

Incremental Releases

Users and Stakeholders Will Love

How to deliver functionally complete, valuable,
incremental releases

Jeff Patton
ThoughtWorks
jpatton@thoughtworks.com

Copyright is held by the author/owner(s).
OOPSLA'06, October 22–26, 2006, Portland, Oregon, USA.
2006 ACM 06/0010.

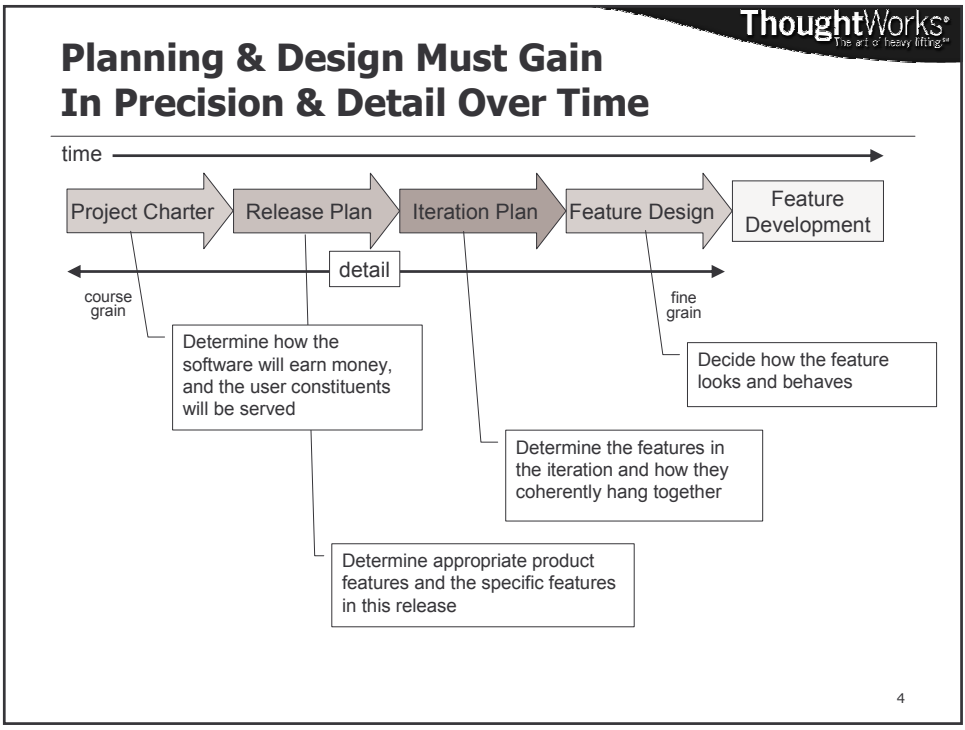
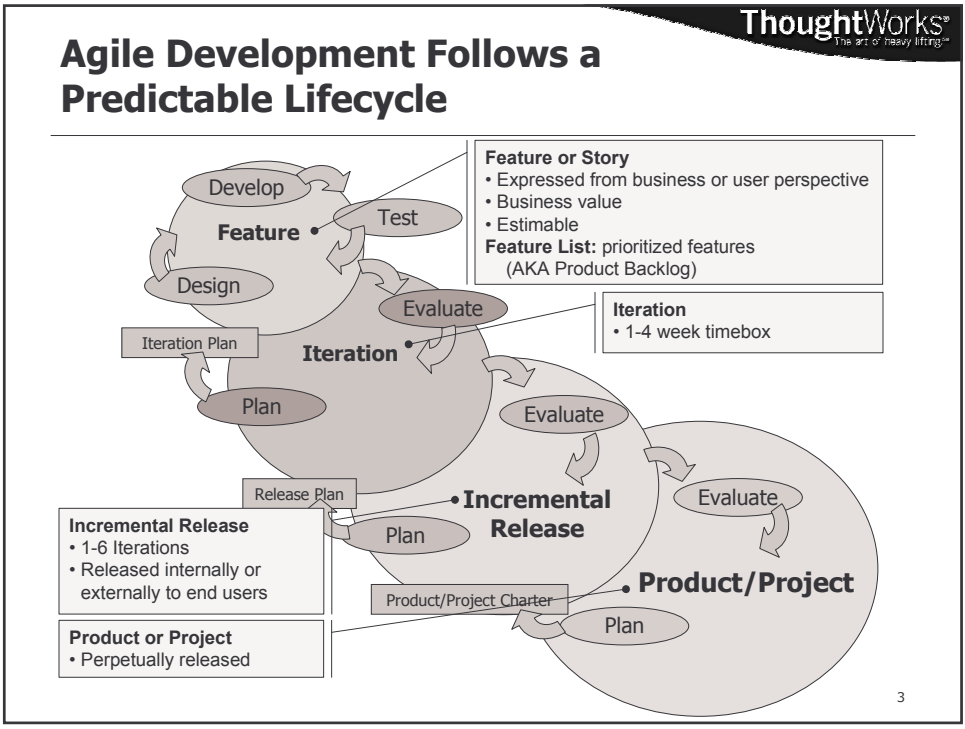
Please join a work group of 4-6 people – thanks.

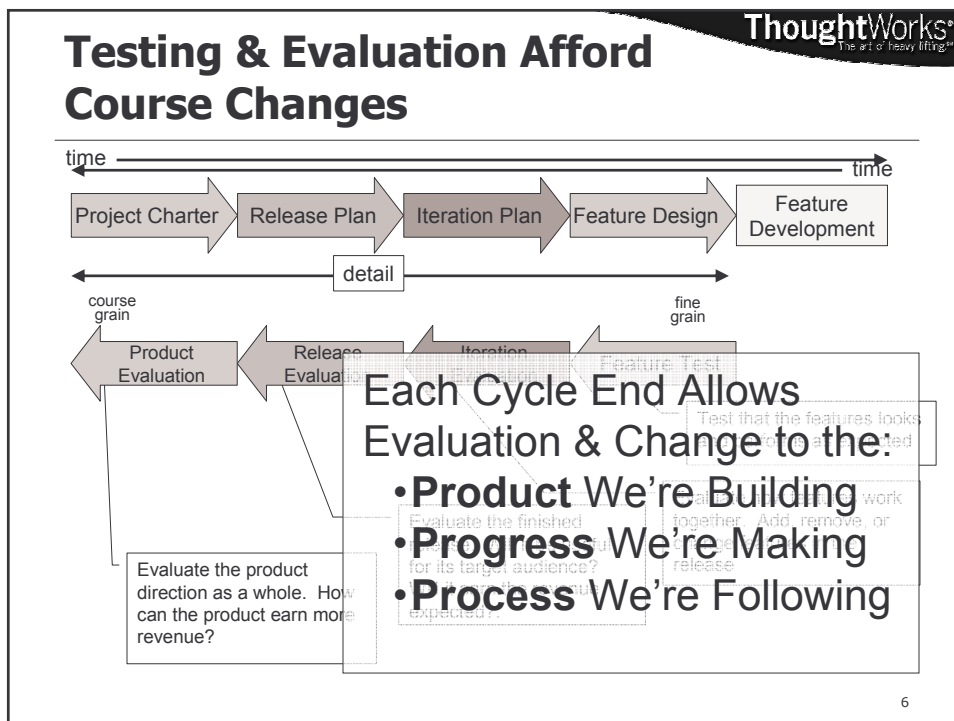
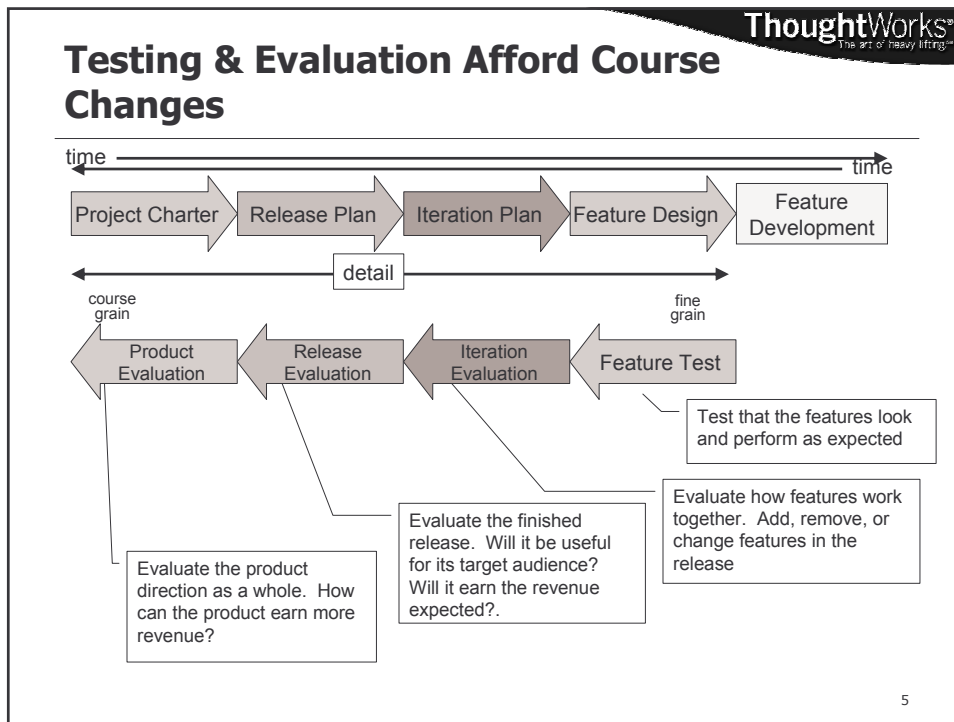
Section 1: Incremental Development, ROI, and Use

1:30 – 3:00

- The Agile iterative development incremental release lifecycle
- Return on Investment and Rate of Return from Incremental Release
- User tasks and user stories
- Modeling use with a user task model

- Goals:
 - o Understand Agile's incremental release, iterative development lifecycle
 - o Understand the financial and risk reduction benefits of incremental release
 - o Understand how the usage of the release is critical to capturing those benefits





Incremental Release Increases Return on Investment

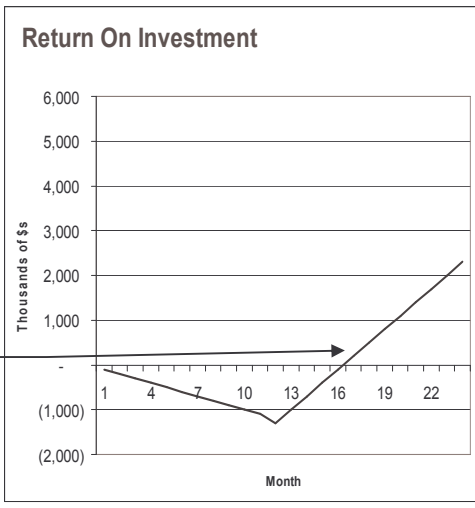
- ❑ Software begins to earn its return after delivery and while in use
- ❑ The sooner the software begins earning money:
 - the sooner it can recoup its development costs,
 - and the higher the overall rate of return
- ❑ Increasing release frequency adds costs that must be taken into account
 - additional testing costs
 - promotion costs
 - delivery costs
 - potential disruption to customers
- ❑ The impact on ROI for early release can be dramatic
- ❑ The impact on cash flow even more dramatic


Evaluating Return on 4 Release Strategies for the Same Product Features

- ❑ All features delivered and in use earn \$300K monthly
 - About half the features account for \$200K of this monthly return
- ❑ Features begin earning money 1 month after release
- ❑ Each month of development costs \$100K
- ❑ Each release costs \$100K

Single Release
12 months

total cost: \$1.3 M
 total 2 year return: \$3.6 M
 net 2 year return: **\$2.3 M**
 Cash Investment: \$1.3 M
 Internal Rate of Return: **9.1%**

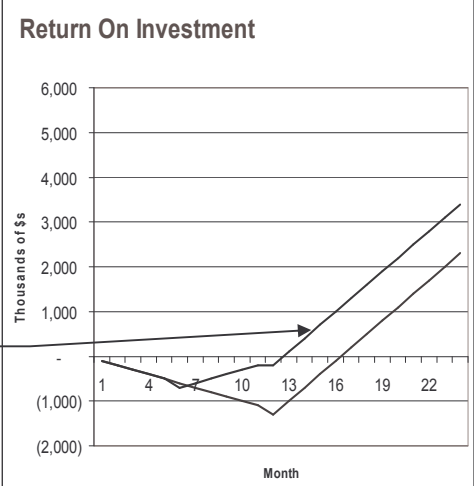





Evaluating Return on 4 Release Strategies for the Same Product Features

- ❑ All features delivered and in use earn \$300K monthly
 - About half the features account for \$200K of this monthly return
- ❑ Features begin earning money 1 month after release
- ❑ Each month of development costs \$100K
- ❑ Each release costs \$100K

Semi Annual Release
 6 months increment
 total cost: \$1.4 M
 total 2 year return: **\$4.8 M**
 net 2 year return: **\$3.4 M**
 Cash Investment: \$.7 M
 Internal Rate of Return: **15.7%**



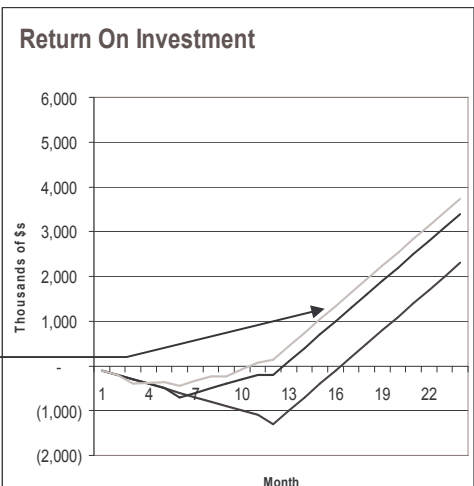
9



Evaluating Return on 4 Release Strategies for the Same Product Features

- ❑ All features delivered and in use earn \$300K monthly
 - About half the features account for \$200K of this monthly return
- ❑ Features begin earning money 1 month after release
- ❑ Each month of development costs \$100K
- ❑ Each release costs \$100K

Quarterly Release
 3 months increment
 total cost: \$1.6 M
 total 2 year return: **\$5.3 M**
 net 2 year return: **\$3.7 M**
 Cash Investment: \$.44 M
 Internal Rate of Return: **19.1%**

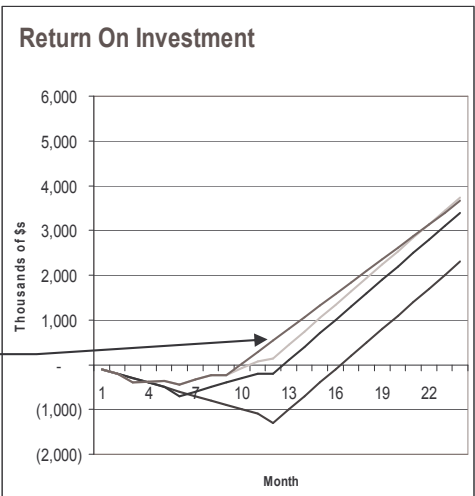


10

Evaluating Return on 4 Release Strategies for the Same Product Features

- ❑ All features delivered and in use earn \$300K monthly
 - About half the features account for \$200K of this monthly return
- ❑ Features begin earning money 1 month after release
- ❑ Each month of development costs \$100K
- ❑ Each release costs \$100K

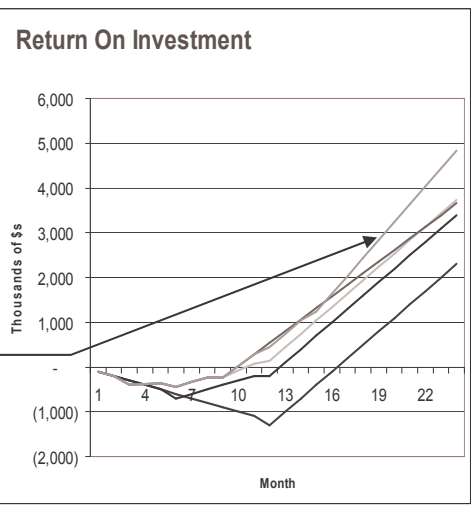
Quarterly Release – drop the last release
 3 months increment
 total cost: \$1.2 M
 total 2 year return: \$4.9 M
 net 2 year return: **\$3.7 M**
 Cash Investment: \$.44 M
 Internal Rate of Return: **20.4%**



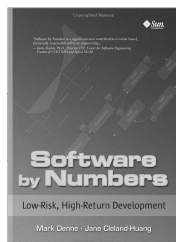
Continuing To Add Features May Not Pay The Same Level Of Return

- ❑ Continue development for one additional quarter
- ❑ Additional high value features add \$100K monthly increase to return

Quarterly Release – continue with 5th release
 3 months increment
 total cost: \$2 M
 total 2 year return: \$6.2 M
 net 2 year return: **\$4.24 M**
 Cash Investment: \$.44 M
 Internal Rate of Return: **19.0%**



Software By Numbers & Project Portfolios



- ❑ Software by Numbers [Denne & Cleland-Huang] describes Incremental Funding Methodology [IFM]
 - Goal to reduce necessary cash outlay
 - Make projects self-funding
 - Increase return on investment
- ❑ SBN Tools:
 - <http://dactyl.cti.depaul.edu/ifm/default.htm>
- ❑ SBN introduces the concept of Minimal Marketable Feature – MMF - the smallest sized feature that would have marketable value
- ❑ SBN simple financial models provide guidance on evaluating multiple projects in a portfolio

13

Building & Evaluating Complete Releases Helps Reduce Risk

- ❑ Prove general architectural approach
- ❑ Validate domain model
- ❑ Perform user acceptance testing
 - Showing users complete workflow lets them effectively evaluate and give feedback
- ❑ Test for performance
- ❑ Test for load
- ❑ Deploy in target environment

14

To Capture Return On Investment, the Delivered Product Must Be Used

- ❑ To plan an incremental release we must consider:
 - Users
 - User goals
 - User's current work practice, including current tools and processes
 - Work practice after each product release

15

Today's Business Problem *Barney's Media*

- ❑ As you review this problem
 - Think about business goals for building this software
 - Where will the organization earn money from building this software?
 - How will they measure return?



- Identify Users & Goals
 - Who will use this software in pursuit of what goal?
 - Don't forget the business people who've paid for the software – how and why might they use it?

Activity: everyone read the business problem. Take a few minutes to discuss the problem as a team. Did you learn anything from the discussion you hadn't thought about when reading the problem?

16

ThoughtWorks
The art of heavy lifting™

Software Is A Tool People Use To Help Meet Goals, Tasks are the Actions They Perform

```
graph TD; A[I have Goals] --> B[I'll reach this goal by performing some Tasks]; B --> C[I'll seek out TOOLS that help be better perform my task];
```

- ❑ Goal:
 - Reach the end of my life with my own teeth still in my head.
- ❑ Tasks:
 - Clean teeth
 - Visit a dentist
- ❑ Tools:
 - Toothbrush
 - Toothpaste
 - Running water
 - Floss
 - Dentist
- ❑ Understand goals, then tasks before identifying tools.
- ❑ Validate tools by performing tasks and confirming goals are met.
- ❑ Defer detailed *tool* design decisions by identifying and planning for task support.

17

ThoughtWorks
The art of heavy lifting™

User Interface Designers Often Use the Tasks & Activities to Describe What People Do

- ❑ Tasks have an objective that can be completed.
- ❑ Tasks decompose into smaller tasks.
- ❑ Activities are used to describe a continuous goal, one that might use many tasks, but may never really be completed.
- ❑ "Read an email message" is a task, "Managing email" is an activity.

```
graph TD; subgraph activity; direction TB; T1[task] --> T2[task]; T2 --> T3[task]; T3 --> T4[task]; T1 --> T2; T1 --> T3; T1 --> T4; end;
```

18

Tasks Have A Goal Level



Cloud or high summary level: very high level ongoing goals that may never be completely achieved but that I'll use summary level goals to drive towards.



Kite or summary level: long term goals that I'll use various functional level goals to achieve.



Sea or function level: tasks I'd reasonably expect to complete in a single sitting.

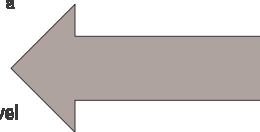


Fish or sub-function: smaller tasks that by themselves may not mean much, but stitched together allow me to reach a function level goal.



Clam or low sub-function level: small details that make up a sub function goal.

Plan releases using tasks at sea level and a bit below



19

A Good User Story Models the Use of the System

- ❑ Originally eXtreme Programming described a user story as a small amount of text written on an index card to function as a reminder for a conversation between developer and customer
- ❑ From Wikipedia:
"A **user story** is a software system requirement formulated as one or two sentences in the everyday language of the user."
- ❑ The user story form credited to Rachel Davies in Cohn's User Stories Applied combines user, task, and goal:

As a [type of **user**]
I want to [perform some **task**]
so that I can [achieve some **goal**]

As a harried shopper
I want to **locate a CD in the store**
so that I can purchase it quickly, leave, and continue with my day.

20

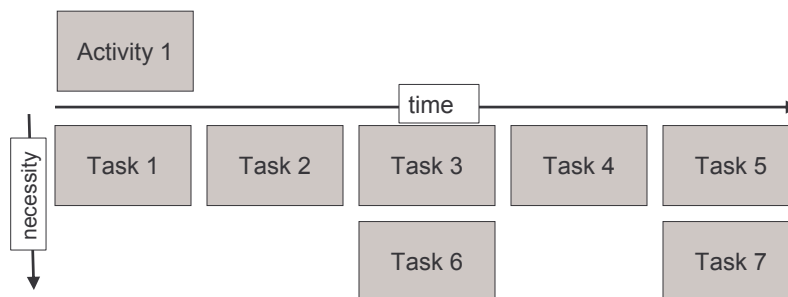
Build Release Plans Using User Tasks to Defer Feature Design

- ❑ Understand goals then user tasks before identifying tools to support them.
- ❑ Validate tools by performing tasks and confirming goals are met.
- ❑ Defer detailed *tool*/ design decisions by identifying and planning for task support.
- ❑ The idea of "latest responsible moment" comes from Lean Software Development.
 - Put off decisions as long as you can: to the latest responsible moment.
 - But it's the latest *responsible* moment, not the "*last possible*" moment. That wouldn't be responsible.

21

A Task Workflow Model Organizes Tasks to Represent Workflow

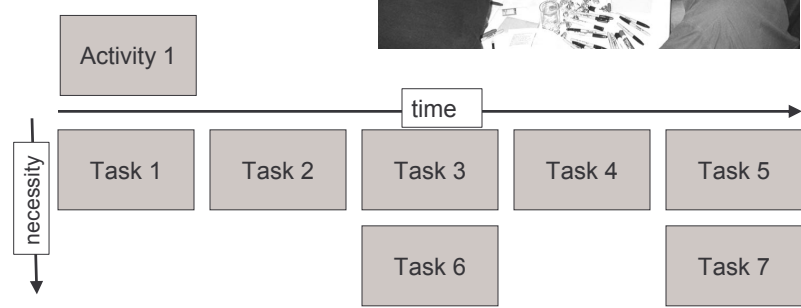
- ❑ To build a simple task workflow model:
 - Draw a left to right axis representing time, a top to bottom axis labeled necessity
 - Identify high level activities performed by users of the system and place them above the time axis in the order that seems reasonable
 - Within each activity, organize tasks in the order they're most likely completed
 - Move tasks up and down depending on how likely they are to be performed in a typical instance of use



22

Exercise: Build a Simple Task Model

Activity: using the pre-printed activity and task cards, build a simple task workflow model for Barney's



Incremental Releases Users and Stakeholders Will Love

How to deliver functionally complete valuable incremental releases

Please return from our break on time at 3:30

Jeff Patton
ThoughtWorks
jpatton@thoughtworks.com



Section 2: Thinning Feature Scale, Incremental Release Planning, Iterative Release Construction

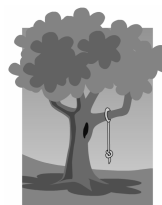
3:30 – 5:00

- Understanding tasks, features, and feature scale
- Thinning a release while retaining business value
- Leveraging the task model for incremental release planning
- Managing the incremental design and development of a product release
- Successfully using user stories in incremental development
- Differentiating between project and product success

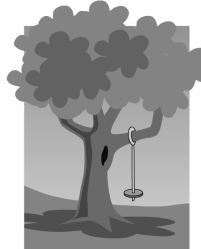
25

Considering Feature Scale

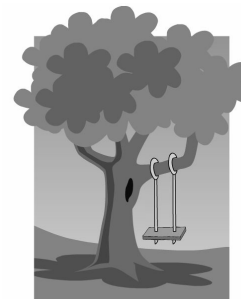
- Given a task like “swing from tree,” a variety of feature design solutions exist to support the task which vary widely in scale
- Managing scale appropriately is an important part of managing scope
- When initially planning the delivery of a set of features, the scale of each feature must be considered
- Much of detail scale management happens during design and development
 - o Close to the time the functionality is needed
 - o In the context of other features, time constraints, development capacity, and other projects in the portfolio



low cost



moderate cost



high cost

26

The Car Metaphor

- Consider the job of building a car incrementally.
- Omitting necessary features may make the product useless – this makes prioritization difficult
- Scaling all features to highest level increases cost
- To control the cost of the car, we scale the features back to economical levels

Feature List

- Engine
- Transmission
- Tires
- Suspension
- Breaks
- Steering wheel
- Driver's seat
- ...



Mercedes-Benz

27

The Characteristics of a Feature Used For Managing Scale

- Necessity:** what minimal characteristics are necessary for this feature?
 - o For our car a minimal engine and transmission are necessary – along with a number of other features.
- Flexibility:** what would make this feature more useful in more situations?
 - o For our car, optional all-wheel-drive would make it more useful for me to take on camping trips. A hatchback might make it easier for me to load bigger stuff into the back.
- Safety:** what would make this feature safer for me to use?
 - o For our car adding seat belts and making the brakes anti-locking would make the car safer.
- Comfort, Luxury, and Performance:** what would make this feature more desirable to use?
 - o I'd really like automatic climate control, the seats to be leather, and a bigger V6 engine.

28

When Planning a Software Release, Thin Software Features Using the Same Guidelines

- When planning a software release, start with tasks that users will perform
- Add in flexibility as necessary
- Add in safety as necessary
- Add in comfort, luxury, and performance as it benefits return on software investment

29

Necessity: support the tasks the users must perform to be successful

- If software doesn't support necessary tasks, it simply can't be used
- A feature or set of features that minimally support each required task meets necessity guidelines

While planning a software release, features to support some tasks may not be necessary if the user can easily use a tool they already have or some other manual process to work around the absence of the feature in your software.

30

Flexibility:

support alternative ways of completing tasks or tasks that are less frequently performed

- Adding flexibility to a system adds alternative ways of performing tasks or support for less frequently performed tasks
- Sophisticated users can leverage, and often demand more flexibility
- Complex business processes often demand more flexibility

To estimate the level of flexibility needed, look to the sophistication of the users using the software and to the complexity of the work being performed. Expert users appreciate more flexibility. Complex business processes require more flexibility.

31

Safety:

help users perform their work without errors and protect the interests of the business paying for the system

- Adding safety to a system protects the users from making mistakes with features such as data validation, or process visibility
- Safety characteristics of a feature often protect the interest of the business paying for the software by implementing business rules
- Sophisticated users can work without safety features, while novices often need them
- Complex business rules often demand more safety features

To estimate the level of safety needed consider the expertise of the users of the system and the number of rules the business would like to see enforced. Novice users may need more safety features. Complex business processes may require more safety rules..

32

Comfort, Performance, and Luxury: allow users to do their work more easily, complete their work faster, and enjoy their work more

ThoughtWorks®
The art of heavy lifting™

- ❑ Adding comfort, performance, and luxury features allows your users to:
 - complete their work more easily
 - complete their work more quickly
 - enjoy their work more
- ❑ Often the return on software investment can be increased by adding these types of users
- ❑ Comfort features benefit frequent, long term use of the software
- ❑ Sophisticated users can benefit from performance features
- ❑ Those making buying decisions often look at luxury features

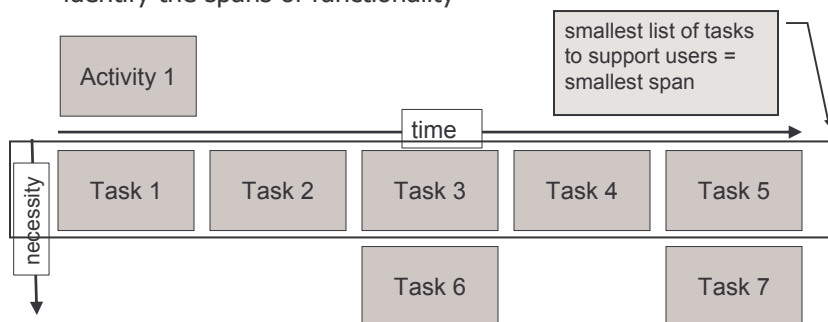
To estimate the amount of comfort, performance, and luxury necessary consider the affects of these features on the sales, adoption, and use of the software.
Look more closely at the financial drivers when estimating.
Opportunities for increasing return on investment drive additions to comfort, performance, and luxury features.

33

Using Our Task Model to Identify Features that Span Our Business Process

ThoughtWorks®
The art of heavy lifting™

- ❑ The Task Model we've built identifies the major activities and tasks that span the business functionality
- ❑ A successful software release must support all necessary activities in the business process
- ❑ This type of task model is referred to as a Span Plan since it helps identify the spans of functionality



34

ThoughtWorks
The art of heavy lifting™

Identify Releases In a Span Plan By Slicing Horizontally

- Choose coherent groups of features that consider the span of business functionalities and user activities.
- Support all necessary activities with the first release
- Improve activity support with subsequent releases

35

ThoughtWorks
The art of heavy lifting™

Sliced Span Plans

- Slices often take irregular shapes to ensure coherent groups of product features

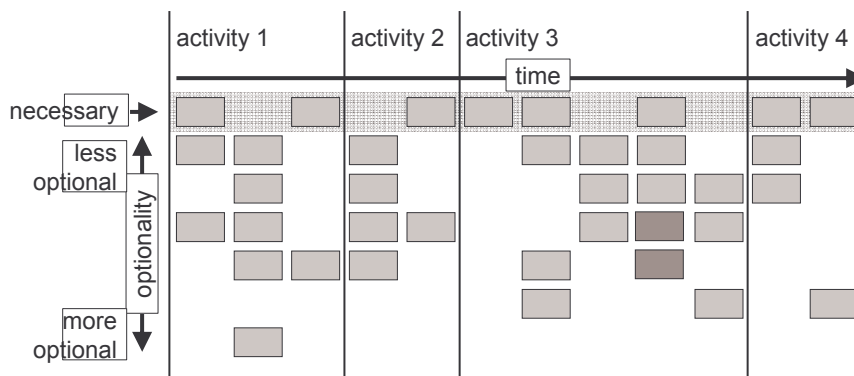
36

Use Feature Thinning Guidelines to Reduce the Size of a Release

- ❑ The topmost row of the span could be the first, smallest release
- ❑ By minimizing a release we can realize financial and risk reduction benefits earlier
- ❑ The top span represents the minimal tasks users need to accomplish to reach their goals. How can we split these “high level stories” into smallest parts?
 - Can the feature(s) to support a task have reduced safety?
 - Can the feature(s) to reduce a task have less comfort, performance, and luxury?
 - Are there optional tasks that can be supported in a subsequent release?
 - For necessary tasks, look at the steps – or subtasks that make up the task. Can any of those steps be made optional?
 - Move cards around the model, or split cards into multiple cards to defer task support, or specific feature characteristics till later releases

37

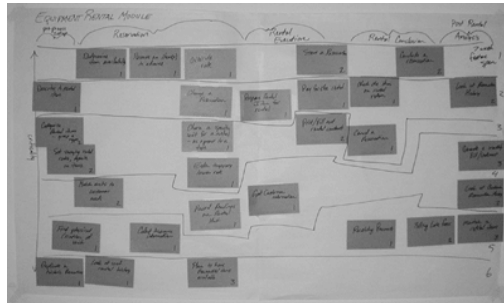
Splitting Span Plan Tasks



- ❑ Consider tasks more optional
- ❑ Split tasks into optional parts

38

Use Span Planning & Feature Thinning Guidelines to Yield Small Coherent Releases



Activity:

- Thin your span plan using feature thinning guidelines
 - Identify 3 candidate releases for Barney's
 - As a group discuss what sorts of features might support each task, and if and how they could be thinned
-
- Thin support for tasks using the following guidelines:
 - o Necessity: is supporting this task necessary in this release?
 - o Flexibility: does supporting this task add flexible alternative ways of doing things?
 - o Safety: does supporting this feature add safety for the user or business paying for the software?
 - o Comfort, Performance, and Luxury: does supporting these tasks make the software easier to use, faster to use, more enjoyable to use?

39

Building Features Incrementally Takes On Unnecessary Risk

- The software can't be validated fully until all or almost all features are built
- Errors in estimation put features built at the end of the release design and development cycle at risk
- Incrementally adding features often results in hard tradeoffs near the end of development time
 - o Disproportionately reducing the scale of features left to build
 - o Removing features from scope

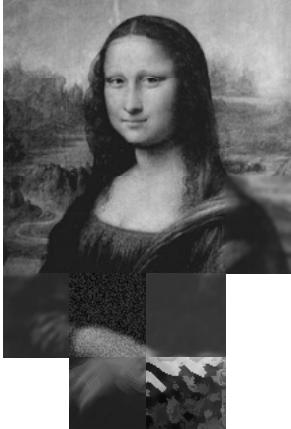


40

ThoughtWorks®
The art of heavy lifting™

Building Features Incrementally Takes On Unnecessary Risk

- ❑ The software can't be validated fully until all or almost all features are built
- ❑ Errors in estimation put features built at the end of the release design and development cycle at risk
- ❑ Incrementally adding features often results in hard tradeoffs near the end of development time
 - Disproportionately reducing the scale of features left to build
 - Removing features from scope

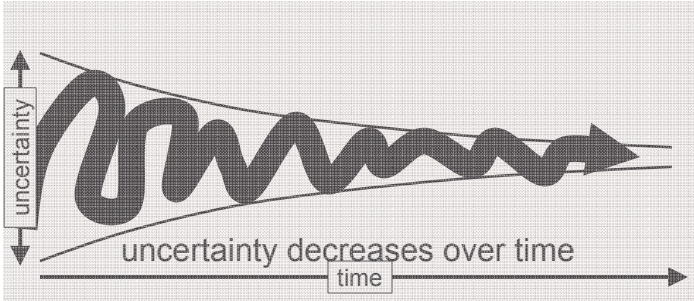


41

ThoughtWorks®
The art of heavy lifting™

The Closer We Get To Implementing Features, The More We Know

- ❑ Barry Boehm first described the cone of uncertainty applying it to estimation accuracy over time
- ❑ Steve McConnell applied the same principle to certainty of product requirements
- ❑ Defer specific feature decisions to the "latest responsible moment" as described in Poppendiek & Poppendiek's Lean Software Development

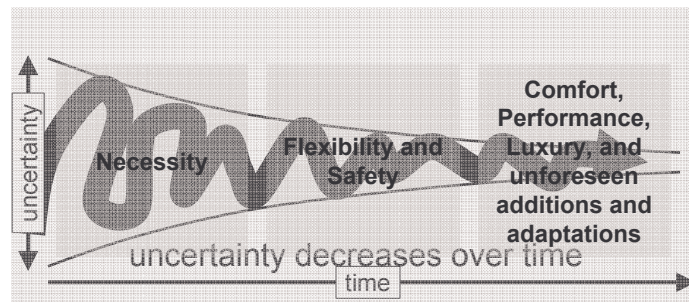


cone of uncertainty source: Barry Boehm (estimation), elaborated by Steve McConnell (requirements)

42

Divide Release Design & Development Into "Trimesters"

- ❑ Build a simple system span of necessary features first
- ❑ Add flexibility and safety next
- ❑ Finish with comfort, performance, and luxury
- ❑ Reserve time in the remaining third for unforeseen additions and adaptations



cone of uncertainty source: Barry Boehm (estimation), elaborated by Steve McConnell (requirements)

43

This Product Thickening Strategy Slowly Brings The Product Into Focus

- ❑ Just as an artist envisions an entire painting by starting with a sketch or an under-painting and slowly building up detail, apply the same strategy to "thicken" the product from simple necessities through to full featured product.



44

Agile Development's User Stories Are as Big As You Want Them To Be

- ❑ Extreme Programming's original story definition asked that stories be completed by a team in 1-3 weeks
- ❑ Scrum Story sizes may vary depending on the timeframe and context
 - Project Backlog is not the same as Sprint backlog – sprint backlog items are smaller tasks
 - Project Backlog items are split into smaller sizes as they move toward the top of the backlog
- ❑ Current Extreme Programming and Agile thinking continues to push user stories to be smaller
 - 1-3 days for a single developer to implement
- ❑ Allow your stories to have a life and lifecycle of their own before eventually arriving into an iteration for development

45

Plan With Task-Centric User Stories, Iteratively Build With Thinned Feature-Centric User Stories

- ❑ Consider the concept of high level stories that may never be done – but rather just done enough to meet the goals of *this* release
 - Iteration level feature-centric user stories for the same user task may reoccur in multiple releases
- ❑ When user tasks form the foundation of user stories, the quality of user experience for that task can always be improved
- ❑ Take advantage of iteration level validation and planning periods to:
 - Evaluate the product as it's taking shape
 - Write iteration level feature-centric user stories
 - Use a task-centric span plan to manage the product features developed as they relate to activities and tasks supported by the product release

46

Project Success is Not Product Success

- Placing focus on finishing all scope on time and in budget may equate to *project* success
- Finishing all intended scope on time, and under budget does not guarantee product use or quality of use

- Focus on **effective support of user tasks**
- Focus on achieving **stakeholder and user goals**
- Focus on **getting product into use** to begin earning revenue for its business stakeholders

47

Incremental Releases Users and Stakeholders Will Love

How to deliver functionally complete valuable incremental releases

Jeff Patton
ThoughtWorks
jpatton@thoughtworks.com

Thanks.