MANCHESTER 1824
The University of Manchester

Evolutionary Design: Introduction

Andy Carpenter

School of Computer Science

(Andy.Carpenter@manchester.ac.uk)

Elements these slides come from Sommmerville, author of “Software Engineering", and are copyright Sommerville

BUFD Assumption

"Given a complete and correct requirements specification, we can create the best design for the system up front"

Hold?

Requirements change

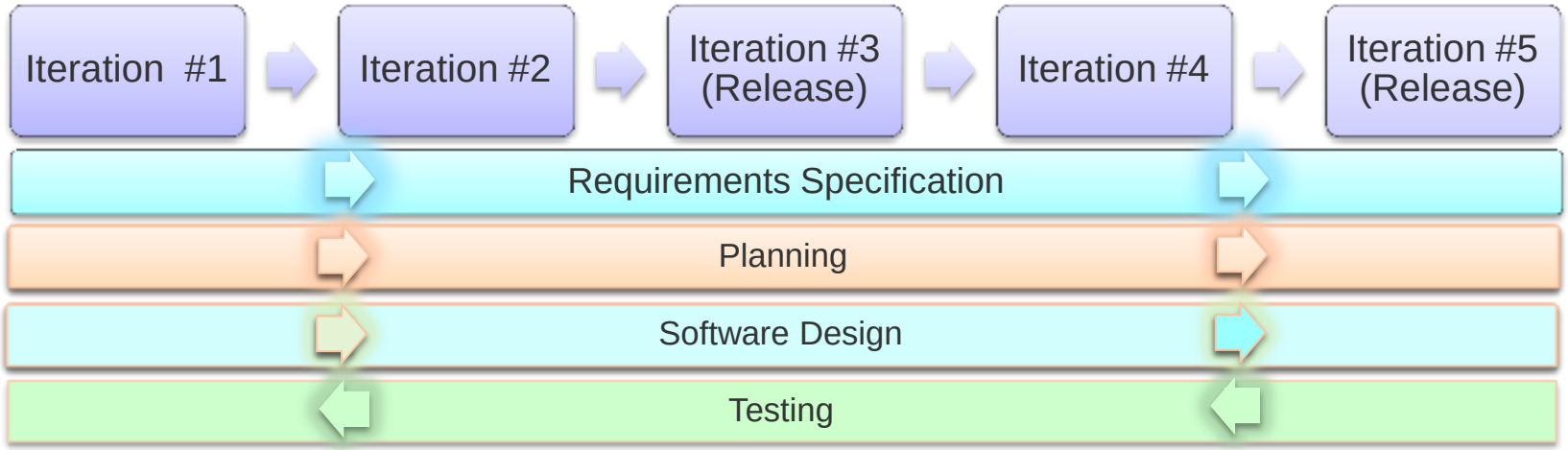
Humans make mistakes

Technologies change

Design is choice

"Good" design does not last

Evolutionary design ≠ hacking!



Emergent design

MANCHESTER 1824
The University of Manchester

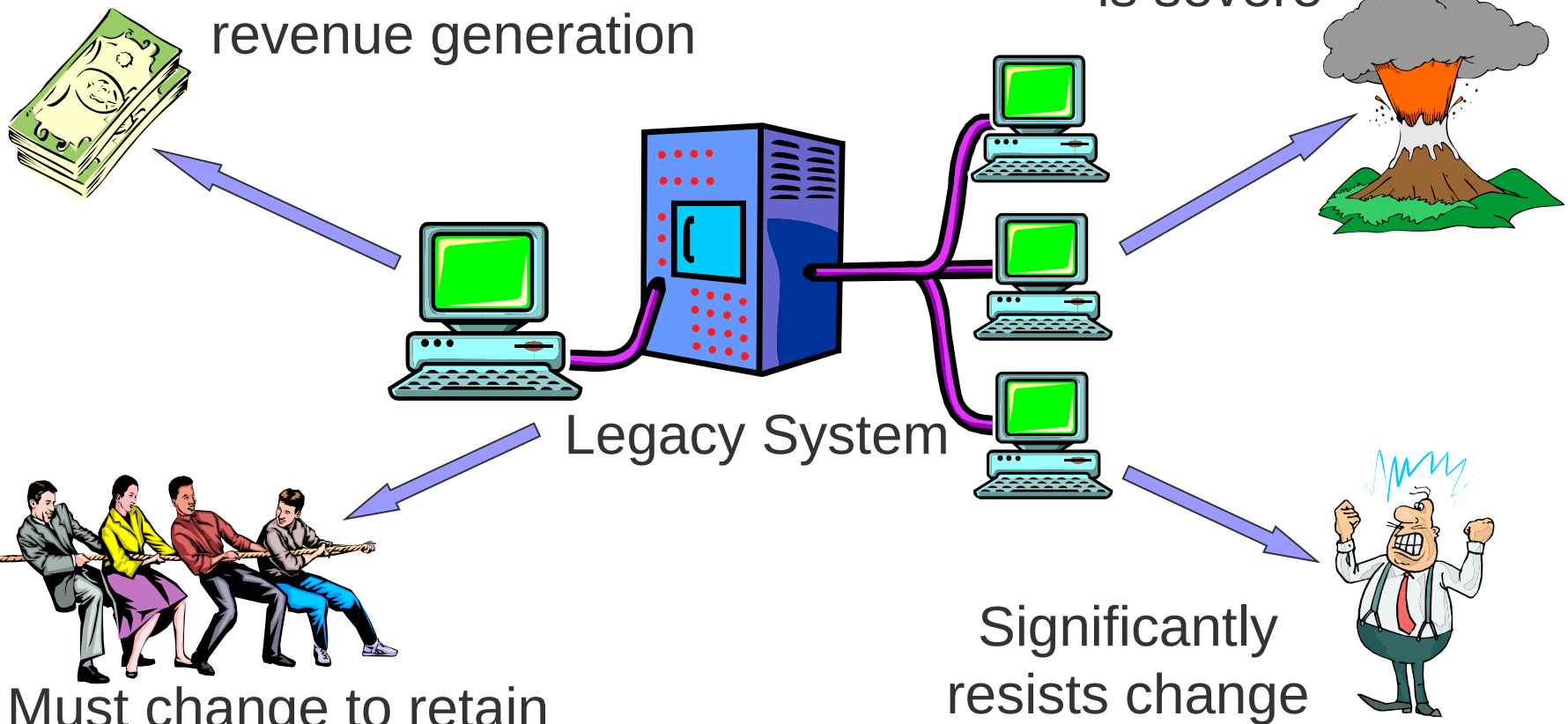
What is a Legacy System?

Legacy Systems are an Asset

- Particularly in terms of their data
- Example: BT and their customer database
  - new marketing techniques, such as churn analysis

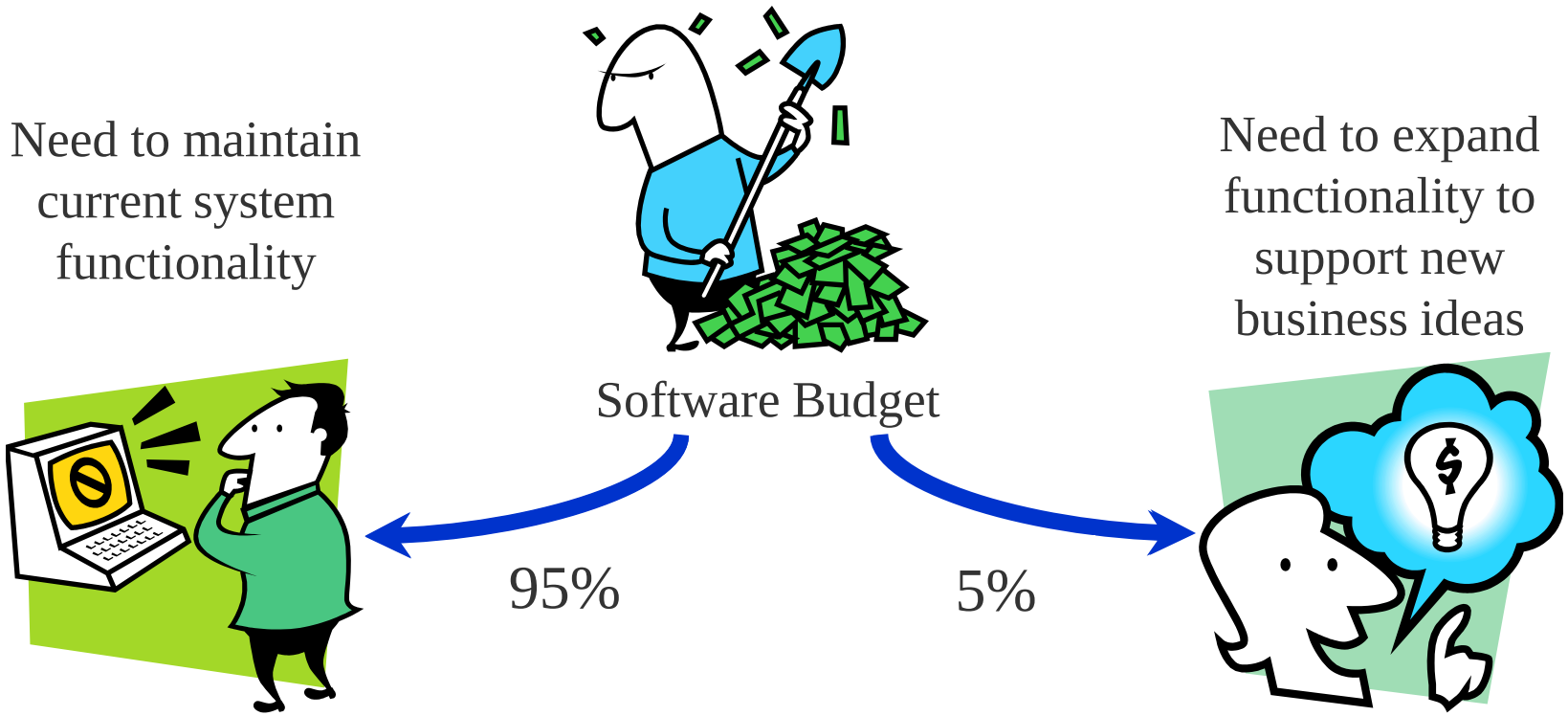
- Also, in terms of system integration
- Example: just-in-time stock management
  - Integrate inventory systems with order handling systems to support high efficiency warehouse management/production schedules

The Legacy Dilemma



Maintenance Bound

- Some organisations found themselves in the position of being maintenance bound



Lehman's Second Law

The Law of Increasing Complexity

"As a large program is continuously changed, its complexity, which reflects deteriorating structure, increases unless work is done to maintain or reduce it."

M. Lehman (1980)

Lehman, M. M. (1980). "On Understanding Laws, Evolution, and Conservation in the Large-Program Life Cycle". Journal of Systems and Software 1: 213–221.

Evolving Legacy Systems

- Perform preventive maintenance to improve maintainability

- Rebuild the system from scratch (as if current system did not exist)

Evolutionary Design: Where?

Local code level	System code level	System component level
Refactoring	Test-driven design	Incremental migration

Open question: can these measures (especially TDD) lead to a principled, high-quality design, according to current notions?

Refactoring

Preventive maintenance

Who done?

How often done?

In eXtreme Programming

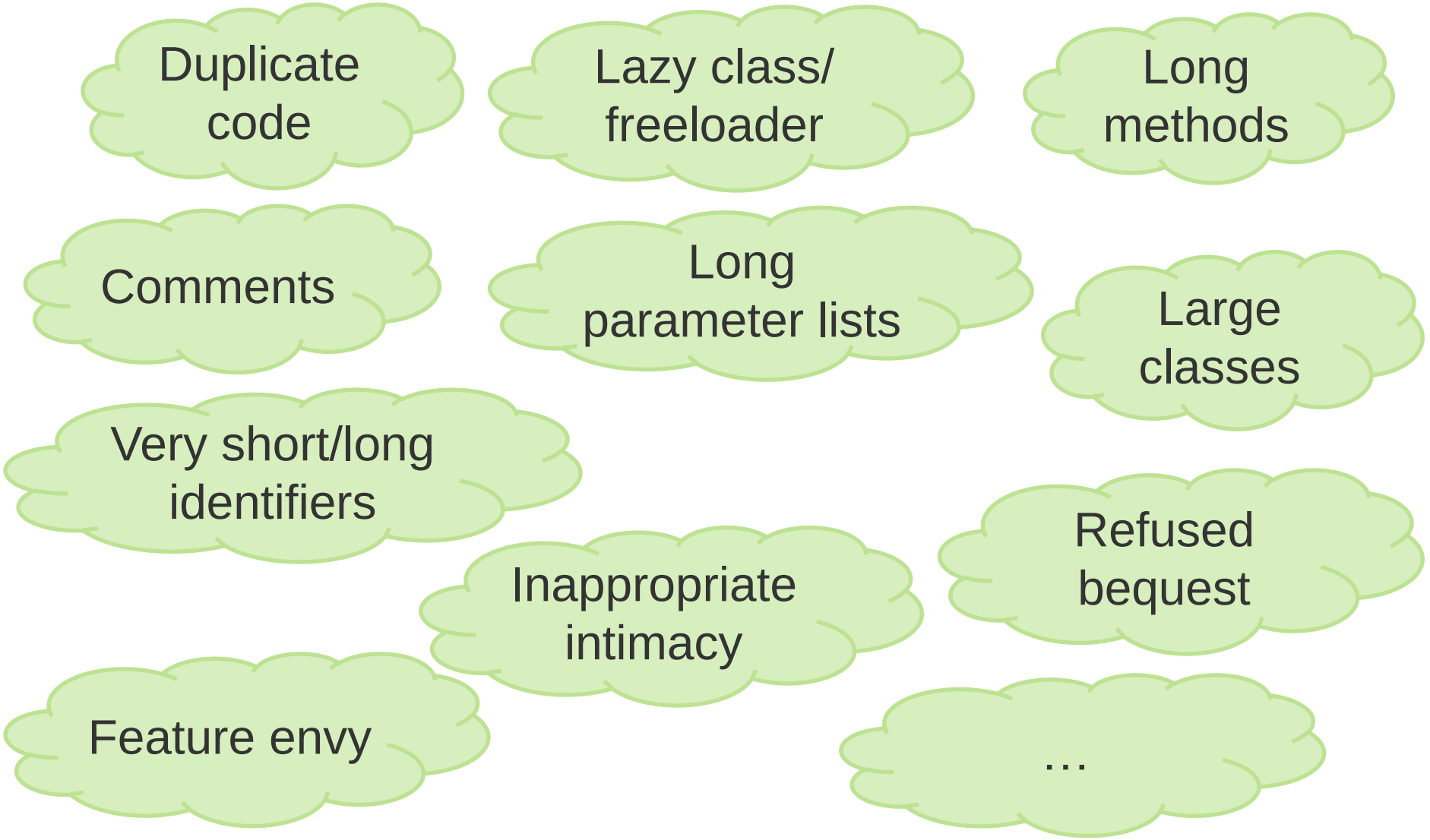
Removing bad practice

Backdoor refactoring

Feasible with extensive tests

IDE support

Code Smells


Duplicate code
Lazy class/ freeloader
Long methods
Comments
Long parameter lists
Large classes
Very short/long identifiers
Refused bequest
Inappropriate intimacy
Feature envy
…

MANCHESTER 1824
The University of Manchester

Comments?

- Sometimes comments indicate lazy programming
  - use better names
  - extracting code into methods
- What is a good comment?

- Why try to reduce comments?

Comment Removal Example

```
public class Matcher {
  public boolean match(int[] expected, int[] actual,
      int clipLimit, int delta) {
    // Clip "too large" values
    for (int i = 0; i < actual.length, i++)
      if (actual[i] > clipLimit)
        actual[i] = clipLimit;
    // Check for length differences
    if (actual.length != expected.length)
      return false;
    // Check that each …
    if (Math.abs(expected[i] - actual[i]) > delta)
        return false;
    return true;
  }
}
```

Example without Comments

```
public class Matcher {
  public boolean match(…) {
    if (actual.length != expected.length)
      return false;
    clipLarge(actual, clipLimit);
    return elementsSimilar(expected, actual)
  }
  private clipLarge(int[] actual, int clipLimit) {
    for (int i = 0; i < actual.length, i++)
      if (actual[i] > clipLimit)
        actual[i] = clipLimit;
  }
  private elementsSimilar(int[] a, int[] b) {
    for (int I = 0; I < a.length; i++)
      if (Math.abs(a[i] - b[i]) > delta)
        return false;
    return true;
  }
```

Refactoring During Development

- The chicken and egg problem of design
- The Design/Refactoring cycle
  - Don’t spend ages searching for the “best” design
  - Go with what’s sensible now
  - Refactor later when you know more
  - (this is all very Agile!)
- Limitations of refactoring for preventive maintenance?

MANCHESTER 1824
The University of Manchester

Evolutionary Design

- Refactoring is key to evolutionary design
  - Move step by step towards a better design
  - Changes incorporated at each step
  - Fosters programmer culture of always looking for opportunities to improve
  - Gives programmers courage to change code for better
    - Small, reliable, repeatable changes
    - Backed up by safety net of automated tests

- But – not enough by itself!

MANCHESTER 1824
The University of Manchester

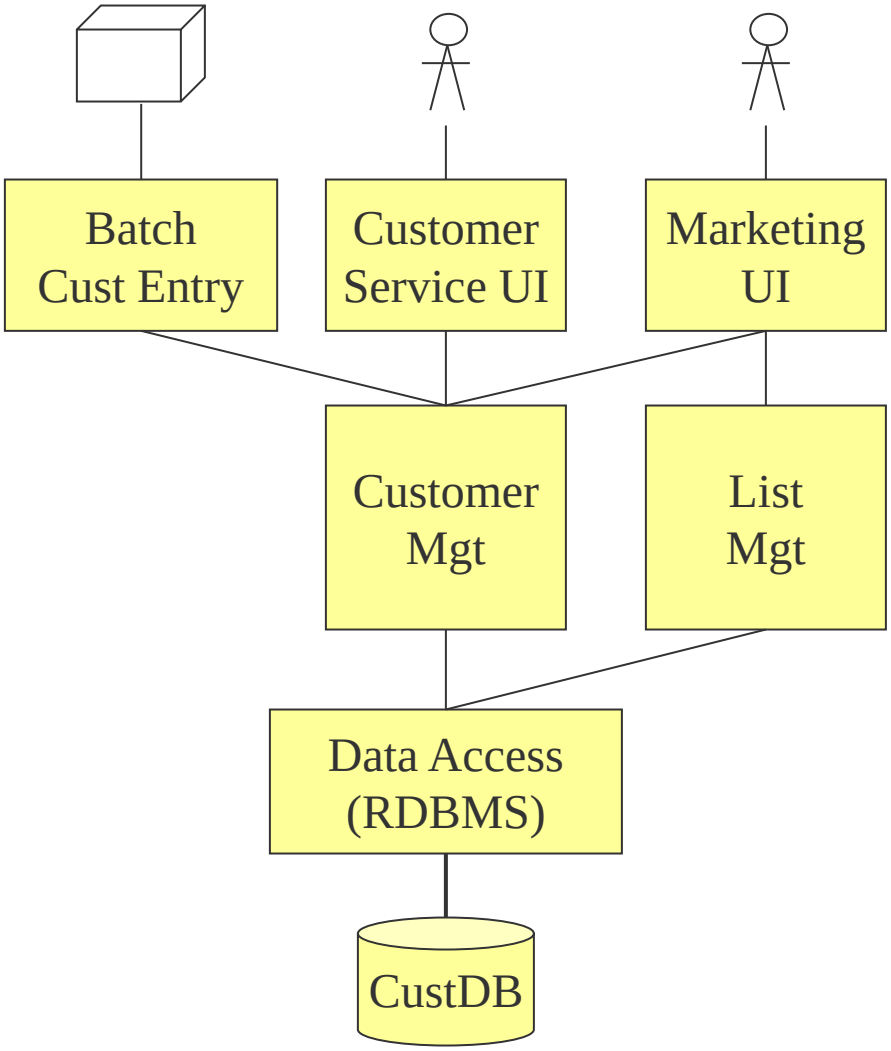
Evolutionary Design

- Refactoring doesn't change the code behaviour
  - Cannot help us with perfective or corrective maintenance
- Need a way to guide behavioural changes to code that leads to a principled, high quality design
  - How can we make good system-level design decisions, while making small, incremental changes?
- Any ideas?

Dividing Up the Work

- Software architecture split into communicating components

- One possibility is to implement component by component
- A good idea?



Dividing Up the Work

- Advice: implement in thin end-to-end slices
- To ensure slices are meaningful and deliver value, use acceptance tests to define slices
- What does this sound like?