Meeting Tight Software Schedules Through Cycle Time Reduction

Dennis J. Frailey

Raytheon Company

frailey@acm.org

Raytheon

Sponsored by ACM and Raytheon Lectureship Program

Copyright Statement

Note

The comments herein are my personal views and are not official positions of ACM or of Raytheon Company.

Outline

- Background - Why Cycle Time is Important
  - Cycle Time Reduction Issues and Examples
  - Defining Cycle Time
- Three Fundamental Problems
- Typical Causes
- Spotting the Symptoms and Opportunities
- Fixing the Problems - Five Principles
- A Few Lessons Learned

Why Cycle Time is Important

Cycle Time Gives You a Competitive Edge

- You sell your software while the competitor is still completing theirs
- You start the next software product while the competitor is still completing the current one
- Lower your costs
- Or start development later in the program cycle
- And allow less time to change requirements

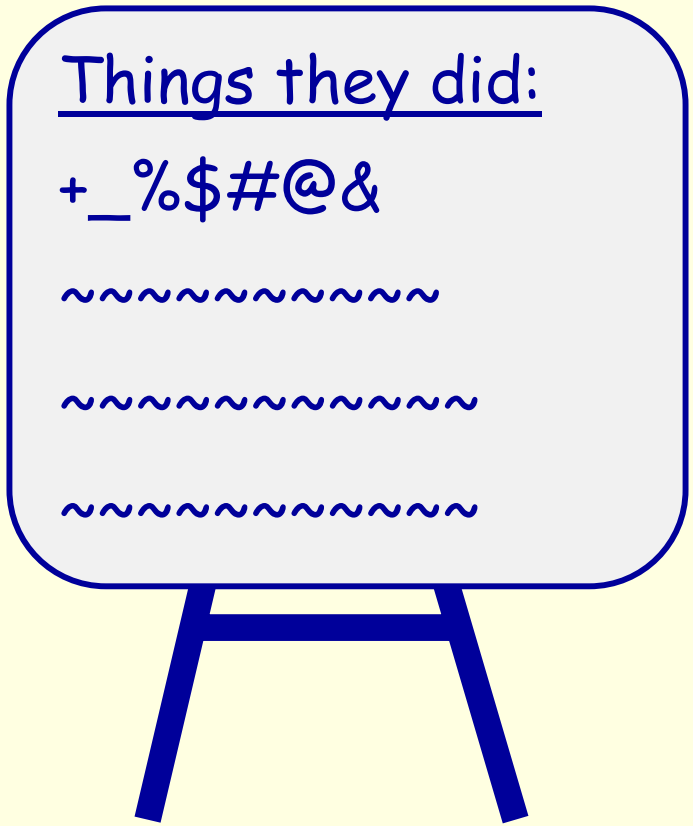
How Can Cycle Time be Improved?

- The following video illustrates how to improve cycle time
- As you watch, think of ideas that might be applicable to software development

What did they do to reduce cycle time?

Some Lessons from the Cycle Time Video

- What did they do?
- Is there a software counterpart?



Definition of Cycle Time

Cycle time is the time required to execute all activities in a process, including actual processing time AND all waiting time

Consider a "10 minute" oil change

How do You Measure Cycle Time?

STATIC CYCLE TIME

The average of the actual cycle times (CT) experienced by some number (n) of products

$$\text{Cycle Time} = \frac{CT_1 + CT_2 + CT_3 + CT_4 + \ldots + CT_n}{n}$$

- But this is not always easy to measure when many of the products are only partway through the process

  … so we need a dynamic measure

Dynamic Cycle Time

DYNAMIC CYCLE TIME

The total work in process divided by the throughput of the process

$$\text{Cycle Time} = \frac{\text{WIP (products being developed)}}{\text{THROUGHPUT (products produced/unit time)}}$$

WIP = Work in Process

For related background, see Gross and Harris, in reference list, p83.

How is Cycle Time Improved?

- Doing every process step faster?
- Working longer hours?
- Piling up work?

How is Cycle Time Improved?

- Doing every process step faster?
- Working longer hours?
- Piling up work?

Three Fundamental Problems

- Variability
  - Some parts of the process are starved while other parts are producing excessive output
  - Performance becomes inconsistent and unreliable
  - This is what causes traffic jams!
- Complex Processes
  - More work to do than is necessary
  - More opportunities to make mistakes
- Bottlenecks and Constraints
  - Things that slow everything down

Typical Causes of Cycle Time Problems

	Variability	Complexity	Bottlenecks
Misplaced Priorities	✓		✓
Incompatible Tools		✓	✓
Inefficient Procedures	✓	✓	✓
Poor Planning	✓	✓	✓
etc.	✓	✓	✓

Spotting the Symptoms and the Opportunities

- Symptom: excess WIP or "work in process"
  - work waiting to be done that is not being done -- waiting in queues instead
  - something is holding up the process
- Causes: limited capacity, poor processes, poor execution, or various barriers imposed by the organization

$$\text{Cycle Time} = \frac{\text{WIP (products being developed)}}{\text{THROUGHPUT (products produced/unit time)}}$$

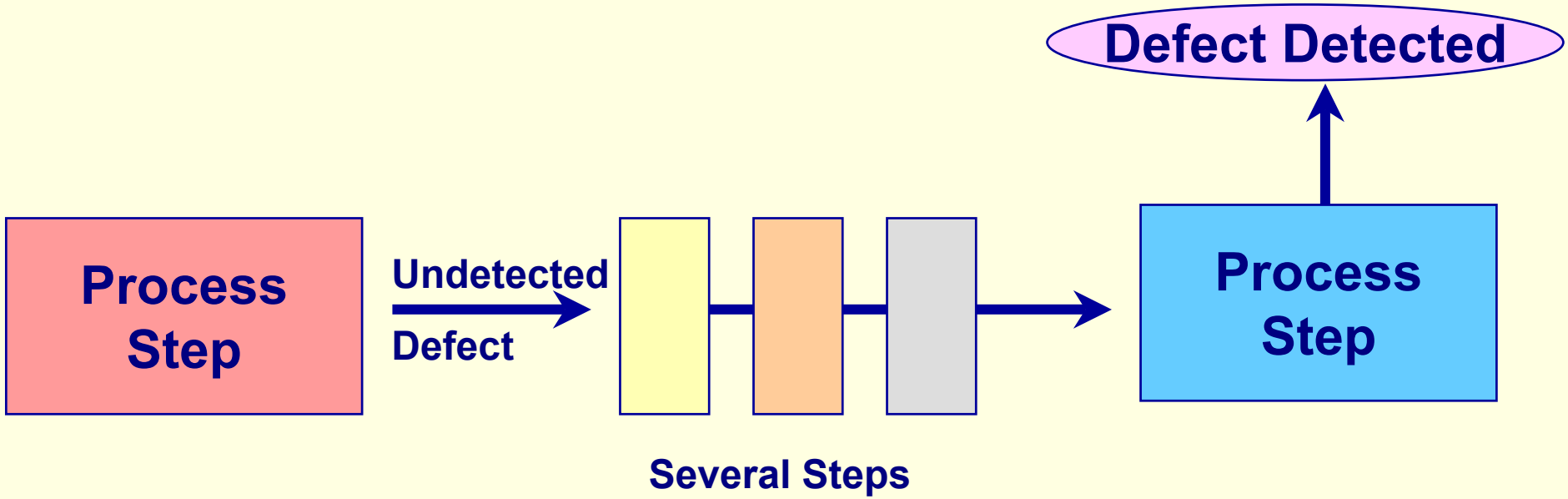
Examples of Excessive WIP for Software

- Code waiting to be tested
- Designs waiting to be coded
- Specifications waiting to be inspected
- Change requests waiting for approval
- Hundreds more…

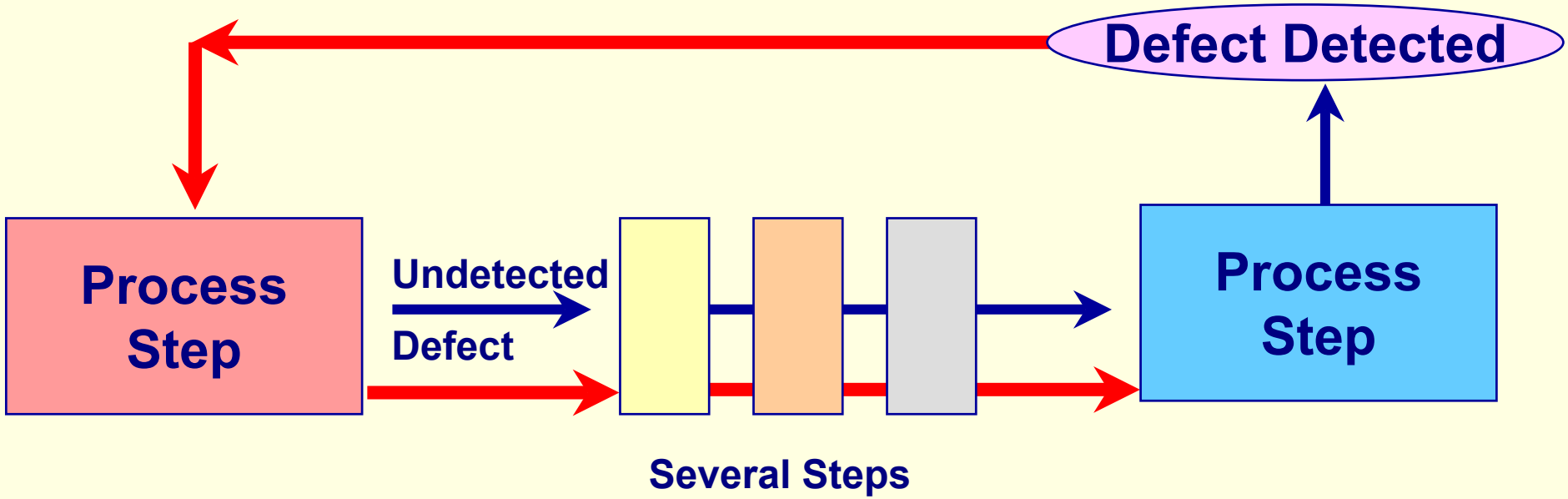
Other Symptoms of Cycle Time Problems

- Long waits and queues
- High inventory levels
- Excessive overtime
- Rework / scrap

Reducing Rework - The Impact of Defects on Cycle Time



Doing it Over Again



Rework costs money and time

Look for Rework

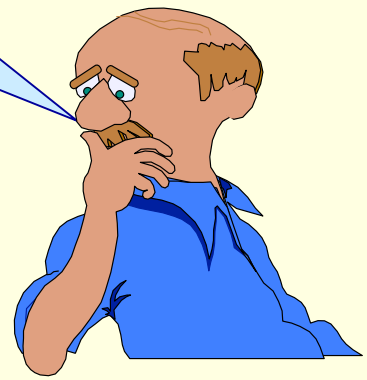
- REWORK is anything you do because you didn't do it right the *first time*
  - debugging
  - correcting documentation
  - correcting designs
  - correcting requirements
  - retesting
  - responding to customer complaints

Isn't Rework Inevitable?

- SOME rework is necessary, *but most is not*
- Total rework is a measure of *process efficiency*
- You probably have a *lot more rework* than you think

Fixing the Problems -- Five Principles of Cycle Time Improvement

Warning … some of these are counter-intuitive



Where are the Opportunities?

- ~30% of the improvement comes from technical changes
  - Process changes
  - Tool changes
  - Changing rules and operations
- ~70% of the improvement comes from organizational, cultural and environmental changes, such as
  - Education
  - Communication
  - Management
  - Teamwork

Principle #1

Look at the Entire Process

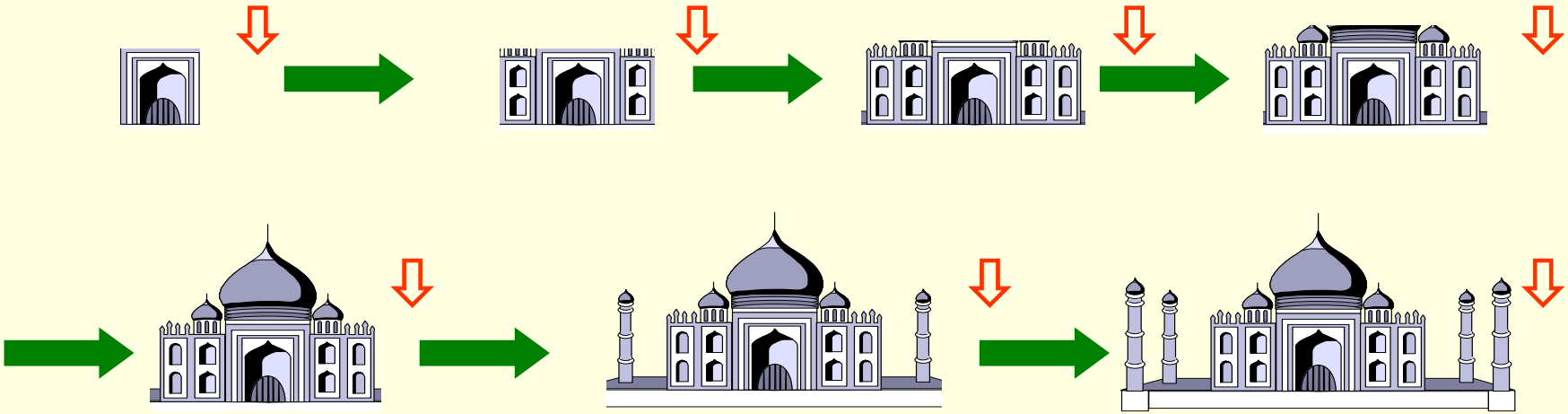
- Don't optimize only your local part of the process
- Speeding up every step of the process will cost a lot and will not help as much as speeding up the bottlenecks
- Beware of inappropriate reward systems!

Principle #2

Gain from Cycles of Learning

Sometimes it is better to do the job many times, in small chunks, than to do it all at once

Changes are received and processed here (⇩)



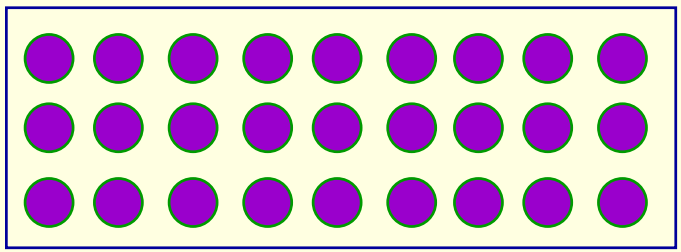
Ways to Implement the Cycles of Learning Principle

- Try a new compiler or debugging tool on one module first to see how well it works
  - Then decide whether to use it on more modules
- Write one module using the new language
  - Then decide whether to use the new language on a larger scale
- Take one subset of the features through the whole process flow first to work out the quirks
- Test server performance using dummy data

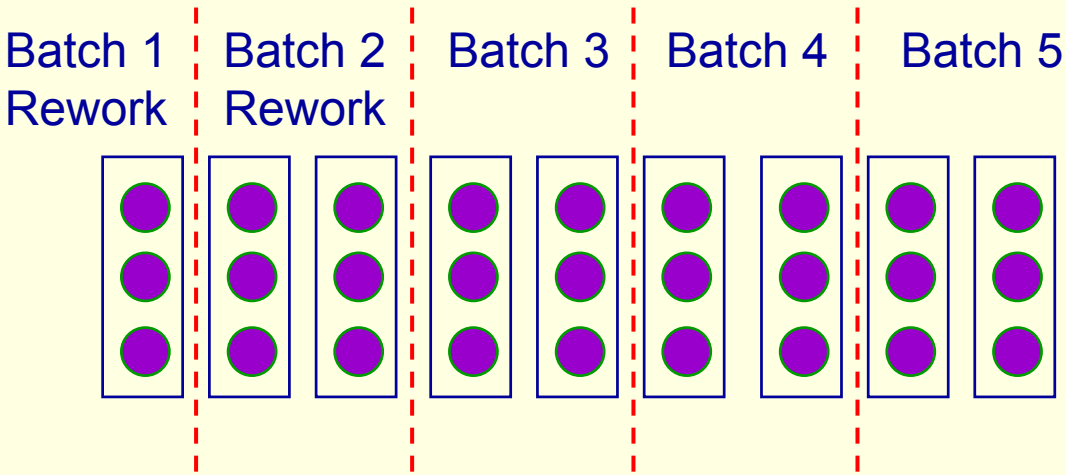
Principle #3

Use Small Batches

- Important when requirements change a lot or the process is new



The work to be done



Small batch principle

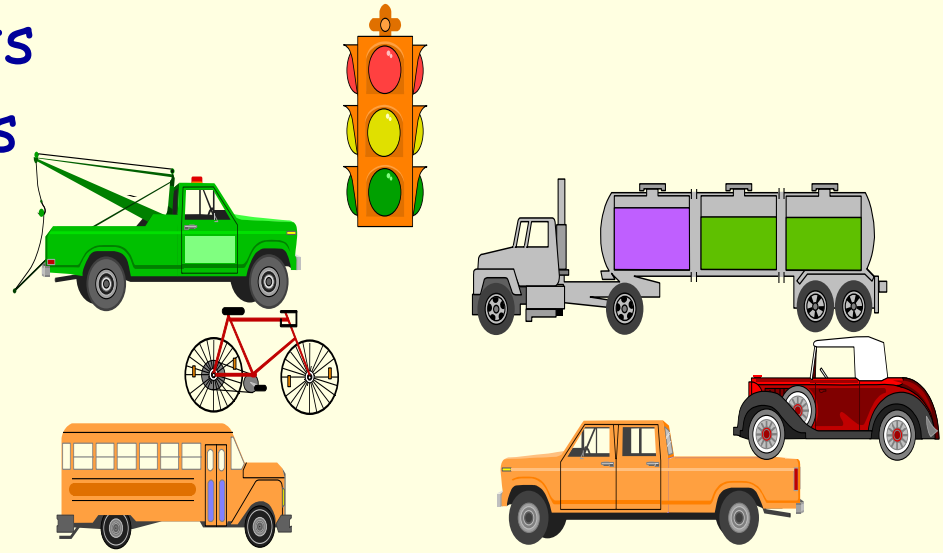
- allows for finding errors and requirements changes without large amounts of rework

Principle #4

Achieve Smooth Flow

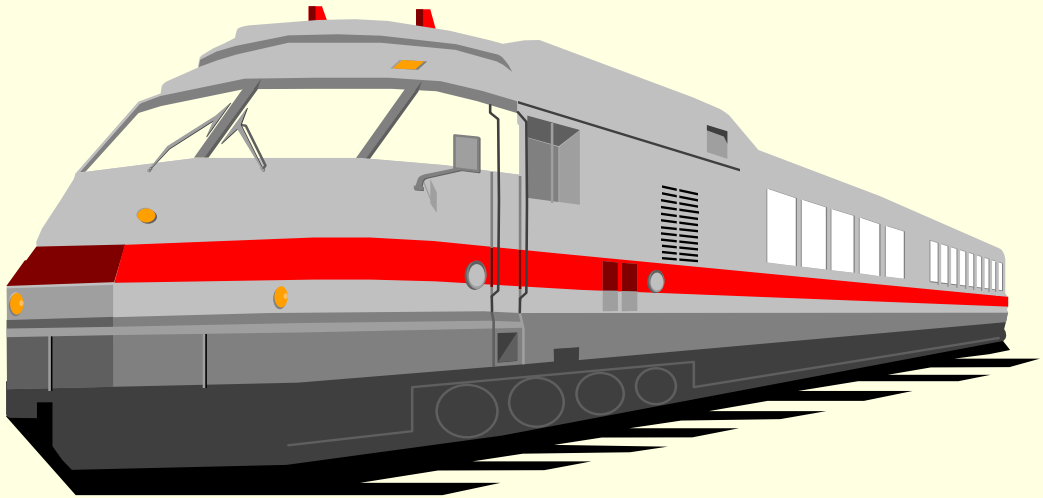
The typical process runs unevenly, like vehicles on a city street

- Lots of entrances and exits
- Vehicles of different sizes and speeds
- Some drivers uncertain of what they want to do
- Lots of stoplights to "control" the flow (mainly to prevent collisions)
- Note: streets are usually crowded



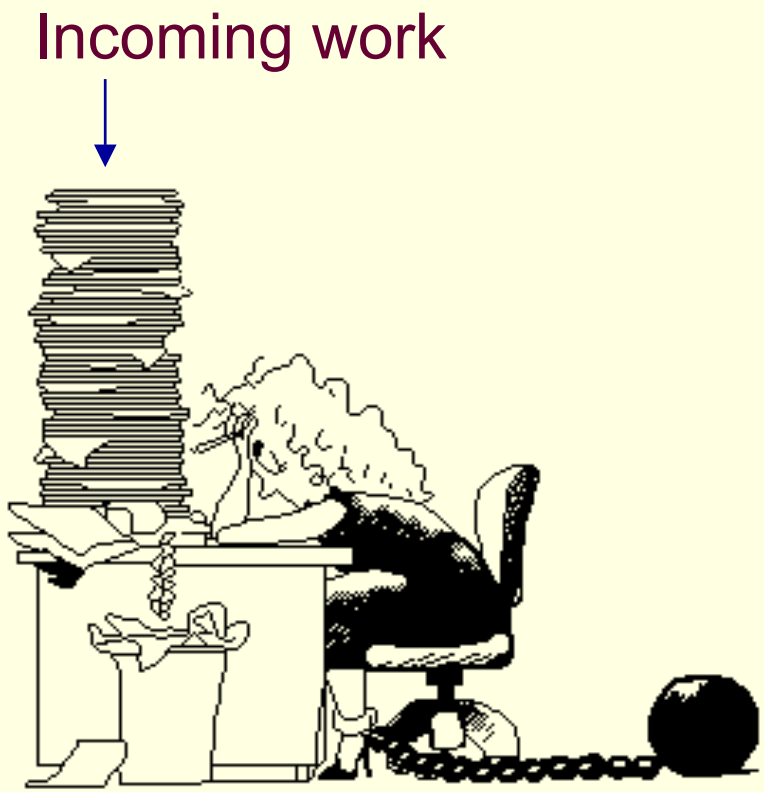
The Ideal is Smooth Flow

- The ideal process flows smoothly, like a train running on tracks
  - Note: tracks are empty most of the time



What Prevents Smooth Flow?

- Bottlenecks and constraints that lead to:
  - **Queues and waits**
  - **Work in process**
- For example:
  - Work piling up
  - Machines or software not being available
  - Excessive approval requirements
  - People pulled off projects
  - Excessive rework
  - Product stuck in test



Principle #5

Avoid the Naïve, Obvious and Wrong Solutions to Cycle Time Problems

- Such as …
  - Shortening the longest step of the process
  - Shortening every step of the process
  - Cutting the overhead without assessing the impact

A Few Lessons Learned

Be Careful Whom you Reward

- Know the difference between *busy* and *productive*
- Examine the *value produced*, not the effort spent

The Tortoise Sometimes Beats the Hare

Independent Observers May See Problems and Opportunities the Best

- Practitioners generally focus on their work and on what they THINK is happening rather than on what IS happening
  - They tend not to see all of the waits, queues, etc. that they cause themselves
  - Their perception of how they spend their time is generally incorrect
  - They are too busy getting the job done to see how they might improve it

The Athletic Coach Analogy

- Just as athletes rely on coaches, software engineers need to learn to trust in others to observe and help them do better

Software Developers Are Accustomed to Improving Cycle Time

- Think of your software development process as a large computer program that runs too slow.
  - How would you make it run faster?
- Imagine how you would speed up a computer program …….
- Then draw analogies to the software development process …
- And improve the process the way you would improve a program

Summary

- Background - Why Cycle Time is Important
  - Cycle Time Reduction Issues and Examples
  - Defining Cycle Time
- Three Fundamental Problems
- Typical Causes
- Spotting the Symptoms and Opportunities
- Fixing the Problems - Five Principles
- A Few Lessons Learned

References

- Deming, W. Edwards, *Out of the Crisis*, MIT Press, 1982.
- Goldratt, Eliyahu M. & Jeff Cox, *The Goal*, (North River Press, 1984.) Also, *Theory of Constraints*, *It's Not Luck*, and *Critical Chain Management*.
- Gross and Harris, *Fundamentals of Queueing Theory* (Wiley).
- Swartz, James B., *The Hunters and the Hunted*, (Portland, Oregon, Productivity Press, 1994) ISBN 1-56327-043-9.

END

Questions?
Comments?