Automated Assistance for Eliciting User Expectations

Orna Raz[†], Rebecca Buchheit[‡], Mary Shaw[†], Philip Koopman[*], Christos Faloutsos[†]
[†]School of Computer Science, [‡]Civil Engineering Department, [*]ECE Department
Carnegie Mellon University
Pittsburgh PA 15213 USA
E-mail:{orna.raz, rebecca.buchheit, mary.shaw, koopman, christos}@cmu.edu

Abstract

People often use software for mundane tasks and expect it to be dependable enough for their needs. Unfortunately, the incomplete and imprecise specifications of such everyday software inhibit many dependability enhancement techniques because these require a model of proper behavior for failure detection. We offer a user-centered approach for creating a model of proper behavior. This approach is based on satisfying the user expectations—software behavior the user relies on—rather than demanding perfect specifications. It utilizes data mining through a novel template mechanism, to help users make their expectations precise. The resulting precise expectations can then serve as proxies for missing specifications in detecting unexpected data behavior. We concentrate on data feeds: continuous streams of data, a challenging example of everyday software. Using our method on a real world data feed, it took just hours to detect problems that had taken the data providers months to detect independently. These problems surprised even our user—a domain expert that had previously analyzed the same data feed. Systematic analysis further supports the usefulness of our method.

1. Introduction

If all software had perfect specifications—precise, complete, and correct, increasing the dependability of everyday software elements would be straightforward: use the specifications as a model of proper behavior and detect a failure when the software's behavior is outside the specifications.

Unfortunately, specifications are rarely, if ever, perfect. Moreover, it is neither cost-effective nor feasible to strive for perfect specifications for *everyday software elements*—elements incorporated in applications that are neither mission nor safety critical. Yet, the utility of such elements would greatly increase with increased dependability.

Data feeds are an example of everyday software elements and the one we use in our work. A data feed is a time ordered sequence of observations on output. Data feeds may remain under the control of their providers and may have many users relying, in different ways, on behavior the providers did not anticipate. Many challenging, real world software elements fall under the category of data feeds, including Internet services and software elements that process sensor data or perform monitoring activities. Examples include quotes for a stock, weather forecasts, and the truck weigh-in-motion data we use in this paper.

We propose a user-centric approach for coping with incomplete specifications of data feeds. Our method helps users make their expectations about data feed behavior precise. It can then automatically detect *semantic anomalies*—data feed behavior that falsifies these expectations. It applies statistical and machine learning techniques to help discover meaningful information in the data. These techniques precisely characterize various aspects of the data. However, to characterize *relevant* behavior, our method must elicit the user expectations as well. It does so via a novel *template* mechanism. In essence, templates document the predicates of the inference techniques.

The template mechanism is the main contribution of this paper. The case study provides empirical evidence in support of its usefulness.

Each user relies on a data feed in a certain way and expects the behavior of the data feed to support this usage. Therefore, a given user may only care about a subset of the data feed properties. Moreover, a user may care about behavior that is missing from existing specifications or even unnoticed by the providers. However, users' expectations are informal and imprecise, though they are reasonably accurate. For example, a user may expect trucks reported by an on road scale to be physically feasible but may not be able to specify all the properties and values that define such feasibility.

Our approach has the advantages of (1) requiring no knowledge about inputs or implementation details, including source code or binaries and (2) requiring no user data mining expertise. All it assumes is that (1) it can observe the data feed over time, as the user uses it, (2) this usage will tolerate recognition and repair of faults rather than require prevention, and (3) the user has enough domain knowledge to select predicates from a list our method automatically generates. We talk about anomalies rather than failures because our approach, like any dynamic analysis, is potentially unsound. However, our case study shows it can be highly useful in practice.

Our approach is domain independent. Encouragingly, it was able to produce results that were interesting within the application domain of our case study: monitoring systems in civil engineering; domain specific details and results are described in a paper intended for civil engineers [21].

Our case study is a real world truck "weigh-in-motion" (WIM) system using a standard data feed from the Minnesota Department of Transportation. Jackson [13] uses a similar example to introduce his problem frames. Truck WIM data is common in the transportation domain, where civil engineers use it for analyses such as road wear. A scale located in a traffic lane of a road weighs every axle that passes over it. It records the weight on the axle, the time of day, the lane the axle was in, and any error codes. Software components analyze this data to map axle data to vehicles, estimate the speed and length of the inferred vehicles, calculate a measure of load on an axle called ESAL (Equivalent Standard Axle Load), classify the vehicle type, eliminate passenger cars from the data, and (purportedly) filter out unreasonable values.

In our case study, a domain expert (the second author) interacted with the template mechanism to create a model of proper behavior for the WIM data feed from her informal expectations. These informal expectations can be summarized as: (1) vehicles in the same class should be similar and (2) vehicles should be physically feasible. Our method successfully turned these vague expectations into precise predicates. We used the resulting model for anomaly detection and compared it to existing documentation of the data feed. We show that the template mechanism is effective; we measure effectiveness both by the insights the user gains (the usefulness of the process) and the detection and misclassification rates (the usefulness of the resulting model).

2. The template mechanism of our approach

Our approach has three major stages: (1) setting up a model of proper behavior by eliciting precise user expectations; this stage relies on a novel template mechanism and is the focus of this paper, (2) using the precise expectations as a proxy for missing specifications to detect semantic anomalies in the data feed; previous work [22] discussed this stage, and (3) updating the precise expectations to account for evolving system behavior or user expectations; we defer this stage to future work.

These three stages may be viewed as a process governing the data and control flow among the mechanisms underlying our approach. These mechanisms are: (1) the *technique tool kit*—a collection of existing statistical and machine learning techniques that we support and adapt; Section 2.2 provides details, (2) the *template mechanism*—a mechanism that guides the human attention required in making expectations precise using templates that document the predicates a particular technique can output; Sections 2.1–2.2 provide details , and (3) the *anomaly detector*—a mechanism that checks the predicates that are the precise user expectations and reports as anomalies data feed observations that falsify predicates. The anomaly detector utilizes the precise expectations as a model of proper behavior.

2.1. Process and premises

We characterize a predicate inference technique by the types of predicates it can produce. Templates capture the form of these predicates. For example, an inference technique may find a probable range for the values of a given attribute, e.g., the length attribute. The corresponding template would be $\# \leq$length$\leq$ #, where # is a numeric value.

The template mechanism operates as follows:

1. Select tool-kit techniques appropriate to the data and problem.
2. Run the selected techniques to infer predicates over subsets of the data.
3. Ask the user to classify each predicate as either "accept", "update", or "reject".
4. Use the classification to instantiate templates.
5. Use the instantiated templates to filter the output of the tool kit techniques.
6. Give the filtered output to the anomaly detector and present to the user the resulting anomalies and their templates. Allow the user to change the classification.
7. Goto 2 or terminate when the user is happy with the classification.

An inferred predicate is a "complete instantiation" of a template. The template mechanism uses this complete instantiation for templates of "accept" predicates. Classifying a predicate as either "reject" or "update" may make the template instantiation partial by rendering the instantiation of all the numeric values in one or more dimensions void. See Section 2.2 for examples.

The template mechanism treats the predicate inference techniques as black boxes and uses the instantiated templates to filter the predicates a technique infers. It constructs and updates the model of proper behavior from instantiated templates of "accept" and "update" predicates. It will never present the user or the anomaly detector with predicates that match templates of previously rejected predicates. The template mechanism eliminates techniques that are not relevant for this user and data: it will not employ an inference technique if the user rejects all the predicates that are associated with this technique.

Premises of our template mechanism include (1) it is easier for a user to choose from a list of inferred predicates than to create this list, so having a machine synthesize the list is helpful and (2) it is easier for a user to understand expectations about data behavior when presented with examples. It is especially useful to examine examples of anomalous behavior, with the predicates that flagged them as anomalous.

2.2. Inference techniques and their templates

Our technique tool kit currently consists of five existing techniques that is supports and adapts: Rectmix (described below), Percentile (described below), K-means [19] (a clustering algorithm with hard membership), Association Rules [1] (a technique that produces probabilistic rules in an 'if then' form), and Daikon [10] (a program analysis tool that dynamically discovers likely invariants over program executions). We selected these techniques because they expose different aspects of the data and because their output is easy for a human to understand.

To select the most promising techniques for the problem, our method looks at the match between: (1) the data type and a technique (utilizing measurement scales [11]) and (2) the user expectations and the vocabulary of a technique. For the WIM data, this analysis found that the most promising techniques are Rectmix and Percentile: the predicates they output match the data types and describe data behavior relevant to the expert expectations. For this data feed, the other techniques either describe irrelevant behavior or produce predicates that are less precise or redundant with respect to the Rectmix and Percentile predicates. Therefore, we concentrate on the Rectmix and Percentile techniques and describe their templates. Details about the other techniques can be found in [20].

2.2.1 The Rectmix technique

Rectmix [18] is a clustering algorithm that supports soft membership (a point can probabilistically belong to multiple clusters). The clusters it finds are hyper-rectangles in N-space. Rectmix provides a measure of uncertainty called sigma (an estimate of the standard deviation) for each dimension. Anomalies are points that are not within a rectangle. Though clusters rarely have a hyper-rectangle shape in reality, Rectmix has the significant advantage of producing output that is easy to understand: a hyper-rectangle is simply a conjunction of ranges, one for each attribute (see Table 1). Rectmix has two parameters: the number of rectangles and the number of sigmas of uncertainty to allow.

Rectmix always outputs hyper-rectangles, so it has a single template: $\# \leq A_1 \leq \# \wedge ... \wedge \# \leq A_n \leq \#$, where n is the number of attributes. The *dimensionality* of a template is the number of attributes in the template. Table 1 gives an example of user classification for predicates that Rectmix outputs for a subset of the WIM data. The corresponding templates have numeric values in one dimension—the axle attribute—because the user chose to void the other attribute values. For example, the template for the first predicate is #≤length≤# ∧ #≤ESAL≤# ∧3≤axles≤3 ∧ #≤weight≤#.

2.2.2 The Percentile technique

Percentile outputs a probable range for the values of each attribute. The x percentile of a distribution is a value in the distribution such that $x\%$ of the values in the distribution are equal or below it. Percentile calculates the range between the x and 100-x percentiles and allows $y\%$ uncertainty. Percentile only assumes that the distribution values are somewhat centered and is insensitive to extreme values.

Class	Length ∧	ESAL∧	Axles ∧	Weight
Update	20–42	0–.43	3–3	12–29
Update	23–44	0–1.2	2–3	26–47
Reject	13–100	0–.45	2–7	7–40
Update	23–29	0–6.7	2–4	27–71

Table 1. Example of Rectmix predicates classification

Class	Predicate	Template
Update	40≤speed≤88	#≤speed≤#
Update	17≤length≤39	#≤length≤#
Reject	.06≤ESAL≤.9	#≤ESAL≤#
Update	3≤axles≤3	#≤axles≤#
Update	12≤weight≤49	#≤weight≤#

Table 2. Example of percentile predicates classification and instantiated templates

Percentile has a single template: #≤A≤#. Table 2 gives an example of user classification and resulting instantiated templates for predicates that Percentile infers over a subset of the WIM data. Percentile (x=25, y=25%) works well for speed, length, axles, and weight, but not for ESAL (ESAL seems to be exponentially distributed).

Rectmix and Percentile differ: Rectmix finds correlations among common attribute values whereas Percentile simply finds common values for a single attribute.

3. Case study hypothesis

The case study explores the hypothesis that the template mechanism is effective in eliciting precise user expectations and that the resulting precise expectations are a "good enough" engineering approximation to missing specifications, for the purpose of semantic anomaly detection.

The case study supports the hypothesis by showing that (1) The precise expectations are useful in detecting semantic anomalies in the WIM data and (2) The user gains insights about the WIM system through interaction with the template mechanism and through analysis of anomalies.

4. Data and methodology

In a WIM system multiple algorithms process raw sensor data, as introduced in Section 1. Unfortunately, processing and sensors are error prone. Errors may manifest as real vehicles that are not in their correct class (they are very different from other vehicles in their assigned class) or vehicles that are physically improbable. These are the kind of anomalies our expert cares about.

The data we use in our experiments is experimental data the Minnesota Department of Transportation collected by its Mn/ROAD research facilities between January 1998 and December 2000. The data has over three million observations for ten vehicle types that characterize commercial vehicles. Vehicle types differ mainly by their number of axles and whether they consist of a single unit, a single trailer, or multi trailers. The number of observations the system collects varies by vehicle type.

We characterize the WIM data feed as a time-stamped sequence of observations. Each observation has attribute values for a single truck: date and time (accurate to the millisecond), vehicle type (one of ten classes), lane (one of two classes), speed (mph), error code (one of twenty five classes), length (feet), ESAL (dimensionless), number of axles, and weight (kips—kilo-pounds).

We first look for clusters and select attributes (details can be found in [20].). As a result, the template mechanism interacts with the user for each vehicle type (class) separately and gives the selected attributes to techniques in the tool kit.

For the purpose of validating our template mechanism, we selected three out of the ten vehicle types the data contained: the most common vehicle type (type 9, about two million observations) and two additional types (types 4 and 6, about one hundred thousand observations each).

A domain expert set up a model of proper behavior. We gave the model to the anomaly detector. To simulate the nature of on-line data, we divided the data into subsets of two thousand consecutive observations each.

5. Results

We briefly summarize the results of our case study. We present graphs and tables for one of the three vehicle types we examined (type 6). The results for the other two types (types 4 and 9) are rather similar.

The “update” predicates of Tables 1 and 2 are an example of precise user expectations for vehicle type 6.

The *detection rate* calculates how many attributes the model flags as anomalies out of the total number of attributes. It is an objective measure because the results of using the model for anomaly detection are binary: normal or anomalous.

Figure 1 shows the detection rate of the Percentile predicates. The analogous figure for Rectmix is similar.

The y-axis in a plot gives the total number of anomalies in one of the data subsets, according to the criterion the plot specifies, e.g., length anomalies. Notice that the y-axis scale differs among plots. The x-axis is the sequential subset index. The first column in Figure 1 summarizes the number of anomalies for each attribute. The plots in the second and third columns summarize the anomalies that are due to attribute values that are lower or higher, respectively, than the range bounds.

Table 3 summarizes the average detection rate over the subsets of each vehicle type. It gives the detection rate over

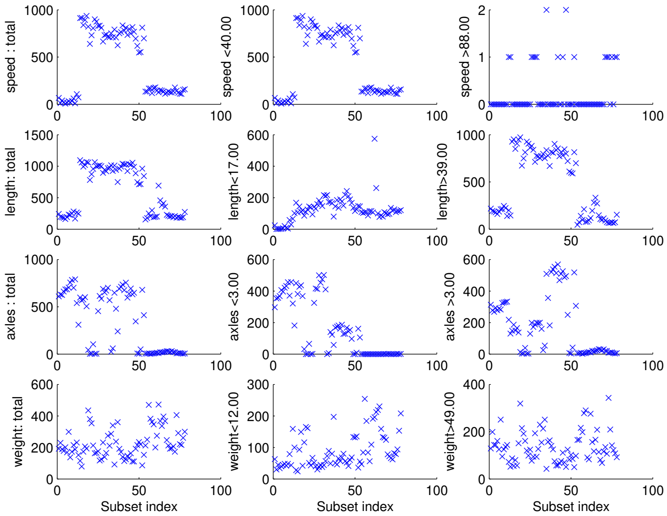

Figure 1. Counts of anomalies detected using Percentile predicates for vehicle type 6

Rectmix	Vehicle type	Average detection rate (%) Total	Length	ESAL	Speed	Axles	Weight
	4	15.5	42.5	7.7		4.4	7.4
	6	10.9	37.7	0.4		0.6	4.8
	9	2.3	5.0	3.4		0.0	0.9
Percentile	4	8.4	8.1		0.8	10.2	14.6
	6	20.2	30.5		22.2	17.0	11.3
	9	0.8	1.0		0.3	0.0	1.9

Table 3. Average detection rate

all attributes and a break-down by attribute.

Small differences in the ranges for length and weight result in large differences in the detection rate, indicating that the values for these attributes are closely concentrated. The exact cut-off point between normal and anomalous is, therefore, not clear from the data.

The overall *misclassification rate* is defined as $\frac{FP+FN}{Nor+Ab} = \frac{Ab+FP-TP}{Nor+Ab}$ [23], where True Positives (TP) are correctly detected anomalous data, False Positives (FP) are normal data falsely detected as anomalous, False Negatives (FN) are undetected anomalous data, Normal (Nor=TN+FP) is data that is actually anomaly-free, and Abnormal (Ab=TP+FN) is data with actual anomalies.

Determining the above measures is subjective even though WIM documentation exists. This is because, on the one hand, the documentation is sometimes incomplete and imprecise, and on the other hand, it sometimes describes behavior that neither Rectmix nor Percentile can express.

To determine Ab, FP, and TP, our expert set constraints based on analyzing both the anomalies flagged by the anomaly detector and the differences between the inferred and documented models. Table 4 summarizes the resulting misclassification rate, averaged over the data subsets of

Vehicle type	Average misclassification rate (%)	
	Rectmix	Percentile
4	8.5	3
6	2.3	2.3
9	1	.8

Table 4. Average overall misclassification rate

each vehicle type. The rates are reasonable for a human to handle.

6. Analysis

The user gained insights by interacting with the template mechanism and by analyzing the resulting anomalies. This is an especially encouraging result because not only is our user a domain expert, but she also previously analyzed this data (though for a different purpose) [5]. In addition, the techniques inferred predicates that confirmed the expert knowledge about the system. This raised our confidence in the results and contributed to better understanding how the system works.

We first enumerate data behavior that surprised our expert. We then present her suggestions for explaining this behavior and enumerate the insights she gained by becoming aware of this behavior.

When looking at the anomalies detected by using her precise expectations as a model of proper behavior, the expert found the following data behavior surprising. This behavior is depicted in Figure 1. The data shows

- A large number of axle anomalies. In particular, the data shows a surprisingly large number of one axle vehicles. However, trucks should have at least two axles and the WIM system software should have detected such anomalies.
- A large number of slow vehicles.
- A large number of over-length vehicles. In particular, for type 6 vehicles, a large number of anomalies have the value of a system built-in length limit.
- A correlation between slow and over-length vehicles.
- A substantial decrease in the above anomalies starting with data subset number 54 (observed at Nov. 1999).
- An exception to all of the above for the most common truck type (type 9): the exceedingly large number of anomalies does not apply to it.

The expert suggested causes for this surprising behavior: The large number of anomalies may be due to (1) inaccurate physical sensing, (2) unintended interaction effects among the various software components. E.g., the component that should eliminate infeasible values— the filtering algorithm—may not properly clean the output of the component that should identify the vehicle type—the classification algorithm, and (3) boundary problems in the classification algorithm.

The decrease in the number of anomalies may be due to a software update in the classification or filtering algorithms or a re-calibration of the WIM scale. The similar behavior of multiple attributes and vehicle types suggests this change or update was system wide. The exception for the common vehicle type suggests that the system is tuned for this type.

The correlation between slow and over-length vehicles corroborates the expert knowledge.

The major insights our expert gained from the above analysis are as follows:

- The data behavior strongly suggests that there was a system-wide change in the WIM system starting November 1999.
- The system (both hardware and software) seems to be calibrated for the most common type of trucks. This, in turn, seems to adversely affect the accuracy of vehicle identification and classification of other types.
- The interaction of the various software components seems to occasionally have undesirable effects.

The data providers confirmed the expert insights and cause analysis, including the system wide change in Nov. 1999. They were unaware of the behavior that surprised our expert until recently. It turns out that the WIM scale has two different modes for weighing an axle. The various algorithms made inconsistent assumptions about the weigh mode. As a result, they occasionally assigned values to the wrong attribute. The next algorithms in the chain did not recognize the problem and made calculations based on the incorrect data. Type 9 vehicles are cleaner because one of the many software providers recognized a problem and made an undocumented correction for type 9. In addition, the system is physically calibrated for this type.

The above strengthens our belief in the usefulness of our method and demonstrates the benefits of automated elicitation support. To set up the model, the expert invested less than 10 hours. The anomaly detection was fully automated and quick (a few minutes). In comparison, it had taken the data providers several months to independently notice the same problems.

7. Related work

The main contribution of this paper is the template mechanism— a means of specifying user expectation and consequently checking these expectations to detect anomalies. Work most closely related includes approaches that either have a similar emphasis on users and their intent [16, 24, 15, 17] or perform various dynamic analysis based on observable behavior [10, 9, 2, 8, 14, 12]. However, that work often requires source code, binaries, or cooperation from the software providers and has a different domain.

We use existing unsupervised learning techniques. Co-training [4] tries to reduce the effort that labeling data for supervised learning requires. Active learning [7] tries to select good training data for a technique. We ask the user to classify the output of a technique, rather than its input.

Many people have been analyzing WIM data. However, most are concerned with transportation issues, not data quality. [6, 5] did domain specific quality analysis.

8. Conclusions

We introduced a promising means for eliciting user expectations about data behavior: the template mechanism. Our case study provides empirical evidence in support of the effectiveness of the template mechanism: (1) The model was useful for anomaly detection. It enabled detecting actual anomalies that the expert cared about: classification problems and unlikely vehicles. In addition, the misclassification rate was reasonable for a human to handle. (2) The expert gained insights about the WIM system. The data providers confirmed the expert insights.

Moreover, the case study results corroborate the benefits of interacting with the template mechanism to make expectations precise and of analyzing the resulting anomalies. Our method: (1) detected hardware and software problems from observed data only. It detected, for example, problems that were caused by mis-calibration, software modifications, or state changes, (2) promptly detected these problems, and (3) increased the understanding of existing documentation. For example, the exact cut-off point between normal and anomalous was not clear from the data though it was clear (for upper bounds) from the documentation, suggesting the documentation bounds may be too strict.

9. Acknowledgments

We thank the Auton Lab [3] for making their dataset processing and analysis software (SPRAT) available to us, the Minnesota Department of Transportation for their WIM data, and Dan Pelleg for allowing us to use his Rectmix code and for his comments. This research is supported by NSF under Grant ITR-0086003, by the Sloan Software Industry Center at Carnegie Mellon University, by the NASA High Dependability Computing Program under cooperative agreement NCC-2-1298, and by the General Motors Collaborative Research Laboratory at Carnegie Mellon. This material is based in part upon work supported by the National Science Foundation under Grant Number 9987871, and by the EUSES Consortium via the National Science Foundation (ITR-0325273).

References

[1] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *SIGMOD 93*, 1993.

[2] G. Ammons, R. Bodik, and J. Larus. Mining specifications. In *POPL*, 2002.

[3] Auton Lab. URL: `http://www.autonlab.org`. Accessed April 2003.

[4] A. Blum and T. Mitchell. Combining labeled and unlabeled data with co-training. In *COLT: Workshop on Computational Learning Theory*, 1998.

[5] R. Buchheit. *Vacuum: Automated Procedures for Assessing and Cleansing Civil Infrastructure Data*. PhD thesis, Carnegie Mellon University, Civil Engineering Dept., 2002.

[6] R. Buchheit, J. Garrett Jr., S. McNeil, and M. Chalkline. Automated procedures for improving the accuracy of sensor-based monitoring data. In *AATT*, 2002.

[7] D. Cohn, Z. Ghahramani, and M. Jordan. Active learning with statistical models. *Journal of Artificial Intelligence Research*, 4:129–145, 1996.

[8] W. Dickinson, D. Leon, and A. Podgurski. Finding failures by cluster analysis of execution profiles. In *ICSE*, 2001.

[9] D. Engler, D. Y. Chen, S. Hallem, A. Chou, and B. Chelf. Bugs as deviant behavior: A general approach to inferring errors in systems code. In *18th ACM Symposium on Operating Systems Principles*, 2001.

[10] M. Ernst, J. Cockrell, W. Griswold, and D. Notkin. Dynamically discovering likely program invariants to support program evolution. In *TSE*, 2000.

[11] N. E. Fenton and S. L. Pfleeger. *Software Metrics*, chapter 2. PWS Publishing Company, 2nd edition, 1997.

[12] S. Hofmeyr and S. Forrest. Architecture for an artificial immune system. In *Evolutionary Computation Journal*, 2000.

[13] M. Jackson. *Problem Frames: Analysing and Structuring Software Development Problems*, chapter 4.3.3, 5.4. Addison-Wesley, 2001.

[14] T. Lane and C. E. Brodley. Approaches to online learning and concept drift for user identification in computer security. In *KDD*, 1998.

[15] P. Langley. The computational support of scientific discovery. *International Journal of Human-Computer Studies*, 53, 2000.

[16] S. McCamant and M. Ernst. Predicting problems caused by component upgrades. In *ESEC/FSE*, 2003.

[17] R. C. Miller and B. A. Myers. Outlier finding: Focusing user attention on possible errors. In *UIST*, 2001.

[18] D. Pelleg and A. Moore. Mixtures of rectangles: Interpretable soft clustering. In *ICML*, 2001.

[19] P. H. R. Duda and D. Stork. *Pattern Classification,*. John Wiley and Sons, 2nd edition, 2000.

[20] O. Raz, R. Buchheit, M. Shaw, P. Koopman, and C. Faloutsos. Eliciting user expectations for data behavior via invariant templates. Technical report, CMU-CS-03-105, 2003.

[21] O. Raz, R. Buchheit, M. Shaw, P. Koopman, and C. Faloutsos. Detecting semantic anomalies in truck weigh-in-motion traffic data using data mining. *Journal of Computing in Civil Engineering (JCCE)*, 2004. Accepted.

[22] O. Raz, P. Koopman, and M. Shaw. Semantic anomaly detection in online data sources. In *ICSE*, 2002.

[23] P. Runeson, M. Ohlsson, and C. Wohlin. A classification scheme for studies on fault-prone components. In *Product focused software process improvement*, 2001.

[24] J. Sousa and D. Garlan. Aura: An architectural framework for user mobility in ubiquitous computing environments. In *IEEE/IFIP Conference on Software Architecture*, 2002.