The Nature of Software Development, Part One

CSCI 5828: Foundations of Software Engineering
Lecture 23 — 11/10/2015

Goals

- Cover the material presented in Part One of our Agile textbook
  - Chapters 1 to 5
  - (Seems like a lot, but this book is a quick read)
- General comments on the book
  - This isn't your traditional textbook!
  - It is a series of informal essays from a master of software development
    - The goal: Understand the simplest expression of what software development is all about
      - When faced with complexity, simplicity in our own work is key
        - it helps us reduce the number of accidental difficulties we inject into our own work!

The Natural Way (I)

- Ron Jeffries acknowledges that software development is hard
  - Who is Ron Jeffries? See here for more info.
    - He's been a software developer since the early 60's and written a lot of software
    - He helped to invent Extreme Programming
    - He has written extensively on-line about Agile and XP
- But, he asserts in the introduction that if you are careful, there is a "natural path" for software development that can be stated simply
  - *In software development, focus on delivering value early and often*
    - The rest of the book is an attempt to explain this statement!

The Natural Way (II)

- Life (and complexity) will intervene and knock us off the Natural Way
  - We won't always be on the path due to the accidental and essential difficulties of software engineering
- However, the Natural Way serves as a guide, reminding us that we can get things back on track and providing steps to get there
- In particular
  - by reminding us to focus on value and
  - by reminding us to identify the easiest way to start delivering that value to ourselves and to our customer

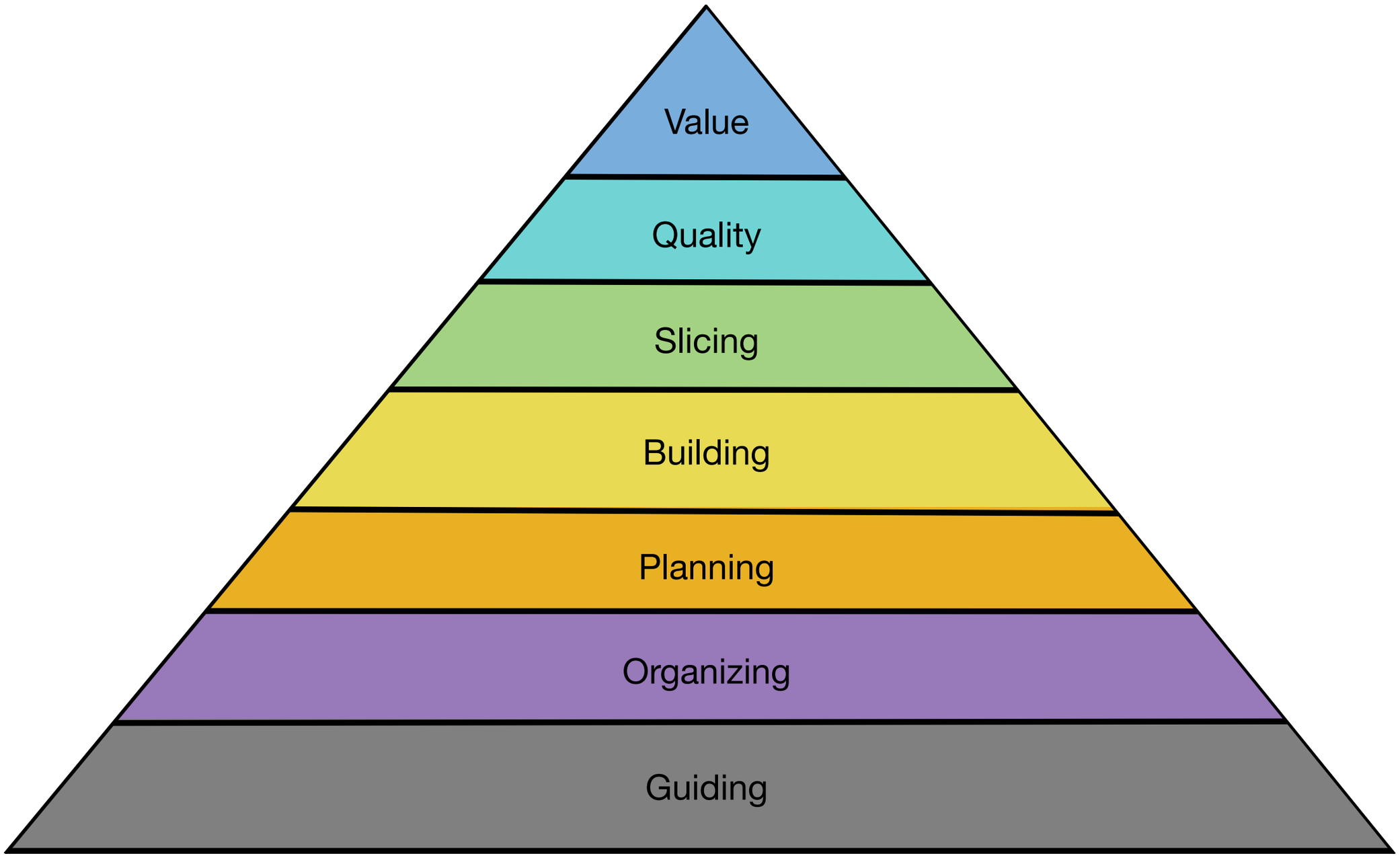
Characteristics of the Natural Way

- So, Jeffries asserts that there is a natural way to develop software and it serves everyone well
  - It serves ends users well by delivering value to them quickly
  - It serves our company well since
    - it provides ROI sooner
    - it provides important information quickly
    - it provides the ability to course correct as needed
- It serves management well since it lets them
  - see what is really going on in a project (allowing them to act if needed)
  - it makes information visible so they do not have to dig for it
- It serves developers well since it provides them with clear direction and gives them the freedom to use their skills to build what's needed

Warning

- Jeffries warns us:
  - The Natural Way is simple but it is not easy
    - The ideas in this book are guidelines
      - you will need to think about the ideas and how they might apply to you and your situation
      - you will (perhaps) need to change your behavior and identify ways to make your approach (or your company's approach) to developing software simpler
        - the "frequent delivery of visible value" is easy to understand but difficult to implement
- His book does not tell you HOW to implement these ideas
  - it instead offers ideas about WHY software should be delivered this way

The Natural Way in a Nutshell



Layers At A Glance: Getting to Value

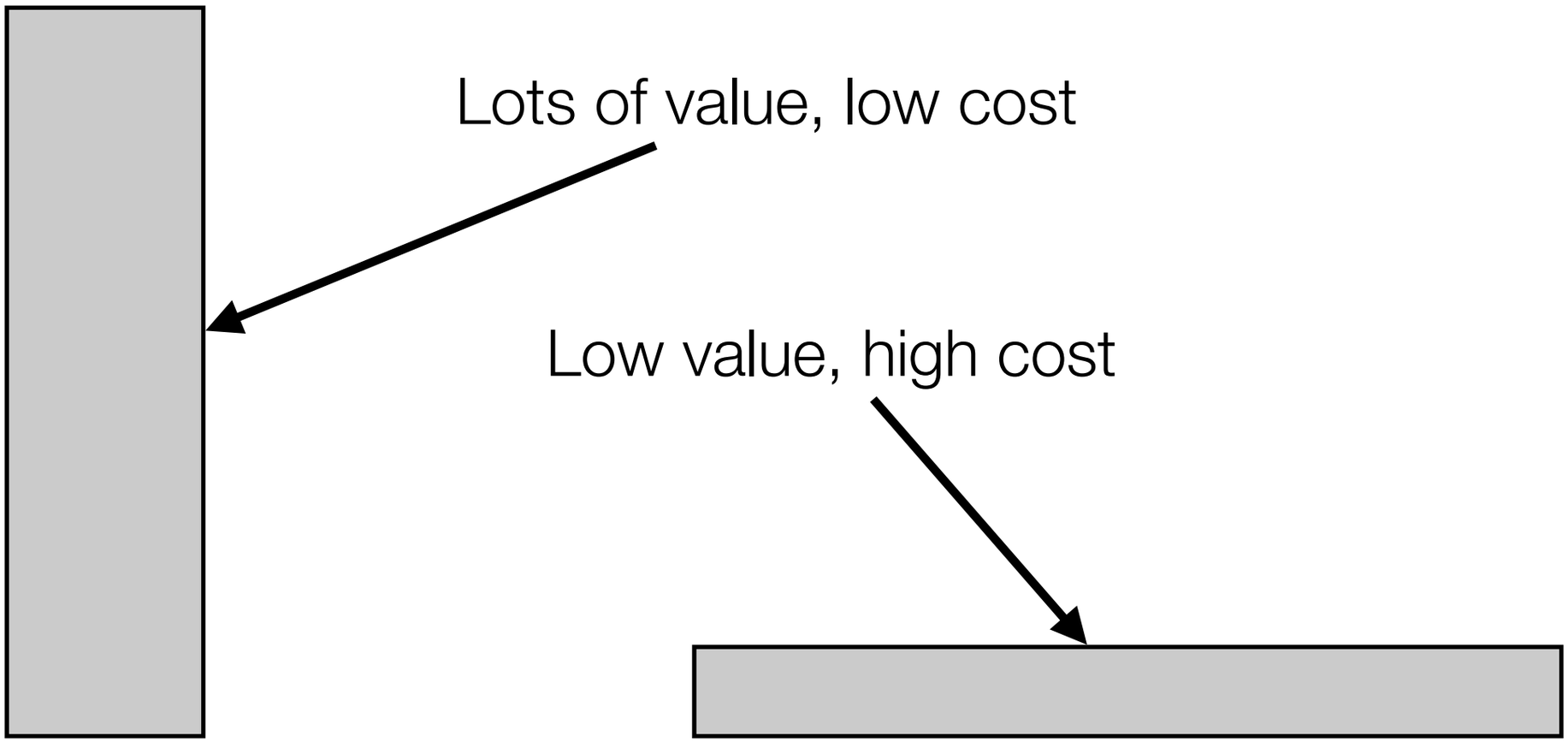
- We start by creating a team responsible for creating value for a customer; We **guide** them by watching what they produce and providing feedback
- We **organize** teams to get work done. We organize our project around *features* since *users find value in features;* they are easy to organize around
- We **plan** our projects by working on features our client needs
- We **build** our product by implementing features, providing them with frequent delivery of value
- We **slice** our features down to the minimum needed to provide value; we build on these features over time to provide more value; our system is already ready for deployment
- We build our system using techniques and tools that lead to high **quality**; we work in a way that is sustainable
- In this way, we deliver **value** to our customer; value is “what you want”

What is Value?

- Jefferies asserts that there is no mystery with respect to what “value” is
  - Value is what we want; in software, we typically want features
    - features in a system provide value to its customers
      - each time we finish a feature, we show it to the customer
        - they see value quickly; we get feedback and information about what to build or work on next
- A key theme is that value starts when the system is deployed
  - If you wait too long to give the user something, they don’t see value in what you’re building
  - this theme then leads us to focusing on small valuable features since it is easier to create/complete small features

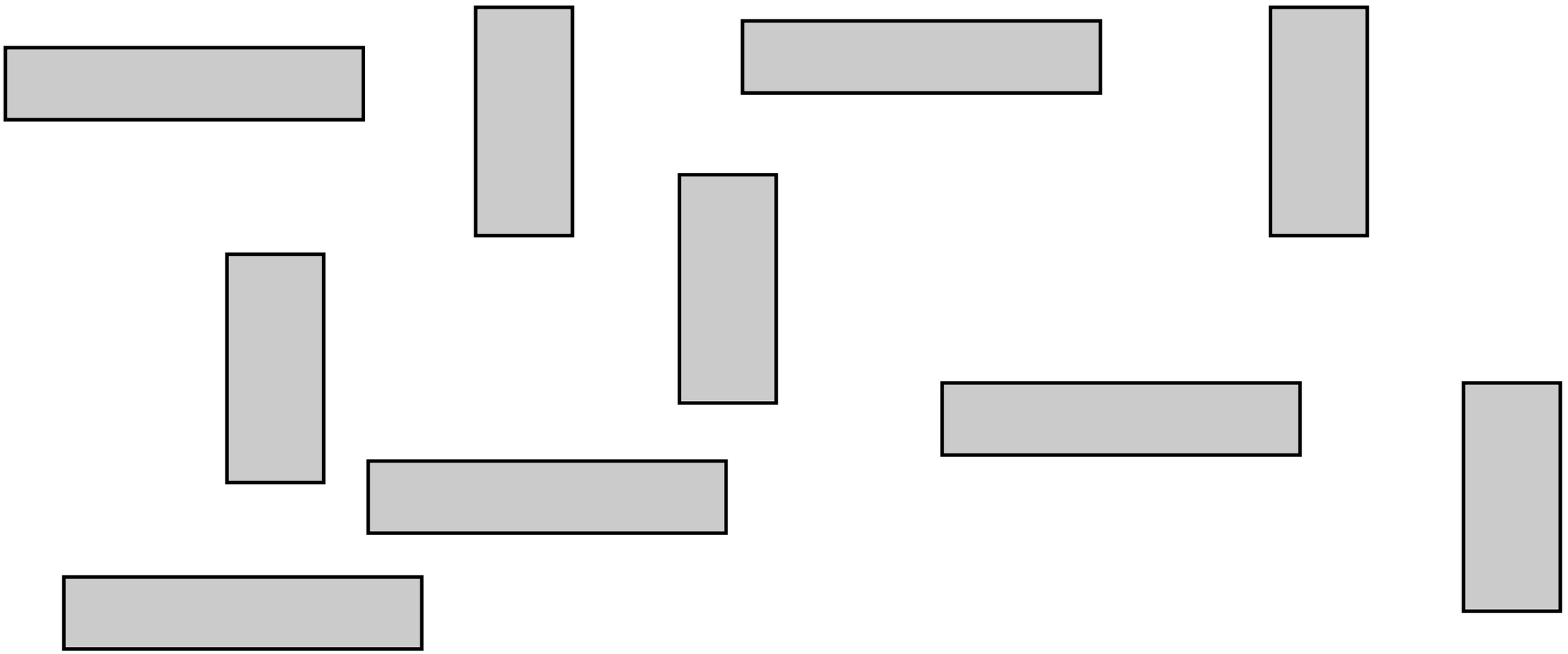
The Argument Made Graphically (I)

Imagine each feature is a rectangle; width is cost/time; height is value



The Argument Made Graphically (II)

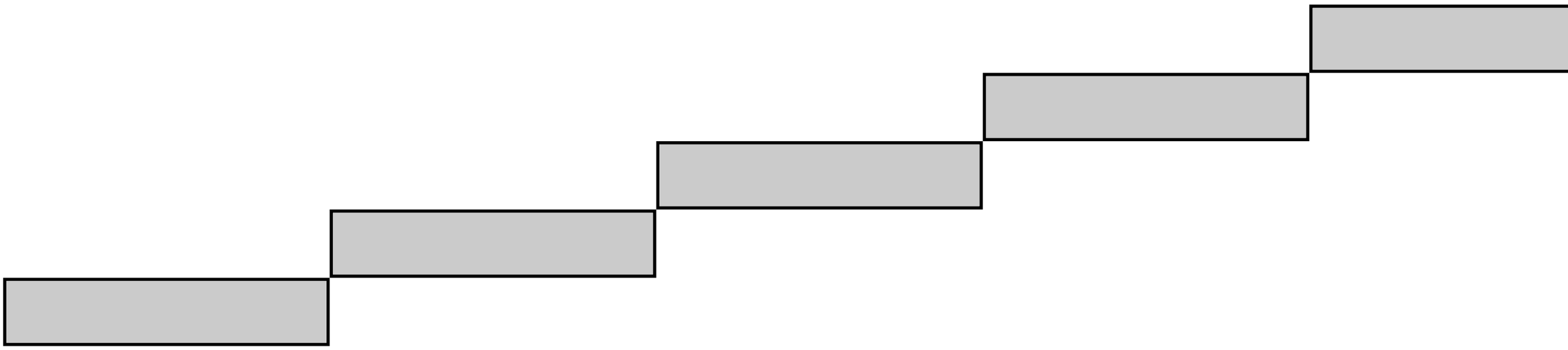
As a team, you're presented with a bunch of features



which ones should we build first?

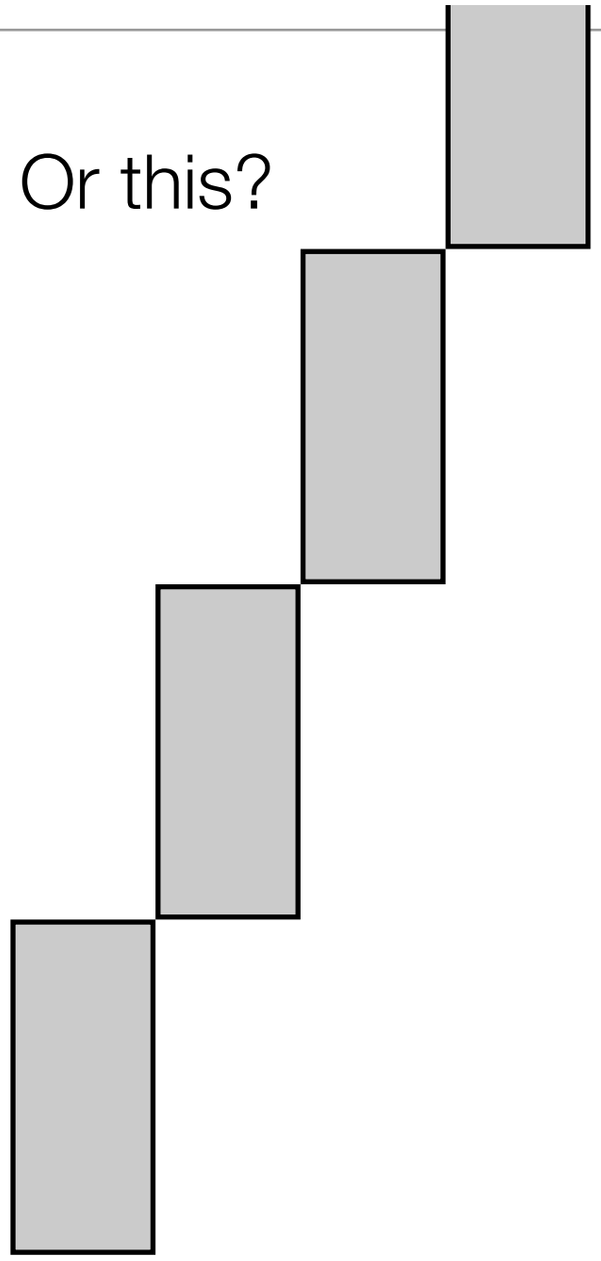
The Argument Made Graphically (III)

Should we do this?

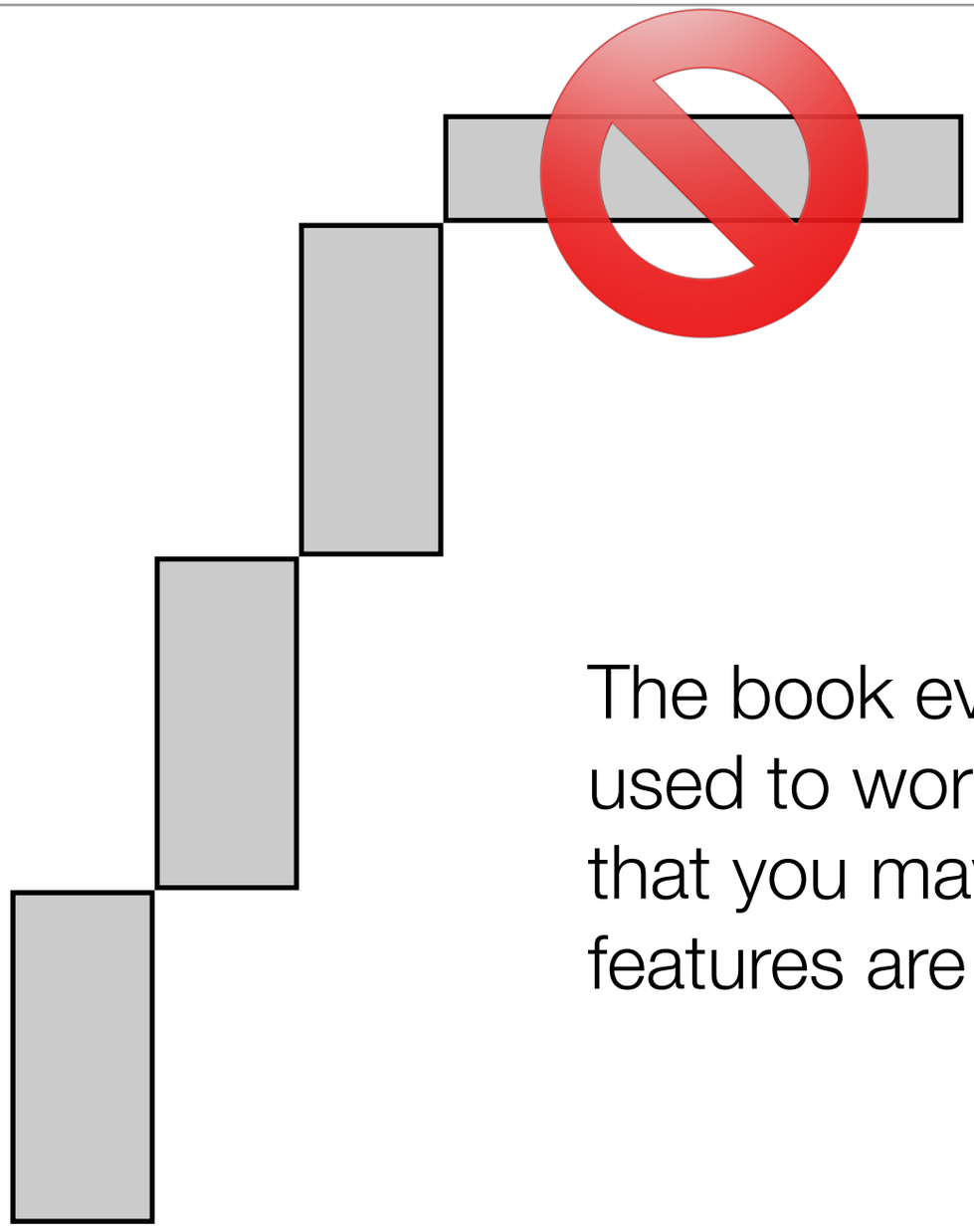


The Argument Made Graphically (IV)

Or this?



The Argument Made Graphically (V)



The book even makes the point that if you're used to working on short, high value features that you may decide that long, low value features are just not worth the effort!

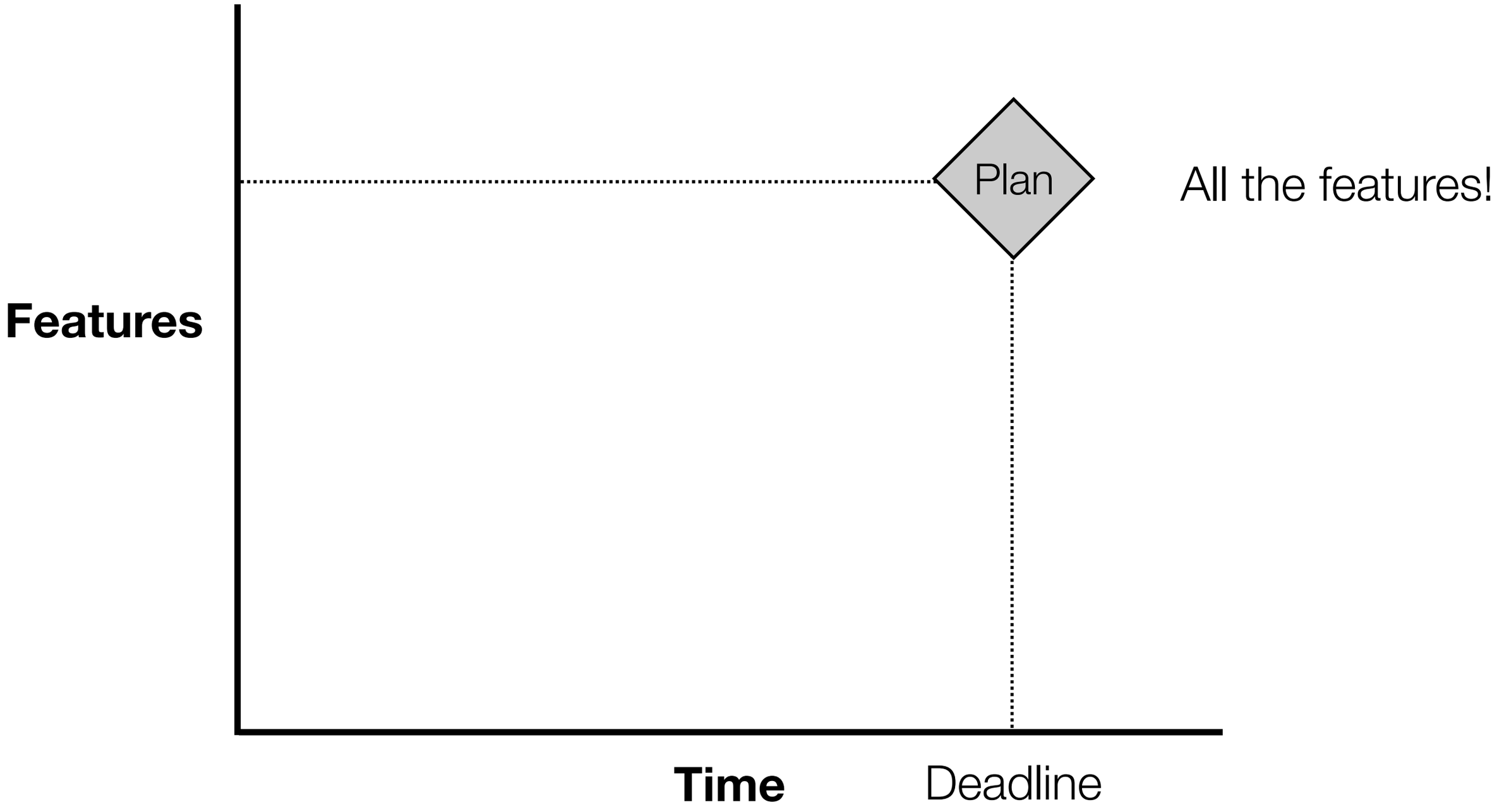
Discussion

- Jeffries point is that the best value for a software development project comes from small, value-focused features that are delivered frequently
  - Now, your development situation will never be as clear cut as the preceding slides
    - but remember this book is not about “how”, it’s about “why”
- In this case, why it is important to really understand what your customer (or user) wants
  - how can we deliver value to them
    - as fast as we can

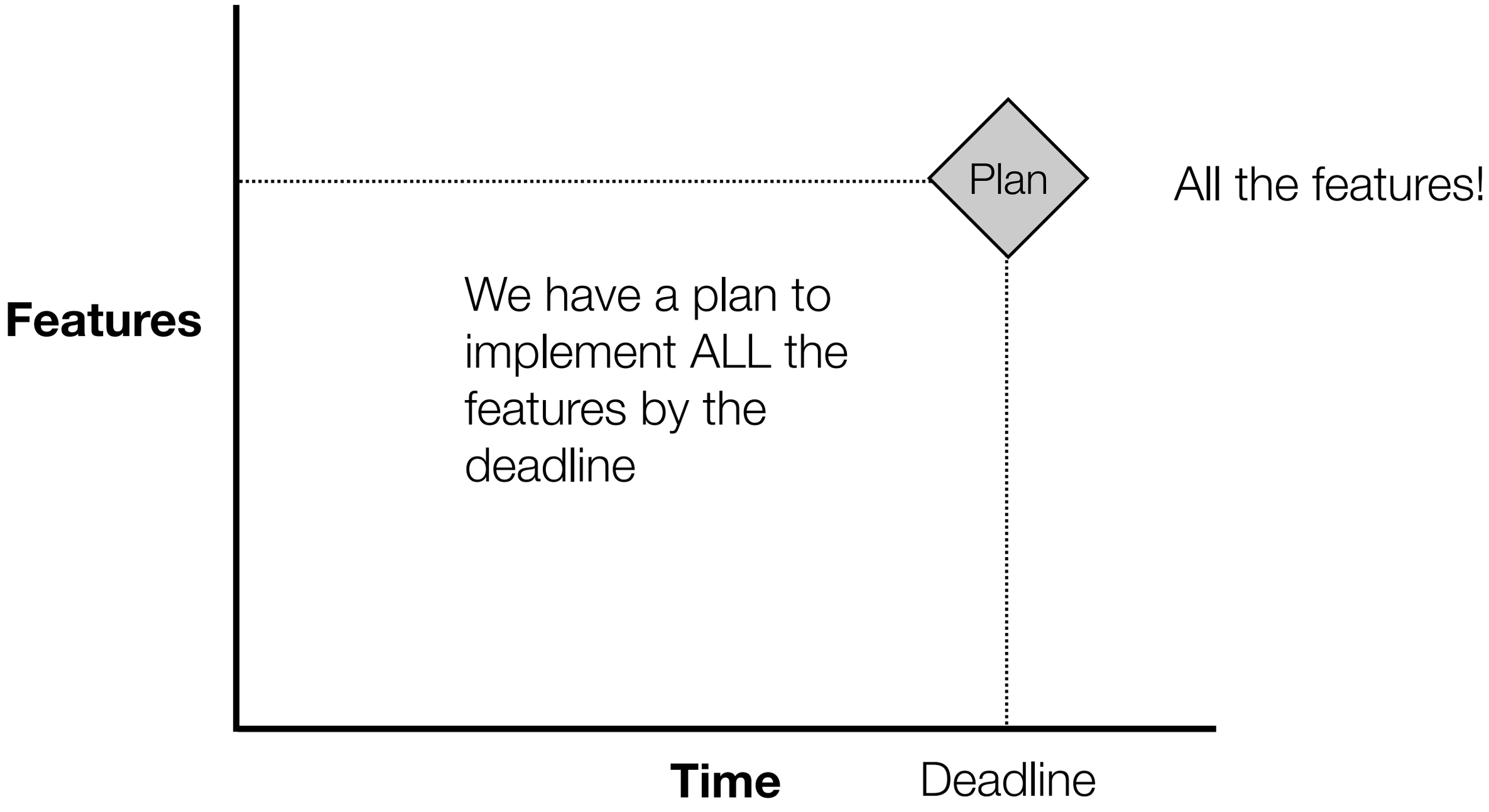
Guiding

- To create value for our customer/user
    - we need to guide our team
    - and, Jeffries asserts, guiding goes better feature by feature
- To see why, he makes the following argument

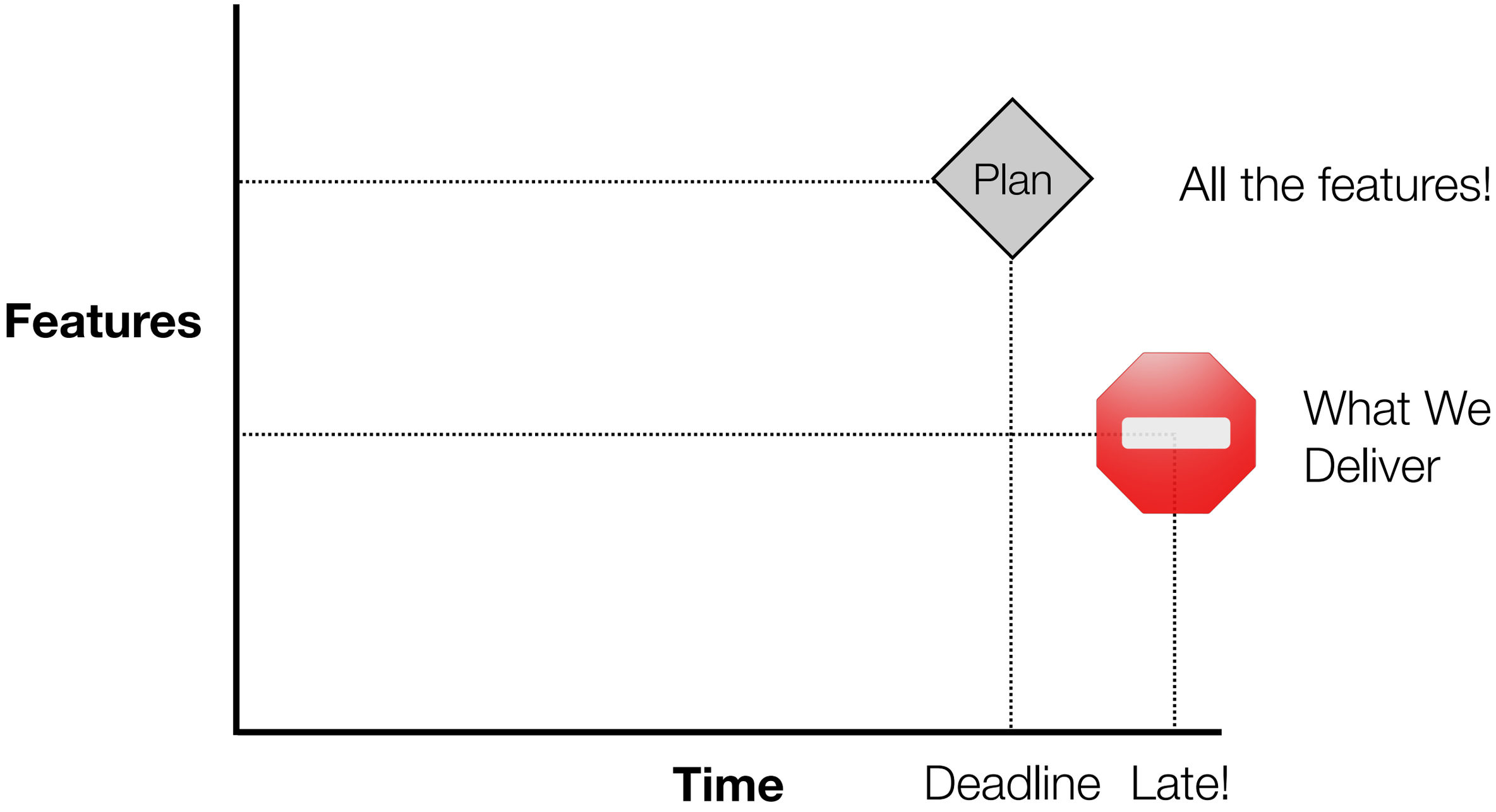
Every Project Ever



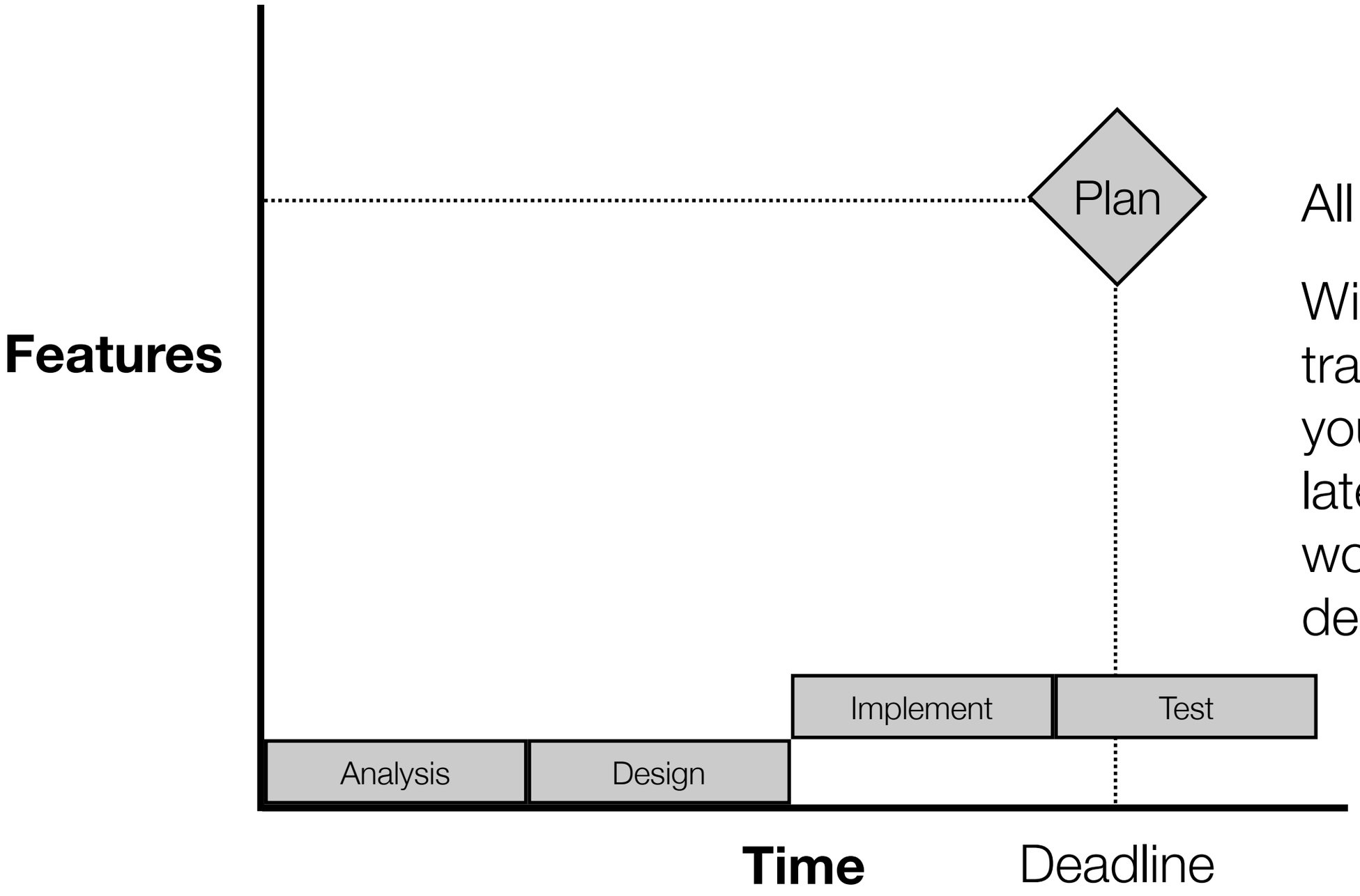
Every Project Ever



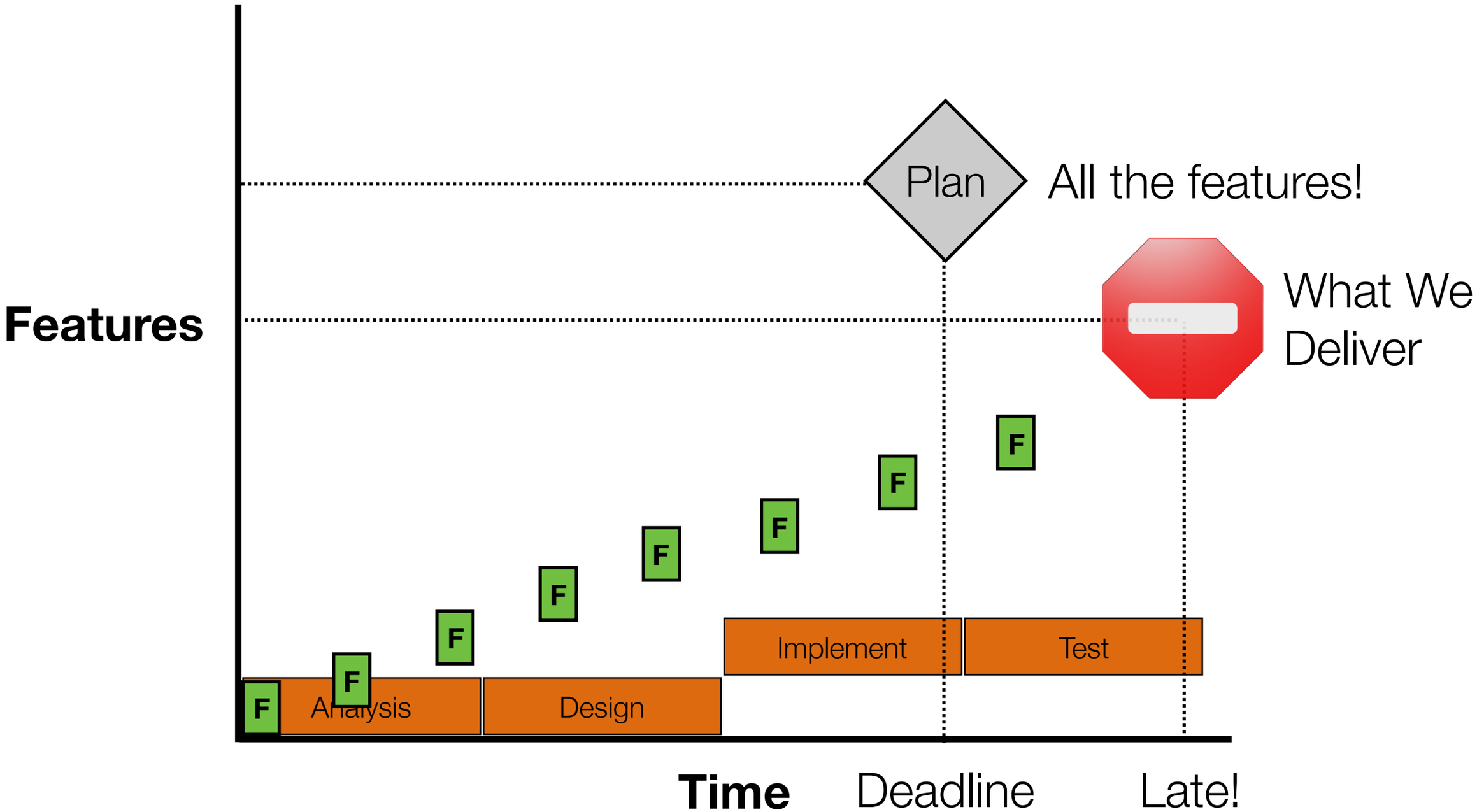
The Problem? Nothing ever goes to plan



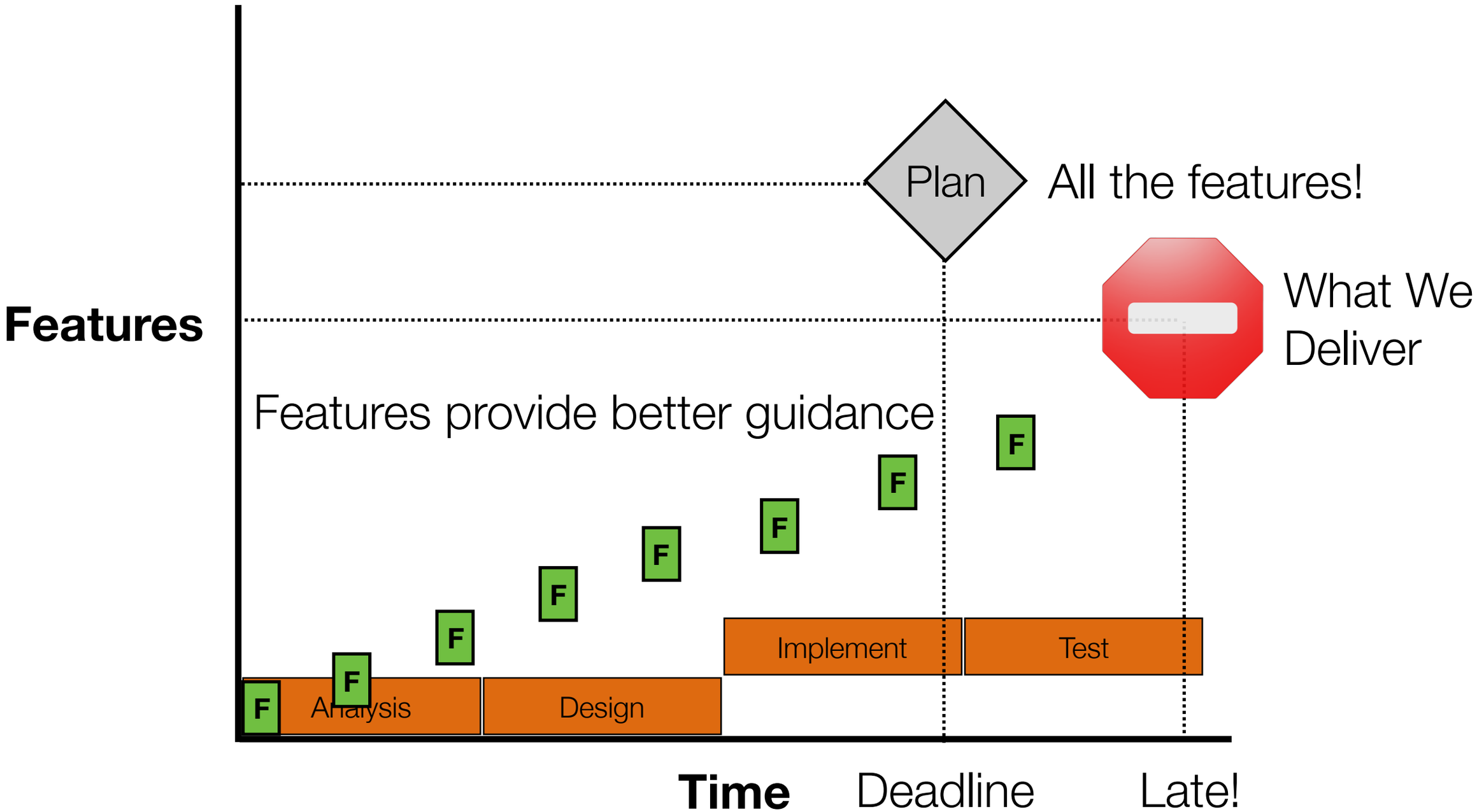
The problem with waterfall



The Natural Way (Features) Provides More Information Faster than Waterfall



The Natural Way (Features) Provides More Information Faster than Waterfall



Organizing by Feature

- We want to develop software by creating small, high-value features
  - We have seen that we get better guidance about our progress if we do
- How do we organize around features?
  - Here Jeffries is talking about how do you organize your software teams?
- The issue of team organization keeps appearing in our discussions
  - It really is an important decision that *influences everything* that a software development organization can accomplish
    - Who does what?
    - Who talks to whom?
    - Who is in charge?

Organizing by Skill

- Many organizations group teams by skill
    - This team does analysis
    - This team does design
    - This team does coding
    - This team does testing
- The problem?
    - You have multiple teams that have to be coordinated
        - Delays are built into your process since one team may not be ready to take work from the previous team; if the analysis team gets behind, EVERYONE is behind even before they do one thing for a project!

Organizing by Feature

- It should come as no surprise that Jeffries recommends that teams should be *organized by feature*
    - Each team should be small and should be assigned a feature
        - Consequence: the team needs all the people and all the skills it needs to create the feature
            - UX, analysis, design, QA, persistence, middleware, etc.
- Advantages
    - Multiple features can be built in parallel
    - Management knows how work is distributed
    - Each feature gets dedicated attention
    - responsibility for a feature and authority to implement it are aligned

Arguments Against (I)

- “Our company is not organized that way…”
  - Jeffries is arguing that your company needs to change
    - Using Brooks’s terminology, we can see that Jeffries is asserting that
      - teams organized by skill
        - or for reasons other than feature
      - are introducing accidental difficulties into your development process
- Why? You have built in delays into your process with team coordination that wouldn’t be there if you switched your technique
  - If a team is completely responsible for a feature, there’s no hand offs that need to be coordinated; the need for coordination goes down

Arguments Against (II)

- “We don’t have enough specialists…”
  - Jeffries is arguing to not become too fixated on the term “expert”
    - Sure, you won’t have enough UX or database experts to have one/team
    - But you probably have people who are “very good” at UX or “good” at databases and interested in learning more
      - Let them take on those roles
        - in addition, people are *spectrums* not points
          - “I’m good at programming; I know how to configure and use several databases; I know some front-end technologies”
- Now, let your team members self identify as being *really interested* in one role over another; let them talk to other people interested in that same role => form a community around it and let them teach each other

Planning

- When we start a project, we have a “product vision”
  - Visions are grandiose; they are not explained in terms of features
- So, how do we transform a vision into the features we need to
  - create software by the frequent delivery of high-value features
  - guiding our projects via our progress on completing those features
  - and organizing our teams to build those features?
- Planning!
  - As Jeffries says, planning is indispensable
    - He quotes a WW II, US military general: “Plans are useless but planning is indispensable”
    - What does this mean?

The utility of planning

- Planning as an activity keeps us oriented on our goals
  - What are we trying to accomplish? By when? For how much money?
  - This is something we ALWAYS need to be aware of
- BUT
  - making detailed plans upfront can get us into trouble
    - “Plans are useless…”
- BUT the act of planning is always useful… we need to consider lots of ideas to find the good ones, we need to have discussions to identify our high value features, but we have to do so knowing things will change and thus we don’t clean to our old (now useless) plan but instead we create a new plan!

Planning by Jeffries

- “Here’s a better way: set a time and money budget; produce the most valuable features first; keep the product ready to ship at any time—and stop when the clock runs out. Quite likely we’ll even stop before the deadline, because we’ve already got the important stuff done. We deliver the bulk of the value in far less time, for far less money.”

- To get started, Jeffries now starts to lay out the a fairly typical approach to Agile software development

  - Hmm, I wonder why it would come off as fairly typical?

    - Oh right, he helped to invent it in the first place! 😃

Agile (as presented by Jeffries)

- Agile development is the act of continuous planning by feature splitting
  - We will always be planning because we're always acting on implementing the highest value feature that can be implemented in a short amount of time
    - These features will be tracked via user stories; each story should describe a feature that would take two to three days to implement
      - Jeffries recommends AGAINST converting stories to tasks
        - because tasks will take the customer OUT of the process
      - Instead, convert big stories into smaller stories until each one takes only 2 to 3 points to implement
- These features are then implemented in the context of iterations of a few weeks in length

Estimates (Jeffries doesn't like them!)

- "We plan each iteration right before it begins. To decide how much work to take on, we need to understand the work. As a team, we discuss the work. The team's Product Champion presents one feature at a time, followed by a brief team dis- cussion about what it'll take to accomplish the feature. Everyone stays engaged, and the team understands the feature before committing to do it. I don't recommend estimating the individual work pieces at all. Instead, understand them, and then look at the aggregate and decide how much of it the team can do. If estimates really help the team, go for it. But be careful! The point isn't to make good estimates—the point is to do good work at a consistent pace."

- He further describes generating estimates as pernicious since it is an activity that takes you away from the work and they are almost always wrong
  - it's more important, he asserts, to get work done at a consistent pace
    - just always be making progress towards your goal

Stretch Goals

- During short-term planning (i.e. planning an iteration), it's tempting to add "stretch goals" or to try for "just one more feature"
    - Jeffries says "Please do not do this. It's devastatingly destructive."
- Why?
    - Because as soon as you put this in front of a team, they will want to do it
        - They will push themselves to achieve that extra goal; they will hurry and inject defects they wouldn't have otherwise;
            - once you have those defects, you have to fix them and that puts you behind; your "hurrying" ends up slowing you down
- The overall goal is SUSTAINABLE, CONSISTENT delivery of features

Wrapping Up

- We've now reviewed the first three layers of the "natural way" for developing software
    - guiding, organizing, and planning
- These our the foundational practices that lead to achieving our goal
    - producing value for our customer/user via the consistent delivery of high-value features
- We will discuss the next three layers next time!