A Declarative Specification System for More Agile Measurement, Analysis, and Visualization Targeted at Software and Systems Engineering

Thesis Proposal

Larry Maccherone
2009 July 14

Institute for Software Research
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Thesis Committee:

William Scherlis (advisor)
James Herbsleb
Anita Sarma, University of Nebraska
Barry Boehm, University of Southern California

ABSTRACT

There is essentially universal agreement that better understanding (as in knowledge and situational awareness) generally leads to better decisions and outcomes. However, there are competing ways to gain better understanding. The more precise means which include measurement, analytics, and visualization are in competition with less precise means including observation, memories, impressions, intuition, and folk lore (collectively hereafter referred to as "tacit knowledge").

For some types of questions, measurement is clearly superior. "What is the diameter of this axle?", for instance. Even for our domain, most would agree that using a consistent line of code (LOC) measure is a better way to gauge the size of a software product than intuition. Unfortunately, for too large a number of important questions, developers would rather rely upon intuition and tacit knowledge. I identify two reasons for this: (1) the burden of conducting measurement, analysis, and visualization in our domain is too high; and (2) the connection between the things that can be readily measured and the understanding that is useful for making decisions (hereafter referred to as "predictive power") is not strong enough. I believe perception is worse than reality on both of these fronts but efforts to change that perception have met with mixed results. Nevertheless, if we can drive down the cost and increase the predictive power of measurement we can increase the number of questions that are answered with more precise means.

I make the following suggestions that outline an opportunity to improve this situation:

- **Avoid preplanning costs and limitations**. The dominant uses of measurement in our domain operate on the premise that gathering and analyzing data is expensive and to minimize this expense, we must utilize careful preplanning and deliberation. Ironically, the preplanning itself adds to the cost and it comes with its own limitation, namely that questions not posed up front remain unanswered. However, the increasing richness and accessibility of data that is passively gathered erodes the foundation of this reasoning and makes an ad-hoc, just-in-time approach to measurement feasible. Such an approach would allow us to answer the questions of immediate concern even if we did not think of them before we started work.
- **Reinforce and utilize tacit knowledge rather than compete with it.** Agile methods rely more upon tacit knowledge possessed by the team members than they rely upon documentation. A big improvement to current measurement regimes can be achieved if it works in conjunction with tacit knowledge; allowing the user to explore the data, see the forest for the trees, rapidly confirm or refute hypotheses, devise new questions from the output of current analysis, and spiral in on the answers.
- **Iterate rapidly**. Another core belief of Agile is that you are better off trying something, getting feedback, and rapidly evolving it, than you are spending a lot of time up front on design. A more agile regime for measurement would similarly benefit from more rapid iteration. The more measures (and visualizations) we can try, the more likely we are to find ones with greater predictive power (or better ability to express reality).

- **Borrow from the business intelligence community**. The introduction of powerful abstractions for ad-hoc, interactive, and visual analysis (like those found in the business intelligence community carefully adapted to our domain) would further enable this more agile approach.
- **Consider coordination, dependencies, and relationships**. Of particular concern to our domain are issues of coordination. Software development is done on interdependent components by individuals that have a variety of different relationships (social, organizational, geographic, etc.). When these social and technical relationships are aligned things go more smoothly. If our measurement systems took these dependencies into account, they could have predictive power with respect to these important issues.

In this dissertation, I propose a way to declaratively specify a measurement, analysis, and visualization system that leverages the opportunities and suggestions above. Creating a measurement regime using this specification syntax and associated tooling should be a significant improvement over traditional means along the dimensions mentioned (avoiding preplanning, using tacit knowledge, iterating rapidly, utilizing better abstractions, and considering socio-technical relationships). I identify this approach as, "more agile measurement". This dissertation will try to show that we can achieve this greater agility of measurement.

TABLE OF CONTENTS

1. INTRODUCTION, BACKGROUND, AND CONTRIBUTORS

1.1. MEASUREMENT, KNOWLEDGE, AND MAKING BETTER DECISIONS

In the 18th century, Lord Kelvin posited, "that when you can measure what you are speaking about, and express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meager and unsatisfactory kind." [1] However, the connection between measurement and knowledge goes back even further than that. The pyramids of ancient Egypt and Mesoamerican civilization [2] could not have been oriented to the position of the sun on solstice days without mathematical models for both the position of the sun as well as models for engineering and construction. The pyramid builders observed a phenomenon – the motion of the sun – that varied over nested temporal cycles (day and year); took measurements of the position of the sun at various points in the day and year; and then devised a predictive model that was used as an input to their design and engineering efforts.

Fast forward to a similarly wondrous engineering effort in the modern age… sending a man to the moon. NASA developed wide reaching measurement capability for the space program. Techniques for using computed tomography (precursor to medical CAT scanning) to examine components for hidden imperfections were developed so that weight-costing safety factors could be minimized. They did not limit their pursuit of deep understanding and predictive power to the physical realm. They also furthered the cause of measurement in the area of project management decisions. While they invented neither the PERT chart nor the Gantt chart, they applied them both on a scale never before seen and they evolved the concept of a hierarchical work breakdown structure (WBS) to accommodate the massive effort to get a human to the moon, keep him alive while doing it, and somehow bring him back to the earth.

With these tools, NASA engineers and project managers were able to understand the critical path, and make decisions about where to allocate resources and focus risk mitigation. They had a clear understanding of the work remaining, and once they made sufficient progress, they could use the data from their earlier work to not only predict the effort remaining but to also decide how much confidence they should place in their prediction.

For both the pyramid builders and NASA, it is precisely the prediction power of their models that enabled engineering decisions. Without the ability to predict the outcome of various alternatives, decisions are made on hunch and intuition; they are, "of a meager and unsatisfactory kind." Such decisions may lead to a usable outcome but they rarely result in wondrous products and are almost certainly not accomplished at minimal cost.

A simple classroom experiment that I invented and have conducted many times illustrates this point on a much smaller scale. Three students from the class are lined up an arm's length apart. At one end of this human chain, a bowl full of marbles is placed; at the other end an empty bowl. These students are expected to pass the marbles down this virtual "bucket brigade" until all the marbles are moved. A short distance away, three other students are placed around a bowl with the same

number of marbles. They also have an equidistant empty bowl. The students in this "bucket runner" group are expected to pick up a marble, walk to the other bowl, drop it, and return for another until all of their marbles are moved. Once everyone is in place and expectations are explained, I take a survey of the class asking which group they think will finish first. The vast majority of folks think the bucket brigade will finish first. The last time I conducted this experiment, all of the roughly 30 people in the room (except for me) thought the bucket brigade would finish first – not surprising since it is generally accepted that a bucket brigade is the most efficient way to bring water to a burning building. However, their intuition is wrong. The myth of the superiority of the bucket brigade is busted. Every time, the bucket runners finish first… and by a wide margin (25-50% from memory).

It's not terribly important that marbles get moved from one bowl to another in the most efficient manner, but the impact of technical, project management, and process decisions on a software or systems engineering effort can be critical the success of the project. Should we do inspection? Should we do test driven development (TDD)? How much of each and in what combinations should we do? Should we live with a code base that has accumulated significant technical debt or should we refactor it? Or re-write it from scratch? Should sub-teams be organized by specialty or should the organizational structure match that of the architecture? What changes in architecture and team structure lead to minimal effort wasted on coordination? What should we do next? To what completion date should we commit?

Without data and analysis to help make these decisions, our conclusions are likely to be similarly "meager and unsatisfactory."

The dissertation proposed by this document attempts to bring about an incremental improvement both in the adoptability and effectiveness of measurement capability for software and systems engineering. The expected result for users of my approach include (1) better understanding of the current situation, and (2) an improved ability to predict the likely cost and impact of alternative choices. This is in turn is expected to improve the quality of decisions that are made – decisions about what elements to include in their processes and in what proportions and order; decisions about how best to organize teams and structure technical artifacts; decisions about what to do next and what commitments to make.

Throughout this introduction, I identify a number of "motivating observations". They are distinguished from "assumptions" because the hypothesis and questions posed later do not directly depend upon them. The research is still valid without them. However, the degree to which they prove to be true will generally correlate with the potential impact this work will have on the state of the practice. The first such motivating observation is the fundamental motivation for this work.

Motivating observation 1: *Better measurement leads to better understanding, which leads to better decision making, which, in turn, leads to better outcomes.*

The document is organized as follows:

- Section 1 includes this introduction and a discussion of relevant background on the history of measurement for software and system. It includes a discussion of several areas from which my

proposed approach draws. It identifies the opportunity posed by emergence of Agile methods and it describes the nature of the improvement in measurement necessary to capitalize on this opportunity as criteria to be used to evaluate my proposed approach.

- Section 2 is an illustrating example of how an Agile team could use the approach that I am proposing.
- Section 3 presents the proposition of my dissertation and the research questions I will need to validate to defend that proposition.
- Section 4 contains a discussion of the expected contributions for this research.
- Section 5 describes similar existing approaches to create a language for specifying measurement.
- Section 6 provides a status update on the current development of the apparatus I intend to use for this research. It includes a brief explanation of the syntax for the declarative measurement, analysis and visualization specifications that my proposed system uses.
- Section 7 provides elaboration on a particularly significant contribution of the proposed approach to measurement – that is measurement and visualization that allows users to reason about socio-technical relationships and dependencies. This elaboration is accomplished by describing an example tool, Tesseract, which is the output of earlier research in which I participated to explore socio-technical measurement and visualization.

1.2. ADVANCED STATISTICAL TECHNIQUES FOR SOFTWARE AND SYSTEMS

While Lord Kelvin, pyramid builders, and NASA all used measurement to gain better understanding and make decisions, there is another area whose example is more revealing for the discussion herein; that is the application of statistical process control (SPC) to the manufacture of durable goods. Starting in the 1960s and 1970, experts in the art of measurement including Deming [3, 4] and Crosby [5] showed automobile manufacturers (and later, electronics manufacturers) how to define their processes, measure them, and keep them under control. Rather than "test" each product as it came off the line, they could take key process measures from a sample of the parts and sub-assemblies in progress. Only if those measures showed that the process had gone out of control, was intervention necessary. This approach was applied so successfully in the electronics world that Motorola (among others) was able to create "six sigma" processes, which is defined as processes which produce no more than 3.4 defective parts per million opportunities [6, 7]. The output of these processes can often be shipped without *any* final testing. This approach, and its derivatives, is generally accepted today as the way to mass produce physical goods.

It should come as no surprise, then, that we have attempted to apply these concepts to the area of software and systems development. The general idea behind most process improvement efforts for software in the last couple decades has parallels in the realm of manufactured goods. The basic idea behind arguably the most well known of these efforts, the Capability Maturity Model (CMM), and its derivatives, is that you first gain control of your processes by clearly defining them and accomplishing consistent execution (levels 2 and 3) and then you use measurement to improve them (levels 4 and 5) [8, 9]. Six Sigma techniques are frequently used in conjunction with CMM/CMMI level 4 and 5 efforts [10].

1.2.1. THE TEAM SOFTWARE PROCESS

An example of a high maturity CMM-based process is the Team Software Processes (TSP) [11]. TSP is a well defined, mostly level 5 process created by Watts Humphrey, who also created the CMM and stewarded by the Software Engineering Institute (SEI), who also has stewardship of the CMMI. TSP calls for teams to maintain the following **base measures**:

1. **Size**. Line of code (LOC) accounting is expected to be maintained for the production of code parts. Added, deleted, changed, and generated/reused numbers are expected to be gathered at various levels of the product hierarchy. Size data in other units (pages for design documents, the number of tables and fields in a database, etc.) is expected to be gathered for other artifacts.
2. **Time**. The time spent working on each artifact is tracked and broken down by work phase (design, design review, code, code review, unit test, etc.).
3. **Defects**. Defect logs are maintained both during and after production. Defect entries are also tagged for the phase they were injected and removed and the time needed to find and fix each defect is gathered.
4. **Schedule**. The original as well as any updated commitments are considered base measures in TSP. This data is used to gauge how close the project is tracking to the current commitment.

These base measures are then combined to form **derived measures**. The most simple derived measures are just ratios of these base measures including productivity in LOC/hour, defect density in defects/KLOC, and defect removal rate in minutes/defect. These simple derived measures when combined with the phase and/or work breakdown hierarchy can be used to calculate things such as how much time "should" be spent on a code review of a particular part. TSP recommends that code review rates not exceed 200 LOC/hour. Code parts with review rates significantly higher than that can be flagged for additional review.

Because of these measures, users of TSP have deep understanding of where their time is spent and this information is explicitly used to make project management decisions but it is also useful in motivating behavior change. For instance, TSP-trained developers are often convinced of the benefits of code review by comparing how much time it takes to remove a defect found in code review compared to how much time it takes to remove one found in testing. This is a particularly powerful motivator because the data is for their own team or their own personal process (developers on a TSP team typically spend 60-120 hours in Personal Software Process (PSP) training before starting on their TSP team). Similarly, the need for robust design and design review practices is supported by their own data comparing how much time it takes to remove a defect injected in design compared to one injected in code. These examples are particularly useful when trying to accomplish developer behavior change.

Perhaps the most advanced measures in TSP center around estimating and schedule tracking. The TSP includes no less than four methods for producing an estimate for a particular artifact. The more advanced of these consider historical data on similar parts and include the ability to create a statistical prediction interval for the final estimate. However, I have been involved in the coaching or training of over thirty TSP teams and only a few of them were able to maintain stable technology,

team membership, and process structure long enough to gather enough consistent data to take advantage of this powerful estimating and prediction interval capability. The data is still useful for the additional purposes mentioned above (among others) but the benefit side of a cost/benefit calculation is diminished by the low likelihood of being able to take advantage of this powerful estimating and prediction interval capability.

Another advanced estimating technique that uses multiple regression, COCOMO [12, 13] which comes from the work of Barry Boehm among others, also suffers from this difficulty. Users of COCOMO can benefit immediately from use of the effort multipliers supplied with the method, but differences associated with their particular technology, skills, process, etc. may not be accurately reflected until they gather enough data to calibrate the model to their situation.

Putting the high burden of data gathering aside, users of a calibrated COCOMO model and the more advanced TSP estimation techniques can achieve impressive schedule predictability. Similarly, TSP teams have been very effective at producing very low defect software. Figure 1 shows schedule deviation data for early TSP teams published by the SEI. This data was taken from 20 projects from 13 organizations and compiled in a report by Noopur Davis, et. al. in 2003 [14]. These 20 TSP projects finished an average of 6% later than their original commitment. The data from the Standish Group as reported by the SEI, indicates that over 50% of all software projects are either canceled or finish over 100% late, with the average schedule error for unsuccessful projects being 222% late.

Figure 2 shows defect density data from the same report. While I know from personal experience that the numbers for more recent projects do not exhibit the 20 to 100 fold improvement in defect density shown in this chart, every organization that I have worked with has experienced significant improvements in defect density. I believe that the greatly increased use of design review, code

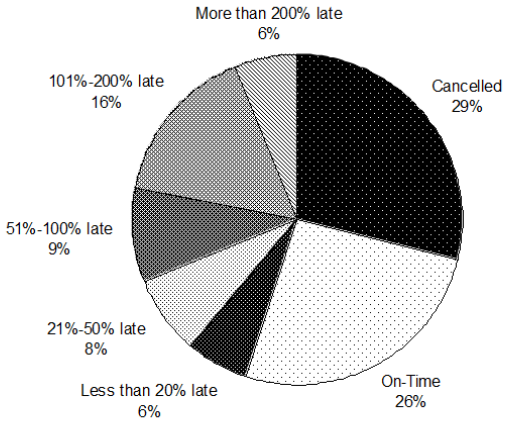
Performance Category	TSP Impact Study (2003)	Typical Industry Performance (Standish Group)
Schedule error average	6%	
Schedule error range	-20% to +27%	

FIGURE 1. TSP SCHEDULE DEVIATION RESULTS

Performance Category	TSP Impact Study (2003)*	Typical Industry Performance
System test defects per thousand instructions	0.4 avg. 0.0 to 0.9	2 to 14
Released defects per thousand instructions	0.06 avg. 0.0 to 0.2	1 to 7
System test effort (% of total effort)	4% avg. 2% to 7%	40%

FIGURE 2. TSP DEFECT RESULTS

review and inspection techniques are the primary reason for this improvement. The measurement that TSP teams gather as well as the measures that are gathered by developers during PSP training, are key factors in motivating the increased use of these practices. Further, the ability to monitor the work in process with measures such as the code review rate allow teams to ensure compliance with their own policies with respect to the reviews and inspections.

While it is hard to say how much of these benefits are attributable to the measurement aspects of TSP as opposed to other aspects (process steps, roles, etc.), it is clear that measurement is central to the TSP message and its stewards certainly believe that measurement is a key reason for these outcomes.

Motivating observation 2.1: *A significant portion of the benefits realized by projects using high maturity processes is attributable to behavior modifying feedback and improved decision making that comes from their use of measurement.*

With these outcomes, why then have we have not seen universal adoption of these techniques as we have for their parallels in the realm of manufactured goods? The majority of software development is still produced by organizations that would not achieve a level 2 assessment for CMMI. Organizations who are not taking advantage of these methods may be missing out on significant opportunities for improvement.

1.2.2. GOAL-QUESTION-METRIC, PRACTICAL SOFTWARE MEASUREMENT, AND ISO/IEC 15939

No discussion about software measurement would be complete without mention of Vic Basili's goal-question-metric (GQM) [15-17] and the similar guidance provided in Practical Software Measurement (PSM) [18] which is captured in the specification ISO/IEC 15939 [19]. While the TSP is prescriptive (think "instance") and the CMMI includes criteria for assessing the maturity of a measurement practice (think "specification"), GQM and PSM are meta models that can be used to institute a measurement capability tailored to the user's situation (think "abstract base class plus pattern guidance on how to turn it into an 'instance'").

GQM consists of the following:

- **Goal**. Identify organizational objectives.
- **Question**. Devise questions whose answers would indicate progress (or the lack thereof) towards those goals.
- **Metric**. Document measures that will answer these questions. Then, clearly identify (1) base measures that are to be gathered, (2) formula necessary to calculate derived measures from the base measures, and (3) a description of the indicators that will be used to answer the questions (threshold level, trend up or down, etc.).

PSM strongly references GQM and uses the same terminology. ISO/IEC 15939 provides these definitions and describes in detail what elements are needed in a fully defined "information product". The PSM community also offers tools and sample measure definitions.

Note that the CMMI's guidance on how to approach measurement for software and systems is similar. The specific goals and practices for CMMI's Measurement and Analysis process area starts as follows:

- Align Measurement and Analysis Activities
  - Establish Measurement Objectives
  - Specify Measures
  - Specify Data Collection and Storage Procedures
  - Specify Analysis Procedures

The underlying assumption supporting the guidance about software measurement in GQM, PSM and CMMI is that measurement is expensive and we must be deliberate and careful so that the benefits of measurement are maximized and the costs minimized. However, if we can drive down the cost of measurement, this foundation is eroded and developers can be more opportunistic about its use. One of the biggest difficulties of a measurement regime is the burden of gathering base measures during the development process. However, the almost universal use of source code repositories with revision tracking capability provides a sufficiently rich platform for a large portion of measurement needs. In fact there is an annual conference dedicated to mining source code repositories [20]. Further, there is growing use of tools that passively gather other useful base measures including work-item history, defect logs, automated build-logs, communication records, and at the extreme end of the spectrum, IDE telemetry.

There are efforts underway to both (1) increase the richness of the data that is automatically gathered, and (2) make it easier to pull together the data for analysis. Three prominent examples that each take a different approach are Hackystat, IBM's Jazz, and Patrick Smacchia's code query language (CQL).

Hackystat [21, 22] is an open source framework stewarded by Philip Johnson at the University of Hawaii that enables the gathering of data from the tools that are commonly used by software developers. It has "sensors" for various source code repositories, build tools, automated testing frameworks, static analysis tools, IDEs, etc. These sensors normalize the data into XML files which can then be gathered and processed for analysis.

Jazz [23, 24] takes a different approach. It is an integrated platform which includes repositories for artifacts, work-items, requirements, and communication records, among others. An advantage of this unified approach is that the relationship between entities is automatically gathered as the work is produced. For example, when using a source code repository that is separate from the work-item database, it is often difficult to accurately identify which artifact changes are associated with which work-item entries. In Jazz, this difficulty is avoided [25].

Different even still is the approach taken by Patrick Smacchia's code query language (CQL) [26] and similarly targeted work by Santanu Paul and Atul Prakash [27]. CQL is an SQL-like language that allows a user to answer any number of questions about their code. The easiest way to describe CQL is by example. From Smacchia's web site:

- Which public methods have more than 30 lines of code?

```
SELECT METHODS WHERE NbLinesOfCode > 30 AND IsPublic
```

- Which classes implement System.IDisposable?

```
SELECT TYPES WHERE IsClass AND Implements "System.IDisposable"
```

- Which methods have been refactored recently and is not thoroughly covered by tests?

```
SELECT METHODS WHERE CodeWasChanged AND PercentageCoverage < 100
```

CQL was first available for Microsoft's .NET (NDepend) and recently became available for Java (XDepend). CQL actually crosses over the line between simple data gathering and into the realm of measurement and analysis that this dissertation is targeting. This is not a problem. Rather, the platform that CQL provides is ideal for the type of analysis and visualization that I am proposing just as SQL databases are the most common platform for business intelligence analytics and visualization.

If these sources of information can be more effectively used in an ad-hoc fashion, planning costs are minimized and the costs of calculation and analysis are only incurred at the moment where they are needed to make a particular decision. Questions that were not thought of in advance can be answered and the costs associated with questions that were anticipated to be significant but turned out not to be important, will never be incurred.

Motivating observation 2.2: *Traditional approaches to software measurement incur increased pre- planning and deliberation costs for two reasons: (1) so the opportunity to gather the data is not lost once the development process has begun, and (2) as a tradeoff for reduced gathering and analysis costs. However, the increasing richness and accessibility of data that is passively gathered erodes the foundation of this reasoning and makes an ad-hoc, just-in-time approach to measurement feasible in terms of cost and effectiveness.*

1.3. COORDINATION, DEPENDENCIES, AND RELATIONSHIPS

Let us revisit the bucket brigade example from section 1.1. A secondary point to be drawn from my bucket brigade classroom experiment is that the key difference between the two teams is one of coordination. The bucket brigade team incurs overhead "costs" from the relatively simple coordination necessary to accomplish each marble handoff. Dropped marbles resulted in "rework."

The cost of coordination in a software project can be much more expensive (and troublesome) than a simple marble handoff. Coordination costs and decisions associated with managing them can be a significant factor in determining the success or failure of a software effort [28].

Development environments increasingly reflect the fact that artifacts, developers, and tasks are intrinsically bound together in a software project. While editing a file of source code, for example, many other artifacts are likely to be relevant. Considerable research effort has focused on using a variety of techniques such as static code analyses[29,30], task definition [31], text analysis [32], and records of prior developer activity to identify these related artifacts [33] and make them easily accessible when they are likely to be useful.

There is also an increasing interest in understanding and using relationships among individuals in a team to improve software development. Research has focused on increasing awareness among developers about each other's relevant activities, and on using the social relations among developers to identify implicit teams [34] or to predict software defects [35]. Such efforts often draw on social network analysis (SNA).

So far, these two streams of research have mostly been separate from each other. Yet, both of these sets of relationships – the technical and the social – become much more useful when they are considered together [36]. It is difficult, for example, to judge whether a given pattern of communication is adaptive or dysfunctional without understanding the dependencies in the tasks being undertaken by the communicators. For instance, developers who are modifying interdependent code modules but not communicating may indicate potential future integration problems. Research has shown that development work proceeds more efficiently when patterns of communication match the logical dependencies in the code that is being modified [28].

This match between the networks of artifacts and the networks of people, has a long history [37, 38], but has only recently become a focus of research in software engineering. Understanding and using analysis showing the degree of match, or congruence, between the social and technical aspects of a project is vital for supporting collaboration and coordination on software projects [28]. While developers have intuitively known this for a long time, and software architects actively engage in social engineering while creating architectural design [39], we have relatively few tools or practices that provide socio-technical information in useful, actionable ways.

This aspect of measurement is hardly addressed in the literature and almost never systematically addressed on development projects; perhaps because other, more mature areas from which our measurement practices are drawn (manufacturing and business) do not exhibit difficulties associated with coordination and socio-technical congruence to the degree that we experience them in software and systems engineering. Regardless of the reason for this deficiency, the introduction of abstractions that enable a more systematic way to manage these issues could turn out to be the most significant contribution of this dissertation.

I draw heavily upon the work of Jim Herbsleb, Anita Sarma, Marcelo Cataldo, and Patrick Wagstrom for this aspect of the research. They are largely responsible for expanding my original thinking on

measurement to include this aspect. The abstractions, analysis, and visualizations that my work will hopefully make more accessible, is largely derived from their research.

Motivating observation 3: *A significant improvement in the impact of measurement can be accomplished if it is particularly powerful at capturing measurement that considers coordination, dependencies, and relationships.*

1.4. AGILITY OF MEASUREMENT

This section is a discussion about adding agility to our software measurement practices. While it considers the characteristics of Agile methods of software development, it is not a marketing analysis on how to make measurement be adopted by Agile practitioners. Rather, it attempts to identify the characteristics that would constitute a more agile approach to measurement. This may help to bring the benefits of measurement to Agile practitioners but I am equally interested in helping to bring more agility to plan-driven methodologies.

TABLE 1. ADOPTION OF CMMI

	2009
Not aware	13%
Not using	29%
Analyzed and rejected	4%
Investigating	8%
Trying to reach Level 2	12%
CMMI Level 2, 3 or 4	20%
CMMI Level 5	14%

Participants: 392 (January 2009)

1.4.1. THE RISE OF AGILE

In the last few years, we have seen the rise of Agile methods; and despite the fact that Agile methods have existed for a much shorter period of time, they have similar, if not higher adoption rates. Compare the two surveys [40, 41] shown in Table 2 and Table 1.

Note that respondents who are "Not aware", "Not using", or "Analyzed and rejected" is 30% for Agile, and 46% for CMM/CMMI. Also note the rapid rise of Agile adoption from 2005 to 2008 and the fact that the Agile survey lags the CMM/CMMI one by eleven months. The difference is likely to be larger now. One must be cautious about comparing these two sets of adoption rate data. The definition of CMMI is very clear and this survey might not include positive responses from users of

TABLE 2. ADOPTION OF AGILE METHODS

	2008	2005
Not aware	13%	26%
Not using	13%	16%
Analyzed and rejected	4%	3%
Investigating	14%	14%
Pilot projects	8%	4%
Partial implementation (adoption of some agile practices)	17%	17%
Partial deployment (some projects are using this approach)	14%	12%
Deployed (all new projects are using this approach)	17%	8%

Participants: 512 (February, 2008), 232 (2005)

derivative approaches like iCMM or the Team Software Process. Contrast that to Agile where it is possible for an organization to claim they are "doing Agile" when they are only doing *some* of the Agile practices – as evidenced by the 17% "Partial implementation" figure above. Nevertheless, when looking at these figures, it is hard to conclude that CMMI is completely satisfying ongoing process improvement needs and Agile methods are filling a void left by plan-driven methods.

Motivating observation 4.1: *The growing use of Agile methods represents an interesting opportunity to rethink our approach to measurement for these environments. An approach to measurement that aligns well with Agile, might be effective at bringing the benefits of measurement to projects and organizations where traditional approaches to software measurement are resisted.*

1.4.2. AGILE: OBSERVATIONS, CHARACTERISTICS, AND IMPORTANT CONCEPTS

Before an approach a more agile approach to measurement can be identified, it is import that we understand the nature of agility with respect to software development. Existing Agile methods are the best source of information from which to draw these observations and characterizations. In this section, I rely heavily upon the work of Barry Boehm and Richard Turner documented their book, Balancing Agility and Discipline: A Guide for the Perplexed [42] as well as a related paper [43] they wrote regarding their observations on the subject. This work is particularly relevant because it focuses on bringing together agility and discipline which mirrors my dual goals of adding agility to plan-driven methods and bringing the benefits of measurement to Agile methods.

Boehm and Turner's observations on balancing discipline and agility are shown in Table 3.

Further, they note that while Agile methods are lightweight, this characteristic is not the distinguishing factor. I extracted the relevant characteristics and important concepts from chapter 1 of Balancing Agility and Discipline. They are shown in Table 4.

To compliment Boehm and Turner's observations with respect to people and the emergent aspects

TABLE 3. BOEHM AND TURNER OBSERVATIONS

NUMBER	OBSERVATION
1	Neither Agile nor plan-driven methods provide a methodological silver bullet that slays Fred Brooks' software engineering werewolf [36].
2	There are definite home grounds for pure agile and pure plan-driven methods, although the actual extremes are rarely populated.
3	Future applications will need both discipline and agility.
4	Some balanced methods are emerging.
5	Build your method up. Don't tailor it down.
6	Focus less on methods – more on people, values, communications, and expectation management.

TABLE 4. CHARACTERISTICS AND IMPORTANT CONCEPTS FOR AGILE

CHARACTERISTICS AND IMPORTANT CONCEPTS
Iterative and incremental
Actively involve users to establish, prioritize, and verify requirements. Presents actual product for feedback rather than requirements. Uses retrospectives for process feedback.
Rely upon tacit knowledge as opposed to documentation
Self-organizing, meaning teams determine the best way to handle work
Emergent such that the processes and work structure is recognized during the project rather than predetermined
Just-in-time design (aka "You aren't going to need it (YAGNI)") which implies delaying the addition of infrastructure or robust design until it is needed
Embracing change
Simple design
Fast cycle/frequent delivery

of agility, I offer my own observations about the need for users to sometimes guide the direction of analysis.

Linking of measurement with decision can be convoluted using existing measurement systems because there are many different kinds of decisions and each is informed by a different slice through the observables. The presumption of plan-driven approaches to measurement is that these needs can be anticipated in advance. However, when exposed to the harsh reality of the real world where change is the norm, this is a fragile proposition. What is needed to supplement current systems are tools and mechanisms that allow the user to guide the analysis; to take into account partial or uncertain knowledge, insight, and intuition; which lead to hypothesis that might explain the situation; which can then be rapidly and cost effectively confirmed or refuted (using the system); so that more hypothesis can be tried; generally increasing the chances of a correct final interpretation of the situation; and better decisions.

Motivating observation 4.2: *The presumption made by plan-driven approaches that measurement needs can be anticipated in advance does not always hold true. An approach to measurement that allows the user to guide the analysis using tacit knowledge can supplement a measurement regime when unexpected information needs arise.*

1.4.3. RAPID SEARCHING AND HILL CLIMBING

There are a huge numbers of potential measures. Looking at the list of papers presented annually at the Mining Software Repositories (MSR) conference [20] and you can start to see the potential opportunity. However, only a small subset of all possible measures contain useful information. If a measurement system can be evolved more rapidly and with lower cost for each iteration, more alternatives can be tried and the chance of finding more predictive measures and analyses (as well as more easily understood visualizations) is likely to increase.

Further, for the iterations that result in a better measure than the previously best one, we need a way to memorialize the improved version so that it is available for comparison with the next attempt. Saving the entire source code for each version is possible for hand-coded measures and visualizations but this would be cumbersome. What is needed is way to concisely and precisely describe each iteration; a representation that captures the essence of the measurement without the distracting details of how it is calculated and displayed. The artificial intelligence (AI) community uses the phrase "hill climbing" when discussing this concept [44, 45].

Motivating observation 4.3: *If a measurement system can be evolved more rapidly and with lower cost for each iteration, more alternatives can be tried and the chance of finding more predictive measures and analyses increases. A concise and precise representation to capture the essence of the measurement is needed both to enable this rapid searching and to memorialize a measurement once an improved version is found.*

1.4.4. CRITERIA FOR A MORE AGILE APPROACH TO MEASUREMENT

Boehm and Turner's observations number 1 through 4 cannot be readily turned into criteria for a more agile approach to measurement. However, I take encouragement from them because all four observations touch on the topic of using Agile and plan-driven approaches together. Brining the measurement benefits traditionally associated with plan-driven methods to a wider audience by

TABLE 5. CRITERIA FOR A MORE AGILE APPROACH TO MEASUREMENT

NUMBER	CRITERIA
1	**People over process**
1a	**People**. Relies upon tacit knowledge; System is interactive, allowing the user to guide the direction of the analysis
1b	**Communication**. Has communication, not compliance, as its primary purpose
1c	**Coordination.** Includes capability to take communication and coordination data as input
1d	**Self-organizing**. Teams determine which measurements to gather
2	**Emergent rather than predetermined**
2a	**Not predetermined**. Built up from examples and patterns rather than tailored down from a provided base
2b	**Just-in-time**. Measurement needs are identified and satisfied during the project rather than predetermined (must be easy to get started with simple and usable measures)
3	**Evolved rapidly**
3a	**Feedback from "doing" rather than "thinking".** Relies upon feedback from use of actual measurement system; System allows user to instantly jump from measurement design to measurement usage and back again
3b	**Iterative and incremental**. Embraces change
3c	**Rapid searching and hill climbing.** Includes a concise and precise syntax that captures the essence of a measurement definition so that alternatives can be rapidly tried and memorialized once a better one is found
3d	**Use of patterns and powerful abstractions.** Uses a relatively small set of common measurement patterns and abstractions that maximize rapid evolution

evolving an approach that aligns better with Agile methods is a goal of this dissertation. Conversely, another goal is to bring more agility to the measurement systems currently used in conjunction with plan-driven methods. Both of these goals fit with the message of their first four observations.

Pulling from Boehm and Turner's remaining two observations as well as their characteristics and important concepts list plus including my own observations, I propose criteria for a more agile approach to measurement found in Table 5. I will not spend time here elaborating more on these criteria nor will I attempt to evaluate the current design of my tools against this criteria. Both are likely to evolve significantly during the course of the research. I include them here merely as a starting point and to show that identifying satisfactory criteria from the literature and observations is possible.

Motivating observation 4.4: *It is possible to identify the characteristics of a measurement regime that would (1) make the adoption of measurement by Agile practitioners more likely, and (2) bring more agility to the measurement practices of plan-driven methods.*

1.5. BUSINESS INTELLIGENCE (BI)

Manufacturing is not the only possible source for ideas on how to evolve a more agile approach to measurement for software and systems. The business community has mature business intelligence (BI) practices which are way ahead of current practice for software and systems in certain aspects. BI practices of data warehousing and data mining have the luxury of assuming the universal availability of passively acquired data. The business community has also gone through several generations of tools that are focused on ad-hoc, just-in-time analysis and several characteristics are almost universally present in BI tool usage. I identify the relevant ones here:

1. Separate tooling for extract transform and load (ETL) functionality and analysis/visualization functionality [46]. Even when the same vendor offers both types of capability, it is provided by separate products. Mimicking this approach would benefit our domain because novel approaches to analysis and visualization are currently held back by the lack of availability of normalized data accessible in a unified manner. Currently our data is stored in myriad of siloed databases.
2. An emphasis on visual output to enhance comprehension of tabular information [47]
3. User-driven subtotaling and other forms of aggregation that take into account hierarchies in the data being analyzed [46,48,49]
4. Enabling the user to see the forest for the trees (primarily an outcome of 2 and 3) [47]
5. Rapid cycling from analysis to visualization and back again during which the output of one query is combined with tacit knowledge and intuition to form another query eventually spiraling in on the information need [47]

1.5.1. OLAP

Characteristics 2, 3, and 4 are enabled by an abstraction referred to as an online analytical processing cube, or simply an OLAP cube (aka: multidimensional cube or a hypercube).

1.5.1.1. OLAP DIMENSIONS:

As implied by these aliases, an OLAP hypercube can have any number of dimensions. Each dimension is associated with a particular field in a flat or relational database and the values along that dimension are all the values that show up in that field for the data set being analyzed. For instance, a retailer might specify "Sleeve length" as a dimension and possible values might be "long" and "short". "Sex M/F" might be another dimension.

1.5.1.2. OLAP DIMENSION HIERARCHY

Some dimensions may have hierarchy that is used to organize subtotals and other aggregation. A dimension that is commonly specified with a hierarchy is the time dimension. A user might want to look at sales data by month, quarter, or year. When provided with this hierarchy, the OLAP system will calculate the measures for each level/value combination in the hierarchy. Product family information is also commonly hierarchical. For example:

- Shirts
  - Casual
    - T-shirts
    - Other
  - Dress
    - Button down
    - Straight color
  - Sports
    - Golf
    - Workout
- Pants
  - …

For software and systems, a more useful time hierarchy might be iteration, version, and release. Likewise, our "product hierarchies" are generally specified as packages and indicated with a "/" in the filename. Successful use of this OLAP abstraction in our domain requires these adaptations.

1.5.1.3. OLAP MEASURES

The cells in this hypercube are filled with measures that are the aggregation of data found in the source database. For instance, "sum" type measures are very common and used to aggregate sales data for particular cell in the hypercube. Mens, Casual, Long sleeve shirts sold in April-09, for instance. Each cell can be filled with more than one measure. A user might want to know the count of the number of shirts sold as well as the aggregated sales price.

1.5.2. APPLICABILITY OF OLAP TO SOFTWARE AND SYSTEMS MEASUREMENT

The OLAP abstraction is applicable to software and systems measurement. Looking back at the description of base and derived measures for TSP found in section 1.2.1, it is not hard to see how the aggregating functionality of OLAP could be used to tally the total time spend in a particular process phase for a particular part. This type of aggregation need is frequently specified throughout the software measurement literature – in TSP, in PSM specs, in CMMI instances. However, I have found no similar attempt to apply a general abstraction such as OLAP to this issue. Each sum or

count is specified with a combination of text and formulas. The closest I have seen is an attempt by David Tuma to generalize the by-phase aggregation done in his Software Process Dashboard tool, which is often used with PSP and TSP. Tuma refers to this operation as "bucketing" [50]. While useful, it is more akin to the sum() formula found in spreadsheets than to OLAP.

Further, while there exist measures that are explicitly targeted at the package level, the practice of looking at a generalized measure in the context of several layers of hierarchy is not found. This ability to perform analysis at various levels in a hierarchy is particularly powerful for managing detail and allowing the user to see the forest for the trees. A user can look for a pattern at the package level that would be hard to see at the file level. A user can compare data from one iteration that went well to one that did not to identify potential reasons for the difference.

Motivating observation 5: *A generalized abstraction for hierarchical multidimensional aggregation similar to the OLAP abstraction found in the business intelligence community would enable precise and more concise measurement definitions and facilitate the use of software measurement in a more interactive and ad-hoc fashion.*

2. ILLUSTRATING EXAMPLE

Agile methodologies assume that the team is continually refactoring their code so that at any moment, the current system represents the simplest and most evolvable design possible. Unfortunately, this is a fragile proposition for several reasons. First, it is in conflict with the Agile expectation that usable new functionality will be delivered every iteration. "We don't have time to refactor. We have to get these features out." Second, the YAGNI philosophy encourages developers to just do the minimum to implement this new functionality. "Refactoring would help a future that may never come." Finally, many organizations start to use Agile methods before their teams are manned by developers with sufficient skills in design patterns or automated testing. "We'll get started with Agile now and they'll pick it up as we go." So, new Agile teams after several iterations, frequently find that they have accumulated significant technical debt.

The data and scenario used in this section was inspired by an experience I had with a team that I was coaching at a large developer of financial software. This team had just completed its seventh iteration of its first Agile project. The process that the team was using was developed specifically for the vendor and has elements of both TSP (a mostly CMM level 5, plan-driven process) and Scrum (the most popular Agile method). It utilized 30 day cycles so they were in their seventh month of development. Their original commitment to their management had them delivering a certain set of functionality by the end of the thirteenth iteration so they were a little more than half way done.

During the planning meetings between iteration 7 and iteration 8, the team discussion centered around the fact that their velocity has slowed considerably over the last several iterations. Velocity is a measure used by Agile teams to track how many "story points" of functionality the team can produce each iteration. If it did not pick back up, they would not be able to deliver the desired amount of functionality in the time remaining. The team initiated a discussion to diagnose the

problem and someone suggested that the problem was due to accumulated technical debt. The resulting nods and groans indicated consensus.

So the code has gotten crufty. What do they do about it? Do they dedicate an iteration to refactoring? Will that cost them more than it will save in the remaining time? How do they figure out if it will be worth it?

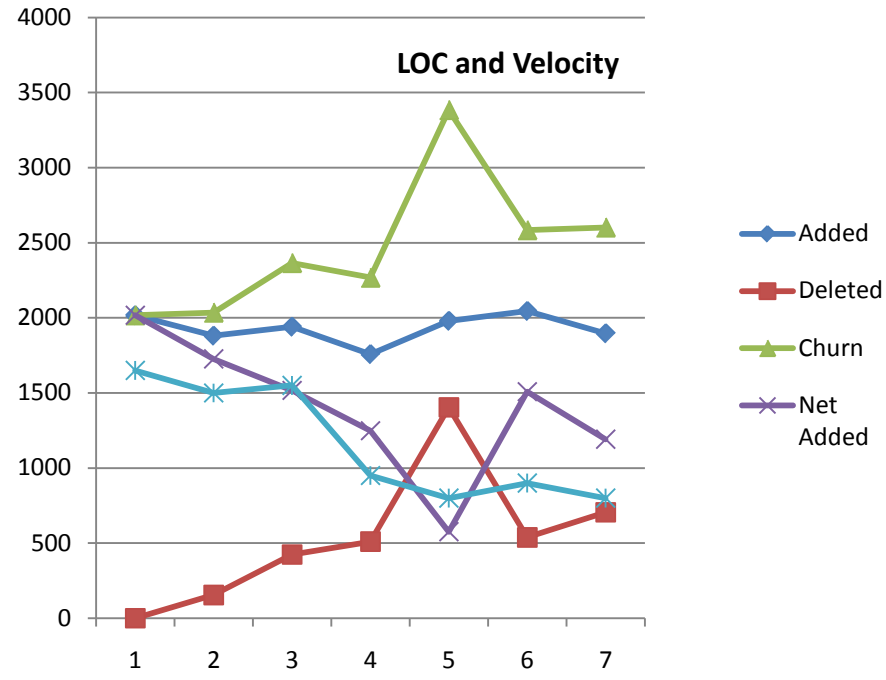


FIGURE 3. LOC METRICS AND VELOCITY

The discussion continued and someone remember that the two folks, Mary and John, on the team who are primarily working on the UI layer actually did a refactoring of their code in iteration five. They had been part of a lunch-time reading group which had just finished reading the Head First Design Patterns book. Mary and John wanted to refractor their code to apply what they had learned and to make the code more evolvable and they got permission from the team to do so for iteration 5 because there was very little UI work chosen for that iteration.

Mary and John indicate that they enjoy working with the code more since the refactoring. Great, but the team only has data showing the velocity of the entire project, not the portion that John and Mary worked on. How can they tell if the refactoring that they did was worth it for them so they can use that outcome to make the decision about whether or not to spend a full iteration refactoring the entire codebase. Specifically, they desired a way to answer the question, "Will the cost of an iteration dedicated to refactoring have a net positive RIO before the first release date?

The team was aware of research that showed that code churn metrics and dependencies could be used to predict failures [51]. They were also aware of research that indicated complexity metrics could be used to identify good candidates for refactoring [52,53]. Maybe code metrics could be used to see if the refactoring that Mary and John did was worth it. So the team queried the source code repository and generated the graph in Figure 3.

From the graph, the team noticed that their velocity generally tracks the net added LOC (R^2=0.75) which is calculated by subtracting the lines deleted from the lines added. Churn (added plus

deleted) is mostly level but they noticed the spike in iteration five for the deleted and churn measures. That was the iteration where the UI sub-team did their refactoring.

So, at least for this limited data set, the net added metric can be used as an indicator of velocity. While they *do not* have velocity data for just the UI sub-team, they *can* extract net added metrics for just that portion of the code. They modified their code query scripts and generated the graph in Figure 4.

The UI sub-team actually had a negative number for net added LOC in iteration five, but in the two iterations following, they were back up to their original level. Furthermore, it is only two data points, but it doesn't appear to be decreasing as rapidly as it was before.

Using the experience of the UI sub-team as a model for what would happen to the rest of the team (not guaranteed but their best judgment based upon the information they had at the time), the team projected that they would be better off spending the next iteration paying down their technical debt.

This approach to measurement allows users to explore the data, and put it in context with tacit knowledge that they possess (like the refactoring done by the UI team). It allows the user to notice patterns and then helps them take the next step. It assumes that, given the right tools, the folks doing the work are better able to interpret the data than a predetermined indicator.

The analysis in this example was done using custom Python scripts to mine the source code repository, more Python to manipulate the data and calculate derived measures, and a spreadsheet to graph the data. It was not difficult, but it was not trivial either and had a software engineering researcher (me) not been involved with the project, the team would have made the decision based upon their intuition. If successful, the tools proposed by this dissertation will be in the hands of teams like this in the future. They will use it to answer questions of immediate concern and make more informed decisions.

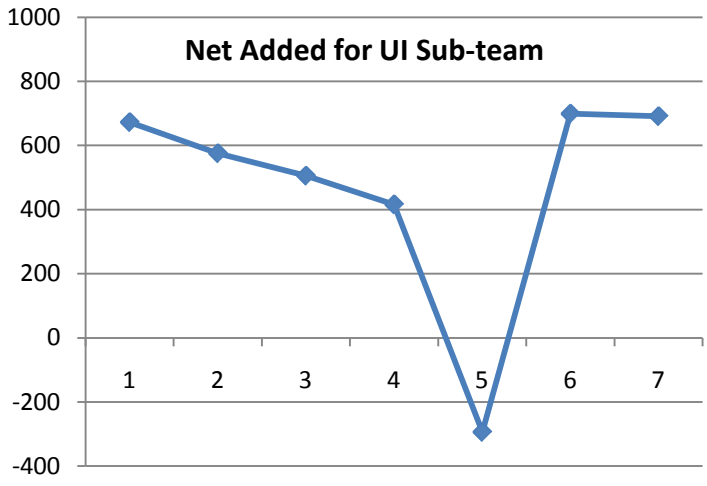


FIGURE 4. NET ADDED LOC FOR UI SUB-TEAM

2.1. WHAT TO TAKE AWAY (AND NOT TAKE AWAY) FROM THIS EXAMPLE

The team in this illustrating example had a hypothesis that velocity correlated with net added LOC. While my dissertation research is ongoing and particularly if I can place my tools in the hands of real developers on real projects, we are likely to uncover potentially predictive

relationships. When this occurs, we will queue it up for further investigation, but that is not the primary focus of this dissertation nor the message I want the reader to take away from the illustrating example just presented. The point of sharing this example (as well as the dissertation as a whole) is not to prove or disprove this correlation (or others like it). Rather, my goal is to give developers the ability to find their own patterns and correlations by enhancing their insight with tooling that allows them to rapidly explore a myriad of possibilities.

The relationships they find may not be broadly applicable but they may be useful for their particular situation especially when combined with the specific knowledge they have. Right now, if a development team has a hunch that one alternative is better than another and the cost of confirming that hunch is too high, they will just go with the hunch. But if I can give them a way to cost effectively do some quick analysis, then maybe they will chose the other alternative… or alternatively be more confident in their hunch. In the illustrating example above, the team might not have assumed that the improvements in velocity were enough to make up for the cost of a refactoring and the outcome for the team might not have been as favorable.

More generally, agility of measurement can enable teams to make better decisions than with current capabilities. This dissertation accepts this statement as a premise. What it will try to show is that we can achieve agility of measurement.

3. PROPOSITION, RESEARCH QUESTIONS, AND PLAN

I have started to create a system that targets the opportunities outlined by the observations presented herein. The system is tentatively named Lumenize and it represents the apparatus that I will use to validate the proposition and research questions that follow.

3.1. PROPOSITION

Easy initial use and rapid evolution are the keys to exploring the myriad of possible measurements, analysis, and visualizations that might provide insight to the current situation. Interactivity to both guide the measurement system design and the direction of the analysis is the key to making maximum advantage of the user's tacit knowledge.

It is possible to define a measurement and visualization specification system for non-trivial software and system engineering analytics that is a significant improvement over hand coded measurement and visualization in terms of: (1) lower effort to produce, (2) more rapidly evolution, and (3) increased interactivity in both design and use. The system would be easy enough to use so that end-users like managers and process engineering staff could design their own interactive systems without programming and flexible enough that a programmer who understood the system could add new analysis and visualization capability to the already significant built-in capability.

3.2. RESEARCH QUESTIONS AND PLANS FOR VALIDATION

To demonstrate the above proposition, I suggest the following research questions and plans to validate them.

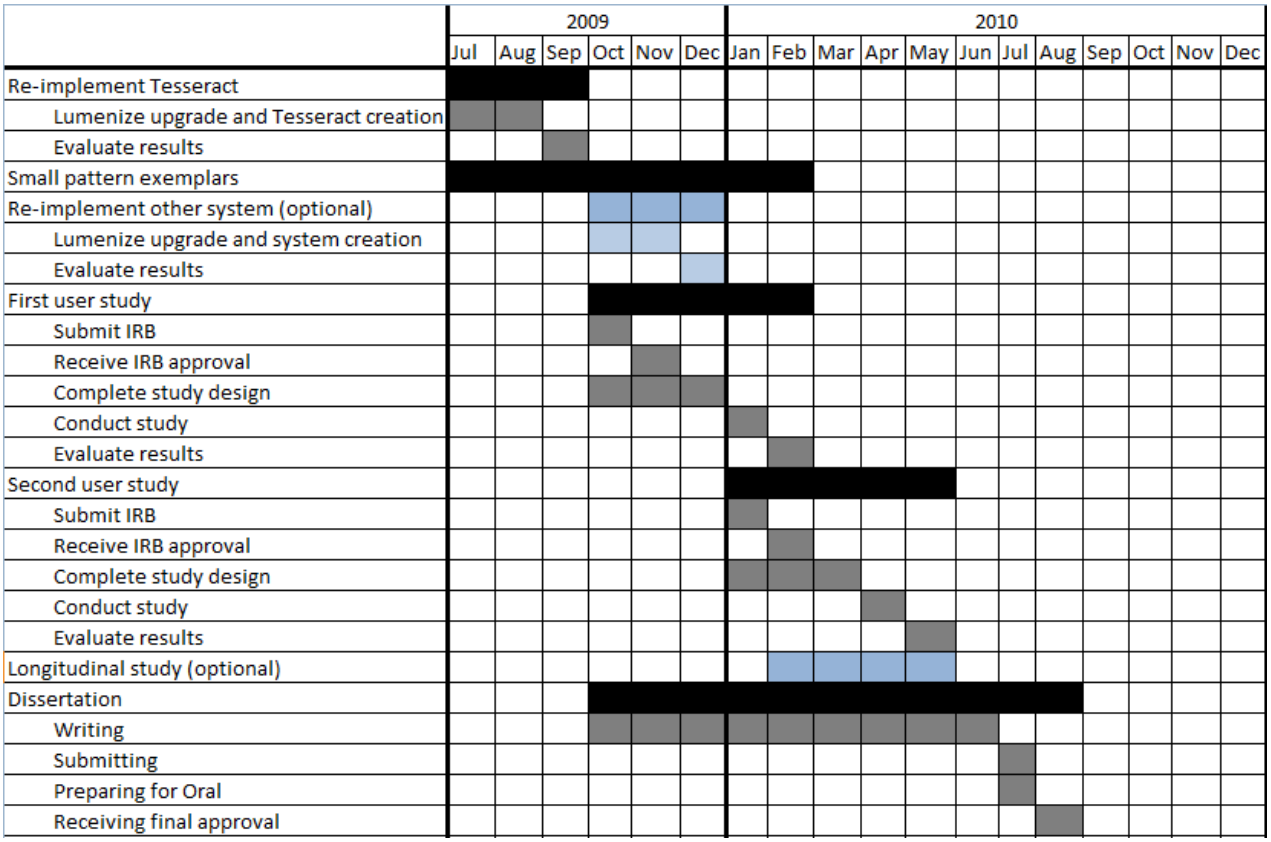
QUESTION	PLAN FOR VALIDATION
Sufficient expressivity. Is it possible to create a system that is sufficiently expressive to enable the creation of a non-trivial and interactive analysis and visualization system?	**Re-implement Tesseract**. Use Lumenize to re-implement a significant analysis and visualization tool (Tesseract, see below) that was originally created by hand-coding. **Small pattern exemplars**. Implement a series of small one-off single-visualization examples to demonstrate the breadth of measurement patterns that Lumenize is capable of expressing. **Re-implement other system (optional)**. Re-implement another large system like the measurement system we developed for the DoD or the Team Software Process measurement information system.
Faster and smaller. Is the use of such a system an improvement in time and size over by-hand production of a similar measurement capability?	Track time for development during the "sufficient expressivity" efforts and measure the final size. Compare these with by-hand equivalents.
Rapid evolution. Does the system enable rapid evolution of a measurement regime?	Change some significant aspect of the Lumenize-based version of Tesseract (and possibly the other validation proposed for the "sufficiently expressive" question) and track how much effort it takes to accomplish the change. Compare this to how much effort it takes to accomplish the same change in the original hand-coded version of Tesseract. A candidate for this "change" is to adapt Tesseract to use IBM's Jazz data repositories. Another candidate is to adapt Tesseract to use a systems engineering data set (as opposed to a software product one).
Sufficiently easy to learn. Is the system sufficiently easy to learn that a new user can create a useful analysis and visualization design with a reasonable amount of training?	**First user study**. Conduct a user study where the subject is first trained on the system and then asked to use it to create a specific analysis/visualization.
Sufficiently easy to evolve. Is the system sufficiently easy for a user to reuse and adapt an existing analysis and visualization design?	As part of the same user study mentioned above, given Lumenize code, the subject will be asked to modify it to accomplish something additional/different. I may give them the Lumenize-based Tesseract code and ask them to evolve it to accomplish some new capability or I may ask them to evolve their own creation, or both. Additional validation (optional): For developer-users, I may

	ask them to accomplish the same change in an equivalent hand-coded measurement and visualization (the original hand-coded Tesseract for instance)
Encourages iteration. Do users get to their final analysis and visualization by iterating over possibilities?	**Second user study**. Conduct a user study (probably separate from above) where the subject is asked to design an analysis that achieves a particular goal. Look at the steps and intermediate forms that they tried before setting on their final product. How many steps did they go through? Was the final result better because they could iterate rapidly?
Encourages curiosity. Once users see the result of an analysis or visualization does that lead to other questions?	In the same user study as above, ask the users at the beginning if the question at hand seems important to answer and if there are any other questions that they think might be useful. Then after they are done, ask them if they can think of any additional questions and identify the ones that come from seeing the output of the analysis.
Validation from real-world use (optional)	**Longitudinal study (optional)**. If the opportunity presents itself, I hope to be able to see how Lumenize could be valuable outside of a laboratory setting. At this time, it is impossible to identify the specific questions and means of validation that I would employ for such a retrospective study but they would likely cover roughly the same ground as the proposed laboratory experiments.

3.3. TIMELINE

My anticipated timeline is shown in Figure 5.

FIGURE 5. TIMELINE



3.4. RISKS AND MITIGATION

3.4.1. LUMENIZE IMPLEMENTATION RISK

The largest potential risk of not being able to come up with an appropriate declarative syntax has already been dispatched as I have already figured out how to say what it needs to say. However, there is still some risk that I will not be able to come up with a way to automatically implement all the possible specifications I have imagined. If that is the case, I will reduce my scope. Even if I have to reduce it to the already implemented parts, that should still be sufficient.

3.4.2. TIME RISK

I anticipate started a new job in the next few months. I should have completed the re-implementation of Tesseract by then and started on the first user study (IRB and design). I should also have been able to create at least one small exemplar by then. However, I do not know how much time I will have after that. It will depend upon the nature of my eventual employment whether I will be able to work on the dissertation as part of my job. If I run short of time to dedicate to this work, my first adjustment will be to drop the optional elements of the plan and if that is not sufficient my timeline will have to stretch out.

3.4.3. RISK THAT USERS CANNOT BE TAUGHT THE SYSTEM

While I think it is easy enough for me to comprehend, I may not be able to teach it to others in the time I anticipate. That might call for extending the training time which would impact the feasibility (cost and time) of the user studies. To mitigate this risk, I will pilot the training well in advance of the studies and make any adjustments before the start of the first study.

3.4.4. RISK FOR LONGITUDINAL STUDY

If I am unable to find a suitable candidate for a longitudinal study I will drop this optional element of my plan.

4. EXPECTED CONTRIBUTIONS

I expect that the contribution of this work will fall into the following categories:

1. Validation that "agility of measurement" (which includes aspects of easy initial use, rapid evolution, and interactivity of both the design and use) is possible and hopefully indications that it is also valuable.
2. The creation of a declarative specification syntax and system for measurement, analysis, and visualization with innovative capabilities including:
    - Seamless transition back and forth from design (specification of analysis transformations and visualization views) to usage (actually conducting analysis and looking at visualizations)
    - Seamless transition back and forth from graphical to textual representation (of the language) "round-tripping"; where the graphical representation makes it easy to get started and the textual representation makes it possible to specify aspects not easily accomplish in a graphical representation (example: formulas)
    - Expresses complex measurements concisely and precisely to enable easy reuse and mashups
    - Maintains traceability back to original entities even through many layers of transformation which enables relationship discovery and exploration
3. Insight into the usefulness of measures and visualizations for socio-technical relationships and dependencies. Tesseract enables visualization of socio-technical congruence. We have also identified other socio-technical analytics that we would like to explore. For instance, we have an idea for a way to visualize cohesion and coupling. Further, we think that we have come up with a proxy for these measures that can be extracted from source code repository history logs without using static analysis. We can use Lumenize to validate these analysis and visualization ideas. The rapid evolution capability of Lumenize should allow us to accomplish this exploration much more rapidly and hopefully achieve a better final result.

4. Transporting and adapting valuable concepts and abstractions from Business Intelligence (BI) with improvements for the domain of software and systems. OLAP analysis is a powerful abstraction with wide reaching potential use. It is the central abstraction used by the powerful analysis and visualization capability of Lumenize. I have already adapted the concept in at least two significant ways that are better suited for our domain. The first is that I can trivially generate network/graph data from OLAP slices. The second is that I allow a delimiter in the data itself to indicate hierarchy (as opposed to it being provided separately as is the case for business-oriented OLAP implementations). By themselves, these adaptations are not earth shattering. But combined with others that are likely to result from this work, they fulfill a critical requirement for their use in our domain.

5. EXISTING APPROACHES

The apparatus for validating the proposition of this dissertation manifests as a declarative syntax for the specification of measures along with the accompanying infrastructure to automatically consume these specifications and generate useful analysis and visualization. I have found a number of efforts to create similar capability.

5.1. OBJECT-ORIENTED METRIC DEFINITIONS

Tang and Chen [54] demonstrate a means to compute object oriented design metrics from UML models. Similarly, El-Wakil et. al. [55] uses XQuery to specify metrics run against XML-encoded UML. Misic et. al. [56] uses Z to define a metamodel to express several metrics using the number of instances of a given type. My work is distinct from this work on the most basic level that this work targets metrics for object-oriented design, while mine is higher level approach to address measurement, analysis, and visualization in general without regard to how the base measures are actually captured.

5.2. MORE GENERIC METAMODELS FOR CODE METRICS

Monperrus et. al. [57] describes a model-driven engineering (MDE) metamodel that is both a domain-specific language (DSL) for calculating generic metrics and is targeted at providing metrics for other DSLs . It is particularly flexible and could be used in a wide range of situations including feasibly process metrics for which my tools are particularly capable. However, like the object-oriented metrics specification systems mentioned above, they are primarily targeting characteristics of the code rather than general process metrics. Also, they do not include support for specifying the accompanying visualization and interactivity. Guerra et. al. [58] enables the expression of generic and DSL metrics similarly to Monperrus et. al. but does so visually but like Monperrus et. al. does not enable the specification of measure visualization and is targeted primarily at code metrics.

5.3. PATTERNS AND DSLS FOR GENERIC SOFTWARE MEASURES

The work by Lindvall et. al. [59] to identify reusable measurement patterns is a useful base for my work. At a high level, it attempts to identify a means where parts of a measurement specification could be reused. My work could be validated against the list of patterns identified by their work. However, their approach is textual and intended for human consumption not automatic implementation. Perhaps closest to my work is the work by Mora et. al. [60] on the Software Measurement Modeling Language (SMML). It closely follows the terminology from PSM - ISO/IEC 15939 and it is a complete DSL, allowing the specification of measurement that could be automatically implemented although it is unclear from reading the publication available that they have taken it that far. The language itself is visual and easily understood. However, in both of these cases, the primary difference is that the goal of their work is to make it easier to conduct the preplanning aspects of measurement within the GQM/PSM paradigm whereas my approach is designed to work in situations where preplanning has not been done and enable ad hoc, just-in-time measurement, analysis and visualization. As such, they stop at the specification and do not demonstrate how the user is to consume or interact with resulting measures. It may be feasible to convert an SMML specification into a Lumenize specifications and enable automatic implementation.

6. CURRENT STATUS OF DEVELOPMENT FOR LUMENIZE

I have tentatively given the name of Lumenize to the system that consumes the declarative syntax for specifying measures, analysis, and visualization. This section provides a brief explanation of the current state of Lumenize syntax to give the reader a better feel for how measures, analysis, and visualization will be specified. From this, the reader should be able to make a preliminary evaluation of the expressiveness of the proposed syntax.

I use a micro-snippet of data from a source code repository commit log shown in Table 6 as a running example to illustrate the features and syntax of Lumenize.

TABLE 6. EXAMPLE SOURCE CODE COMMIT LOG DATA

commit	date	file	del	added
1	2009-01-01	db/node/visitor	0	54
2	2009-01-02	db/node/observer	0	130
2	2009-01-02	ui/button	0	276
3	2009-01-03	db/node/observer	7	10

To extract measures, the first thing a user needs to do is tell Lumenize where to get the data. The underlying raw data in Lumenize is referred to as a BaseModel. "Model" was chosen to align with the Model-View-Controller/Presenter patterns. The syntax for all Lumenize specifications is in the serialization format known as JSON. The specification for our example BaseModel looks like this:

```
{
      commit: {file: "data/commit.xml"},
      commitDetail: {
            file: "data/commit_detail.xml",
            derivedFields: {
                  churn: {evalFunction: "row.added + row.deleted"},
                  net_added: {nativeFunction: "netAdded"}
            }
      },
      file: {file: "data/file.xml"},
      person: {file: "data/person.xml"}
}
```

This specification tells Lumenize to load the data from four XML files. The file for commitDetail is shown in Table 6. The other three files simply contain metadata about their respective entities (commit, file, and person). Note that the data in these files is relational. Each commitDetail entry points to a particular commit entry which in turn points to a particular person entity.

Our example specification includes a couple of derived fields to be added to each row in the commitDetail table. The above code shows two different ways to specify a derived field. See Appendix A for more detail on how to specify row functions in Lumenize.

The next operation in our running example is to create a MatrixAggregator. MatrixAggregator is simply the Lumenize name for 2-dimensional online analytical processing (OLAP) cube. As described in section 1.5, the OLAP abstraction comes from the business intelligence (BI) community. It can be thought of as an advanced pivot table commonly found in spreadsheet applications. It enables the bucketing of data based upon multiple dimensions. Lumenize also supports n-dimensional OLAP cubes but the 2-dimensional form is shown here for the sake of brevity. Here is the declarative syntax:

```
{
	type: "MatrixAggregator",
	inputs: {
		dataProvider: "baseModel.commitDetail"
	},
	rows: {  // referred to as "i" inside MatrixAggregator
		fieldRefString: "_file.name",
		hierarchyDelimeter: "/"
	},
	columns: {  // referred to as "j" inside MatrixAggregator
		fieldRefString: "_commit.when"
	},
	measures: [
		{type: "sum", field: "deleted", name: "del"},
		{type: "sum", field: "added"},  // default name = field value
		{type: "sum", field: "churn"},
		{type: "sum", field: "net_added", name: "net"},
		{type: "count", name: "count"},  // no field necessary
		{type: "contributors"},  // no field or name necessary
		{type: "min", field: "churn", name: "min_churn"},
		{type: "max", field: "churn", name: "max_churn"},
	]

}
```

The above Lumenize code shows all of the standard measures types currently supported by Lumenize. The sum, count, min, and max measure types are fairly obvious. The contributors measure actually a pointer to each entity in the BaseModel that contributed to the cells' totals. This functionality is used to implement the cross-linking features discussed later in the section about Tesseract. Users can also specify their own measures in much the same way that derivedFields are specified.

Just looking at the churn measure, the above specification would result in output like that shown in Table 7. If the user were to run this analysis on the full commit log, she could identify which parts of the code underwent churn and at what time.

Notice how the system was able to understand the hierarchy of code sub-directories and key off of the "/" to group the measures appropriately. You can expand or collapse these groupings at any level. Furthermore, you can generate graphs at any level. The tooling also makes it easy to group the columns into a release/iteration hierarchy but that is not shown in this simple example.

TABLE 7. LUMENIZE MATRIXAGGREGATOR VISUALIZER

	2009-01-01	2009-01-02	2009-01-03
- db/node	54	130	17
visitor	54	0	0
observer	0	130	17
+ ui/button	0	276	0
Total	54	406	17

Also not shown in this example is functionality to do filtering and selection of subsets of the overall data. This capability is present in the current system and more such capability is planned. Current filtering functionality allows the user to specify filter functions in a similar manner as the functions for derived fields are specified.

6.1. CREATING NETWORKS WITH LUMENIZE

Lumenize includes support for viewing measures in the context of an entity relationship graph. This functionality will make a connection between two artifacts by figuring out which files are frequently committed together. This turns out to be a surprisingly good heuristic for determining artifact interdependency [61] that works even in situations where code analysis fails (different programming languages, non-call-graph dependencies, etc.).

We continue to use the same MatrixAggregator as before. However we use a network matrix and visualization. Here is the Lumenize code to aggregate the number of times two files have been committed together:

```
{
      nodes:"file",
      edges:"commit",
      edge_weight:"count",
      node_size:"count"
}
```

TABLE 8. NETWORK MATRIX

	db/node/visitor	db/node/observer	ui/button
db/node/visitor	1	0	0
db/node/observer	0	2	1
ui/button	0	1	1

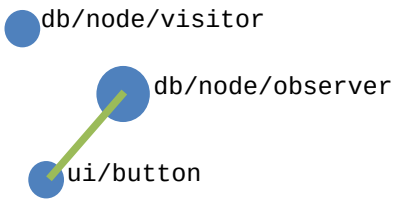


FIGURE 6. NETWORK VISUALIZATION

The above code results in a network matrix shown in Table 8 and the visualization shown in Figure 6.

The diagonal of the tabular representation indicates the number of times a particular file was committed. The other cells represent the number of times two different files were committed together. In our simple example, the only two files ever committed together were "db/node/observer" and "ui/button".

7. TESSERACT: A SOCIO-TECHNICAL BROWSER

Lumenize is being used to re-create a system named Tesseract, about which a paper was presented at ICSE 2009. The original version of Tesseract was built by hand and my role on the team was as primary implementer. This allowed me to understand what facilities were needed to easily construct such a system. Tesseract is being re-built using Lumenize as I add the capability to

address that particular aspect. For the parts that have already been converted, there is an 11:1 reduction in the lines of code necessary to re-create the functionality.

Tesseract analyzes different project archives, such as source code repositories, issue databases, and communication records to determine the interrelationships, which are then graphically displayed via four juxtaposed panes, enabling users to easily explore the data set.

The critical advantage of Tesseract's approach is that it enables the interactive exploration of the different connections among different project entities. Users can change the perspective of their investigation by drilling down on specific artifact(s) or developer(s). As an example, a user might drill-down to only the developers he personally knows to find whether any of his acquaintances have expertise which would help with his current task. Other user actions to facilitate investigations include: 1) highlighting, in yellow, all related entities in the other panes, and 2) hovering over a node to display additional information and measures related to the node.

7.1. TESSERACT USAGE SCENARIO

Figure 7 provides two snapshots of project history for a large open source software project: one relatively early, the other later. We can make the following observations from Figure 7 (top): (1) Stephen Walther is the primary contributor having changed literally every file (every node in the upper left is yellow when we click on Stephen in the upper right); (2) Stephen is central and in contact with most other developers (green lines between Stephen and other developers), but very few developers are communicating among themselves (red lines); (3) the file network is densely connected indicating a high degree of coupling; and (4) this time period shows a continuously increasing list of open issues (stacked area chart on the bottom).

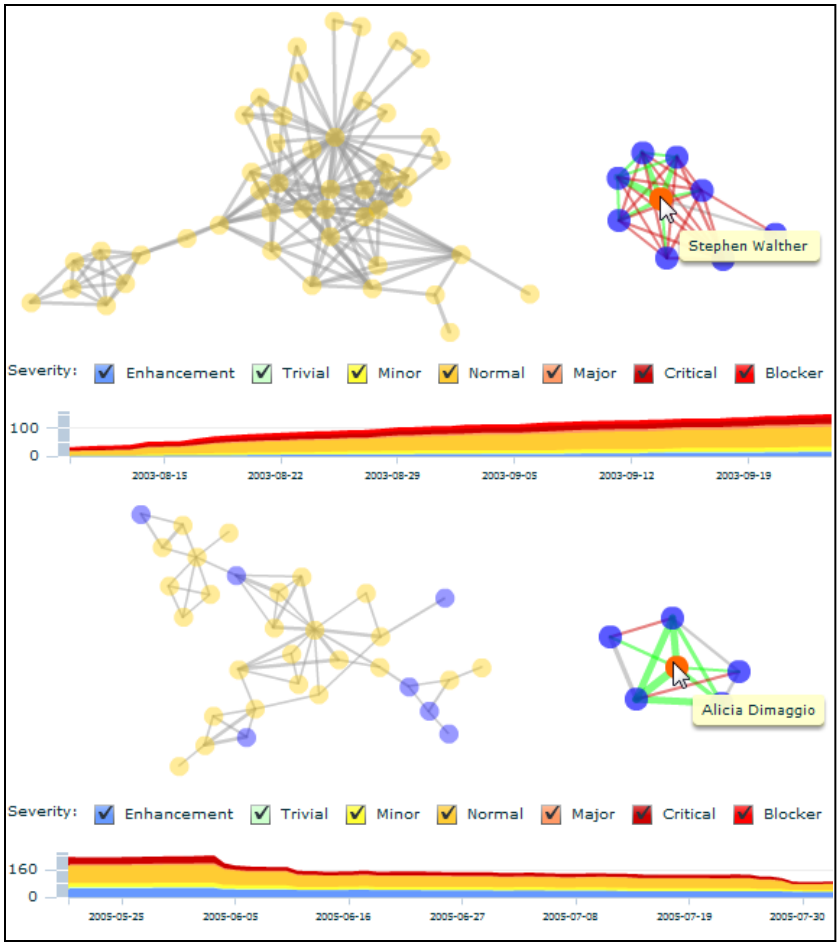


FIGURE 7. LEADERSHIP PATTERNS

When we investigate a later time period, Figure 7 (bottom), under a different leader, Alicia, we see very different patterns in the percentage of files directly edited by Alicia, communication channels, artifact dependency, and the trend for open issues. From this view, it is impossible to tell if there is a causal relationship and what that might be. However, the team doing the work might propose some explanation and they may even be able to use the tool to confirm their hypothesis.

8. CONCLUSION

This work starts with the premise that better understanding leads to better decisions and outcomes. Measurement is a more precise means of achieving understanding than observation, intuition or folk lore but it the bredth of concepts that can be understood with current measurement capability as well as the cost of gathering anything beyond simple measures causes many developers to rely upon these less precise means even when making important decisions. Throughout this proposal, I have discussed ideas on how to drive down the costs and increase the predictive power of measurement for software and systems. I have mentioned interactivity, rapid iteration, using tacit knowledge, consideration of socio-technical relationships, and a number of other ways to alleviate this problem. I have argued the case for each of these potential improvements separately but I believe they can be even more powerful if combined into a systematic approach. For lack of a better phrase I identify this approach as, “more agile measurement”. While argued for herein, this dissertation accepts as a premise that more agility of measurement will result in more use of and better measurement. What it will try to show is that we can achieve this greater agility of measurement.

APPENDIX A. FUNCTIONS ILLUSTRATE A KEY DESIGN THEME OF LUMENIZE; THAT OF "CODE" AS MERELY CONFIGURATION AND WIRING

Similar to how a spreadsheet allows you to put a formula in a particular cell, Lumenize makes extensive use of user defined functions. These allow the user to specify derived fields and filter functions, among other purposes. There are several types of functions in Lumenize:

1. Eval functions
2. Native functions
3. Classes that implement IRowFunction
4. Classes that implement IMeasure

Eval functions. Functions of type "evalFunction" can be specified by simply providing a formula, as in "row.added + row.deleted". This functionality is adapted from RIA1's excellent D.eval library and is the approach that will be used most of the time because these functions can be specified by simply providing the evalFunction in a text field; meaning that there is no need for the user to have to use the Flex compiler. Also, almost any valid ActionScript 3 can be used including control flow, local state, and even internal function definitions. For instance, the user could specify an evalFunction like this:

```
"
      function nonsenseFunction(x:int, y:int):int {
            return (x*x + y) * values.scaling
      }

      var nonsenseVar:int =
          nonsenseFunction(row.deleted, row.added)
      var returnValue:String

      if (nonsenseVar > 100) {
            returnValue = "big nonsense"
      } else {
            returnValue = "little nonsense"
      }

      return returnValue
"
```

The user defined nonsenseFunction is called from within the main body of the actual evalFunction.

As shown in the examples above, the current row is available to the context of the code. Additional context is available in the form of the "values" scratchpad area. These "values" are typically bound to UI controls. This allows the results of a function to depend upon user input. For instance, the user could alter the output of the function above by changing the values.scaling value at runtime.

Native functions. Alternatively, you can specify a native ActionScript 3 function (nativeFunction) to perform the calculation. The function must be defined with parameters that allow Lumenize to pass in the row and values contexts. Users may choose this option for performance reasons but I anticipate those to be rarely needed.

IRowFunction. EvalFunctions specified by an end-user and nativeFunctions specified by a developer-user are wrapped by a class that implements the IRowFunction interface. The developer-user could create their own class that implements this interface. This might be a good alternative if the function needs to maintain state between function calls, although in all cases that I've come across so far, the values scratchpad area has been sufficient for maintaining state.

Design theme. The only reason I even mention it, is to highlight a design theme of Lumenize's. A common design theme of the Lumenize system is that functionality provided to the end-user is typically just a thin wrapper over highly patternized internal structure of Lumenize. The reason Lumenize can so easily implement its declarative specification language is that the code for each part of a Lumenize measurement specification is passed into a class that implements that functionality using the factory pattern. The Lumenize code is simply a big parameter for the Lumenize core and its factories. Lumenize code is just wiring and configuration settings.

Measure functions. Lumenize also allows the developer-user to specify their own measure functions for MatrixAggregator and OLAPAggregator cells. Since Lumenize provides a rich set of built-in measure functions, I have not added the ability for end-users to add their own measure functions but using the pattern established for row functions and the "code as configuration and wiring" design theme, this would be trivial.

APPENDIX B. FLASH, FLEX, AND ACTIONSCRIPT

Lumenize is written using Adobe's Flex framework which runs in the Flash virtual machine.

The programming language for Flex/Flash is called ActionScript 3 (AS3). AS3 is most easily described as Java or C# with optional dynamic typing. The inheritance model used by almost all AS3 code is class-based and the vast majority of AS3 code is also statically typed. AS3 includes support for classes, interfaces, error handling, and the common scope modifiers that developers expect (public, private, protected, etc.). The primary development environment for Flex is actually built upon Eclipse, the most popular Java IDE. AS3 source code is compiled to byte code and the Flash virtual machine has advanced functionality like just-in-time compilation to native code although it is clearly behind Java and .NET from the perspective of VM and compiler optimizations… but catching up.

While it is missing nothing of importance from Java or C#, AS3 has some advantages over these. It allows for dynamic typing and enables powerful meta-programming capability that Lumenize depends upon. The event model is built deeply into the platform as is the UI layer enabling rich user interface design that looks and behaves exactly the same on every platform. It is easily run in a web browser but the Adobe Integrated Runtime (AIR) also allows Flash apps to be run on the desktop with full access to the local machine. The measurement system designer for Lumenize utilizes AIR so that users can import data and save their designs to their local machine.

APPENDIX C. REFERENCES

[1] P. Tunbridge and I.O.E. Engineers, *Lord Kelvin, his influence on electrical measurements and units*, IET, 1992.

[2] V. Katz, *The mathematics of Egypt, Mesopotamia, China, India, and Islam : a sourcebook*, Princeton: Princeton University Press, 2007.

[3] W.E. Deming, *Some theory of sampling*, Courier Dover Publications, 1966.

[4] W.E. Deming, *The new economics: For industry, government, education*, MIT press, 2000.

[5] P.B. Crosby, *Quality is free: The art of making quality certain*, New American Library New York, 1979.

[6] G. Tennant, *SIX SIGMA: SPC and TQM in Manufacturing and Services*, Gower Publishing, Ltd., 2001.

[7] P.S. Pande and R.P.;.C. Neuman, *The Six Sigma Way: How GE, Motorola, and Other Top Companies are Honing Their Performance*, New York: McGraw-Hill Professional, 2001.

[8] M.C. Paulk and C.M.U.S.E. Institute, *The capability maturity model*, Addison-Wesley Pub. Co., 1995.

[9] M.B. Chrissis, M. Konrad, and S. Shrum, *CMMI*, Addison-Wesley, 2003.

[10] C.B. Tayntor, *Six Sigma software development*, CRC Press, 2002.

[11] W.S. Humphrey, *Introduction to the team software process(sm)*, Addison-Wesley, 1999.

[12] B. Boehm, *Software cost estimation with Cocomo II*, Upper Saddle River NJ: Prentice Hall, 2000.

[13] B. Boehm, *Software engineering economics*, Englewood Cliffs N.J.: Prentice-Hall, 1981.

[14] N. Davis and J. Mullaney, "The Team Software Process (TSP) in Practice: A Summary of Recent Results," *Software Engineering Institute, CMU/SEI-2003-TR-014*, 2003.

[15] V.R. Basili, G. Caldiera, and H.D. Rombach, "The goal question metric approach," *Encyclopedia of Software Engineering*, vol. 1, 1994, pp. 528-532.

[16] R. van Solingen and E. Berghout, *The Goal/Question/Metric Method: A practical guide for quality improvement of software development*, McGraw-Hill Cambridge, UK, 1999.

[17] V.R. Basili, "Software modeling and measurement: the Goal/Question/Metric paradigm," 1992.

[18] J. McGarry, D. Card, C. Jones, B. Layman, E. Clark, J. Dean, and F. Hall, *Practical software measurement: objective information for decision makers*, Addison-Wesley, 2002.

[19] I. ISO, "IEC 15939: 2002 Software Engineering: Software Measurement Process," *International Organization for Standardization/International Electrotechnical Commission., Geneva*, 2002.

[20] "MSR 2009: 6th IEEE Working Conference on Mining Software Repositories."

[21] P.M. Johnson, "Project Hackystat: Accelerating adoption of empirically guided software development through non-disruptive, developer-centric, in-process data collection and analysis," *Department of Information and Computer Sciences, University of Hawaii*, 2001.

[22] P.M. Johnson, *Hackystat system*.

[23] S.H. Li-Te Cheng, S. Ross, and J. Patterson, "Jazzing up Eclipse with collaborative tools," *Proceedings of the 2003 OOPSLA workshop on eclipse technology eXchange*, 2003, pp. 45-49.

[24] K. Bauer, "Experience with Collaboration Systems in Undergraduate Software Engineering Courses."

[25] T.H.D. Nguyen, A. Schröter, and D. Damian, "Mining Jazz: An Experience Report."

[26] P. Smacchia, "Code Query Language 1.0 specification."

[27] S. Paul and A. Prakash, "Supporting queries on source code: A formal framework," *Ann Arbor*, vol. 1001, pp. 48109-2122.

[28] M. Cataldo, P.A. Wagstrom, J.D. Herbsleb, and K.M. Carley, "Identification of coordination requirements: implications for the Design of collaboration and awareness tools," *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, ACM New York, NY, USA, 2006, pp. 353-362.

[29] S.A. Bohner, R.S. Arnold, and Arnold, *Software change impact analysis*, Citeseer, 1996.

[30] R.S. Arnold, "The Year 2000 Problem: Impact, Strategies, and Tools," *Software Change Impact Analysis*, 1996, p. 44.

[31] M. Kersten and G.C. Murphy, "Using task context to improve programmer productivity."

[32] D. Cubranic, G.C. Murphy, J. Singer, and K.S. Booth, "Hipikat: A project memory for software development," *IEEE Transactions on Software Engineering*, vol. 31, 2005, pp. 446-465.

[33] R. DeLine, A. Khella, M. Czerwinski, and G. Robertson, "Towards understanding programs through wear-based filtering," *Proceedings of the 2005 ACM symposium on Software visualization*, ACM New York, NY, USA, 2005, pp. 183-192.

[34] C. Bird, D. Pattison, R. D'Souza, V. Filkov, and P. Devanbu, "Chapels in the Bazaar? Latent Social Structure in OSS," *16th ACM SigSoft International Symposium on the Foundations of Software Engineering, Atlanta, GA*, 2008.

[35] A. Meneely, L. Williams, W. Snipes, and J. Osborne, "Predicting failures with developer networks and social network analysis," *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, ACM New York, NY, USA, 2008, pp. 13-23.

[36] A. Sarma, L. Maccherone, P. Wagstrom, and J. Herbsleb, "Tesseract: Interactive Visual Exploration of Socio-Technical Relationships in Software Development."

[37] J.D. Herbsleb and R.E. Grinter, "Splitting the organization and integrating the code: Conway's law revisited," *Proceedings of the 21st international conference on Software engineering*, IEEE Computer Society Press Los Alamitos, CA, USA, 1999, pp. 85-95.

[38] D.L. Parnas, "On the criteria to be used in decomposing systems into modules," 1972.

[39] R.E. Grinter, "Recomposition: putting it all back together again," *Proceedings of the 1998 ACM conference on Computer supported cooperative work*, ACM New York, NY, USA, 1998, pp. 393-402.

[40] "Adoption of Agile Methods," *Methods and Tools*, Feb. 2008.

[41] "Adoption of the CMMI," *Methods and Tools*, Jan. 2009.

[42] B.W. Boehm and R. Turner, *Balancing agility and discipline: A guide for the perplexed*, Addison-Wesley Professional, 2003.

[43] B. Boehm and R. Turner, "Observations on balancing discipline and agility," *Agile Development Conference*, 2003.

[44] N.J. Nilsson, *Principles of artificial intelligence*, Springer, 1982.

[45] M. Minsky, "Steps toward artificial intelligence," *Computers and thought*, vol. 406450, 1963.

[46] S. Chaudhuri and U. Dayal, "An overview of data warehousing and OLAP technology," *ACM Sigmod Record*, vol. 26, 1997, pp. 65-74.

[47] E.F. Codd, S.B. Codd, and C.T. Salley, *Providing OLAP (on-line analytical processing) to user-analysts: An IT mandate*, Codd amp; Associates, 1993.

[48] E. Thomsen, *OLAP solutions: Building multidimensional information systems*, Wiley, 2002.

[49] A. Berson and S.J. Smith, *Data warehousing, data mining, and OLAP*, McGraw-Hill, Inc. New York, NY, USA, 1997.

[50] D. Tuma, "The Software Process Dashboard Initiative."

[51] N. Nagappan and T. Ball, "Explaining failures using software dependences and churn metrics," *Microsoft Research, Redmond, WA*, 2006.

[52] B. Du Bois and T. Mens, "Describing the impact of refactoring on internal program quality," *ELISA workshop*, p. 37.

[53] M. Pizka, "Straightening spaghetti-code with refactoring," *Proc. of the Int. Conf. on Software Engineering Research and Practice-SERP*, pp. 846-852.

[54] M.H. Tang and M.H. Chen, "Measuring OO design metrics from UML," *Lecture Notes in Computer Science*, 2002, pp. 368-382.

[55] M. El-Wakil, A. El-Bastawisi, M. Riad, and A. Fahmy, "A novel approach to formalize object-oriented design metrics," *Evaluation and Assessment in Software Engineering*, 2005, pp. 11-12.

[56] V.B. Misic and S. Moser, "From formal metamodels to metrics: An object-oriented approach," *Technology of Object-Oriented Languages, 1997. TOOLS 24. Proceedings*, 1997, pp. 330-339.

[57] M. Monperrus, J.M. Jezequel, J. Champeau, and B. Hoeltzener, "A model-driven measurement approach," *Proceedings of the ACM/IEEE 11th International Conference on Model Driven Engineering Languages and Systems (MODELS'2008)*, Springer, 2008.

[58] E. Guerra, J. de Lara, and P. Díaz, "Visual specification of measurements and redesigns for domain specific visual languages," *Journal of Visual Languages and Computing*, vol. 19, 2008, pp. 399-425.

[59] M. Lindvall, P. Donzelli, S. Asgari, and V. Basili, "Towards reusable measurement patterns," *11th IEEE International Symposium Software Metrics, 2005*, 2005, pp. 21-21.

[60] B. Mora, F. García, F. Ruiz, and M. Piattini, "SMML: Software Measurement Modeling Language," *Department of Computer Science. University of Castilla-La Mancha Technical Report UCLMTSI-003*, 2008.

[61] M. Cataldo and J.D. Herbsleb, "Communication networks in geographically distributed software development," *Proceedings of the ACM 2008 conference on Computer supported cooperative work*, San Diego, CA, USA: ACM, 2008, pp. 579-588.