

# *Appropriate Agile Measurement: Using Metrics and Diagnostics to Deliver Business Value*

Deborah Hartmann  
Agile Process Coach  
[deborah@hartmann.net](mailto:deborah@hartmann.net)

Robin Dymond  
Agile Management Consultant  
[robin.dymond@gmail.com](mailto:robin.dymond@gmail.com)

## **Abstract**

*Agile Software Development continually measures both our product and the process used to create it, to allow improvement. With increased popularity, more risk-averse groups are being drawn to Agile, bringing with them modes of evaluation incompatible with Agile values and principles. These outmoded metrics drive dysfunctional behaviors which threaten the integrity of an emerging Agile culture. This paper collects some of the current thinking on appropriate Agile metrics, and proposes simple tools which organizations or teams may use to develop more congruent ways to measure Agile work.*

## **1. Introduction**

Agile Software Development uses an empirical model to make complex work manageable. Ongoing measurement is inherent in this approach, whether anecdotal or formalized. Early Agile projects were led by "innovators", people comfortable with change and risk-taking, who started small and navigated this unknown territory intuitively. Some of these have generously written on the subject, now emboldening more risk-averse "early adopters" and "early majority" [11] to also try Agile on for size. The innovators themselves have moved on to uncharted territories like enterprise implementations and regulated industries, areas also involving more risk-averse participants, who naturally ask: How do we really know this is working?

Unfortunately, old habits die hard, and too often the means used to monitor Agile implementations are out of step with the new paradigm being taught, particularly the metrics of plan-driven PMOs. Having witnessed that measurement drives behavior, the authors are concerned that these improper metrics, in addition to wasting resources, also skew team behavior in counter-productive ways and undermine the effort toward culture change inherent in Agile work. To make

matters worse, there is confusion between two different exercises: measuring to deliver value (the objective of Agile work) and data-gathering to document or justify the Agile approach in the wider world of methodologies.

This paper collects existing information to outline a more congruent approach to measurement for Agile delivery of customer value. It proposes a distinction between long-term, overarching organizational "metrics" and the more temporary, contextual "diagnostics" used to guide local process improvements. The authors additionally propose some simple tools for designing sound metrics for a given project.

## **2. Measure Wisely: Apply Heuristics**

*"Tell me how you will measure me  
and I will tell you how I will  
behave."*

- Goldratt [5]

How can a team measure wisely? This compilation of heuristics from various sources suggests some tests which may be applied when designing measurements for the Agile process.

### **A Good Agile Metric or Diagnostic ...**

1 **Affirms and reinforces Lean and Agile principles.** Supports the customer-intimate and value focused traits that reinforce Agile principles. This requires that people who understand Agile participate in metrics design. The truism "you get what you measure" reminds us that counterproductive behaviors may ensue if you reinforce the wrong things (ex: overtime, % utilization, paperwork).

2 **Measures outcome, not output.** In an Agile environment where simplicity or "maximizing the

amount of work not done" is promoted, the most spectacular outcome might be achieved by reducing planned output while maximizing delivered value. Outcomes are measured in terms of delivered Customer value.

3 **Follows trends, not numbers.** Measure "one level up" [7] to ensure you measure aggregated information, not sub-optimized parts of a whole. (Ex: the financial community has a mature sense of typical industry "shapes" or trends). Aggregate above the individual team level for upper management use. To promote process health, do not track at levels more granular than "a team", and "an iteration".

4 **Belongs to a small set of metrics and diagnostics.** A "just enough" metrics approach is recommended: too much information can obscure important trends. Aggregate upward to roll interrelated or competing factors into a single big picture (see previous item).

5 **Is easy to collect.** For team-level diagnostics the ideal is "one button" automation - where data is drawn from operational tools (i.e. the Product Backlog, acceptance test tools, code analyzers). For management use, avoid rework (ex: PowerPoint) and manipulation of lower level data, aggregation is preferable.

6 **Reveals, rather than conceals, its context and significant variables.** Should be visibly accompanied by notes on significant influencing factors, to discourage false assumptions and facilitate improvement.

7 **Provides fuel for meaningful conversation.** Face-to-face conversation is a very useful tool for process improvement. A measurement isolated from its context loses its meaning. Note: It's a good sign when people talk about what they've *learned* by using a metric or diagnostic.

8 **Provides feedback on a frequent and regular basis.** To amplify learning and accelerate process improvement, metrics should preferably be available at each iteration retrospective, and at key periodic management meetings.

9 **May measure Value (Product) or Process.** Depending on where problems lie, diagnostics may measure anything suspected of inhibiting effectiveness. Consider the appropriate audience for each metric, and document its context/assumptions to encourage proper use of its content. And remember: you get what you measure!

10 **Encourages "good-enough" quality.** The definition of what's "good enough" in a given context must come from that context's Business Customer or their proxy, not the developers.

### 3. One Key Metric Drives the Organization

Historically, software development has often been relegated to cost-center status. The change of focus from cost to value is inherent in the Agile paradigm shift. The book *Lean Software Development* [8] (which adapts the concepts of Lean Manufacturing for use in the software industry) provides useful tools for continuous improvement within the Agile inspect-and-adapt cycle. The first tenet of *Lean Thinking* [9] is: "Define value in the eyes of the customer". While there are significant differences between manufacturing and the more research-oriented software world, this principle is equally applicable in both domains. Agile principles such as "our highest priority is to *satisfy the customer*" and "deliver *working* software frequently" directly reflect this same objective (emphasis added).

This raises the question: who is the Customer? Author Jim Collins [3] suggests that, at an organizational level, this should be the owner or stockholder, or perhaps the end user. The second question then becomes: "what does Value look like?" In his book *Good to Great* [3], Collins discusses the attributes and behavior that are common and unique to companies that have gone from a long history of mediocre results to a long run of great results. One identified aspect is referred to as the "Hedgehog Principle". In this principle, three questions are answered: "what can we be the best in the world at?", "what can we be passionate about?", "what is my one economic driver?" This economic driver then serves as the one key metric by which all aspects of the organization can be measured. For example, at Walgreens, their driver is profit per customer visit. Other large institutions have chosen slightly different metrics. But Collins' research shows that all good-to-great companies have chosen a single economic driver metric that guides all decision making. [10]

Ideally, this key metric is set by executive management. Sometimes, while an organization matures in its implementation of Agile, it may be constrained to measure more locally – in the software arena this could be at the level of team performance or IT delivery. However, this is inadequate and potentially illusory. Software is simply inventory until its value is realized, which requires participation of non-developers: Research, Marketing, Sales, etc. Measuring value in this more holistic way encourages collaboration within the enterprise to create real value – products which not only "work" but are, for example, usable, marketable and which sell.

## 4. Measure What You Deliver: Value as the Key Metric

Agile methods encourage businesses to be accountable for the value produced by software development efforts. The key metrics we use should allow us to monitor this accountability. Metrics should help validate businesses that make smart software investments and teams that deliver business value quickly.

One benefit of value-driven work is this: a project which delivers high-value features to a customer early in the project may become self funding during the course of the project. “Business Value Delivered” is what teams should be optimizing for, and what the organization should be tracking as a key metric.

Business value is best determined by business stakeholders and developers together, at the level of specific project components - features or groups of features that can be given a specific cost and customer value.

### 5.1 Calculating Business Value Delivered

Financial measurements of interest to the business include the amount of capital required and the return on investment (ROI). Both of these measures can be captured with Value. Value is defined as software put into production that can return an investment over time. The more rapidly high-value software can be rolled out, the quicker value is realized.

There are a number of ways to measure value. These include Net Present Value (NPV), Internal Rate of Return (IRR), and Return on Investment (ROI). Each method has its benefits and drawbacks. We recommend that the business determine its preferred method for valuing its investment using a generally accepted financial model.

Developers cannot determine the value of software in isolation, business involvement is essential. To determine the value of features or groups of features, accurate numbers may seem elusive, so we recommend that an iterative approach be taken. Take the business case for the project, and divide overall business value among the features being developed. As with agile requirements, it is better to get something done quickly and leave refinements for later. We recommend a similar approach to the financial model. Once a first release is in production and real financial data is available the financial model can and should be refined.

In addition, Poppendieck [8] suggests assignment of an accountant to every development team to work out economic models for the products they deliver.

Anderson and Schragenheim [1] have elaborated accounting models for valuation of both software and services, and further detailed information on determining value can be found in *Agile Estimating and Planning* [2], chapter 10: Financial Prioritization. Note that this is not simply about counting *cost*, an outdated model, but rather *value*.

### 5.2 The Compounding Return of Value

With an agile project that delivers value early and often, the return on investment begins as soon as the first software features are released into production. Value can be accounted for in two ways: at the close of the iteration when working software is completed and demonstrated, or at the time of release to production, which may mean the accumulation of a number of iterations of delivered features. We recommend accounting for value at the time of release, although “potential value” delivered with each iteration is an adequate measure for the delivery team that cannot release to production due to external constraints (i.e. part of a large multi-project program). In accounting for Value, the business needs to determine the value of specific features or groups of features. For example, if four planned releases of features account for 60%, 25%, 10%, and 5% of the business value, then this should be reflected in the accounting. The team should then ask: is it possible to release features that deliver 30% of the business value at an earlier time?

## 6. Distinguish Diagnostics from Metrics

Once an organizational “key metric” is set, teams will, in addition, need to determine their own local measurements in support of achieving this goal. We propose that these supporting metrics be called *diagnostics* because they are used to *diagnose & improve* the processes that produce business value. They exist and are valid in the context of a particular process and its inherent constraints (for example: a software development team). It is recommended that diagnostics be carefully designed, and applied only for a fixed length of time or the conditions of discontinuing its use are known when it is applied. This avoids ongoing measurement for measurement’s sake, and retains focus on the key metric, which should drive the creation and use of different diagnostics at different times, as needed.

Note that not everything that can be measured should be! The heuristics laid out in section 2 may help teams choose their diagnostics wisely. In addition, existing reporting and metrics should be evaluated and removed if not sufficiently contributing to optimizing the key metric, or if they meet too few of the heuristic criteria laid out in section 2.

An example of a contextual diagnostic: a team decides to take measures to improve the “flow” of their work by reducing interruptions. To provide empirical evidence they will count “interruptions per day”, and review this diagnostic during retrospectives for three iterations, at which point the team aims to have resolved the problem. Clearly, the effort to track this metric becomes wasteful in itself once the problem is regulated (or if the team decides it must be lived with). Another example: “obstacles escaping iterations” could be used to motivate management to address stubborn obstacles hindering creation of business value. However, the longer such a metric is used, the more likely it is to be unconsciously “gamed”. In the spirit of empirical process, it should be used for a defined period, at which time an evaluation should determine next steps, if any are needed.

## 7. Metric / Diagnostic Evaluation Checklist

Once a choice is made to measure some aspect of a team’s product or process, it may be helpful to work through this checklist to be clear on its intention, use and potential for abuse.

- Name:** this should be well chosen to avoid ambiguity, confusion, oversimplification.
- Question:** it should answer a specific, clear question for a particular role or group. If there are multiple questions, design other metrics.
- Basis of Measurement:** clearly state what is being measured, including units. Labeling of graph axes must be clear rather than brief.
- Assumptions:** should be identified to ensure clear understanding of data represented.
- Level and Usage:** indicate intended usages at various levels of the organization. Indicate limits on usage, if any.
- Expected Trend:** the designers of the metric should have some idea of what they expect to see happen. Once the metric is proven, document common trends.

- When to Use It:** what prompted creation or use of this metric? How has it historically been used?
- When to Stop Using It:** when will it outlive its usefulness, become misleading or extra baggage? Design this in from the start.
- How to Game It:** think through the natural ways people will warp behavior or information to yield more ‘favorable’ outcomes.
- Warnings:** recommend balancing metrics, limits on use, and dangers of improper use.

**Figure 2: Metric Planning Checklist**

Following is an example, applying the checklist to the key metric “value”.

### 7.1 Key Metric: Business Value Delivered (checklist)

**Name:** Business Value Delivered

**Question:** What is the rate of return on the investment, when will it begin, when will it break even, and what will we earn?

**Basis of Measurement:** Net cash flow per iteration

**Assumptions:** Delivery of value for a project team occurs at the end of the iteration; however the business only realizes value when software enters production.

**Level and Usage:** Project, Program, and Portfolio (all levels)

**Expected Trend:** Highest valued features will be delivered early in the project, with diminishing returns as remaining value is extracted.

**When to Use It:** In evaluating, planning, prioritizing, and executing projects. This key metric should be widely understood by all project participants.

**When to Stop Using It:** When the investment is retired.

**How to Game It:** Deliver software too early in the project (with no real user value) so that ROI is not really occurring at the planned rate. Use unrealistic assumptions in the valuation.

**Warnings:** Valuing features or groups of features is tricky, and requires numerous assumptions, which should be documented. Re-test and calibrate the valuation model once software is in production and real financial performance data is available.

## 8. Useful Diagnostics

### 8.1 Velocity (checklist)

**Name:** Velocity

**Question:** How much software can my team deliver per iteration?

**Basis of Measurement:** Story points or “ideal engineering hours”

**Assumptions:** The team is delivering working software every iteration.

**Level and Usage:** Velocity is most useful at the project level. It allows the team to forecast how much work they can expect to complete based on prior efforts.

**Expected Trend:** Velocity can be affected by many things: Changing team members, obstacles, toolsets, difficulty of feature or amount of learning required, etc. will lower the velocity of the team. Barring unexpected obstacles, a stable team on the same project with the required resources will generally gain in velocity during the course of the project, then plateau.

**When to Use It:** Velocity is a very useful metric for the team, and should be used during the course of the project once work has started.

**When to Stop Using It:** In a longer project when the team, resources, and technology are all stable, velocity will also become stable. The team may suspend collecting velocity since it is "known."

**How to Game It:** Velocity is only meaningful to the exact team providing the data - each team will estimate their work differently from other teams. For example one team might estimate 1000 hours while another 600 hours for the same work, and both will complete the work in the iteration. Comparing velocity of teams is problematic. If teams know they are getting compared, they can inflate their estimates, and hence increase their velocity while completing the same amount of work.

**Warnings:** Velocity is not the same as value. A team with excellent velocity could spend months quickly and effectively delivering software that does not have the investment potential. Comparing velocity of teams is problematic (see above) and should be avoided: this diagnostic is a barometer for the team itself, as a unit. Team member velocities are problematic: velocity should be measured at the team level, since that is the unit that must self-organize to produce value

### 8.2 Other Contextually Useful Diagnostics

Space does not allow exploration of numerous diagnostics here, so they are simply outlined (further detail may be obtained from the authors). Following are some examples deemed useful in contexts familiar to the authors – of course, the reader must judge whether they are also useful in his or her own situation:

- Agile practice maturity
- Obstacles cleared per iteration
- Team member loading
- Obstacles carried over into next iteration
- User Stories carried over into next iteration
- Iteration mid point inspection
- Unit tests per user story
- Functional (Fitness) tests per user story
- Builds per iteration
- Defects carried over to next iteration

## 9. Be Prepared for Resistance

Change is difficult, and new diagnostics may be resisted if their message is problematic or discomfiting (ex: if they fly in the face of prevailing optimism). Metrics can become political: Agile increases process transparency, so any who once hid behind reports and documentation will find this shift challenging, and may react strongly. Some fear that measurement will reveal individual or organizational weaknesses, and these people may politically undermine the effort. Measurement may be perceived as a loss of control. For these reason, when developing diagnostics, it is strongly recommended to involve performers from the measured activities. The book *Fearless Change* [6] offers strategies (and cautions) for enlisting resisters constructively in an organizational change process.

It is not uncommon for an organization to try to restrict process improvement to a particular department. Be aware that local optimization is likely to throw a spotlight on bottlenecks *outside* the department undergoing improvement. Improved throughput may even *create* new bottlenecks in other departments. A diagnostic set by a sub-organization (like IT or R&D) should be a temporary phenomenon, because optimizations based on such local measurements may create system-wide effects that can actually decrease overall organizational performance, negating the intent of the local optimizations. The sources of this effect may be difficult to detect [8]. It is advisable to promote a more direct relationship between organizational strategy and software measurement, but this is not always obvious: difficulty determining what to measure may indicate dissonance

between the objectives of various stakeholders (ex: Business Ops, IT, upper management).

A good understanding of the heuristics for Agile metrics may be helpful in countering resistance. However, delivering business value is often the best argument overall.

## 10. Conclusion

Measurement techniques inherited from prior methodologies may hamper teams trying to make the shift to high performance with Agile methods. However resources do exist, as outlined in this paper, to help teams design good metrics for Agile process improvement. To promote the creation of value as the primary measure of progress, the authors recommend choosing one key metric that is closely tied to the economics of investment. All subordinate metrics should be recognized as simply tools to achieve improvement vis-à-vis the key metric, and not as ends in themselves – and so it is suggested that they be called “diagnostics” as a reminder of their temporary and secondary status. The heuristics in Section 2 and checklist in Section 7 may also help teams think about healthy metrics and diagnostics. The use of these tools should be evaluated by multiple teams over the next year to determine their soundness.

## 11. Acknowledgements

This paper was shaped by the many annoying but important questions asked by our esteemed colleagues in Virginia, particularly Mishkin Berteig. Our thanks to all of you.

## 12. References

- [1] Anderson, David J., and Eli Schragenheim, *Agile Management for Software Engineering: Applying the Theory of Constraints for Business Results*, Prentice Hall, 2003.
- [2] Cohn, Mike, *Agile Estimating and Planning* Prentice Hall, 2006
- [3] Collins, Jim, *Good to Great: Why Some Companies Make the Leap... and Others Don't*, Collins, 2001.
- [5] Goldratt, Eliyahu M., *The Haystack Syndrome—Sifting Information Out of the Data Ocean*, North River Press, 1990.
- [6] Manns, Mary Lynn and Linda Rising, *Fearless Change: Patterns for Introducing New Ideas*, Addison-Wesley Professional, 2004.
- [7] Mary Poppendieck, “Measure UP”, 2003, retrieved Dec. 26, 2005 from <http://www.poppendieck.com/measureup.htm>
- [8] Poppendieck, Mary and Tom Poppendieck, *Lean Software Development: An Agile Toolkit*. Addison Wesley Professional, 2003.
- [9] Womack, James P. and Daniel T. Jones, *Lean Thinking: Banish Waste and Create Wealth in Your Corporation*, Revised and Updated, Free Press, 2003.
- [10] Mishkin Berteig, “Appropriate Metrics”, 2005, Retrieved Dec 15, 2005 from [http://www.agileadvice.com/archives/2005/05/appropriate\\_metrics.html](http://www.agileadvice.com/archives/2005/05/appropriate_metrics.html).
- [11] Moore, Geoffrey A., *Crossing the Chasm*, 2002, Collins.

**Deborah Hartmann** has over twenty years experience as a developer (programmer, analyst, team lead) and is a Scrum Master/Practitioner and Agile team coach. Her most recent experience with enterprise Scrum rollout brought home the dangers and opportunities in measuring Agile teamwork. Deborah is passionate about helping teams replace dysfunctional processes with common sense and collaboration, and has used Scrum to replace both heavy PMO culture and ad hoc development. Deborah facilitates OpenSpace for XPday North America, is co-founder of ScrumToronto and is active in the international Scrum community.

**Robin Dymond** has spent 16 years helping companies deliver complex software products and services. Robin brings extensive knowledge on software development practice and process improvement to his clients. He has turned around software development projects, and implemented best practices in object oriented design, software patterns, and development processes using the Agile family of methodologies. As a Software Management Consultant Robin is rolling out Scrum with XP to teams working on large financial enterprise systems redevelopment and upgrades, mentoring staff coaches and providing consulting and recommendations on Lean Agile Program implementation. Robin is an advisor to the Calgary Agile Users Group, and mentored a year long agile student project for the University of Calgary. His current interests include enterprise conversion to Agile methods, and Return on Investment (ROI) analysis of Agile methods. Robin Dymond received a BSc. in Electrical Engineering from the University of Calgary.