Rock Crusher Guidance Article

Crushers

contributed by Steve Adolph

Contents

“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

Crushers

Purpose

The purpose of the Rock Crusher model is to visualize ALL the work required to go from “concept to cash,” not just the work of coding. Often the work required to learn precisely what to build is invisible and therefore unmanaged.

A Crusher is a backlog item (or Rock, in this metaphor), that makes learning “precisely what to build” visible and manageable. Crushers capture the important learning work that must be done in a reductive transformation development process to go from a wild speculative concept to delivering a valuable working solution. Typically, the output of a Crusher is knowledge, often expressed as a model, which can then guide the Crushing of large abstract Rocks into smaller, more concrete ones.

While Crushers do not necessarily result in “working software,” they create the knowledge required to decide precisely what to build. They create economic value by reducing the risk of building the wrong thing and creating the knowledge necessary to align individuals with the ultimate solution. Like all well-formed backlog items, Crushers are objective, and verifiable, and have a means of testing that they are done, usually in the form of testable acceptance criteria. “Accepted” Crushers are presented at the iteration review, like all other work that has been accepted by the Backlog Owner.

Created During the Crush Meeting

Crushers:

- Are typically created during a Crush Meeting, when you find that significant work is required to create the verifiable knowledge need to guide the Crushing of a Rock.
- Are added to the backlog and prioritized by the Backlog Owner, just like any other backlog item.
- Result in a model of some kind that can be verified and then used to guide the Crushing of the Rock.
- Aren’t the only way to guide Crushing and shouldn’t be the automatic go-to for a team.
    - For example, Spikes are a useful tool for understanding a problem and generating the knowledge required to Crush a Rock. But it isn’t just coding Spikes that generate knowledge. Simply coding part of the system and presenting that to the Stakeholder may be a more effective way to gain the knowledge required. As a team, consider all the options available to you.

The decision whether to use a Crusher, a code Spike, or just code part of the system to create the required Crushing knowledge, is driven by economic considerations. Simply, what is the fastest and cheapest way to generate the knowledge required to understand what needs to be built? The team, along with their Backlog Owner (and Initiative Owner and Product Manager, as applicable), make these decisions. Sometimes, it is cheaper and faster to learn by creating a model; at other times, you learn faster and more cheaply by building an increment of the system.

Crushers and INVEST

Crushers are backlog items and, just like other ready backlog items, should satisfy Bill Wakes 2003 INVEST criteria. For Crushers, the INVEST criteria may be interpreted as:

Independent	Completion and acceptance of this Crusher does not depend on the completion of another future Crusher.
Negotiable	Crushers are created collaboratively with the team to generate the knowledge the team needs to better understand the backlog items, and inform their subsequent decisions about implementation.
Valuable	The learning the Crusher will create is valuable to the team and will help inform their subsequent design and implementation choices – including the Product Owner's prioritization decisions.
Estimatable	The team can estimate the Crusher. If the team can't properly size the Crusher, then they don't know enough about Crusher to perform it.
Small	The Crusher is something that can be delivered within a single iteration, or within the appropriate SLA for a Kanban-based team.
Testable	There is a clear, well-defined set of acceptance criteria which the team can definitively say "yes, we did learn that," or "no, we did not learn that."

Crusher Examples

A good Crusher can be written in the format:[1]

- Learning: What do we need to learn and why do we need to learn it?
- Acceptance: How will we know if what we learned is correct or has a chance of working?

Example of a possible Crusher very early, when an Initiative size Rock is tossed into the Crusher	Create a use case brief to decide what are the primary categories of services the system will provide to its users. **Acceptance** • Brief identifies the main actors and their primary workflows. • Reviewed by <stakeholder> and has their agreement.
Example of a possible Crusher for understanding the behaviour of a system	Develop a State Transition Model of the Engine Control Module (ECM) Function so we can populate the state transition table. **Acceptance** • Executable model of the ECM state transition diagram responding to stimuli. • Reviewed by <stakeholder> and has their agreement.

1 Loosely related to the classical user story format of "AS A <persona> I NEED TO <what> SO THAT <why>"

Crusher Bad Smells

Crusher is Not "Right-sized"

This smell occurs when the Crusher is not completed within a single iteration (or within the appropriate SLA for a Kanban team). The intent of the Rock Crusher is to make the work required to learn precisely what to build visible, and to enable the team to effectively manage that work. If the Crusher is excessively large, or allowed to roll over sprint to sprint, or is simply open ended, then the work is effectively invisible and unmanaged. Like all good "Ready" backlog items, a Crusher needs to be right-sized.

Crusher is Not Scoped with Well-Defined Testable Acceptance Criteria of its Learning Outcome

This smell usually occurs when Crushers specify "what" they will deliver, but not the why. Simply saying "create a use case model for backlog item xyz" may not foster learning. The point of the Crusher is to foster learning, so it is necessary to declare what we need to learn, or what we expect to learn.

Crushers are "Private" to the Analyst

This smell occurs when the analyst believes they have personal ownership of the Crusher process and fully direct it. This reduces transparency and alignment. The purpose of the Rock Crusher is to help guide the team in "...deciding precisely what to build" and quickly create alignment. This can't be accomplished if the Analyst treats the resulting models and work as their personal property. The result is an old fashion "throw it over the wall" mentality where the Analyst creates user stories and the team simply implements them. This situation can occur where the analyst role is performed by someone with an analyst-related job title, such as Business Analyst or Business Architect.

Crushers are Tasks

This smell occurs when the analysis work for implementing a Rock during an iteration is broken out into functional Rocks such as analysis, design, implement and test. This can easily lead to waterfalling an iteration and create excessive work in progress. Just as we don't create separate Rocks for coding and for testing, we don't create a separate Rock for analysis if it is a normal part of the work the team can do to implement a Ready Rock in a single iteration. It isn't unusual for an Analyst to be a development team member who collaborates and swarms with developers and testers to implement a Rock and deliver a solution increment. Crushers are about creating smaller more concrete Rocks, not about partly implementing Rocks.

Most of the Backlog Items are “Crushers”

This smell occurs when Crushers are used to implement a stealth big-up-front design process. While the first iteration or two of a BIG Rock may be highly biased towards a lot of Crushers, you should be suspicious if working solution increments do not soon start emerging.

Mandating Crushers as Part of the Development Methodology

This smell occurs when the organization’s software development process mandates the use of Crushers rather than letting the team determine the economic utility of the Crusher. What we mean by mandating the use of Crushers is that the team has no choice in the decision whether to use Crushers or not. This often results in a lot of “busy-work” creating models that delays progress, when either a spike or simply implementing a part of the system would create validated learning faster. The resulting Crusher is not valuable in INVEST terms.

Mandating the Fidelity of a Crusher

This smell frequently occurs in regulated organizations where traceability is used for demonstrating compliance. Models are often a big part of this traceability. However, there is a significant difference between the models created by a Crusher and the models we require for traceability and compliance. The intent of a Crusher model is to quickly and economically generate verifiable knowledge for a team. A traceability model is primarily a documentation model which accurately describes the intended behaviour the system needs to have with a reasonably high level of fidelity that can be used by auditors to validate the resulting system. This level of fidelity may be far above and beyond what the teams require to create validated learning quickly and economically and, thus, is like mandating Crushers as part of the development methodology. The Crusher is not valuable in INVEST terms.

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