l e a n

software development

Specialization & the Economies of Scale

Where Adam Smith went wrong

mary@poppendieck.com Mary Poppendieck www.poppendieck.com

The Pursuit of Simplicity

The Question:

How do we deal with Complexity?

The Answer:

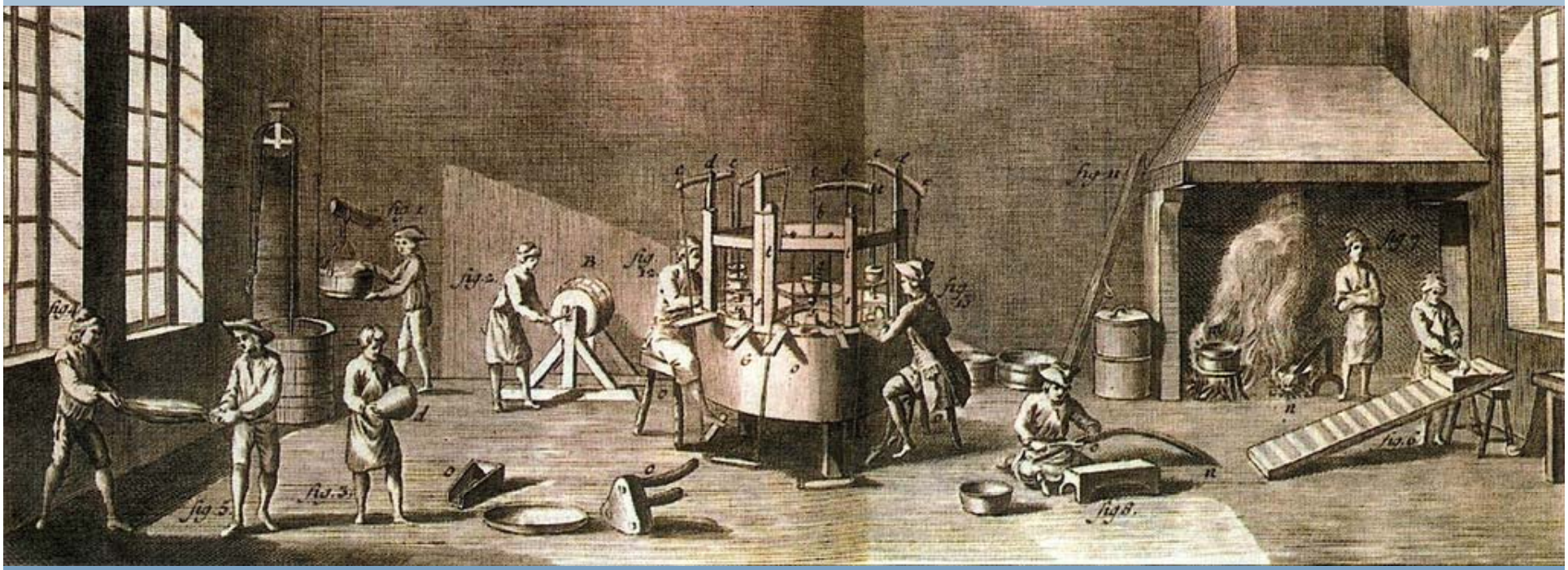
Divide and Conquer!

The Real Question:

Along what Lines do we Divide?

Specialization

The Pin Factory – Adam Smith (1776)



One man draws out the wire, another straights it, a third cuts it, a fourth points it, a fifth grinds it at the top for receiving the head; to make the head requires two or three distinct operations; to put it on, is a peculiar business, to whiten the pins is another; it is even a trade by itself to put them into the paper; and the important business of making a pin is, in this manner, divided into about eighteen distinct operations.

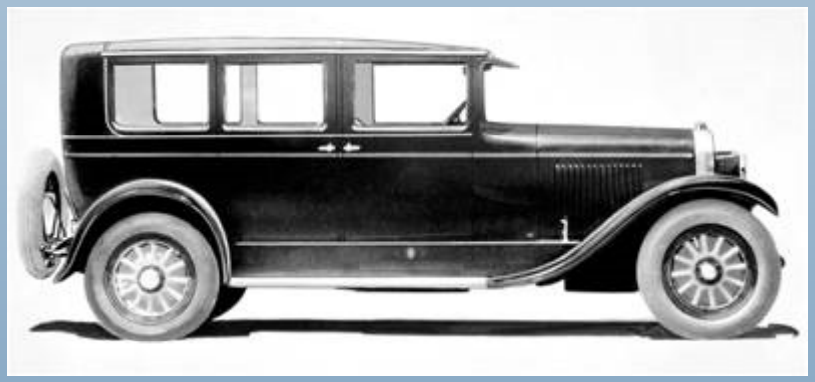
The Ultimate Pin Factory

Ford River Rough citra1930

"You can have any color you want, as long as it's black."

The Enemy: Variety

A Car for Every Purse and Purpose

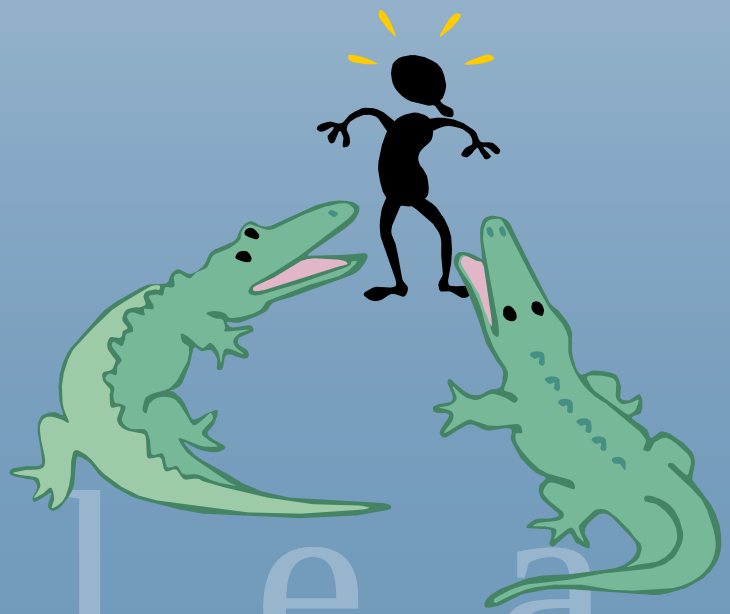


What's wrong with Variety?

1. The Cost of Changeover
   - Batch & Queue
2. The Cost of Delay
   - Cash Flow
   - Obsolescence
3. The Cost of Mistakes
   - Hidden Defects
4. The Cost of Motion
   - Moving, tracking, storing
5. The Cost Complexity
   - ***Scheduling***
   - Learning

Little's Law

$$\text{Time Through the System} = \frac{\text{Number of Things in Process}}{\text{Average Completion Rate}}$$



The Work Cell: Divide by Part, not Process

Aircraft Sidewall Fastener

Before: $25-$30

Extrusion

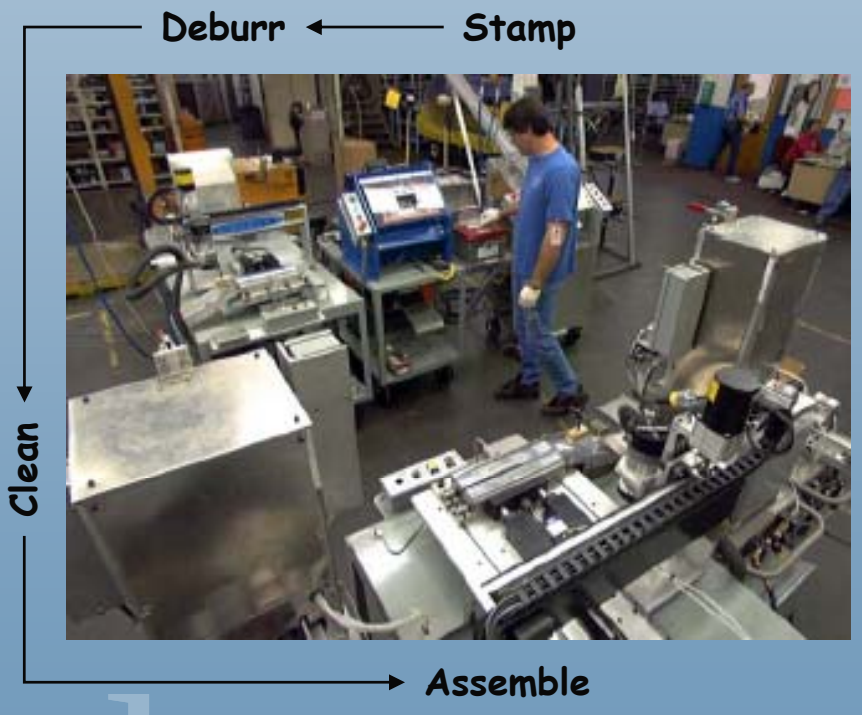
Machining

Paint

Deburr

Assemble

After: $2.5 - $3



Design Techniques for Meeting Market Driven Target Costs
Benny Leppert, Lean Design & Development, Chicago, 2005

Dealing with Complexity in Software Engineering

1968: Quality by Construction

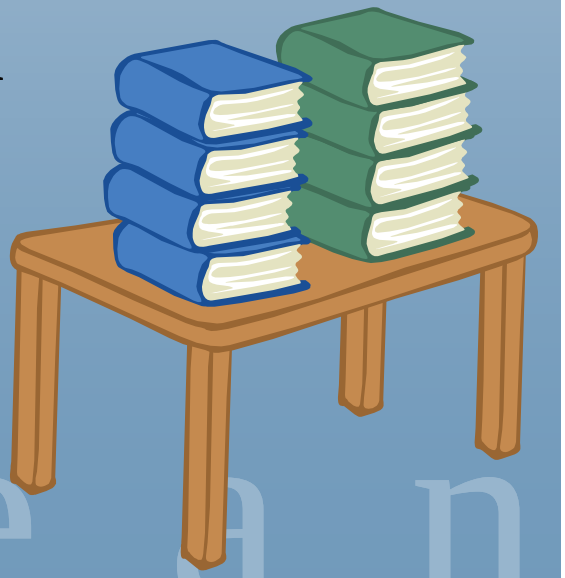
Edsger Dijkstra – [Structured Programming]

- ✓ "Those who want really reliable software will discover that they must find means of avoiding the majority of bugs to start with, and as a result the programming process will become cheaper. If you want more effective programmers, you will discover that they should not waste their time debugging – they should not introduce bugs to start with."

1972: Step-Wise Integration

Harlan Mills – [Top-Down Programming]

- ✓ "My principle criterion for judging whether top down programming was actually used is [the] absence of any difficulty at integration."



Divide by Hierarchy / Order of Execution

Dealing with Complexity
An Orthogonal View

1972: Information Hiding

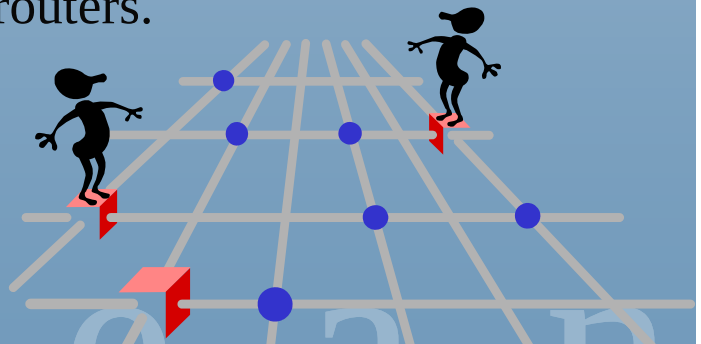
Dave Parnas – [Criteria for Decomposition]

- ✓ Divide program into modules based on their responsibility and the likeliness of future change, rather than structure, hierarchy, or order of flow.

1974: Internet Protocol

Vinton Cerf and Robert Kahn – [TCP/IP]

- ✓ Each distinct network stands on its own, no internal changes required.
- ✓ No information retained by the gateways and routers.
- ✓ No global control at the operations level.
- ✓ Communications on a best-effort basis.



Divide by Matching the Domain

- ✓ ***Commonality & Variability Analysis***

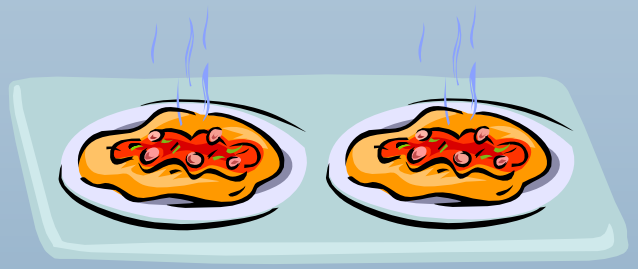
Case Study: Complexity at Scale

Amazon.com

It's all about scale.

2000 – Hit the wall

2001 – Started transition to services



- ✓ Encapsulate data and business logic
- ✓ Basic Services and Consolidator Services
- ✓ Each Owned by a 2PT with end-to-end responsibility

Amazon CTO Werner Vogels

"If you need to do something under high load with failures occurring and you need to reach agreement – you're lost."

Conway's Law

"Organizations which design systems are constrained to produce designs which are copies of the communication structures of these organizations."

Dealing with Complexity
The Specialization View

The Problem

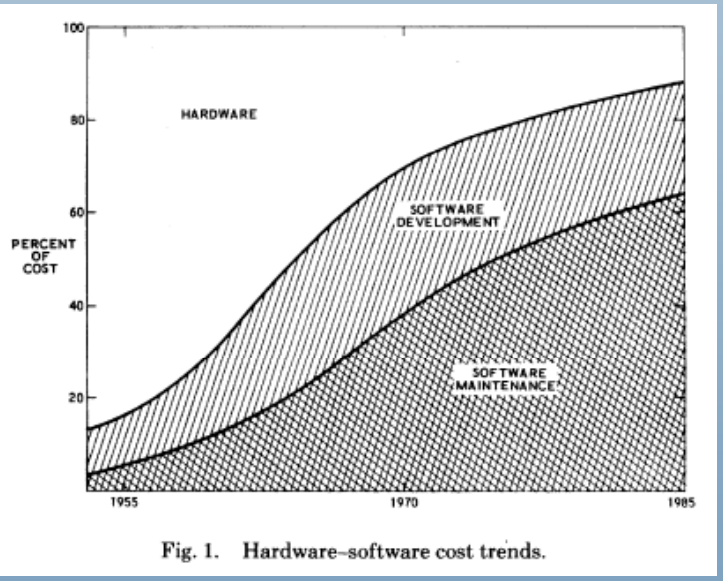


Fig. 1. Hardware–software cost trends.

The Solution

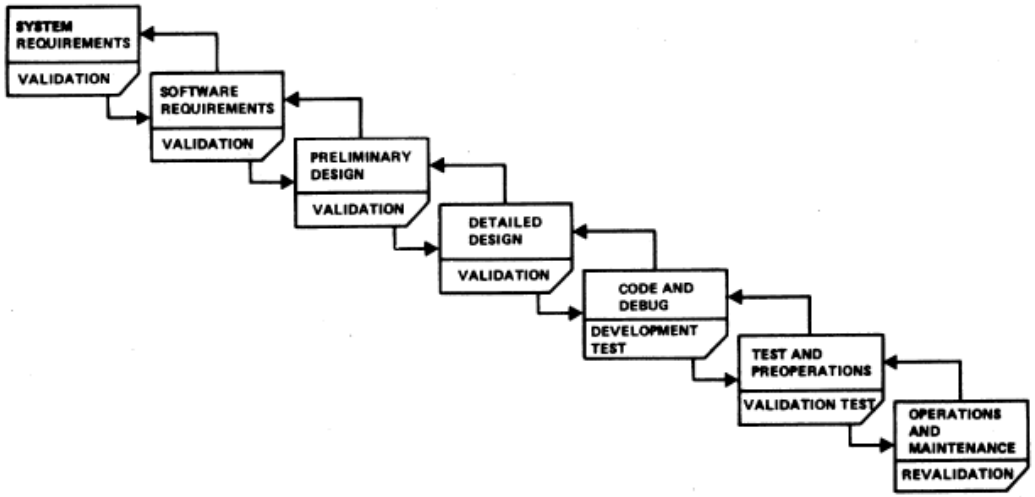


Fig. 2. Software life cycle.

1976: Software Engineering – Barry Boehm
IEEE Transactions on Computers

What's wrong with Specialization?

Manufacturing

1. The Cost of Changeover
   - Batch & Queue
2. The Cost of Delay
   - Cash Flow
   - Obsolescence
3. The Cost of Mistakes
   - Hidden Defects
4. The Cost of Motion
   - Moving, tracking, storing
5. The Cost Complexity
   - ***Scheduling***
   - Learning

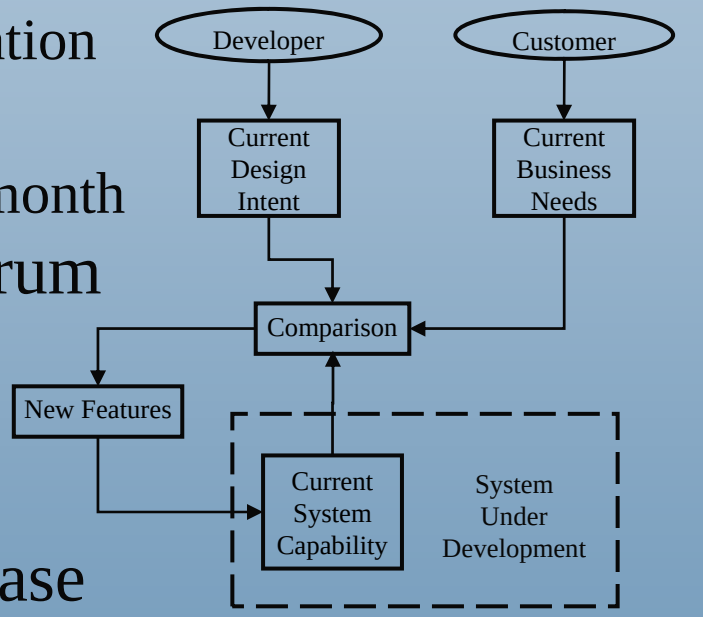
Software Development

1. The Cost of Integration
   - Wait to Test
2. The Cost of Delay
   - Cash Flow
   - Obsolescence
3. The Cost of Mistakes
   - Hidden Defects
4. The Cost of Task Switching
   - Thrashing
5. The Cost Complexity
   - ***Learning***
   - Scheduling

Case Study: Reducing Complexity with Iterations and Feedback

WebSphere® Service Registry and Repository

- ✓ 10 month deadline – didn't know the details
  - × Solution: Customer feedback every iteration
- ✓ Early Access Program
  - × Customers download new version each month
- ✓ Customers feedback on discussion forum
  - × Direct developer-customer interaction
- ✓ Changed course midstream
  - × Customer feedback beat marketing input
- ✓ Phenomenal sales the first day of release
  - × Customers knew what they would get
- ✓ Support Calls down by an order of magnitude
  - × Mental model of customers and developers matched

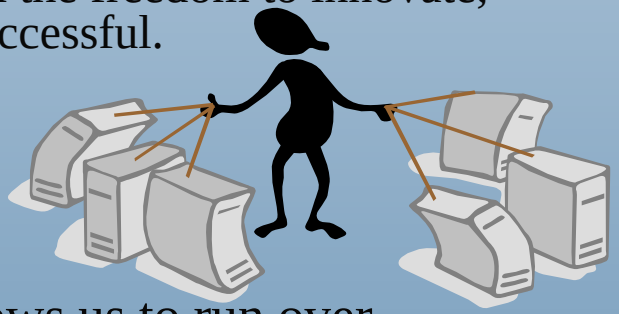


Case Study: Reducing Complexity with Early Testing

Helen's Story

1. **Carlos** – With Agile, you don't need to worry about documentation, right?
2. **Rene** – Our criteria for 'done' was not defined well enough. Technical debt grew badly.
3. **Charlie** – We tried again and this time the results were awesome!

Lesson – Give your teams the right support means giving them the freedom to innovate, but also give them the tools and training they need to be successful.



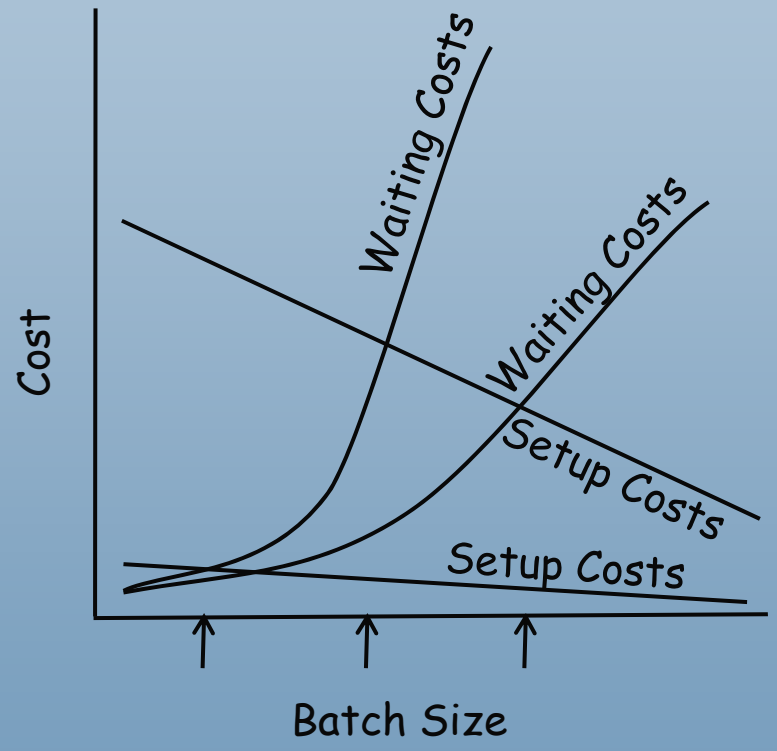
Charlie's Story

- ✓ I went to your class, and I told myself you were nuts.
- ✓ Jeff developed a test strategy and set of tools that allows us to run over 1500 test cases every night on 10 environments, something that used to take days. By morning every bit of code that was written the day before has run against all possible databases and operating systems.
- ✓ We wrote separate functional test suites. You can run any test suite quickly with no interdependence. We build every two hours. We wrote plug-ins for the development environment so tests can be run from any sandbox.
- ✓ We are in great shape – **2** orders of magnitude of defects going into hardening.

Batch Size

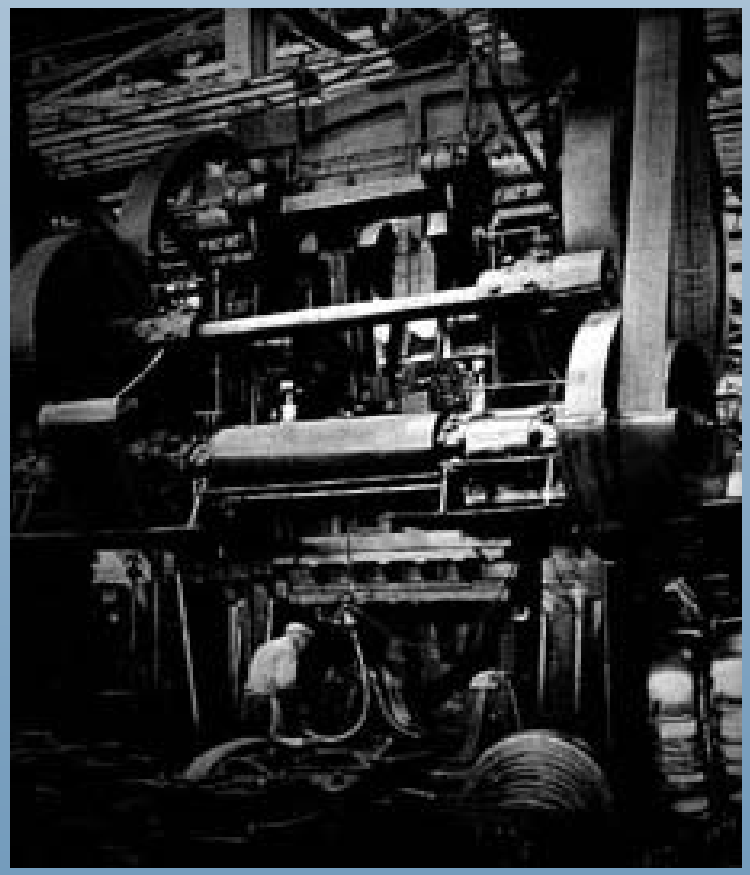
What is Batch Size?

- ✓ The amount of information transferred at one time
  - × The % of specifications completed before development begins
  - × The amount of code tested in a system test
- ✓ Compare:
  - × Cost of setup (linear)
    - ❖ Test set-up and execution
  - × Cost of waiting (hyperbolic)
    - ❖ Find/fix defects long after injection
- ✓ Waiting costs are:
  - × Usually hidden & delayed
  - × Often larger than expected
- ✓ The Lean Approach:
  - × Recognize true waiting costs
  - × Drive down setup overhead



From Don Reinertsen
www.roundtable.com/MRTIndex/LeanPD/ART-reinertsen-INT2-1.html

Single Digit Set-up



Manufacturing

Common Knowledge:

- ✓ Die changed have a huge overhead
- ✓ Don't change dies very often

Taiichi Ohno:

- ✓ Economics requires frequent die change
- ✓ *One Digit Exchange of Die*

Software Development

Common Knowledge:

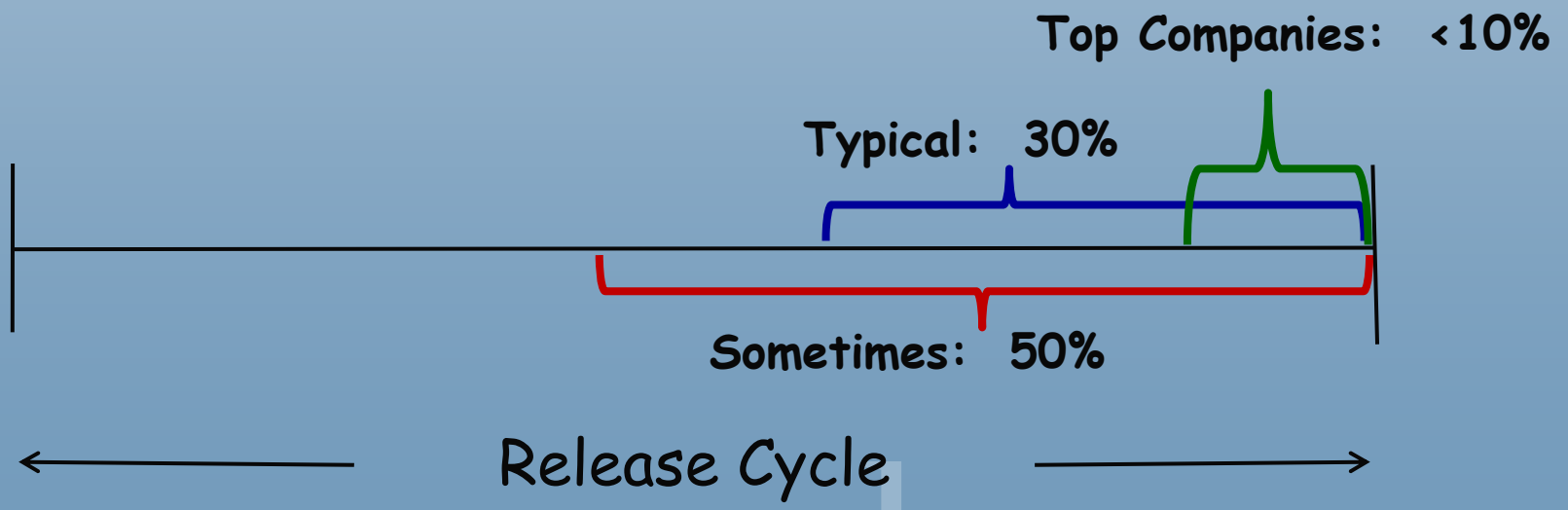
- ✓ Releases have a huge overhead
- ✓ Don't release very often

Lean:

- ✓ Economics requires many frequent releases
- ✓ *One Digit Releases*

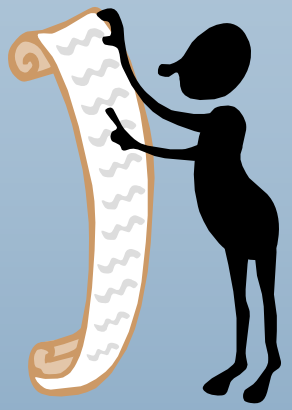
How Good are You?

When in your release cycle do you try to freeze code and test the system? What percent of the release cycle remains for this "hardening"?

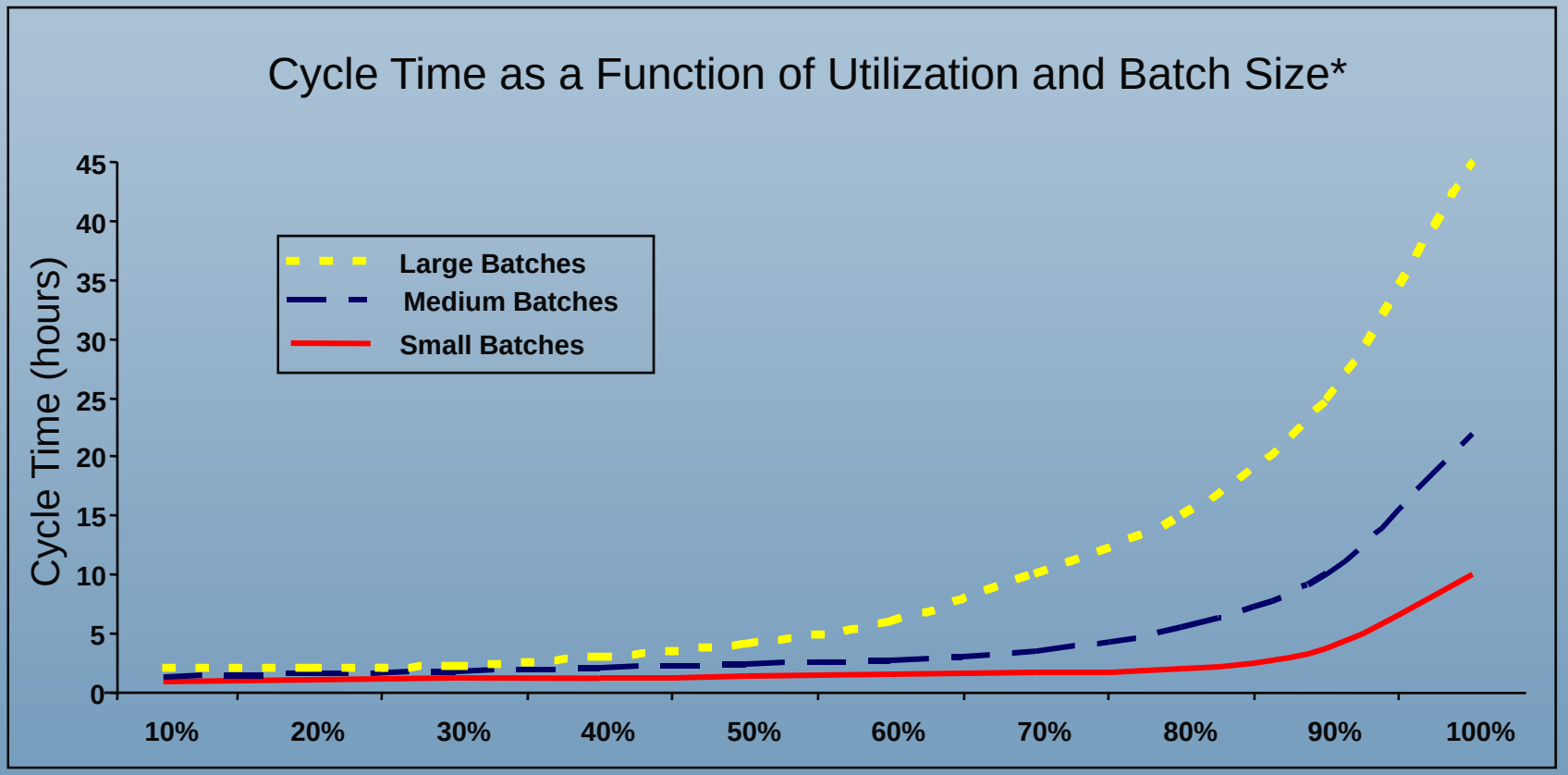


Our Policies Continue to Promote Specialization & the Economies of Scale

1. Full utilization of our most skilled workers is considered essential, so we sort work into batches and assign each batch to the appropriate specialist, making maximum use of their time and skills.
2. It is so difficult to abandon batch and queue mentality that we fail to see queues that are staring at us in the face. We are blind to lists of customer requests that would take years to clear.
3. We pull workers off of their current job to rush a yet-more-important job through our system. We ask people to work on three, five, ten or more things at once.
4. We use computer systems to make sure that everyone is busy all of the time. We schedule work and assign teams with an eye to full utilization.
5. We create annual budgets or long project plans that justify every person by committing to what they will deliver.



The Utilization Paradox



High Performance ***Thrashing***

*This assumes batch size is proportional to variability.

Economies of Scale: The Empire State Building

September 22, 1929
Demolition started

January 22, 1930
Excavation started

March 17, 1930
Construction started

November 13, 1930
Exterior completed

May 1, 1931
Building opened

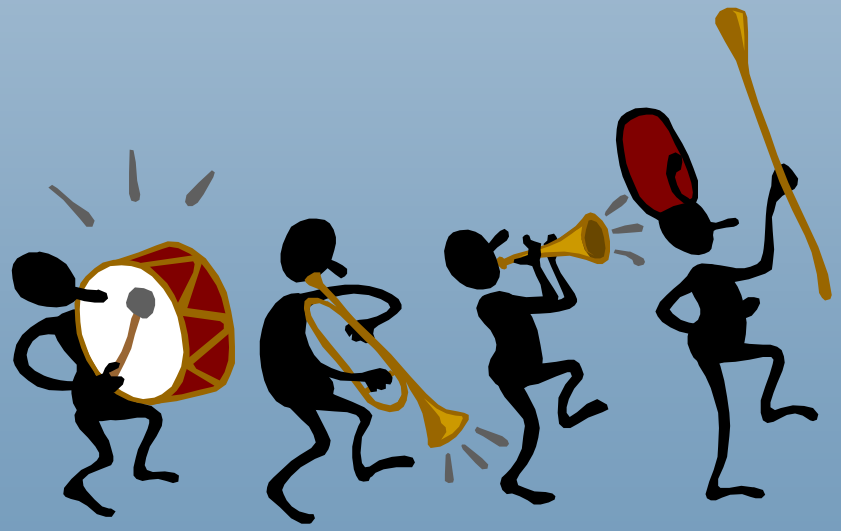
Exactly on time
18% under budget

One Year Earlier:

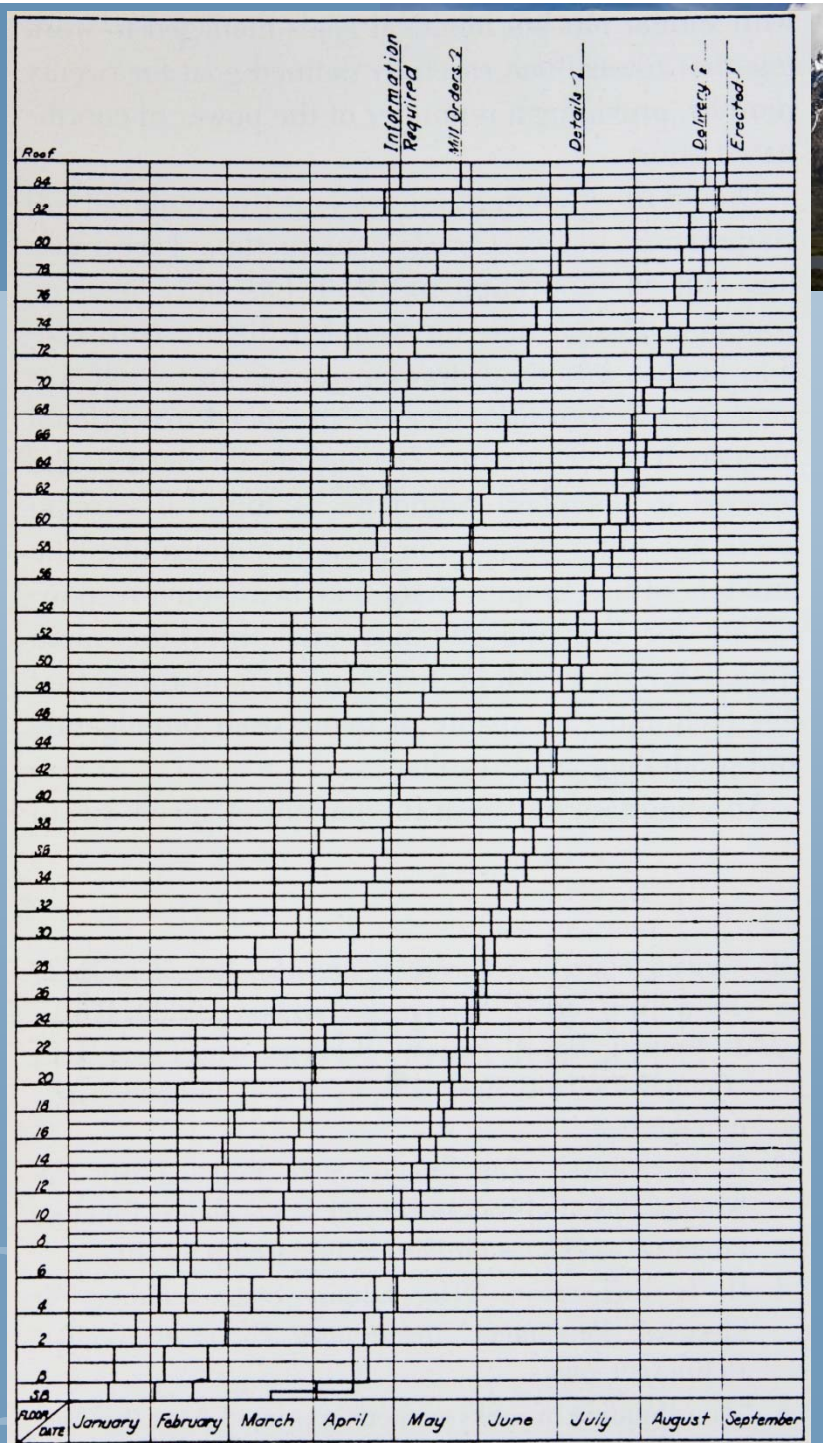
How did they do it? The key: Focus on FLOW.

Steel Schedule

We thought of the work as if it were a band marching through the building and out the top.



From: "Building the Empire State"
Builders Notebook: Edited by Carol Willis



The Four Pacemakers

1. Structural Steel Construction
   - ✓ Completed September 22, 12 days early
2. Concrete Floor Construction
   - ✓ Completed October 22, 6 days early
3. Exterior Metal Trim &Windows
   - ✓ Completed October 17, 35 days early
4. Exterior Limestone
   - ✓ Completed November 13, 17 days early

From: "Building the Empire State"
Builders Notebook: Edited by Carol Willis

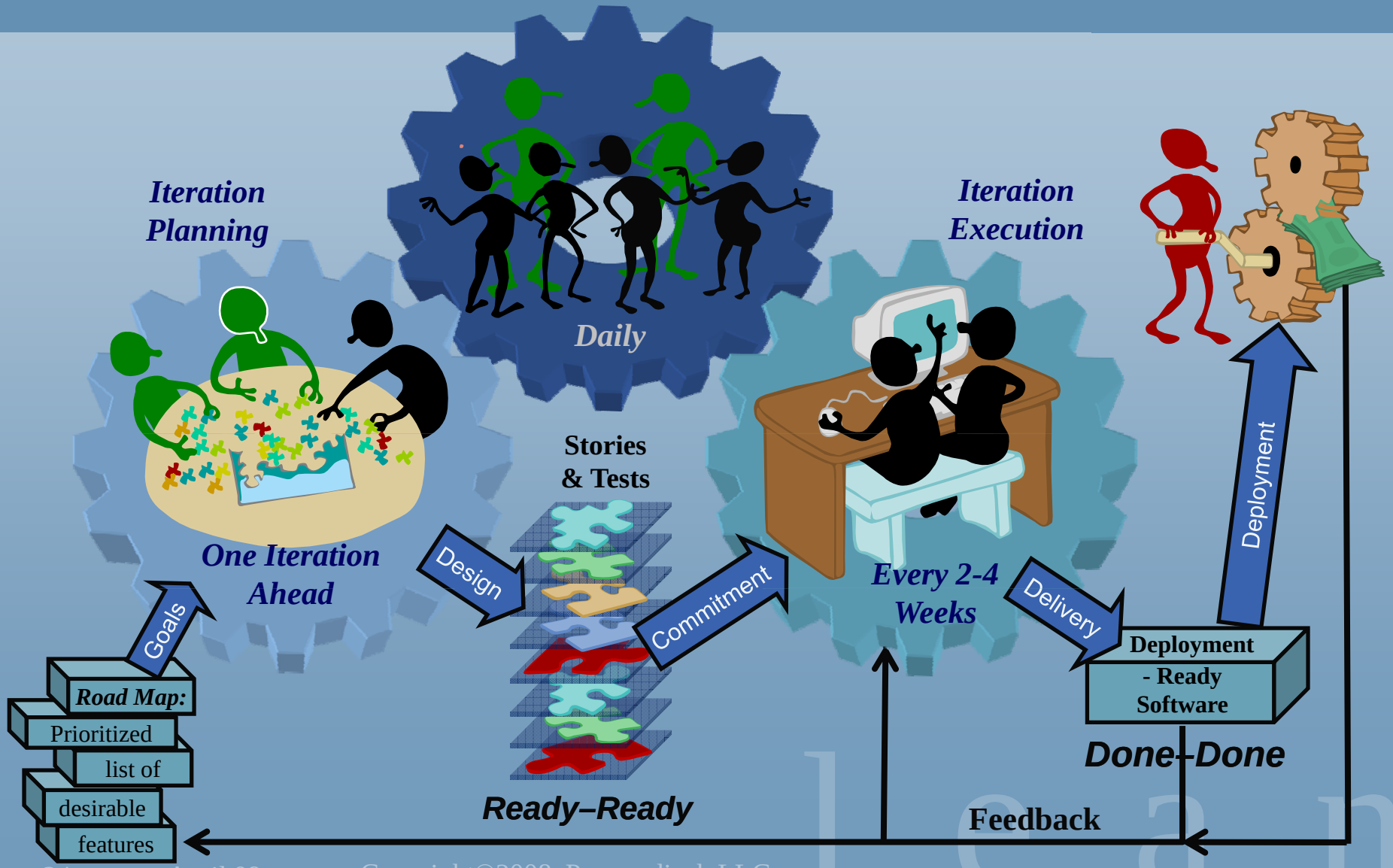
Schedule the Constraint

What is the constraint?

- Information about the problem to be solved?
- Understanding the best solution to the problem?
- Available Developers?
- Available Equipment?
- What is the constraint in YOUR domain?

Use a pacemaker schedule for the constraint.

Iterative Development



l e a n

software development

Thank You!

More Information: www.poppendieck.com

mary@poppendieck.com Mary Poppendieck www.poppendieck.com