Agile Project Planning

CSCI 5828: Foundations of Software Engineering
Lecture 16 — 03/08/2012

© Kenneth M. Anderson, 2012

Goals

- Review material from chapters 6-8 of our Agile textbook
- Agile Project Planning
  - User Stories
  - Estimates
  - Iterations
  - Burn-Down / Burn-Up Charts
- Compare our textbook's material with variations presented in other resources
  - Such as Head First Software Development
- Learn more from Fred Brooks!

© Kenneth M. Anderson, 2012

Pushing back on Requirements

- A key input to software development projects are requirements
  - Requirements Elicitation: Work performed to understand the problem
  - Requirements Specification: Information used to develop a solution
    - Describes “what” we are going to do, leaving the “how” to design
- The problem with traditional software life cycles is that
  - we attach too much importance in obtaining completeness
  - and too much busy work in documenting requirements to the nth degree
- As the book says, “Heavy documentation as a means of capturing requirements has never really worked for software development”

© Kenneth M. Anderson, 2012

Why?

- A team invests so much effort into creating traditional requirements documents that they start to
  - fear change
    - they know that any change request will require hours of work updating the requirements document to comprehensively and completely reflect the change
  - focus on the spec and build what it says; ignoring the customer
    - talking to the customer will bring change (and they fear change)
  - they encounter ambiguity and make bad guesses and false assumptions
    - this will not be caught because they are not talking to the customer

© Kenneth M. Anderson, 2012

Wasted Effort

- This approach of insisting on complete requirements upfront ends up wasting a lot of time and money
  - It conditions a team to be resistant to change
  - It can cause a project to fail when seemingly a lot of effort has been invested but there's little to actually show for the work
- In addition, even complete requirements fall prey to the ambiguities of language
  - (see book for an example concerning the sentence "I didn't say she took the money.")
  - You can try to remove ambiguity by adding more text but then you encounter the issue that there's more to read and people are more likely to miss things and less likely to read the document deeply

© Kenneth M. Anderson, 2012

What's Needed? Communication

- Having effort devoted towards complete documentation of the requirements hinders what is really needed
  - COMMUNICATION
- and lots of it
  - between the customer and the development team
  - between the team and external stakeholders
  - as well as within the team itself
- Our goal is to produce a "shared understanding" of the purpose of the project and what we are trying to achieve
  - what's the problem we are trying to solve and what features will help us solve it

© Kenneth M. Anderson, 2012

Unit of Communication: The User Story

- To facilitate communication
  - agile requirements are documented by user stories
    - not to be confused with use cases from the OO A&D realm
    - use cases are lower-level constructs that document scenarios a user might take with our system to accomplish a task
- A user story is a short description of a feature the customer wants
  - They are traditionally written on index cards
    - to prevent us from writing too much about the feature
    - Recall, agile approaches are trying to get away from the practices of traditional life cycles that prevented progress from being made

User Stories (I)

- User Stories should be short and be written in the user’s language
  - They should be at a high level of abstraction
    - “Users get notified about comments that appear on their posts”
    - “Create Post”, “Edit Post”, “Delete Post”, “Comment on Post”
  - They are purposefully not detailed to remind us that
    - we do not know enough about the problem domain
    - what we do know will change
    - we need to talk to the customer about this particular feature some more
    - we do not yet know when this feature will be assigned to an iteration

© Kenneth M. Anderson, 2012

User Stories (II)

- User stories should capture something that has value to the customer
  - Otherwise, why would they care?
- One tough aspect for developers in generating user stories is that customers really do not care about "cool technology choices"
  - The fact that you are using Hadoop will not impress them
  - But, "Classify millions of tweets to highlight those that negatively mention the customer's company" will get their interest
- User stories should attempt to identify features that fully exercise the system "end-to-end" from the UI to the controller tier to the model tier and back
  - We're looking for features that will require an application architecture to be defined and fleshed out in order to deliver that feature to the customer

User Stories (III)

- The book discusses the INVEST acronym for highlighting other characteristics that should be true of our user stories
  - Independent
    - Priorities shift, we need to be able to trade one story for another
  - Negotiable
    - Customer needs to be flexible about how some stories are implemented
  - Valuable
    - Must mean something to customer
  - Estimatable and Small
    - These characteristics go hand-in-hand; the smaller a story is, the easier it is to estimate exactly how long it will take
  - Testable
    - If something is testable, we can determine when we are done

Story Elicitation

- When having a conversation with the customer, you need to be on the look out for opportunities to convert what they say into user stories
  - "I want my site to stream videos of the top places to hike. I want people to annotate those streams with 'insider info' that provides more information about the trails than what is contained in the report by the Park Rangers."
- User Stories
  - Integrate video streams of top trails into website
  - Allow users to comment on video streams
  - Retrieve daily Park Ranger report for each trail

© Kenneth M. Anderson, 2012

Constraints

- Some stories will sound more like constraints than features
  - “The website must be really fast”
- This example is
  - too low level
  - oriented towards the technical rather than business concerns
- Try to rewrite to make them testable
  - The website will load each of its pages in under five seconds
- Keep them in the working set of stories, even though they are never assigned to an iteration; let them serve as reminders to the team about what they are trying to achieve

© Kenneth M. Anderson, 2012

Template

- If you have difficulty generating stories then fall back on this template
    - As a <type of user>
    - I want <some goal>
    - so that <some reason>
- Example from Book
    - As a surfer who likes to sleep,
    - I want to check local surf conditions via a webcam
    - so that I don’t have to get out of bed if there are no waves

© Kenneth M. Anderson, 2012

To get a project started (after inception)

- Host a workshop (a.k.a. retreat) for all stakeholders to generate user stories
- Get a big room that allows people to host plenary sessions or break up in small groups
- Generate lots of requirements-oriented artifacts
  - Personas, flowcharts, scenarios, architecture diagrams, concept art or designs, storyboards, paper prototypes, and more
- Write lots of stories (looking for 20-40; about 3 to 6 months of work)
  - Work your way through the artifacts above and generate story ideas as you go along
    - look for stories that could be completed in 5 to 10 days
    - a few “epic” stories are fine, but keep them to a minimum

© Kenneth M. Anderson, 2012

Estimates

- After you have created a set of stories, you need to assign estimates to them
  - Assigning estimates is the first step towards being able to identify a realistic deadline for the project
- Agile, as you might expect, takes a different approach to estimates than traditional software life cycles

The Problem with Estimates

- In the Mythical Man Month, 37 years ago, Fred Brooks identified the typical problems associated with software estimates
  - Software engineers often do not use formal methods to develop estimates
  - Software engineers are also optimistic about what they can accomplish
    - a side effect of the creative joy that can be associated with programming (more on that in a minute)
  - As a result, the estimates are not realistic and and thus easy to miss
- Another problem is that we are asked to make an estimate about a project that is not fully specified (and may change)
  - As our book says “accurate up-front estimates are not possible”

© Kenneth M. Anderson, 2012

An additional problem: adding people to a project

- Another problem that traditional life cycles encounter on a project
    - When a project is not going to make its (unrealistic) estimates
        - managers add more people to the project hoping to catch up
- Agile disagrees, it insists that you should
    - fix budget and time up front
        - this means fixing team size as well (people == money)
    - strive for high quality always
    - be willing to drop unimportant features to meet deadlines or as new functionality is identified
- To understand, why adding people to a project is so bad, let’s return to Brooks

© Kenneth M. Anderson, 2012

Mythical Man-Month (I)

- Famous essay (and the title of Brooks's famous 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)

© Kenneth M. Anderson, 2012

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.”

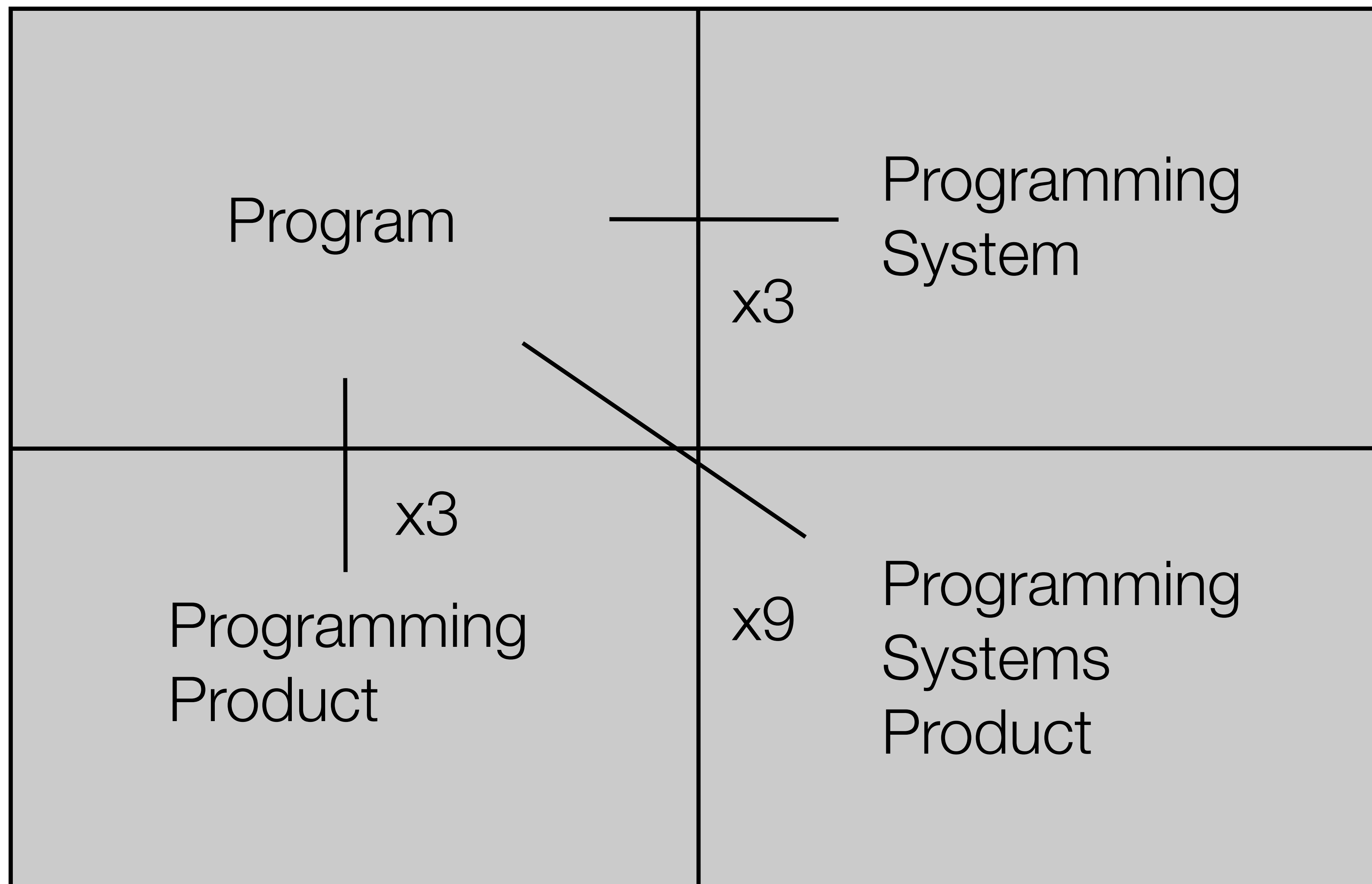
© Kenneth M. Anderson, 2012

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

© Kenneth M. Anderson, 2012

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

© Kenneth M. Anderson, 2012

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!

© Kenneth M. Anderson, 2012

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!”

© Kenneth M. Anderson, 2012

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

© Kenneth M. Anderson, 2012

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!”

© Kenneth M. Anderson, 2012

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)

© Kenneth M. Anderson, 2012

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)

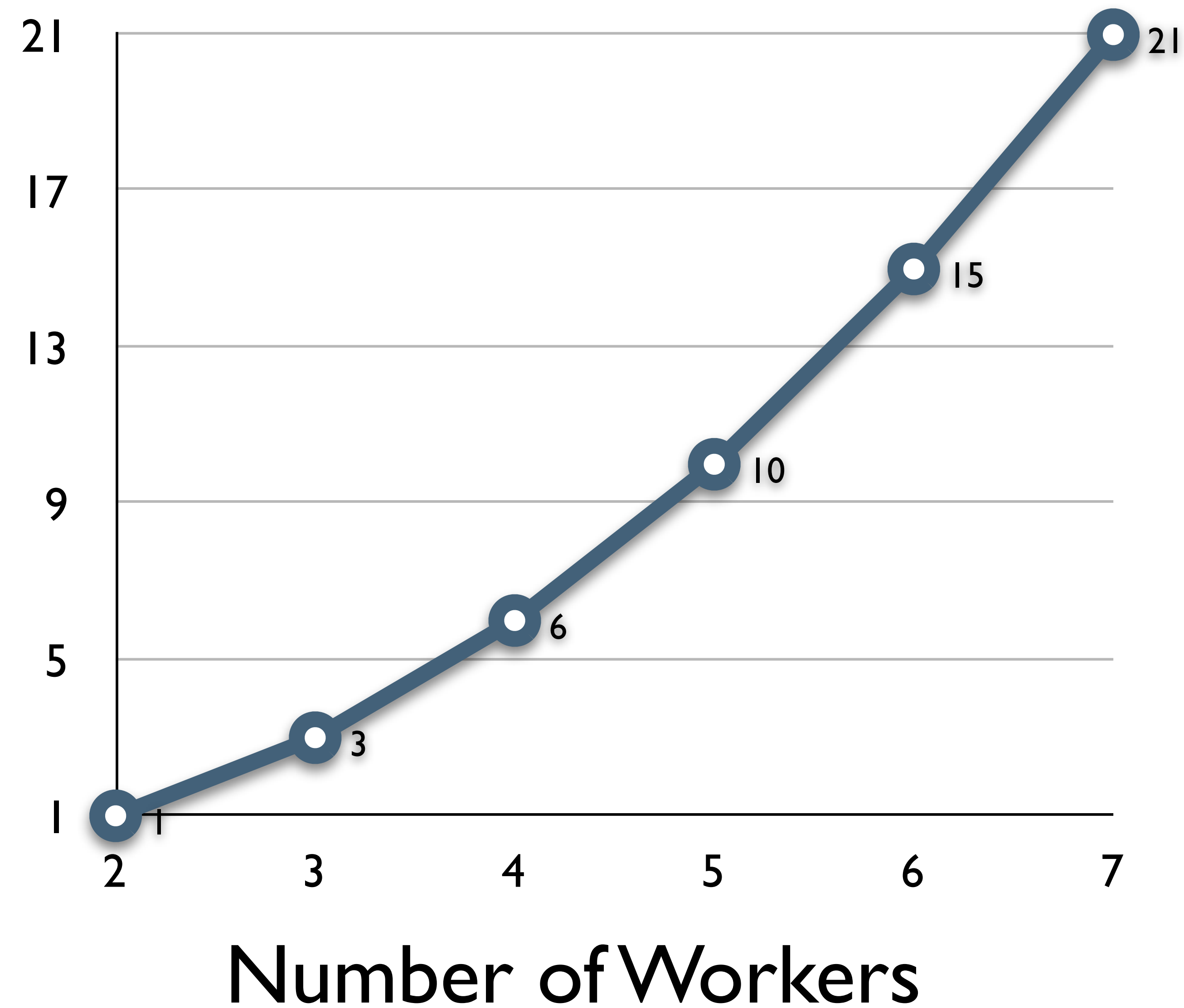
© Kenneth M. Anderson, 2012

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

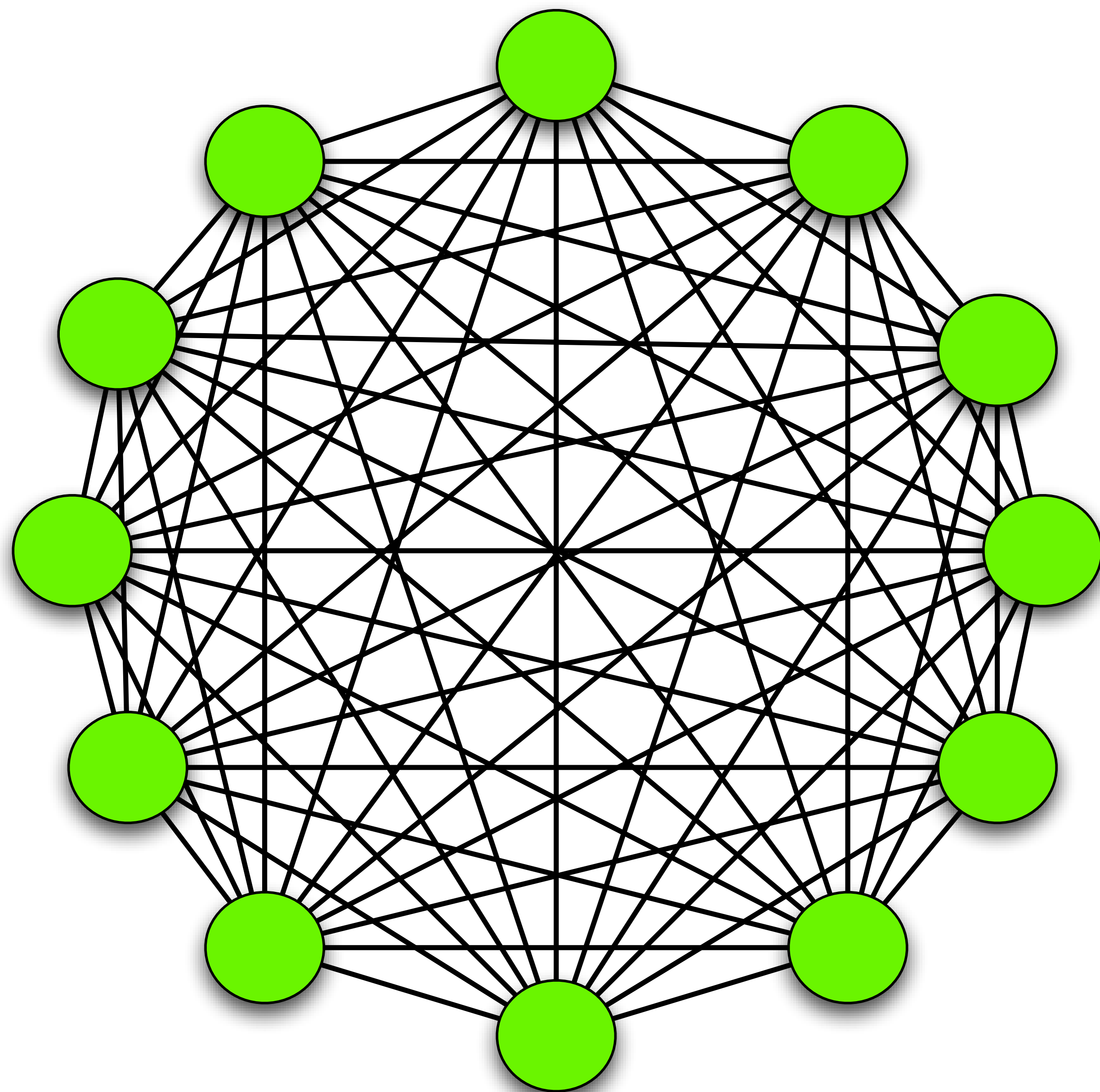
© Kenneth M. Anderson, 2012

Mythical Man-Month (VI)



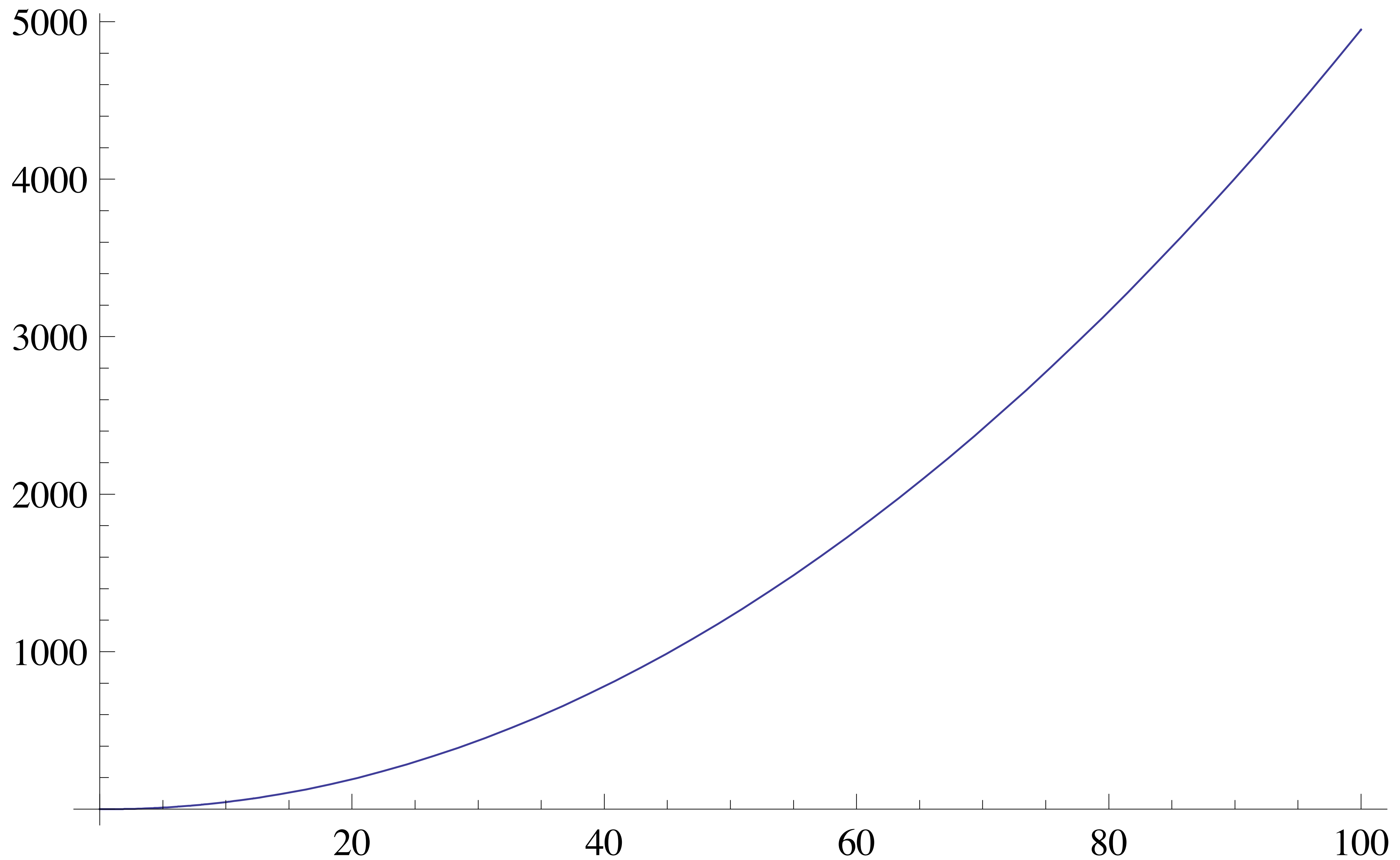
© Kenneth M. Anderson, 2012

Mythical Man-Month (VII)



Another way to look at it

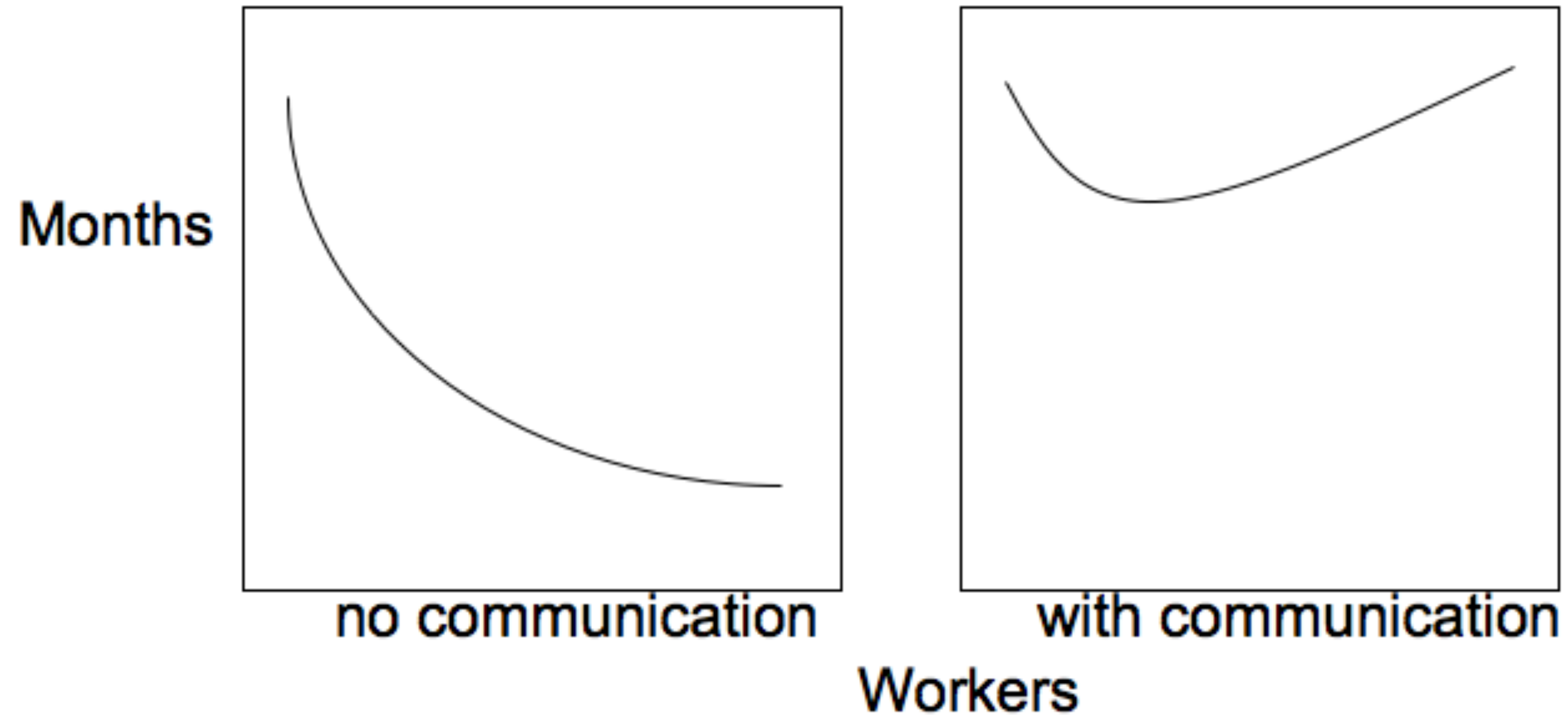
© Kenneth M. Anderson, 2012



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

© Kenneth M. Anderson, 2012

Some benefit, then none



© Kenneth M. Anderson, 2012

New Approach

- Our goal with our initial estimates is to answer the question
  - Is this project even possible
    - given our resources
    - and the general timeline our customer has provided
- The Agile approach to estimates involves
  - building something (1 or 2 user stories converted to working software)
  - measure how long it took (what eventually is called “team velocity”)
  - use that for planning the rest of the project

© Kenneth M. Anderson, 2012

Points not Days

- The secret is not to deal with estimates that talk about hours/days/weeks
  - Instead, we assign points to a user story to indicate relative lengths
    - “This is a short user story” -- 1 point
    - “This story will take a while” -- 10 points
    - “This story is about three times longer than that short one” -- 3 points
- Team velocity then is a measure of “how many points can we complete in a single iteration”
  - For our first couple of iterations, we simply guess and see what happens
    - Our guess will be wrong but if we stay consistent in assigning points to stories, the velocity we can achieve will soon make itself known

© Kenneth M. Anderson, 2012

Benefits of Points

- Using points has a number of benefits
  - Humans are good at working with relative sizes
    - All we are trying to do is capture the “bigness” of a task
  - Points remind us that our estimates are guesses and we are doing other things (team velocity, iteration planning, etc.) to verify them
  - Relative estimates rarely change (Task A IS twice as big as Task B)
  - This type of system is fast, simple, and easy to learn
- Are “points” necessary?
  - No, some teams use “t-shirt sizes”: small, large, XL
  - When you do that, you need a way to map that into iteration planning

© Kenneth M. Anderson, 2012

Triangulation: Another benefit of the point system

- Triangulation refers to the fact that once you have sized a few user stories
  - and converted them to working software
- you'll be in a much better place to size future stories
  - That task is similar to the one we did last week, and that was a 2 point task
- You can then quickly categorize all remaining user stories or any new ones
  - You can even use triangulation when your team velocity is not yet clear
  - If you've decided that one database-related task is 5 points before your first iteration, you're likely to classify all such tasks as 5 points

© Kenneth M. Anderson, 2012

Planning Poker

- A popular estimation technique in agile methods
- Addresses the problem in which two or more team members come up with wildly different estimates for a story
  - i.e. when a single user story generates estimates of say “3 points”, “10 points”, and “100 points” from three different developers
- The underlying cause for these different estimates is assumptions; what did you assume was true or not true about the project to generate the number that you did?

© Kenneth M. Anderson, 2012

Example

- “Add a comment on a product page”
- One team member might think:
  - “Simple. We need a form, a script to process the form, and a place to store the comment in the database. 3 days.”
- Another might think:
  - “Hmm. How do we relate the comment to the product? Do we have one comment table per product in the database? Will I need to change the product class? Maybe there is code from some other place in the system that I can re-use. 2 weeks.”

Example, continued

- Finally, another might think:
  - “Ugh. Complete database re-design. No code to re-use (this is the first time we’re allowing comments). What user interface should we use? Can the user embed HTML in their comments? How do we handle smileys? How will this impact the product model class? Do we keep the comments forever? Do we need moderation? Can a user edit a past comment? Who gets to delete comments? Yuck!! 3 months!”
- Based on your assumptions, you’ll get completely different numbers. How do you get these assumptions to the surface? Planning Poker!

© Kenneth M. Anderson, 2012

Planning Poker (I)

- Create “deck” of cards. 13 cards per “player”.
  - Each card contains an estimate spanning from “already done” to “wow this is going to take a long time”.
    - 0, .5, 1, 2, 3, 5, 8, 13, 20, 40, 100 days
  - One card has a “?” meaning “not enough information”
  - One card has a coffee cup meaning “lets take a break”

- Note: this is different from our text book; our text book says you do NOT need this many cards, it just confuses things
  - What I’m presenting here is from Head First Software Development

© Kenneth M. Anderson, 2012

Planning Poker (II)

- Place a user story in the middle of the table
  - Each team member thinks about the story and forms initial estimate in their heads
- Each person places the corresponding card face down on the table; note: estimate is for entire user story
- Everyone then turns over the cards at the same time
- The dealer marks the spread across the estimates

© Kenneth M. Anderson, 2012

Planning Poker (III)

- The larger the difference between the estimates, the less confident you are in the estimate, and the more assumptions you need to highlight and discuss
- So, the next step in planning poker is
  - Put assumptions on trial for their lives
- Have each team member list the assumptions they made and then start discussing them
  - You need to criticize the **assumption** *not the person*
- Goal is to get agreement on what assumptions truly apply

© Kenneth M. Anderson, 2012

Planning Poker (IV)

- If the assumptions reveal a misunderstanding of the requirements, then go back to the client and get that misunderstanding clarified
- Otherwise, start to eliminate as many assumptions as possible, then have everyone revise their estimates and play planning poker again to see if the spread has decreased
  - Your goal is convergence. Once estimates cluster around a common number, assign that number and move to the next story
- Do this until all user stories have a consensus estimate assigned
  - If any ambiguities remain, consult with the customer and try again

© Kenneth M. Anderson, 2012

Planning Poker (VI)

- Things to watch out for
    - Although implied in the previous slides, don't do one card at a time with multiple customer sessions each time
        - Value your customer's time
            - Process each card, identifying assumptions/misunderstandings that need clarification; THEN meet with customer
    - Big estimates (== bad estimates)
        - They indicate that the story is too big; decompose; try again
        - Iterations are typically 20 work days (1 month) or less
        - Estimates longer than 15 days are more likely to be wrong than those shorter than 15 days; (others think 7 days is upper limit)

© Kenneth M. Anderson, 2012

What’s Missing? Priorities

- At this point, we have
    - a set of user stories
    - estimates assigned to each story
        - and, importantly, estimates that we have thought carefully about
- The next piece of the puzzle is customer priorities
    - We need to know from the customer what stories are the most important
    - We will use that information when we assign stories to iterations
        - Iterations will be some multiple of a work week (5 days, 10 days, etc.)
        - Once you decide on iteration size, you should hold it constant

© Kenneth M. Anderson, 2012

Agile Planning

- Now that we have
  - user stories with estimates and priorities
- It’s time to engage in Agile planning
  - Agile plans are dynamic; we update them as we get new information
    - This can lead to a “love/hate relationship” by managers with Agile methods
      - They **love** the visibility that this type of planning provides about a project; And, they **hate** the visibility this type of planning provides about a project (!)
      - There is no hiding slippages, there’s no ability to fool yourself about the state of the project

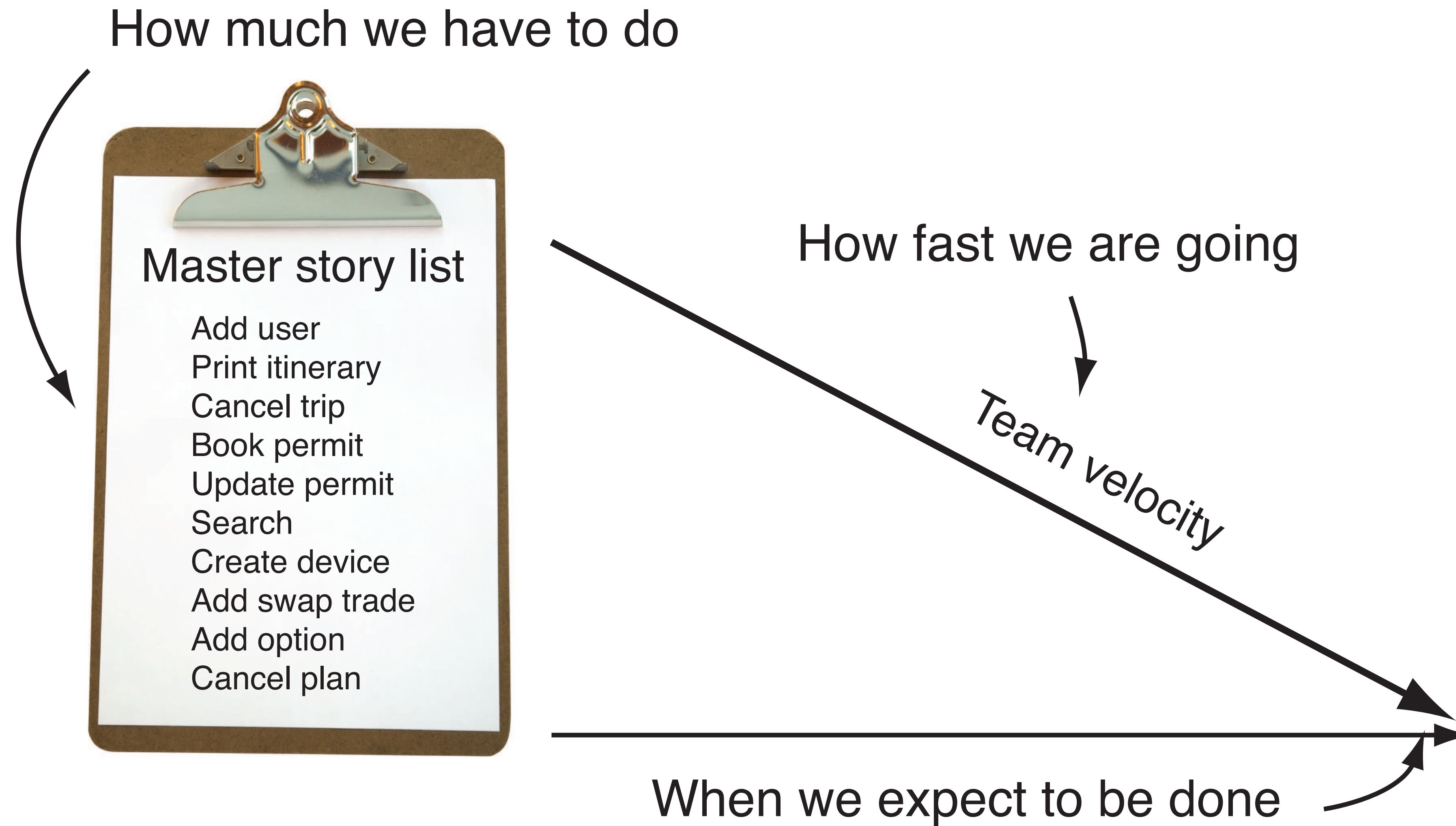
© Kenneth M. Anderson, 2012

Problems with Static Plans

- Agile planning is a response to traditional static plans
  - Plans based on guesstimates with no acknowledgment up front that things might change
- The problem? THINGS CHANGE!
  - Team members leave
  - Your initial estimates are wrong and you are going slower than planned
    - (Surprise!)
  - Your customer changes their mind
  - Your deadline changes (is moved forward, not back)
- Rather than ignore these things, agile planning deals with them head on!

© Kenneth M. Anderson, 2012

Agile Planning In a NutShell



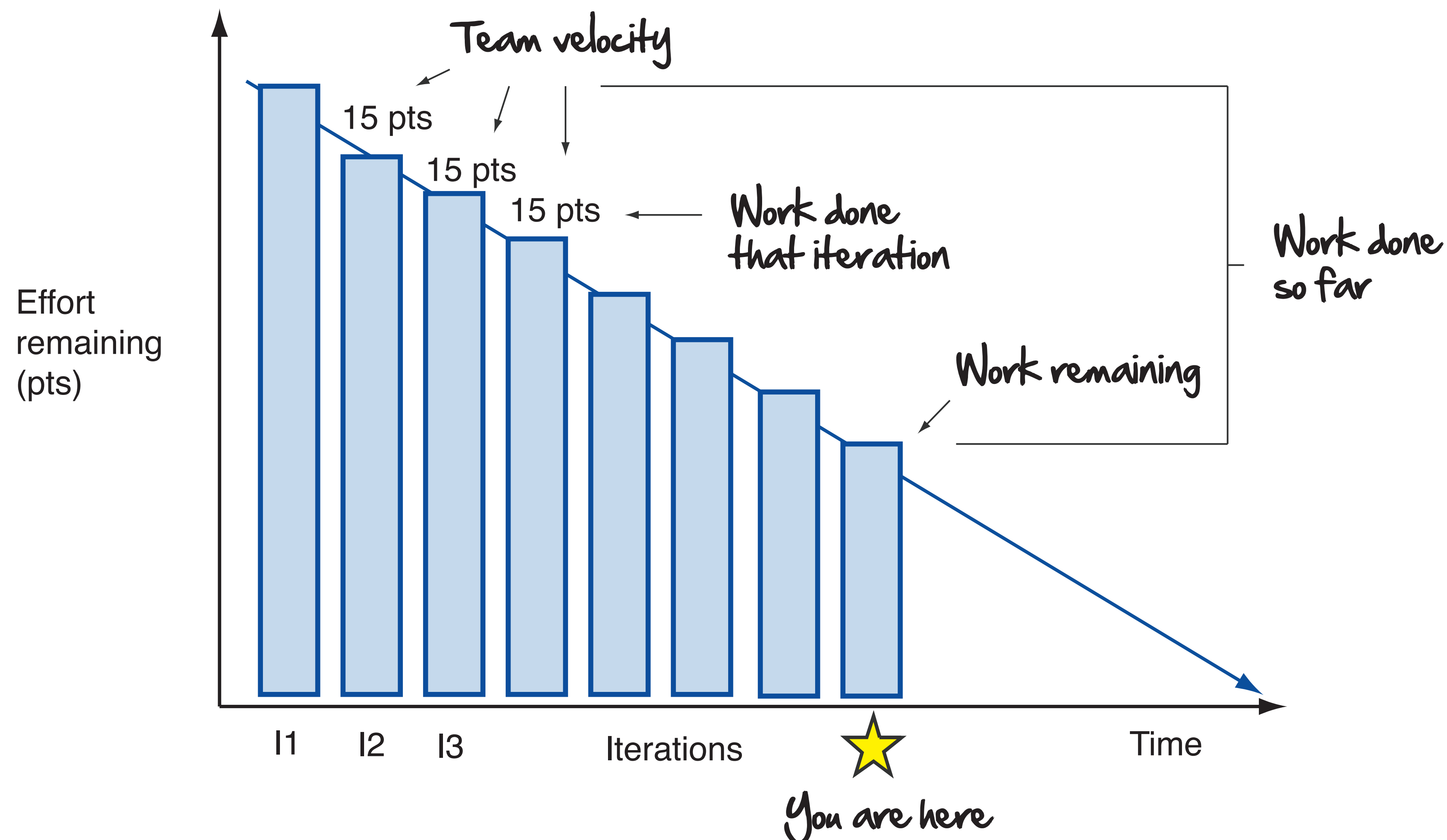
Velocity

- To get started, we estimate our team velocity
  - How many points will we convert to working software in our first iteration?
- To set our first estimate for the entire project, we divide our total number of points by our estimated velocity
  - This tells us the (approximate) number of iterations we will need to produce the system
    - iterations = total number of points / estimated velocity
- We can then get started on the first iteration to see how good our estimates are
  - after a few iterations, we'll be able to calculate actual velocity
    - velocity = points completed (working software) / number of iterations

© Kenneth M. Anderson, 2012

Burn-Down Charts (I)

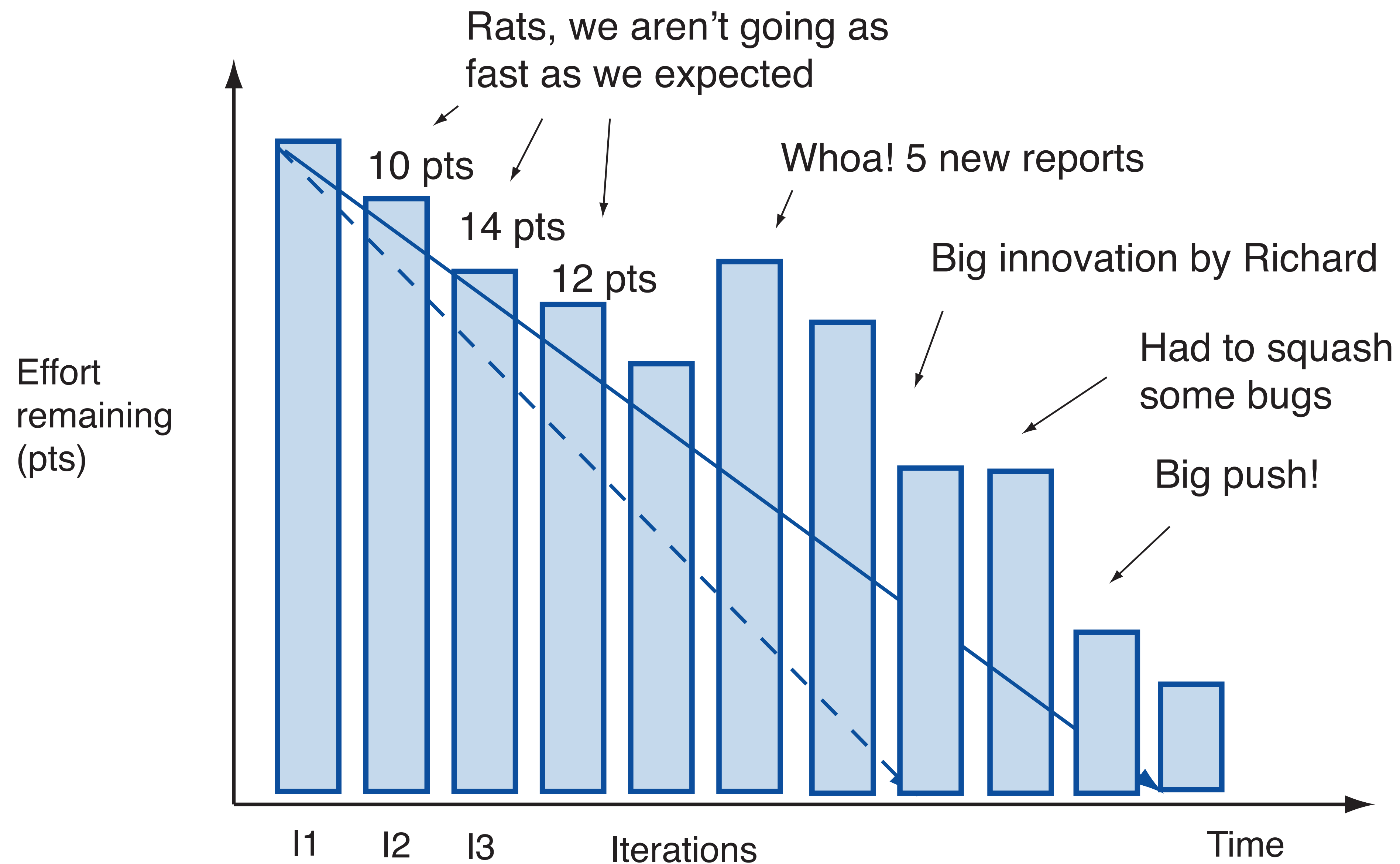
- To track our progress, we then use a burn down chart
  - Ideally, the chart looks like this



© Kenneth M. Anderson, 2012

Burn-Down Charts (II)

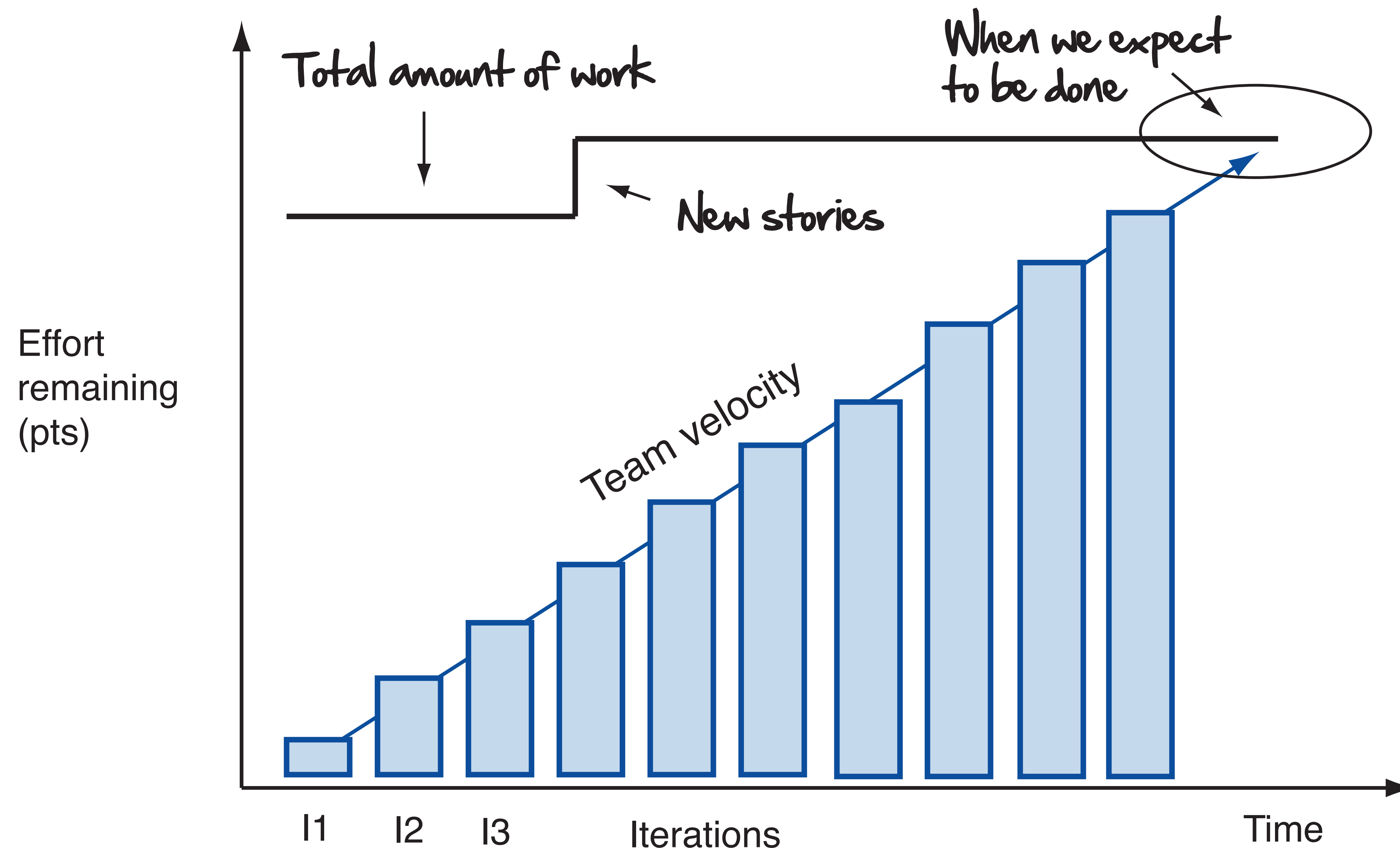
- But in actuality will look like this



© Kenneth M. Anderson, 2012

Burn-Up Chart

- Some teams prefer the burn-up chart, since it shows slippage a bit better



Not just for the big picture view

- Burn-down and burn-up charts are useful for tracking progress across iterations
  - but they can also be used **during** an iteration
- Such charts show the total effort planned for the iteration (in terms of points)
  - Each column shows how many points were completed that day
    - And, a user story is only completed when it is ready to deploy
- Such charts can show the impact of new feature requests to the customer
  - and ensure that they are really serious about making the request

© Kenneth M. Anderson, 2012

Dealing with Slippage

- Inevitably, the charts will reveal that the team is not going to make the initial deadline
  - This is not a cause for alarm, we knew we were guessing at the time we made the estimate
- Now, however, we are in a place to deal with the situation with facts
  - We know the estimates for the remaining stories and have confidence in them
  - We know the priorities and what we are trying to achive
  - We have fixed budget, time, and quality
  - It’s time to flex the scope: drop unimportant features or swap old features with new (estimated, prioritized) features

© Kenneth M. Anderson, 2012

Pushing back the deadline

- If a customer decides that they do not want features dropped
  - they have one option, pushing back the deadline
- But, when they do this, they are fully aware that means that
  - the budget has just increased
  - and they've decided its worth the expense
- Contrast this with the traditional situation
  - "What do you mean you won't make the deadline?"
  - "You're now over budget! And, your project is LATE!"
- Now, if you decide to miss the initial targets, it's a planned event

© Kenneth M. Anderson, 2012

What's Next?

- Execution
  - Now that we have a plan in place
    - Estimated, prioritized stories assigned to iterations
  - we need to execute the plan
- We'll look at the mechanics of agile iterations in a future lecture

© Kenneth M. Anderson, 2012

Summary

- Agile Project Planning
  - User Stories
  - Estimates and Priorities
    - Planning Poker
  - Iterations
  - Burn-Down / Burn-Up Charts
- Also learned about the mythical “person month” and Brooks’s Law about adding developers to a late software project (don’t do it!)

Coming Up Next

- Lecture 17: Intermediate Cucumber
- Lecture 18: Review of Midterm

© Kenneth M. Anderson, 2012