Effective software development in the 21st century

The new face of software engineering

Alistair Cockburn
http://Alistair.Cockburn.us

Presented by Mike Dwyer
Principal Agile Coach
Big Visible Solutions
www.bigvisible.com

Developing software consists of people making ideas concrete in an economic context

People inventing and communicating,

Solving a problem *they don't yet understand*
which keeps changing

Creating a solution *they don't yet understand*
which keeps changing

Expressing ideas in languages *they don't understand*
which keep changing

To an interpreter unforgiving of error

Making decisions with limited resources
and every choice has economic consequences

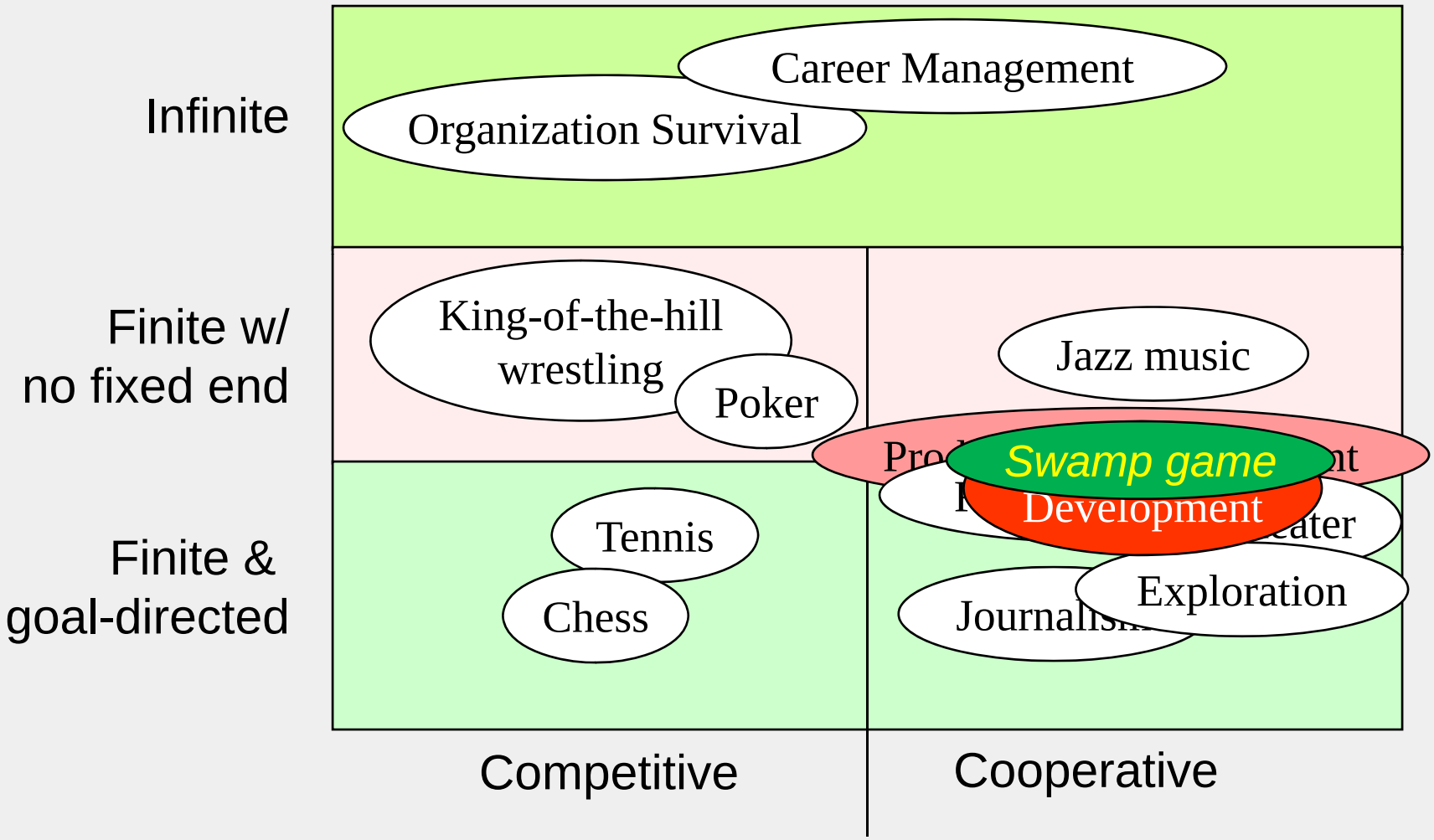
1.

Software development is a

Cooperative Game of

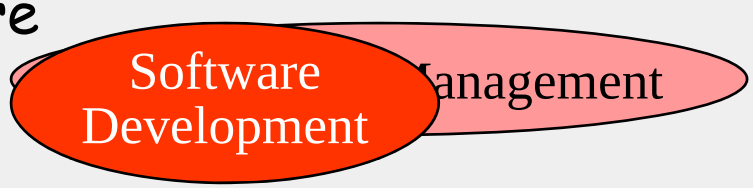
Invention and *Communication*

Games consist of 'strategies' & '*moves*'.
Moves are not right/wrong, but stronger/weaker.



... with two conflicting subgoals:

Deliver this software

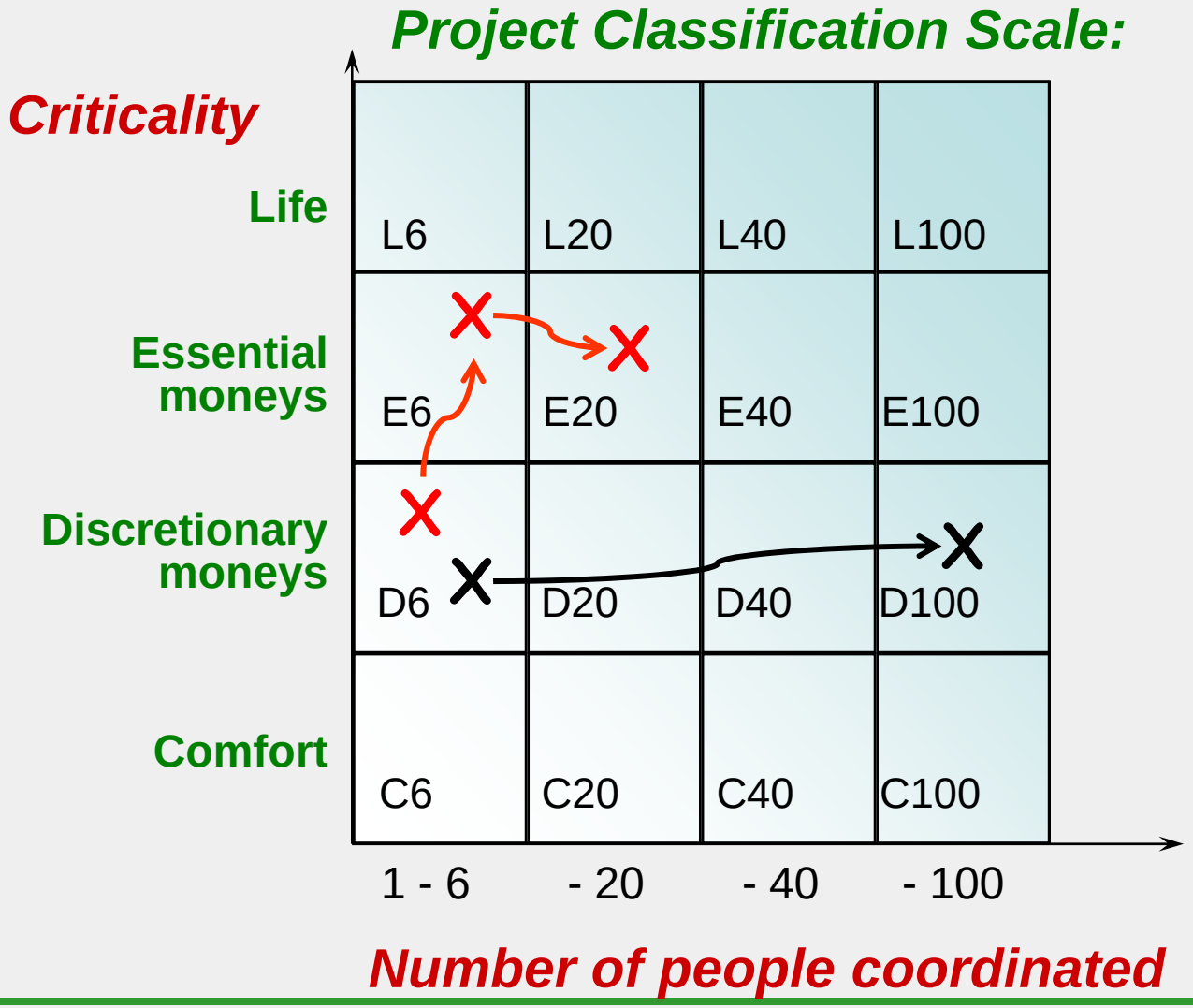


Set up for the next game

The situations *never repeat!*

The **Cooperative Game** idea teaches us about **teamwork** and **strategies** in situations *that never quite repeat.*

Adjust to your situation



Face-to-face is the most effective -
Try *video !*

(Courtesy of Thoughtworks, inc.)

People issues determine a project's speed

Can they easily *detect* something needs attention?

(Good at Looking Around)

Will they *care* enough to do something about it?

(Pride-in-work; Amicability)

Can they *effectively* pass along the information?

(Proximity; face-to-face)

2.

Software development is a *craft*
(7 or more of them!)

Craft teaches us to pay attention to our skills and to our medium

Software development consists of these major crafts:

- 1 Deciding what to build
- 2 Managing *(people and projects)*
- 3 Modeling
- 4 Designing the external view
- 5 Large-scale design *(architecting)*
- 6 Fine-scale design *(programming)*
- 7 Validating the work

PEOPLE Learn Skills in a 3-stage Progression:
Shu / Ha / Ri

Shu : Follow — Learn a technique

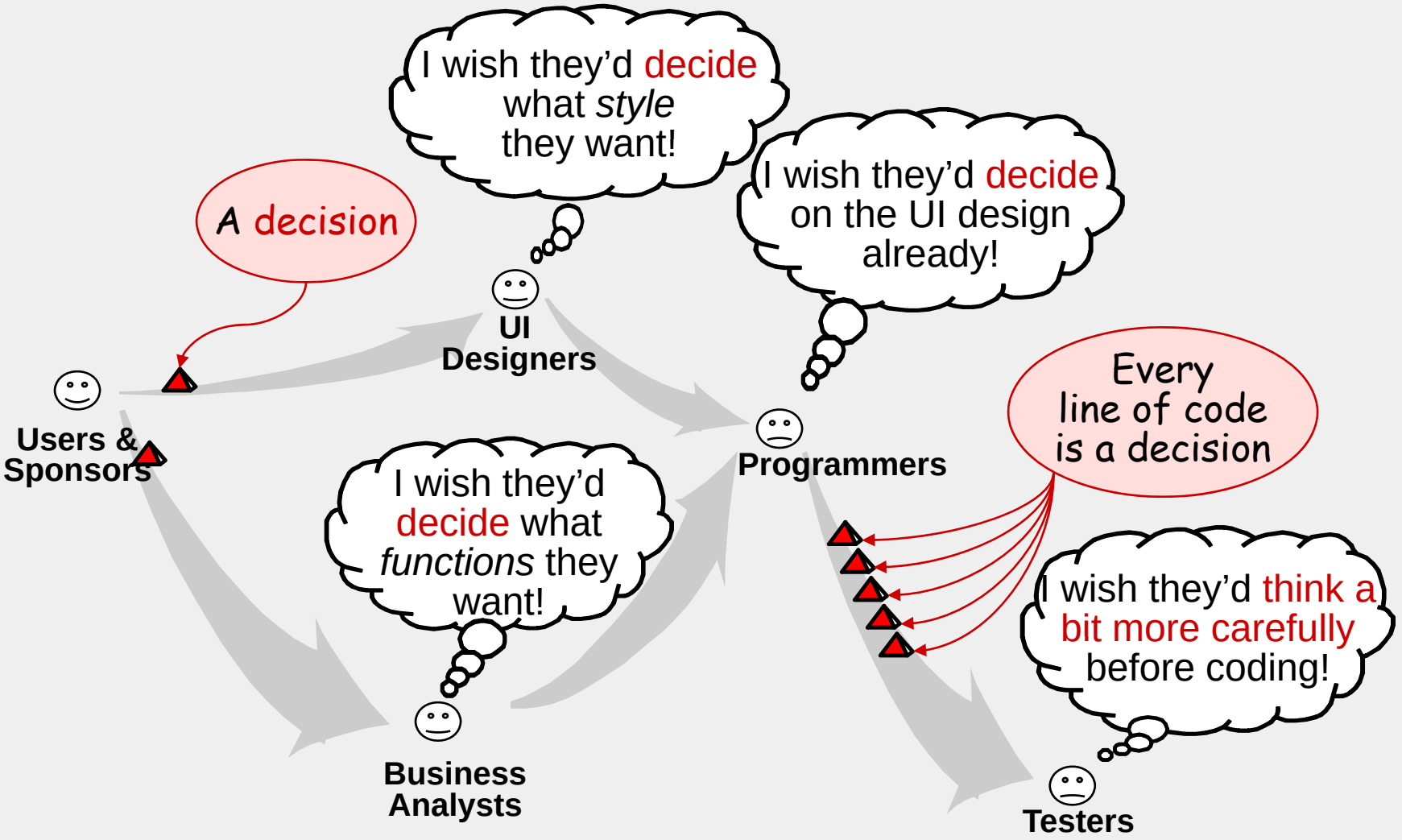
Ha : Break away — Collect techniques

Ri: Fluent — Blend techniques

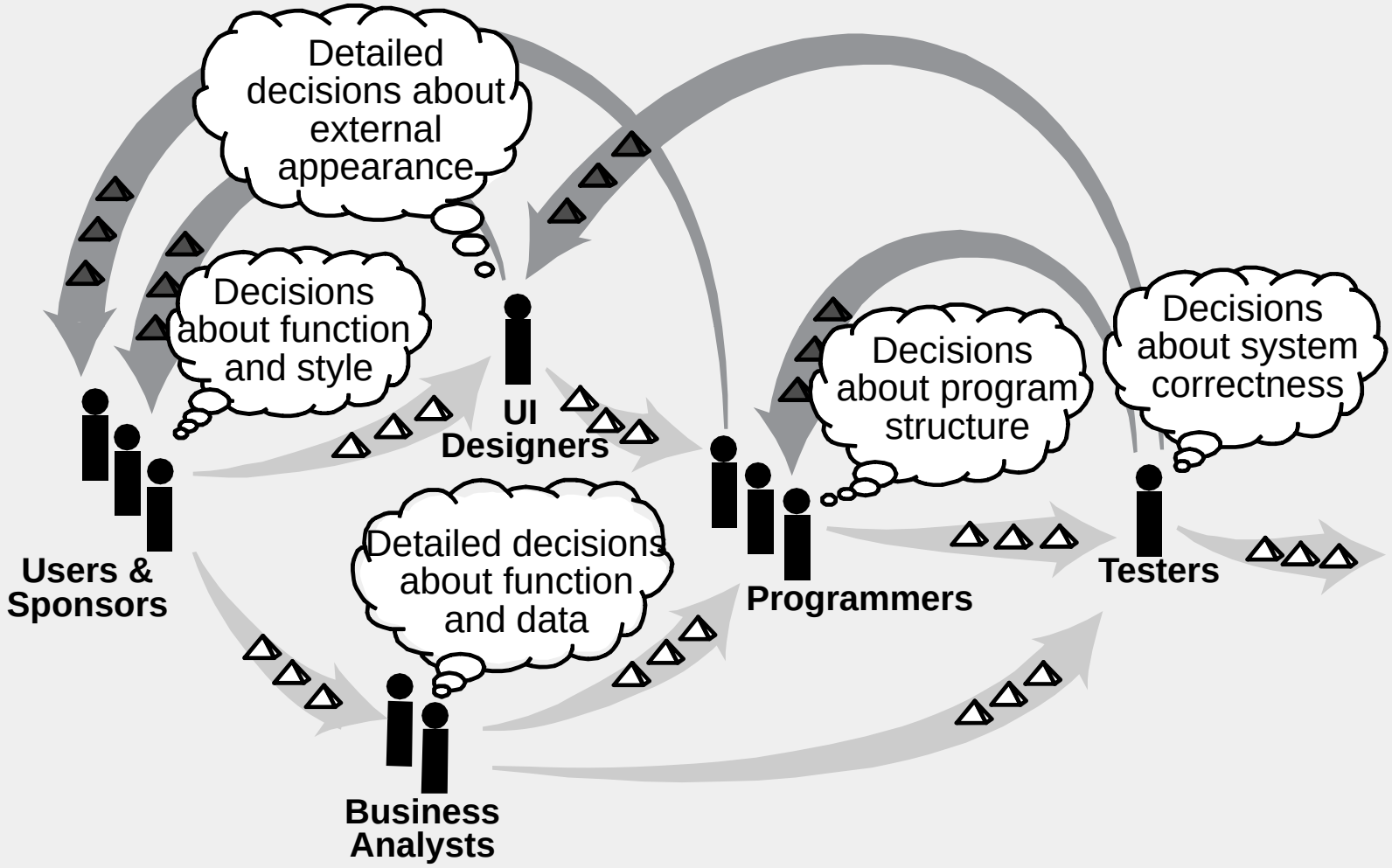
3.

Use *Lean* Processes

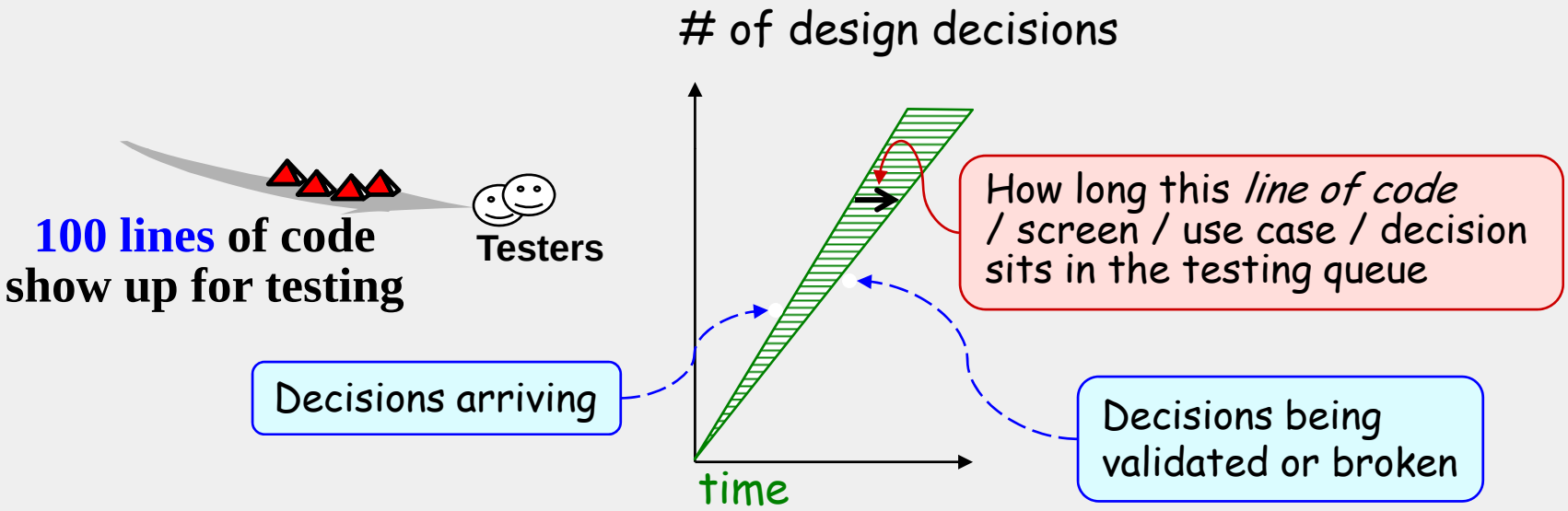
Software development looks like manufacturing if the unit of inventory is the unvalidated decision!



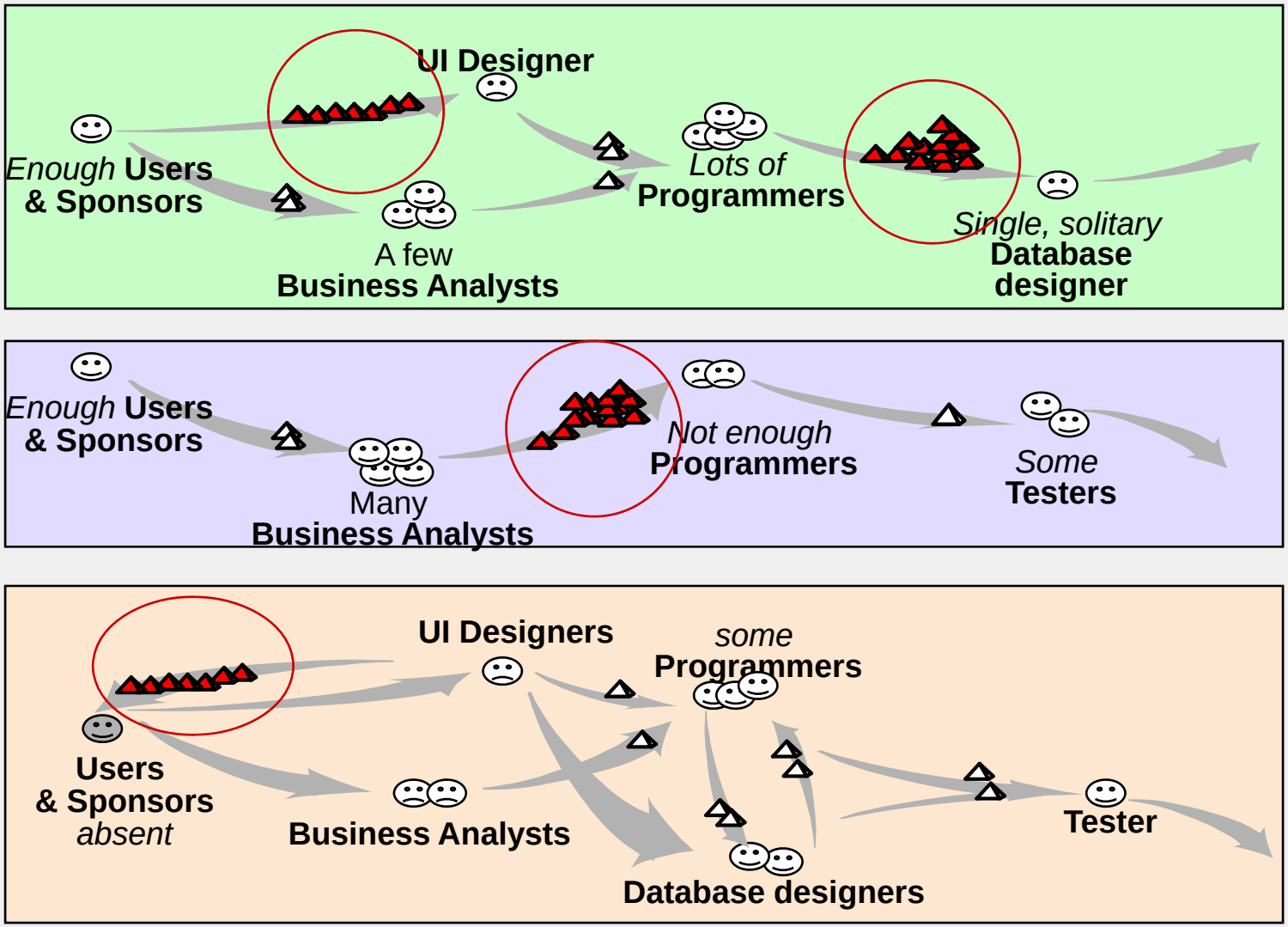
Software development has correction loops



Lean Manufacturing strategies apply directly
aim for continuous flow



Lean Manufacturing strategies apply directly: watch for backed-up queues



4.

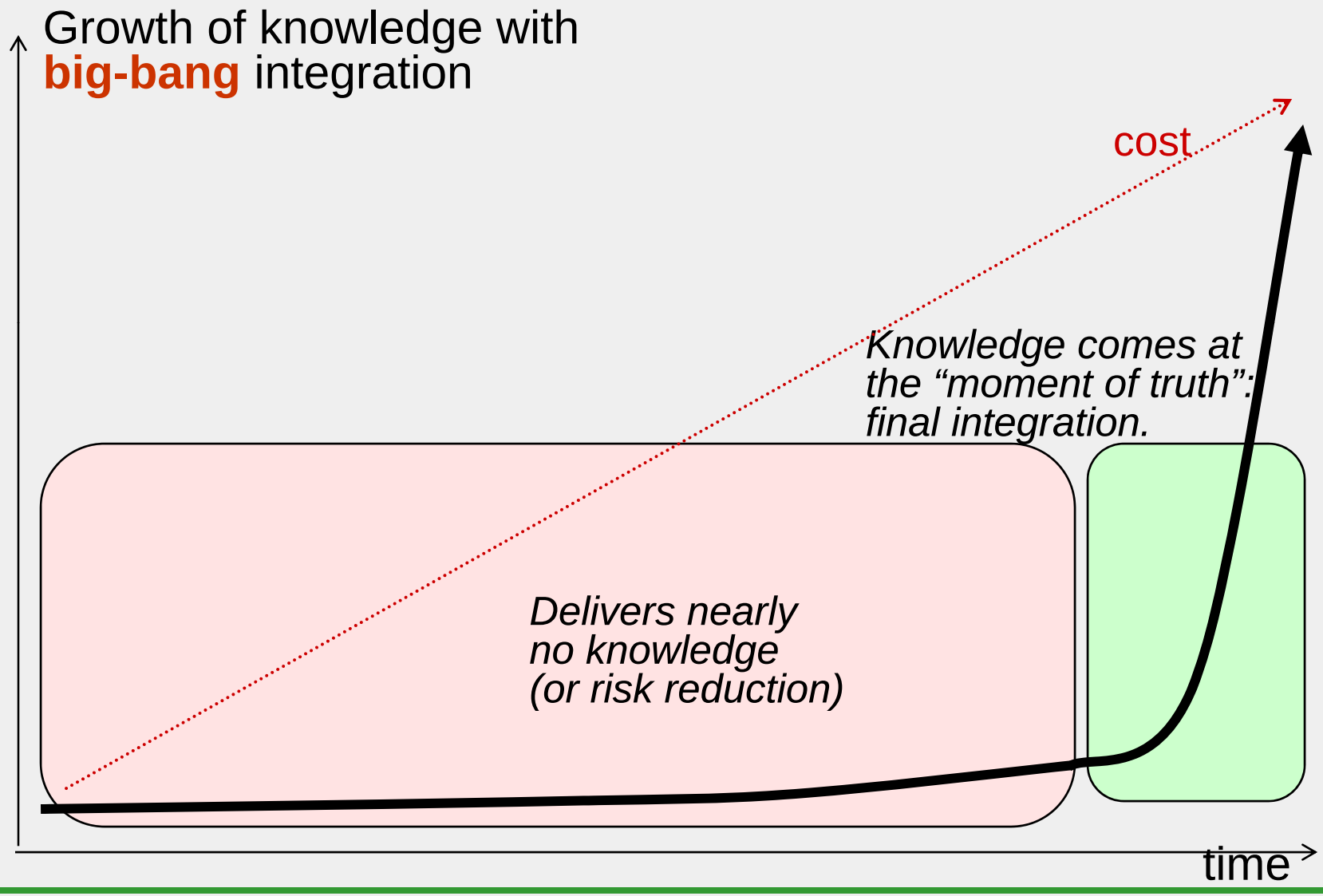
Design is *Knowledge Acquisition*

We know how to build the system *after* we ship it!

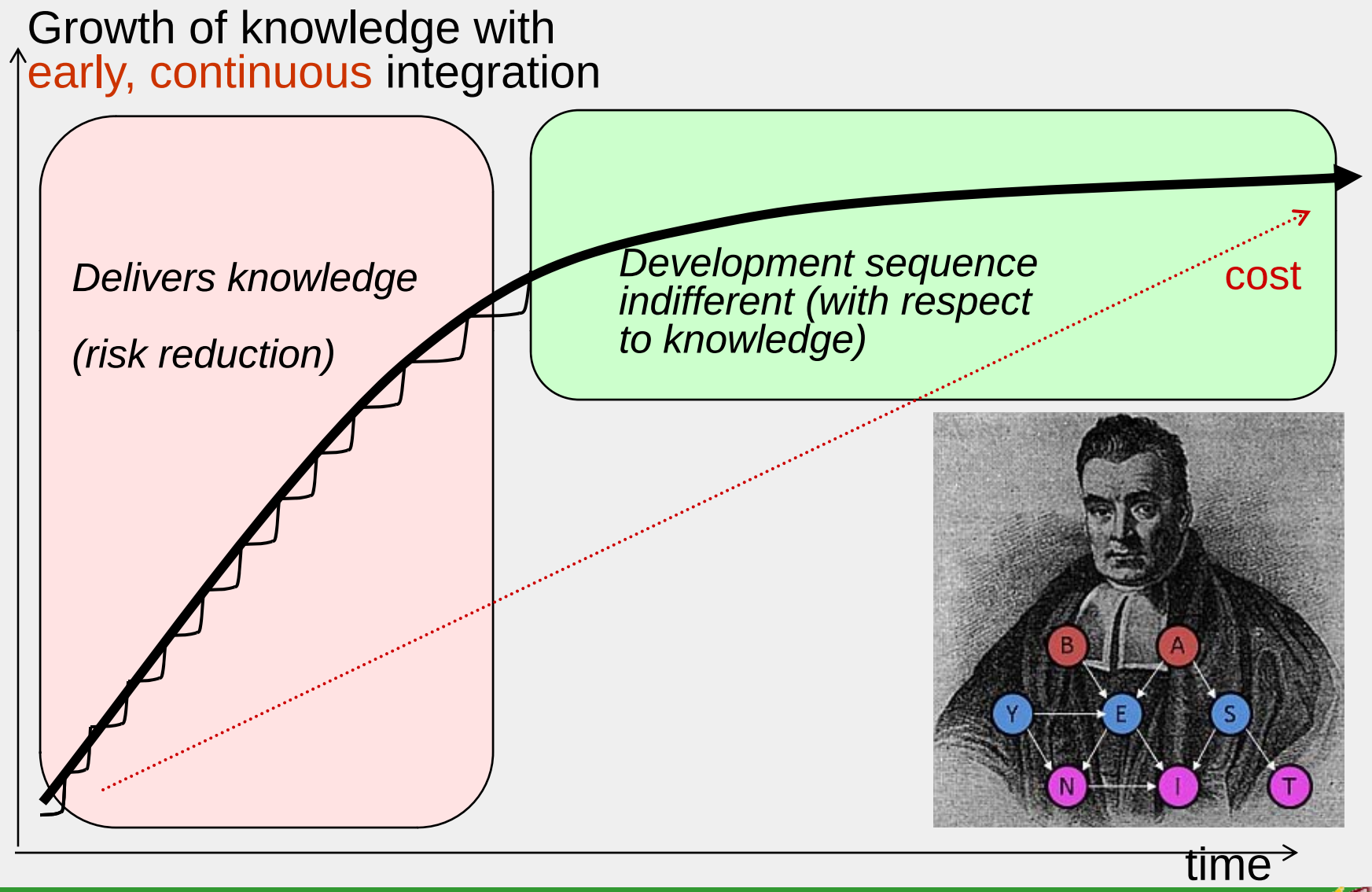
We learn how to build the system
by correcting the mistakes we make!

How can we make use of these depressing facts?

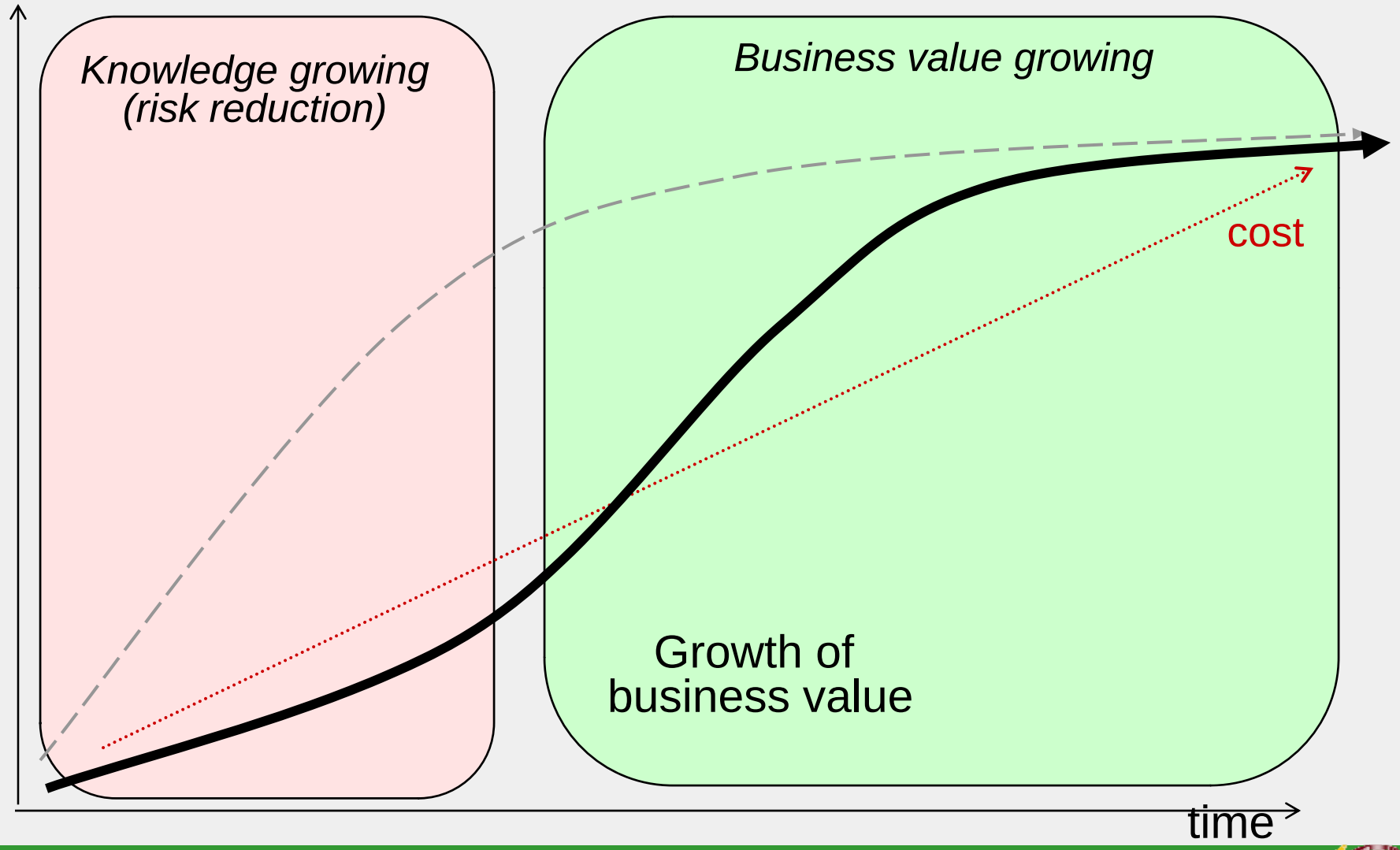
Waterfall is a *late-learning* strategy



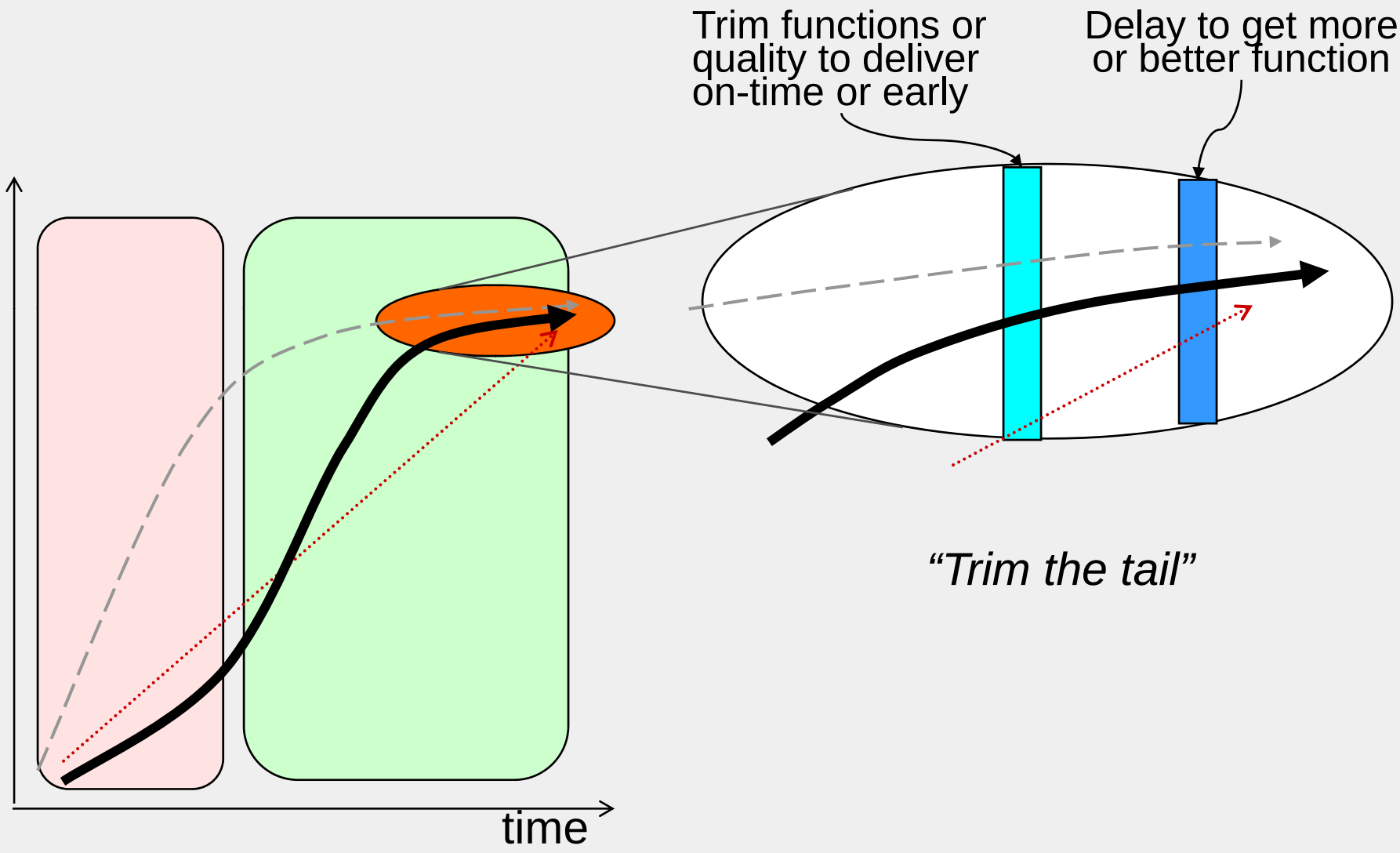
We can pay to *learn* early in the project



Develop for business value once risks are down



Be able to choose to deliver by *value or date*



"Trim the tail"

In the 21st century, software engineering will use *craft, cooperative game & lean principles*

Craft

developing skills in a medium

shu - ha - ri progression

Cooperative game of invention and communication

teamwork, communication, strategies

Lean processes ("unvalidated decisions = inventory")

small queues, cross-trained people, ...

Design as knowledge acquisition

early integration

pay to learn early

trim the tail at the end

Read more on all these topics at http://Alistair.Cockburn.us