We usually select some representative or "typical" NL input for certain phenomena, create an expected output, create a *failed* UT, and make it pass. An additional challenge is that since students' responses are noisy, sometimes choosing "typical" text is hard. Ideally, unit tests are supposed to be written before or at the same time as the code; we were able to do that for approximately 40% of the code. The rest of the unit testing was being written after the code was written. For legacy code, we have covered around 10-20% of the code.

In conclusion, we strongly believe like Degerstedt and Jönsson (2006), Agile and Test-Driven Development form a most-suitable environment for building NLP-based applications.

Acknowledgments

Special thanks to Kenneth Willian, and Rene Lawless.

References

Augustine, S. Managing Agile Projects. 2005. Published by Prentice Hall Professional Technical Reference. ISBN 0131240714, 9780131240711. 229 pages.

Degerstedt, L. and Jönsson, A. 2006. *LINTest, A development tool for testing dialogue systems.* In: Proceedings of the 9th International Conference on Spoken Language Processing (Interspeech/ICSLP), Pittsburgh, USA, pp. 489-492.

Leacock, C. and Chodorow, M. 2003. *C-rater: Automated Scoring of Short-Answer Question.* Journal of Computers and Humanities. pp. 389-405.

Sukkarieh, J. Z., & Blackmore, J. To appear. *c-rater: Automatic Content Scoring for Short Constructed Responses.* To appear in the Proceedings of the 22nd International Conference for the Florida Artificial Intelligence Research Society, Florida, USA, May 2009.

Vanderwende, L. and Dolan, W. B. 2006. *What Syntax Can Contribute in the Entailment Task.* J. Quinonero-Candela et al. (eds.). Machine Learning Challenges, Lecture notes in computer science, pp. 205-216. Springer Berlin/Heidelberg.