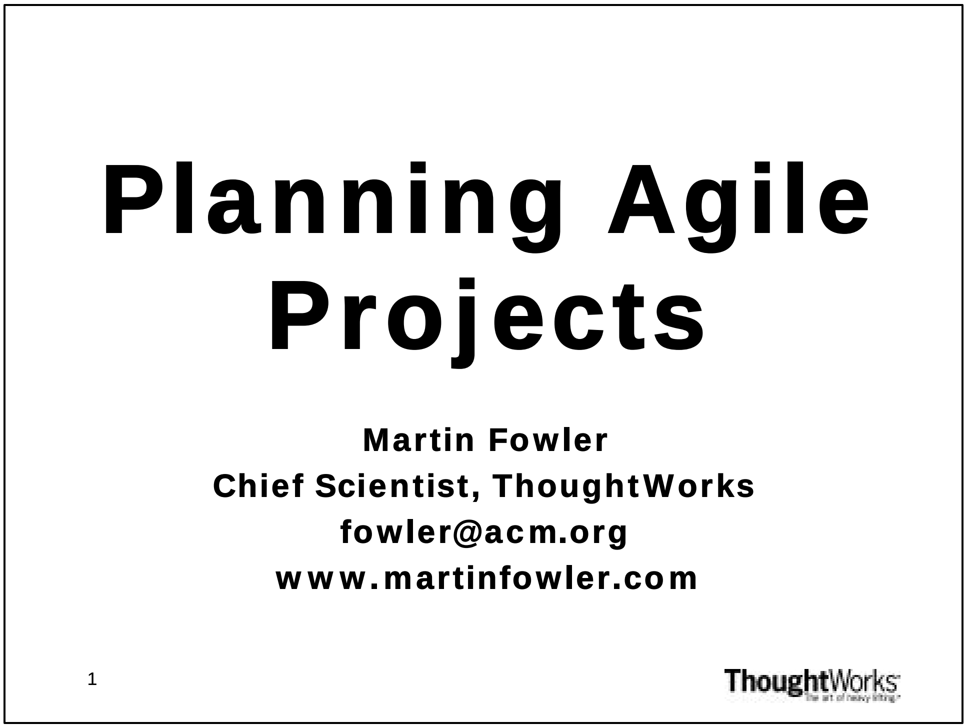
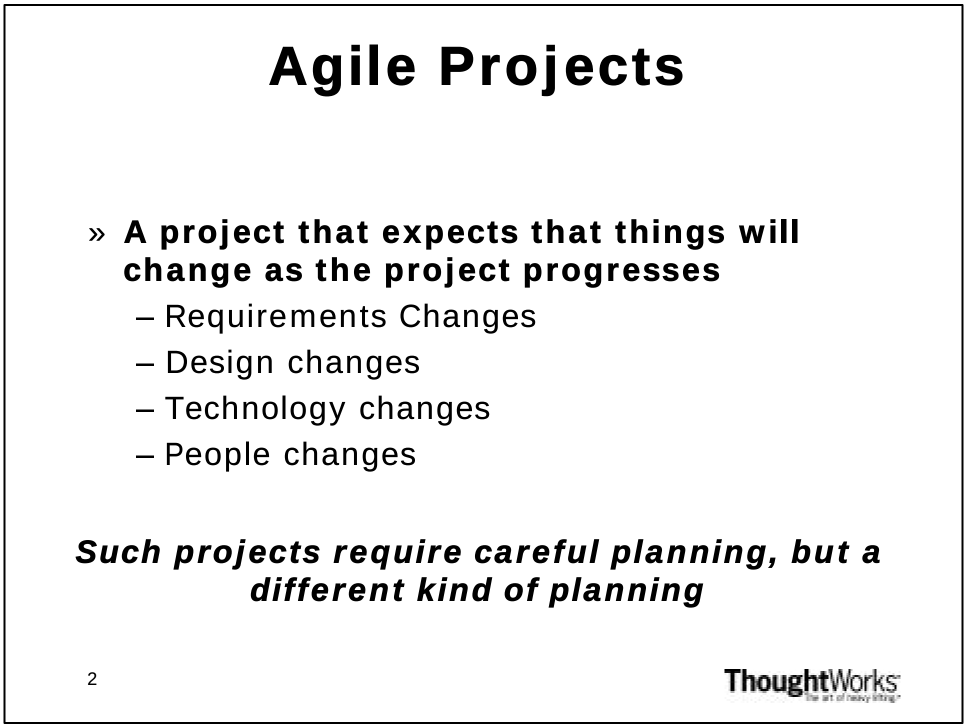


Planning Agile Projects

Martin Fowler
Chief Scientist, ThoughtWorks
fowler@acm.org
www.martinfowler.com



Agile Projects

» **A project that expects that things will change as the project progresses**
  - Requirements Changes
  - Design changes
  - Technology changes
  - People changes

Such projects require careful planning, but a different kind of planning

Agile Methodologies

- » **New breed of methodologies that have discipline without bureaucracy**
- » **E.g.:**
  - – XP (Extreme Programming)
  - – Crystal / Highsmith ASD
  - – Feature Driven Development
  - – SCRUM
  - – DSDM

Agile Manifesto

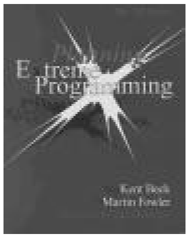
We Value:

Individuals and Interactions	***over***	**Process and Tools**
Working Software	***over***	**Comprehensive Documents**
Customer Collaboration	***over***	**Contract Negotiation**
Responding to Change	***over***	**Following a Plan**

www.agileAlliance.org

Agile Planning

- » **Planning with the expectation of change**
- » **Ideas based on those in Planning XP**
  - – Concepts are effective in any agile environment
  - – Add in key ideas from other agile processes
- » **Beck and Fowler, *Planning Extreme Programming*, Addison-Wesley, 2001**



Agile and Predictive

Agile

- » **Only rough plans beyond a few months**
  - – Low precision
- » **Long term plans are expected to change**
  - – Things don’t go according to plan
- » **Detailed plans in short horizons**
  - – Two weeks to two months

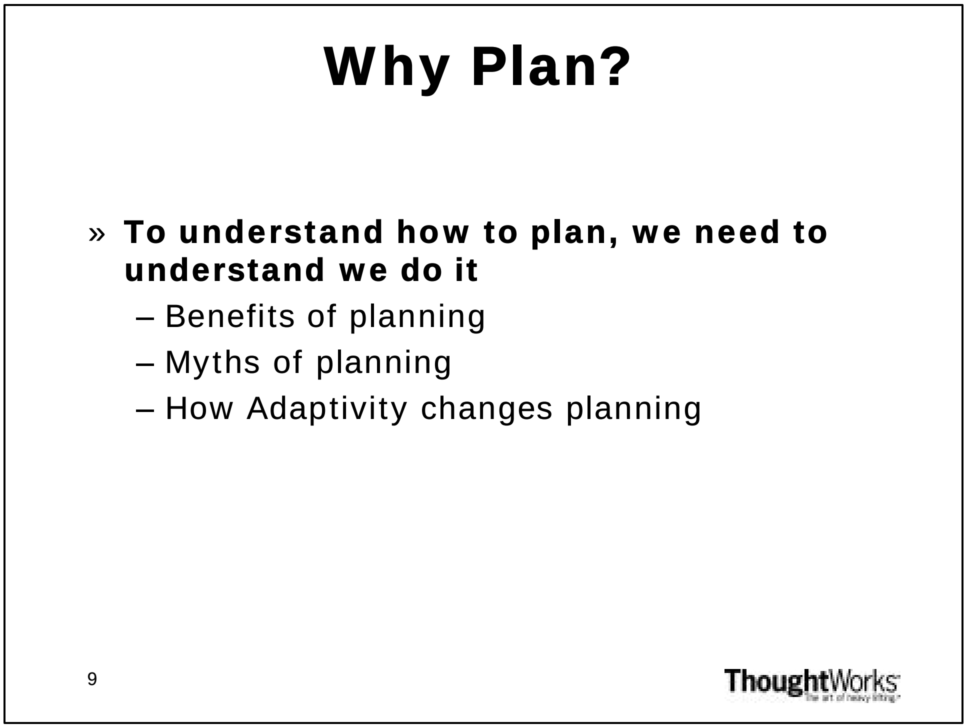
Predicitve

- » **Figure out everything that needs to be done before beginning**
- » **Figure out best way of doing it**
- » **Long planning horizon of a year or more**
- » **Deviations from plan are problems**

Do you need Agile?

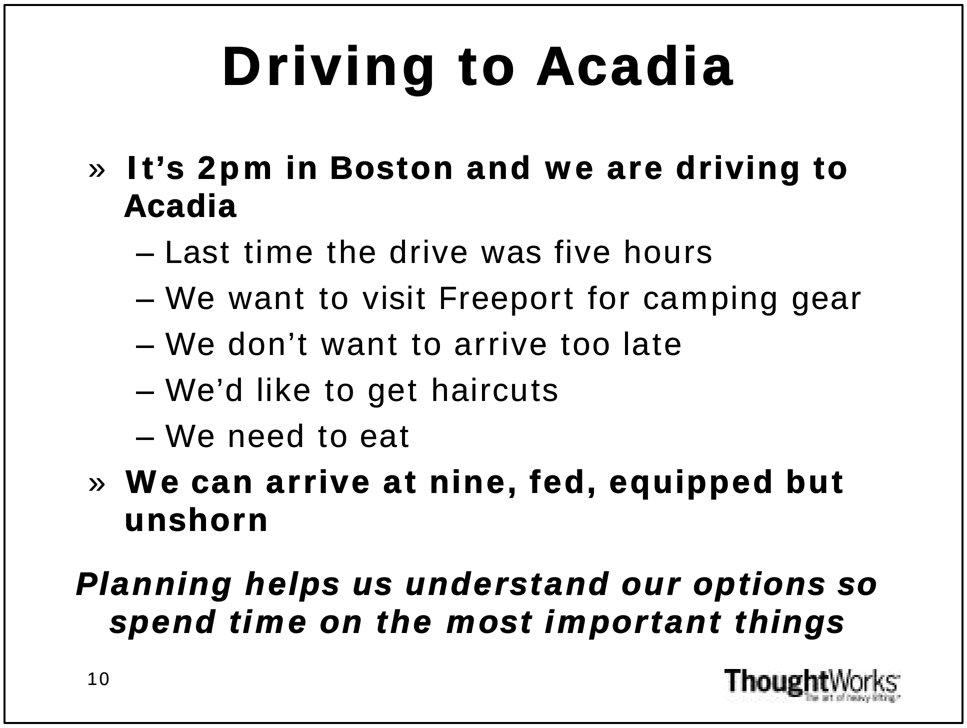
- **What would it take to requirements stable?**
  - Do people understand what's needed now?
  - Do you understand costs?
  - Is the business changing?
- **Would it be good to have stable requirements?**
  - Is a late change in requirements a competitive advantage?

Why Plan?



Why Plan?

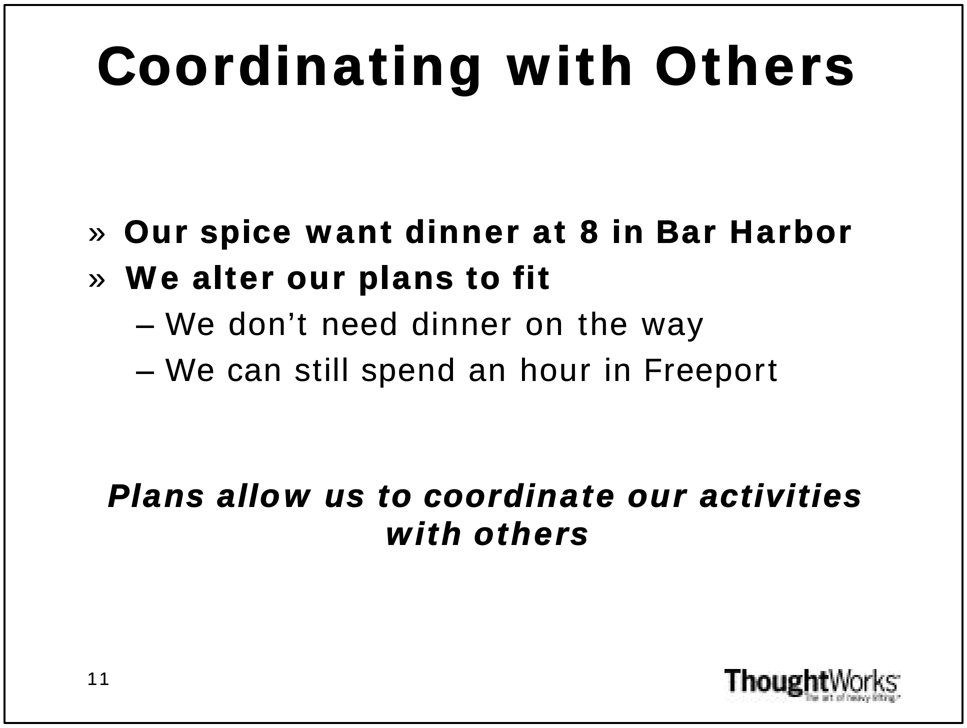
- » **To understand how to plan, we need to understand we do it**
  - – Benefits of planning
  - – Myths of planning
  - – How Adaptivity changes planning



Driving to Acadia

- » **It's 2pm in Boston and we are driving to Acadia**
  - – Last time the drive was five hours
  - – We want to visit Freeport for camping gear
  - – We don't want to arrive too late
  - – We'd like to get haircuts
  - – We need to eat
- » **We can arrive at nine, fed, equipped but unshorn**

Planning helps us understand our options so spend time on the most important things



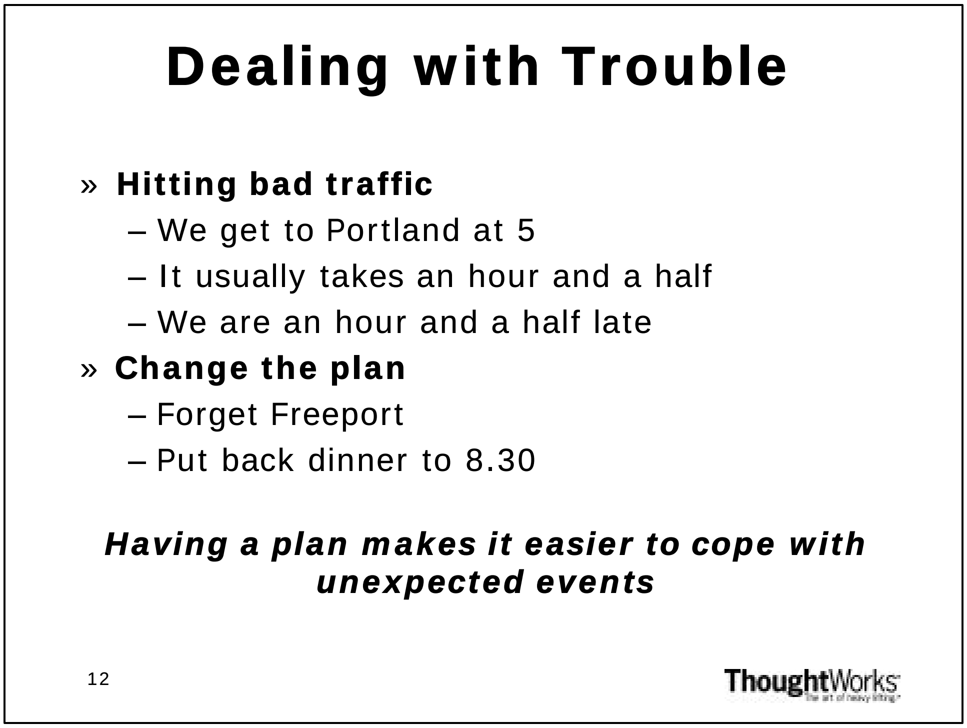
Coordinating with Others

» **Our spice want dinner at 8 in Bar Harbor**

» **We alter our plans to fit**

- We don’t need dinner on the way
- We can still spend an hour in Freeport

Plans allow us to coordinate our activities with others



Dealing with Trouble

» **Hitting bad traffic**

- We get to Portland at 5
- It usually takes an hour and a half
- We are an hour and a half late

» **Change the plan**

- Forget Freeport
- Put back dinner to 8.30

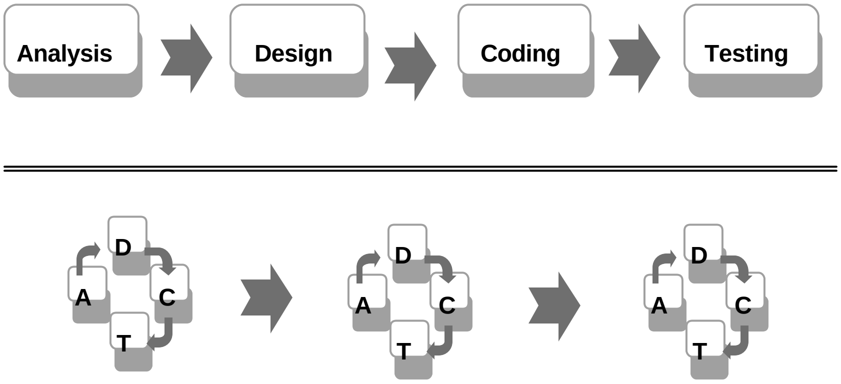
Having a plan makes it easier to cope with unexpected events

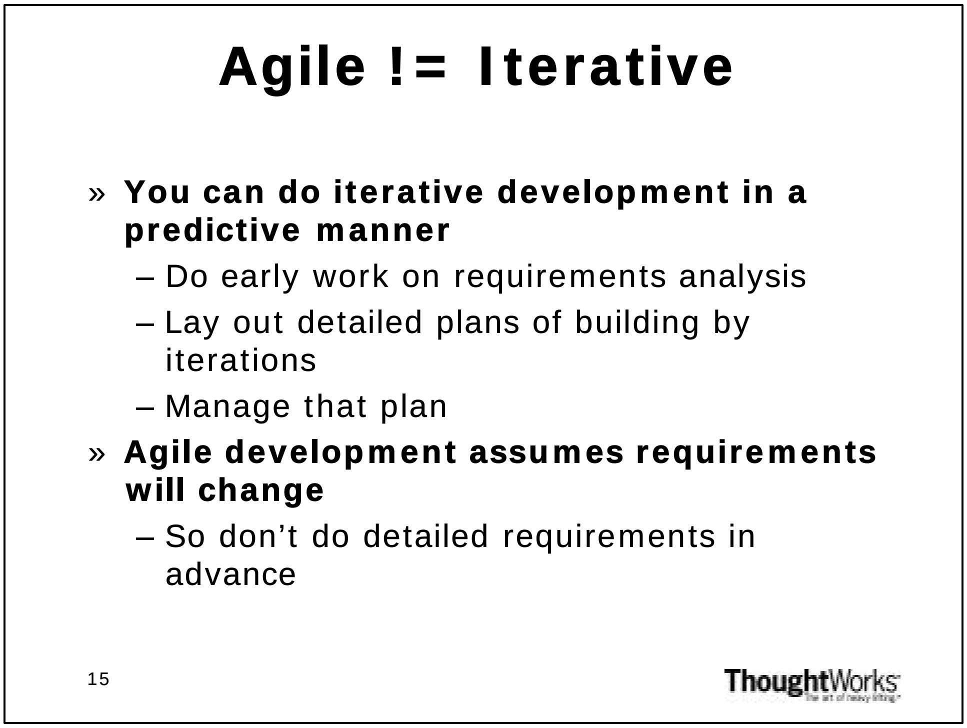
Planning needs tracking

» **You need to know where you are**
- Where are we? (Portland)
- What time is it? (5pm)
- How long did it take us last time (hour and a half)

Must have clear picture of visibility.
This is hard for software

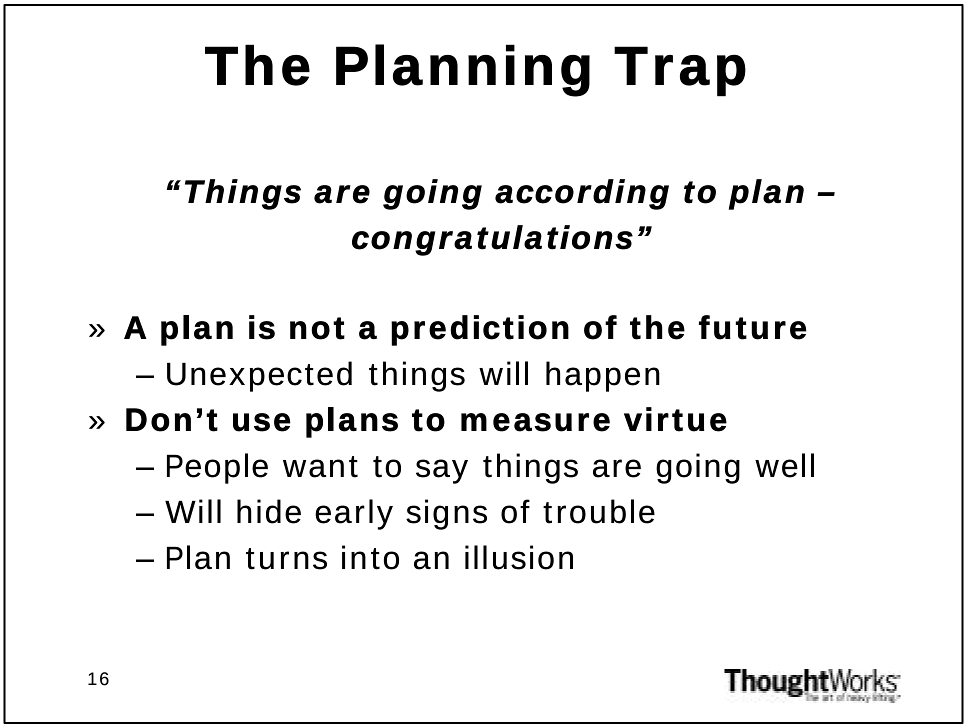
Waterfall and Iterative





Agile != Iterative

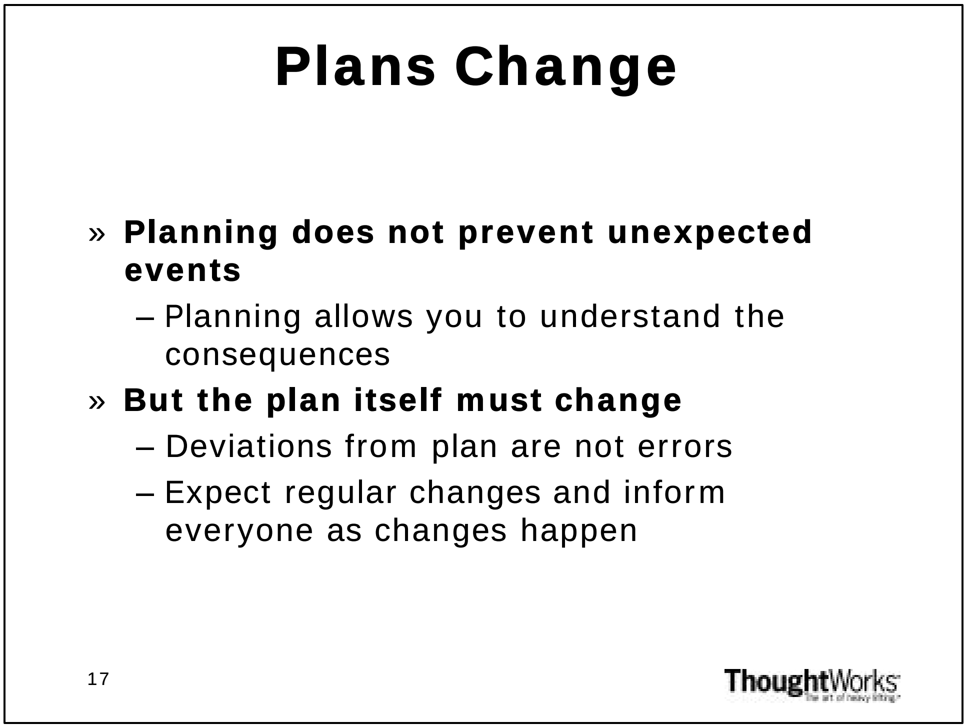
- **You can do iterative development in a predictive manner**
  - Do early work on requirements analysis
  - Lay out detailed plans of building by iterations
  - Manage that plan
- **Agile development assumes requirements will change**
  - So don't do detailed requirements in advance



The Planning Trap

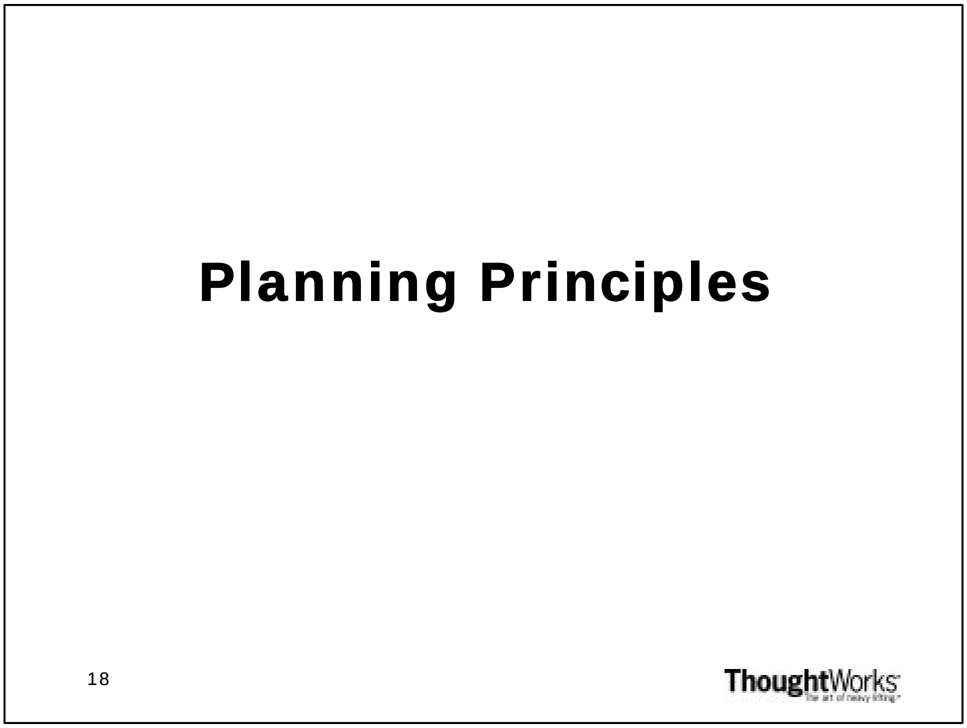
"Things are going according to plan – congratulations"

- **A plan is not a prediction of the future**
  - Unexpected things will happen
- **Don't use plans to measure virtue**
  - People want to say things are going well
  - Will hide early signs of trouble
  - Plan turns into an illusion

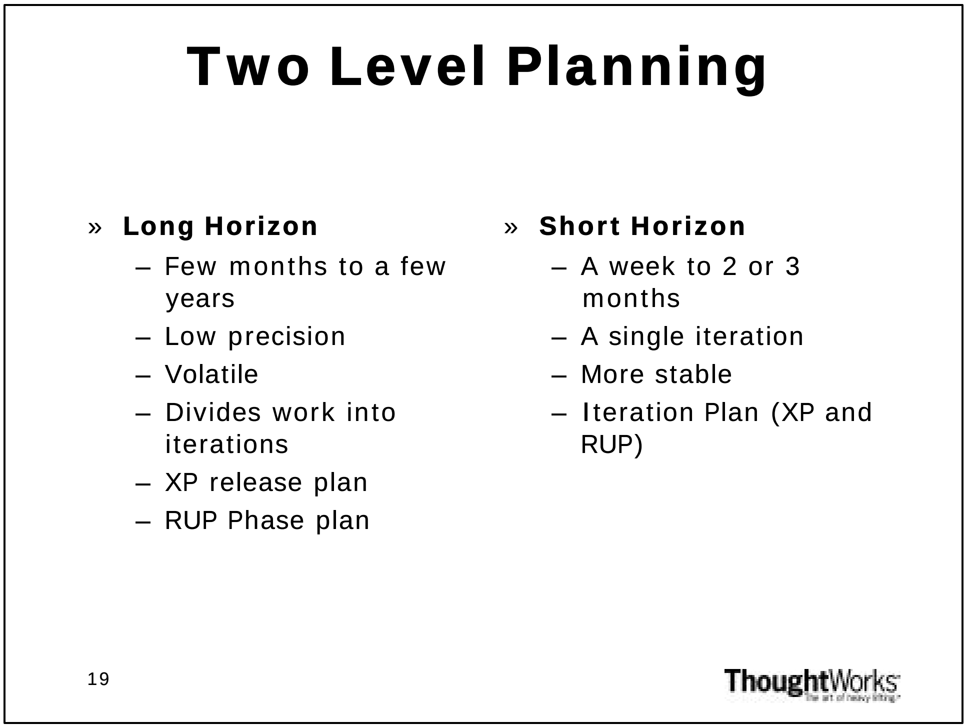


Plans Change

- **Planning does not prevent unexpected events**
  - Planning allows you to understand the consequences
- **But the plan itself must change**
  - Deviations from plan are not errors
  - Expect regular changes and inform everyone as changes happen



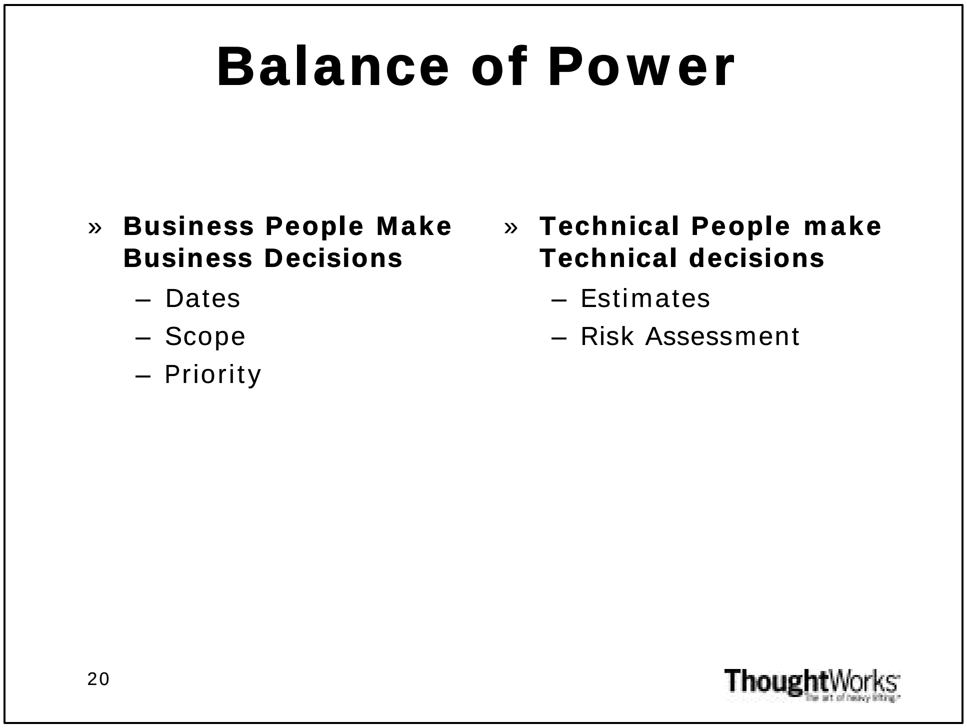
Planning Principles



Two Level Planning

» **Long Horizon**
- Few months to a few years
- Low precision
- Volatile
- Divides work into iterations
- XP release plan
- RUP Phase plan

» **Short Horizon**
- A week to 2 or 3 months
- A single iteration
- More stable
- Iteration Plan (XP and RUP)



Balance of Power

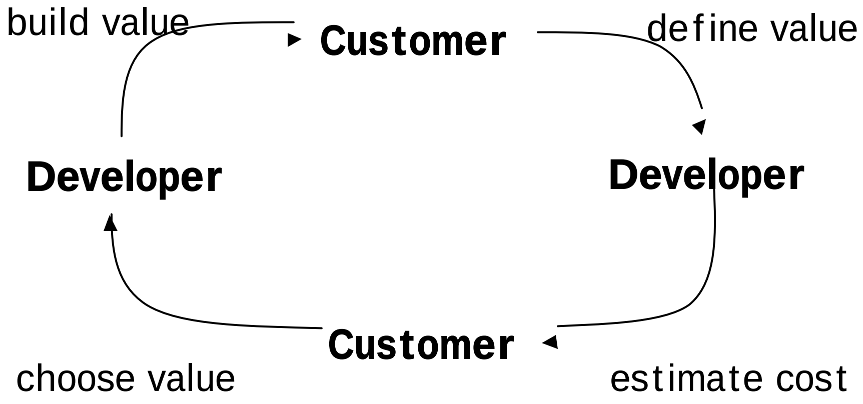
» **Business People Make Business Decisions**
- Dates
- Scope
- Priority

» **Technical People make Technical decisions**
- Estimates
- Risk Assessment

Business Leadership

- **Needs efficient decision making from business**
  - XP's Customer
  - Product Manager
  - Part of team
- **Skills**
  - Understands domain
  - Understands how software can add value
  - Determined to deliver a little value regularly
  - Can make decisions on priority
  - Accepts responsibility for project outcome

The Circle of Life



build value

Customer

define value

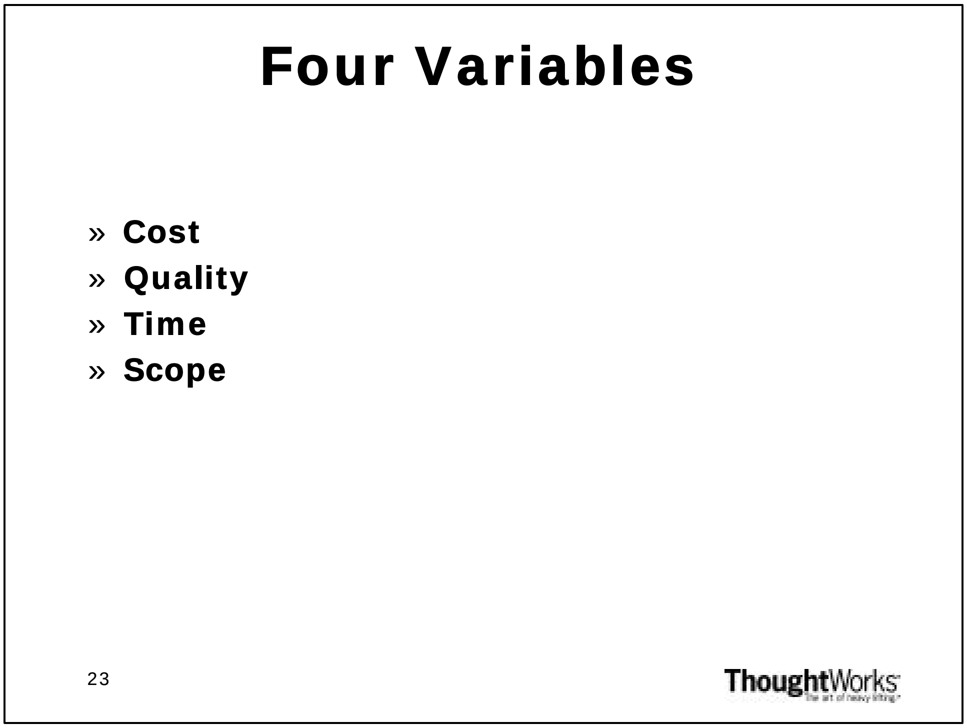
Developer

Developer

Customer

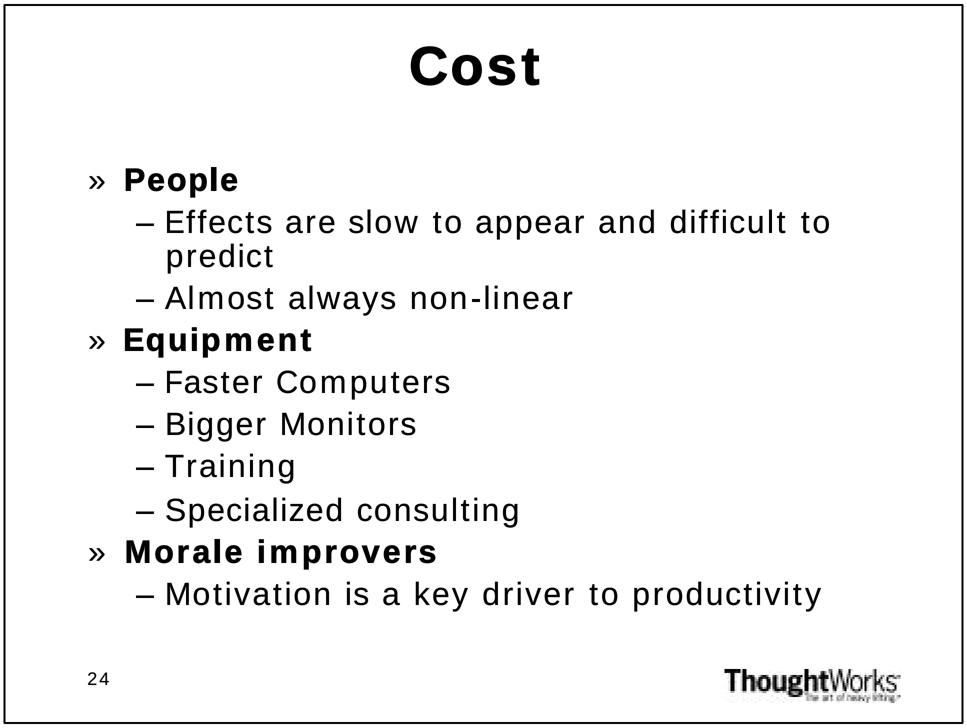
choose value

estimate cost



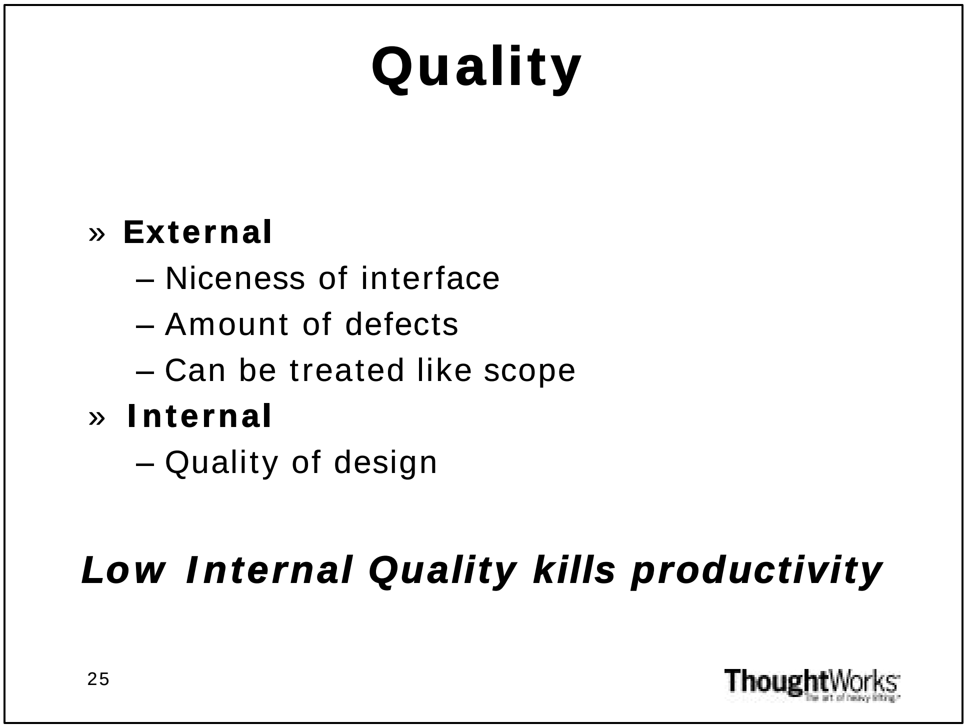
Four Variables

- » **Cost**
- » **Quality**
- » **Time**
- » **Scope**



Cost

- » **People**
  - – Effects are slow to appear and difficult to predict
  - – Almost always non-linear
- » **Equipment**
  - – Faster Computers
  - – Bigger Monitors
  - – Training
  - – Specialized consulting
- » **Morale improvers**
  - – Motivation is a key driver to productivity

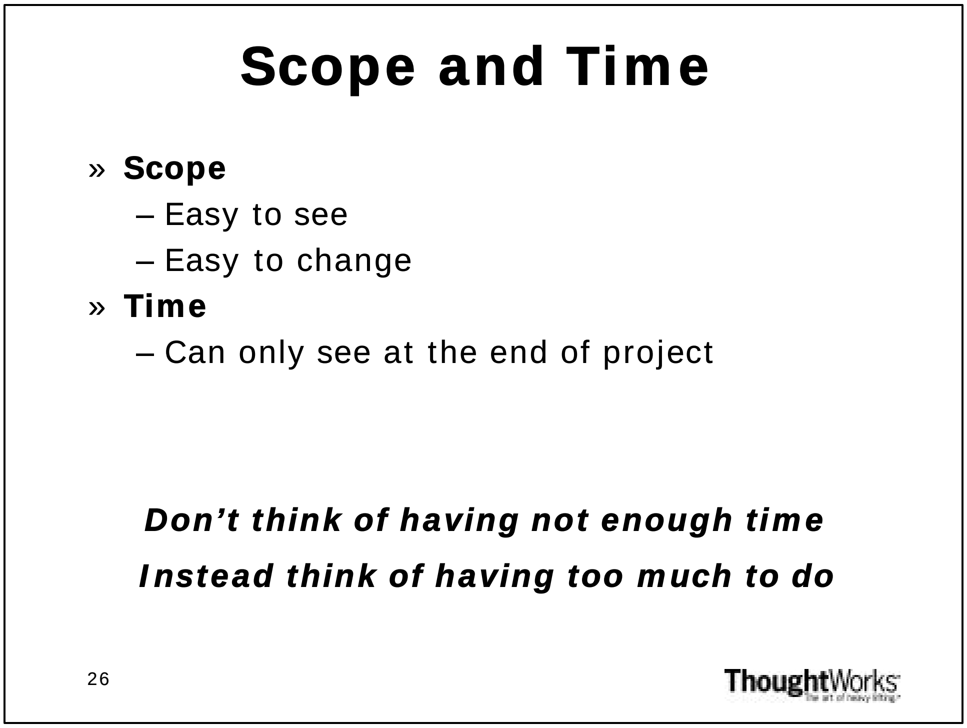


Quality

- **External**
  - – Niceness of interface
  - – Amount of defects
  - – Can be treated like scope
- **Internal**
  - – Quality of design

Low Internal Quality kills productivity

25



Scope and Time

- **Scope**
  - – Easy to see
  - – Easy to change
- **Time**
  - – Can only see at the end of project

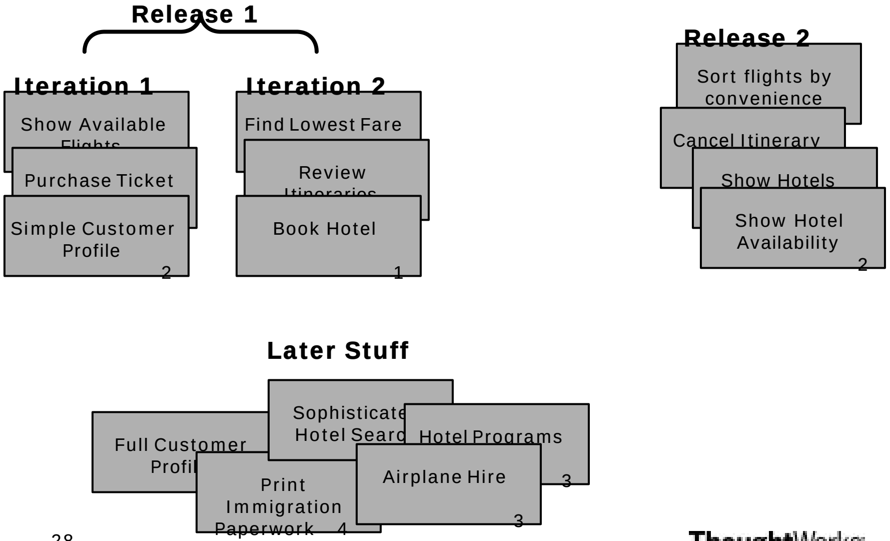
Don’t think of having not enough time

Instead think of having too much to do

26

Release Planning

Release Plan



Release 1

Iteration 1

Show Available Flights

Purchase Ticket

Simple Customer Profile 2

Iteration 2

Find Lowest Fare

Review Itineraries

Book Hotel 1

Release 2

Sort flights by convenience

Cancel Itinerary

Show Hotels

Show Hotel Availability 2

Later Stuff

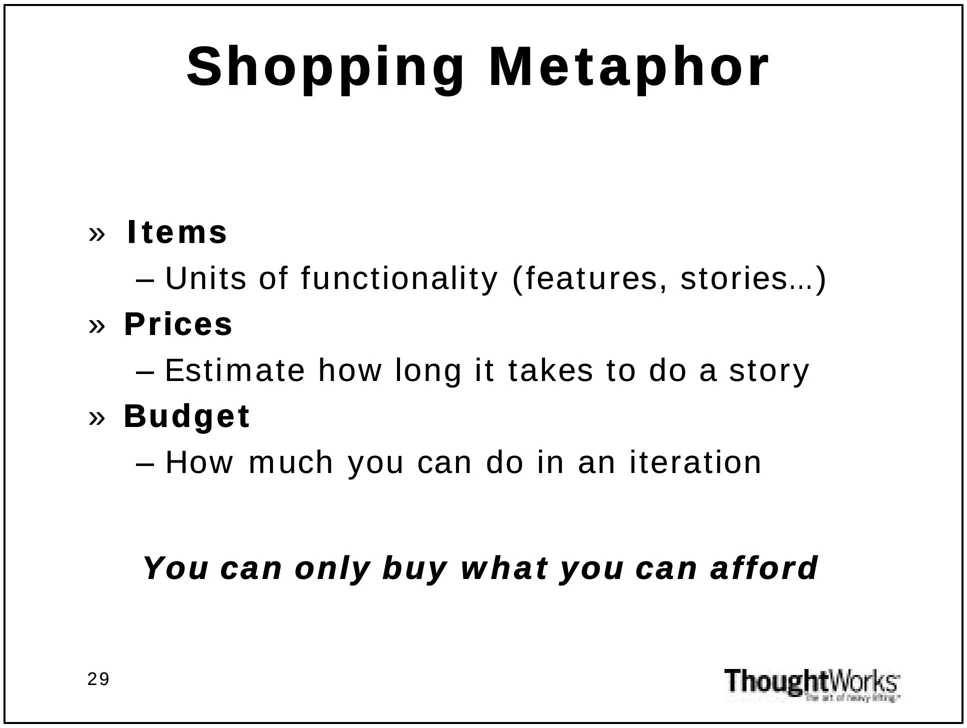
Full Customer Profil

Sophisticate Hotel Searc

Hotel Programs 3

Print Immigration Paperwork 4

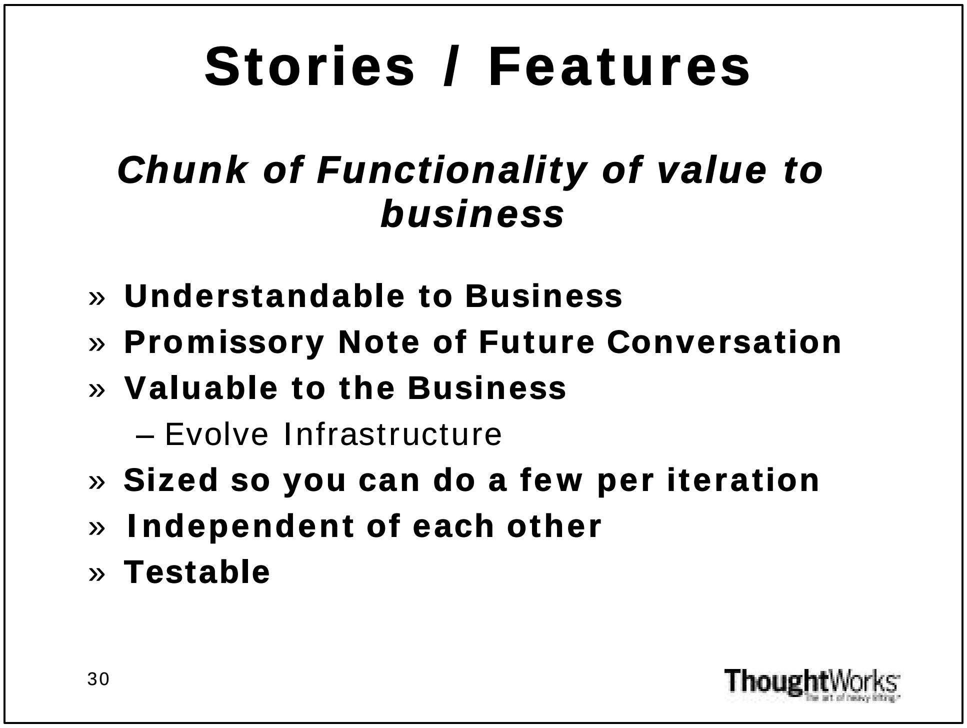
Airplane Hire 3



Shopping Metaphor

- » **Items**
  - – Units of functionality (features, stories...)
- » **Prices**
  - – Estimate how long it takes to do a story
- » **Budget**
  - – How much you can do in an iteration

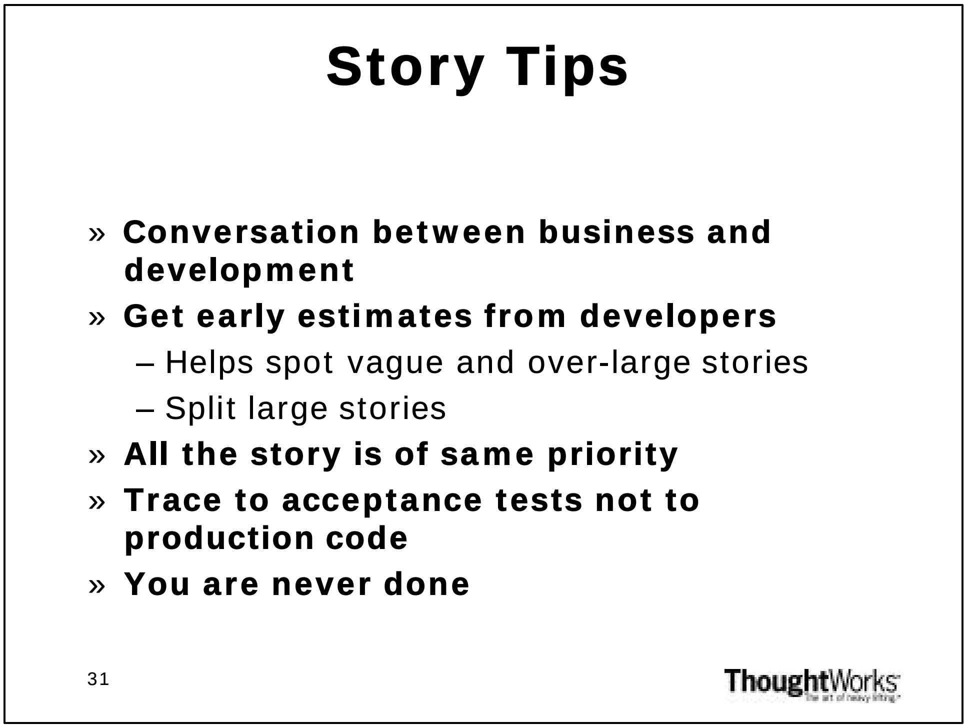
You can only buy what you can afford



Stories / Features

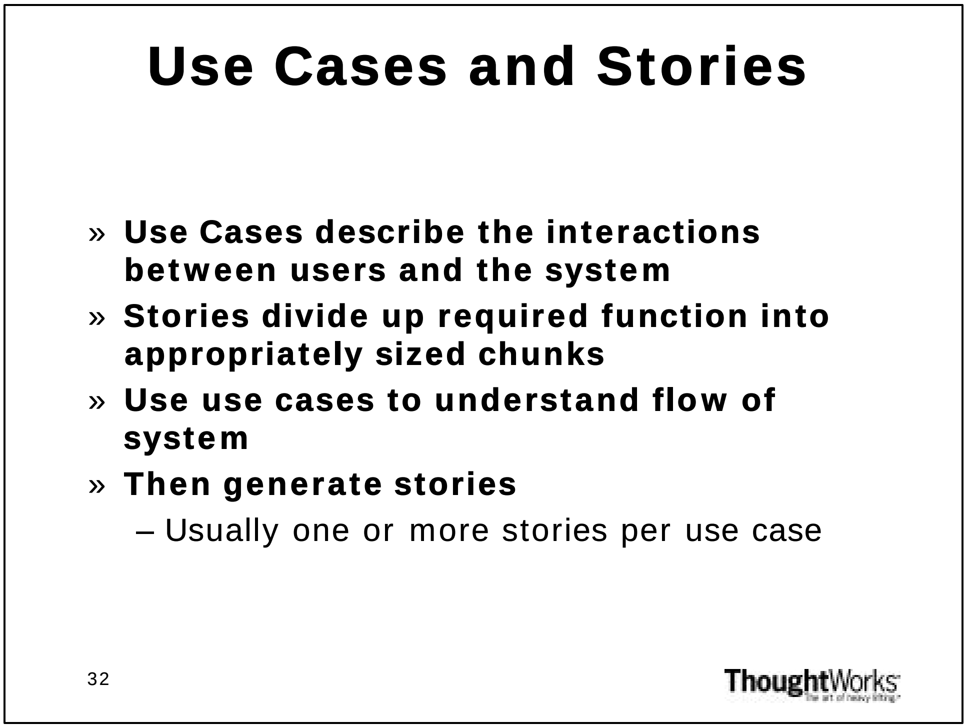
Chunk of Functionality of value to business

- » **Understandable to Business**
- » **Promissory Note of Future Conversation**
- » **Valuable to the Business**
  - – Evolve Infrastructure
- » **Sized so you can do a few per iteration**
- » **Independent of each other**
- » **Testable**



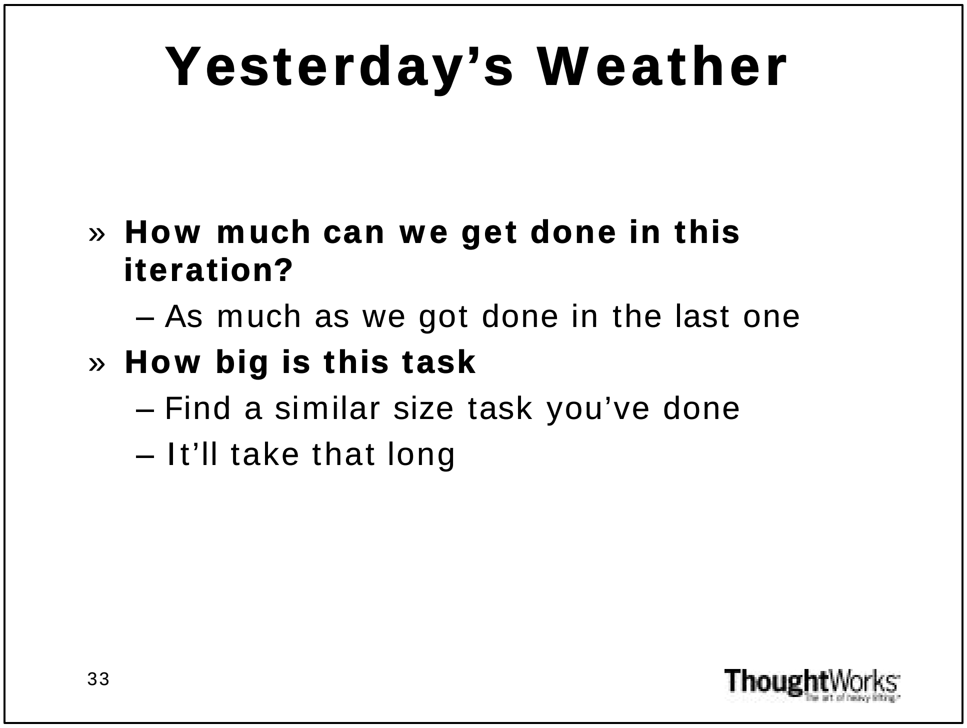
Story Tips

- » **Conversation between business and development**
- » **Get early estimates from developers**
  - – Helps spot vague and over-large stories
  - – Split large stories
- » **All the story is of same priority**
- » **Trace to acceptance tests not to production code**
- » **You are never done**



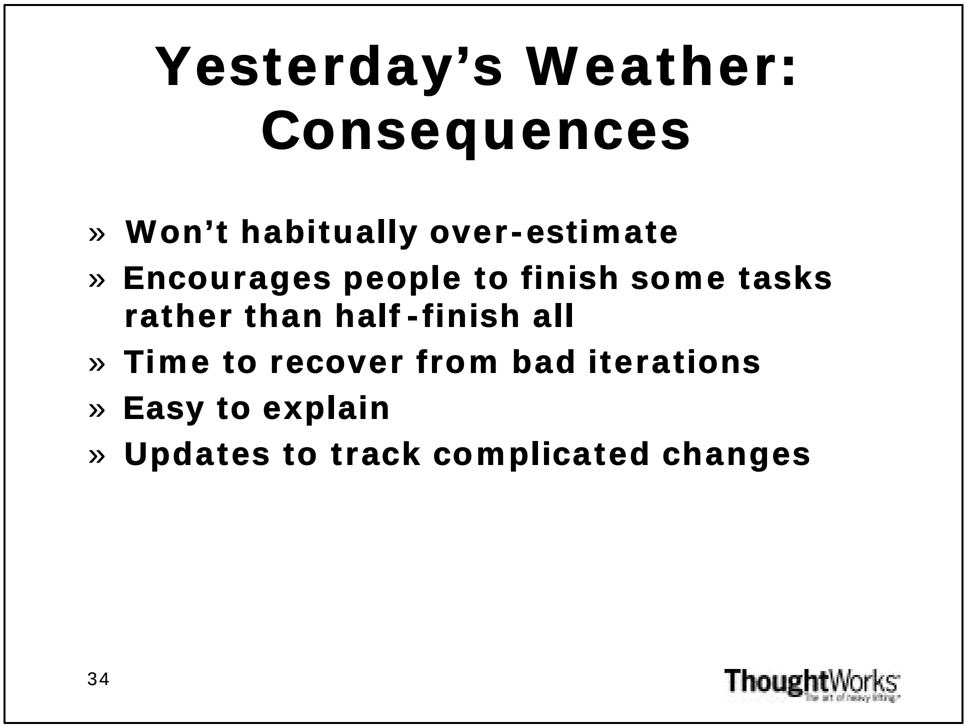
Use Cases and Stories

- » **Use Cases describe the interactions between users and the system**
- » **Stories divide up required function into appropriately sized chunks**
- » **Use use cases to understand flow of system**
- » **Then generate stories**
  - – Usually one or more stories per use case



Yesterday’s Weather

- » **How much can we get done in this iteration?**
  - – As much as we got done in the last one
- » **How big is this task**
  - – Find a similar size task you’ve done
  - – It’ll take that long



Yesterday’s Weather: Consequences

- » **Won’t habitually over-estimate**
- » **Encourages people to finish some tasks rather than half-finish all**
- » **Time to recover from bad iterations**
- » **Easy to explain**
- » **Updates to track complicated changes**

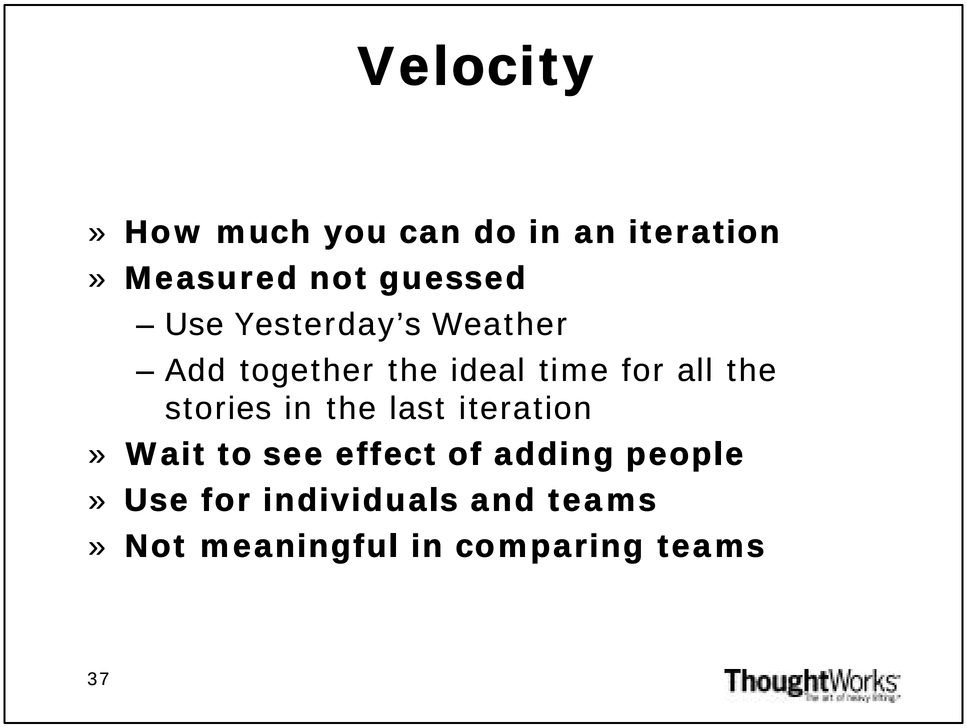
Estimating Stories

- » Find something you did that is of around the same size
- » Look at records to see how long it took
- » That's your estimate

Units of Estimation

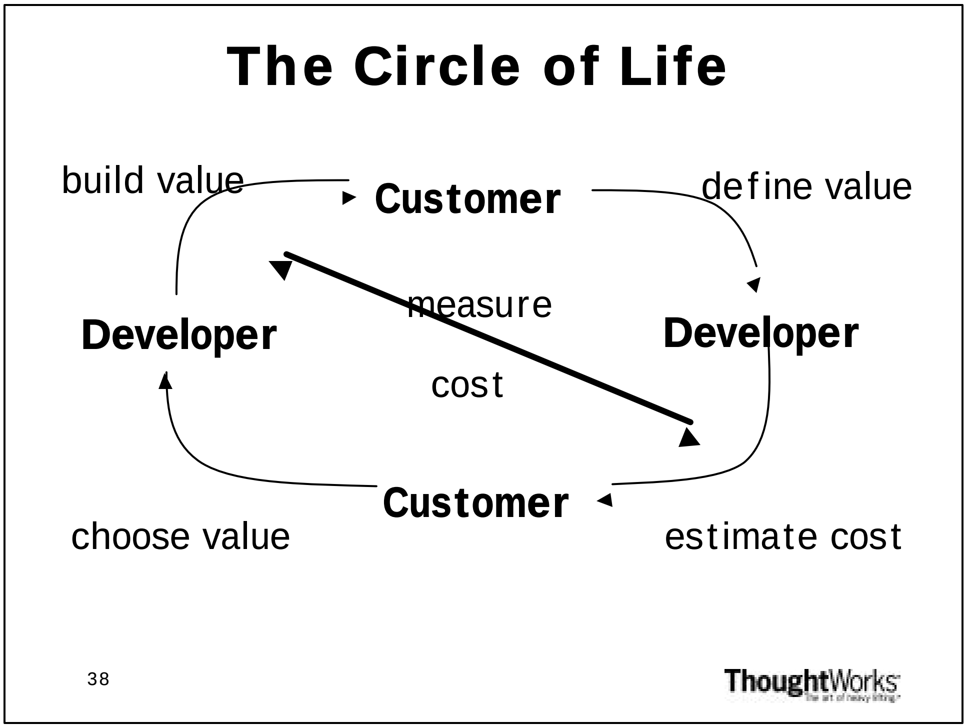
- » Ideal Time
  - – How much effort it would take without distraction
  - – 5 ideal days == 5 ideal development days
- » Gummi Bears

The units don't matter as long as they are consistent

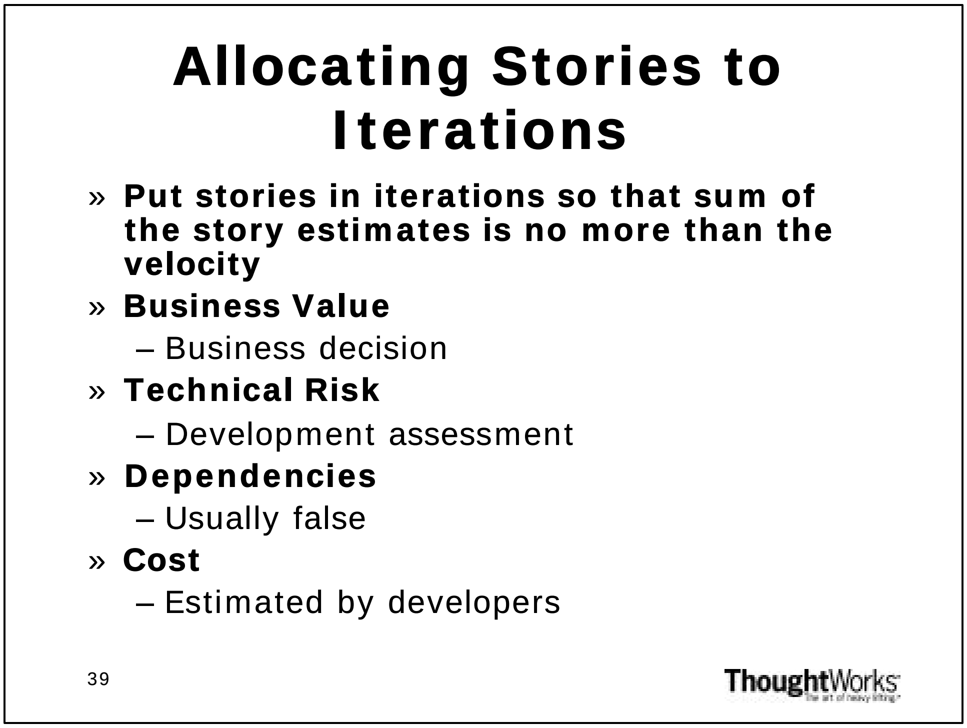


Velocity

- » **How much you can do in an iteration**
- » **Measured not guessed**
  - – Use Yesterday’s Weather
  - – Add together the ideal time for all the stories in the last iteration
- » **Wait to see effect of adding people**
- » **Use for individuals and teams**
- » **Not meaningful in comparing teams**

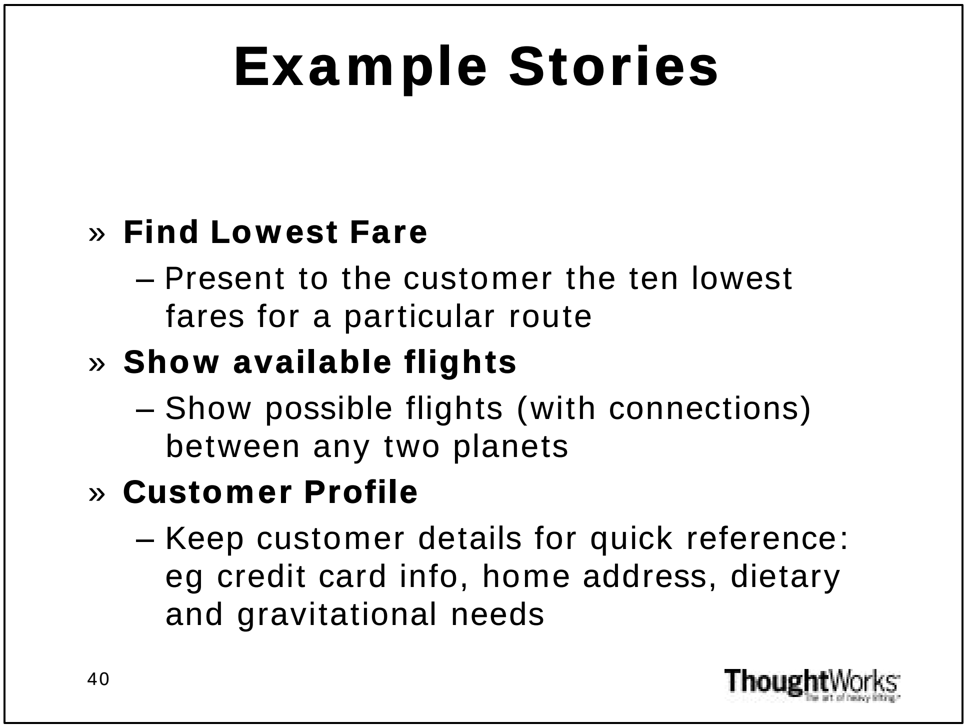


The Circle of Life



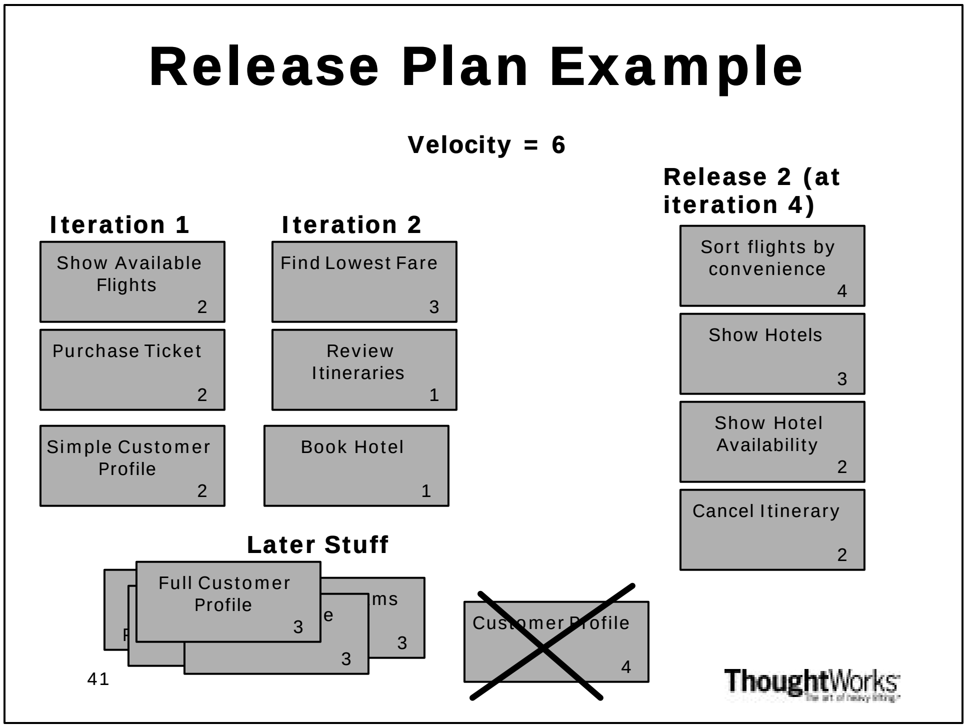
Allocating Stories to Iterations

- **Put stories in iterations so that sum of the story estimates is no more than the velocity**
- **Business Value**
  - Business decision
- **Technical Risk**
  - Development assessment
- **Dependencies**
  - Usually false
- **Cost**
  - Estimated by developers



Example Stories

- **Find Lowest Fare**
  - Present to the customer the ten lowest fares for a particular route
- **Show available flights**
  - Show possible flights (with connections) between any two planets
- **Customer Profile**
  - Keep customer details for quick reference: eg credit card info, home address, dietary and gravitational needs



Release Plan Example

Velocity = 6

Iteration 1

Story	Points
Show Available Flights	2
Purchase Ticket	2
Simple Customer Profile	2

Iteration 2

Story	Points
Find Lowest Fare	3
Review Itineraries	1
Book Hotel	1

Release 2 (at iteration 4)

Story	Points
Sort flights by convenience	4
Show Hotels	3
Show Hotel Availability	2
Cancel Itinerary	2

Later Stuff

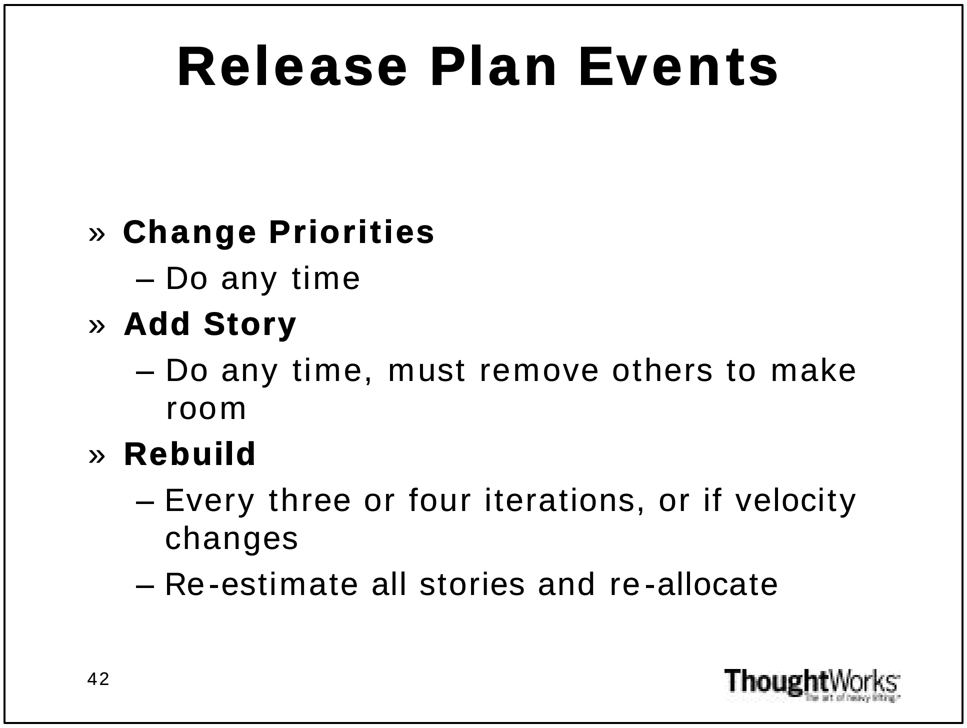
Full Customer Profile 3

ms 3

e 3

~~Customer Profile~~ 4

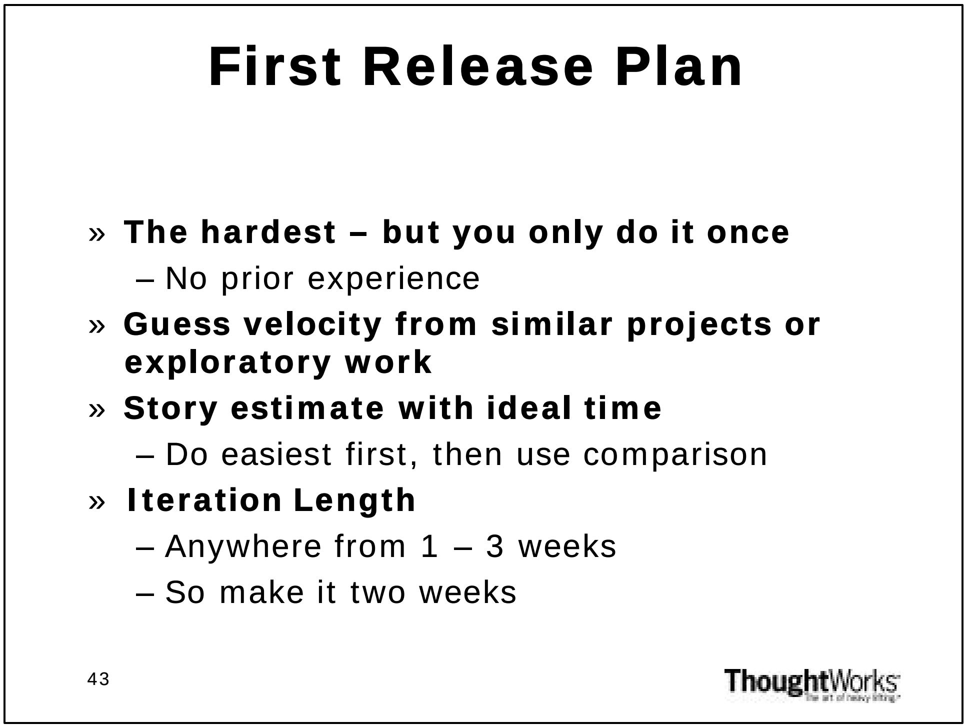
ThoughtWorks



Release Plan Events

- » **Change Priorities**
  - – Do any time
- » **Add Story**
  - – Do any time, must remove others to make room
- » **Rebuild**
  - – Every three or four iterations, or if velocity changes
  - – Re-estimate all stories and re-allocate

ThoughtWorks



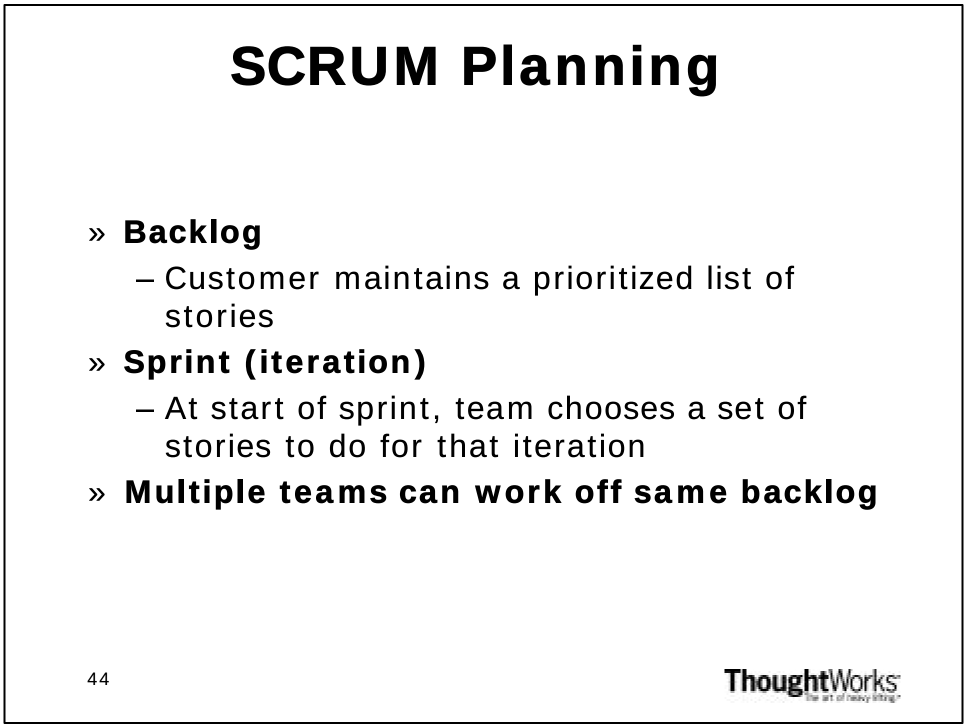
First Release Plan

» **The hardest – but you only do it once**
  - No prior experience

» **Guess velocity from similar projects or exploratory work**

» **Story estimate with ideal time**
  - Do easiest first, then use comparison

» **Iteration Length**
  - Anywhere from 1 – 3 weeks
  - So make it two weeks

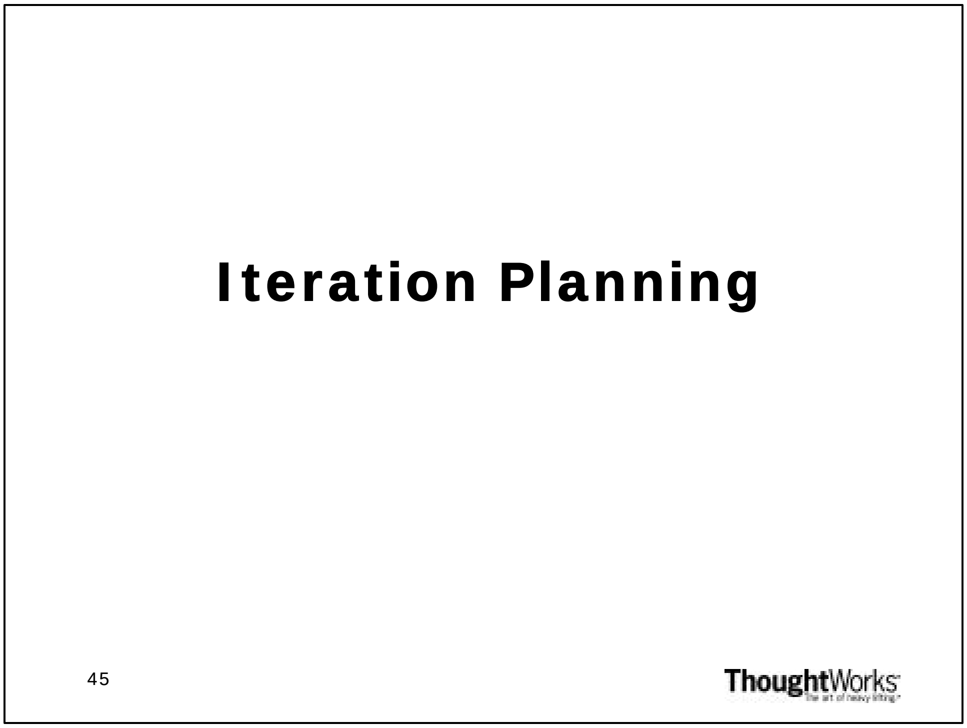


SCRUM Planning

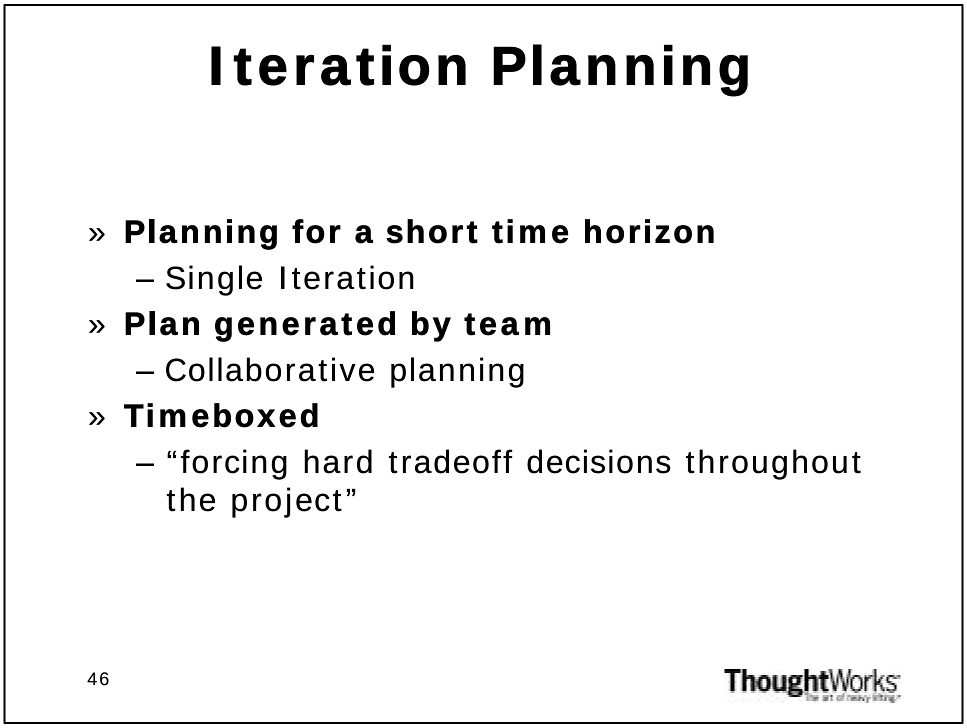
» **Backlog**
  - Customer maintains a prioritized list of stories

» **Sprint (iteration)**
  - At start of sprint, team chooses a set of stories to do for that iteration

» **Multiple teams can work off same backlog**

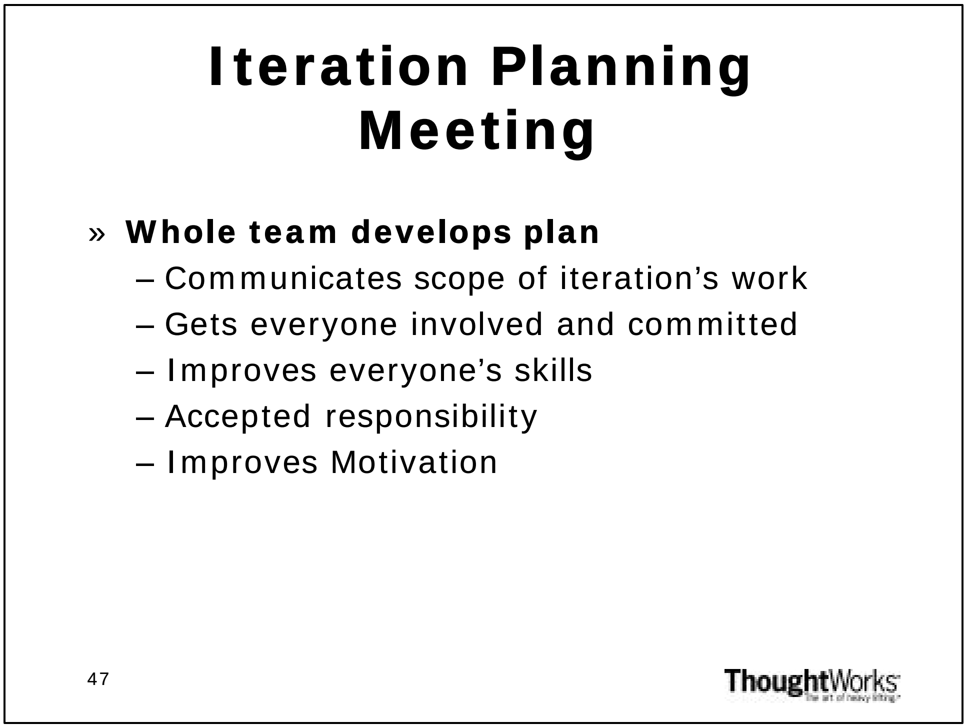


Iteration Planning



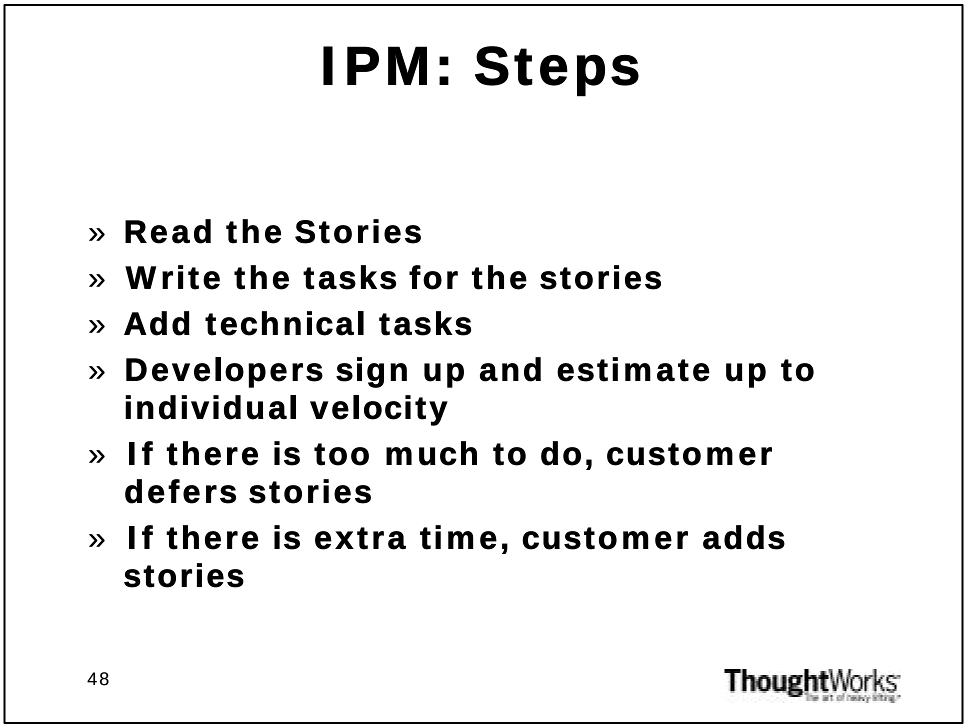
Iteration Planning

- » **Planning for a short time horizon**
  - – Single Iteration
- » **Plan generated by team**
  - – Collaborative planning
- » **Timeboxed**
  - – “forcing hard tradeoff decisions throughout the project”



Iteration Planning Meeting

- » **Whole team develops plan**
  - – Communicates scope of iteration’s work
  - – Gets everyone involved and committed
  - – Improves everyone’s skills
  - – Accepted responsibility
  - – Improves Motivation



IPM: Steps

- » **Read the Stories**
- » **Write the tasks for the stories**
- » **Add technical tasks**
- » **Developers sign up and estimate up to individual velocity**
- » **If there is too much to do, customer defers stories**
- » **If there is extra time, customer adds stories**

IPM: Reminders

- » Individuals can sign up for whatever they like
- » Tasks can be shared across stories
- » Don’t worry about dependencies
- » Task estimates may not add up to story estimates
- » Customer chooses what to defer or add
- » Individuals own tasks, useful for stories to be owned too.

Tracking

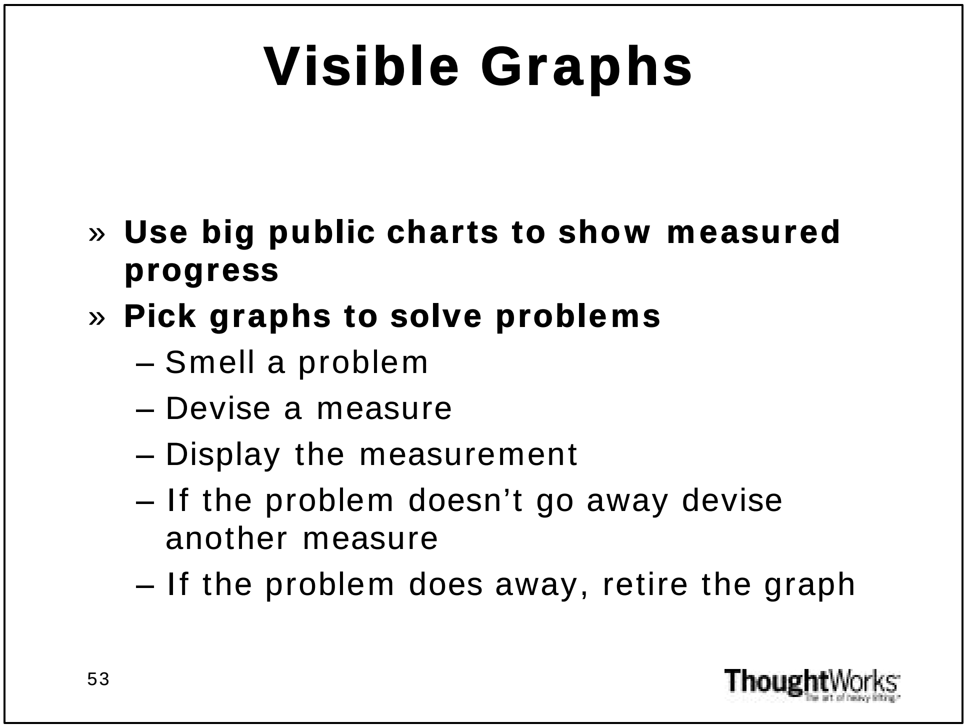
- » **Roughly twice a week ask for each task:**
  - – How many ideal days have done on it so far?
  - – How many ideal days will it take before it’s done
- » **Look for Too Much to Do**
  - – Hand off to other developer
  - – Get help
  - – Ask customer to defer

When is Done?

- » **Tasks**
  - – When the programmer says so
- » **Stories**
  - – When the customer says so
  - – Tests should run but may not be perfect
- » **Iteration**
  - – At the end of the timebox

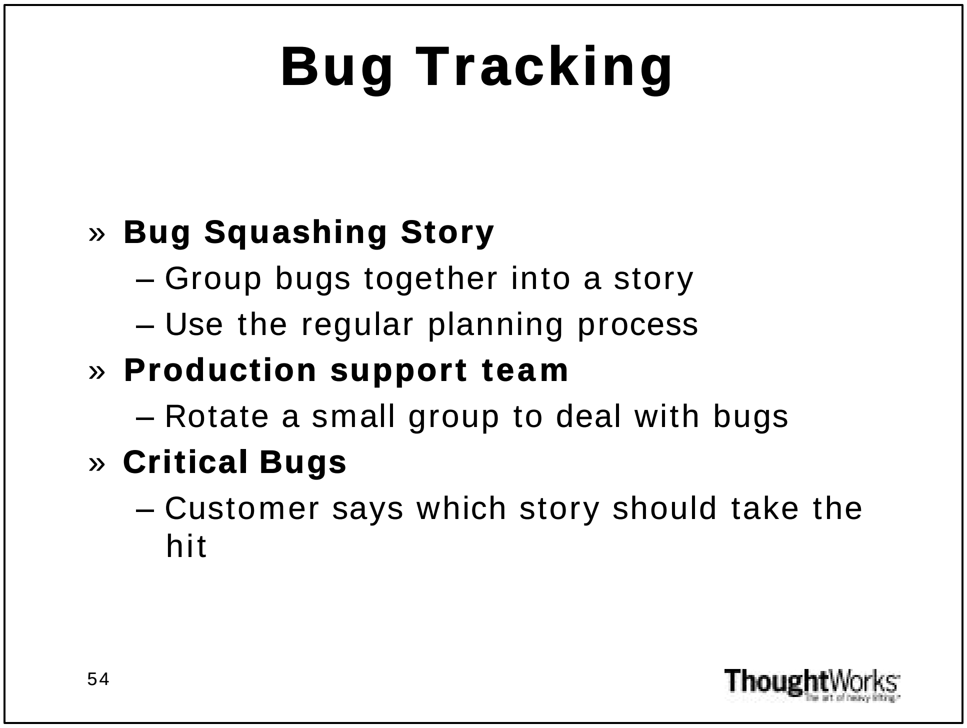
Stand Up Meetings (Scrums)

- » **Every day have a short meeting with everyone to coordinate**
- » **Everybody says**
  - – What I did since the last stand up
  - – What I intend to do in the next 24 hours
  - – What blocks are in my way
- » **Defer large issues to subgroup**



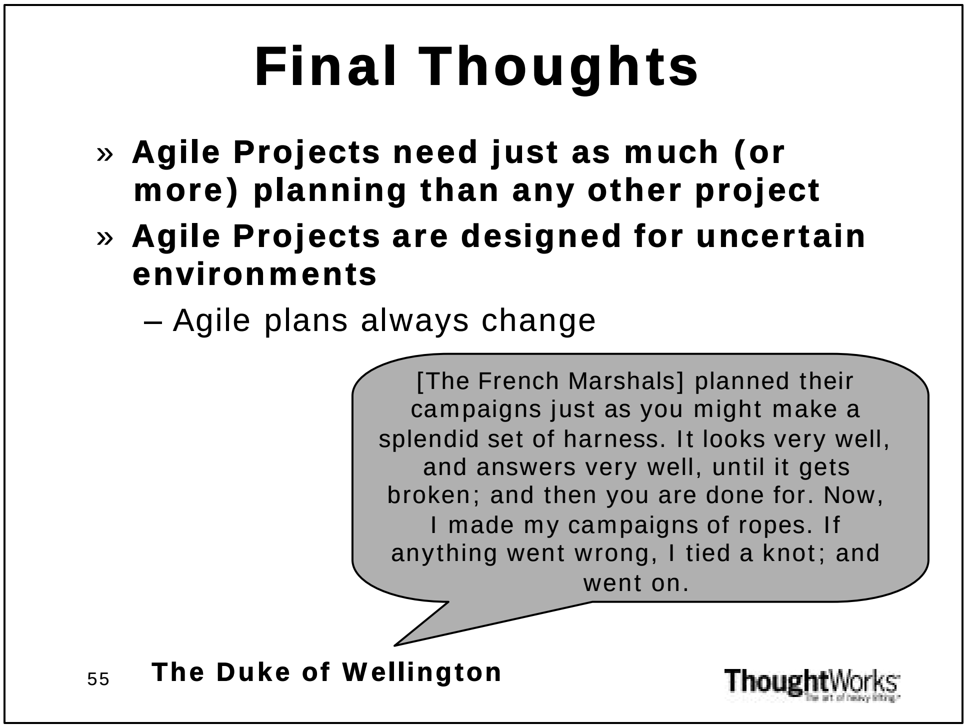
Visible Graphs

- » **Use big public charts to show measured progress**
- » **Pick graphs to solve problems**
  - – Smell a problem
  - – Devise a measure
  - – Display the measurement
  - – If the problem doesn’t go away devise another measure
  - – If the problem does away, retire the graph



Bug Tracking

- » **Bug Squashing Story**
  - – Group bugs together into a story
  - – Use the regular planning process
- » **Production support team**
  - – Rotate a small group to deal with bugs
- » **Critical Bugs**
  - – Customer says which story should take the hit



Final Thoughts

- **Agile Projects need just as much (or more) planning than any other project**
- **Agile Projects are designed for uncertain environments**
  - – Agile plans always change

> [The French Marshals] planned their campaigns just as you might make a splendid set of harness. It looks very well, and answers very well, until it gets broken; and then you are done for. Now, I made my campaigns of ropes. If anything went wrong, I tied a knot; and went on.

The Duke of Wellington