Project Planning

Kenneth M. Anderson
University of Colorado, Boulder

CSCI 5828 — Lecture 7 — 02/02/2010

Goals

- Review material from Chapter 3 of Pilone & Miles
  - Customer Priorities
  - Milestones
  - The dangers of adding more people
    - Tar Pit and the Mythical Man-Month
  - Velocity
  - Big Board

Project Planning (I)

- Or, what to do if your estimate is too big?
  - In lecture 5, we looked at gathering requirements, creating user stories and assigning estimates to those stories
  - The goal: to create a total estimate for the project
  - You then deliver this estimate to the customer and see if it meets their expectations
    - Note: the techniques described in lecture 5 are not the entire story, we'll get more detail about we actually need to do to create an accurate estimate as we move forward

Project Planning (II)

- In the Orion's Orbits example, the answer was clear
  - Our estimate: 489 days (~1.85 years of development time!!)
  - Customer's Ideal Deadline?
    - 90 days
      - (sigh)

Project Planning (III)

- What to do?
  - Scope the Problem
    - Focus on most critical functionality and see if customer is willing to focus on that subset
    - In general, “scope the problems” means drop features until the remaining features can be completed by the original due date
      - In this example, it means “drop/delay features until a system that meets the customer’s most critical needs can be completed by the customer’s due date”
- Who does the scoping? The customer

Milestone 1.0

- In particular, we are attempting to define what features will go into “milestone 1.0”
  - Milestone 1.0 == first major release to the customer
    - In iterations, you show software for feedback but do not generally deploy the software for production use
    - With milestones, you are delivering software that will go into production use

Milestone 1.0 Do's and Don'ts

- Do balance functionality with customer impatience
  - Help customer understand what can be done before the deadline
  - Help them understand that features are being delayed not dropped
- Don't get caught planning nice-to-haves
  - You need to focus on what's needed: mission critical fun.
- Don't worry about length (yet)
  - You're trying to understand your customer's priorities

Sanity Check

- Once you have identified the features that need to go into Milestone 1.0, recheck your estimate
  - In the book, since you have less features, the new estimate comes to 273 days (3/4 of a year)
  - You still have 90 days to complete the work
  - And we are assuming a team size of 1 person
- In this situation, we would be forced to reprioritize with the customer and cut functionality to the bone
- OR…

Add More People

- … we could add more people!
- Lets increase the team size to 3 people
  - 273 / 3 = 91 days of work and we have 90 days left
    - That should do the trick assuming a few sleepless nights as the deadline approaches, right?
- WRONG!
  - First, we have 90 calendar days, not 90 work days!
    - Recall that we get roughly 20 works days per month
    - So a team of 3 can accomplish roughly 180 days worth of work over 90 calendar days ASSUMING ALL GOES WELL

Wrong, continued

- Second, you can't assume that the customer won't change things on you as you move forward
  - even with all this angst about cutting back from the "blue sky" version of the project to arrive at milestone 1.0
- Third, you can't assume that the two new developers will be up to speed on the project and ready to put full productive work days into the project on day one
  - With three people, we now have
    - two people to train and get ready to work on the project
    - three communication paths to manage (previously zero)

Indeed

It's time for a

Brooks Intervention

(Fred Brooks, that is.)

Mythical Man-Month (I)

- Famous essay (and the title of Brooks SE book)
- It looks at the unit of the man-month
  - sometimes used by management to schedule large projects
- I will henceforth refer to the man-month as the person-month (which is what it should have been called originally)

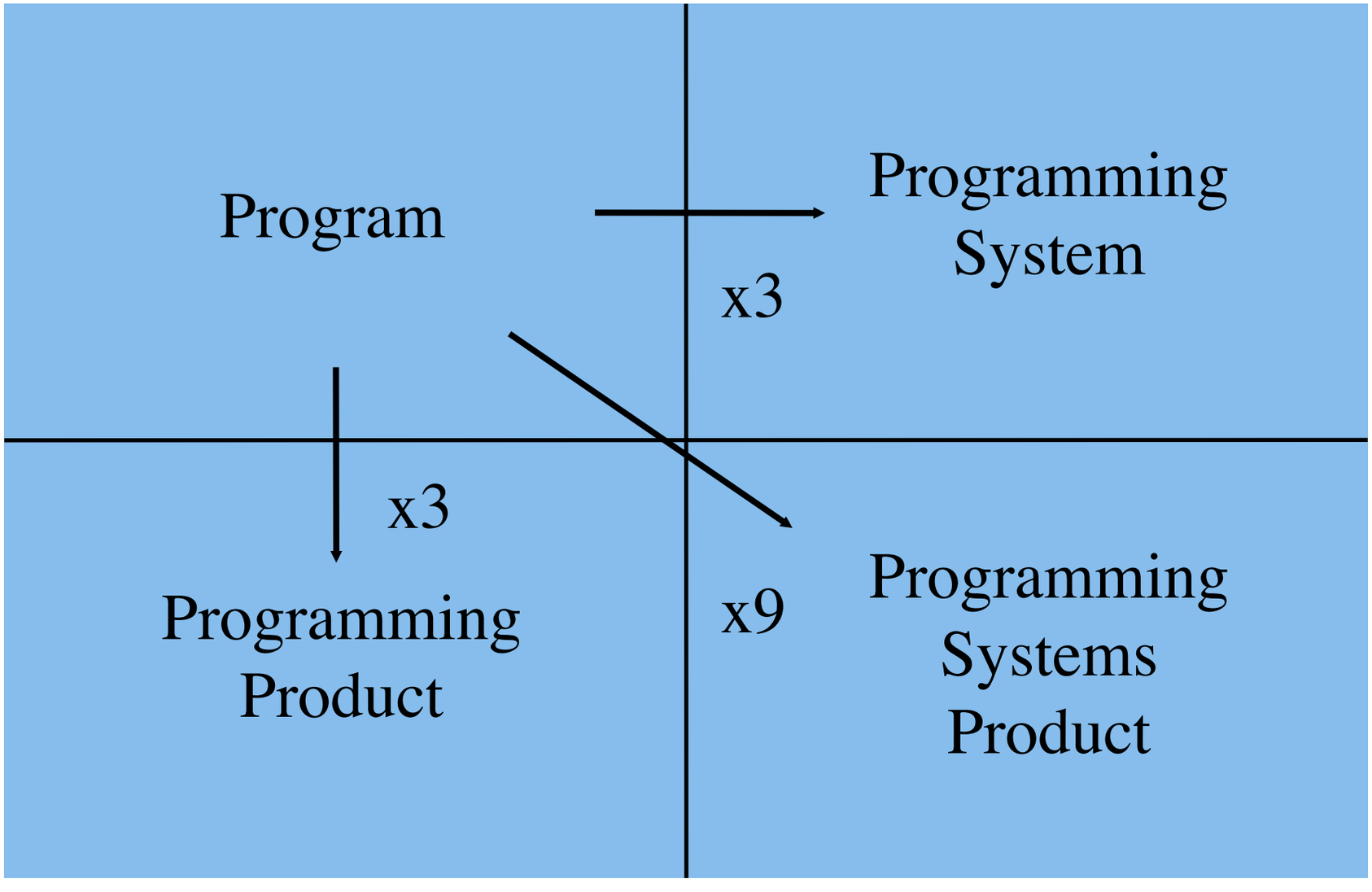
But First: The Tar Pit

- Developing large systems is “sticky”
  - Projects emerge from the tar pit with running systems
    - But most missed goals, schedules, and budgets
    - “No one thing seems to cause the difficulty--any particular paw can be pulled away. But the accumulation of simultaneous and interacting factors brings slower and slower motion.”

The Tar Pit, continued

- The analogy is meant to convey that
  - It is hard to discern the nature of the problem(s) facing software development
- Brooks begins by examining the basis of software development
  - e.g. system programming

Evolution of a Program



What makes programming fun?

- Sheer joy of creation
- Pleasure of creating something useful to other people
- Creating (and solving) puzzles
- Life-Long Learning
- Working in a tractable medium
  - e.g. Software is malleable

What's not so fun about programming?

- You have to be perfect!
- You are rarely in complete control of the project
- Design is fun; debugging is just work
- Testing takes too long!
- The program may be obsolete when finished!

Why are software project's late?

- Estimating techniques are poorly developed
- Our techniques confuse effort with progress
  - The Mythical Man-Month
- Since we are uncertain of our estimates, we don't stick to them!
- Progress is poorly monitored!
- When slippage is recognized, we add people
  - "Like adding gasoline to a fire!"

Optimism

- "All programmers are optimists!"
  - "All will go well" with the project
    - Thus we don't plan for slippage!
  - However, with the sequential nature of our tasks, the chance is small that all will go well!
- One reason for optimism is the nature of creativity
  - idea, implementation, and interaction
  - The medium of creation constrains our ideas
    - In software, the medium is infinitely tractable, we thus expect few problems in implementation, leading to our optimism

Mythical Man-Month (II)

- The unit of the person-month implies that workers and months are interchangeable.
  - However, this is only true when a task can be partitioned among many workers with no communication among them!
- Brooks points out that **cost** does indeed vary as the product of the number of workers and the number of months. Progress does not!
  - When a task is sequential, more effort has no effect on the schedule
    - “The bearing of a child takes nine months, no matter how many women are assigned!”

Mythical Man-Month (III)

- And, unfortunately, many tasks in software engineering have sequential constraints
  - Especially debugging and system testing
    - (Note: open source development challenges this notion a bit)

Mythical Man-Month (IV)

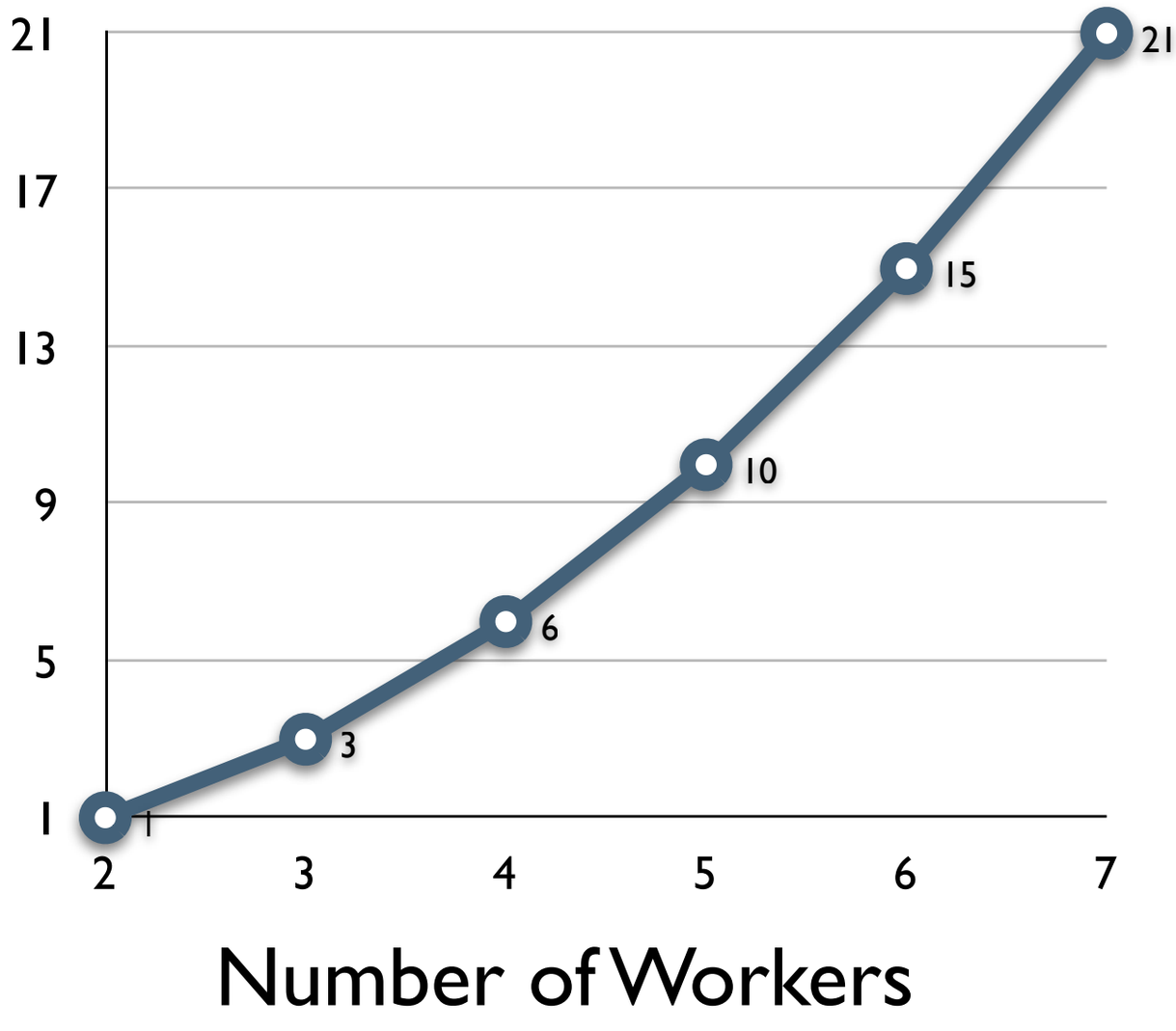
- In addition, most tasks require communication among workers
- In a software dev. project, communication consists of
  - training, and
  - sharing information (intercommunication)

Mythical Man-Month (V)

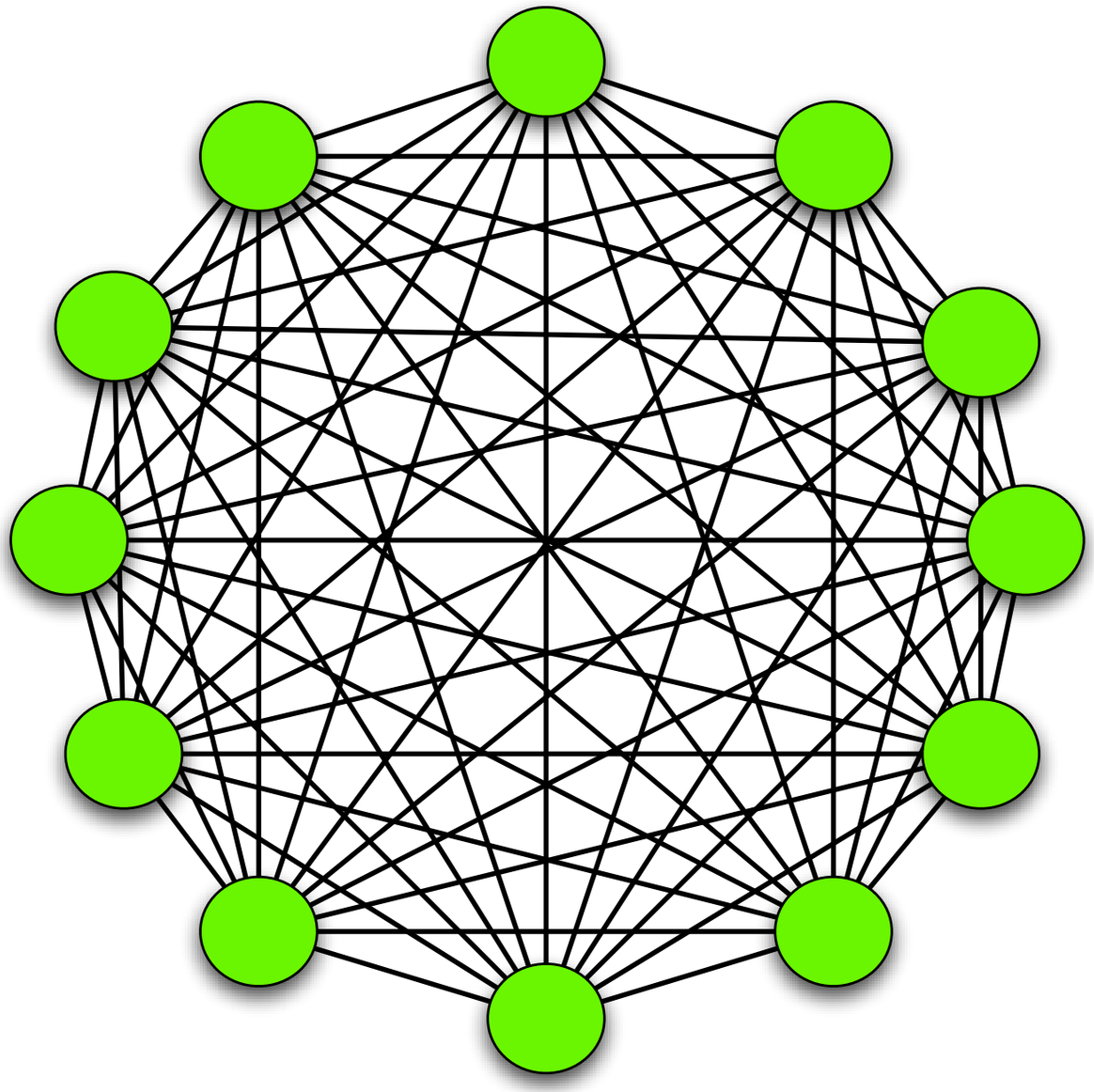
- training will effect effort at worst linearly
  - (i.e. if you have to train N people individually, it will take N*trainingTime minutes to train them)
- intercommunication adds n(n-1)/2 to the effort
  - if each worker has to communicate with every other worker

Mythical Man-Month (VI)

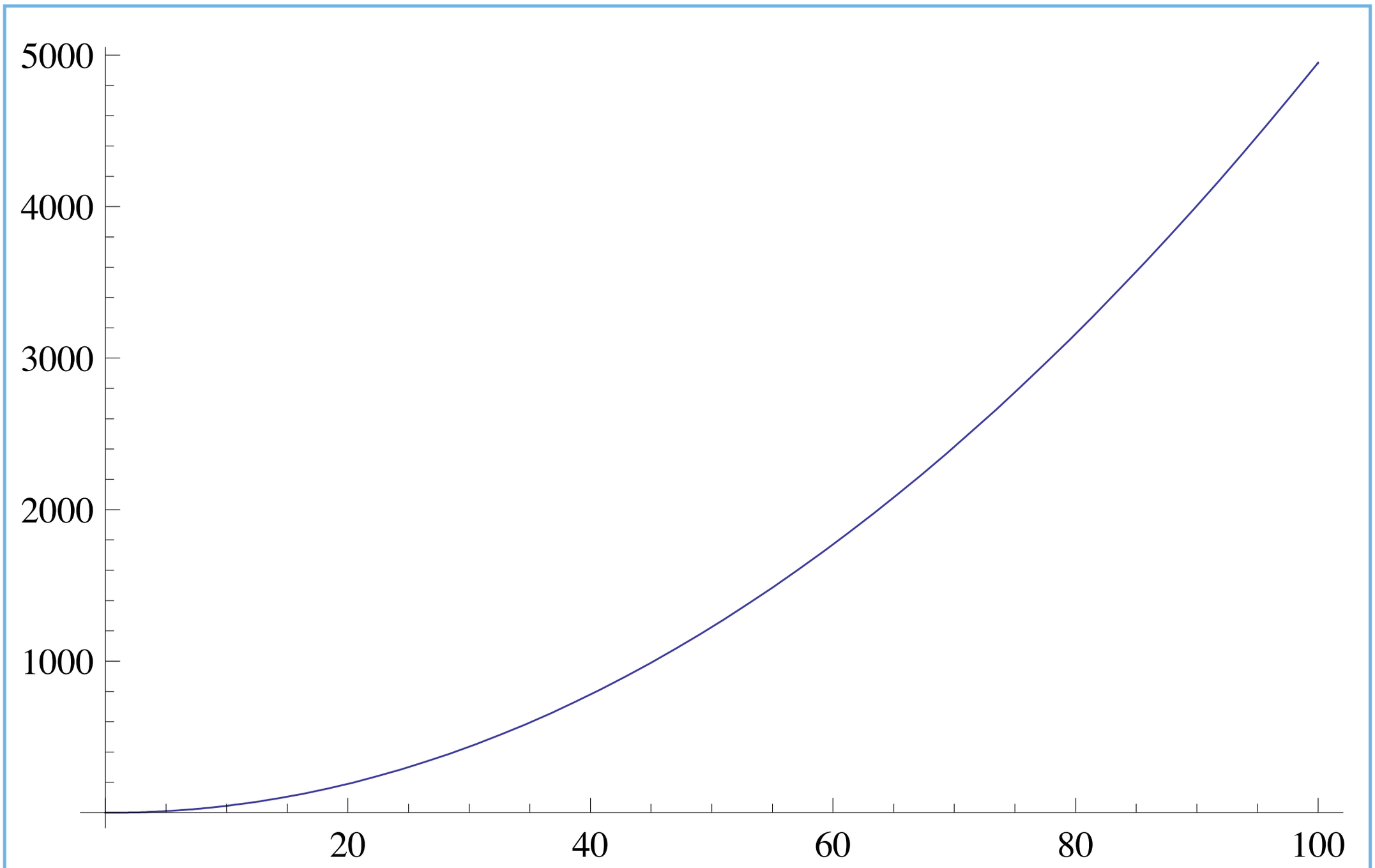
Communication Paths



Mythical Man-Month (VII)



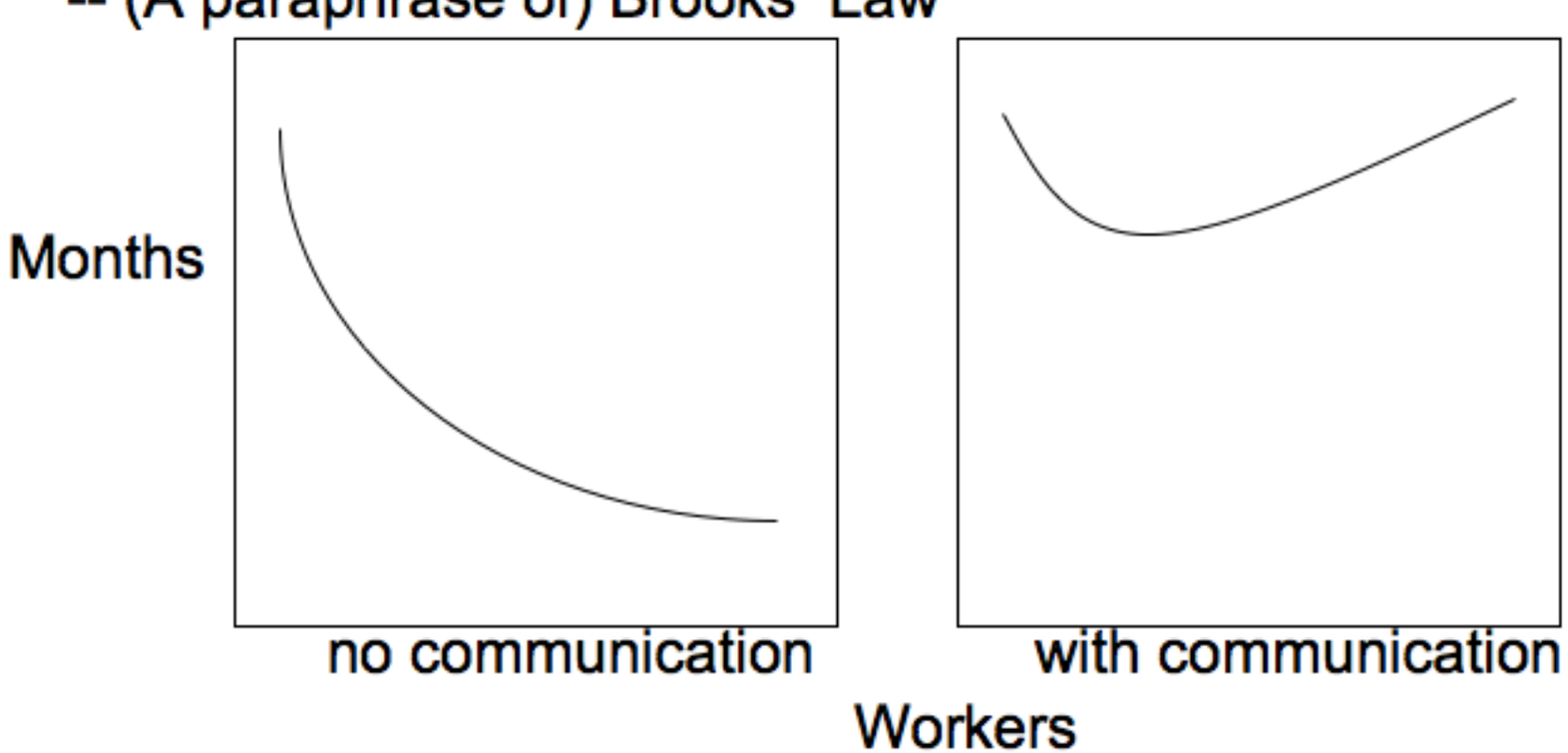
Another way to look at it



A 100 person team has 4950 potential communication paths to manage!

Some benefit, then none

"Adding more people then lengthens, not shortens, the schedule!"
-- (A paraphrase of) Brooks' Law



Back to the Example

- With 3 developers, we start by assuming that they can produce 180 days of development effort
  - (The book used 190 days, but I couldn't figure that out.)
- You then negotiate with the customer until the estimate of all the features in milestone 1.0 is less than 180 days
  - You then create an iteration plan and get to work
    - Keep your iterations short (30 calendar days, 20 work days)
      - It helps you deal with change and keep you focused
    - Keep your iterations balanced (new features, fixing bugs, etc.)

And, now reality sets in

- We can't necessarily assume 180 days of work from three developers over three calendar months
  - During the day there are constant interruptions that prevent developers from remaining "in the flow"
    - rather than 8 productive hours in a work day, you find that you only achieve 5 hours (or less)
- To account for this, agile methods make use of a concept called "team velocity" or "velocity"
  - Velocity is a percentage: given X number of days, how much of that time is productive? A default value is 0.7

Realistic Estimate

$$\frac{\text{project estimate (in days)}}{\text{velocity}} == \text{realistic project estimate}$$

30 calendar days, 20 work days == 14 days of productive time

Yikes!!!!

Example, cont.

- Now, that we know about velocity, we can use it to estimate how many days of productive work we can achieve during each iteration

$$3 \times 20 \times 0.7 == 42$$

number of developers x
working days in iteration x
velocity

- Since we have three iterations: $3 \times 42 == 126$

Example, cont.

- Went from: 3 people could do 270 days of work in 90 days
- To: 3 people could do 180 days of work in 90 days
- To (finally): 3 people could do 126 days of work in 90 days
  - Assuming an overhead of 0.7

- Question: what should we do with our velocity if we add MORE people to the project?
  - How would you change velocity if we shifted to 4 people?
    - or to 10 people?

Managing Customers

- The customer will ~~probably~~ definitely not like the change from 273 days of work possible to 123
  - Since it means a big reduction in what can be accomplished
- What to do?
  - Add an iteration (if they will let you)
  - Explain that overflow work is not lost, just postponed
  - Be transparent about how you came up with your figures
    - You now have an estimate that you can be confident in

The Big Board

- Once you have a realistic estimate and an iteration plan based on that estimate, you are ready to get started
  - You will track your progress with a software development dashboard
    - A large whiteboard that is partitioned to help your team focus on what is happening during the current iteration

User Stories

Title: Book package

Description: An Orion's Orbits user will be able to book a special package with extras online.

Title: Pay online

Description: An Orion's Orbits user will be able to pay for their bookings online.

Title: Show Current Deals

Description: The website will show current deals to Orion's Orbits users.

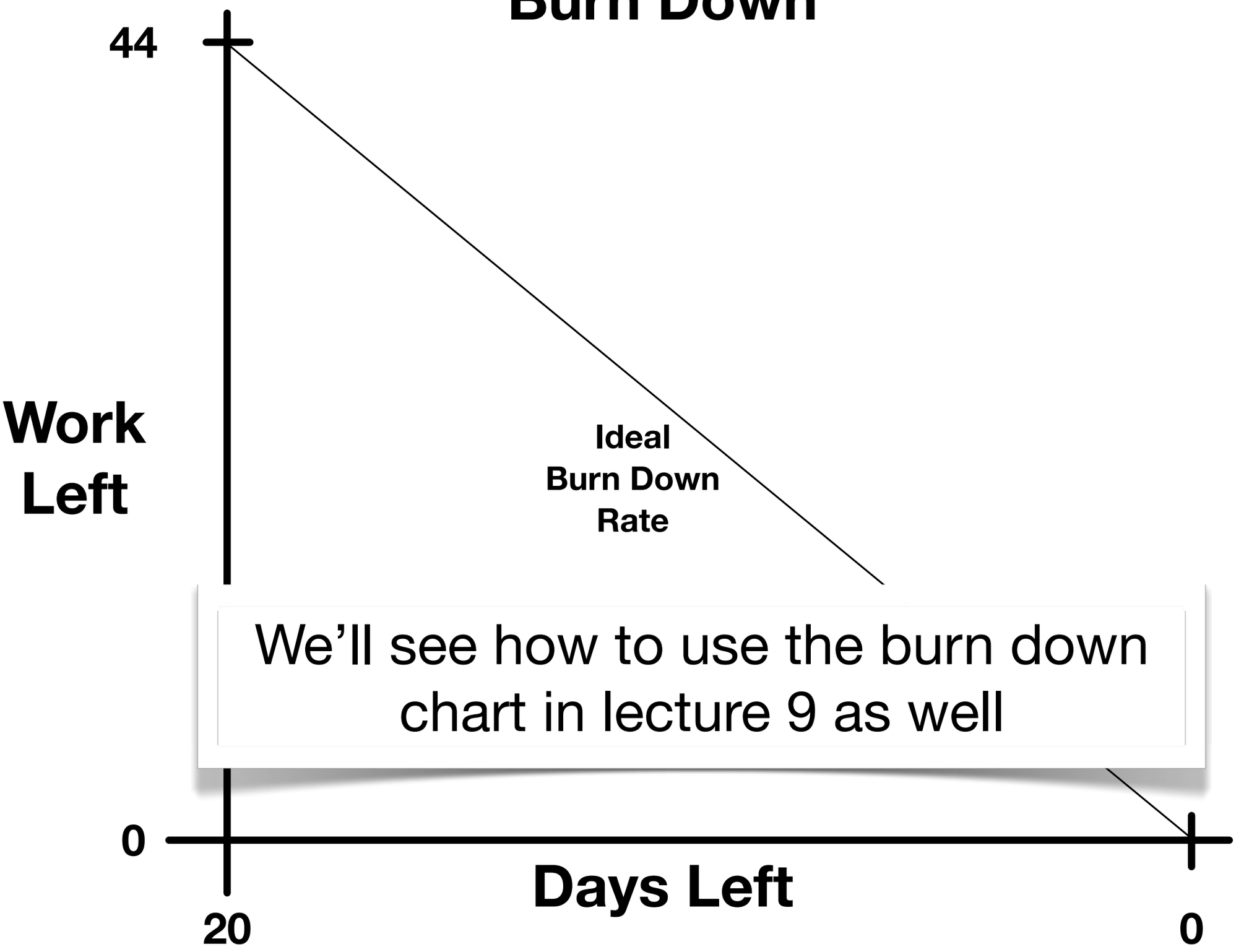
Burn Down

Next

We'll see how to use this board during an iteration in lecture 9

Completed

Burn Down


44
Work Left
Ideal Burn Down Rate
We'll see how to use the burn down chart in lecture 9 as well
0
Days Left
20
0

Wrapping Up

- Discussed
  - Factors that weigh into making an initial project estimate
    - Number of team members
    - Team Velocity
  - Mythical Man-Month
- Introduced
  - The Big Board
  - Burn Down Chart

Coming Up

- Lecture 8: Proving Correctness and Measuring Performance
  - Chapter 3 of Breshears
- Lecture 9: User Stories and Tasks
  - Chapter 4 of Pilone & Miles