Automated Recognition of Low-Level Process: A Pilot Validation Study of Zorro for Test-Driven Development

Hongbing Kou and Philip M. Johnson

Collaborative Software Development Laboratory,
Department of Information and Computer Sciences,
University of Hawai'i,
1680 East-West Rd. POST307,
Honolulu, HI 96822, USA
{hongbing, johnson}@hawaii.edu
http://csdl.ics.hawaii.edu

Abstract. Zorro is a system designed to automatically determine whether a developer is complying with the Test-Driven Development (TDD) process. Automated recognition of TDD could benefit the software engineering community in a variety of ways, from pedagogical aids to support the learning of test-driven design, to support for more rigorous empirical studies on the effectiveness of TDD in practice. This paper presents the Zorro system and the results of a pilot validation study, which shows that Zorro was able to recognize test-driven design episodes correctly 89% of the time. The results also indicate ways to improve Zorro's classification accuracy further, and provide evidence for the effectiveness of this approach to low-level software process recognition.

1 Introduction

While software process research has historically focused on high-level, long-duration phases in software development, increasing attention is now being paid to low-level, short-duration activities as well. While a high-level activity such as "requirements specification" might take from weeks to months to complete, a low-level activity such as "refactor class Foo to extract interface IFoo" might take only seconds to complete in a modern interactive development environment.

The frequency and rapidity with which low-level process activities occur creates new barriers to answering classic software process questions, such as: what process is actually occurring (as opposed to what process is supposed to be occurring), what is the impact of a given process on important outcomes such as productivity and quality, and how could a given process be improved and/or tailored to a new domain?

Fortunately, the increasing sophistication of tool support for software development creates new ways to investigate low-level process. By capturing the behavior of developers as represented in their interactions with software development

Q. Wang et al. (Eds.): SPW/ProSim 2006, LNCS 3966, pp. 322–333, 2006.

tools, it may be possible to gain new insight into what low-level processes are occurring during development and their impact on productivity and quality.

This paper presents recent results from our ongoing research into automated support for recognition and analysis of low-level software processes. Our approach leverages the Hackystat framework for automated software engineering process and product data collection and analysis [1], which provides infrastructure for gathering a broad variety of developer behaviors. On top of Hackystat, we developed a generic, rule-based recognizer system for sensor data called "Software Development Stream Analysis" (SDSA). On top of SDSA, we developed a set of rules and other specializations designed to recognize a specific low-level process called Test-Driven Development (TDD) [2]. The system resulting from this combination of Hackystat, SDSA, and TDD-specific extensions is called "Zorro".

Test-driven development is an interesting low-level process to study because substantial claims have been made for its effectiveness. For example, TDD has been claimed to naturally generate 100% coverage, improve refactoring, provide useful executable documentation, produce higher code quality, and reduce defect rates [2, 3, 4]. It would be a significant contribution to the software engineering community to rigorously test these claims in controlled and/or professional settings to better understand the conditions under which they hold, and to further the evolution of the method itself.

Zorro can automatically monitor developer behavior and produce analyses describing certain sequences of behaviors as test-driven development and other sequences of behaviors as non-test-driven development. If Zorro recognizes TDD correctly, then we would have a powerful mechanism for exploring how test-driven development is used in practice and its effects on quality and productivity. The ease with which Hackystat sensors can be installed and the non-intrusive nature of data collection and analysis would make possible both classroom and industrial case studies into TDD compliance, the potential discovery of alternative processes, and the investigation of the impact of TDD on productivity and quality. Finally, Zorro could be used to teach TDD by providing real-time feedback to the developer on whether they are carrying out TDD or not.

Before we can apply Zorro to these TDD research questions, however, we must answer two general validation questions: (1) Does the system collect the behaviors necessary to determine TDD, and (2) Does the recognizer infer the TDD process correctly from the collected behaviors?

In this paper, we present the design of Zorro and the results from a pilot validation study. To do the validation, we needed an independent source of information about low-level developer behavior to compare to Zorro's. For this purpose, we designed and implemented an open source system called "Eclipse Screen Recorder" (ESR), [5]. ESR is a plug-in to the Eclipse IDE that captures a screen image approximately once per second and produces a quicktime movie of the developer's behaviors with respect to the Eclipse window.

Our validation analysis compared the representation of developer behavior captured by ESR to the representation of developer behavior inferred by Zorro,

and classified the frequency and types of differences between these two independent representations. We discovered that Zorro classifies developer behavior correctly 89% of the time, and also discovered ways we can enhance the system in future to improve its classification accuracy further.

The contributions of this research include initial evidence that Zorro can be an effective tool for automatic recognition of the TDD low-level process. Zorro also provides evidence that SDSA is a useful framework for software process recognition. Finally, our results reveal the importance of validation using independent data sources as a component of the process modelling research process, and the usefulness of mechanisms like ESR for this purpose.

2 Related Work

Osterweil has developed a view of software process research that recognizes two complementary levels: macroprocess and microprocess [6]. Macroprocess research is focused on the outward manifestations of process—the time taken, costs incurred, defects generated, and so forth. Macroprocess research traditionally correlates such outcome measures to other project characteristics, which can suggest the impact of process changes to these outcomes, but which suffers from the lack of any underlying causal theory. Bridging this gap is the province of microprocess research, according to Osterweil, in which languages and formal notations are used to specify process details at a sufficient level of rigor and precision that they can be used to support causal explanation of the outcome measures observed at the macroprocess level. Our research most readily fits into the "microprocess" level, except that instead of producing a top-down language, our approach involves bottom-up recognition.

The Balboa research project, like Zorro, was concerned with inference of process from low-level event streams [7]. In Balboa, the event streams were taken from the commit records of a configuration management system, and finite state machines were created that could model the commit stream data observed in practice. Unlike Balboa, Zorro uses instrumentation attached to the developer's IDE, which enables access to much lower-level events than those available through the commit records of a configuration management system. Also, the Balboa research project was retrospective in nature, with the researchers limited to historical project records. Zorro's focus on active development makes additional research possible, such as the validation studies presented in this paper.

Our research also compares in interesting ways to recent work on understanding processes associated with open source software development processes [8]. In this research, "web information spaces" are mined with the goal of discovering software process workflows via analysis of their content, structure, update, and usage patterns. Our approach in Zorro has both strengths and weaknesses relative to this research. A strength of the Zorro approach is that by attaching instrumentation to the IDE, we can capture more detailed information concerning developer behavior than is possible from inspection of web information spaces.

However, this can also be viewed as a weakness, in that this instrumentation creates an adoption barrier not present when mining already publically available information.

Another strand of related research occurs in the areas of knowledge discovery and data mining, in which time ordered input streams are processed to discover and classify naturally recurring patterns. For example, the Episode Discovery (ED) algorithm supports natural forms of periodicity in human-generated timestamp data [9]. While such approaches are an interesting future research area for SDSA, our current episode discovery algorithm uses rules to decide upon episode boundaries regardless of their frequency of occurrence.

Finally, our research relates to prior research on evaluating test-driven design practices and their impact on productivity and quality [10, 11, 12, 13, 14, 15]. In these studies, researchers had limited ability to verify that the programmers who were supposed to be using test-driven development were, in fact, using that methodology. Zorro, if validated, would be an important contribution to this research community by providing a tool to ensure compliance with the process under the experimental conditions.

3 The Design of Zorro

The design of Zorro is highly modular and consists of three basic layers: Hackystat, an extension to Hackystat called Software Development Stream Analysis, and a set of rules and enhancements to SDSA to support recognition of the TDD process.

3.1 Hackystat

Hackystat is an open source framework for automated collection and analysis of software engineering process and product data that we have been developing since 2001. Hackystat supports unobtrusive data collection via specialized "sensors" that are attached to development environment tools and that send structured "sensor data type" instances via SOAP to a web server for analysis via server-side Hackystat "applications". Over two dozen sensors are currently available, including sensors for IDEs (Emacs, Eclipse, Vim, VisualStudio), configuration management (CVS, Subversion), bug tracking (Jira), testing and coverage (JUnit, CppUnit, Emma, JBlanket), system builds and packaging (Ant), static analysis (Checkstyle, PMD, FindBugs, LOCC, SCLC), and so forth. Applications of the Hackystat Framework in addition to our work on SDSA and Zorro include in-process project management [16], high performance computing [17], and software engineering education [18].

3.2 SDSA

Software Development Stream Analysis (SDSA) is a Hackystat application that provides a framework for organizing the various kinds of data received by Hackystat into a form amenable for time-series analysis. Figure 1 illustrates the start

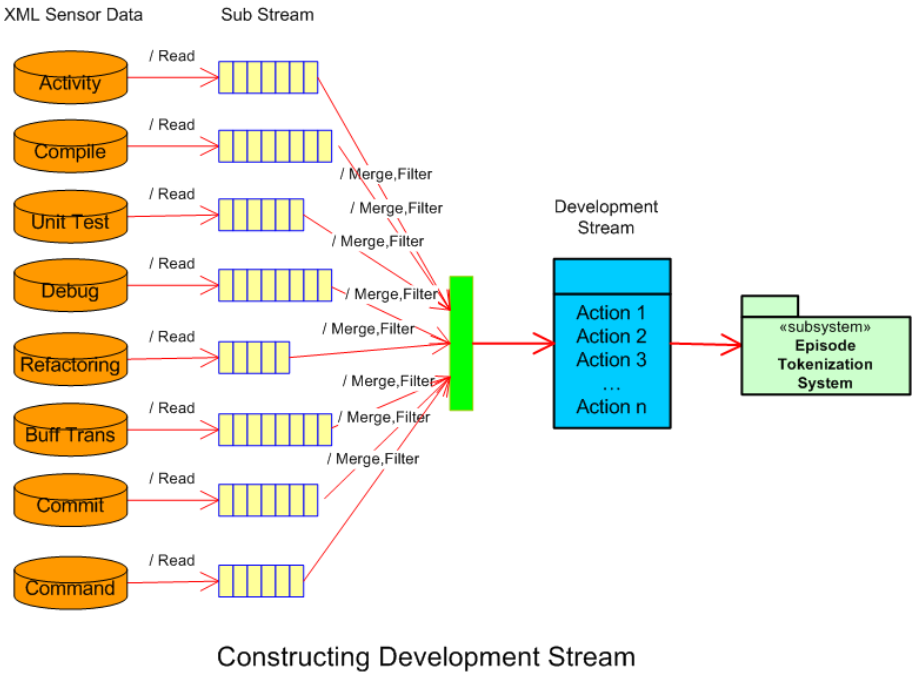

Fig. 1. Development Streams

of this process in which the various kinds of process and product data collected by Hackystat sensors are filtered and merged into an abstraction called a Development Stream.

The next stage of SDSA processing, called Tokenizing, involves partitioning the development stream into a sequence of "episodes" which should constitute the atomic building blocks of whatever process is being recognized. We have developed four kinds of tokenizers for identifying episode boundaries: the commit tokenizer uses configuration management checkins, the command tokenizer uses a distinguished commands or command sequences, the test pass tokenizer uses passing test invocations, and the buffer transition tokenizer uses sequences of buffer transitions. Figure 2 illustrates the process of splitting up the development stream into discrete episodes via tokenizers.

The final step in SDSA is to classify each episode according to the process model of interest. In SDSA, this classification is performed using the JESS rule based system augmented with rules to specify a particular process. Figure 3 illustrates this process.

3.3 SDSA Specializations for TDD

Zorro extends SDSA with rules and analyses oriented to the recognition and classification of TDD behaviors. Figure 4 illustrates the four kinds of behavioral

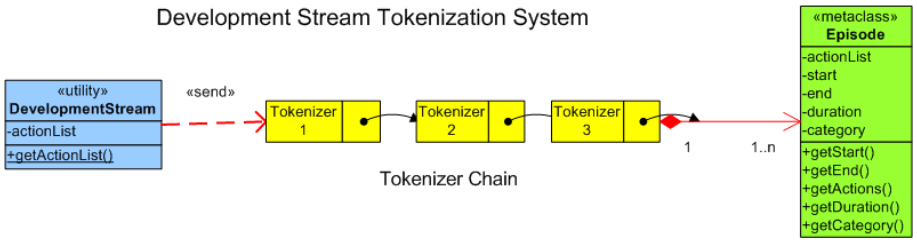

Fig. 2. Tokenizing into episodes

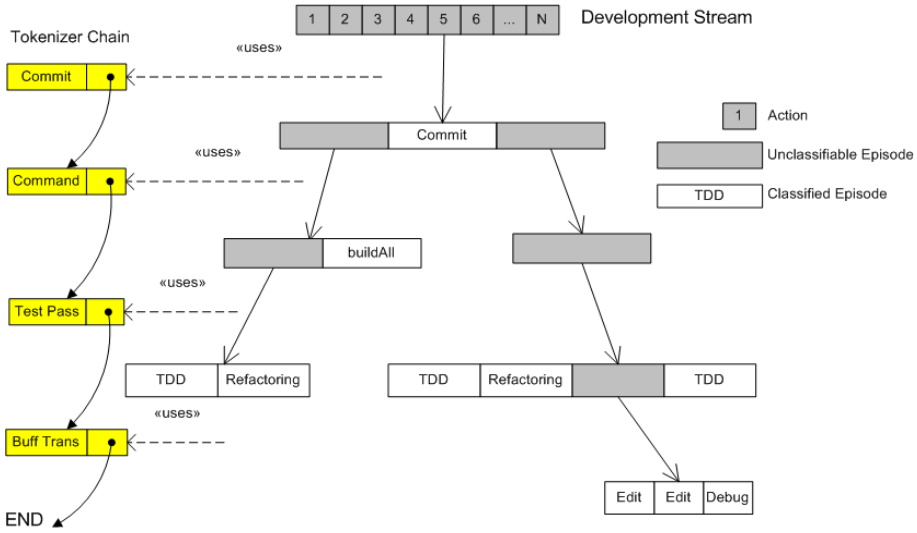

Fig. 3. Episode classification

sequences associated with test-driven development. Zorro includes JESS rules to recognize each of these four kinds of test-driven development behaviors.

Refactoring, in which the developer alters the programs internal structure without affecting its external behavior, is also a valid behavior during test-driven development. Figure 5 illustrates the four kinds of refactoring recognized by the Zorro rule base.

Finally, Zorro includes a user interface in the Hackystat server web application for display of the episodes, their classification, and their internal structure. Figure 6 illustrates the Zorro interface.

4 The Pilot Validation Study

As noted above, in order to feel confident in Zorro as an appropriate tool to investigate TDD, we must answer two basic validation questions: (1) Does Zorro collect the behaviors necessary to determine when TDD is occurring, and (2) Does Zorro recognize test-driven development when it is occurring? To answer

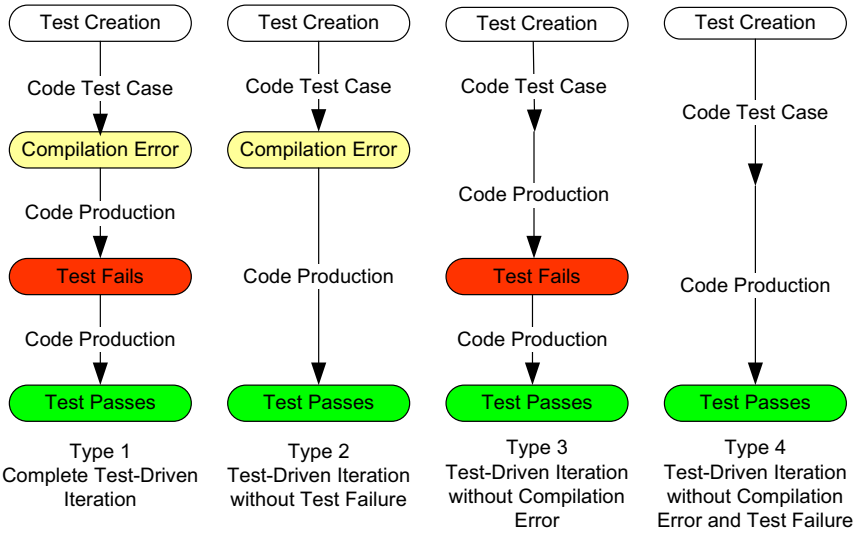

Fig. 4. TDD episode description

these questions, one must somehow gather an independent source of data regarding the developer's behaviors and compare that data to what was collected and analyzed by Zorro.

One approach to validating the system is to have an observer watching developers as they work, and take notes as to whether they are performing TDD or not. We considered this but discarded it as unworkable: the use of a human observer would be quite costly, and given the rapidity with which TDD cycles can occur, it would be quite hard for an observer to notate all of the TDD-related events that can occur literally within seconds of each other. We would end up having to validate our validation technique!

Instead, we developed a plugin to Eclipse that generates a Quicktime movie containing time-stamped screen shots of the Eclipse window at regular intervals. Figure 7 shows the Quicktime viewer with one screen image. The design of ESR allows adjustment of frame rate and resolution: the higher the frame rate and/or resolution, the larger the size of the resulting Quicktime file. We have found that a frame rate of 1 frame per second and a resolution of 960x640 pixels is sufficient for validation, while producing relatively compact Quicktime files (typically 7-8 MB per hour of screenshots). The Quicktime movie created by ESR provides a visual record of developer behavior that can be manually synchronized with the Zorro analysis using the timestamps and used to answer the two validation questions.

Our pilot validation study involved the following procedure. First, we obtained agreement from seven volunteer student subjects to participate in the pilot study. These subjects were experienced with both Java development and the Eclipse IDE, but not necessarily with test-driven development. Second, we provided them with a short description of test-driven design, and a sample problem to implement

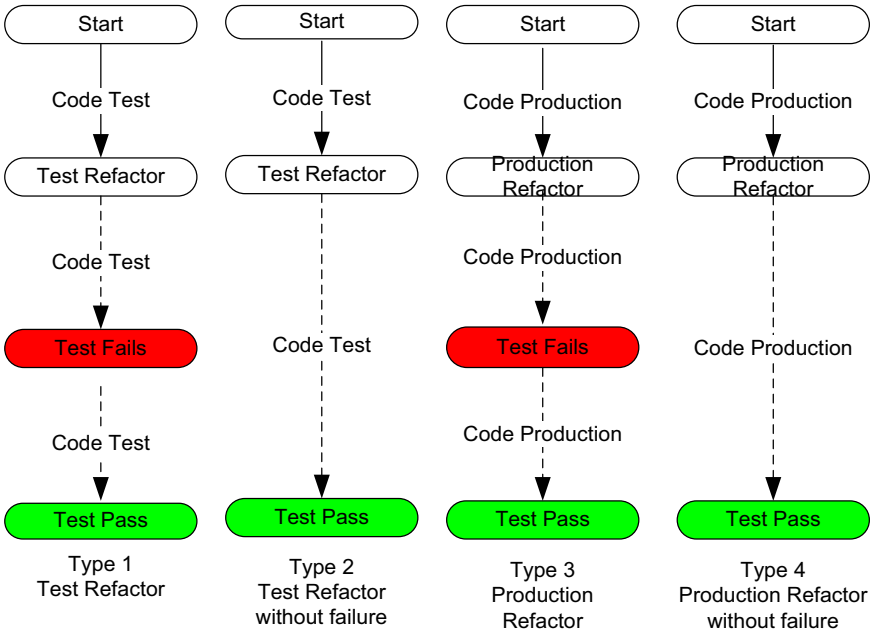

Fig. 5. Refactoring episode description

in a test-driven design style. The problem was to develop a Stack abstract data type using test-driven design, and we supplied them with an ordered list of tests to write and some sample test methods to get them started. Finally, they carried out the task using Eclipse with both ESR and Zorro data collection enabled.

To analyze the data, we created a spreadsheet in which we recorded the results of watching the Quicktime movie and manually encoding the developer activities that occurred. Then, we ran the Zorro analyses, added their results to the spreadsheet, and validated the Zorro classifications against the video record.

5 Results of the Pilot Study

Figure 8 summarizes the results of our analyses. Seven subjects participated, and spent between 28 and 66 minutes to complete the task. Zorro partitioned the overall development effort into 92 distinct episodes, out of which 86 were classified as either Test-Driven, Refactoring, or Test-Last; the remainder were "unclassified", which normally corresponded to startup or shutdown activities.

The most important result of this study is indicated by the "Wrongly Classified Episodes" column, which shows the results of comparing the ESR videos of the developer's Eclipse window to the classifications automatically made by the Zorro recognizer. Out of the 92 episodes under study, 82 were validated as correctly classified, for an accuracy rate of 89%.

The validation analysis also revealed several ways to increase the accuracy of Zorro. First, we discovered that our underlying Hackystat sensor sometimes

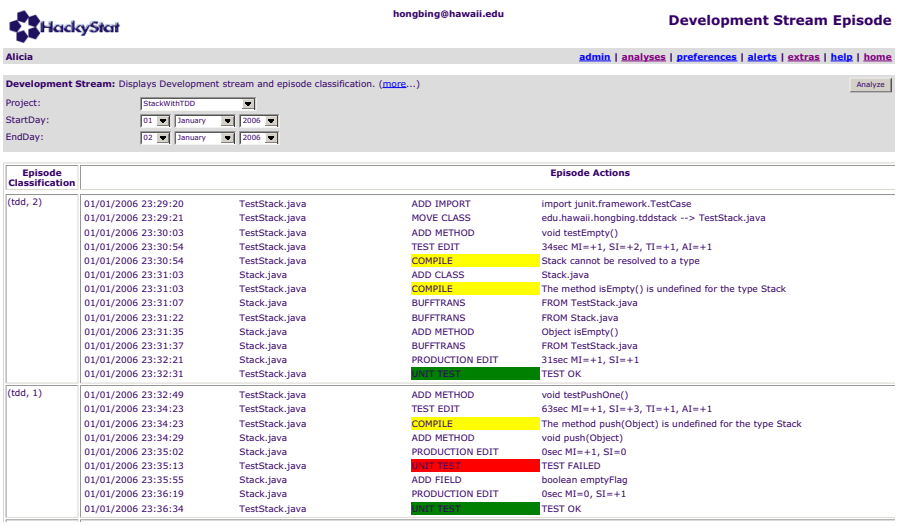

Fig. 6. Zorro interface

failed to record an edit to the program under development when the ESR video showed that the developer made a "quick change" lasting only a few seconds. Second, the sensor also failed to record a compilation error when a change to the production code created a compilation error in the non-active test code. Finally, the current Zorro rule set sometimes failed to partition the development stream along optimal episode boundaries, making it problematic for the classifier to recognize the developer's behaviors during this time period correctly. We intend to fix these issues in the next version of the system, which should raise the accuracy rate significantly.

It is also interesting to review the classification results apart from their accuracy, as they provide insight into the appropriate design of future studies. All four types of Test-Driven Development were recognized by Zorro, although only two of the four types of Refactoring were found. We believe that the simplicity of the software system under development in this study may have been a factor in the limited types of refactoring, and intend to scale up the problem complexity in future studies.

A provocative result of this study is that half the episodes (46) were classified as test-last, even though the subjects were instructed to do test-first development. To some extent, this may also be an artifact of the simplicity of the software under development. But it also reveals a hidden "secret" of test-first development: sometimes, while implementing the code to address one unit test, you can't help but implement additional features as well. At that point, the rational behavior is to implement the unit tests for those additional features, which effectively constitutes test-last design. The nature and frequency of embedded test-last within test-first development is an interesting topic for future research.

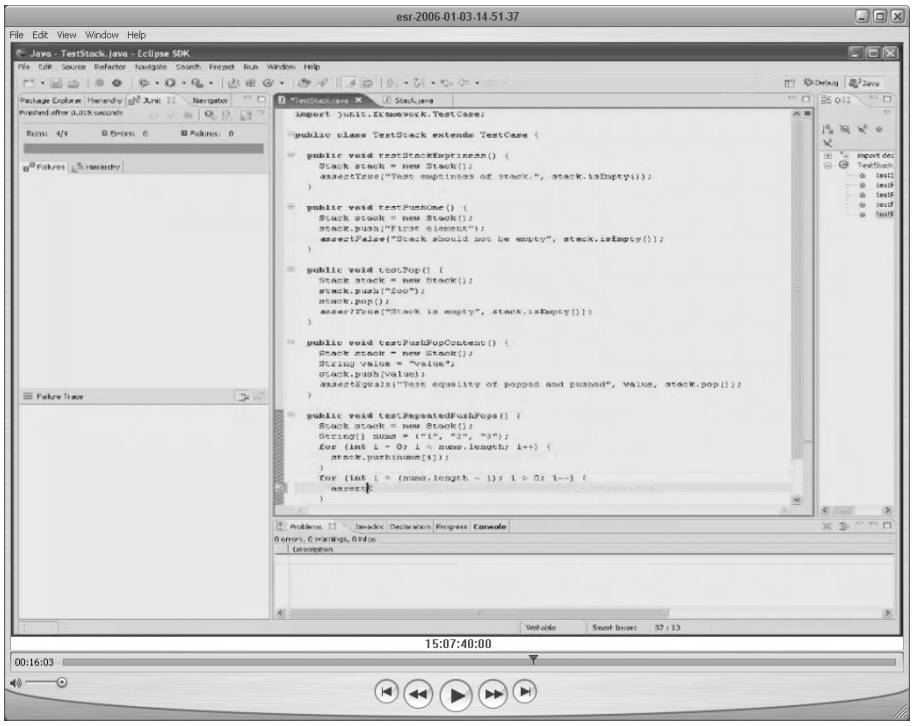

Fig. 7. An ESR Quicktime file

6 Conclusions and Future Directions

The pilot study has been successful in developing an effective validation methodology for the Zorro system, and in identifying several opportunities for improvement to the system that should result in higher classification accuracy in future.

After making these improvements, our next task will be to design and carry out a broad-scale validation study. We intend to expand the total number of subjects participating in the study, and solicit both student and professional developer participation. While we will provide a sample problem to implement in a test-driven design approach, we also hope to collect "in vivo" data from professionals who use test-driven design in their daily work. As before, we will collect both ESR and Zorro data from each subject and analyze it to assess the classification accuracy of Zorro, discover opportunities for improvement in the system, and perhaps discover new insights into the nature of test-driven design.

If the broad-scale validation study results demonstrate that Zorro has achieved high accuracy (95% or better) in recognizing TDD, then we will proceed to the next stage, which is the design of experiments to see how developers use (or don't use) TDD in practice, the factors influencing their decision, and the outcomes of their decisions on productivity and quality.

Subject Index	Duration	Num Episodes	Classified Episodes	Test-Driven Episode Breakdown				Refactoring Episode Breakdown				Test-Last Episodes	Wrongly Classified Episodes	Wrongly Classified Episodes (%)
				Type 1	Type 2	Type 3	Type 4	Type 1	Type 2	Type 3	Type 4			
1	44:53:00	15	15	2	3	1	0	0	1	0	0	8	2	13.3
2	28:17:00	13	13	0	5	0	0	0	0	0	0	8	3	23.1
3	48:00:00	14	14	0	6	3	0	0	0	0	0	5	1	7.1
4	66:32:00	14	14	0	4	0	1	0	0	0	1	8	1	7.1
5	43:14:00	16	11	2	1	0	0	0	1	0	0	7	1	6.3
6	45:57:00	11	11	1	0	1	2	0	0	0	0	7	1	9.1
7	32:40:00	9	8	1	2	1	0	0	0	0	1	3	1	11.1
Subtotal		92	86	6	21	6	3	0	2	0	2	46	10	10.9

Fig. 8. Summary Results

Another area of future research is the application of the SDSA framework to model other low-level software development processes. For example, there are a variety of best practices surrounding when a developer should commit their changes to a configuration management repository which we could model and assess using SDSA along with different sensors and different classification rule sets.

References

1. Johnson, P.M.: Hackystat Framework Home Page. (http://www.hackystat.org/)
2. Beck, K.: Test-Driven Development by Example. Addison Wesley, Massachusetts (2003)
3. George, B., Williams, L.: An Initial Investigation of Test-Driven Development in Industry. ACM Sympoium on Applied Computing **3**(1) (2003) 23
4. Maximilien, E.M., Williams, L.: Accessing Test-Driven Development at IBM. In: Proceedings of the 25th International Conference in Software Engineering, Washington, DC, USA, IEEE Computer Society (2003) 564
5. Kou, H.: Eclipse Screen Recorder Home Page. (http://csdl.ics.hawaii.edu/Tools/Esr/)
6. Osterweil, L.J.: Unifying microprocess and macroprocess research. In: Proceedings of the International Software Process Workshop. (2005) 68–74
7. Cook, J.E., Wolf, A.L.: Automating process discovery through event-data analysis. In: ICSE '95: Proceedings of the 17th international conference on Software engineering, New York, NY, USA, ACM Press (1995) 73–82
8. Jensen, C., Scacchi, W.: Experience in discovering, modeling, and reenacting open source software development processes. In: Proceedings of the International Software Process Workshop. (2005)
9. Heierman, E., Youngblood, G., Cook, D.: Mining temporal sequences to discover interesting patterns. In: Proceedings of the 2004 International Conference on Knowledge Discovery and Data Mining, Seattle, Washington (2004)
10. George, B., Williams, L.: A Structured Experiment of Test-Driven Development. Information & Software Technology **46**(5) (2004) 337–342
11. Muller, M.M., Hagner, O.: Experiment about Test-first Programming. In: Empirical Assesment in Software Engineering (EASE), IEEE Computer Society (2002)
12. Olan, M.: Unit testing: test early, test often. In: Journal of Computing Sciences in Colleges, The Consortium for Computing in Small Colleges (2003) 319
13. Edwards, S.H.: Using software testing to move students from trial-and-error to reflection-in-action. In: Proceedings of the 35th SIGCSE technical symposium on Computer science education, ACM Press (2004) 26–30

14. Geras, A., Smith, M., Miller, J.: A Prototype Empirical Evaluation of Test Driven Development. In: Software Metrics, 10th International Symposium on (METRICS'04), Chicago Illionis, USA, IEEE Computer Society (2004) 405
15. Pancur, M., Ciglaric, M.: Towards empirical evaluation of test-driven development in a university environment. In: Proceedings of EUROCON 2003, IEEE (2003)
16. Johnson, P.M., Kou, H., Paulding, M.G., Zhang, Q., Kagawa, A., Yamashita, T.: Improving software development management through software project telemetry. IEEE Software (2005)
17. Johnson, P.M., Paulding, M.G.: Understanding HPCS development through automated process and product measurement with Hackystat. In: Second Workshop on Productivity and Performance in High-End Computing (P-PHEC). (2005)
18. Johnson, P.M., Kou, H., Agustin, J.M., Zhang, Q., Kagawa, A., Yamashita, T.: Practical automated process and product metric collection and analysis in a classroom setting: Lessons learned from Hackystat-UH. In: Proceedings of the 2004 International Symposium on Empirical Software Engineering, Los Angeles, California (2004)