Backlog Considered Harmful?

contributed by Steve Adolph

"The hardest single part of building a software system is deciding precisely what to build. No other part of the conceptual work is as difficult as establishing the detailed technical requirements. No other part of the work so cripples the resulting system if done wrong."

No Silver Bullet: Essence and Accidents of Software Engineering. Computer
FP Brooks Jr - IEEE Computer Society, Washington, DC, 1987

Does Traditional View of the Backlog Make the "Hardest Part" of Software Development Invisible?

The present mental model of the backlog is often visualized as a "stack of plates." It is harmful because it encourages making the hardest part of building a software system invisible. Classical Agile Methodologies such as Scrum and XP provide little or no guidance for "...the hardest single part of building a software system..." leaving it up to development organizations to determine what happens beyond the Product Owner.

Both the Agile Manifesto and Agile Principles encourage a coding-centric view of software systems and therefore diminish work associated with the "...hardest single part of building a software system..."

Working software — over comprehensive documentation. – Agile Manifesto

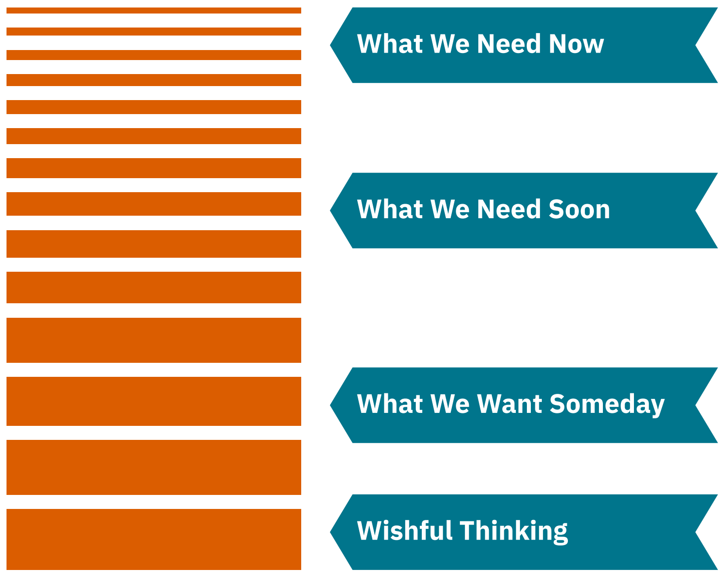
Working software is the primary measure of progress. – Agile Principles

This point of view frequently leads to sub-optimizations of an organizational value stream, where coding-centric development teams can demonstrate performance metrics, but overall throughput through the value stream is low and so is customer satisfaction.

This code-centric view of the backlog treats the backlog as a reservoir of work for a coding team and diminishes the roles that are needed to "...decide precisely what to build...," such as the Business Analyst. The classical model of the backlog may work with smaller teams, but it is not powerful enough to describe strategies for managing the challenges of modern enterprise software development.

A Stack of Plates?

The most common presentation model of a backlog is one that looks like a stack of plates. In this "stacked plates" model, each backlog item appears to be the same as any other. Backlog items are stacked on top of each other and processed in sequence. Showing the items in a stack such as this implies that they are all at the same level of abstraction and readiness.



Stacked Plates Metaphor for Representing the Backlog

Problems with Stacked Plates Model

Coding-Centric

The Stacked Plates model encourages a "coding-centric" view of the backlog making the "…hardest part of building a software system…" invisible. That is, the work required to ideate and refine client wants and needs into backlog items that a software team can implement. The software focus of the Agile Manifesto "working software over comprehensive documentation" and Agile Principles "working software is the primary measure of progress" have sometimes created a mindset that only coding creates value. The work required to "…determine precisely what to build…" is invisible. Yet, it is precisely the work of analysis and design that we want to manage in an agile manner.

This coding-centric view means we don't have guidance for how roles, usually associated with "…determining precisely what to build…" such as the Business Analyst, should work with a backlog.

Impedes Flow

Most models of the backlog tend to show the backlog to the left of a team, encouraging the view that the backlog is a reservoir of work for a team. The result is:

- Analysis and design are often done "up front" using some form of waterfall-agile hybrid.
- There is minimal engagement and collaboration between stakeholders and the team.
- The backlog becomes a committed queue of work that impedes flow, rather than facilitating a flow, of value with rapid learning guiding prioritization decisions. Flow is critical to value creation in Agile.

Disconnects the Team

In classical Scrum, the value stream begins and ends with the Product Owner. Scrum provides no guidance for what happens beyond the Product Owner, that is, where those backlog items came from, and what happens to the increment once it is accepted by the Product Owner.

The team often becomes a group of coding wizards who are disconnected from the problem and the customer experience. The team makes design decisions based on technical implementation requirements without a deep understanding of their customer. This disconnection is visible in numerous organizations that hire armies of technically competent contractors who have little or no understanding of the problem.

Overprocessing

A coding-centric development team will often require detailed technical specifications. The result is the backlog is not a backlog of user stories that encourage collaboration between the team and the Product Owner, but rather a queue of detailed specifications the team pulls and codes.

Usually an “invisible” Business Analyst working upstream of the Product Owner will write detailed technical specification (sometimes near pseudo code) to avoid ambiguity and the risk of building a deficient system.

False Precision

This model tends to encourage a view that backlog consists of lots of small, granular items, all of which are forced rank in priority order. It implies a level of certainty and precision that may not yet exists. If this level of certainty does exist, we may have made prioritization decisions far too early and are using a backlog to masquerade a set of committed requirements as backlog items. If we are following Agile practices in our analysis processes, then we want to defer the creation of granular backlog items, such as user stories, until the last responsible moment.

A Better Model – Rock Crusher Metaphor

The problems we have highlighted are not a dismissal of the backlog itself, rather it is calling out that our typical model of the backlog, one that visualizes the backlog as a stack of plates, is problematic. We need a better model that is:

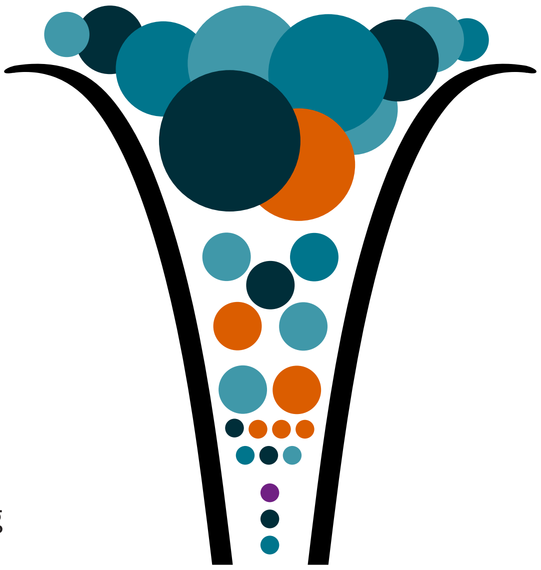
- More nuanced;
- Visualizes all the work required to build a software system;
- Encourages collaboration between the roles; and
- Reflects the uncertainty associated with the backlog items.

A better model is what many have called the "Rock Crusher."

The Rock Crusher is a metaphor for good backlog management practices that visualizes all the team's work.

The intention of the Rock Crusher is to visualize the flow of value through the value stream from concept to cash. From big idea to delivered code. Coding is part of value stream. Furthermore, the Rock Crusher more accurately reflects the state of the work.

The conventional "Stack Plates" model implies that all work in the backlog can be forced rank, whereas the rock crusher calls out the ambiguity of the prioritization of backlog items.


The "Rock Crusher" Model of a Backlog

Learn More

Rock Crusher for Backlog Management

Read:

1. Introduction
2. Ceremonies
3. Implementing
4. Roles
5. Crushes/Backlog Items

Considered Harmful
Anti-Patterns
Re-acquainting
Work not Road Mapped

View:

Rock Crusher Infograph

User Story Infograph