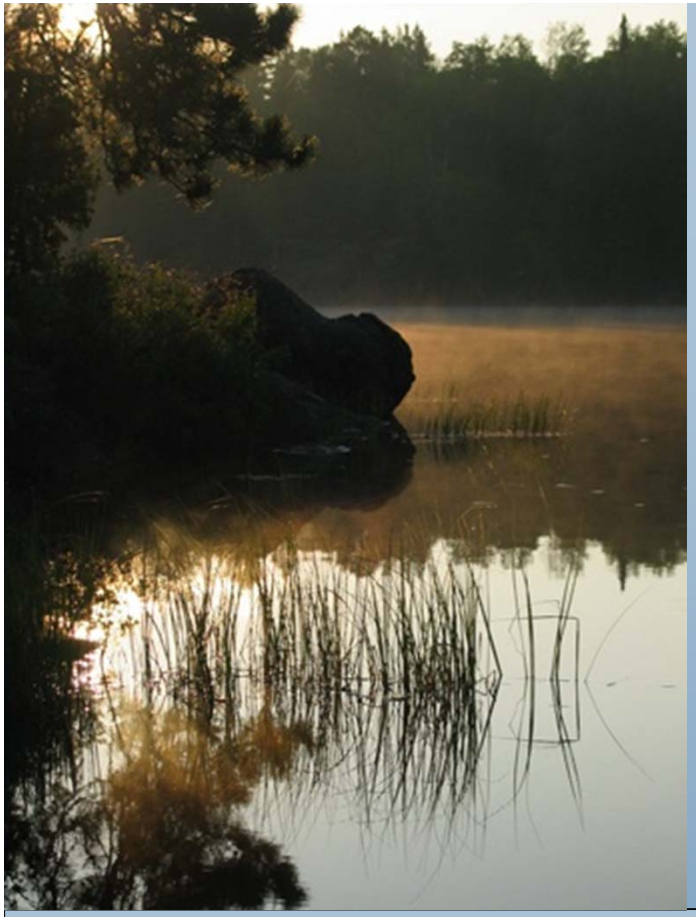


l e a n
software development

Lean Software Development

Discovering Waste

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

Two Kinds of Software Development

Process Support

The Application Development portion of IT organizations.

If you divide IT into Operations and Application Development

THEN

"Standard" Lean Tools are appropriate for IT Operations

Avoid "Standard" Lean Tools for Application Development

Product Development

Software intensive products.

Google

ebay

skype

Almost never referred to as IT by the people who do it.

Development generally does not report to a CIO.

Accounts for a large and growing majority of software developed today

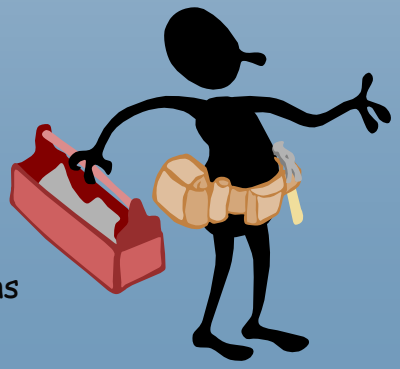
Lean for Development

Old-Fashioned Chocolate Layer Cake

"We baked 130 cakes in search of the perfect wedge."

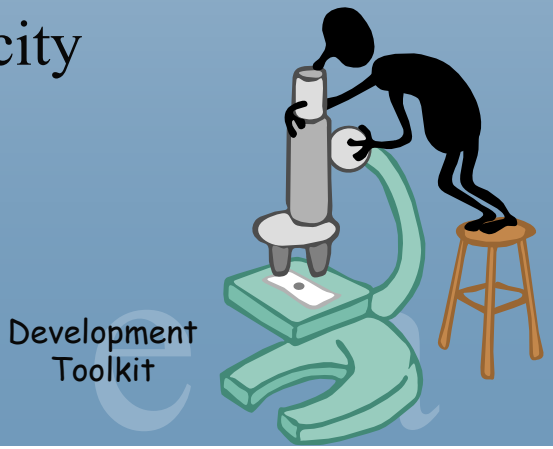
What about?

- ✓ Standard work
- ✓ Do it right the first time
- ✓ Variation
- ✓ 5 S's



Why not?

- ✓ Learning cycles
- ✓ Do it wrong lots of times
- ✓ Manage flow, not projects
- ✓ Simplicity



Software Development

1. Build the Right Thing
2. Build the Thing Right
3. Deliver (& Learn) Fast

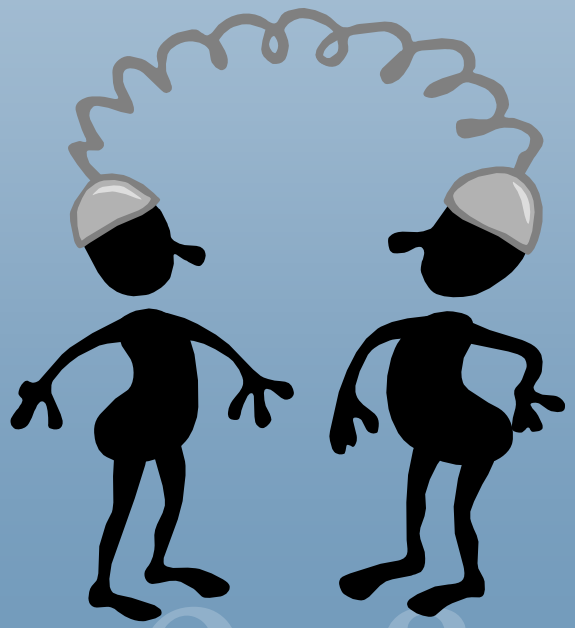
Build the Right Thing

There is nothing so useless as doing efficiently that which should not be done at all. – Peter Drucker

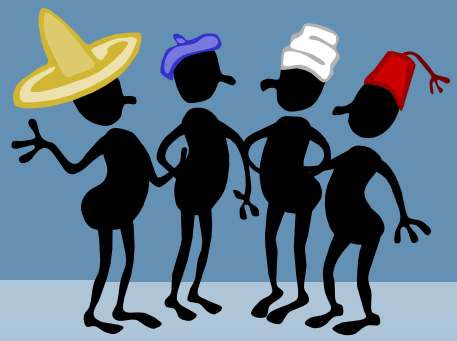
Most product failures are caused by a lack of Customers.

"Don't to what customers say they want, understand their problems and solve them." – Per Haug Kogstad, founder, Tandberg (now Cisco)

Think Like a Customer



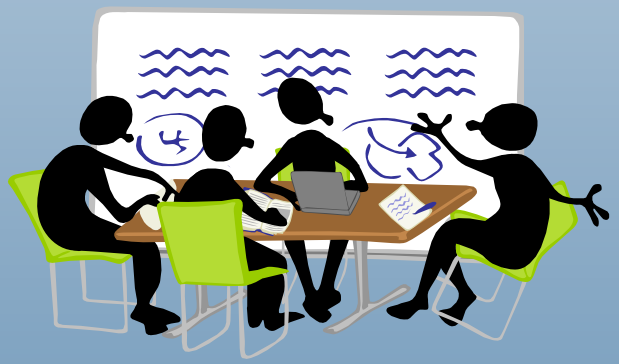
What is Design Thinking?



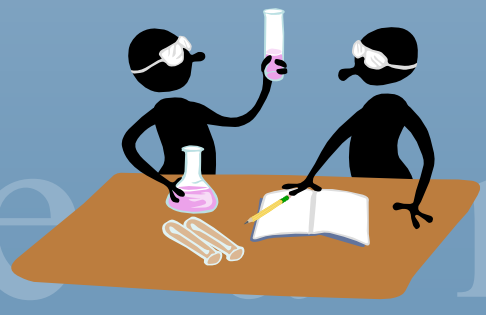
Diverse Design Team

Framing
- ✓ Observe the Situation
- ✓ Conceptualize the Problem

Ideation
- ✓ Obtain Customer Insights
- ✓ Visualize/Prototype Ideas



Experimentation
- ✓ Try Tentative Solutions
- ✓ Refine Mental Models



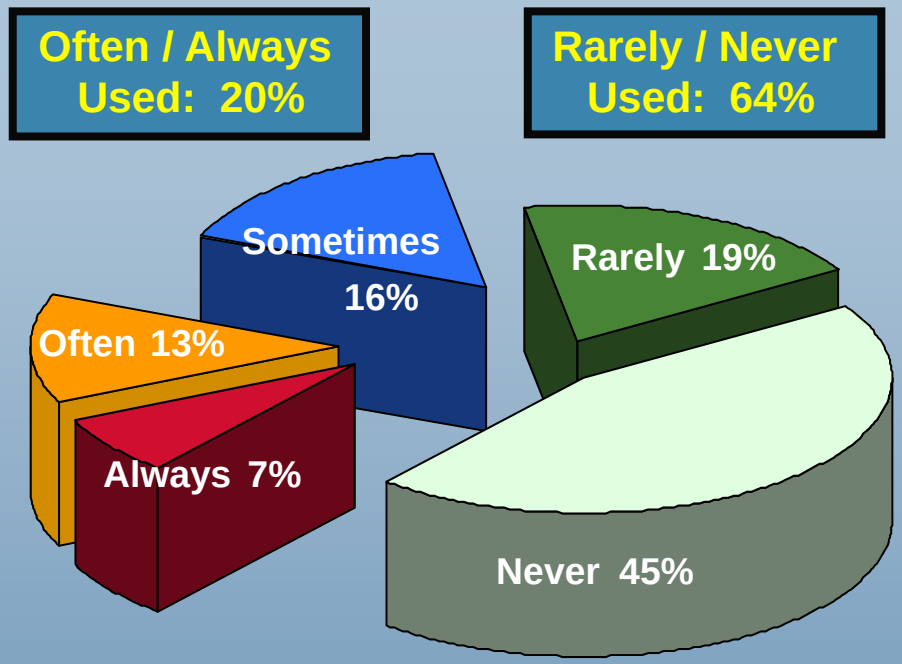
Reframe*

Iterate

*Pivot

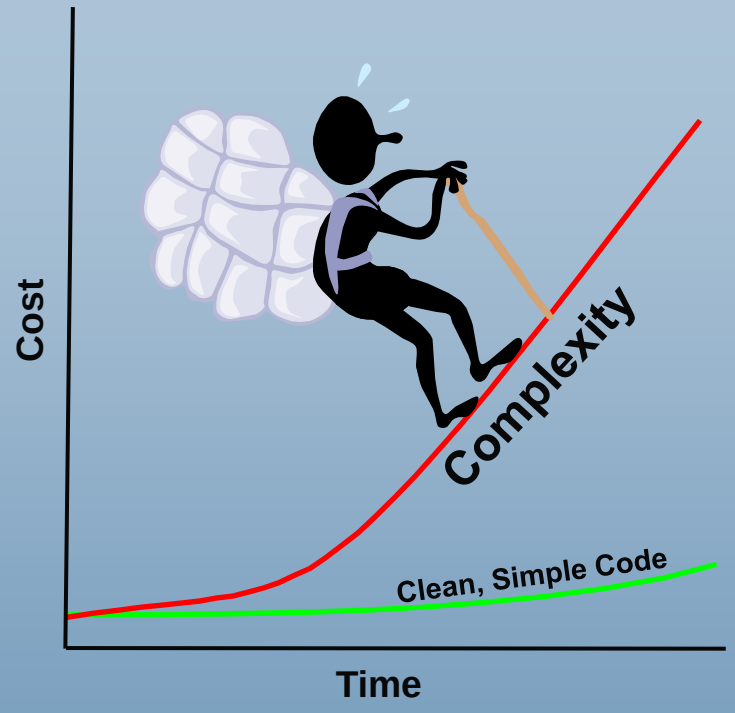
Waste 1: Extra Features

Features / Functions Used in a Typical System



Standish Group Study Reported at XP2002 by Jim Johnson, Chairman

Cost of Complexity



The Biggest opportunity for increasing Software Development Productivity: Write Less Code!

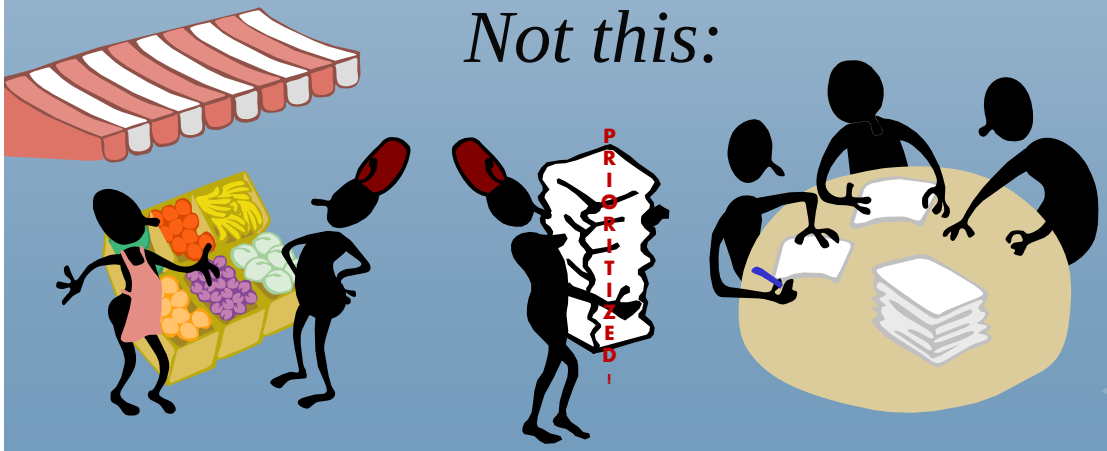
Waste 2: Handovers

*A handover occurs whenever we separate:**

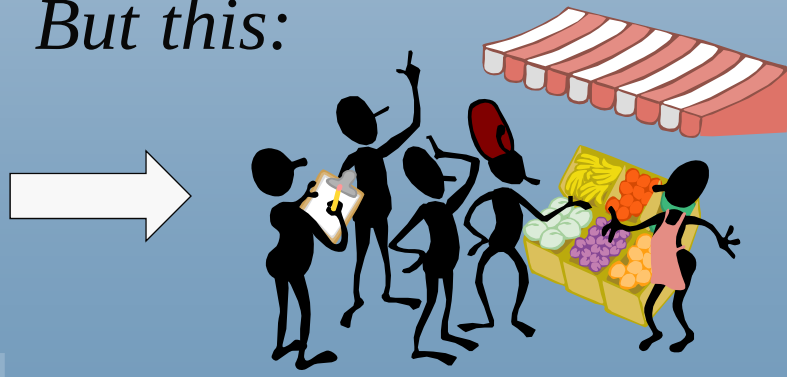
- ✓ Responsibility – What to do
- ✓ Knowledge – How to do it
- ✓ Action – Actually doing it
- ✓ Feedback – Learning from results

*Alan Ward: Lean Product and Process Development

Not this:



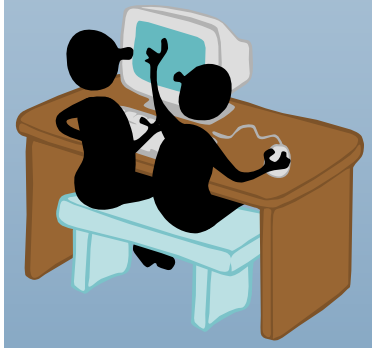
But this:

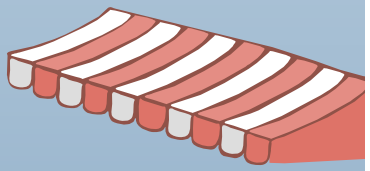


The Lean Startup

Agile Vs. Lean Startup
Adapted from similar chart posted by Joshua Kerievsky, Industrial Logic Blog† August, 2011

Agile	Lean Startup
Product Roadmap	Business Model Canvas
Product Vision	Product Market Fit
Release Plan	Minimal Viable Product
Iteration	Build-Measure-Learn Loop
Iteration Review	Persevere or Pivot
Backlog	"To Learn" List
User Story	Hypothesis
Continuous Integration	Continuous Deployment
Definition of Done	Validated Learning
Acceptance Test	Split Test
Customer Feedback	Cohort-based Metrics
On-Site Customer	"Get Out Of The Building"
Product Owner	Entrepreneur





†https://elearning.industriallogic.com/gh/submit?Action=PageAction&album=blog2009&path=blog2009/2011/agileVsLeanStartup&devLanguage=Java

Software Development

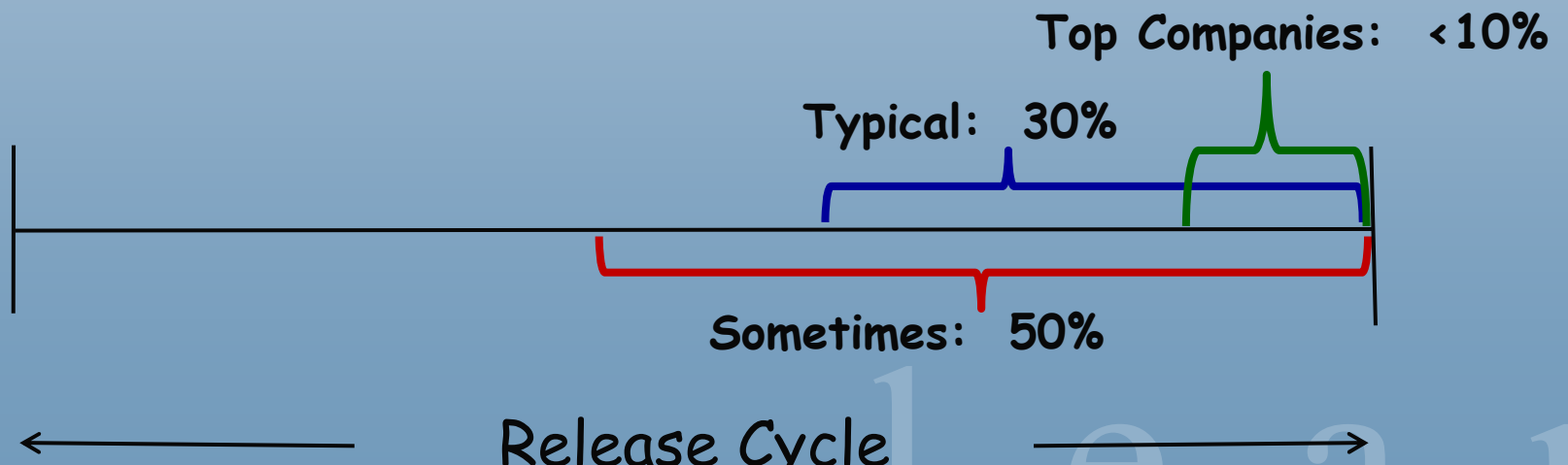
1. Build the Right Thing
2. Build the Thing Right
3. Deliver (& Learn) Fast

Build Quality In

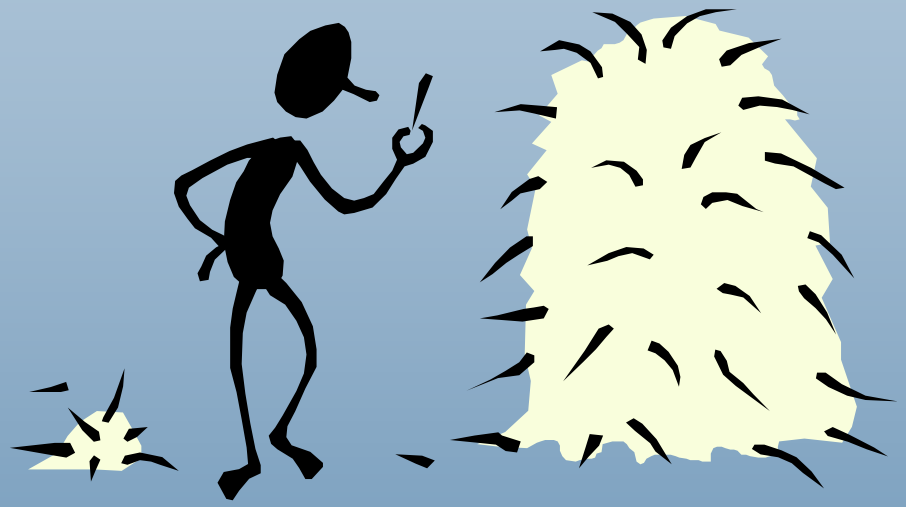
Every software development process ever invented has had the same primary goal – find and fix defects as early in the development process as possible. If you are finding defects at the end of the development process – your process is not working for you.

How good are you?

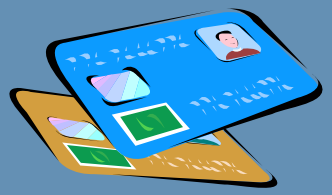
When in your release cycle do you try to freeze code and test the system?
What percent of the release cycle remains for this "hardening"?



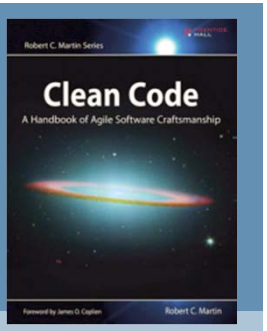
Waste 3: Defects



The Longer Defects are Undetected, the Harder They are to Find.



Waste 4: Technical Debt



Technical Debt: Anything that makes code difficult to change

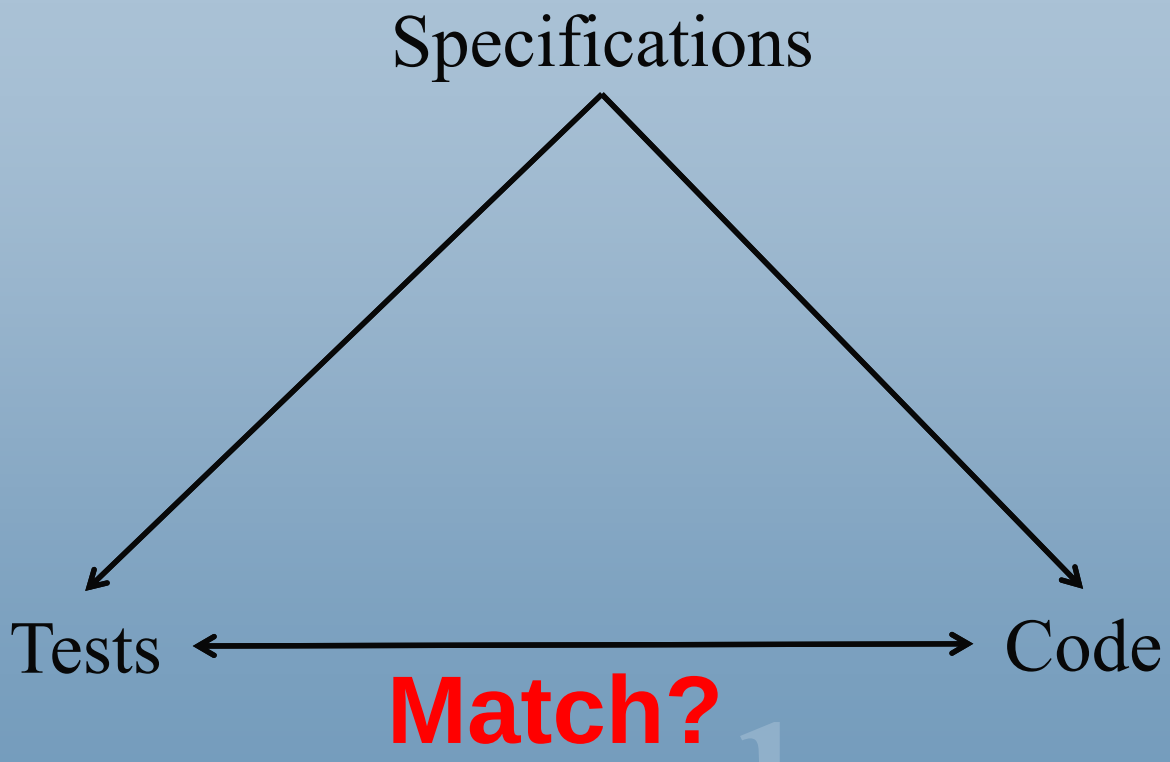
- ✓ Sloppy Code
  Code reviews ⇒ standards, quality, knowledge transfer.

- ✓ No Test Harness (=Poka Yoke)
  Code without a test harness is Legacy Code.

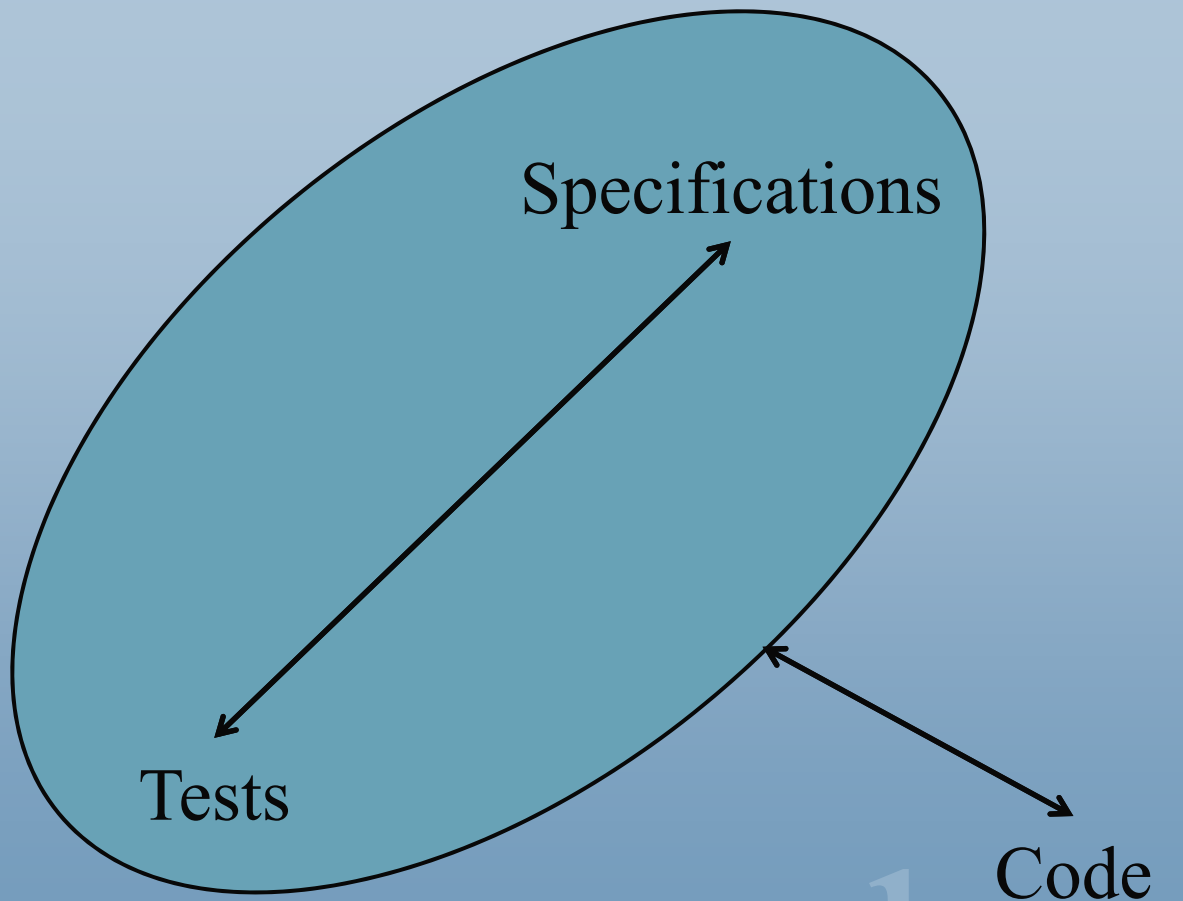
- ✓ Dependencies
  A divisible architecture is fundamental.

- ✓ Unsynchronized Code Branches
  The longer two code branches remain apart, the more difficult they are to merge together.

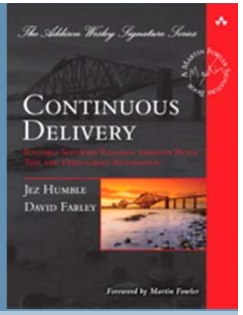
A Defect Injection Process

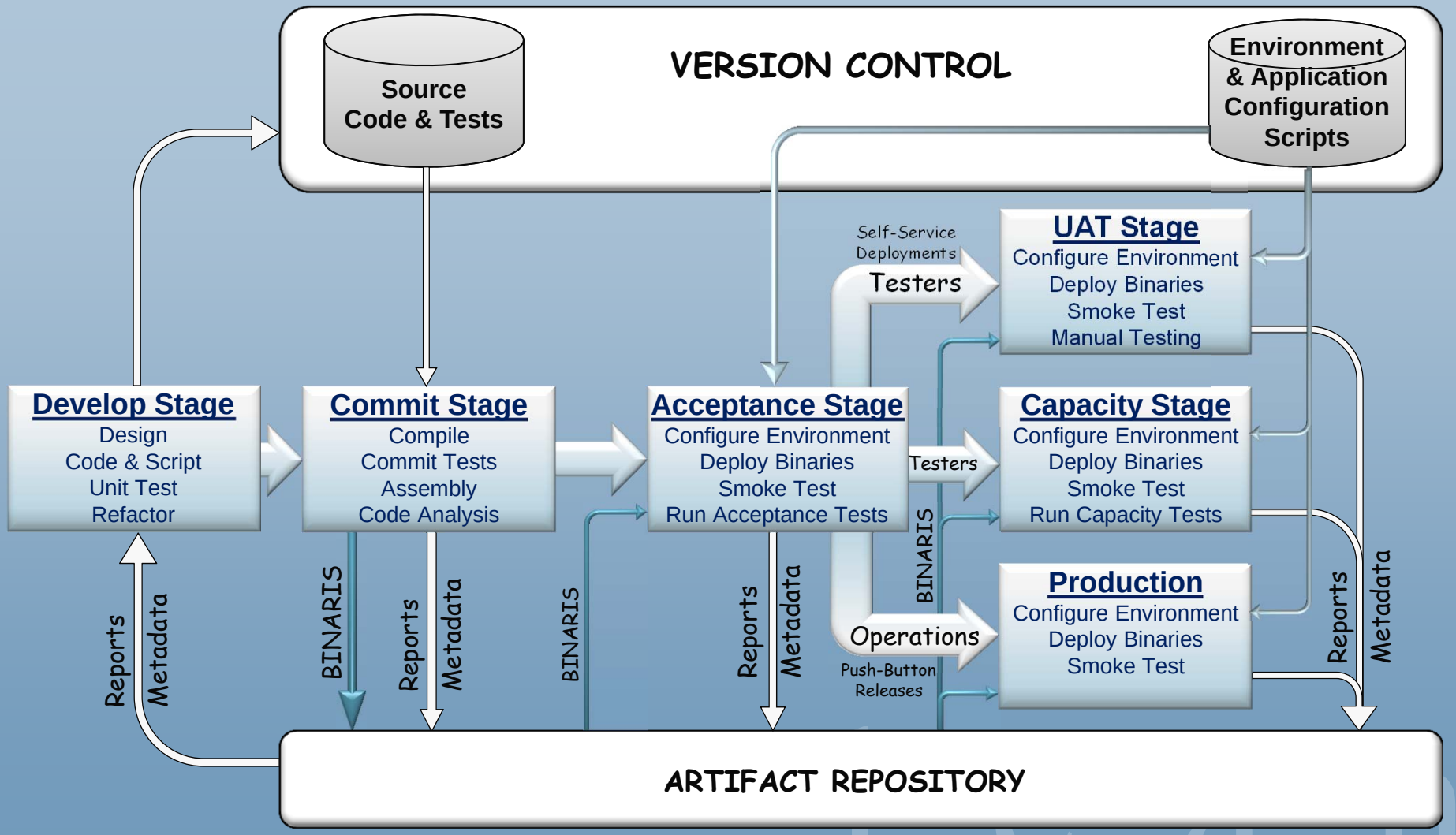


A Defect Prevention Process



Discipline on Steroids

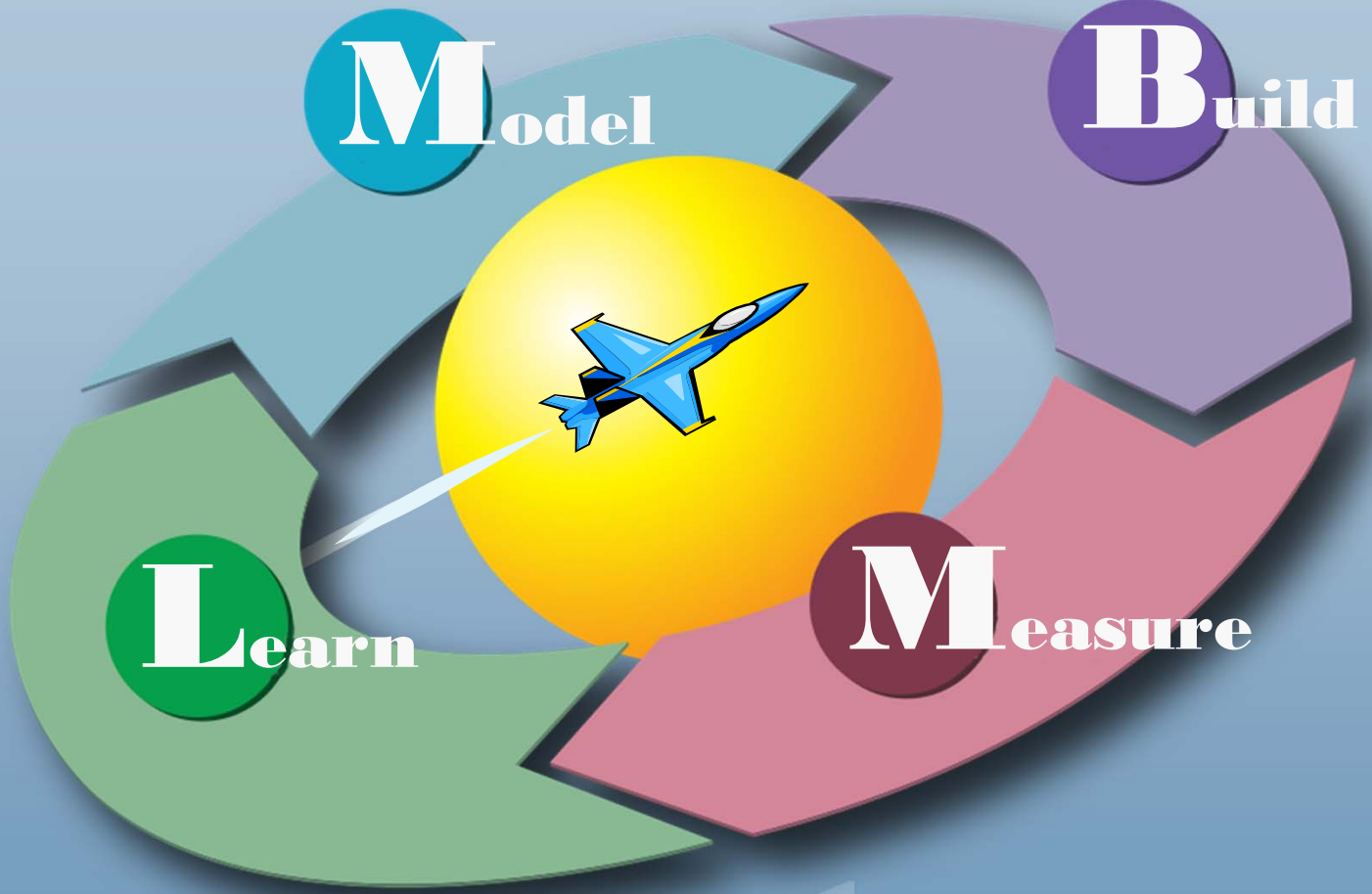




Software Development

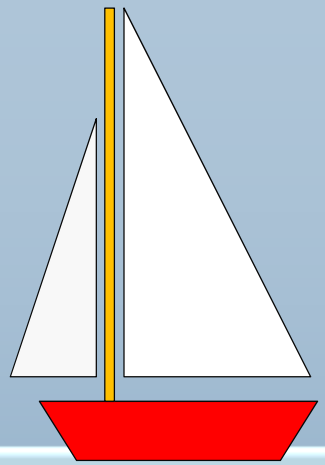
1. Build the Right Thing
2. Build the Thing Right
3. Deliver (& Learn) Fast

The Fastest Learner Wins



lean

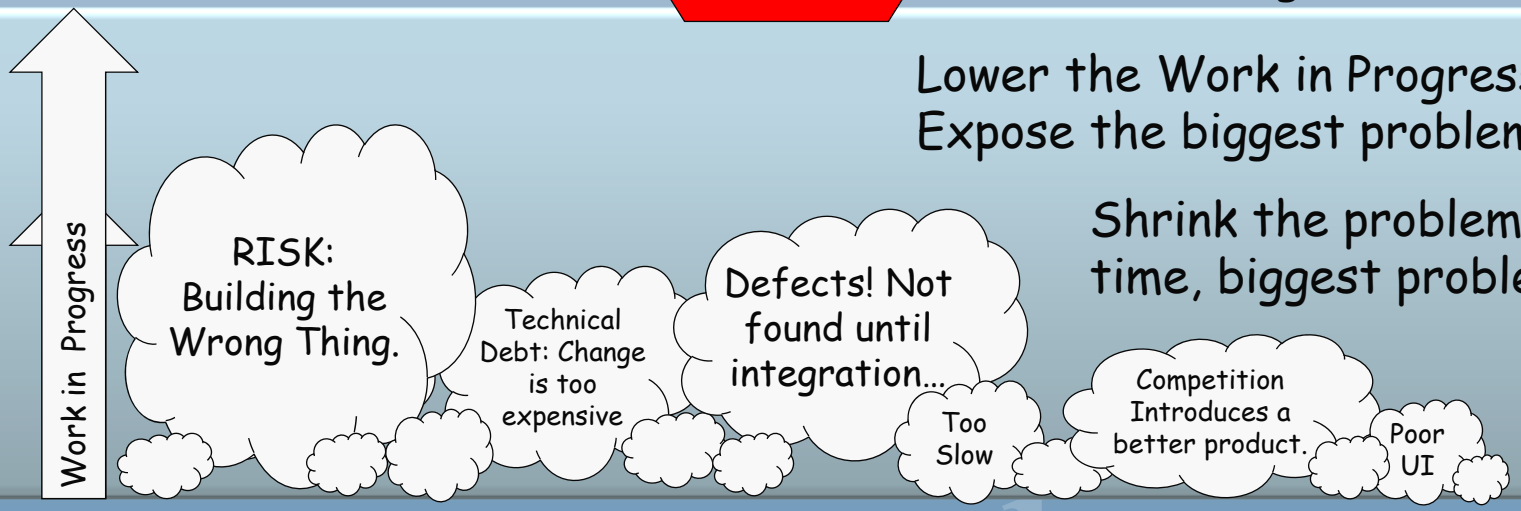
Waste 5: Work in Progress



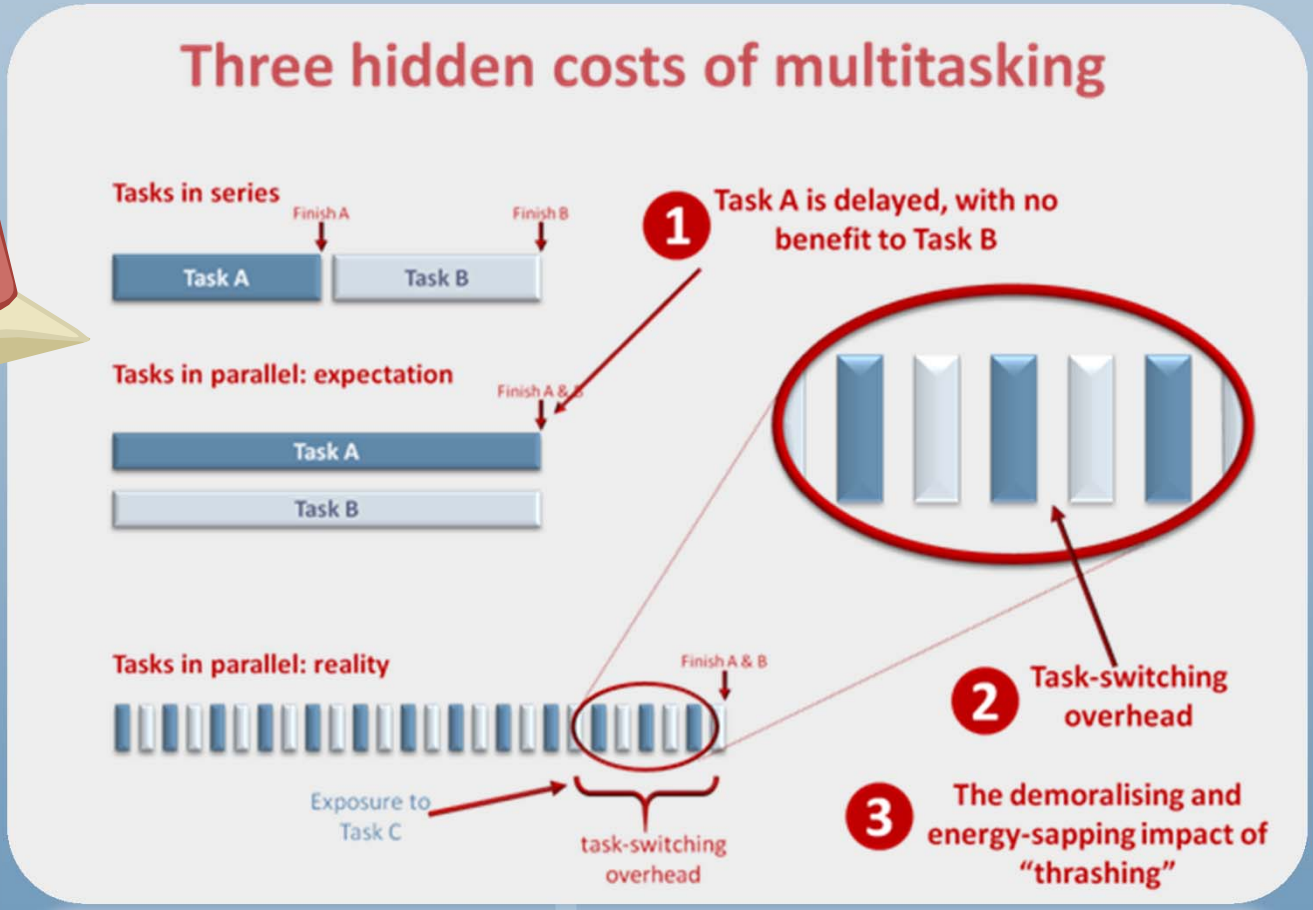
Work in Progress hides problems.

Lower the Work in Progress gradually;
Expose the biggest problems first.

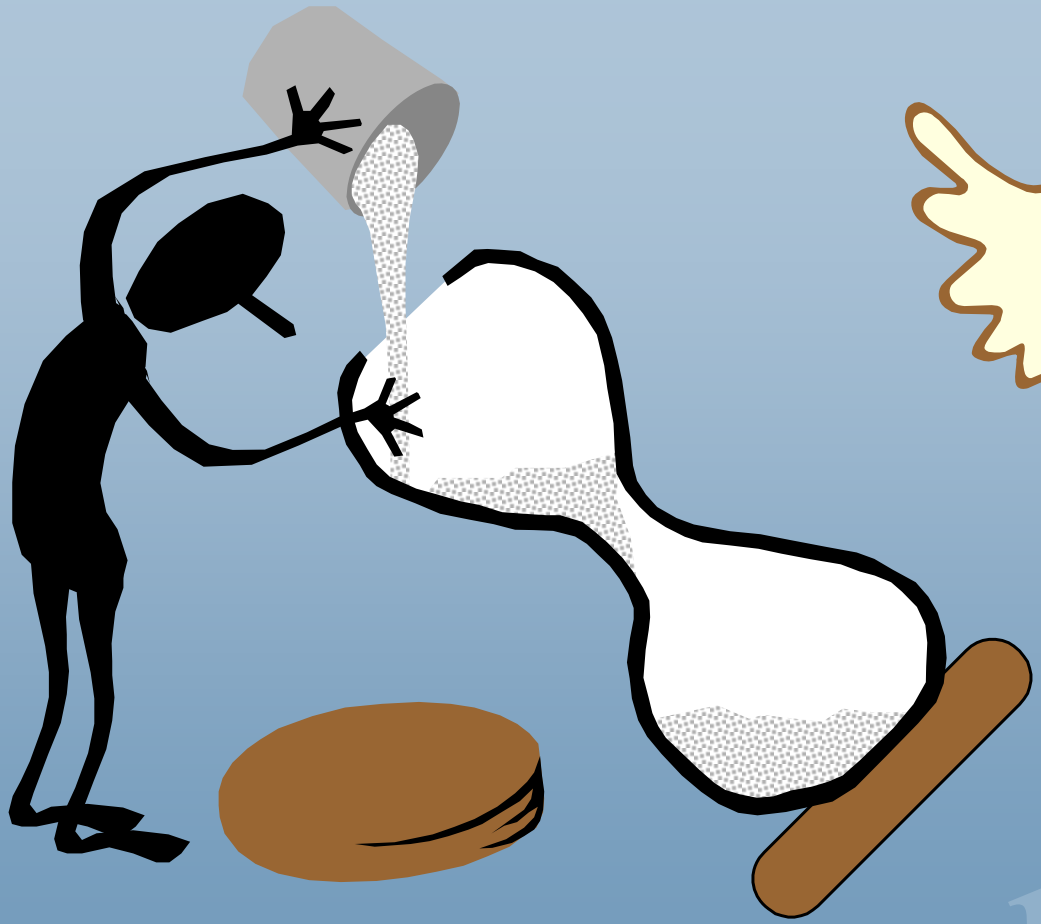
Shrink the problems one at a time, biggest problem first.

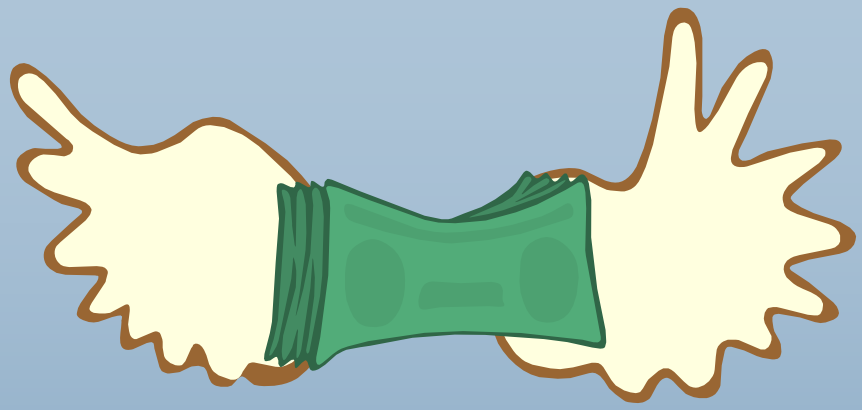


Waste 6: Task Switching

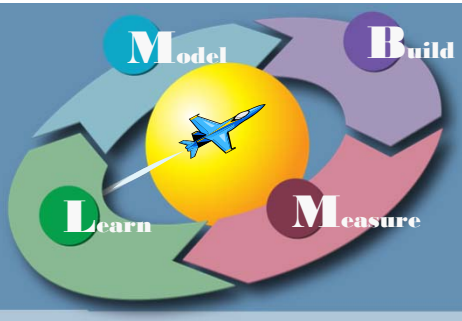


Waste 7: Delays

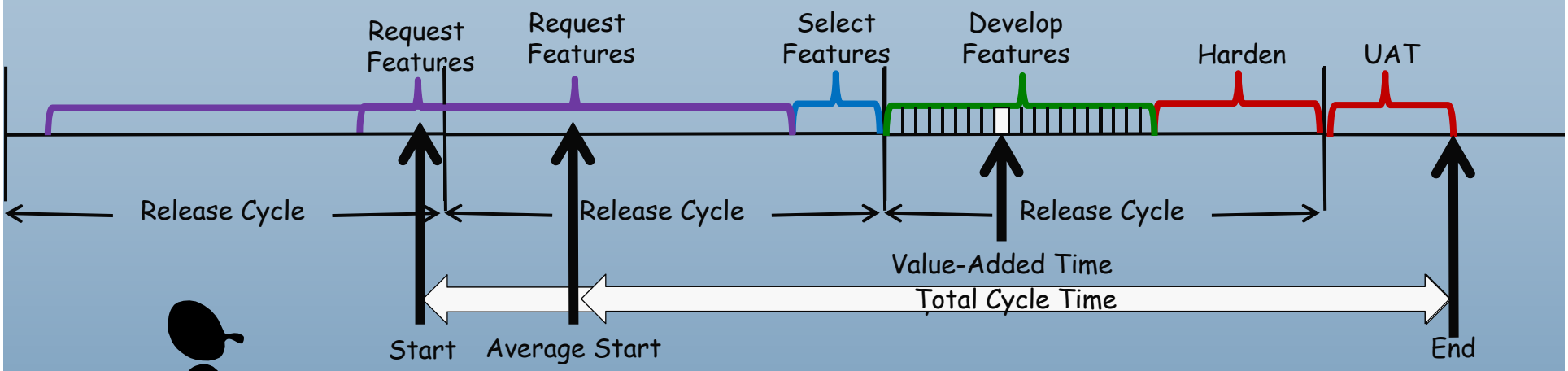


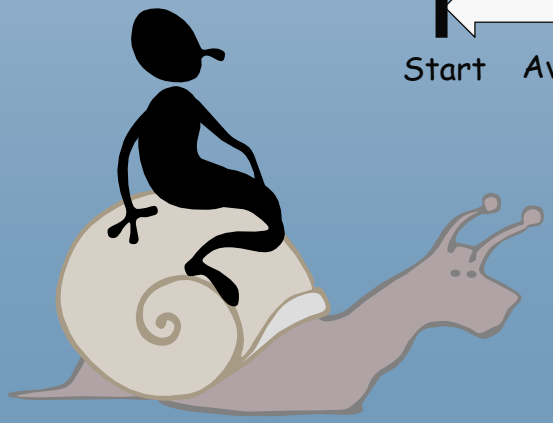


Release Cycle ≥6 Months



Quick & Dirty Value Stream Map:

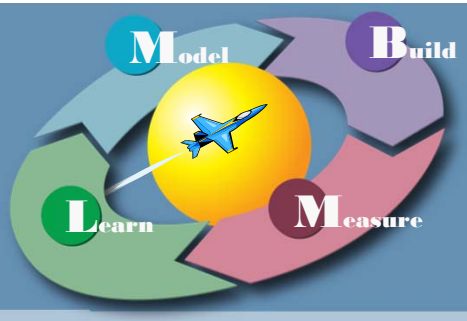




Business Model:

- Software installed at customer site
- Support each release
- Avoid releases

Release Cycle Quarterly

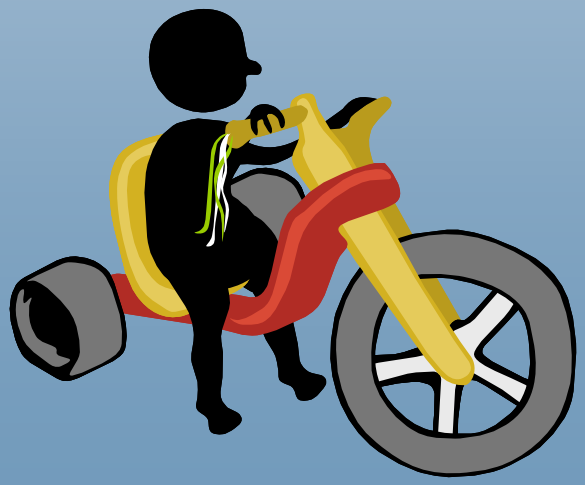


Hardening must be ≤ 2 weeks.

Typically: 2-4 week iterations

Code from each iteration goes to integration testing

Automated integration testing becomes necessary



Business issues:

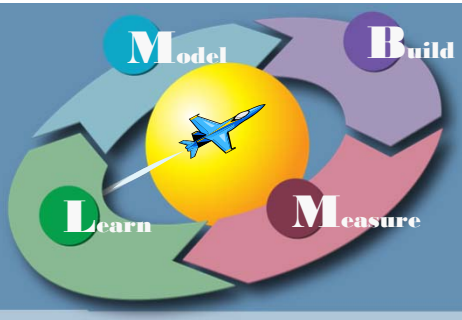
How to price and sell releases?

Which releases to support?

Supporting multiple branches can create a support nightmare

Public vs. Private releases?

Release Cycle Monthly



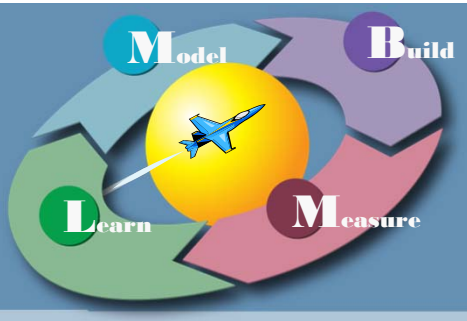
Now you need:

- ✓Cross Functional Team
- ✓Visualization
- ✓Short Daily Meetings
- ✓SBE/TDD working!
- ✓Hardening ≤ 3 days

Business Environment Works best for:

- ✓Software as a Service (SaaS)
- ✓Internal Software

Release Cycle Weekly/Daily/Continuous



Kanban works well

Iterations become irrelevant

High discipline is fundamental

Estimating is largely unnecessary

Rapid cycles of learning drive portfolio decisions

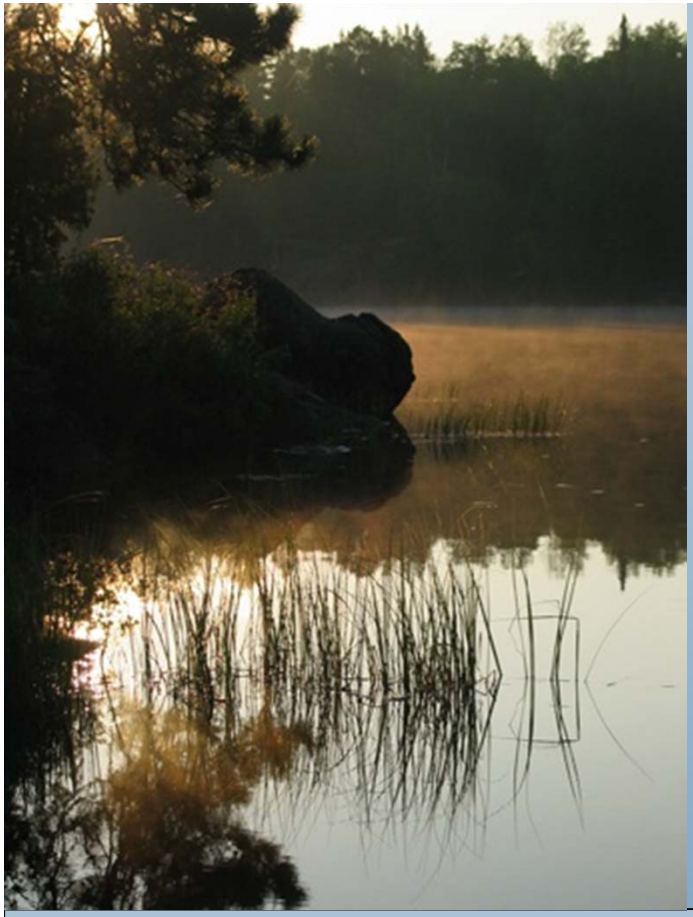
The team is everyone.

DevOps:

Test & deployment automation is essential

Business Issues:

Increasingly common in startups



l e a n
software development

Thank You!

More Information: www.poppendieck.com

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