Software Engineering of NLP-based Computer-assisted Coding Applications

by Mark Morsch, MS; Carol Stoyla, BS, CLA; Ronald Sheffer, Jr., MA; and Brian Potter, PhD

Abstract

The development of production-quality natural language processing (NLP)-based computer-assisted coding (CAC) applications requires a process-driven approach to software development and quality assurance. This should include a well-defined software engineering process, with the specific phases and milestones typically consisting of requirements analysis, preliminary design, detailed design, implementation, unit testing, system testing, and deployment. However, to be successful in a demanding business climate with complex technology like NLP, an organization needs to move beyond a typical "waterfall" approach. The Capability Maturity Model (CMM) defines the key features of a formal software engineering process and provides a ranking system to measure an organization's overall effectiveness in delivering quality software that meets customers' needs. This paper describes the aspects of software development and approaches to testing that yield consistent and high-quality results for NLP-based CAC applications.

Introduction

Applications that utilize NLP present a challenge to the typical methodology in that there is no way to fully specify input requirements for unstructured human language. This can lead to errors in situations when language used in live data goes beyond that used during development and testing. Another common source of error is unintended side effects of changes. Because of the size and complexity of NLP programs, a modification to one part of the system may improve certain situations but break others. For NLP applications to be reliable, the software engineering process must be adapted to minimize these types of errors.

For system design, we describe functionality that can produce robust behavior for new or unfamiliar language. An overview of the software testing process is given for a large-scale environment. Automation of test execution and analysis becomes very important in this type of environment. Thorough regression analysis and a rapid-cycle, iterative approach are two key features of the software testing process.

Background

The Capability Maturity Model (CMM) was developed by the Software Engineering Institute (SEI) at Carnegie Mellon University to help software development organizations measure the maturity of their software development processes and to help software acquirers in evaluating software vendors.[1] More recently, starting in 2001, a new model called the Capability Maturity Model Integration (CMMI) has been developed which builds on the original CMM by integrating the various disciplines—such as software development, systems engineering, integrated product development, and software acquisition—

into a unified model.[2] Like the original CMM, CMMI defines five levels of maturity: (1) initial, (2) managed, (3) defined, (4) quantitatively managed, and (5) optimized. At the initial level, processes are ad hoc, and project success often depends upon the skills of the individual contributors. The higher levels of maturity correspond with more defined and repeatable processes that cover every aspect of the development process. Level 2 includes all of the core activities in the software development process to include requirements management, project planning, project monitoring and control, quality assurance, and configuration management. At level 2 these activities are focused on the project level. For level 3, an organization has defined and repeatable processes that function across multiple projects. A level 3 organization is also capable of improving project performance through product verification and validation, organizational training, and integrated product management. The activities described in this paper primarily fall in level 2 or level 3, which represent high-quality software development processes.

NLP software, like other types of artificial intelligence (AI) software, has often not been developed using a software engineering process. The reasons for this are related to the type of software that is developed and the type of people doing the development. Unlike most software projects, the input requirements for NLP software cannot be fully specified because of the huge scope and variability in human language. Also the programming techniques and algorithms used in NLP software—such as lexical analyzers, parsers, neural networks, Bayesian classifiers, vector processors, and machine learning algorithms—are very complex and require specialized knowledge to create and maintain, even when they are thoroughly documented. Thus, the development of those key portions of an NLP program is often carried out in an evolutionary or experimental manner. Finally, the individuals that develop NLP software are frequently from research institutions or even from domains outside computer science, such as linguistics or cognitive science. These developers often will not have training in the software engineering process.

To develop production-quality NLP software that performs consistently and reliably, there are benefits in applying more structured methodologies. However, critics of the CMM, such as James Bach,[3] do raise valid issues that are of particular importance for AI technology. Three criticisms of CMM that we will mention here are (1) the emphasis of process over the individual, (2) the lack of emphasis on innovation, and (3) the emphasis on activities over results. The algorithms and techniques used in NLP require specialized skills and are typically the product of a series of innovations that are not easy to plan. The methods described in this paper attempt to structure the software development process so that performance is consistent and continuously improving over time by fostering innovative thinking while implementing a robust testing model for measuring results.

Development Process

Developing and maintaining NLP software for CAC applications requires a combination of skills in computer science, linguistics, and medical coding. In our company, personnel with expertise in each of the three individual areas work together to resolve bugs and implement enhancements for the CAC system.[4] Typically, an individual is knowledgeable in one of the three skill areas, so the process is important to facilitate a common understanding of the requirements. In our process, we emphasize the following five practice areas:

1. Requirements Management – Utilizing a defect tracking tool, domain experts file bug reports and requests for enhancements. Each item will include example medical documents to be used for development and testing, and specification of the desired output. Items are assigned severity to assist in ranking, and a priority list for development is created at the start of each update cycle.
2. Rapid Development Cycle – A rapid cycle is motivated by the relatively short timelines to analyze, specify, and implement the regulatory coding changes. For a system in maintenance, this will consist of a series of iterations with weekly build and unit test cycles. For new

development, the iterations will be extended in time to accommodate larger programming tasks.

3. Verification and Validation – In our process the emphasis is on results, so verification and validation are key activities. Verification ensures that the changes work correctly (with zero or minimal regressions), and validation ensures that the right changes have been done. Quality assurance (QA) testing, at both the unit and system levels, verifies system performance. An independent quality assurance team, separate from the NLP development team, performs system verification and validation.
4. Complete Configuration Control – All source code and knowledge base files are maintained within a configuration control system. This ensures that all changes are recorded and documented, and that old build configurations can be recovered if needed.
5. Formal Build and Installation Process – To ensure integrity through testing and deployment, software components ready for systems testing are built into installation packages, and these packages, with written installation instructions, are used to install software into the QA and production environments. Following this process greatly reduces the chances of errors introduced as part of the installation process.

A typical release schedule to address an annual code update, such as for CPT codes, will be about 12 weeks. There are four primary phases in the release cycle:

- Phase 1: Requirements Analysis (Weeks 1–2) – A complete list of bug reports and enhancement requests are compiled and prioritized. This will include all the code updates, broken down into one code change per request. At times, the final code updates will not be known until later in the cycle. In that case, there will be overlap between phases 1 and 2.
- Phase 2: Development and Unit Testing (Weeks 3–9) – Changes are implemented by the development team. The development is interleaved with weekly builds and unit tests to measure progress and detect regressions.
- Phase 3: System Testing (Weeks 10–11) – All changes are completed and verified with a final unit test, and then the software installation package is built and delivered to the QA team. The NLP software is installed in the QA environment, and complete system tests are executed. The QA team analyzes and documents the test results.
- Phase 4: Production Deployment and Documentation (Week 12) – The NLP software is installed into the production environment. User documentation describing the changes is completed and published.

This schedule is similar to many software development projects. What distinguishes these activities is the nature of the development and the testing model. The design and maintenance of this system is described in “LifeCode: A Deployed Application for Automated Medical Coding.”[5] We describe the testing model in the next section. Although the nature of the development approach is specific to the system design so that NLP systems with different designs will have their own type of programming techniques and algorithms, we believe the testing model described here is applicable to a variety of development approaches.

Testing Model

Quality assurance testing measures how well the output of the CAC system compares to the desired results. This is the primary method of verifying the performance of the NLP system. However, the variability of the input and the overall system complexity makes it very difficult to determine how much testing is sufficient for good system verification. The development and QA teams segment the testing into two types, regression and progression, and execute large-scale tests during both unit testing and system testing. Also, the team uses only complete medical documents during testing, which ensures the full context of the text is taken into account. Regression testing is used to detect any negative results of new development. Regression tests are large-scale tests using a statistically significant sample (5 to 7 percent) of monthly production data, for example, yielding more than 150,000 documents for our radiology coding product. Progression tests verify that the changes are functioning as intended. The test size is much smaller, on the scale of hundreds of documents, the same examples identified by medical coding experts during the requirements definition phase.

To reach testing objectives in a large-scale environment, automation of test execution and analysis is essential. We have developed a unit-testing platform and a system-testing platform, both of which can support very large tests (batches of over 150,000 documents). The unit-testing platform encapsulates just the core NLP processing and is used by the development team to run weekly regression tests. The system-testing platform is a copy of the production environment and is used by the QA team for complete system verification. Both platforms store their results in a SQL database, allowing the results of each test iteration to be stored and compared. The analysis of the results is automated through the use of SQL scripts that generate reports showing the changes between any two test runs. A visual evaluation tool is used by medical coding experts to review changes and record a quality score for each change: either better, worse, or neutral. This process and these tools allow the development and QA teams to maintain tight control over the CAC system performance and quickly detect and isolate incorrect results.

Discussion

Even when large-scale regression tests are used to verify performance, an NLP system will confront new or unfamiliar language. In these situations, it is desirable for systems to behave "gracefully." Two qualities of graceful behavior in NLP are (1) the ability to understand more than just patterns of words but also model the underlying semantics and (2) the ability to detect situations when the content of a document is not adequately recognized by the system. The first capability is characterized by the use of supporting ontologies, which are formal representations of knowledge, such as human anatomy, infectious agents, poisons, medications, etc. An ontology allows an NLP system to categorize, relate and generalize the concepts communicated within a medical document. For example, the statement "left calf pain" does not have a direct mapping in ICD-9. However, an ontology can represent that the calf is part of the leg and the leg is a limb, so "left calf pain" can be generalized to "limb pain" which is ICD-9 code 729.5. The second capability means that the system has a method to determine its limits. Similar to a person reading a document and indicating that they do not understand, an NLP system is more useful if it is able to identify text that it does not understand. A variety of techniques can be used to detect these situations, including expert rules, mathematical models, or semi-knowledge (a patented approach used in our system).

A difficult question when evaluating an NLP-based CAC system is determining how well the system can scale across different medical domains and different types of documents. A dictated pathology report will be structured quite differently and will have significantly different content when compared to an inpatient discharge summary. Most current NLP-based CAC solutions are focused on medical specialties, such as radiology, emergency medicine and pathology.[6] NLP has shown that it can scale to perform well for CAC within these particular specialties. Systems that can operate across medical domains require either (1) a different approach to verification and validation than is described here to address an even larger scale or (2) a tighter focus on a single coding system, such as ICD-9, that can be validated within a narrower context. To give users of NLP-based CAC technology confidence in their results, the onus is on system developers to show that they have methods to verify system performance on a large scale.

Conclusion

This system is an example of an NLP-based CAC application that has been developed and maintained using a structured software development process. We believe the activities described in CMM (and now CMMI) benefit both the developer and the acquirer of NLP applications to have greater confidence in a system's performance and reliability. However, we concur with Walker Royce that an iterative, result-driven approach is more effective for software development projects and that an interpretation of the CMM that focuses on the traditional "waterfall" model and produces more documents, plans, and meetings is not a good fit to technology like NLP.[7] To be widely accepted, CAC applications require means of verification that are transparent, repeatable, and scalable. Industry organizations, like AHIMA, can help educate medical professionals and motivate the development of high-quality CAC solutions by defining a verification process applicable across medical domains.

Mark Morsch, MS, is the Vice President of NLP & Software Engineering at A-Life Medical, Inc. in San Diego, CA.

Carol Stoyla, BS, CLA, is the Director of Compliance/Coding/Software QA at A-Life Medical, Inc. in San Diego, CA.

Ronald Sheffer, Jr., MA, is the Manager of NLP Development at A-Life Medical, Inc. in San Diego, CA.

Brian Potter, PhD, is a Principal Scientist at A-Life Medical, Inc. in San Diego, CA.

Notes

1. Software Engineering Institute. “What is CMMI?” February 2, 2006. Available online at http://www.sei.cmu.edu/cmmi/general/general.html, last accessed July 27, 2006.
2. Royce, Walker. “CMM vs. CMMI: From Conventional to Modern Software Management.” *The Rational Edge.* (February 2002). Available online at http://www-128.ibm.com/developerworks/rational/library/content/RationalEdge/feb02/ConventionalToModernFeb02.pdf, last accessed July 27, 2006.
3. Bach, James. “The Immaturity of the CMM.” *American Programmer* 7, no. 9 (September 1994): 13-18. Available online at http://www.satisfice.com/articles/cmm.shtml, last accessed July 28, 2006.
4. Heinze, Daniel, et al. “LifeCode: A Deployed Application for Automated Medical Coding.” *AI Magazine* 22, no. 2 (2001): 76-88. Available online at http://www.alifemedical.com/documents/LifeCodeAIMagazine.pdf, last accessed July 30, 2006.
5. Ibid.
6. AHIMA e-HIM Work Group on Computer-assisted Coding. “Delving into Computer-assisted Coding” (AHIMA Practice Brief). *Journal of the American Health Information Management Association* 75, no. 10 (2004): 48A-H.
7. Royce, Walker. “CMM vs. CMMI: From Conventional to Modern Software Management.”