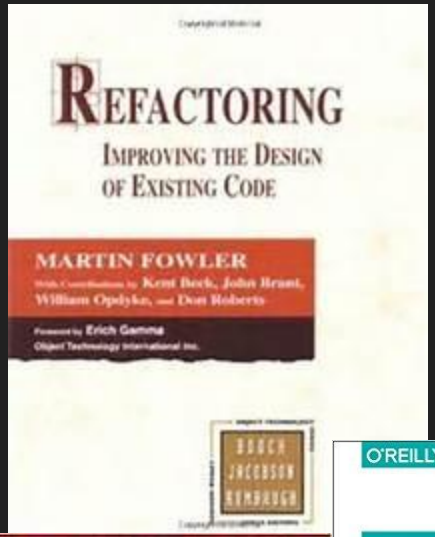
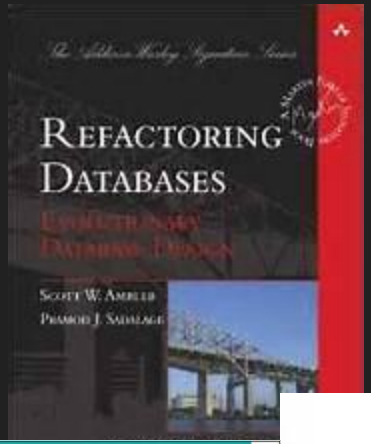
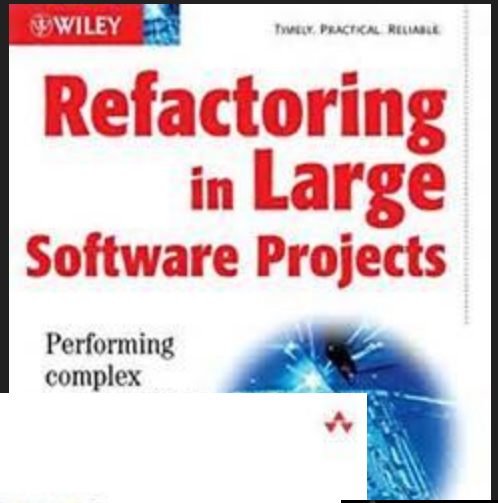
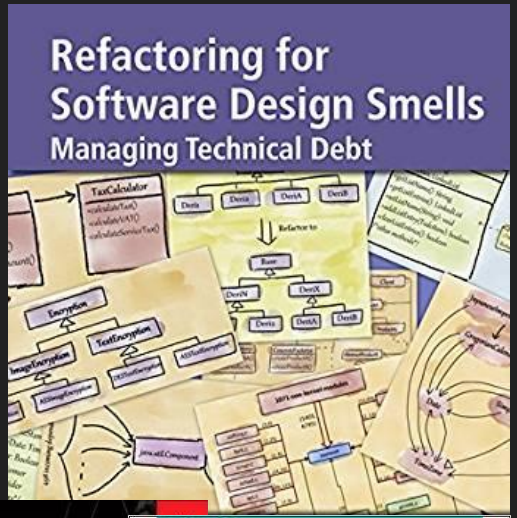
Making the Case for Refactoring to Non-Technical Managers

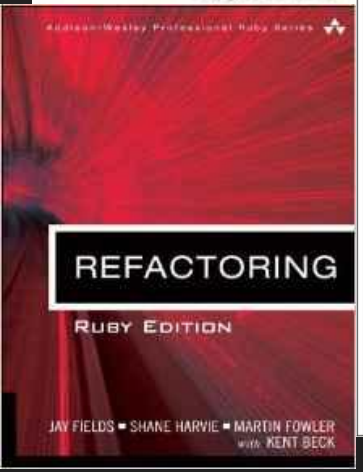
Adam Juda
TapRun, LLC

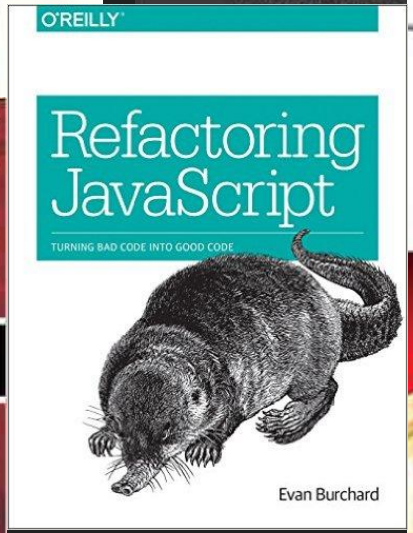

REFACTORING
IMPROVING THE DESIGN OF EXISTING CODE
MARTIN FOWLER
With Contributions by Kent Beck, John Brant, William Opdyke, and Don Roberts
Foreword by Erich Gamma
Object Technology International, Inc.
BOOCH
JACOBSON
RUMBAUGH

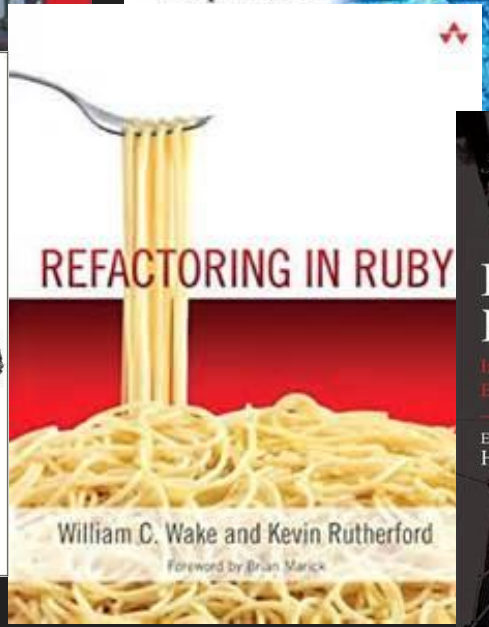

The Addison Wesley Signature Series
REFACTORING DATABASES
SCOTT W. AMBLER
PRAMOD J. SADALAGE

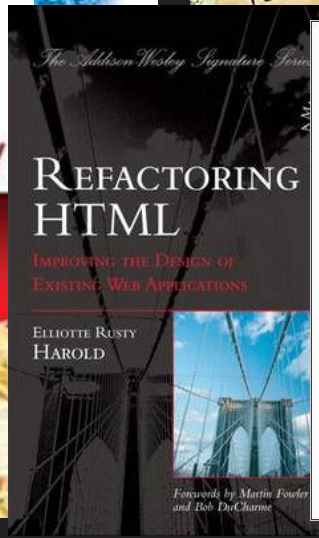

WILEY
TIMELY. PRACTICAL. RELIABLE.
Refactoring in Large Software Projects
Performing complex

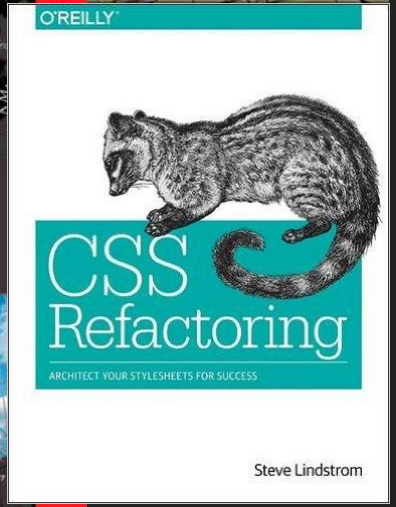

Refactoring for Software Design Smells
Managing Technical Debt


Addison-Wesley Professional Ruby Series
REFACTORING
RUBY EDITION
JAY FIELDS • SHANE HARVIE • MARTIN FOWLER
with KENT BECK

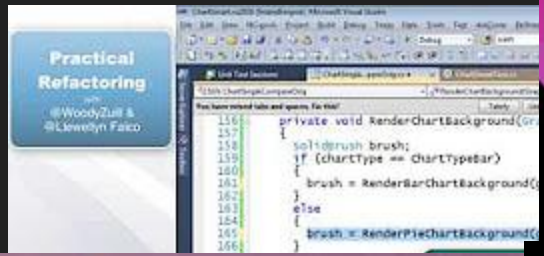

O'REILLY
Refactoring JavaScript
TURNING BAD CODE INTO GOOD CODE
Evan Burchard

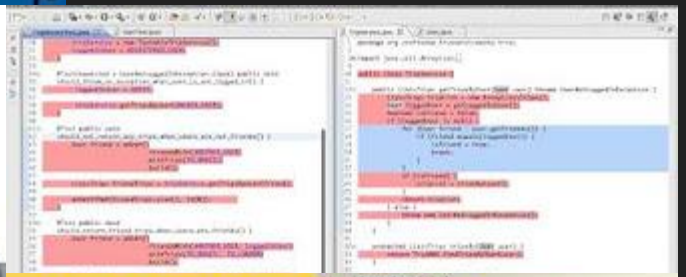

REFACTORING IN RUBY
William C. Wake and Kevin Rutherford
Foreword by Brian Marick


The Addison Wesley Signature Series
REFACTORING HTML
IMPROVING THE DESIGN OF EXISTING WEB APPLICATIONS
ELLIOTTE RUSTY HAROLD
Forewords by Martin Fowler and Bob DuCharme


O'REILLY
CSS Refactoring
ARCHITECT YOUR STYLESHEETS FOR SUCCESS
Steve Lindstrom

CODE REFACTORING





Who am I?

Adam Juda, MSE, MBA, PMP

- Programmer
- Project Manager
- Consultant (TapRun, LLC)

Outline

- What's technical debt?
- How to ask
- Whom to ask
- When to ask
- Q&A

What is technical debt?

Bad software

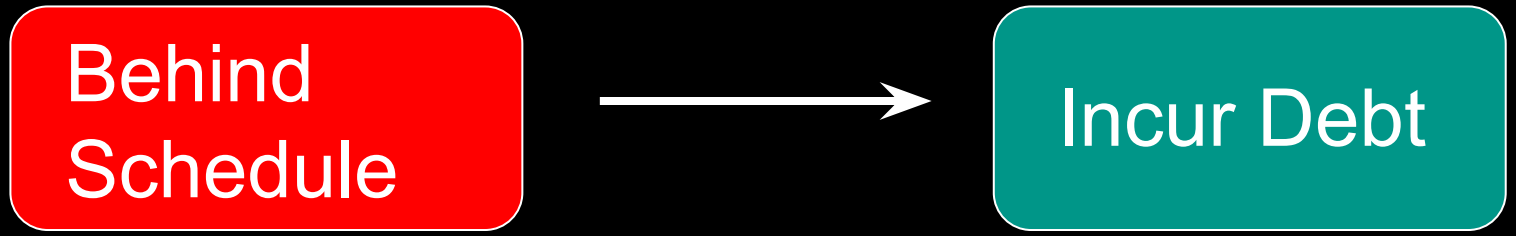
How is it bad?

No coherent design	Bad documentation	Slow build process
Broken tests	Huge classes	High cyclomatic complexity
Huge method signatures	Duplicated code	Not following style guide
Large methods	Tight coupling	Low cohesion
Low maintainability	Low reliability	Low scalability

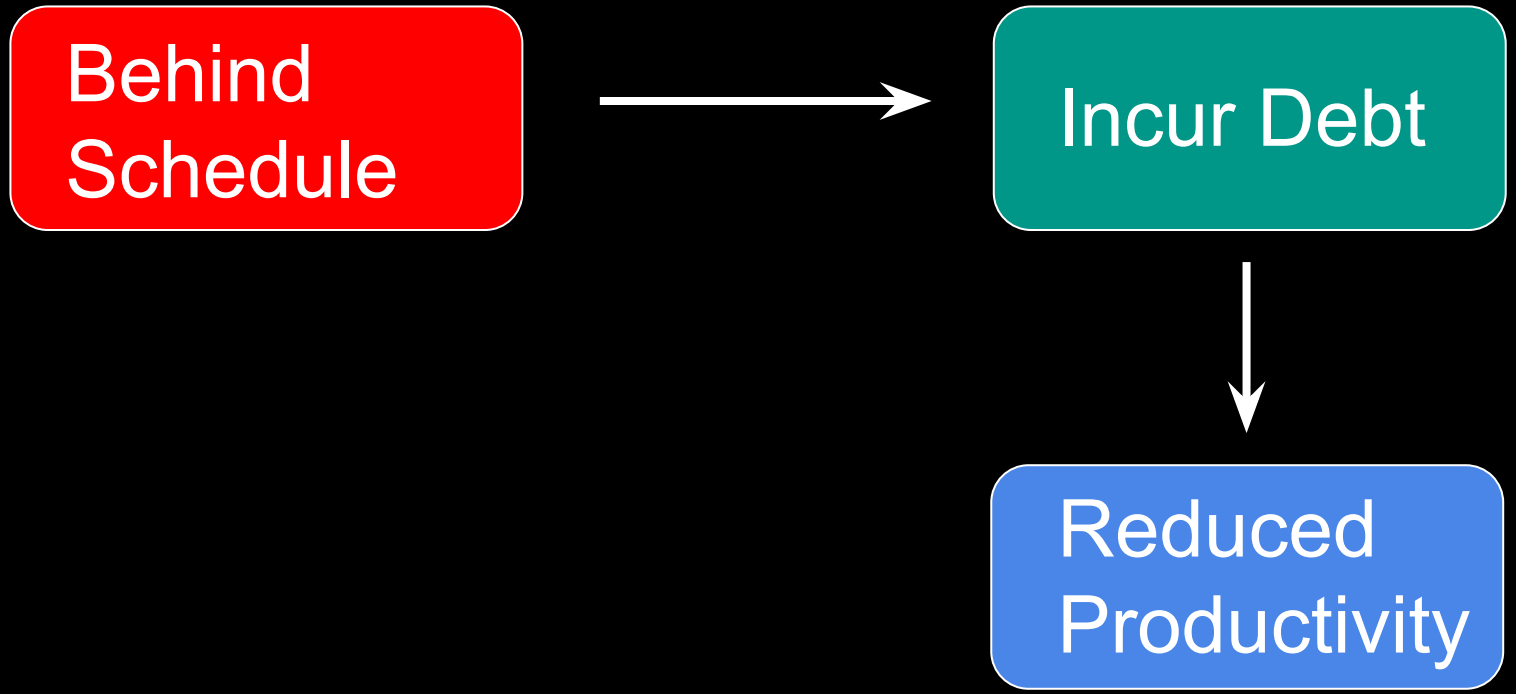
Traditional view of technical debt

Behind Schedule

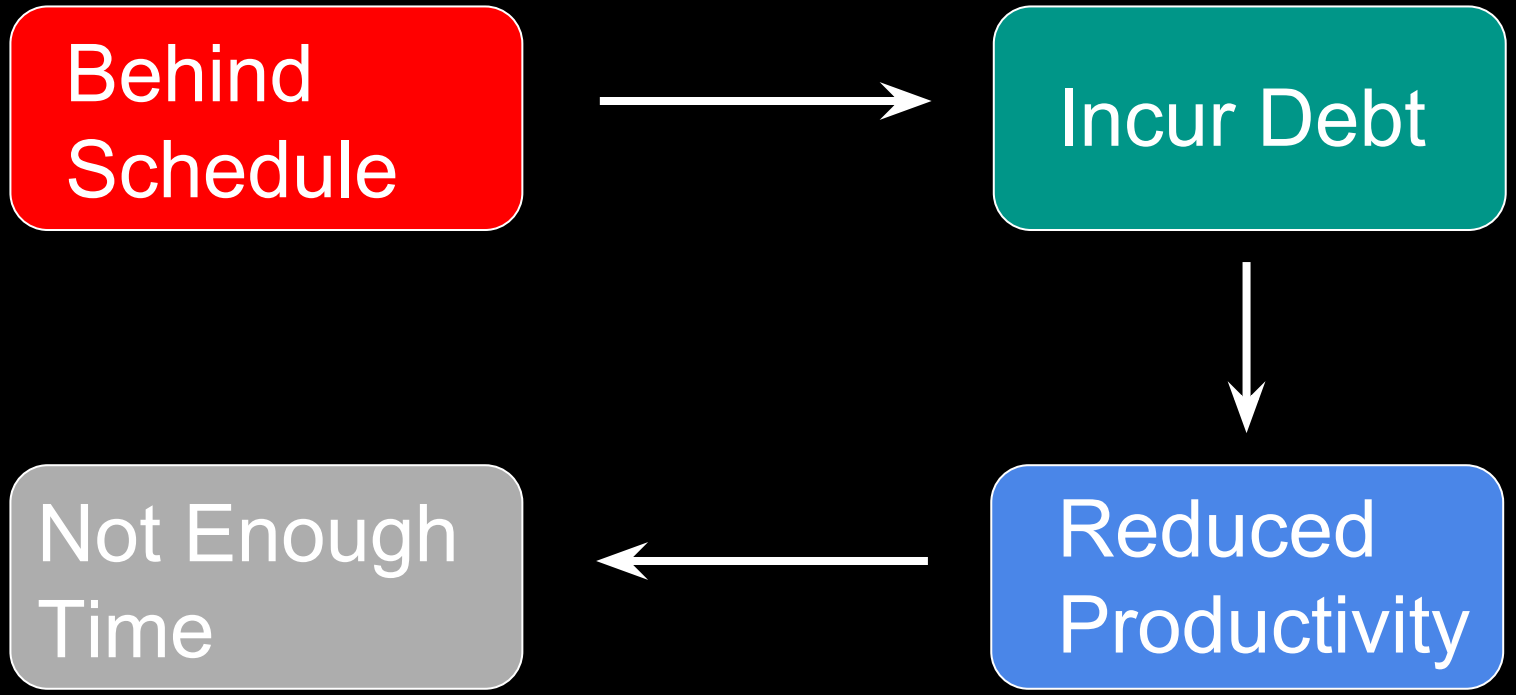
Traditional view of technical debt



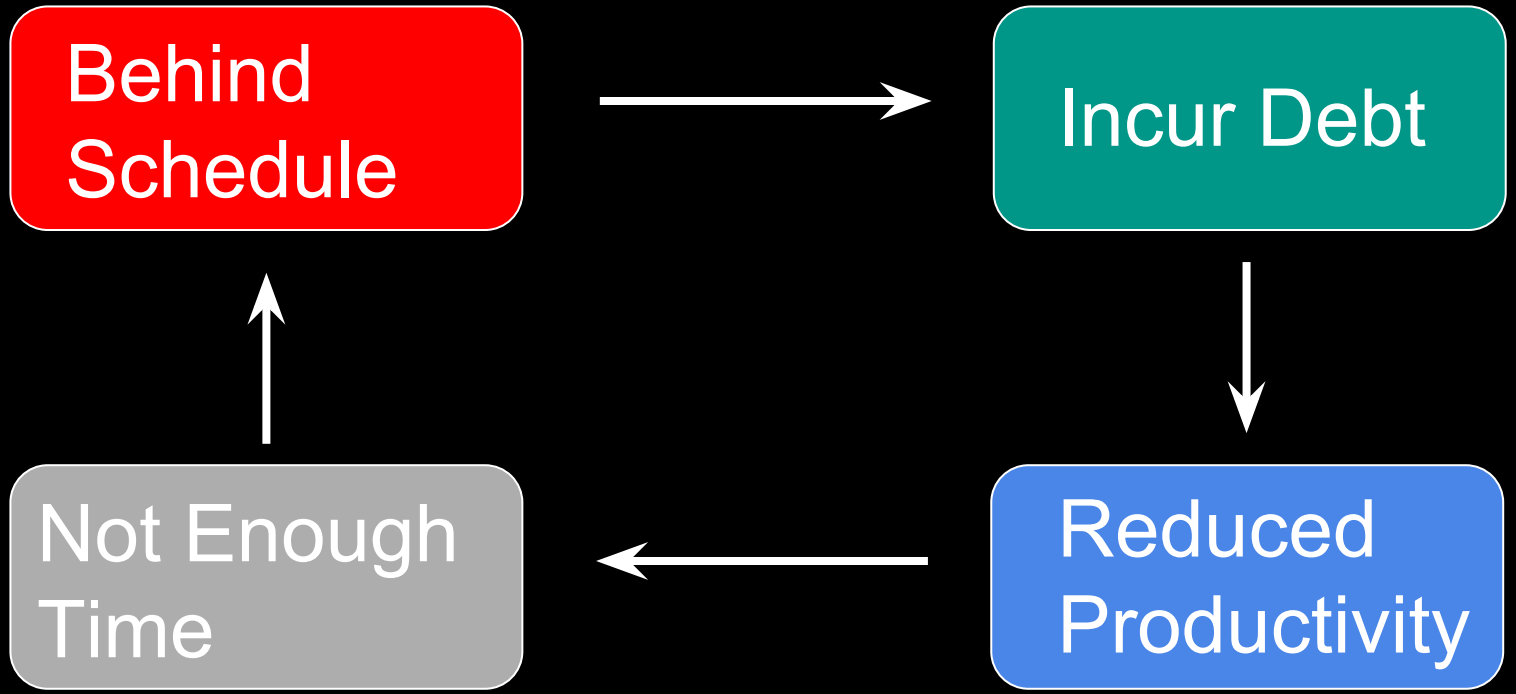
Traditional view of technical debt



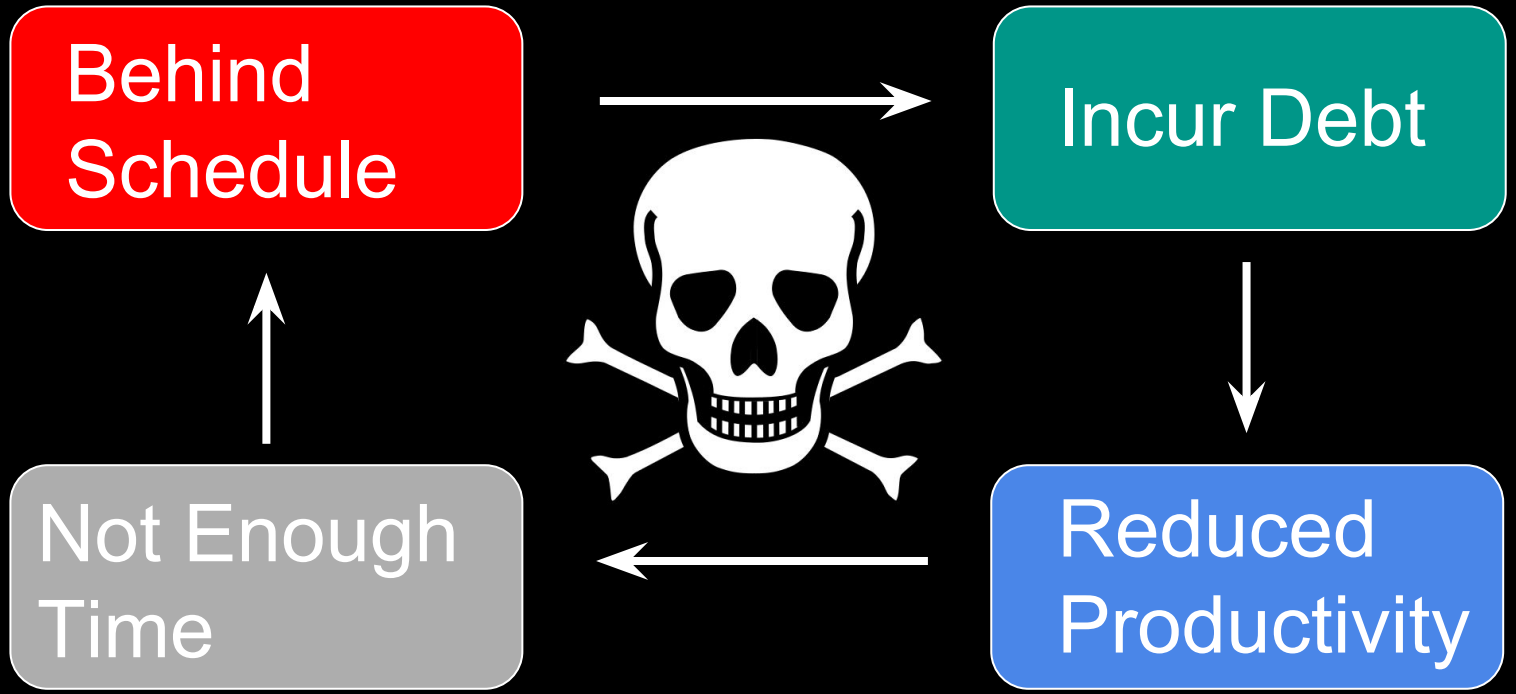
Traditional view of technical debt



Traditional view of technical debt



Traditional view of technical debt



Seven common approaches

Approach #0: I wanna (aka the mountain)

No.

Approach #1: Complaining

- It's painful to work with.
- We gotta, it's gross!
- The code smells...


Stop
whining.

Approach #2: Begging

Please! I'll work extra hard for you.
Let me fix the code!


A favor?
Hmm...

Approach #3: Making analogies

- A car that hasn't had an oil change
- A fishtank that hasn't been filtered
- A litter box that hasn't been emptied

An employee who wants to get fired

Approach #4: Quantitative analysis

The code has a cyclomatic complexity of 40!


...
INITECH

Húsið er á eldinn!

Your house is on fire!

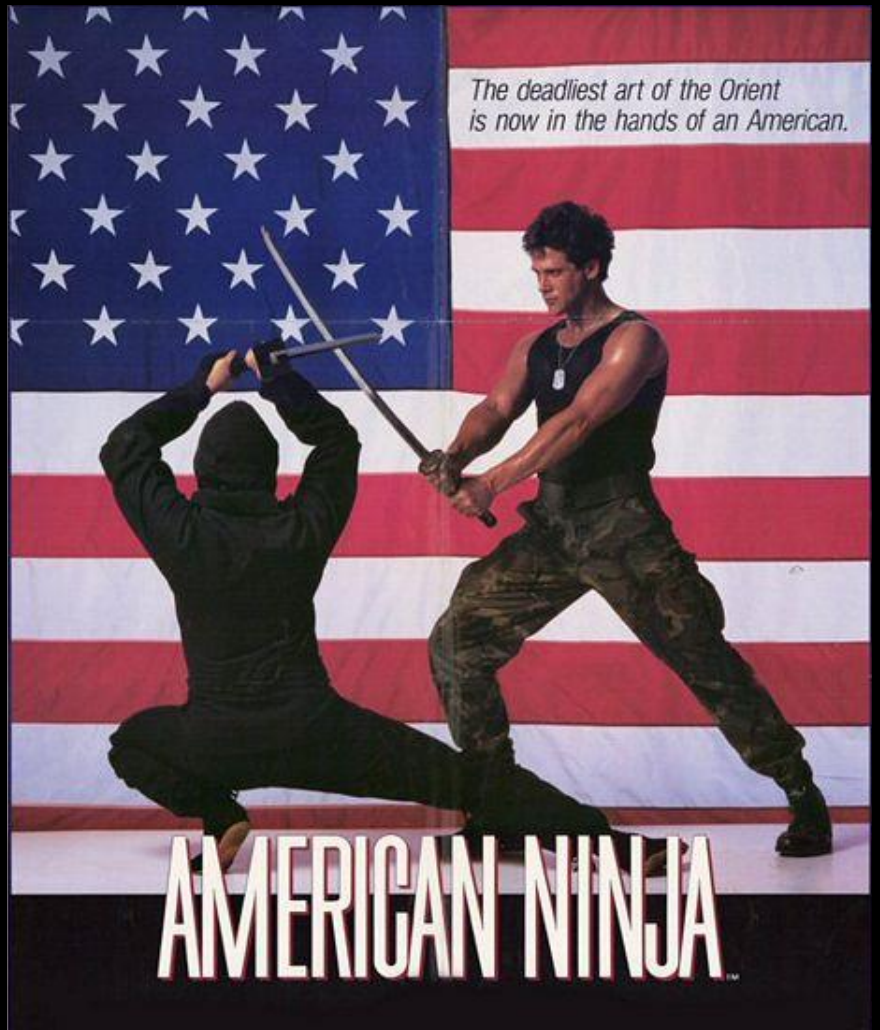
Approach #5: Hypotheticals

- Morale might go down
- Our software might crash
- We might be late


Everything might be fine.

Approach #6: Be a ninja!

AKA "Be a Professional"



What would you say you do here?

Technical debt: Another definition

A trade-off made in order to meet a business objective

"If you are involved in a game, everything ends up being a set of trade-offs."

-- Gabe Newell

(co-founder of Valve)

```
$field1 = '/^def*re+o?/';

preg_match($field1, $field2, $field3,
PREG_OFFSET_CAPTURE, 3);

$field3 == $non_field + $field_field ? 7 : 3;

$ro->sn($field3 . $field1).extt(4, $pep);
```

Start

Anna Davies
Will do, I'll send them to you as soon as I get back from your holi...

Wendy Teo
RE: Dinner tonight? Sounds good. How about the new place with the back patio?
Mail 8

Soccer practice
Rec center
5:30 PM – 8:00 PM
8
Friday

Internet Explorer

Store


Bing

skype™

Maps

SkyDrive

Games

Camera


Desktop

Messaging

71°
Seattle, WA
Mostly sunny 72° / 53°
Weather


Hot air balloon festival takes off in the valley this weekend

Music

Video

"Business staff thinks we can load up technical debt because they never truly see the consequences."

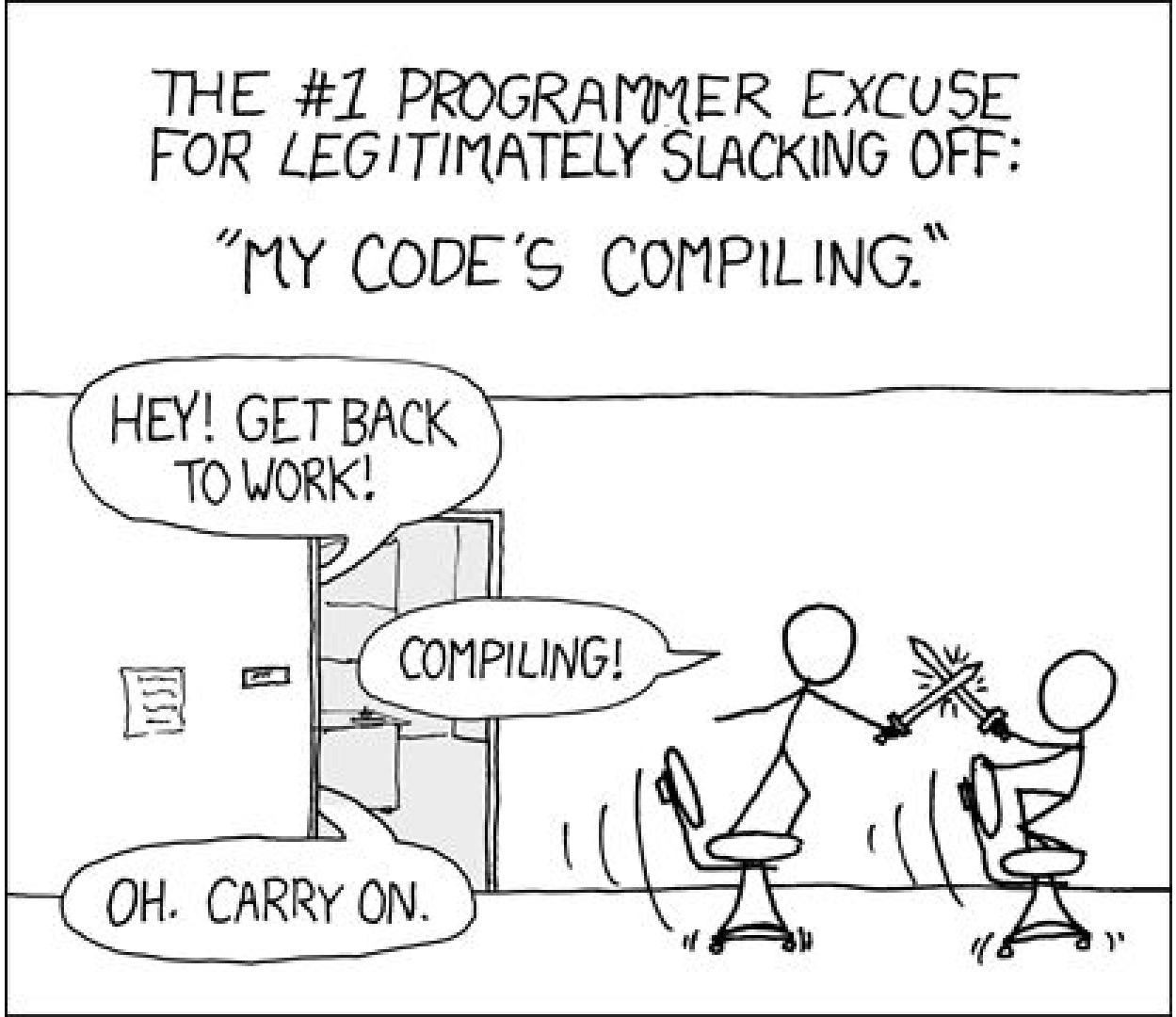
-- Steve McConnell

"But those consequences are there… they are just never expressed in a way that the business staff can engage with."

-- Steve McConnell


$
Refactor
$$$

But I'm just a programmer!



http://xkcd.com/303/

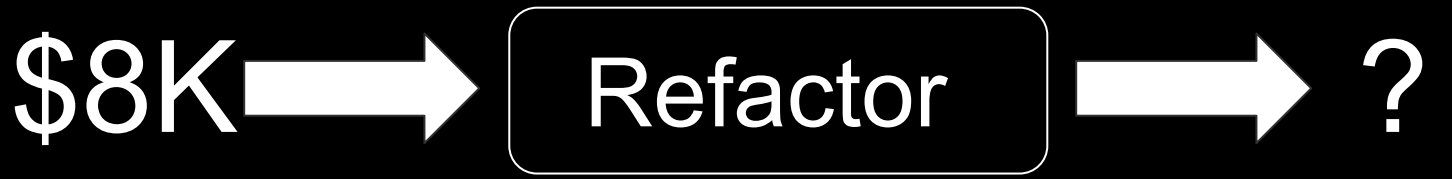
Step #1: Gather information

- Labor - $200/hr
- Build process - 2 hrs
- Builds / year - 100
- Time to rewrite: 40 hrs

Step #2: Calculate the costs

Cost to fix = $200 * 40 hours

= $8,000


$8K
Refactor
?

Step #3: Calculate the benefits

Savings = $200 * 2 hrs * 100 builds

= $40,000


$8K
Refactor
$40K

Dollarize when possible

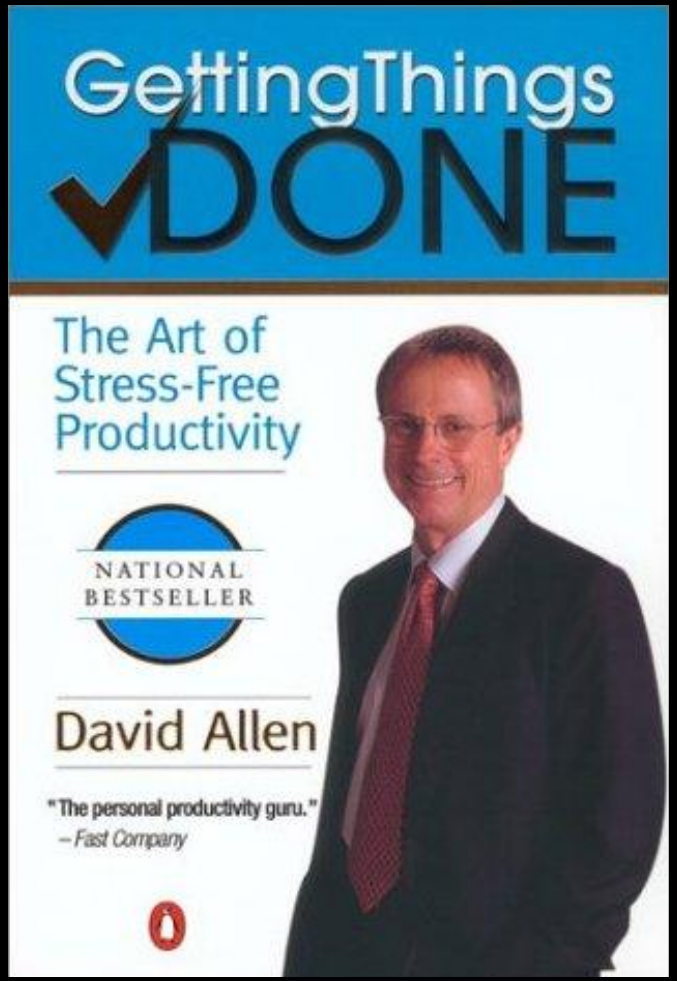
- Labor
- Server time
- Risk of rework
- Employee morale?


Debt
The Scrum ^ Backlog

The case for debt

- Debt Name: _____
- Date: _____
- Short Term Savings: _____
- Long Term Cost: _____
- Cost to Repair: _____
- Notes: _____

There's an unexpected benefit...



Under 3 minutes?

Just do it.

Whom should you ask?

Whom should you ask?

- Controls resources
- Will receive substantial value

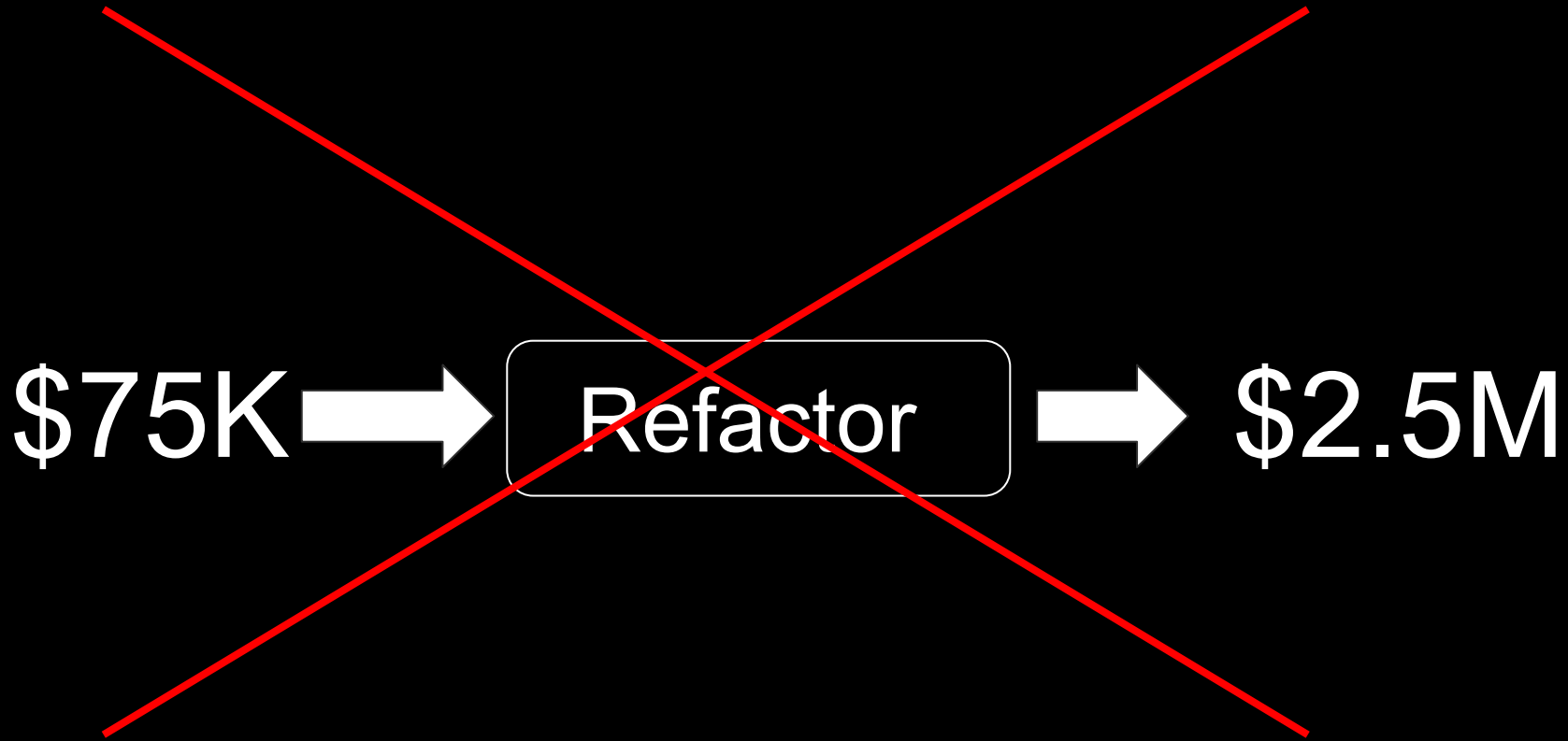
Let's do some more math...

- $25,000 extra to each project
- 100 projects use it
- $75,000 to refactor

Cost to fix: $75,000

Savings: $25,000 * 100 == $2.5 million


$75K
Refactor
$2.5M


$75K
Refactor
$2.5M

- $25,000 extra to each project
- 100 projects use it
- $75,000 to refactor


$75K
Refactor
$25K

Project managers vs product owners...

- Productivity?
- Morale?
- Maintenance costs?
- User satisfaction?
- Long-term strategy?
- Long-term risk?

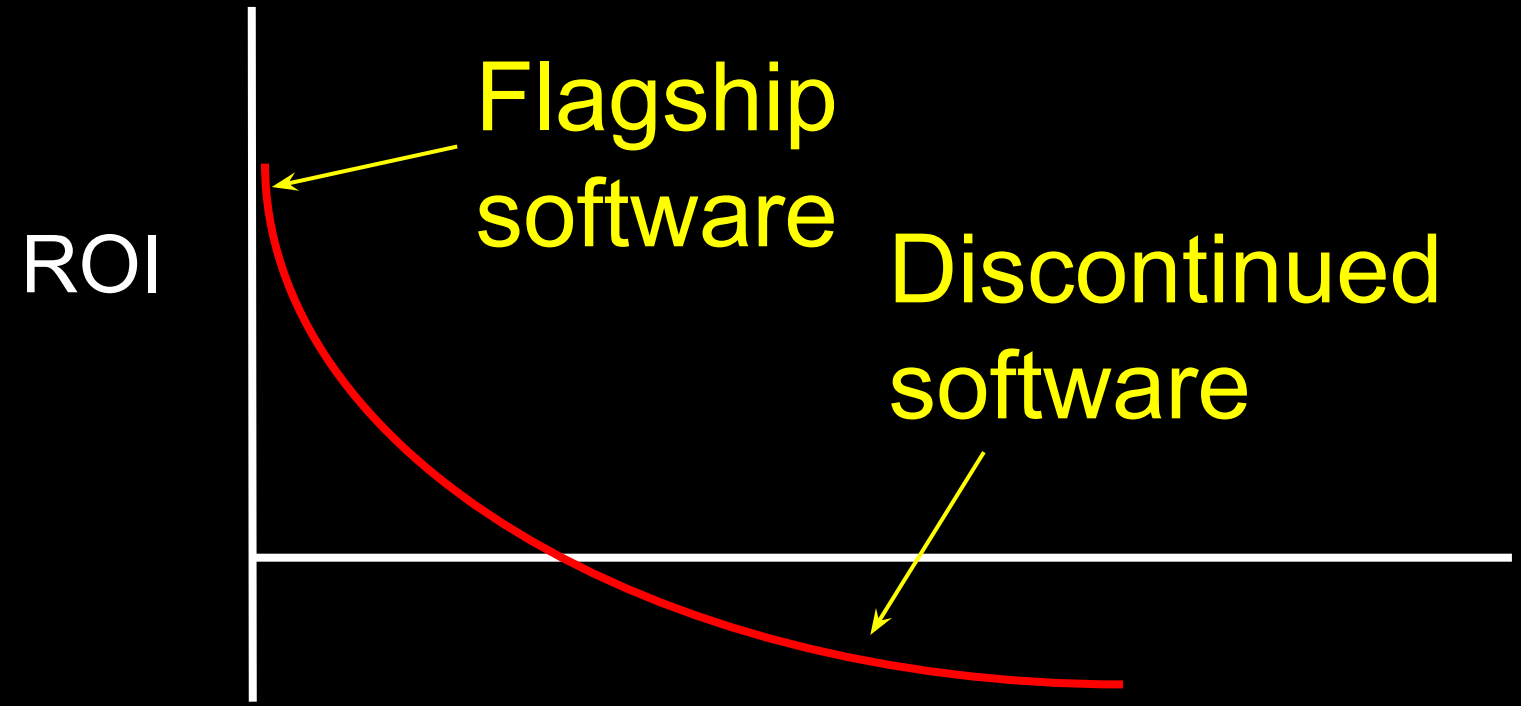
When should you ask?

What others say...

- At the beginning of a project
- At the end of a project
- When technical debt is highest

What I say

- When payback time is smallest
- When payback will be obvious
- When ROI is largest

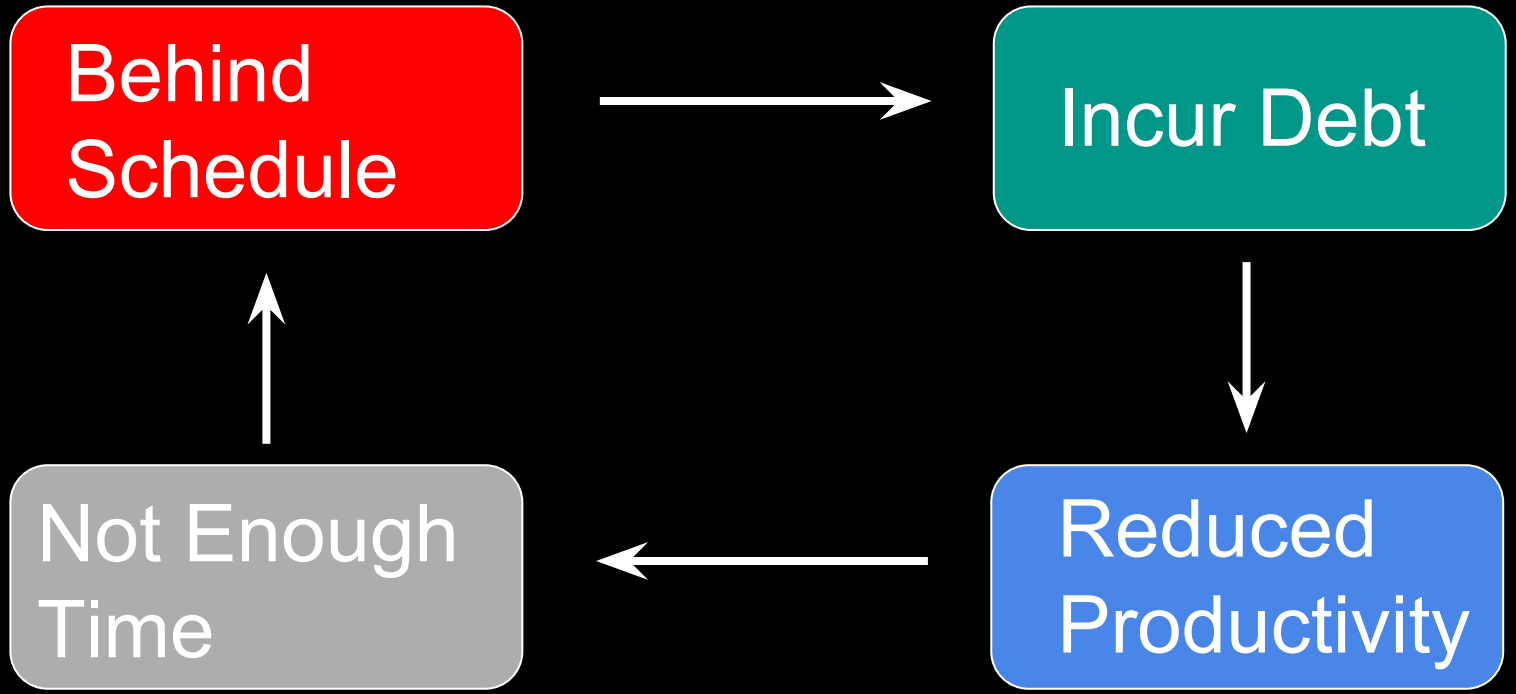

ROI
Flagship software
Discontinued software

Microsoft®
Windows98

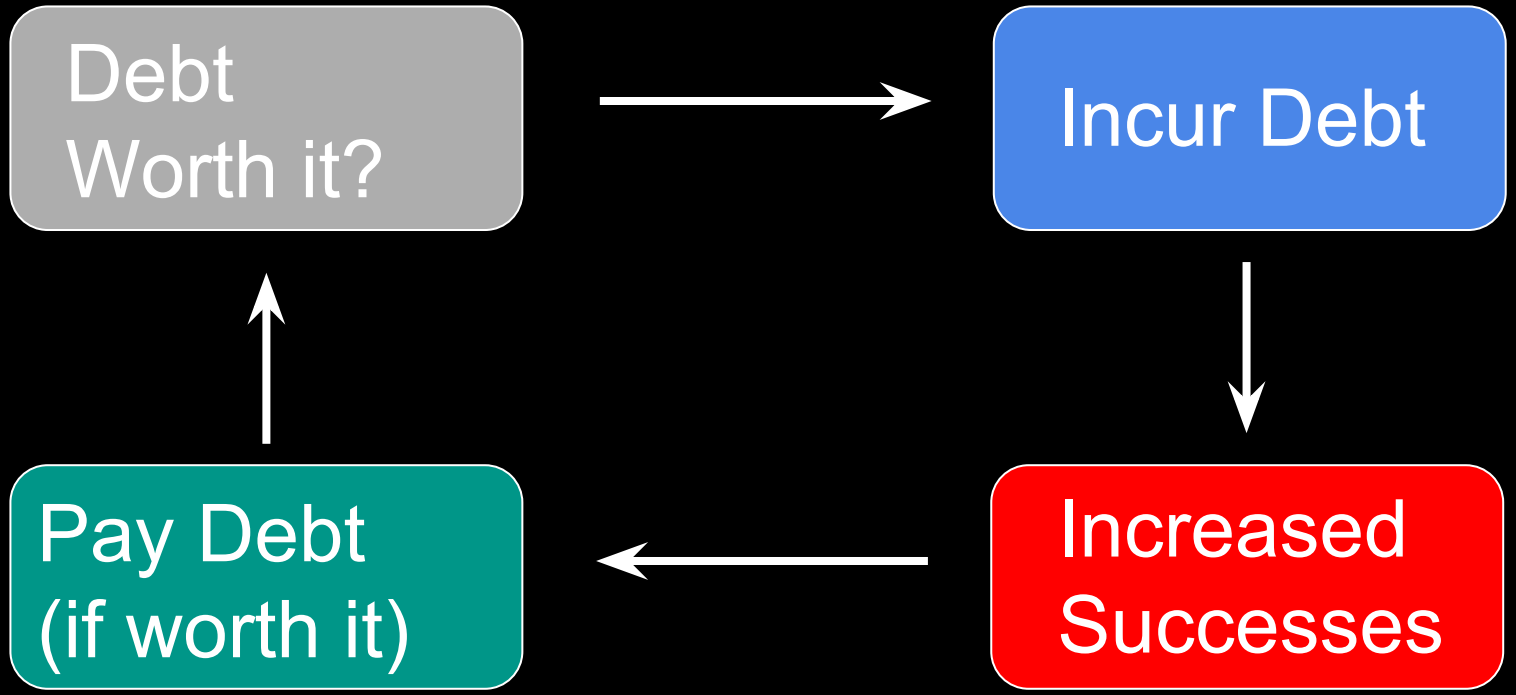
For some values of $1:

$1 != $1

Traditional view of technical debt



Improved view of technical debt



This won't work well if...

- Lack of trust / honesty
- Arbitrary constraints
- Unwillingness to improve
- Incentive not to improve

The five levels of debt monetization

Level 0: Establish open communication

Level 1: Communicate business goals

Level 2: Present options

Level 3: Dollarize options

Level 4: Groom & optimize

Summary

- Technical debt isn't always bad
- Find the right section of code
- Find the right person
- Speak to his incentives
- Speak at the right time

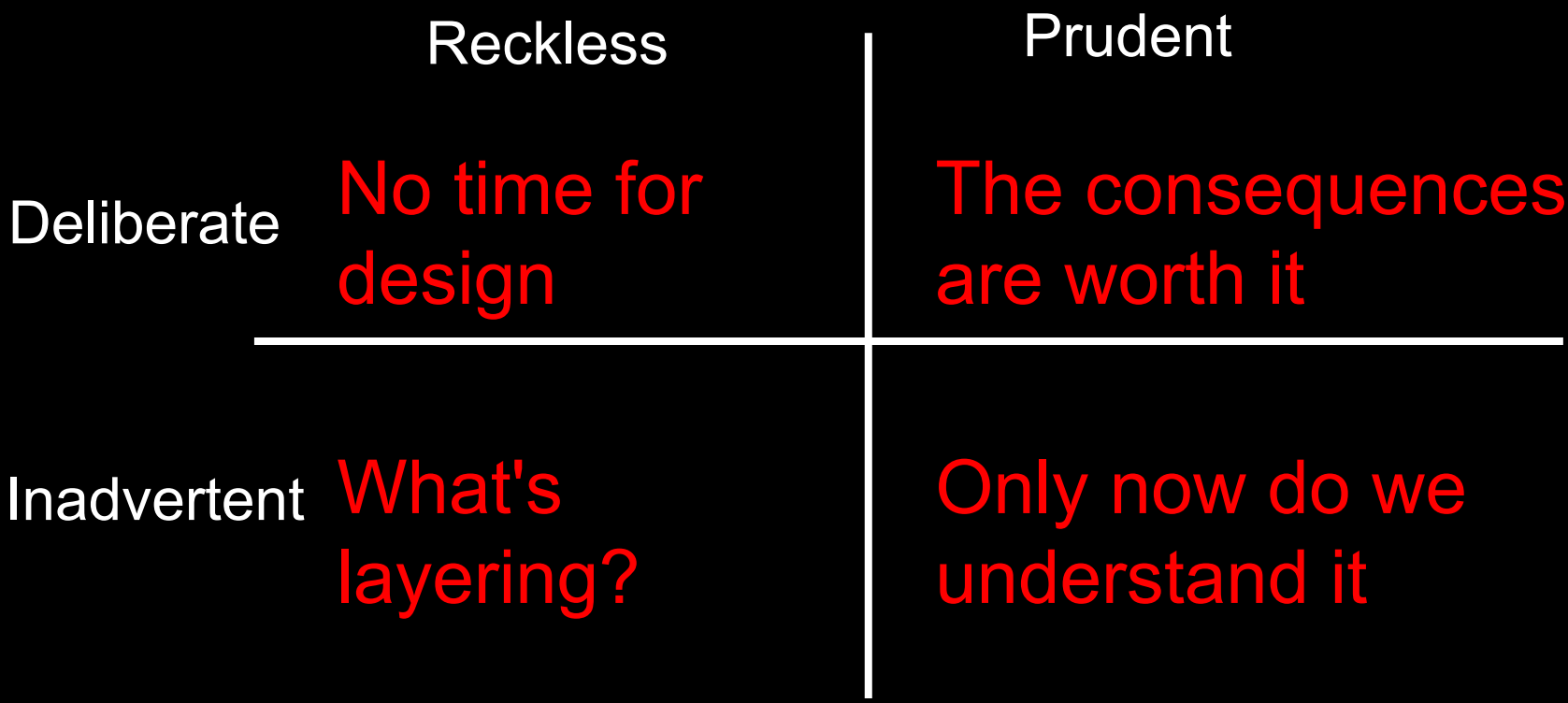
Contact Me

Adam Juda - TapRun, LLC

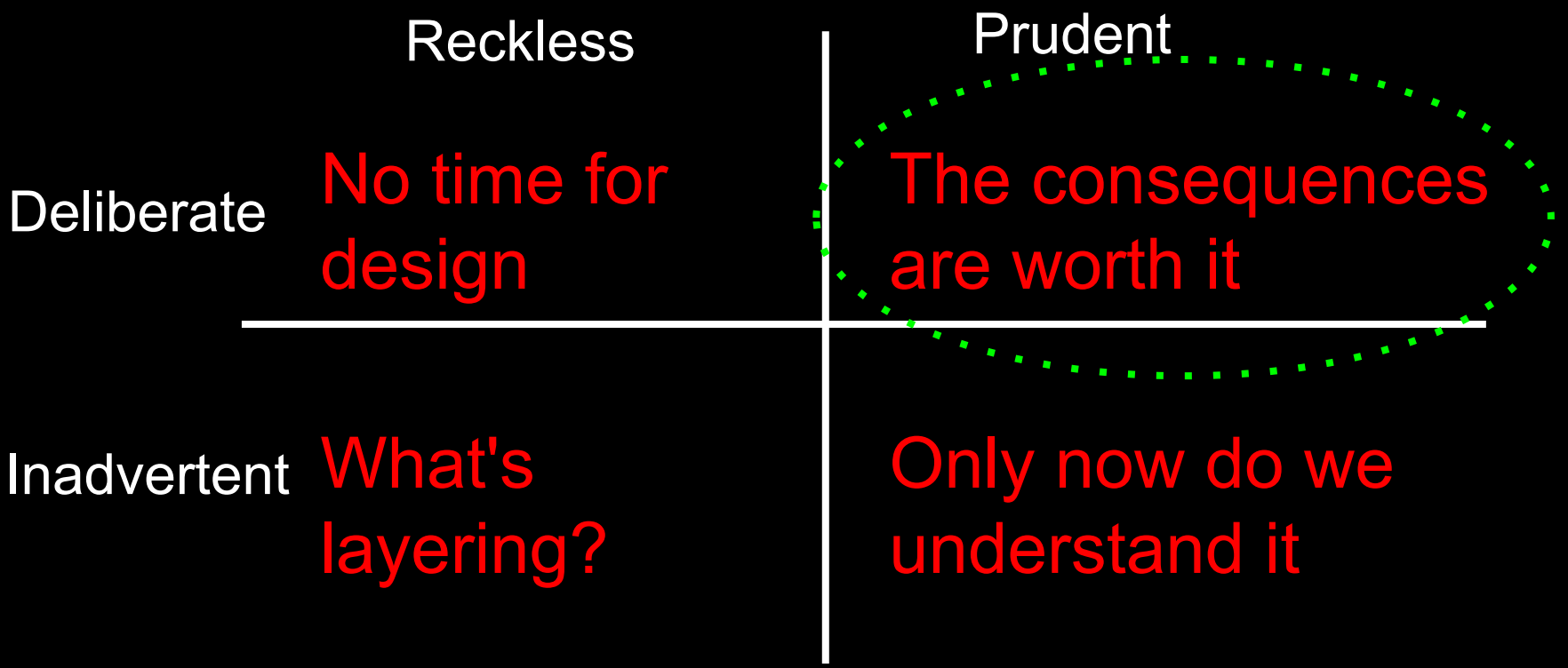
- adam@TapRun.com
- https://TapRun.com

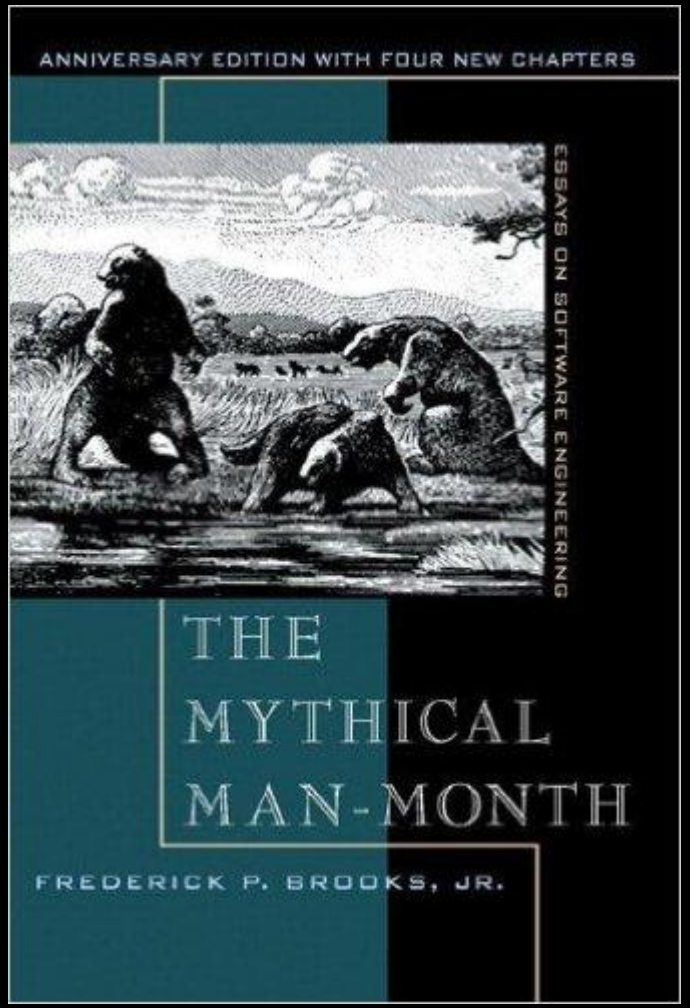
Backup Slides

Martin Fowler's types of debt



Martin Fowler's types of debt

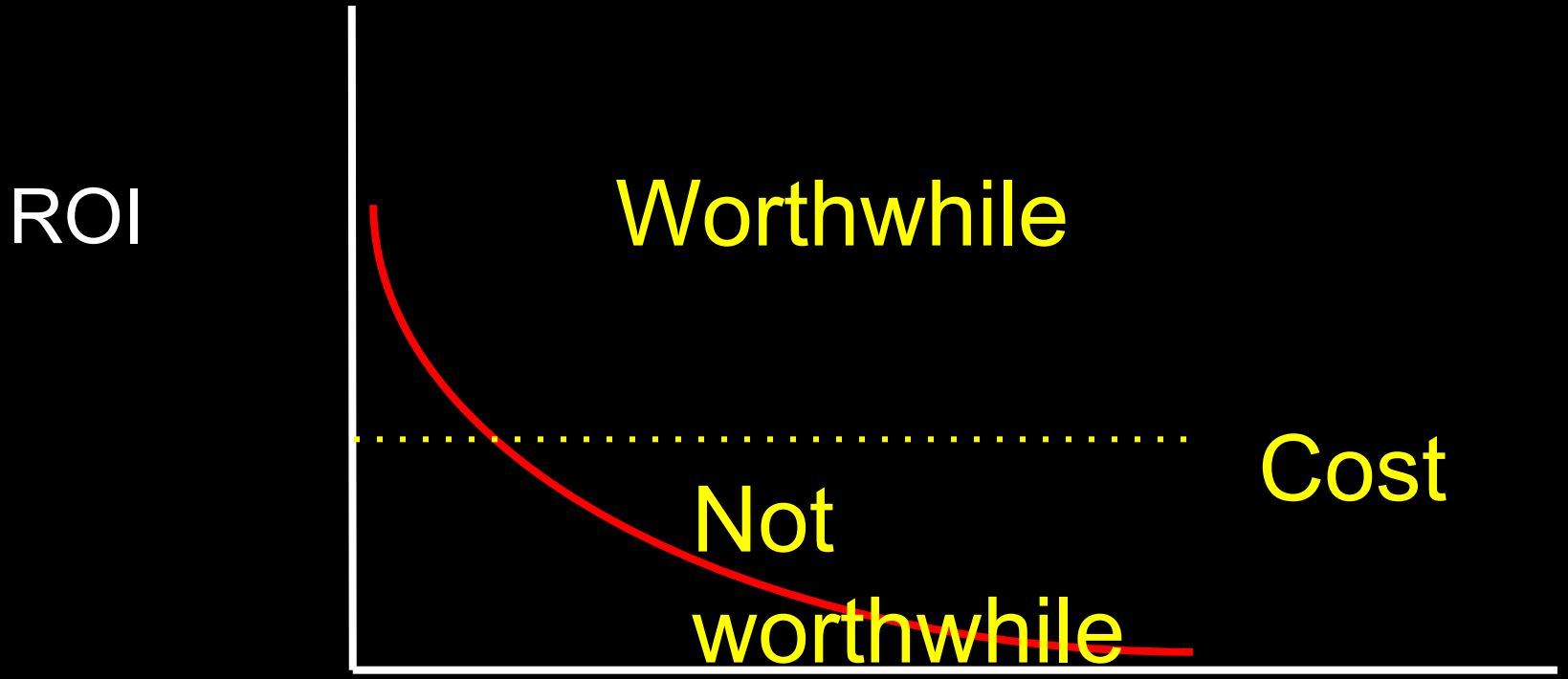


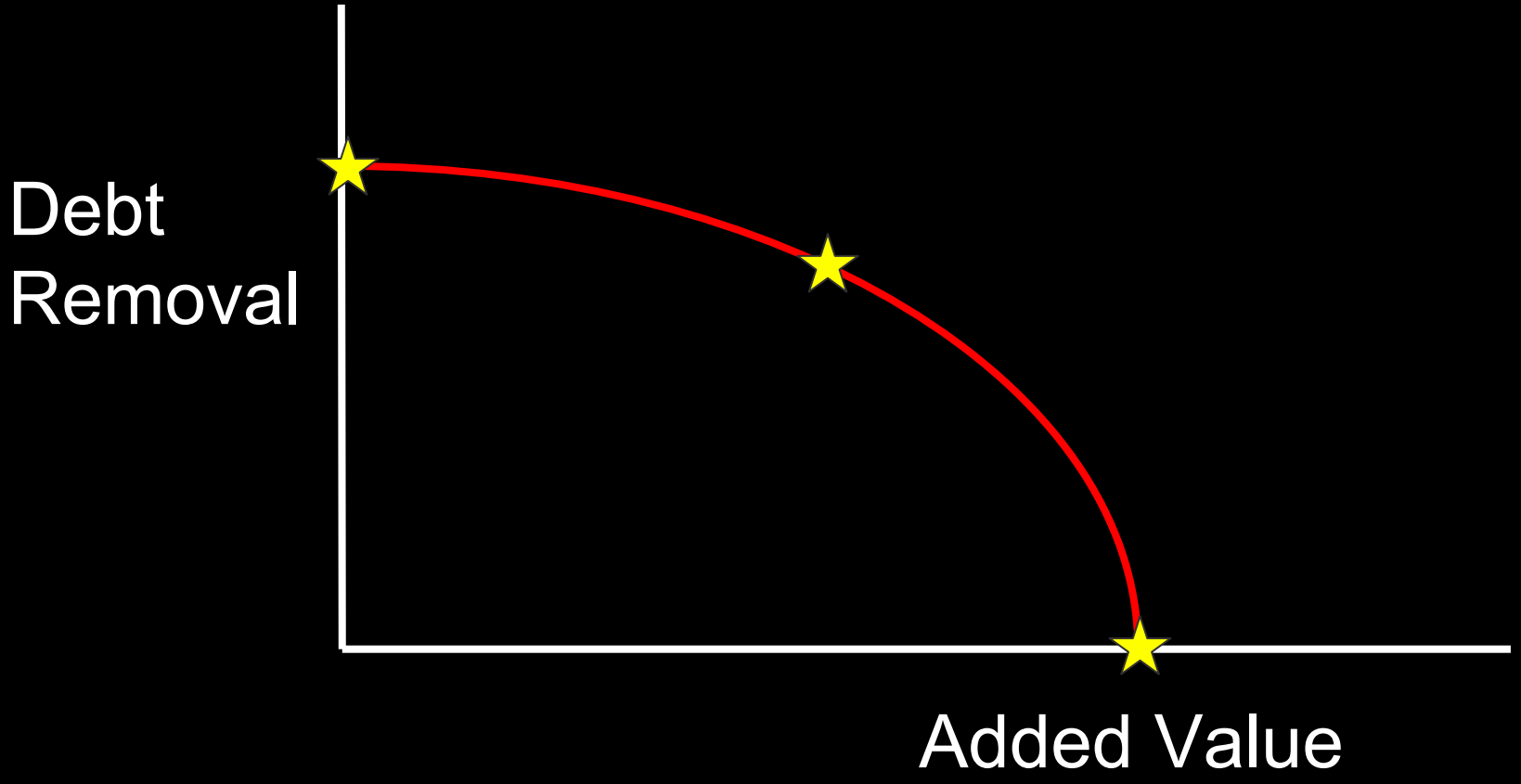


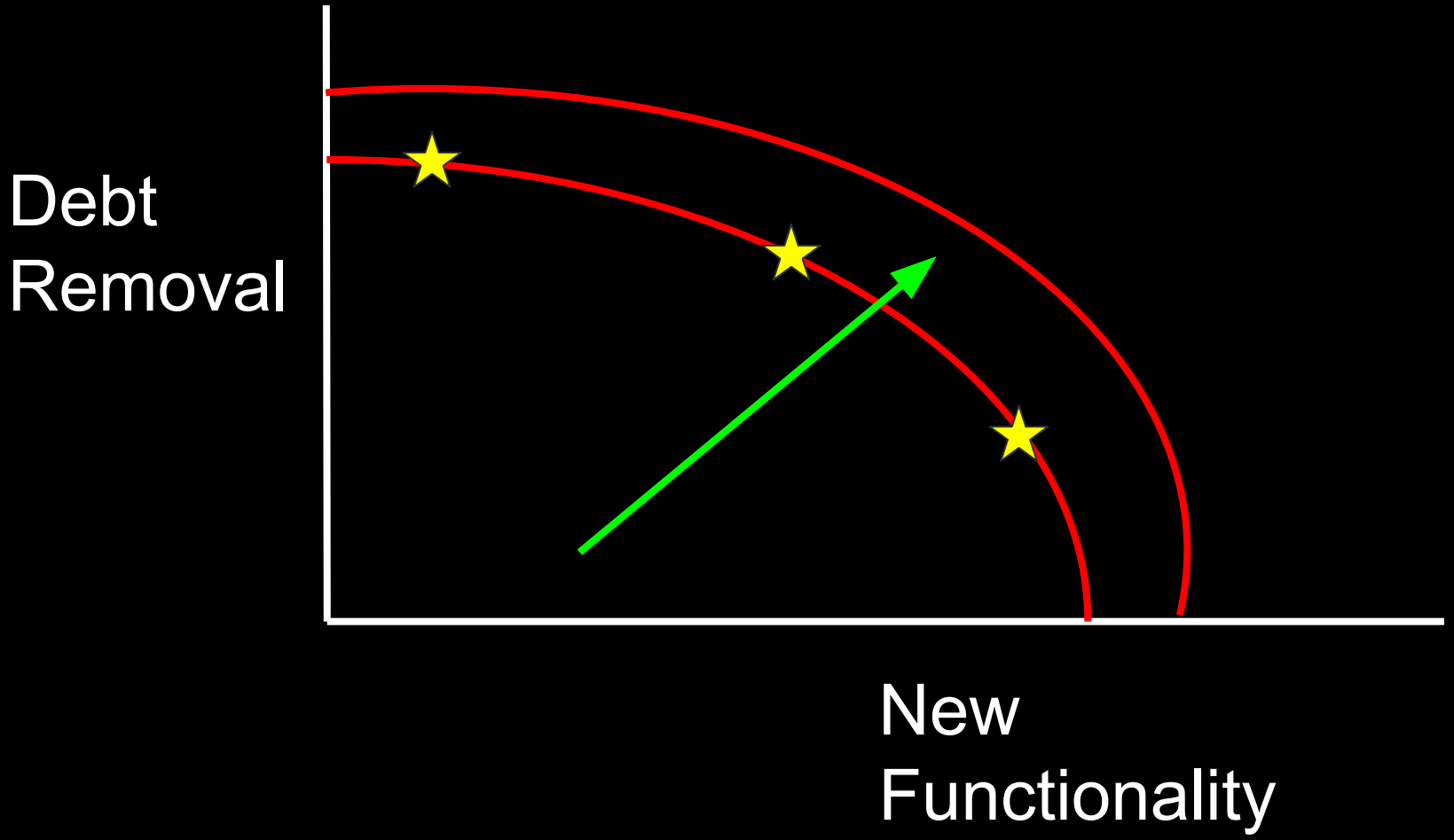
No Silver Bullet: Essence and Accidents of Software Engineering

In a matrix environment...

You can tax the projects


ROI
Worthwhile
Cost
Not
worthwhile


Debt
Removal
Added Value


Debt
Removal
New
Functionality

Estimating with uncertainty

(optimistic +

4* average +

pessimistic) / 6