Towards a DSL for Agile Measurement and Visualization Patterns

Larry Maccherone
Institute for Software Research - Carnegie Mellon University
Larry@Maccherone.com

Abstract

There is little guidance available (beyond velocity and burn down) on how to do measurement for agile projects. Principle-based advice often sounds like it came from pre-agile thinking by calling for predetermined questions and indicators. Flexible analysis tools like spreadsheets lack necessary support for the domain. My research starts with the identification of a set of reusable measurement and visualization patterns for software processes. From that, I have created tooling that allows software development teams to easily define their own measurement systems and conduct ad-hoc inquiry. It assumes that given the right tools, the folks doing the work are better able to interpret the data than a predetermined indicator.

1. Introduction

Agile practices are predicated on the idea that trying to apply someone else's process template to your situation is rarely ideal and often counterproductive. Rather, the agile approach is principled-based, requiring you to adapt as you learn and your situation changes. For this reason, you don't see explicit guidance on exactly what measures to use in your agile project, with the notable exception of guidance on how to calculate velocity and generate burn down charts. Any measurement needs beyond this are typically satisfied with a spreadsheet.

So on the surface, it appears that the approach matches the agile philosophy – start with principles and use a flexible tool to do what you think is best. However, there are several problems with this:

1. **Predetermined questions and indicators.** Much of the principles and guidance published under the auspices of "Agile Measurement" advises you to predetermine questions and indicators for your measurements ([1), (2). This runs contrary to another agile principle which says to maximize visibility and let the team decide how to interpret the information.
2. **Measurement and visualization patterns.** While the available guidance goes too far in requiring teams to design measures with predetermined indicators, it does not go far enough in providing useful assistance in the form of reusable measurement and visualization patterns.
3. **Spreadsheets inadequate.** Spreadsheets and other flexible analytical tools lack features specifically targeted at the unique needs of software process measurement. For instance, they may support the ability to query sales figures from your SAP installation but they lack the ability to easily consume data passively accumulating in your source code repository.

My research specifically targets these three shortcomings. The work started by first identifying a list of patterns commonly found in software process measurement. A domain specific language (DSL) is now being evolved to realize those patterns. This DSL takes into account the constructs typically found in software development projects (source code repositories, issue databases, code hierarchies, releases, iterations, artifact dependencies, teams, coordination needs, etc.). Using these patterns and the DSL, teams can easily define their own measurement systems. The approach works well when you happen to know the questions or trouble indicators up front, but it works equally well in an ad-hoc fashion to enable problem identification, trend detection, and decision making even in the absence of predetermined indicators.

2. Illustrating example

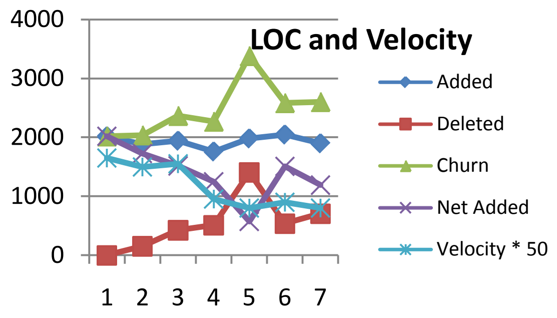
Let us say your team has just completed its seventh iteration. You notice that your velocity has slowed down considerably for the last several iterations and if it does not pick back up, you will not be able to deliver the desired amount of functionality in the time remaining. You start to brainstorm about how to bring the velocity up. Someone suggests that the problem is due to accumulated technical debt. The resulting nods and groans indicate consensus.

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

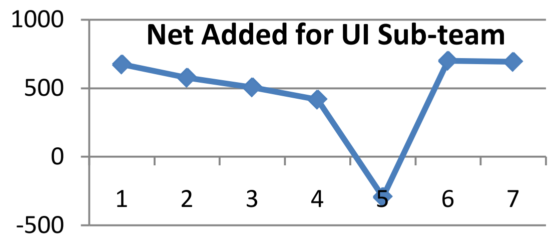
You remember that the two folks, Mary and John, on the team who are primarily working on the UI layer actually did a refactoring of their code in iteration five.

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.

Mary and John indicate that they enjoy working with the code more since the refactoring. Great, but how can you tell if that decision has a net positive RIO before the first release date? You decide to explore and you query the source code repository and generate the following graph.



From the graph, you notice that your velocity generally tracks the net added LOC (R^2=0.75) which is calculated by subtracting the lines deleted from the lines added. You also notice the spike in iteration five for the deleted and churn measures. That was the iteration where the UI sub-team did their refactoring. Next, you decide to see if their net added LOC trajectory was improved after their refactoring. My tools allow you to slice the data based upon a hierarchy delimiter in a particular field. In this case, you know that all the UI code is contained under the subdirectory of "ui/*". You can easily select that sub-directory and chart it.



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 your best judgment based upon the information you have), you project that you will be better off if you spend the next iteration paying down some of your technical debt.

This approach to measurement allows you to explore the data, and put it in context with information you know (like the refactoring done by the UI team). It allows you to notice patterns and then helps you 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.

3. The DSL

You create measures and visualizations with a declarative JSON-based DSL. Consider the following data from a source code repository commit log:

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 operation is to create an online analytical processing (OLAP) cube. The term comes from the business intelligence (BI) community. It is essentially the same idea as a pivot table in a spreadsheet. It enables the bucketing of data based upon multiple dimensions. Here is the DSL:

```
// OLAP Cube definition
{
  calculated_fields: [
    {name:"churn", expression:"del + added"},
    {name:"net", expression:"added – del"}
  ],
  dimensions: [
    {field:"commit"},
    {field:"date"},
    {field:"file",
      hierarchy_type:"delimited",
      delimiter:"/"
    }
  ],
  measures: [
    {field:"del", type:"SUM"},
    {field:"added", type:"SUM"},
    {field:"churn", type:"SUM"},
    {field:"net", type:"SUM"},
    {field:"file", type:"COUNT", name:"count"}
  ]
}
```

The DSL to summarize the churn measure is:

```
// OLAP Pivot Table View definition
{row:"file",
  column:"date",
  measures:["churn"]
}
```

This would result in a pivot table for churn as follows:

	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

If you were to run this analysis on the full commit log, you could identify which parts of the code underwent churn and at what time. The tooling also makes it easy to group the columns into a release/iteration hierarchy but that is not shown in this simple example.

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.

3.1. Creating networks with the DSL

The DSL 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 file are frequently committed together. This turns out to be a surprisingly good heuristic for determining artifact interdependency (3) that works even in situations where code analysis fails (different programming languages, non-call-graph dependencies, etc.).

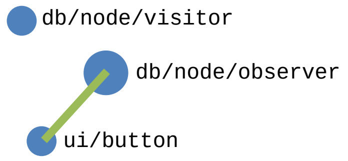
We use the same OLAP cube as the previous example; however we use a network view. Here is the DSL to aggregate the number of times two files have been committed together:

```
// OLAP Network View definition
{nodes:"file",
  edges:"commit",
  edge_weight:"count",
  node_size:"count"
}
```

The above code results in a matrix as follows:

	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

The diagonal represents the number of times a particular file was committed. The other cells represent the number of times two different files were committed together. In this case, the only two files ever committed together were "db/node/observer" and "ui/button". The visualization for the above data looks as follows:



4. Tesseract: a larger system that can be built with the DSL and tools

The DSL and tooling are being used to re-create a system named Tesseract, about which a paper has been accepted for publication 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 from my DSL. Tesseract is being re-built using my DSL and tooling 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.

4.1. Tesseract usage scenario

Figure 1 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 1 (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).

When we investigate a later time period, Figure 1 (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.

5. Advantages over using a spreadsheet or BI tool

Direct use of a spreadsheet or BI tool to accomplish software measurement goals can be frustrating. These tools were designed with the business decision maker in mind and include features that are targeted at that market. We need similar domain specific features for software measurements in order to see widespread use in our field. Existing spreadsheet and BI tools lack direct support for:

- Querying source code repositories and issue databases
- Organizing time hierarchies by release/iteration instead of year/quarter/month
- Building hierarchies from delimiters like those indicated by “/”, “\”, or “.” in artifact URIs
- Placing measures on an artifact dependency or team member relationship map as opposed to a geographical one.

6. Plan for dissertation

To validate my work, I plan to do the following:

- Finish the conversion of Tesseract to using the new DSL and measure the final reduction in lines of code (currently 11:1).
- Reproduce one other system and measure the time it takes to accomplish the conversion as well as the reduction in lines of code.
- Conduct an end user study to determine the difficulty of training the DSL and other usability issues.
- Conduct a longitudinal study where and Agile team uses the tools.

Desired outcomes of the Ph. D. symposium at Agile2009 include:

- Feedback on the overall goals and approach

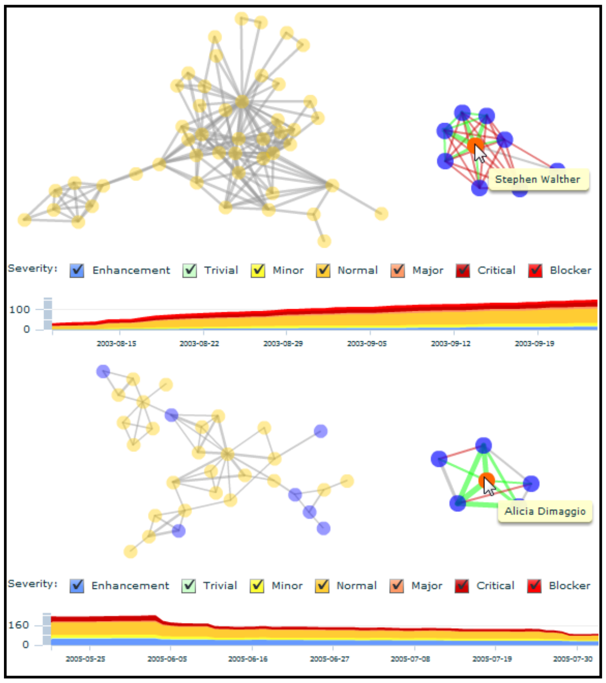


Figure 1 - Leadership patterns

- Feedback on adequacy of validation planned
- Possible volunteers for studies.

7. Advisors

Bill Scherlis and Watts Humphrey

8. References

1. **Gilb, Tom and Brodie, Lindsey.** What's Wrong With Agile Methods: Some Principles and Values to Encourage Quantification. *Methods and Tools.* Summer 2007.
2. *Appropriate Agile Measurement: Using Metrics and Diagnostics to Deliver Business Value.* **Hartmann, Deborah and Dymond, Robin.** 2006. Agile Conference.
3. *Communication Networks in Geographically Distributed Software Development.* **Cataldo, Marcelo and Herbsleb, Jim.** 2008. CSCW. pp. 579-588.