Connecting Outcomes to Project Success

Nobody wants to report that the operation was a success but nevertheless the patient died.

Partial Outcomes

- Outcomes like “requirements gathered, “design complete,” and “code complete,” are partial indicators of project success
- So are outcomes like “June sprint successful,” “unit tests passing,” and “velocity restored to previous average”

It doesn’t matter what methodology you use: you construct a successful project out of successful partial outcomes

What good are evidence-based practices?

- Provide confidence in our ability to achieve successful partial outcomes
- Provide an objective basis for communicating status and making decisions

But… evidence-based practices cannot select the best partial outcomes by themselves. They are ‘necessary but not sufficient’.

"The world is divided into two kinds of people: those who bifurcate and those who don't."

How to select partial outcomes

Divide all possible partial outcomes into two piles: those that reflect the necessary ends from those that reflect your preferred means.

Ends are better than means (although means are better than nothing at all).

Ends and Means

- Customer value
- Correctness
- Done-ness of final software

- Velocity
- Standardization
- Done-ness of infrastructure

While there is value in the items on the right, we value the items on the left more.

Done-ness vs. Done-ness

- Done-ness of the final software is some property of the finished software you can inspect.
- Done-ness of infrastructure is completion of some intermediate product that does not evidence any completion of the final software.

Back to Ends and Means

Delivery of value is the ultimate end. Isn't that how you measure project success?

- Means seem necessary but never sufficient for project success.
- Ends are sufficient in and of themselves.

If you can't deliver value incrementally, done-ness of the final software is a decent proxy.

Evidence-based software development practices

- It's all about confidence and managing uncertainty.
- Objective input + function = probability of well defined outcome.
- Need to then take action to turn evidence into a practice
- Prefer outcomes that reflect 'ends' over outcomes that reflect 'means'

"Friends applaud, the comedy is over."

Ludwig van Beethoven (1770-1827)

Supercharging Evidence-Based Software Development

Turning a tactic into a strategy,
or,
How I Learned to
Stop Worrying and Love XP

"The best strategic plan is useless if it cannot be executed tactically."

Field-Marshall Erwin Rommel
quoted in "Bottom-Up Marketing"

Ends, Means, and Outcomes (The Strategic Approach)

1. Define the desired end result. Often this is one monolithic outcome.
2. Select means for achieving the outcome based on minimizing time and cost.
3. Organize the team's practices around outcomes that reflect the means.

Bottom-Up Thinking

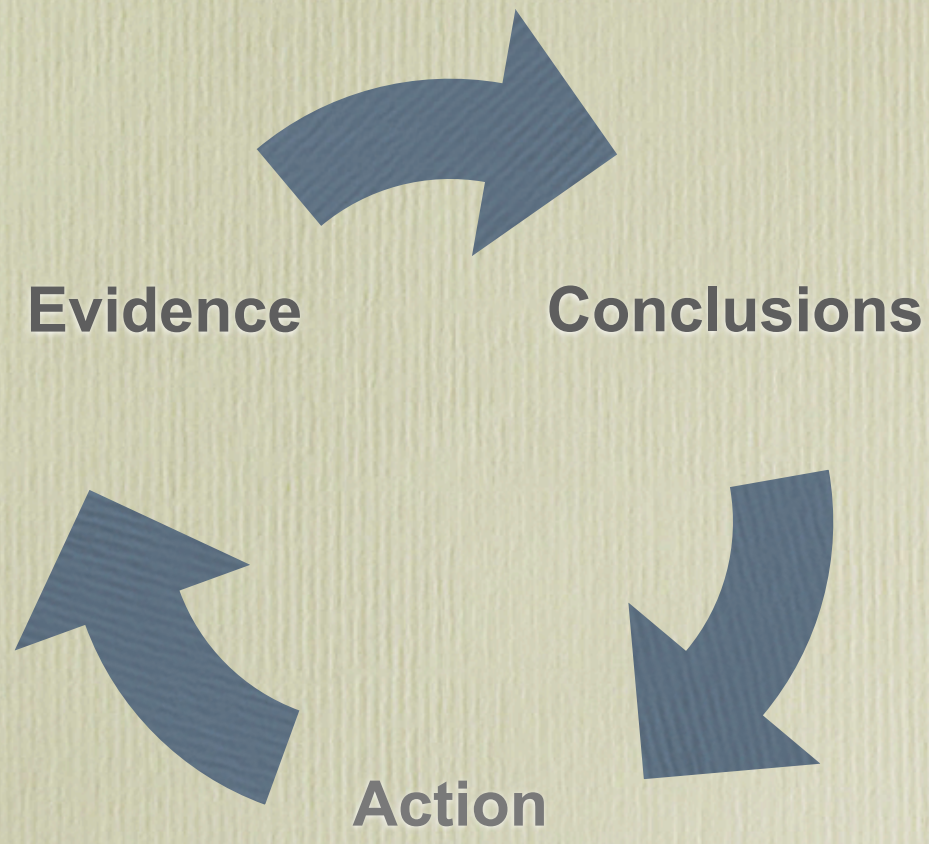
- The tactic is the practice that produces results
- The strategy is the organization of the development team's activities to produce the maximum tactical pressure

Our tactic:
Evidence-based practices

1. Generate/gather evidence
2. Form conclusions or judgments about the potential outcomes
3. Take actions based on evidence to maximize the likelihood of desirable outcomes
4. Create a feedback loop

Result: confidence and the ability to manage uncertainty.

The evidence feedback loop



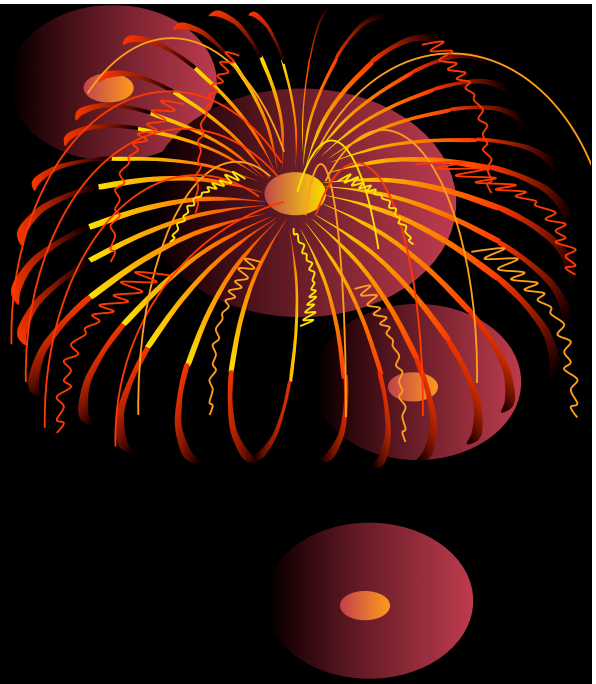
Bottom-Up Evidence-Based Software Development

- Our tactic is the evidence-based feedback loop.
- Our strategy is organizing the team's activities around maximizing the quantity and quality of evidence-based feedback loops.

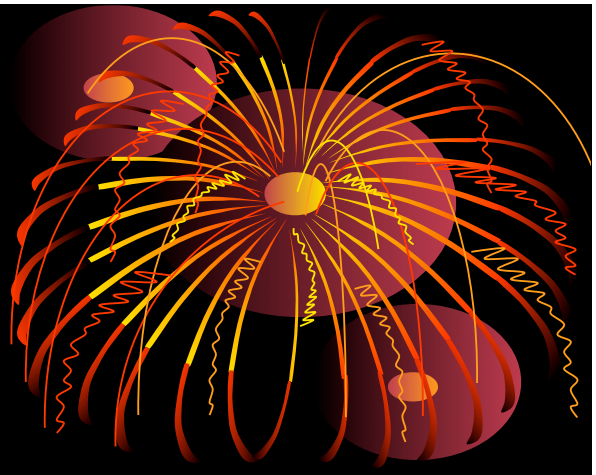
Ends, Means, and Outcomes (The Bottom-Up Approach)

1. Define the desired end result in such a way that you can divide it into many partial 'end' outcomes.
2. Select means for achieving the outcome based on maximizing the quantity and quality of feedback loops.
3. Organize the team's practices around the feedback loop.

Final Exam

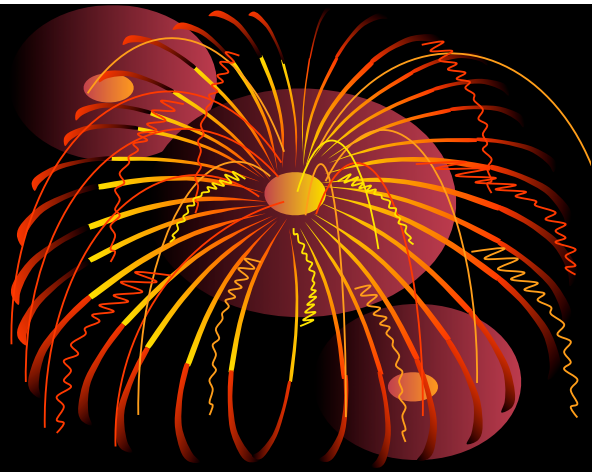


Which of the following practices organize the team's activities around maximizing the evidence feedback loop?

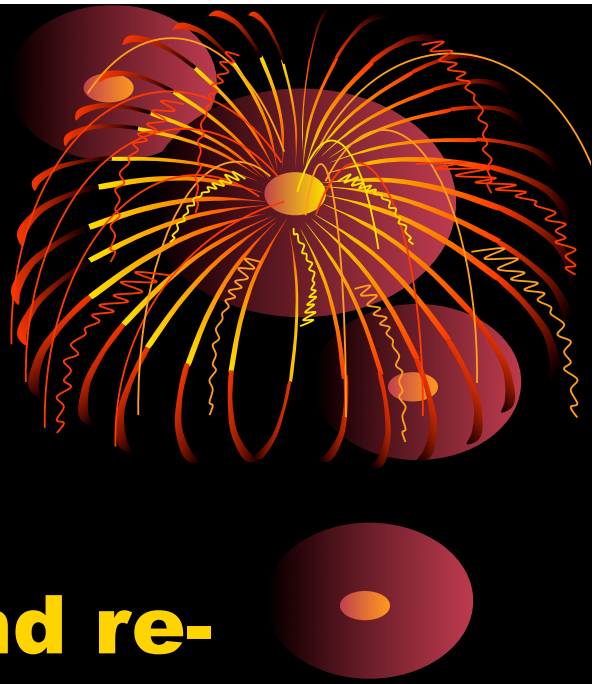


"The development team needs to release iterative versions of the system to the customers often."

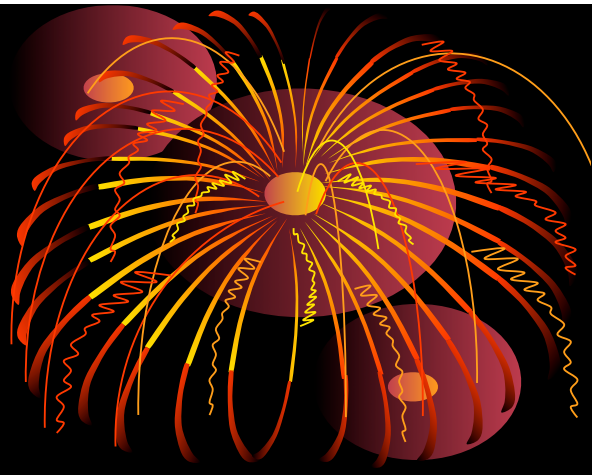
“Move people around to avoid serious knowledge loss and coding bottle necks.”



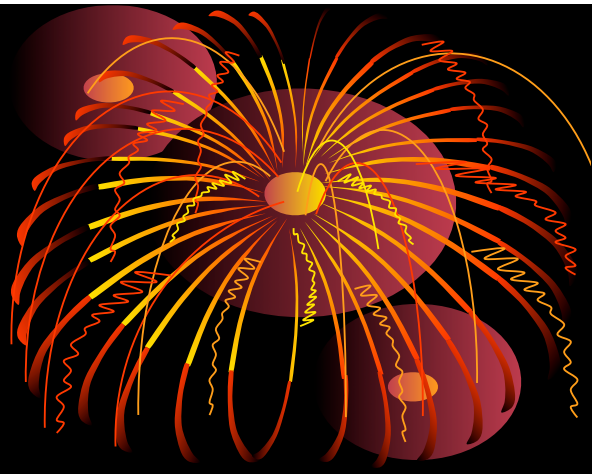
"The release planning meeting is used to discover small units of functionality that make good business sense and can be released into the customer's environment early in the project."



"Use a release planning meeting to re-estimate and re-negotiate the release plan if your project velocity changes dramatically for more than one iteration."



"It is often very difficult to unit test some software systems. These systems are typically built code first and testing second, often by a different team entirely. By creating tests first your design will be influenced by a desire to test everything of value to your customer. Your design will reflect this by being easier to test."



"Refactor mercilessly to keep the design simple as you go and to avoid needless clutter and complexity. Keep your code clean and concise so it is easier to understand, modify, and extend."

http://weblog.raganwald.com