

# Foundations of Agile Development

## Some Questions:

What are the Foundations of Agile Development?  
What benefits does Agile Development bring?  
So how to become Agile?

## Reality Check:

Agile Development is not a Silver Bullet

# Overview of Topics

- Typical Development
  - And some of its problems
- Agile Development
  - Values and Principles
  - Users and Stories
  - Time-Boxing, Relative Sizing and Progress
  - Planning and Adaptability
  - User Stories
  - Use Cases
- Benefits of Agile Development
- Brief Prescription for Agile Development
- Supporting Tools

# Typical Development

- Inputs to development are usually
  - Product Requirements Documents from marketing and business development (new development)
  - Ticket Issues from customers (continuing development)
- Product Requirements Documents
  - full product specification generated up front
  - used by development to create a Software Requirements Document
- Transform Requirements Documents into a Development Plan
  - develop a Gantt chart and Spreadsheets to provide an estimate of the work involved, typically covering a full release
  - progress measured by Task Completion per Resource
- Ticket Issues generally become deviation points for the plan



# Problems with Typical Development

- Here are some, but there are more...
  - Progress is measured by Task Completion
    - Tasks are hard to trace to business value
    - Emphasis is on following a plan
      - The Plan is always out-of-date - changing it is heavyweight
    - Very difficult to assess where we are NOW
      - Often we can tell if we are late, but it is almost impossible to determine how late. Our estimate of how late we are only gets better as we get closer to the Plan deadline. Often too late.
    - Difficult to communicate progress to Stakeholders
  - Incremental delivery of Product is hard as Tasks are not aligned with User Value
  - Incoming Issues deflect and interrupt the Plan – poor integration
  - Emphasis on where we would like to be: not where we are, or where we could be. Basically “wishful planning”

# What is Agile Development?

- Agile Development is development that supports the Agile Value System
- The Agile Value System is based on a set of Agile Principles
- In essence Agile Development seeks to cope Deterministically with Change
  - Progress is measured against business value delivered to the customer
  - Change is welcomed as a sign of better understanding of the business need
  - Effort is only spent as needed to deliver business value - minimising effort wastage in the face of change
  - Past observed progress is used to predict future progress
- Emphasis is on delivering Business Value in a sustainable way

# Agile Value System

## Left valued more than Right

- Individuals and interactions
  - over processes and tools
- Working software
  - over comprehensive documentation
- Customer Collaboration
  - over contract negotiation
- Responding to change
  - over following a plan



# Agile Principles

- The Value System is based on the following principles:
  - Early and continuous delivery of valuable software
  - Welcome changing requirements
  - Deliver working software frequently
  - Business people and developers work together
  - Trust motivated individuals
  - Working software is the primary measure of progress
  - Promote sustainable development
  - Technical excellence and good design
  - Simplicity is essential
  - Self-organising teams
  - Team reflection and adjustment

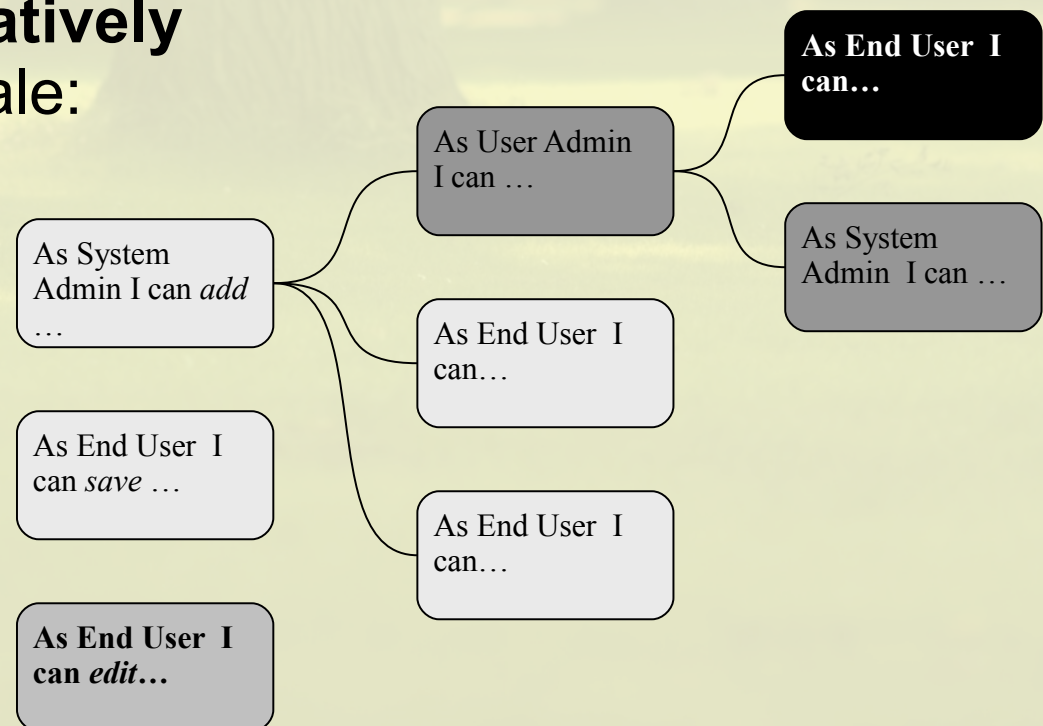
# Who is the Customer?

- Basically anyone who is a stakeholder in the project who will use the project
  - The development team itself can be a Product Support customer
- Ideally there will be a representative cross-section of real customers
  - Identifying actual customers is not always possible
  - In some cases we must use customer proxies from business development, marketing, sales, support and QA
  - Sometimes we need several customer proxies to represent different customer classes
- Customers and Users are generally the same
  - Typically the term customer is used in the broader sense to represent stakeholders who will be charged for using the system



# In Agile Development the Product is Described as User Stories

- User Stories are akin to Features and describe the System from the **user's perspective**
- Capture **Functional** and **Non-Functional** requirements uniformly
- User Stories are **Sized Relatively** using a fixed, non-linear scale: 1, 3, 5, 8, 13, 20, 40, 100
- **Dependencies** are noted
- User Stories typically have an **End User** who consumes an interaction from the system
- Customer can understand User Stories



*Darker stories are larger*

# Identify a Finite, but evolving, set of End Users

- Careful identification of User classes is an important skill requiring stakeholder input
- A shared understanding of the who the Users of a system are is central to communicating effectively the User Stories with the Customer.
- Some typical users are
  - SystemAdmin
  - SecurityAdmin
  - EndUser
  - ProductSupport
- As understanding of a product improves User classes can be split or refined as appropriate
- Some End Users will have very specific business domain roles

# Sample Users

- Here is a table of users identified for a real system
  - You can start with EndUser then split off and add more specialised types of user, eventually relegating EndUser to a non-privileged user
  - In *some* cases the system will be a valid *user*

Role Name	Description
EndUser	typical, non-privileged user
<a href="#">NmsSystem</a>	the system itself
<a href="#">SecurityAdmin</a>	user that can assign permissions and roles to other users and track their actions
<a href="#">SourceAdmin</a>	user that can configure sources
<a href="#">SystemAdmin</a>	user that can install the product
<a href="#">UmaUser</a>	User that accesses NMS only through UMA
<a href="#">UmaAdmin</a>	User that configures our interface to UMA
<a href="#">ViewAdmin</a>	user that can create objects and views and share them with others
<a href="#">ProductSupport</a>	User that needs to support the NMS solution
<a href="#">OdbcUser</a>	User that needs access NMS data directly from the Database e.g. using Safari
<a href="#">NmsDoc</a>	This user is tasked with documenting the NMS System
<a href="#">DrillDownUser</a>	User that accesses the mainframe via Drill down from NIS



# Work in Fixed, Time-Boxed Development Cycles

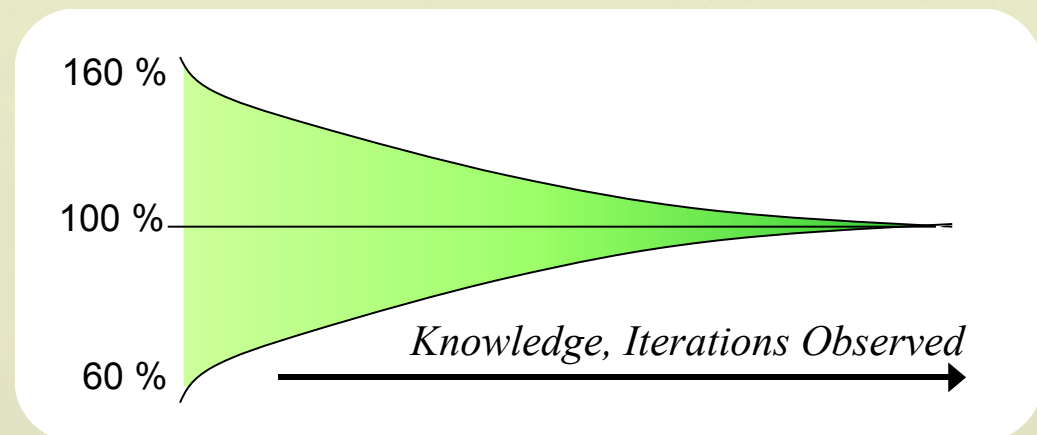
- One or more User Stories are developed in each iteration
- Frequent delivery
  - 1-4 week Iterations, 1-4 Iterations per Milestone.
  - 1-4 Milestones per GA Release
- Time-boxed development
  - Iterations last a predefined length of time
  - Team concentrates on iteration goals – the selected User Stories
  - Team observes development rate based on Story completion
  - Allows computation of effort needed for remaining Stories
- Focus mainly on the next most important thing
  - From the customer's perspective

# Plan to Provide Value

- Plan to deliver business features – the User Stories
  - Accomplishment means integration, test and documentation
  - QA can be done on a per-Story basis
- The Customer is involved in Planning and Evaluating every Iteration
- Plan at different levels – and update continuously
  - Release planning – between every iteration
  - Iteration planning – for each iteration
  - Daily Planning – daily synchronisation in a short stand-up
- Detailed planning only for the next steps
  - Typically the next iteration
- Each iteration finishes with working software

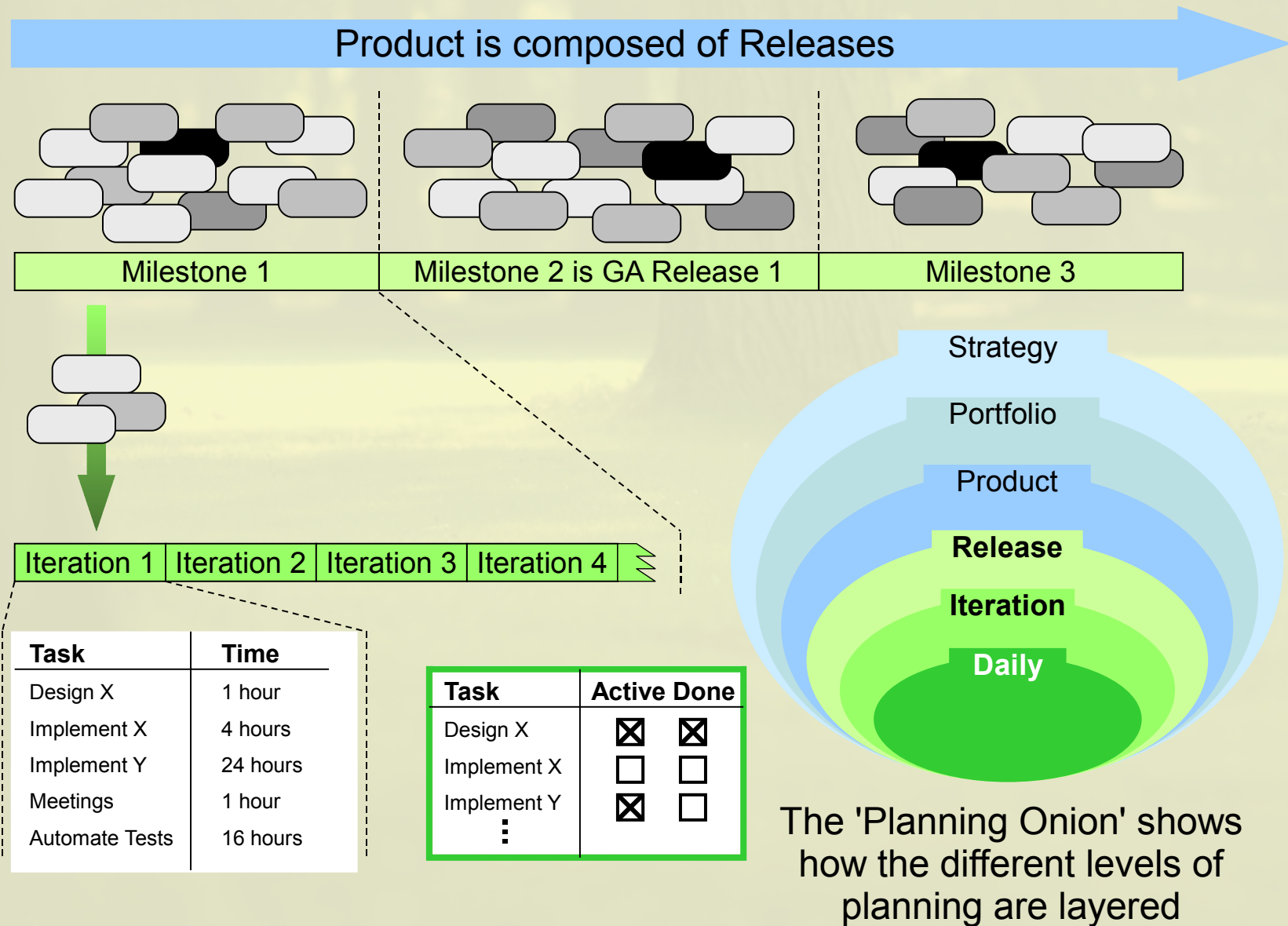
# Measuring Progress and Time

- We use the **Relative Size** of each User Story to derive estimates for duration.
- This is based on **Observed** progress during iterations.
  - How many points worth of User Stories are we likely to complete in an iteration
- The Release Plans can be continually updated.
- Different planning levels have different levels of confidence.
  - Estimates for completion of larger milestones have greater error
- Cone of uncertainty
- Estimates should be provided as a **Date Range**.





# Different Levels of Planning



# Understanding Progress

- Iterations deliver User Stories – product value increases incrementally
  - Release Progress is measured by User Stories Completed
  - Progress within an Iteration is measured by Task Completion
    - Purpose is to detect extreme deviations or critical problems
- Focus is on
  - What Still Needs to be Done, not time passed
  - Continually updating our plans to reflect what is currently the **Most Important Value** for the Customer
    - not protecting our original plan against change
- Release Progress is known accurately on each iteration
- Visibility on Progress is known NOW
- Direction of Focus can change each Iteration

# Agile Methodologies Themselves Adapt

- Each iteration we ask the customer how we are doing
- Each Iteration we evaluate how things are going
  - Iteration Retrospectives are a vital feedback loop in an Agile methodology and should never be skipped
- We can feedback useful changes into the next iteration
- Changes should be given time to evaluate their effect on the methodology
- Typically only make one change per iteration
- Try to allow each change at least 3 iterations to evaluate it fairly
  - Otherwise you risk oscillating



# Elements of an Agile Methodology

- Start
  - New Products
    - Incubate the Ideas
    - Initiate the First Set of User Stories and Milestones
  - Existing Products
    - Consolidate an Initial Set of Stories for a couple of Iterations
- Iterate in Fixed Time Boxes
  - Iteration Planning – Choose User Stories
  - Iteration Development – Complete User Stories
  - Iteration Evaluation – Update User Stories
  - Release Planning – Update Milestones
- Measure User Story Completion per Iteration to estimate future progress

# START



# ITERATE

## INCUBATION

Idea/Opportunity.  
Key Problems to Solve.  
User Needs.  
Market Analysis.  
Product and Loss Analysis.  
Business Constraints.

?? weeks

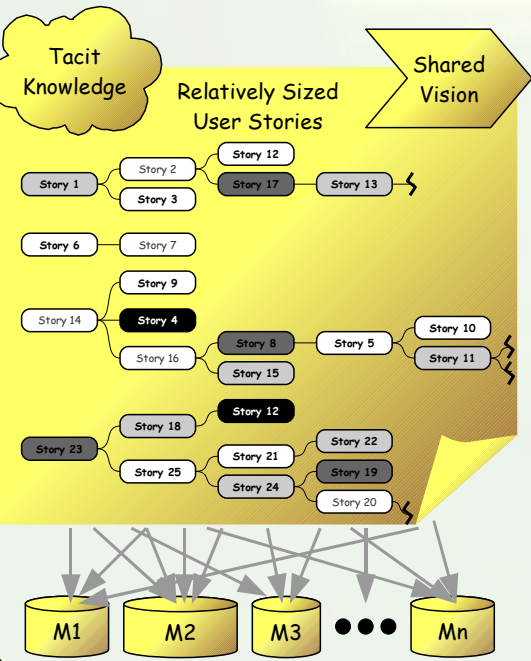
## INITIATION

Initial **User Story** list generated from Customers with Team and other Stakeholder help.  
**Identify Dependencies.**  
**Relatively Size User Stories.**

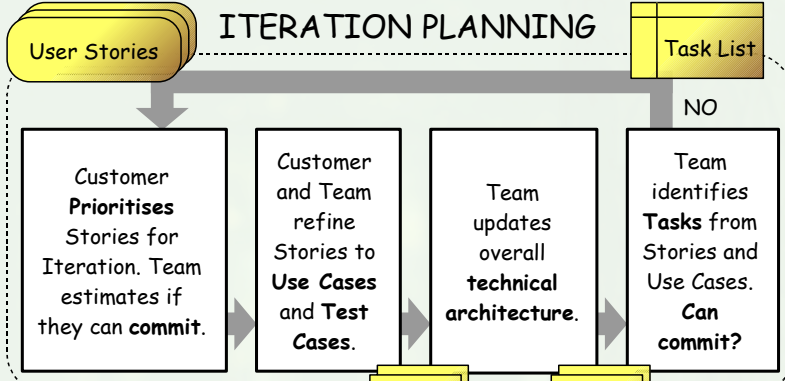
1 week

Customer, Team and other Stakeholders **scope first GA release** based on initial **User Stories** Prioritised into **Milestone Buckets.**

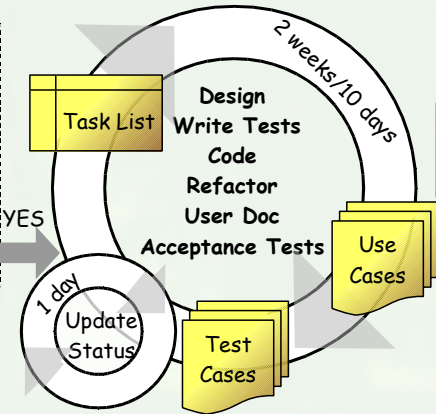
1 week



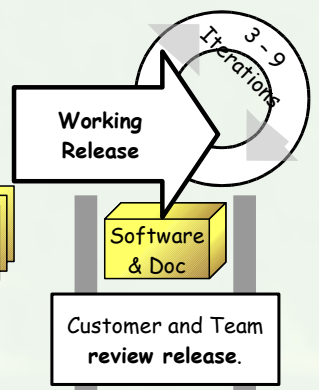
## ITERATION PLANNING



## ITERATION DEVELOPMENT



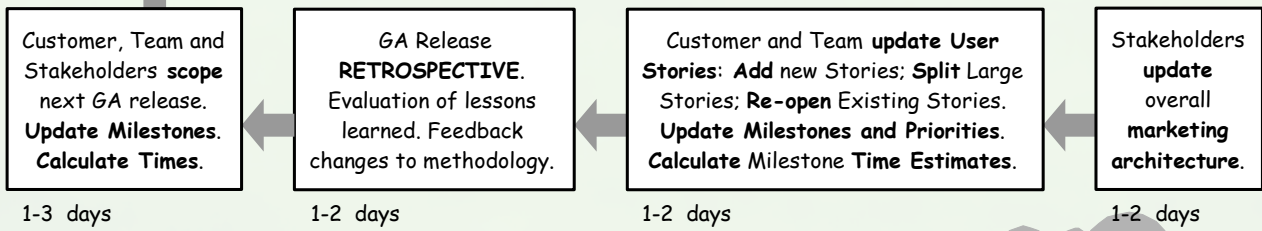
## PLANNED GA RELEASE



## ITERATION EVALUATION



## GA ITERATION EVALUATION

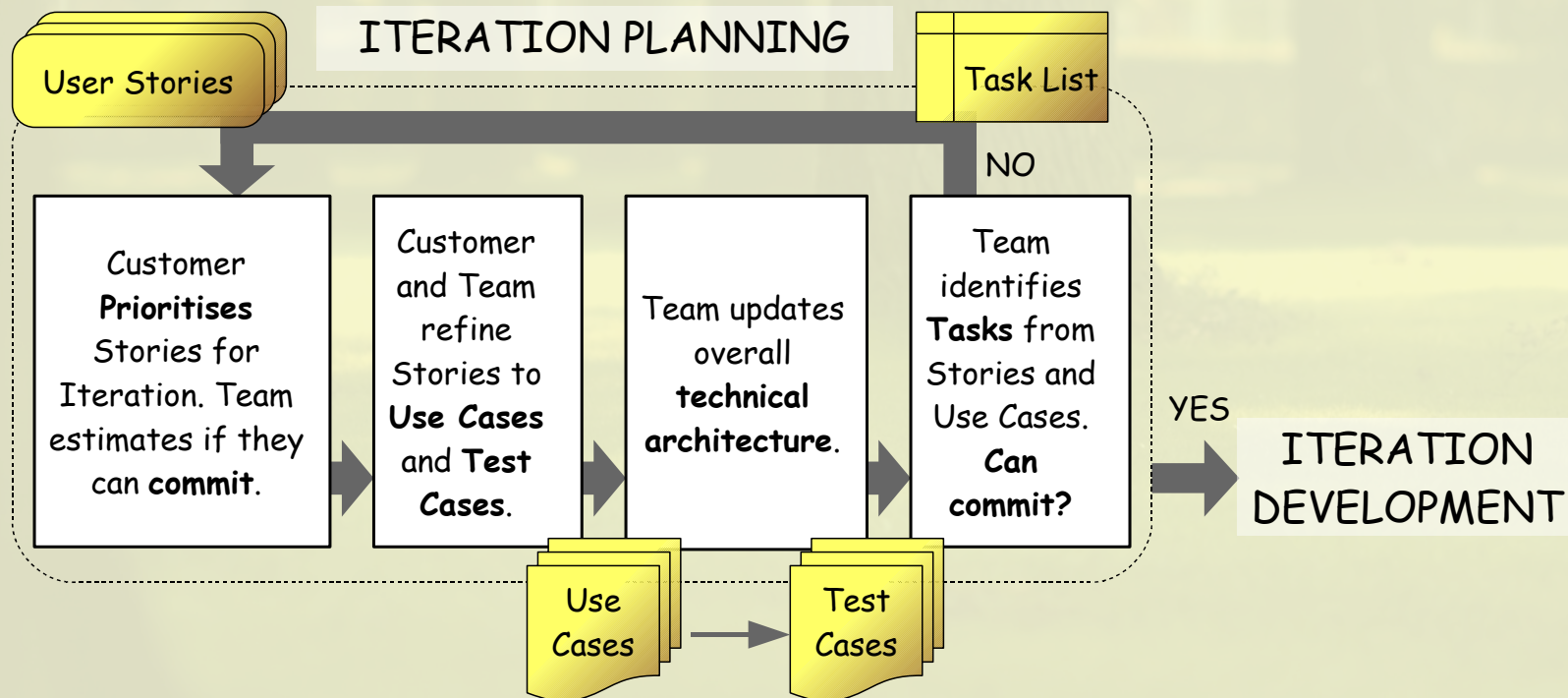


### GA Release



# Iteration Planning

- Adopt a workflow with development entry criteria to help guide the planning, for example:

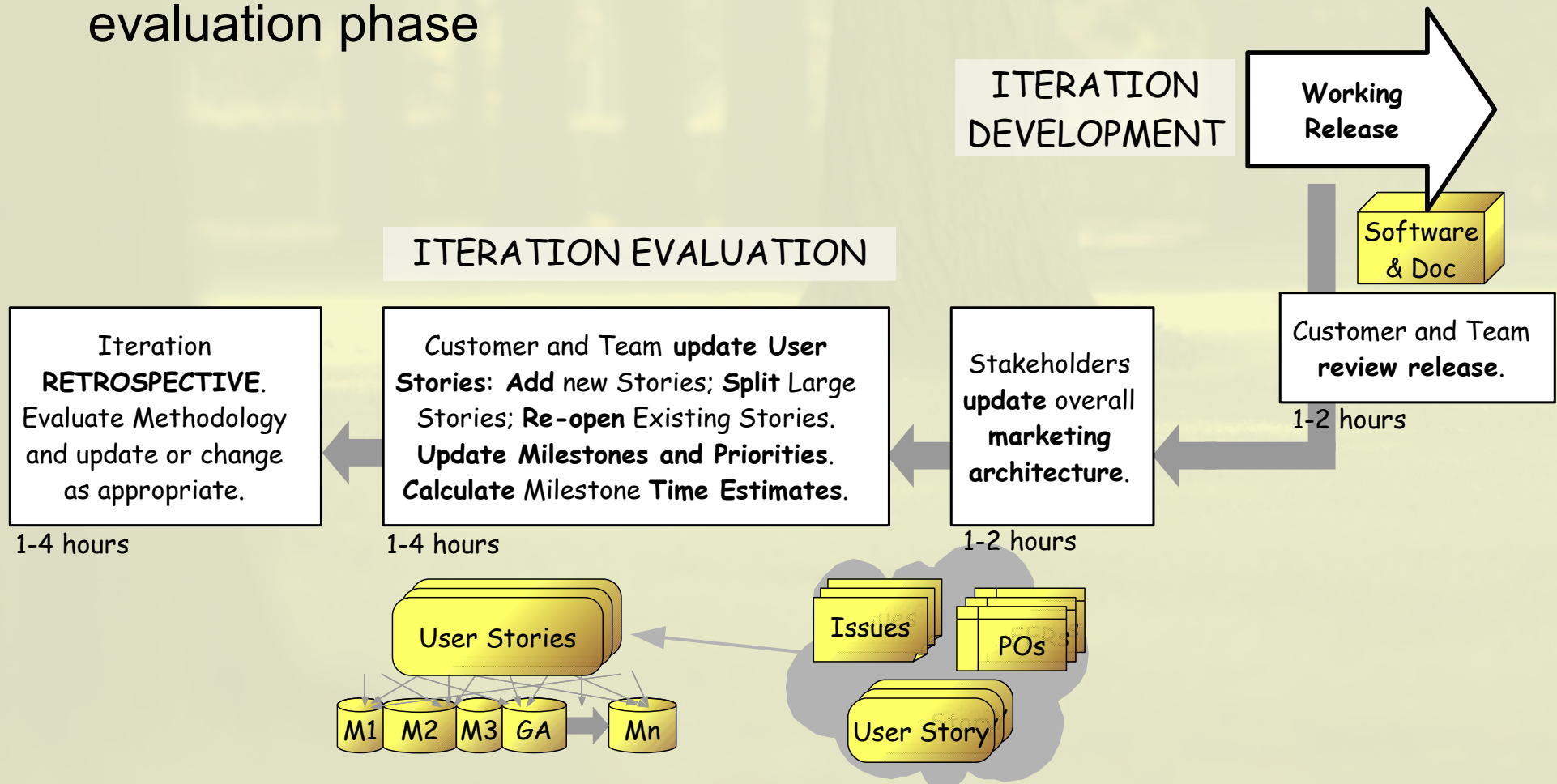


- Use evolving templates to help drive meetings and the expected outcomes
  - Wiki pages are great for this



# Iteration Evaluation

- As with iteration planning have a workflow to help guide the evaluation phase



# The Release Plan is the Roadmap

- Estimated completion date is a Calculated Date Range
- Estimates change as stories are completed, time progresses and the rate of completion changes

## NMS Milestones Roadmap

UPDATED ON Mon 20-Aug-2007: Based on 27.3 points per iteration, 3 weeks per iteration

### Milestone Summary

MileStone	Stories	Points To Do	Total Size	Estimated Completion Date	Progress
<a href="#">UmaPmoDemo</a>	9	N/A	26	COMPLETED	<b>DONE</b>
<a href="#">DexiaBankDemo</a>	13	N/A	103	COMPLETED	<b>DONE</b>
<a href="#">NmsUmaPmoMARelease</a>	40	N/A	219	COMPLETED	<b>DONE</b>
<a href="#">NisRelease1Pt1</a>	56	330	498	Mon 14-Apr-2008 -> <b>Mon 25-Aug-2008</b>	33%
<a href="#">NisRelease1Pt1ValueAdded</a>	15	100	100	Mon 4-Aug-2008 -> <b>Mon 15-Sep-2008</b>	0%
<a href="#">NisRelease1Pt1Bling</a>	9	200	200	Mon 22-Dec-2008 -> <b>Mon 23-Mar-2009</b>	0%
<a href="#">NearTerm</a>	27	103	200	Mon 23-Mar-2009 -> <b>Mon 11-May-2009</b>	48%
<a href="#">LongTerm</a>	44	343	343	Mon 23-Nov-2009 -> <b>Mon 5-Apr-2010</b>	0%

# Story Points per Iteration is based on a Moving Average

- It is possible to use more than one average to offer both long and short terms estimates of progress

Iteration	Points	Avg	Stories Completed	Retro- spective	Development Phase	
					Start	End
<a href="#">6</a>	26	26.00	<a href="#">NMSUserStory4pt1</a> <a href="#">NMSUserStory27pt6</a> <a href="#">NMSUserStory65pt8</a>	<a href="#">6</a>	2006-05-17	2006-05-31
<a href="#">7</a>	32	29.00	<a href="#">NMSUserStory4pt2</a> <a href="#">NMSUserStory65pt11</a> <a href="#">NMSUserStory65pt13</a> <a href="#">NMSUserStory65pt14</a>	<a href="#">7</a>	2006-06-07	2006-06-21
<a href="#">8</a>	24	27.33	<a href="#">NMSUserStory10pt9</a> <a href="#">NMSUserStory56pt1</a> <a href="#">NMSUserStory65pt17</a>	<a href="#">8</a>	2006-06-29	2006-07-13
<a href="#">9</a>	50	33.00	<a href="#">NMSUserStory56pt1</a> <a href="#">NMSUserStory10pt2</a> <a href="#">NMSUserStory10pt10</a> <a href="#">NMSUserStory10pt8</a> <a href="#">NMSUserStory56pt2</a>	<a href="#">9</a>	2006-07-24	2006-08-11
<a href="#">10</a>	24	31.20	<a href="#">NMSUserStory4pt11</a> <a href="#">NMSUserStory10pt11</a> <a href="#">NMSUserStory53pt2?</a> <a href="#">NMSUserStory71pt1</a>	<a href="#">10</a>	2006-09-21	2006-10-04
<a href="#">11</a>	29	30.83	<a href="#">NMSUserStory10pt3</a> <a href="#">NMSUserStory10pt12?</a> <a href="#">NMSUserStory4pt12</a> <a href="#">NMSUserStory8pt5</a> <a href="#">NMSUserStory23pt1</a>	<a href="#">11</a>	2006-10-19	2006-11-02
<a href="#">12</a>	0	26.42		<a href="#">12</a>	2006-11-10	2006-11-27
<a href="#">13</a>	13	24.75	<a href="#">NMSUserStory4pt3</a> <a href="#">NMSUserStory4pt27?</a>	<a href="#">13</a>	2006-12-11	2006-12-22
<a href="#">14</a>	11	23.22	<a href="#">NMSUserStory6pt4</a> <a href="#">NMSUserStory4pt14?</a>	<a href="#">14</a>	2007-01-09	2007-01-22
<a href="#">15</a>	39	24.8	<a href="#">NMSUserStory4pt28</a> <a href="#">NMSUserStory4pt29</a> <a href="#">NMSUserStory28pt17</a> <a href="#">NMSUserStory28pt19</a> <a href="#">NMSUserStory71pt2</a> <a href="#">NMSUserStory71pt3?</a>	<a href="#">15</a>	2007-02-06	2007-02-19
<a href="#">16</a>	39	26.1	<a href="#">NMSUserStory4pt7</a> <a href="#">NMSUserStory27pt1</a> <a href="#">NMSUserStory51pt2</a> <a href="#">NMSUserStory68pt1</a>	<a href="#">16</a>	2007-03-03	2007-03-16
<a href="#">17a</a>	34	26.75	<a href="#">NMSUserStory4pt36</a> <a href="#">NMSUserStory56pt4?</a> <a href="#">NMSUserStory73pt2</a> <a href="#">NMSUserStory73pt3?</a> <a href="#">NMSUserStory73pt4?</a>	<a href="#">17a?</a>	2007-03-23	2007-04-05
<a href="#">17b</a>	34	27.31	<a href="#">NMSUserStory4pt19</a> <a href="#">NMSUserStory22pt2?</a> <a href="#">NMSUserStory50pt3</a> <a href="#">NMSUserStory50pt8?</a> <a href="#">NMSUserStory50pt9</a> <a href="#">NMSUserStory51pt5</a> <a href="#">NMSUserStory73pt11?</a>	<a href="#">17b?</a>	2007-04-06	2007-04-19
<a href="#">18?</a>			<a href="#">NMSUserStory1nt1?</a> <a href="#">NMSUserStory1nt4?</a> <a href="#">NMSUserStory73nt6?</a>	<a href="#">18?</a>	2007-04-24	2007-05-07



# Milestones are Composed of Stories

- Not all milestones are GA Releases
- Using Named Milestones makes it easy to insert new ones

## Milestone UmaPmoDemo 26 Points COMPLETED

100%

[Toggle User Stories](#)

## Milestone DexiaBankDemo 103 Points COMPLETED

100%

[Toggle User Stories](#)

## Milestone NmsUmaPmoMARelease 219 Points COMPLETED

100%

[Toggle User Stories](#)

## Milestone NisRelease1Pt1 498 Points

33%

[Toggle User Stories](#)

- ◆ Size completed by team : 168 points
- ◆ Points Remaining: 330 pts ( ~ 12.09 iterations @ 27.3 pts/iteration )
- ◆ Estimated weeks remaining: 31 to 50 weeks.

Refined User Story	Unrefined User Story	Title	Visibility	Size	Test Complete	Need TCOM and QA?	Doc Complete	Acceptance Complete
	004	<b>SystemAdmin should be able to enjoy an AWESOME automated install on Windows 2000, XP, 2003.</b>						
<a href="#">004pt35</a>	<a href="#">004</a>	<a href="#">SystemAdmin</a> will be given the option of installing an <b>external</b> JDK if a suitable JDK is not found.	<a href="#">NisRelease1Pt1</a>	3	■	Yes	■	■
<a href="#">004pt38</a>	<a href="#">004</a>	<a href="#">SystemAdmin</a> will be able to upgrade NIS without having to uninstall.	<a href="#">NisRelease1Pt1</a>	8	■	Yes	■	■

# Story Completion provides an Immediate Indicator of Progress

## Milestone NisRelease1Pt1 498 Points

33%

### Toggle User Stories

- ◆ Size completed by team : 168 points
- ◆ Points Remaining: 330 pts ( ~ 12,09 iterations @ 27.3 pts/iteration )
- ◆ Estimated weeks remaining: 31 to 50 weeks.

Refined User Story	Unrefined User Story	Title	Visibility	Size	Test Complete	Need TCOM and QA?	Doc Complete	Acceptance Complete
	<b>010</b>	<b>NMS will evaluate exceptions on a continuous basis in the background. There should be no need to run, and attach, a client.</b>						
<a href="#">010pt4</a>	<a href="#">010</a>	<a href="#">SystemAdmin</a> can define active exception timeout (ones not killed by a TMON)	<a href="#">NisRelease1Pt1</a>	1	■	Yes	■	■
<a href="#">010pt5</a>	<a href="#">010</a>	<a href="#">SystemAdmin</a> can define in-active exception lifetime	<a href="#">NisRelease1Pt1</a>	1	■	Yes	■	■
<a href="#">010pt13</a>	<a href="#">010</a>	<a href="#">EndUser</a> will observe that the <a href="#">TmonException</a> state is updated on a timed basis regardless of whether or not new <a href="#">TmonExceptions</a> are being received from the host, that is, <a href="#">TmonException</a> states will age as expected.	<a href="#">NisRelease1Pt1</a>	5	■	Yes	■	■
	<b>026</b>	<b><a href="#">EndUser</a> must be able to drilldown from a monitored object, or a report running against a monitored object to either, another monitored object, another report or a relevant TMON.</b>						
<a href="#">026pt2?</a>	<a href="#">026</a>	<a href="#">UmaUser</a> needs drilldown data from a monitored object (i.e. APPLID) or something to define where to drilldown to. Ensure NIS can identify and persist necessary data, for each TMON, to facilitate drilldown.	<a href="#">NisRelease1Pt1</a>	13	■	No	■	■
<a href="#">026pt3?</a>	<a href="#">026</a>	<a href="#">UmaUser</a> and <a href="#">OdbcUser</a> needs access to a URL to facilitate the drilldown	<a href="#">NisRelease1Pt1</a>	8	■	No	■	■
<a href="#">026pt4</a>	<a href="#">026</a>	<a href="#">DrillDownUser</a> needs to be validated against the mainframe user database	<a href="#">NisRelease1Pt1</a>	8	■	No	■	■
<a href="#">026pt5</a>	<a href="#">026</a>	<a href="#">EndUser</a> must be able to drilldown from TMON exceptions	<a href="#">NisRelease1Pt1</a>	5	■	No	■	■
<a href="#">026pt6?</a>	<a href="#">026</a>	<a href="#">OdbcUser</a> must be able to drilldown from Time Series Data (i.e. APPLID)	<a href="#">NisRelease1Pt1</a>	13	■	No	■	■
	<b>027</b>	<b>SourceAdmin should have the ability, within the context of this product, to report on sources [aggregate instances] available to the product.</b>						
<a href="#">027pt1</a>	<a href="#">027</a>	<a href="#">SourceAdmin</a> can view an up-to-date list of sources	<a href="#">NisRelease1Pt1</a>	13	■	Yes	■	■
<a href="#">027pt9</a>	<a href="#">027</a>	<a href="#">EndUser</a> will observe that the NIS Server will update the Source state at regular intervals so that even if no Sources are received state remains current.	<a href="#">NisRelease1Pt1</a>	5	■	Yes	■	■
<a href="#">027pt10</a>	<a href="#">027</a>	<a href="#">SourceAdmin</a> will expect Source state to update dynamically in the user interface	<a href="#">NisRelease1Pt1</a>	8	■	Yes	■	■
	<b>028</b>	<b>SourceAdmin can configure which sources are collectable.</b>						
<a href="#">028pt1</a>	<a href="#">028</a>	The <a href="#">SourceAdmin</a> can mark sources (aggregate instances) collectible (i.e. making query target more specific).	<a href="#">NisRelease1Pt1</a>	13	■	Yes	■	■
<a href="#">028pt13</a>	<a href="#">028</a>	<a href="#">ProductSupport</a> will have access to an initial set of sensible default collection rules for each aggregate	<a href="#">NisRelease1Pt1</a>	13	■	No	■	■
<a href="#">028pt14</a>	<a href="#">028</a>	<a href="#">SourceAdmin</a> will have access to pre-configured default collection rules	<a href="#">NisRelease1Pt1</a>	13	■	Yes	■	■
<a href="#">028pt15</a>	<a href="#">028</a>	<a href="#">SourceAdmin</a> needs to be able to view a list of aggregates for a given tmon product instance	<a href="#">NisRelease1Pt1</a>	8	■	Yes	■	■
<a href="#">028pt16</a>	<a href="#">028</a>	<a href="#">SourceAdmin</a> can change the collectability attribute of an aggregate (which has default collectable element list)	<a href="#">NisRelease1Pt1</a>	8	■	Yes	■	■
<a href="#">028pt17</a>	<a href="#">028</a>	<a href="#">ProductSupport</a> can view the results of a TMON LFS <a href="#">TimeSeries</a> query for a specified TMON Product LFS Time Series Aggregate in the log file	<a href="#">NisRelease1Pt1</a>	8	■	No	■	■
<a href="#">028pt18</a>	<a href="#">028</a>	<a href="#">OdbcUser</a> can view the contents of a TMON LFS <a href="#">TimeSeries</a> aggregate in the database for a specified TMON Product LFS Time Series Aggregate	<a href="#">NisRelease1Pt1</a>	13	■	Yes	■	■
<a href="#">028pt19</a>	<a href="#">028</a>	<a href="#">TmonDev?</a> can validate <a href="#">TimeSeries</a> representation in the NMS Schema	<a href="#">NisRelease1Pt1</a>	13	■	No	■	■
<a href="#">028pt20</a>	<a href="#">028</a>	<a href="#">OdbcUser</a> can view the contents of any TMON LFS <a href="#">TimeSeries</a> aggregate that is found in the NMS Schema validated in <a href="#">NMSUserStory28pt19</a>	<a href="#">NisRelease1Pt1</a>	13	■	No	■	■
<a href="#">028pt21</a>	<a href="#">028</a>	<a href="#">OdbcUser</a> can expect a consistent mapping between TMON LFS <a href="#">TimeSeries</a> Element Types and NMS Database Types	<a href="#">NisRelease1Pt1</a>	13	■	No	■	■

# User Story Relative Sizing Revisited

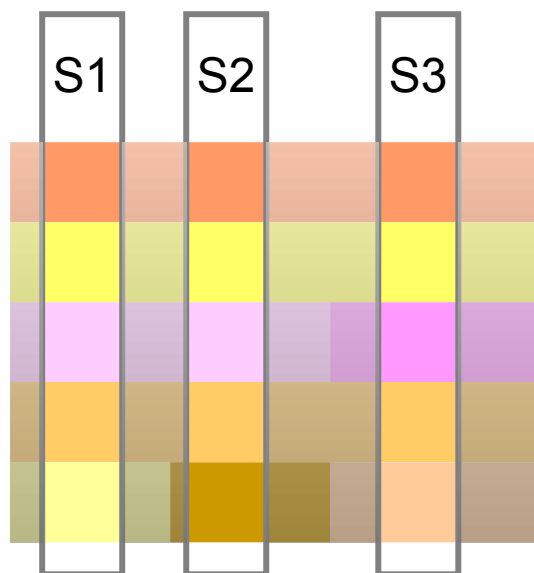
- User Stories are Sized Relatively using a fixed non-linear scale, for example, 1, 3, 5, 8, 13, 20, 40, 100
- Only User Stories below a certain relative size may be added to an iteration,
  - For a 2 week iteration we might allow, 1, 3, 5, 8, 13
  - For a 1 week iteration we might allow, 1, 3, 5
- Stories larger than this must be split into 2 or more smaller Stories.
- Splitting Stories is not so easy as it requires careful thought.
- A important side-effect is a better understanding.



# Recap: Split User Stories to fit in an Iteration

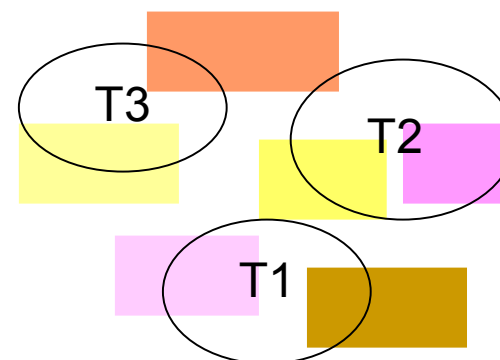
- This ensures we always focus on customer value
  - Customers get slices of the cake, not a selection of the ingredients
  - Remember User Stories grouped into Milestones is the Release Plan

Story Driven Release planning 30% done



Task Driven Release planning  
30% done

- some components missing
- little user value



# User Story Relative Sizing Revisited Again

- Sizing is Relative (that is Comparative to other Stories)
  - Relative sizing can be anchored mentally using time or simple descriptive terms
- The only important consideration is that the Relative anchoring of size remains consistent.
- In general people give consistent appraisals of Relative Size.
  - This helps protect against 'compensated' estimates
  - Let the Points per Iteration adjust the expected development rate, not some artificially modified estimate
- The fixed, finite element, sizing scale helps maintain consistency and reduces the temptation to try to estimate sizing more accurately than is possible.
  - Larger Stories are given a larger variation in size.

# User Story Relative Sizing Anchors

- Typical Relative size mental anchors that can help describe the relative size of a User Story

Relative Point Size	Description	Time (be careful with this one!)
100	Gargantuan/Elephantine	More than a quarter
40	Massive/Colossal	Up to a quarter
20	Huge	less than a month
13	Large	Less than 2 weeks
8	Medium	More than a week
5	Modest	Less than a week
3	Small	2 or 3 days of work
1	Tiny	Less than a day



# Understanding Progress of a User Story – Done or Not Done?

- Stories can be either NOT DONE
  - Not Yet Assigned for Completion
  - Candidate for Completion in this Iteration
  - Selected for Completion in this Iteration
- or DONE (Done Done, REALLY Done)
- A Story is DONE when either,
  - It is Test Complete (there is no relevant doc or use case change so a tested implementation is sufficient to be DONE), or,
  - It is Acceptance Complete (test cases all pass, documentation is available, and business value has been verified)
- Only DONE Stories contribute to Milestone progress

# User Story Completion States that identify when a Story is DONE

- A Story is either:
  - Test Complete (might mean DONE)
    - An implementation and associated tests have been developed for the story
  - Doc Complete
    - Where appropriate User Documentation has been completed to describe the Story for the User. If Doc is required Acceptance Completion is almost always needed.
  - Acceptance Complete (might mean DONE)
    - The Story is assessed as DONE or not, based on the implementation tests, User Documentation, Test Cases (derived from the Use Cases) and Story notes, especially the completion statement. This typically done by QA.

# User Story Detail Grows as it becomes a priority: Initially Not Started

- A key principle of Agile Development is to not pay for what you don't use
- When User Stories are first identified they generally consist of only the Story itself, Relative Size and Dependency notes
- As User Story priority increases we spend more effort elaborating them – lightweight in the beginning
- We are as lightweight as we can be for as long as we can be

28.18 - **OdbcUser** can view the contents of a TMON LFS **TimeSeries** aggregate in the database for a specified TMON Product LFS Time Series Aggregate.

**Relative Size = 13**

## General Notes, Discussion and Background Information

- ◆ What it means to be done
  - ◇ Get one lfs record into the repository
- ◆ Design first effort Schema for LFS [TimeSeries](#) data
- ◆ Dynamically detect presence of LFS [TimeSeries](#) Aggregate table in database
  - ◇ and Dynamically create table for LFS [TimeSeries](#) data if not exists
- ◆ Add Persistence to the LFS [TimeSeries](#) Report class (update persistence manager)
- ◆ Persist LFS [TimeSeries](#) data to the database



# User Story Info: Iteration Candidate

- Pre-Iteration Details:
  - Completion Statement
  - Clarifying Use Cases (optional)
  - Additional Context (optional)

## Iteration 20

### Pre-Iteration

#### Completion Statement

Complete when we can persist at least one row of a specific LFS Time Series aggregate to the database with some of the fields of an appropriate type (other than as strings).

#### Clarifying Use Cases?

Use Case Number	Description
NMSUseCase00?	NMSUseCase00Description?



Edit

#### Additional Context?

This story was previously completed by hard-coding some of the type conversions and building a static SQL statement to identify the elements needing to be persisted for the aggregate. The intention is to attempt this by dynamically building a SQL statement. One time series aggregate will be specified as the test aggregate and a first effort at persisting this will be made. It should be possible then to view at least one row of this aggregate in the database on completion of this story.

IMPORTANT: Being able to complete this story will push the limits of the database library we use to the extent that we may discover during this iteration that it is not possible to achieve this without writing extra code. This may be code to extend the current library, or we may need to look at using an alternative (one has already been identified, however it does not have a native Postgresql backend and that would need to be written).

# User Story Info: Current Iteration

- Iteration Development:
  - Task List (as required)
  - Notes (as required)

## Iteration Development

### Task List

Task Number	Task	Effort Estimate (hrs)
0000	Task:0000Description	00

 Edit

### Notes

The goals of the user story were to persist rows of a specific single lfs aggregate where each row comprised elements of the correct type (from the DMAP). We chose (and hard-coded) was TMONMVS.SYMAIN.040. Previously we had already shown that we could request data for this aggregate and persist it to the database, however previously we used a partially hard-coded insert statement and only persisted the elements as string types.

The work carried out this iteration has moved this forward significantly and all the technical issues with dynamically building a suitable insert statement, as well as the problems with having that statement reference data of varying types, have been addressed with appropriate solutions.

What remains then is the type mapping between the types present in SYMAIN and the types that we will use to represent those in the database.

This involves two steps:

1. To map the database type to the element based on the Input Format, Units, Size and Scale of the element (as determined from the DMAP). This first step is largely complete and a mapping function has been written.
2. This (more complex) step involves the actual conversion of the element value to the appropriate type based on the Input Format, Units, Size and Scale. Mostly this code will be lifted from NaviPlex however there are some differences that must be addressed and that is where the extra complexity lies. It is also possible that we may see elements whose descriptions are not the same as the descriptions previously seen in Real-time aggregates in NaviPlex.

We don't think that there is much more to do to make this work for the SYMAIN aggregate, around 3 days of work. Without the Cellco interruption this would likely have been complete this iteration.

Remember this user story focuses only on SYMAIN and only on the ability to persist elements to the database using a sensible type. It does not cover all possible type mappings (another story addresses this), though certainly the main type conversions will be covered; nor does it address the ability to perform continual, planned collections of LFS data using the store-clock values to help generate follow-on report requests. As such we simply use a hard-coded WHERE LMRKCLK LT XXXX clause (or a WHERE LMRKCLK GT 0).

# User Story Info: DONE

- Post-Iteration Details:
  - Completion Details (Addition TCOM and QA notes)
- Emphasis is on providing only as much information as required to allow adequate verification of completion

## Post-Iteration

### Completion Details ( Additional TCOM/QA Notes )

The goals of the user story were to persist rows of a specific single lfs aggregate (TMONMVS.SYMAIN.040) where each row comprised elements of the correct type (from the DMAP). Previously we had already shown that we could request data for this aggregate and persist it to the database, however previously we used a partially hard-coded insert statement and only persisted the elements as string types.

This iteration has moved this forward significantly and all the technical issues with dynamically building a suitable SQL INSERT statement, as well as the problems with having that statement reference data of varying types. What remains is the type mapping between the types present in SYMAIN and the types that we will use to represent those in the database, which involves two steps:

1. To map the database type to the element based on the Input Format, Units, Size and Scale of the element (as determined from the DMAP). This first step is largely complete and a mapping function has been written.
2. This (more complex) step involves the actual conversion of the element value to the appropriate type based on the Input Format, Units, Size and Scale. This code will largely be lifted from NaviPlex, however there are some differences that must be addressed (data for display vs. data for a database), and we also may see time-series elements whose descriptions were not in the realm of descriptions that NaviPlex is capable of handling.

The remaining work will need to be continued in a future iteration.



# The relationship between User Stories and Use Cases

- Use Cases compliment User Stories
  - A Use Case documents a Goal driven Functional Scenario
  - A User Story typically identifies one step in a Scenario
  - A User Story can also identify a non-functional constraint on a system from a User perspective
- In addition to the Main Success Scenario a Use Case usually identifies one or more failure scenarios
  - provides a good foundation for Test Cases
  - provides the context in which a User Story is enacted
- Use Cases and User Stories are loosely coupled
  - A Use Case can provide the context for several User Stories
  - A User Story can imply several Use Cases

# Functional and Non-Functional User Stories

- Functional User Stories typically describe something a User can do
- Non-Functional User Stories typically describe something a User will experience
  - Predictable scalability
  - Predictable stability
  - Other 'ilities
  - Often these are User-centric ways of capturing constraints on the system
  - Sometimes the System itself will be a User
- Emphasis is on which User will benefit from the non-functional requirement, or which User is most affected by the constraint

# Elements of a Use Case

- A Use Case is a textual artifact that specifically examines one or more scenarios from a User's perspective in the User's attempt to accomplish some specified goal.
- A Use Case
  - Exists in a certain Design Scope
  - Is enacted by the identified User
  - The User is trying to achieve a Goal
  - To achieve the Goal the User must complete a Main Success Scenario
  - Certain Pre-Conditions may be required before starting
  - Completion of the scenario may have Guarantees
  - There may be extension scenarios for deviations



# A Use Case Template

- The template shown is a compromise between brevity and value
- The Use Case Summary Title (USE CASE NAME) captures the essence of the Use Case
- The Main Success Scenario will have at least 2 steps but not too many (say 3 to 10)
- Extensions refer to specific steps in the Main Success Scenario
  - More than four levels deep here implies another Use Case

USE CASE ##	Design Scope Icon	USE CASE NAME	Goal Level Icon
<b>Goal in Context</b>	< a longer statement of the goal, if needed, its normal occurrence conditions >		
<b>Scope</b>	< design scope, what system is being considered black-box under design >		
<b>Level</b>	< user goal level >		
<b>Primary Actor</b>	< a role name for the primary actor, or description >		
<b>Stakeholders &amp; Interests</b>	< list of stakeholders and key interests in the use case >		
<b>Preconditions</b>	< what we expect is already the state of the world >		
<b>Minimal Guarantee</b>	< how the interests are protected under all exits >		
<b>Success Guarantees</b>	< the state of the world if goal succeeds >		
<b>Trigger</b>	< what starts the use case, may be time event >		
<b>Main Success Scenario</b>			
1	First step		
2	Second step		
3	Third step		
<b>Extensions</b>			
*a Condition on whole			
	*a1. Action 1 <a href="#">with link?</a>		
2a. Condition on step			
	2a1. Action on step		
	2a1a Condition on action		
	Last action depth - deeper than this needs it's own use case		
2b. Second condition on same step			
	2b1. Action		
	2b2. Second action		
<b>Technology &amp; Data Variations</b>			
1. The specified item can be:			
	* item 1		



# Choose a Use Case Template and use it consistently

- Scope and Level are important so are also represented with icons for quick browsing
- The Primary Actor will be one of our previously identified users, bridging the gap between User Stories and Use Cases
- Technology and Data Variations are used to detail 'how' something is different
  - Extensions capture 'what' is different from the Main Success Scenario

USE CASE XXX	Design Scope	Use Case Summary Title	Goal Level
Goal in Context			
Scope			
Level			
Primary Actor			
Stakeholders & Interests			
Preconditions			
Minimal Guarantee			
Success Guarantees			
Trigger			
Main Success Scenario			
1			
n			
Extensions			
1a.			
1a1.			
Na.			
Technology & Data Variations			
1			
n			
Comments			

# Use Cases have a Design Scope

- Identifying a design scope helps remove ambiguity
  - Business Use Cases for the Organisation/Enterprise
    - We are discussing the behaviour of the entire organization or enterprise in delivering the goal of the primary actor
      - Black box if we are treating the organisation as a black box
      - White box if we are discussing departments
  - System
    - We are discussing the behaviour of the system we are building
      - Black box if we treat the system as a black box
      - White box if we reveal details of the internals
  - Sub-system
    - We are discussing a sub-system or component of the system we are building

Design Scope	
Icon	Meaning
	Organization (black-box)
	Organization (white-box)
	System (black-box)
	System (white-box)
	Component



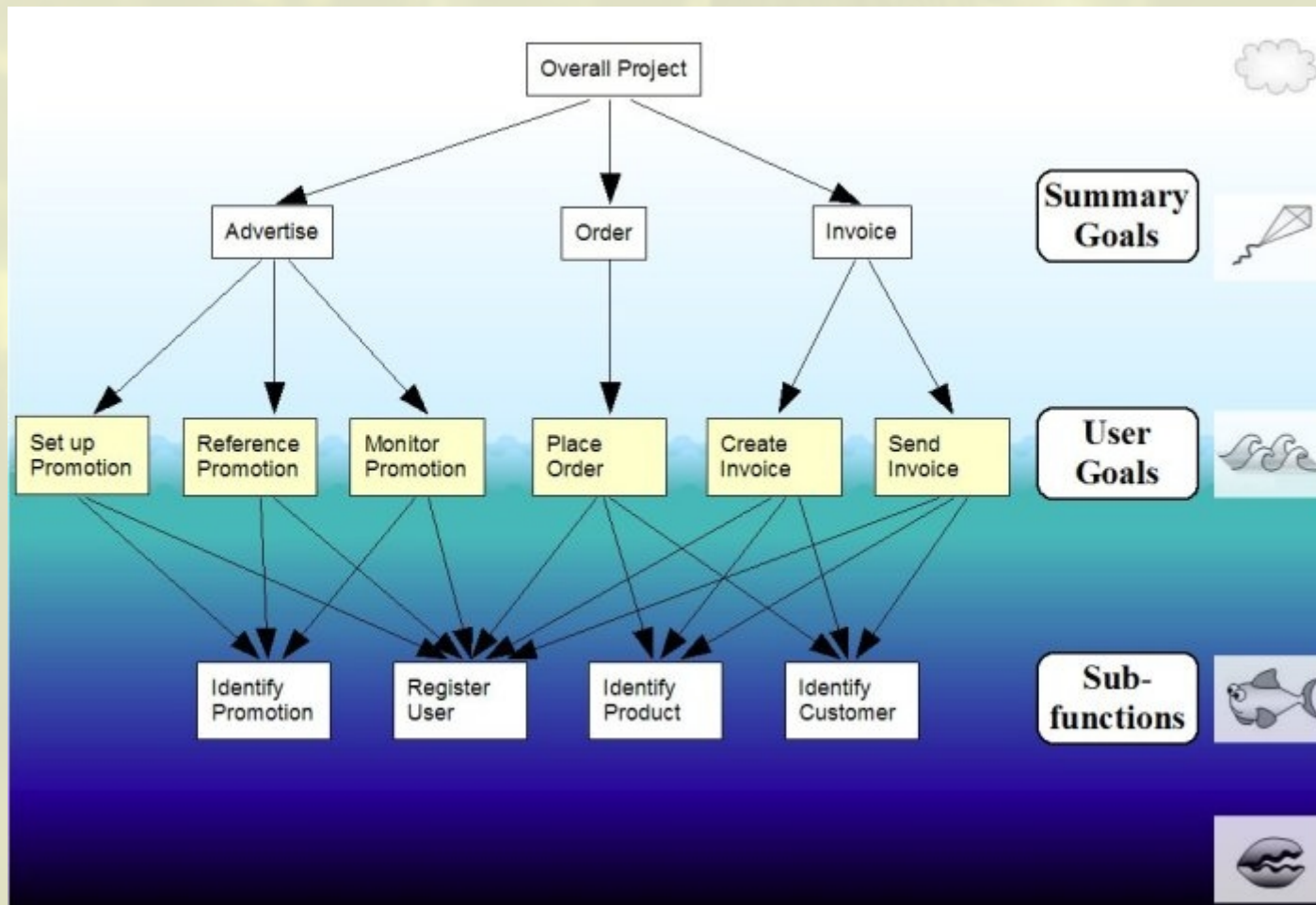
# Use Cases have a User Goal Level

- Every Use Case has a User Goal Level
  - We identify 5 goal levels for Use Cases
- Higher level goals can be unfolded or split into one or more lower level goals
  - Higher level goals provide context for lower level ones
- User Level Goals are the most important and most energy should be spent trying to detect these use cases
  - Subfunction Goals should only be added if they are needed to remove ambiguity
- The Goal Level icons use a Sky-SeaLevel-Sea metaphor to illustrate that most use cases will appear at sea level. There is only one sea level.

Goal Level	
Icon	Meaning
	Very high summary
	Summary
	User Goal
	Subfunction
	Too Low

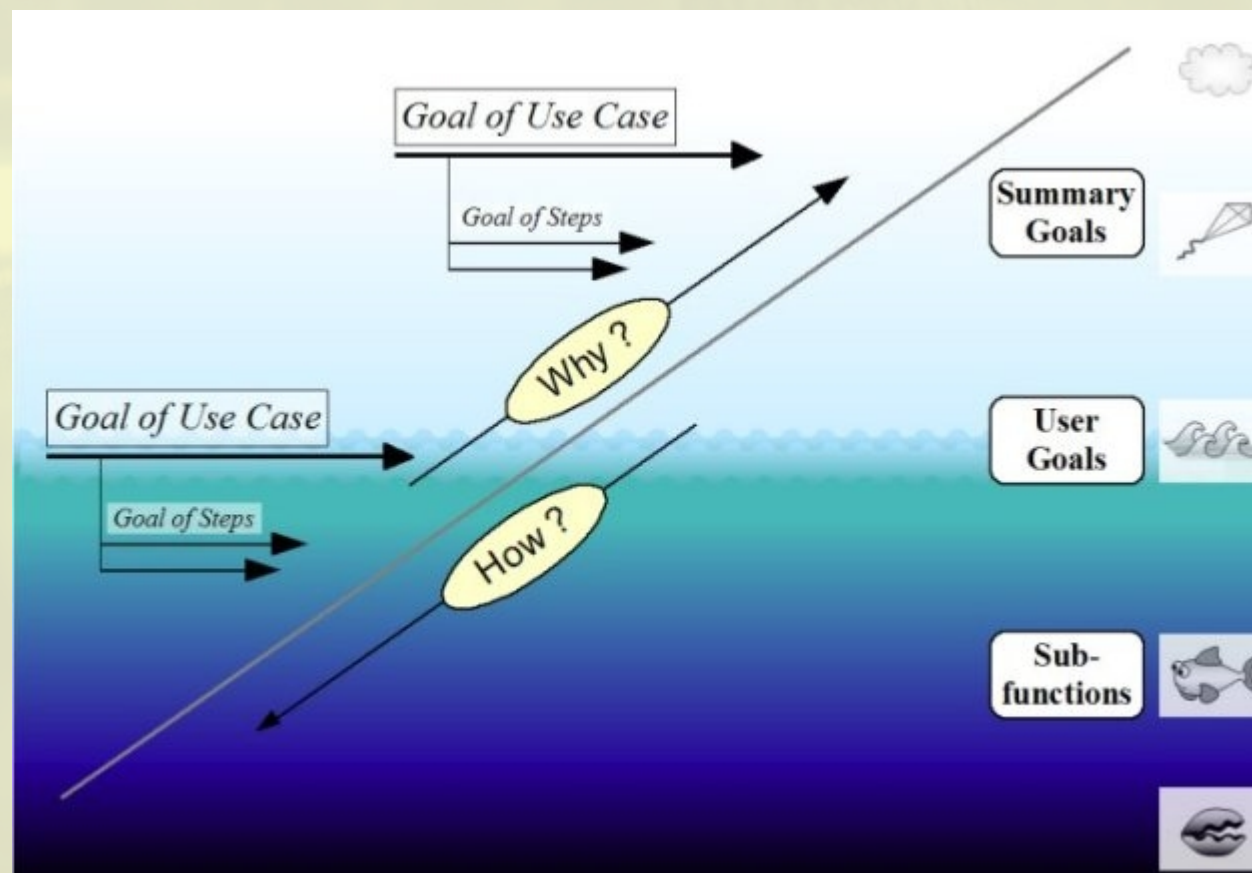
# Use Case Goal Levels: Pictorially

- Use Case goals levels reflect the different usage levels in a system



# Use Case Goal Levels: Emphasis changes as level changes

- Higher level goals address Why, lower level goals address How
  - To move down a level ask How?
  - To move up a level ask Why?







# Use Cases are UI Agnostic

- Emphasis is on WHAT is achieved, not the mechanics of HOW it is achieved
- User Documentation describes the 'how'
- Test Cases for User Interactions can be written in two phases
  - First phase ensures the Use Case path is clear
  - Second phase adds details based on User Doc
- Acceptance tests can validate both the UI and the User Documentation independently
- Use Cases are therefore very useful when used to capture complex User Interactions


# Use Case Summary and Sample

- The Use Case clearly and succinctly provides an interaction capability promise
  - Preconditions and Guarantees are important
  - The Main Success Scenario is not overly long
  - There is no UI information
    - There shouldn't be. Such detail depends on the implementation
  - The Extensions can be implemented and tested separately
- Use Cases are Iteration tools

**USE CASE 201**  **The user needs to be able to configure initial host(s)** 


 Edit

<b>Goal in Context</b>	The user needs to be able to configure initial host(s)
<b>Scope</b>	NMS
<b>Level</b>	User-Goal
<b>Primary Actor</b>	System Administrator
<b>Stakeholders &amp; Interests</b>	System Administrator, NMS Administrator, NMS
<b>Preconditions</b>	Install must be in progress
<b>Minimal Guarantee</b>	System Administrator gets a message
<b>Success Guarantees</b>	Host(s) get configured
<b>Trigger</b>	System Administrator selects to configure the host(s) during the install

 Edit


**Main Success Scenario**

- 1 Sys Admin elects to configure an initial host
- 2 Sys admin enters [Host Information](#) (repeatable)
- 3 Sys admin submits [Host Information](#)(may return to step 3)
- 4 Success message sent to sys admin; install continues.

 Edit

**Extensions**

<a href="#">edit</a>	<b>3a</b> Host already entered
<a href="#">edit</a>	<b>3a1.</b> Message to admin, return to 3 or continue.
<a href="#">edit</a>	<b>3b</b> Invalid Host Information
<a href="#">edit</a>	<b>3b1.</b> Send message; Return to 3
<a href="#">edit</a>	<b>3c.</b> Unable to add <a href="#">Host Information</a>
<a href="#">edit</a>	<b>3c1.</b> Message to Admin

 Edit

# Benefit: Roadmap and Progress Reporting are Side-Effects

- In a typical waterfall development scenario the artifacts that are used to estimate and track progress are generated explicitly
  - Conscious effort is required to produce these
  - Conscious effort is required keep these up-to-date
  - Often the overhead is so large one person in the team ends up doing this work almost exclusively
  - If all of the above is overcome the accuracy of the data is poor and largely based on intuition
- In Agile these artifacts are generated automatically as a side-effect of the methodology
  - The data produced is as accurate as it can be, as it is based on known current progress and observed previous progress



# Benefit: It is possible to 'try' different scenarios to understand the effect

- Expected progress is based on observed progress. We can therefore experiment with alternate Roadmaps to see the expected progress for each
- It is also possible to create artificial scenarios that may impact the Roadmap and observe the effect they may have
  - For example estimate the effect of an increased support workload by modifying the rate of story point completion
- Stakeholders can therefore make considered decisions regarding possible Roadmap variations

# Benefit: The Time Estimate is decoupled from the Effort Estimate

- Estimates of completion time is calculated based on observed progress, not directly from developers' time estimates
  - Developers are no longer under pressure to give overly optimistic estimates of effort, or artificially pad their estimates
  - Stakeholders benefit from a more accurate and honest understanding of future progress making their decisions more informed
- The fixed sizing scale is simple and easy to understand by all stakeholders
  - Areas of high risk are easy to identify as they manifest themselves as large Stories
  - Risk can be tackled by prioritising and splitting large Stories

# Benefit: Waste is minimised and Bottle-necks can be detected

- Minimising waste is an important underlying principle of Agile development
  - Expensive development effort is only used when there is a definite prioritised need for the work
  - If product direction changes at most an iteration's period of work is affected
- It is possible to detect and quantify Resource Bottle-necks
  - Bottle-necks are visible as story point completion rate drops to near zero
  - Potential bottle-necks can manifest themselves by observing falling rate averages or sudden increases in the total amount of prioritised User Stories



# Benefit: Stakeholders can always see who benefits from work done

- The product is described in terms of User Stories and Use Cases, both of which always identify at least one User
- User Stories provide a uniform method of capturing both functional and non-functional (constraints) requirements
  - This makes it easier for stakeholders to prioritise business value need
  - Ticket Issues can be traced back to promised business value
- The product is described in a language that the customer understands
- Development learn to view the product from User perspectives, improving domain understanding

# A Brief Prescription for Agility – As a Team get the basics in place

- Start with the basic framework
  - Choose a fixed iteration length
    - Larger teams favour shorter iterations
    - Try not to have iterations longer than 3 weeks to keep expectations in check
    - Include a planning, development and evaluation phase for each iteration
  - Identify a core set of dependency ordered, relative sized User Stories
    - Make sure you have enough to address the most needed work and sufficient for a couple of iterations
    - You can then get started into an iteration and continue to grow the Story list in subsequent planning and evaluation phases
  - Put in place mechanisms to evaluate when Stories are DONE
    - Such as automated builds which runs tests



# Understand how the Methodology will interface with the Organisation

- Typically when a team becomes Agile it doesn't have any control over the rest of the organisation
- An advantage of Agile methodologies is that progress indicators are produced as a side-effect of the methodology
  - However if the organisation cannot effectively consume those indicators it will be hard gain value from them
- Spend time understanding the interfaces within your organisation so that you can satisfy the need for progress information
  - Where possible generate this automatically from the core information gathered as a side-effect of the methodology
  - Where possible discourage use of 'dead' documents
  - Where possible encourage use of centralised 'living' documentation, for example on a Wiki



# Supporting Tools

- The primary purpose of most tools in Agile development is either to
  - Increase communication bandwidth
  - Minimise overhead for capturing and sharing knowledge
  - Facilitate timely communication of progress to provide good stakeholder visibility
- Typical tools used are
  - Email (and Newsgroups)
  - Instant Messaging and Web meetings
  - Wikis and Web-pages in general
  - Ticket Systems
  - Source Control Management tools
  - Automated Tests and Builds with result reporting

# Typical Tool Use in Agile Development

## User Stories to Milestones

- User Stories
  - Basically a Special kind of Issue (Ticket System)
  - Must be editable with rich content (Wiki)
  - Tickets can be raised against User Stories
- Use Cases
  - Template based for ease of completion and Editable (Wiki, Ticket System)
- Milestones
  - Should be automatically updated as Stories become Done (Wiki, Trac, Script Driven Web-page or similar)
  - Must be visible to Stakeholders (Web-page)
  - Must be clear – this is the main focus for understanding progress

# Typical Tools for Planning, Evaluation and Everything Else

- Release Planning
  - May need to capture dependencies in a graphical form to ease understanding (Graphviz, Freemind)
- Iteration Planning
  - Must have a scratch pad for intent and to capture commitment (Wiki)
  - Links to User Stories (Wiki)
- Iteration Evaluation
  - Should communicate a summary of what was achieved (Wiki)
  - Should have place to capture the outcome of the Retrospective (Wiki)
- Communicate however you can, but always use lowest common bandwidth medium in a group setting.



# Finally – the Popular Perspective



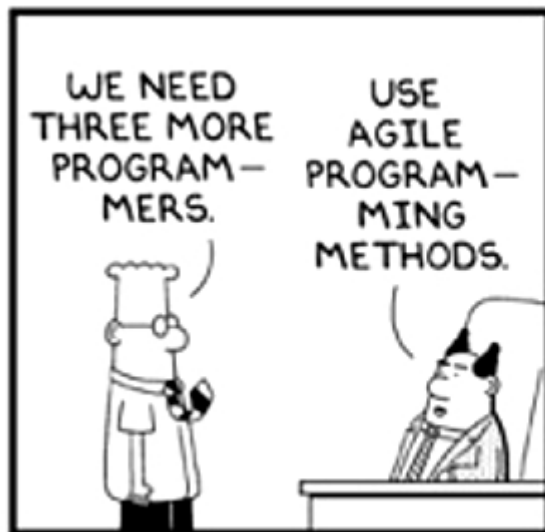
www.dilbert.com  
scottadams@aol.com



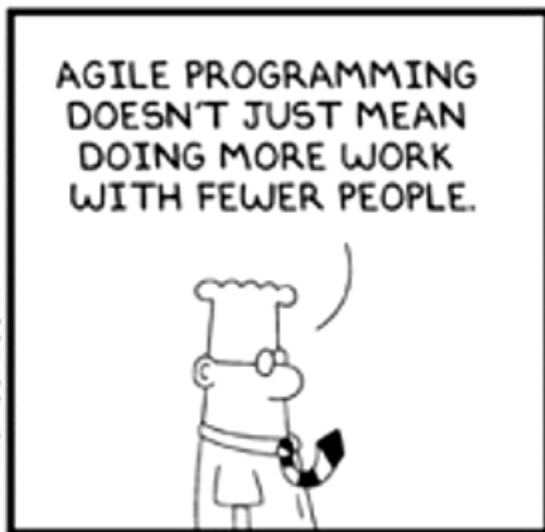
www.dilbert.com  
scottadams@aol.com



© Scott Adams, Inc./Dist. by UFS, Inc.



www.dilbert.com  
scottadams@aol.com



www.dilbert.com  
scottadams@aol.com



© Scott Adams, Inc./Dist. by UFS, Inc.