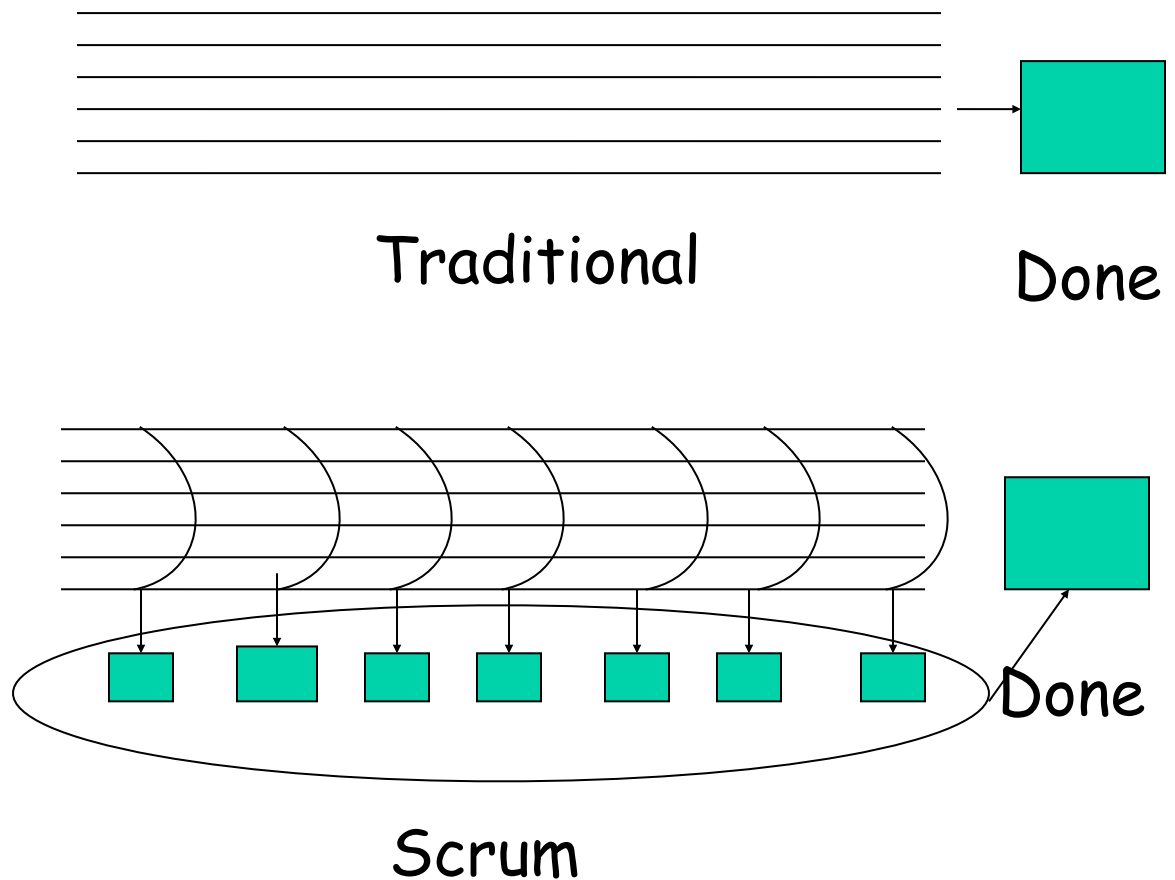


Scrum

Done, Undone, and the Consequences





1. Development team must be able to produce a completely "done" piece every Sprint.
2. Product Owner must inspect and adapt to optimize ROI every Sprint.
3. "Undone" work must be identified.

Done: a complete increment of potentially shippable product

- More time to implement
- Solid engineering practices
- Solid engineering infrastructure
- Required for transparency



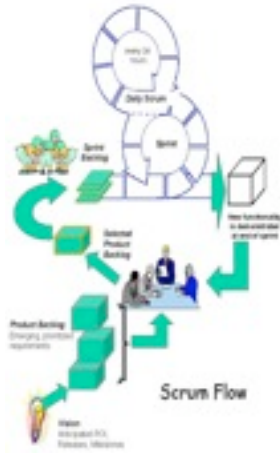
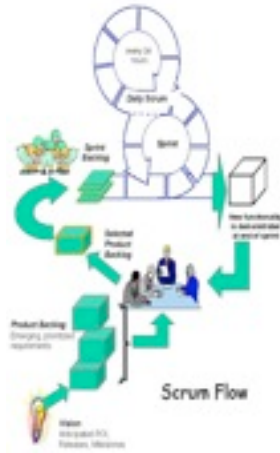
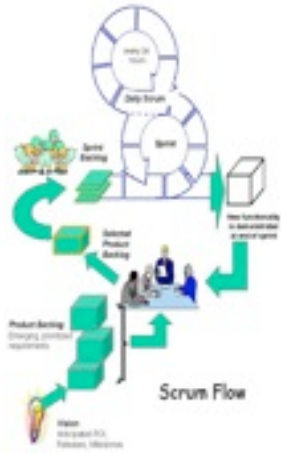
Exercise: Defining "Done"

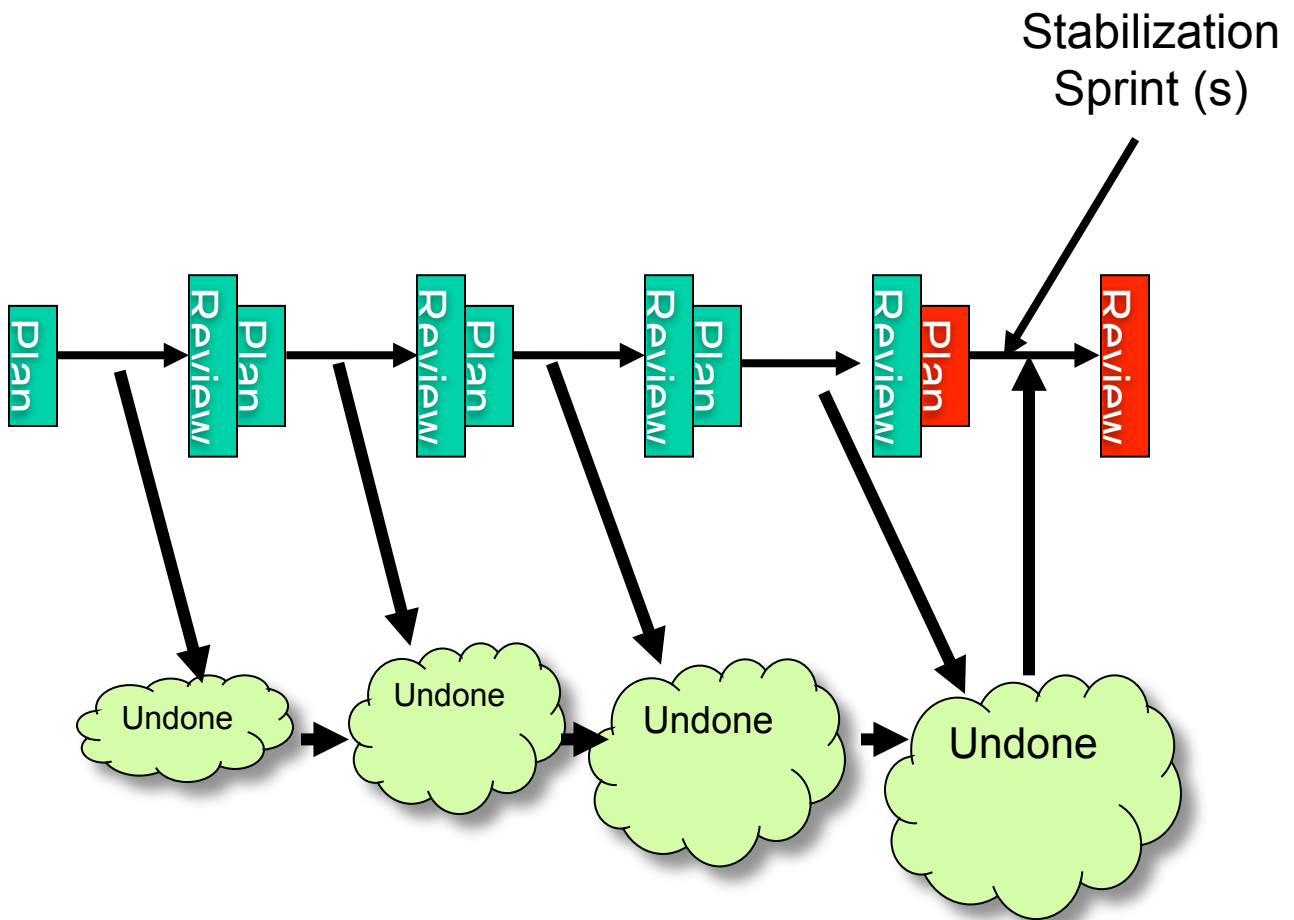
At your table::

- What does "done" mean in your current project?'
- What issues do you see with this definition of done?
- How would you address them?
- What engineering problems do you see with this approach?
- How would you rectify them?

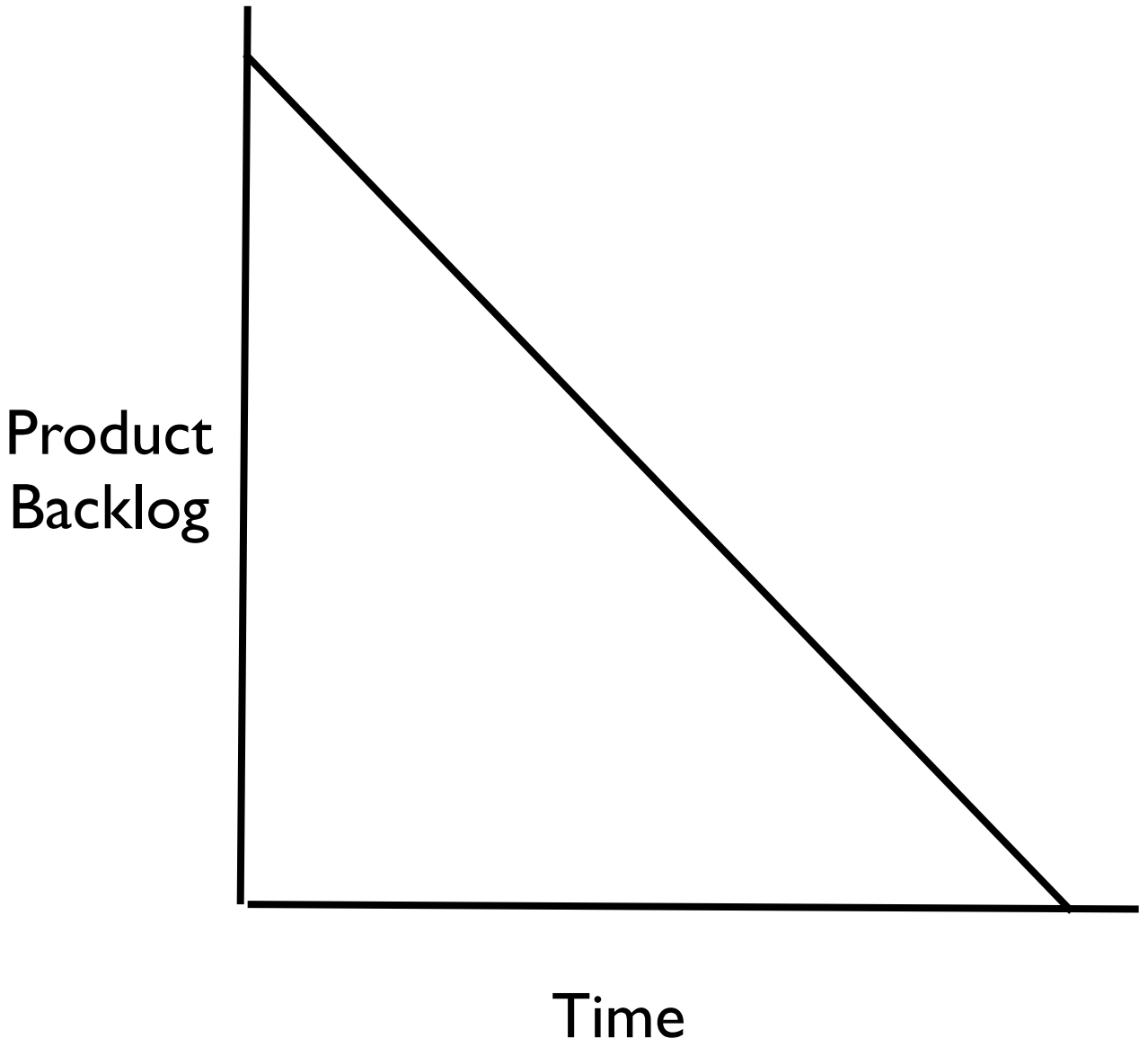
Work Item	Usual	Rec. start	Done
Requirements analysis	25	25	25
Design of architectural components (UI, System, Data	15	15	15
Design review	0	5	5
Design of tests (system, user acceptance, integration)	0	10	10
Design review	0	3	3
Design of documentation	0	2	2
Design Review	0	1	1
Refactoring of existing design	0	0	8
Design of unit tests for new code	0	3	3
Design of unit tests for code to be refactored	0	3	3
Writing new code	10	7	7
Writing refactored code	6	3	3
Code review (or pair programming)	0	4	4
Write functional tests	8	4	4
Write integration tests	0	4	4
Write documentation	4	4	4
Unit test code	0	2	2
Identify and rectify defects	0	2	2
Subsystem/team build	6	2	2
Identify and rectify defects	1	1	1
Unit test for subsystem/team code	0	2	2
Identify and rectify defects	0	2	2
System/integration build	1	1	1
Identify and rectify defects	0	2	2
System, functional tests	1	2	2
Identify and rectify defects	1	2	4
Integration tests	0	0	2
Identify and rectify defects	0	0	5
Performance tests	0	0	1
Identify and rectify defects	0	0	2
Security tests	0	0	1
Identify and rectify defects	0	0	2
Regression test	0	2	2
Identify and rectify defects	0	8	8
Documentation test	0	1	2
Identify and rectify defects	0	1	1
Total work expended requirement	78	118	148
Work remaining per requirement	65	30	0

5

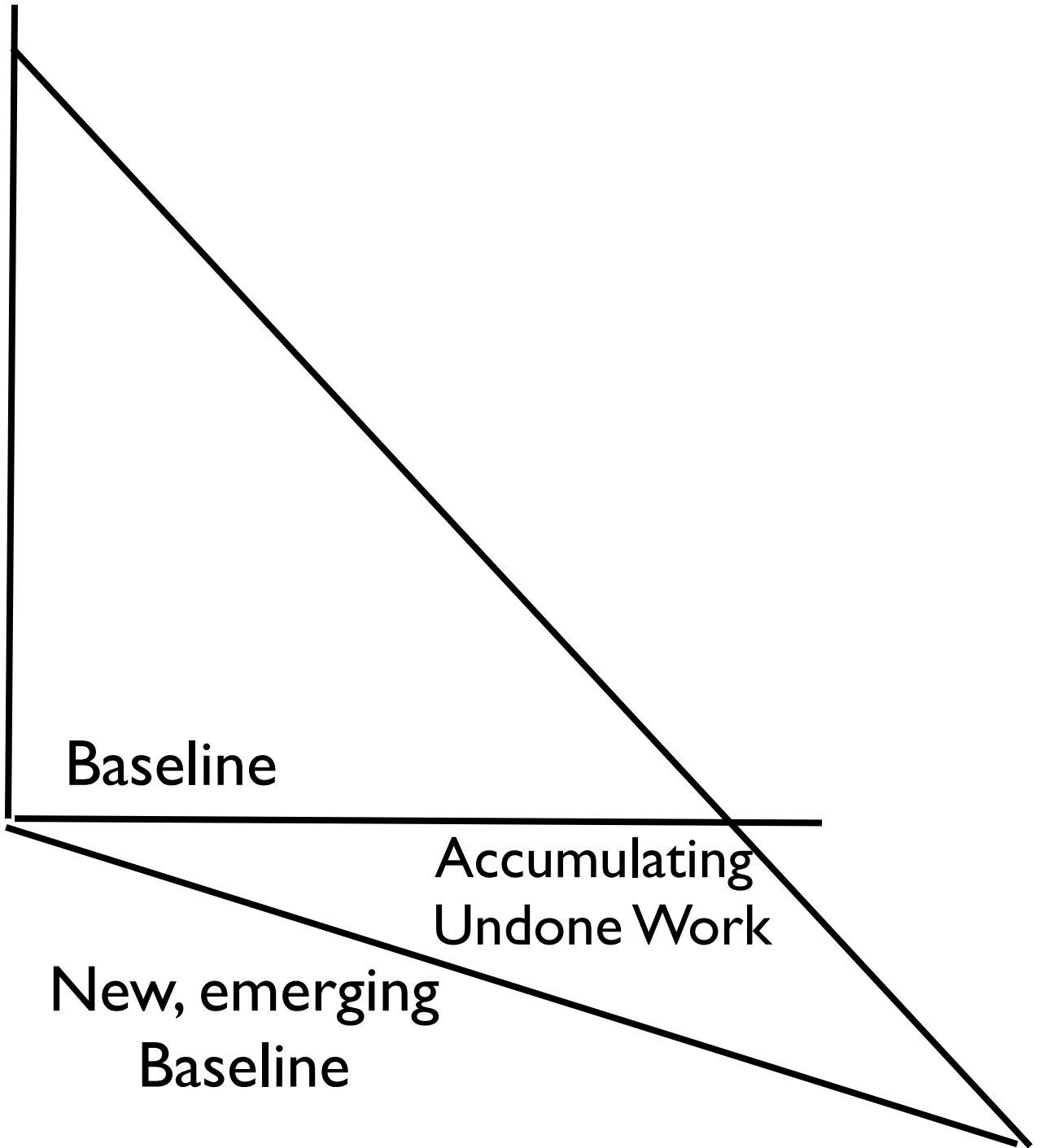




Create Product Backlog Item for “Undone Work”



Product Backlog



Baseline

Accumulating
Undone Work

New, emerging
Baseline

"Done" and "Undone" Work

1. "Undone" work is a Product Backlog item that must be completed prior to any release.
2. It incrementally grows every Sprint, as the proportion of Sprint undone/Sprint done.
3. This increase in release "undone" work appears linear, but the "undone" work increase in more rapid and unpredictable.

"Done" is not defined.



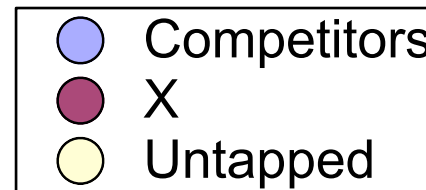
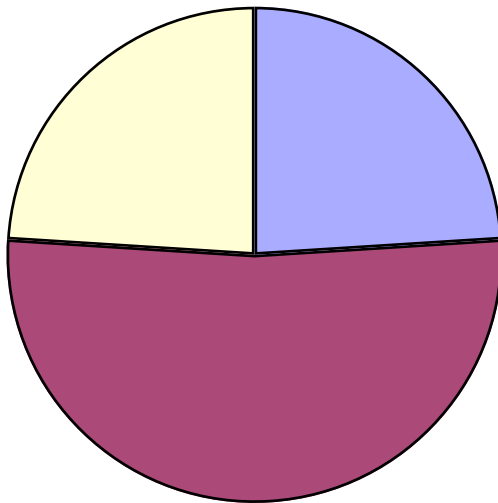
11

1. No stable velocity from which to estimate;
2. Inaccurate product backlog burndown;
3. Product Owner doesn't know progress or status;
4. The Product Backlog probably isn't in good shape;
5. Team doesn't know how much to select in Sprint Planning meeting; and,
6. Product Owner doesn't know what is being inspected at Sprint Review.

The Nexus of the Crisis

1. Ground Zero is the customers belief that they can demand something, and the developers willingness to unconscionably cut quality to support the belief.
2. The most obvious consequences are people who don't like the profession they are in and customers who don't like the profession.
3. The long term consequences are failing products, failing companies, and hateful work. We are there now.
4. The canary is the definition of "done"

Market Segmentation



Operating system market

Competitive advantage through:

1. Quality (as perceived by customer);
2. Functionality (to do everything a customer might want to do, and to do it well);
3. User Friendliness (ease of use and sizzle - iPod);
4. Price (a function of cost); and,
5. Difficulty of conversion to alternatives.

Product cost has several dimensions:

1. Cost of adding new features;
2. Cost of coupling new features to existing functionality, architecture and infrastructure;
3. Cost of maintaining existing product; and,
4. Cost of providing customer support on existing product.

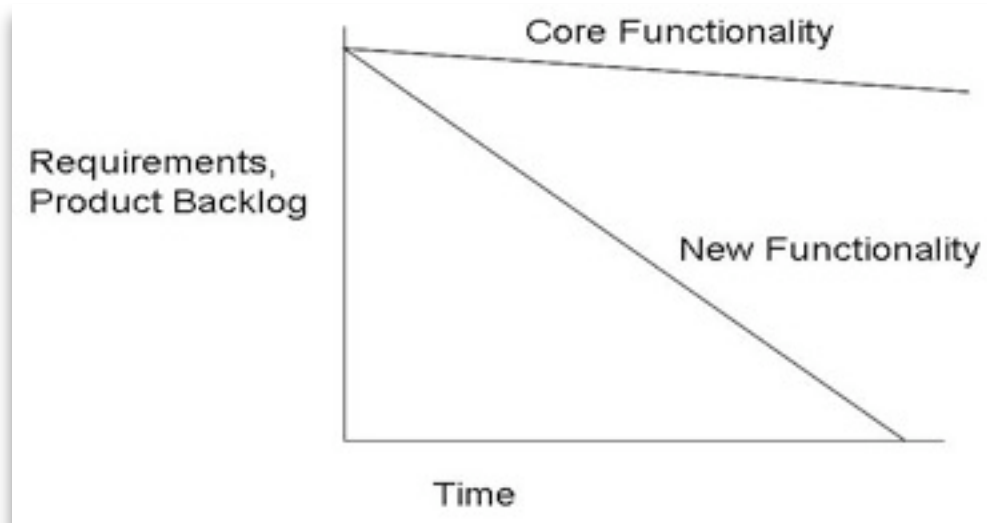
Problem

Most new functionality requires changes to core software;

For many, it takes longer to change core software than build new functionality (below, 20 to 1);

Core software may be a constraint;

Highlighted by iterative, incremental.



Core software

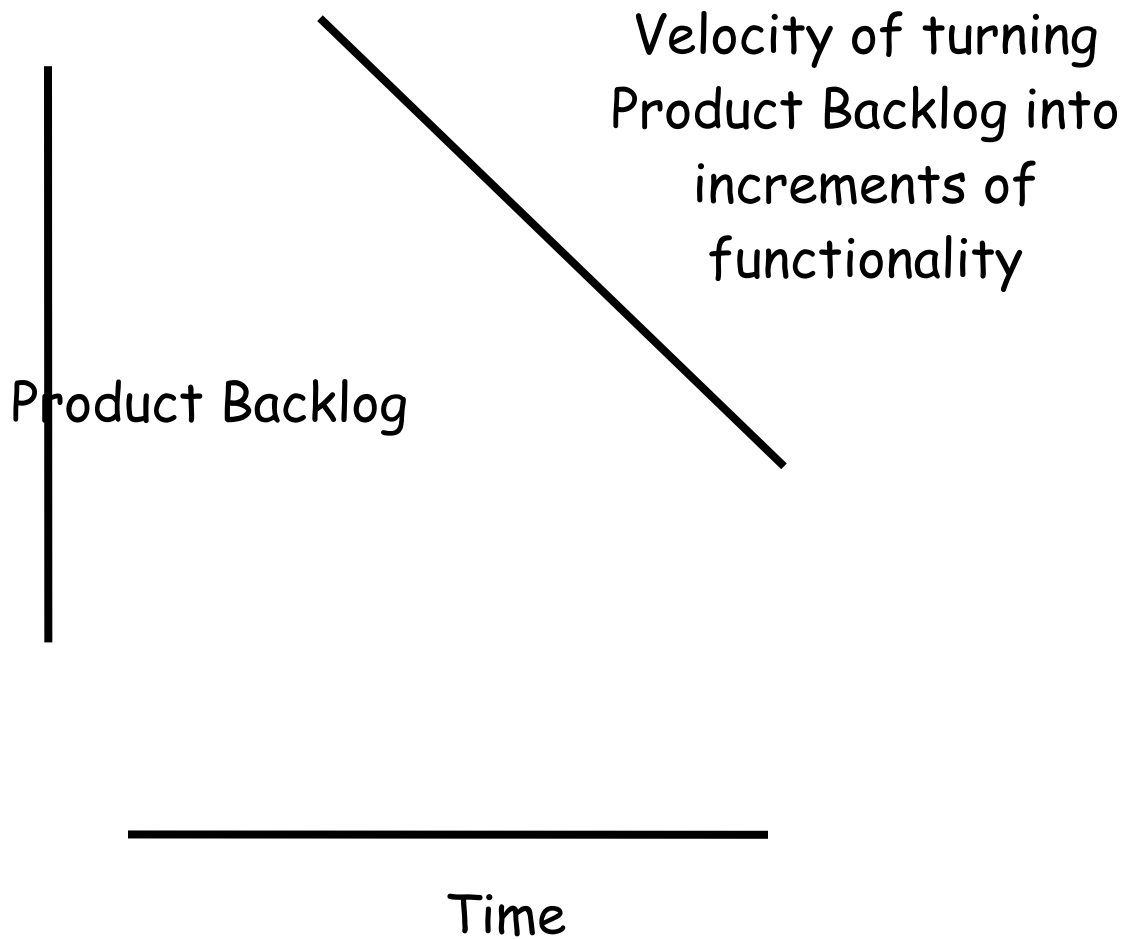
(infrastructure, legacy)

Core to all business processing
... all other functionality hangs
from it.

Common characteristics:

1. Fragile
2. Inadequate test harnesses
3. Only a few people
knowledgeable and willing to
work on it.

Project Management Variables



Other variables: Quality, Value



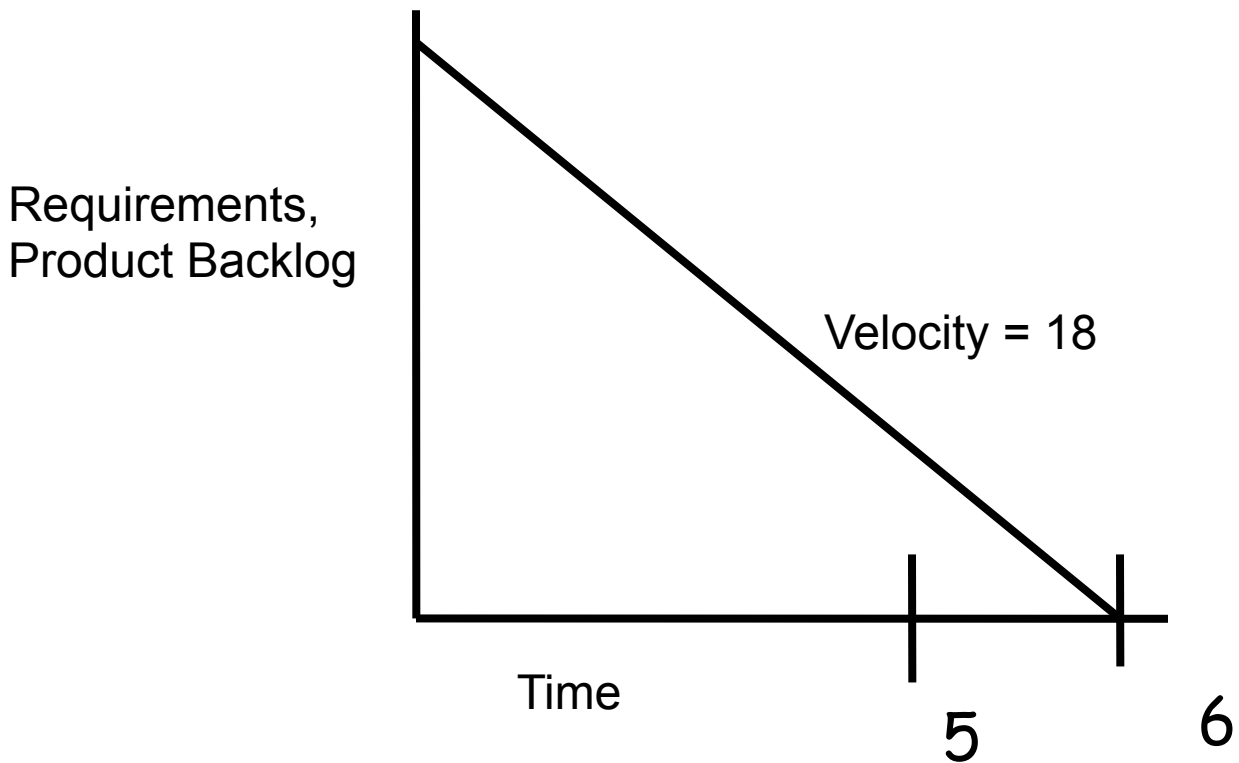
If it takes 4 hours to develop a piece of functionality with quality:

Analysis	Design	Refactor	Code	Test	Doc
50	30	30	40	70	20

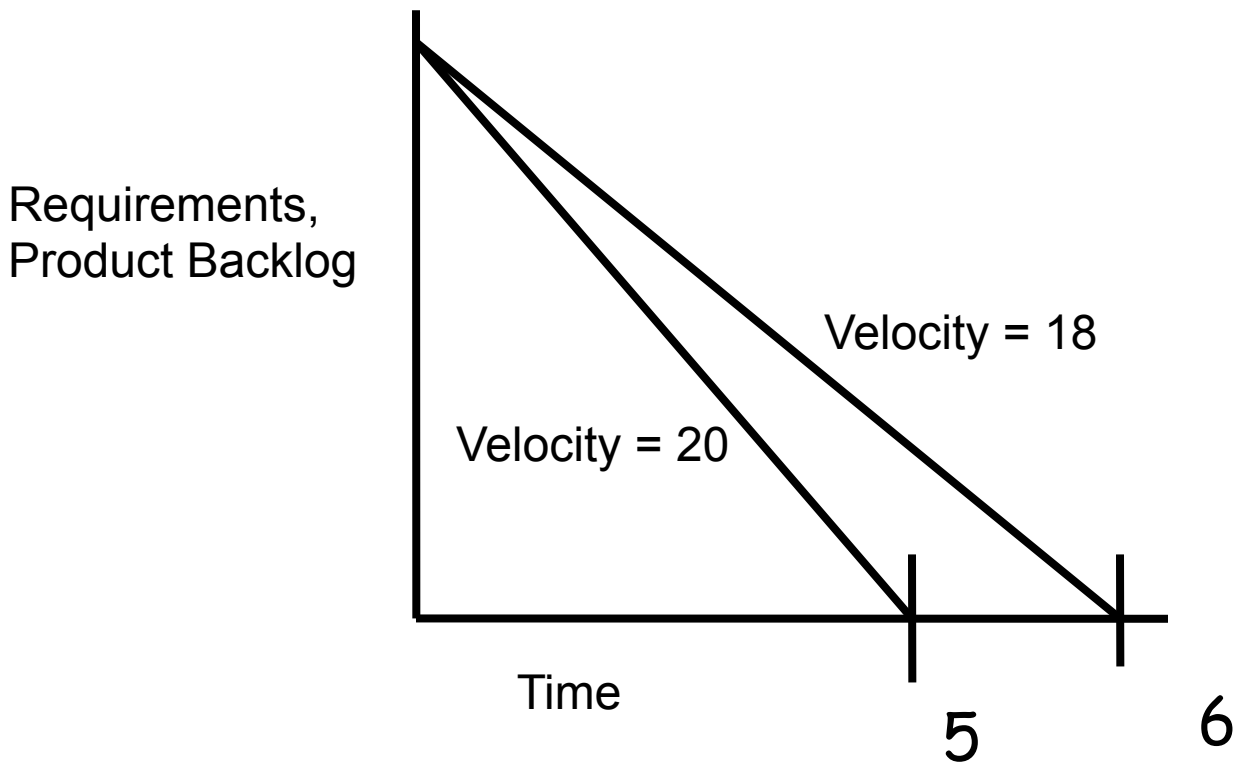
At least three times as much can be done by dropping the quality:

Analysis	Design	Refactor	Code	Test	Doc
20	10	0	20	15	10

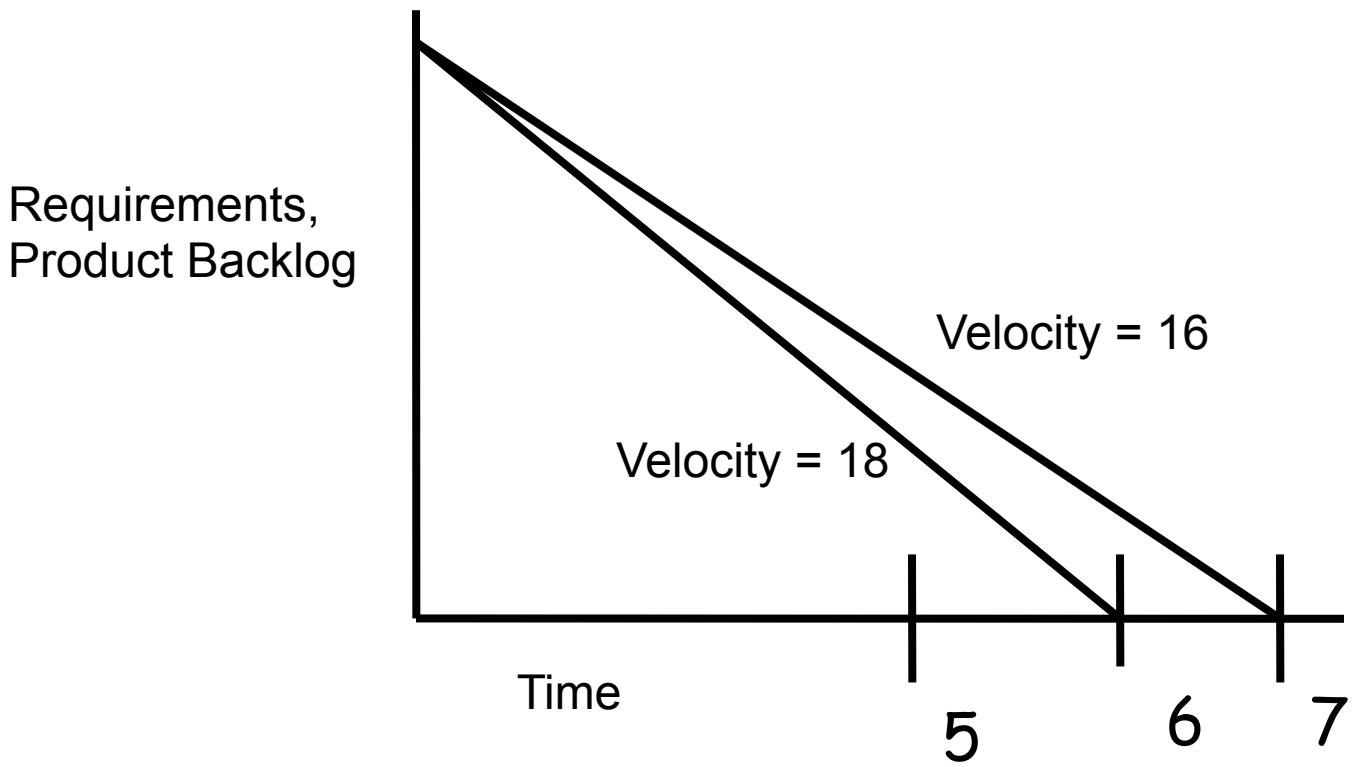
Where does core functionality come from? Is it bought from a malicious competitor?



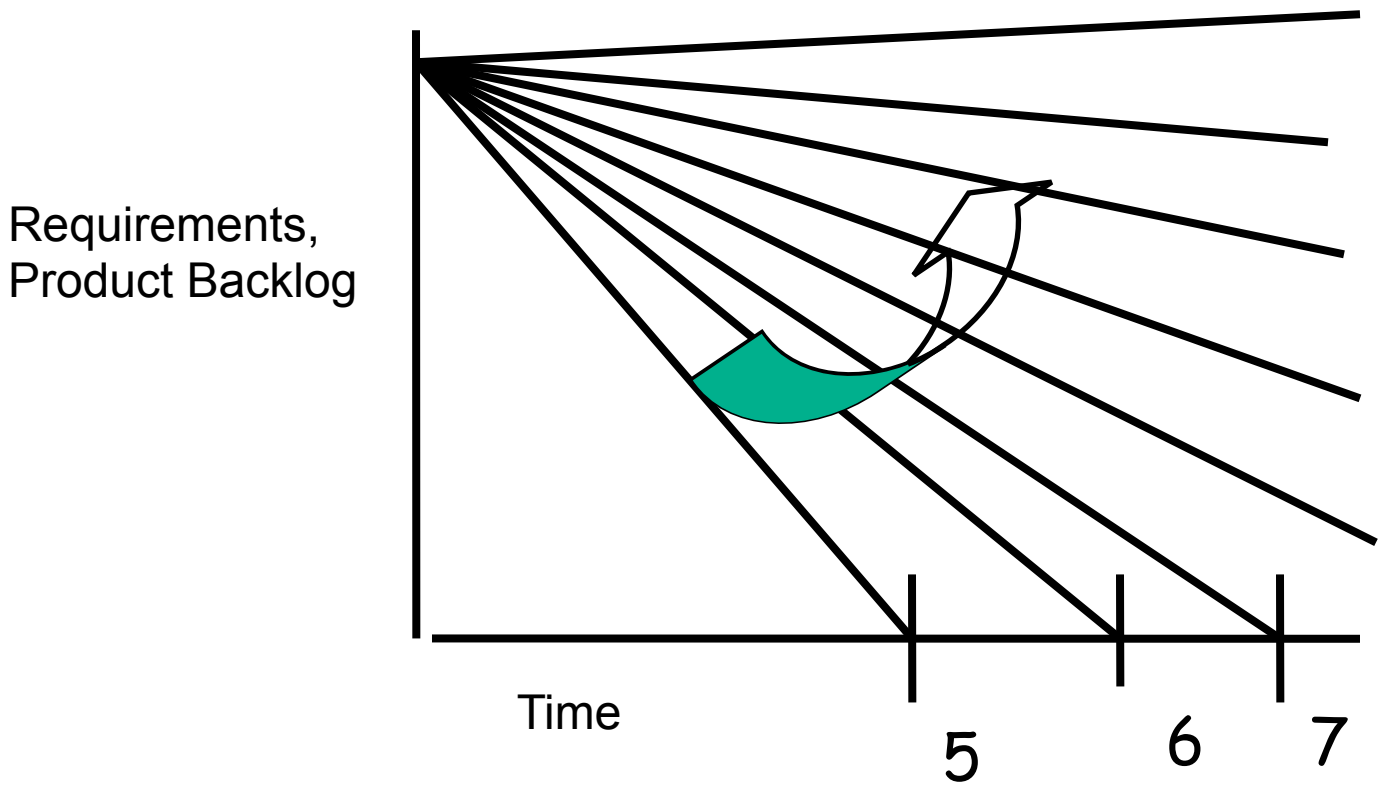
Release 5 and 6



Releases 7 and 6



We dig ourselves into a corner



Exercise

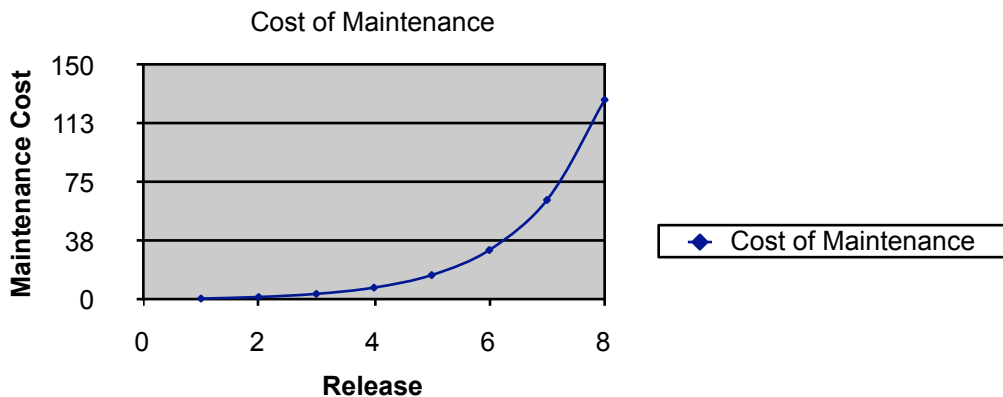
Planned work consists of:

1. Function 1: 20 units of work, 15 new, 5 core
 2. Function 2: 40 units of work, 25 new, 15 core
 3. Function 3: 30 units of work, 20 new, 10 core
- Velocity for new functionality is 15 units of work per Sprint per team.
 - Velocity for core functionality is 5 units of work per Sprint total.

You need a release with all three functions in three months. What do you do?

How to tell the health of your core software:

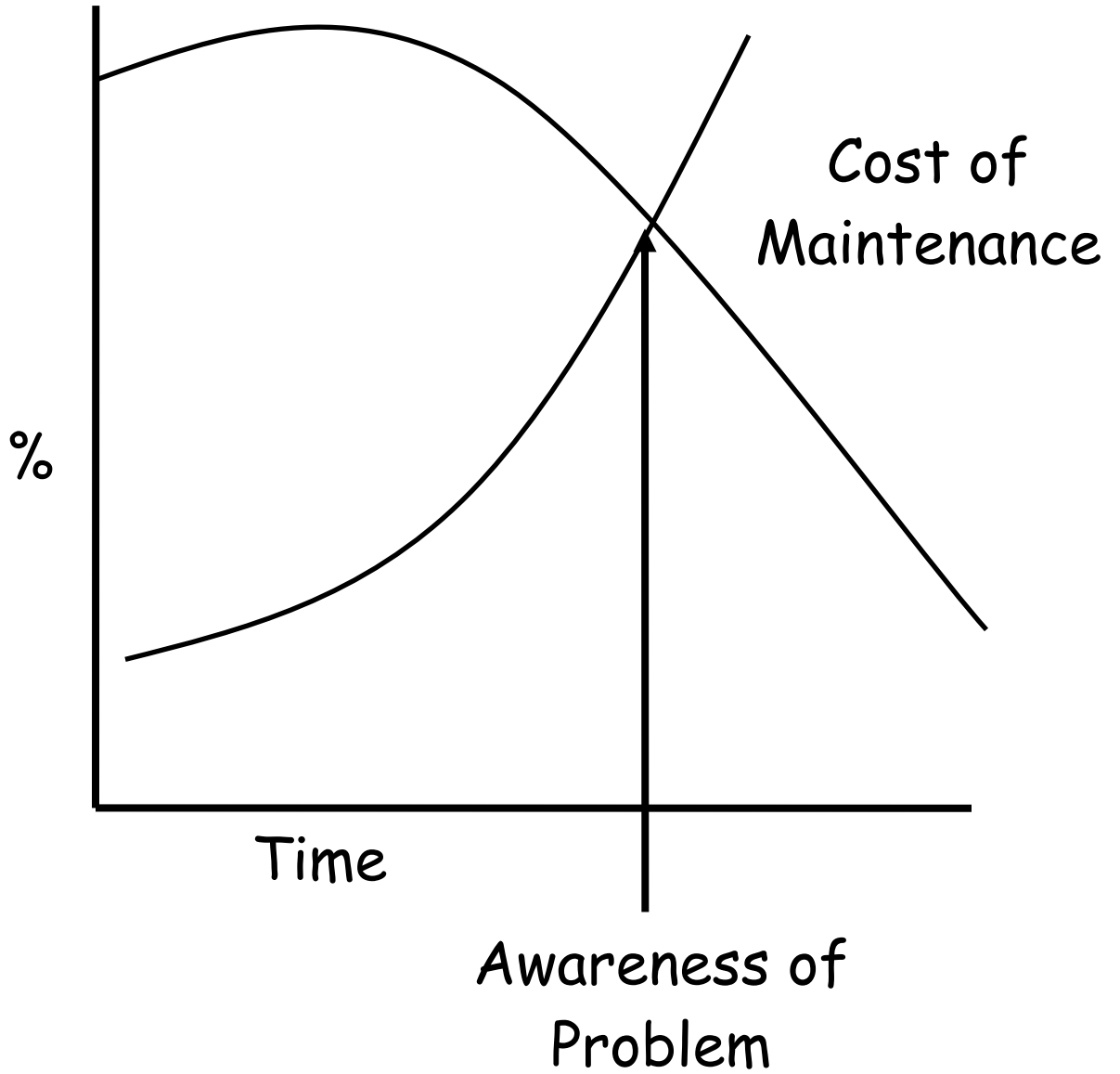
1. Track velocity of development.
2. Track maintenance curve.



The ability to compete is constrained by:

1. Increasing customer discontent with product quality;
2. Excessive cost and time to build new functionality that uses the core;
3. Excessive time to maintain and support existing product as it degrades;
4. Ability of customers to utilize alternatives.

Market Share



Variables to an enterprise

1. The cost of remediating or rebuilding the core product to effectively compete;
2. The rapidity with which the competition is taking market share or the enterprise is losing market share; and,
3. The tolerance of the enterprise's customer base with its increasingly poor quality products

To solve this problem, two beliefs must be unraveled:

1. Customers belief in magic!
2. Developers willingness to lie and cut quality to support this belief.

Scrum's inspect and adapt, coupled with a tight definition of "done" helps face these habits. Scrum's value driven development provides alternatives to cutting quality.



Questions?