

COMP 1531

Software Engineering Fundamentals

Week 06

Agile Development Methodologies

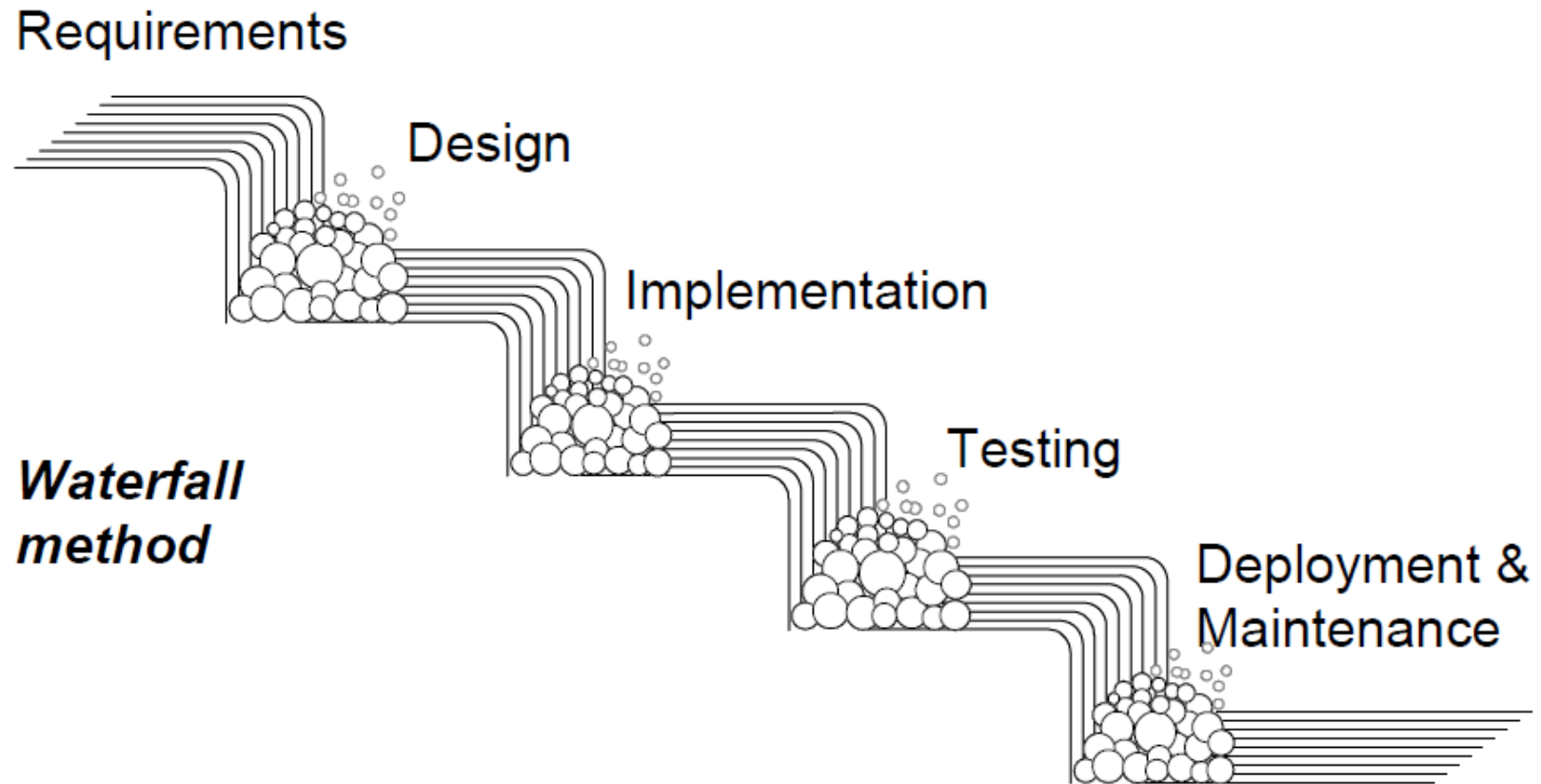
Revision: Software Engineering Life-Cycle

- Software engineering is a complex, organised process with a great emphasis on *methodology*.
- A *software development methodology* lays out a prescriptive process by mandating a *sequence of tasks* such as:
 - Analysis and Specification
 - Design
 - Implementation
 - Testing
 - Release & Maintenance

Software Development Methodologies

- **Elaborate processes:**
 - Rigid, plan-driven documentation heavy methodologies e.g., Waterfall approach
 - Unidirectional, finish this step before moving to the next
- **Iterative & Incremental processes** which develop increments of functionality and repeat in a feedback loop
 - Rational Unified Process [Jacobson et al., 1999]
 - Agile methods (e.g., SCRUM, XP): methods that are more aggressive in terms of short iterations

Revision: Waterfall Model (1970's)



Waterfall process for software development.

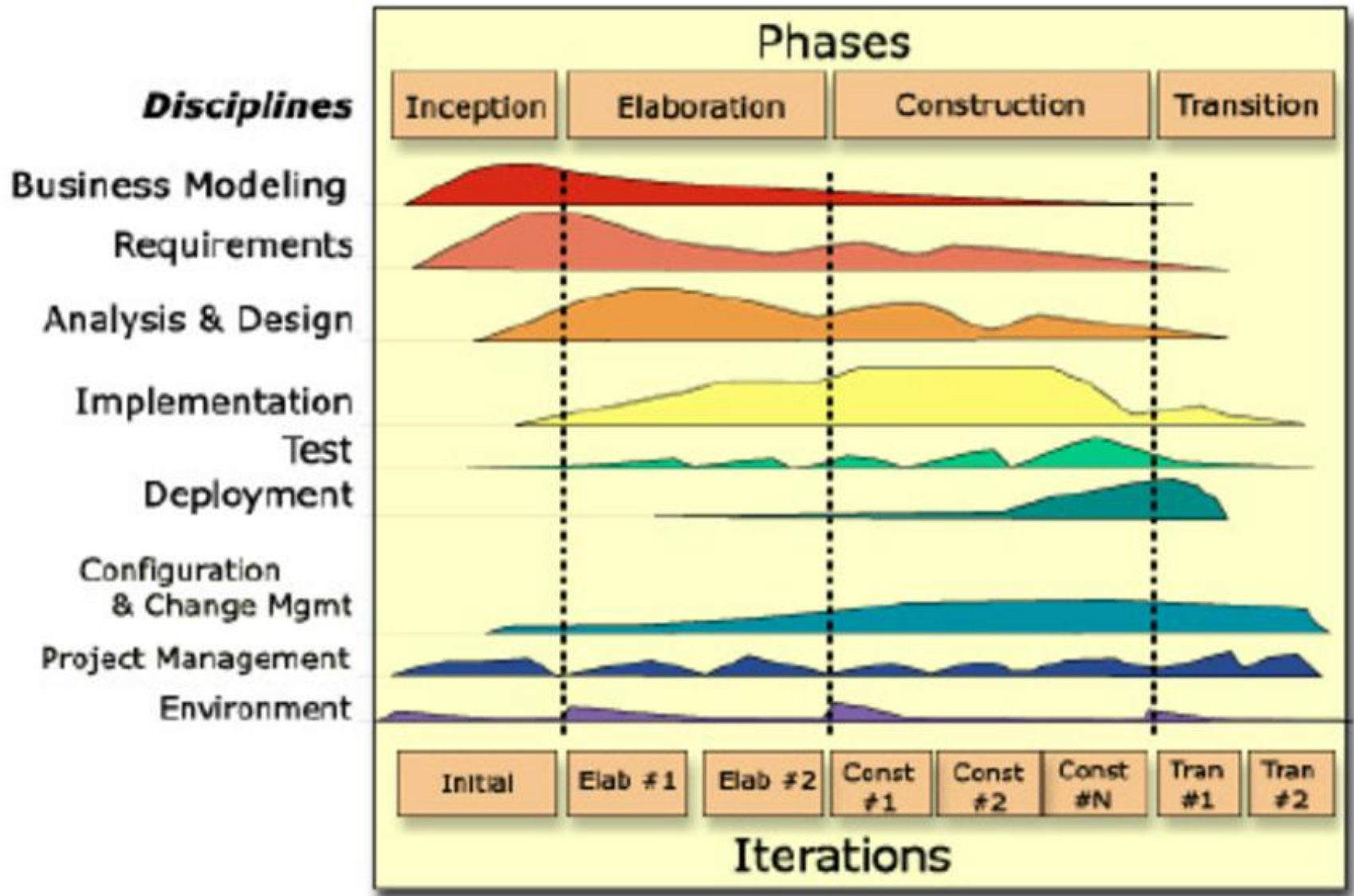
Incremental and Iterative Project Life-Cycle

- **Incremental and iterative** approaches:
 1. Break the big problem down into smaller pieces and prioritize them.
 2. An “iteration” refers to a step in the life-cycle
 3. Each “iteration” results in an “increment” or progress through the overall project
 4. Seek the customer feedback and change course based on improved understanding at the end of each iteration
- An incremental and iterative process
 - seeks to get to a working instance as soon as possible.
 - progressively deepen the understanding or “visualization” of the target product

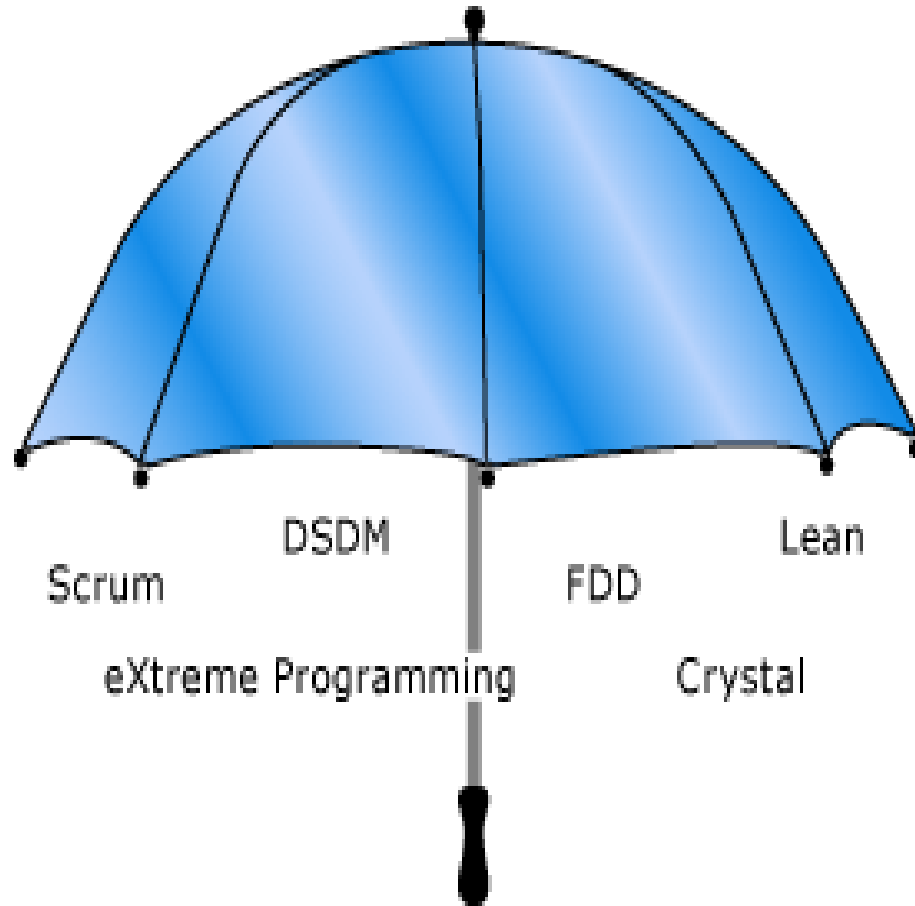
Rational Unified Process (RUP)

- An iterative software development process developed by Ivar Jacobson, Grady Booch and James Rumbaugh and consists of four major phases:
 - **Inception**: scope the project, identify major players, what resources are required, architecture and risks, estimate costs
 - **Elaboration**: understand problem domain, analysis, evaluate in detail required architecture and resources
 - **Construction**: design, build and test software
 - **Transition**: release software to production

RUP: Serial in the large, iterative in the small

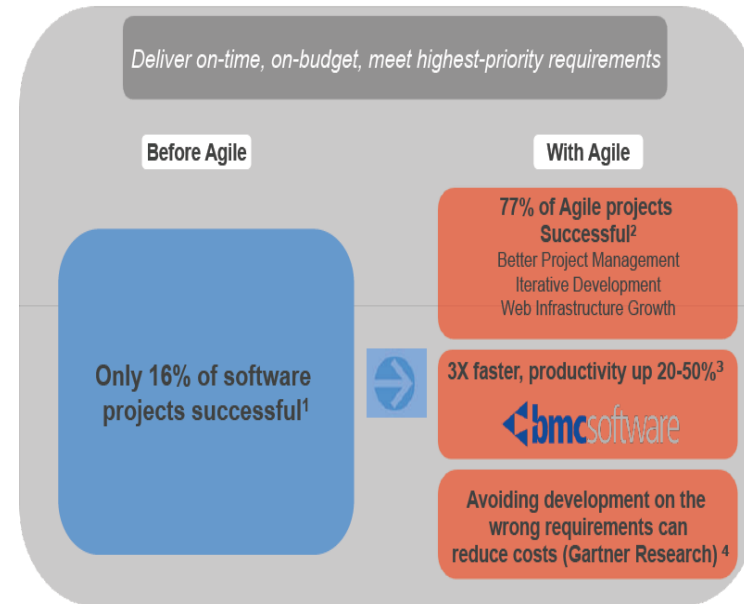


Agile Software Development Methodologies



Why Do We Need Agile?

- **93% increased productivity**
- **88% increased quality**
- **83% improved stakeholder satisfaction**
- **49% reduced costs**
- **66% three-year, risk-adjusted return on investment**



¹ Standish Group Report: There's Less Development Chaos Today, by David Rubinstein *SD Times* March 1, 2007,

² "Agile Has Crossed the Chasm," *Dr. Dobbs's Journal*, July 2, 2007. ³QSMA and Cutter Consortium ROI case study on BMC Software, 2008. ⁴ Gartner, Inc. 2005

³ Why agile - Rally software development corp

⁴ "Agile Methodologies: Survey Results," by Shine Technologies, 2003; 2 Forrester Research, 2004;

Agile Manifesto (Agile Alliance, 2001)

We are uncovering better ways of developing software by doing it and helping others do it.

Through this work we have come to value:

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Extreme Programming (XP)

A prominent agile software engineering methodology that:

- focuses on providing the highest value for the customer in the fastest possible way
- acknowledges changes to requirements as a natural and inescapable aspect of software development
- places higher value on **adaptability** (to changing requirements) over **predictability** (defining all requirements at the beginning of the project) – being able to adapt is a more realistic and cost-effective approach
- aims to lower the cost of change by introducing a set of basic **principles** (high quality, simple design and continuous feedback) and **practices** to bring more flexibility to changes

XP Core Principles: High Quality (1)

- **Pair-programming** - Code written by pairs of programmers working together intensely at the same workstation, where one member of the pair “codes” and the other “reviews”.
- **Continuous Integration** - Programmers check their code in and integrate several times per day
- **Sustainable pace** - Moderate, steady pace
- **Open Workspace** – Open environment
- **Refactoring** – Series of tiny transformations to improve the structure of the system
- **Test-Driven Development** - Unit-testing and User Acceptance Testing

XP Core Principles: Simple Design (2)

- ❖ Focus on the stories in the current iteration and keeps the designs simple and expressive.
- ❖ Migrate the design of the project from iteration to iteration to be the best design for the set of stories currently implemented
- ❖ Spike solutions, prototypes, CRC cards are popular techniques during design
- ❖ Three design mantras for developers:
 - *Consider the simplest possible design* for the current batch of user stories (e.g., if the current iteration can work with flat file, then don't use a database)
 - *You aren't going to need it* – Resist the temptation to add the infrastructure before it is needed (e.g., “Yeah, we know we're going to need that database one day, so we need put the hook in for it?”)
 - *Once and only once* – XP developers don't tolerate duplication of code, wherever they find it, they eliminate it

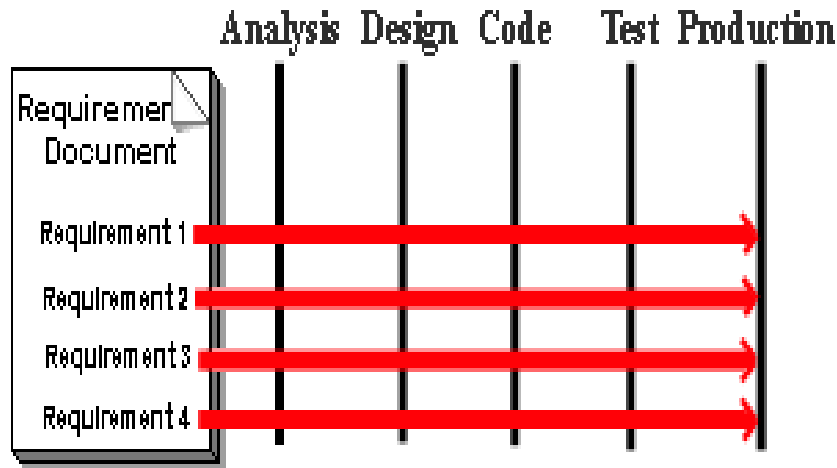
XP Core Principles: Continuous Feedback (3)

An XP team receives intense feedback in many ways, in many levels (developers, team and customer)

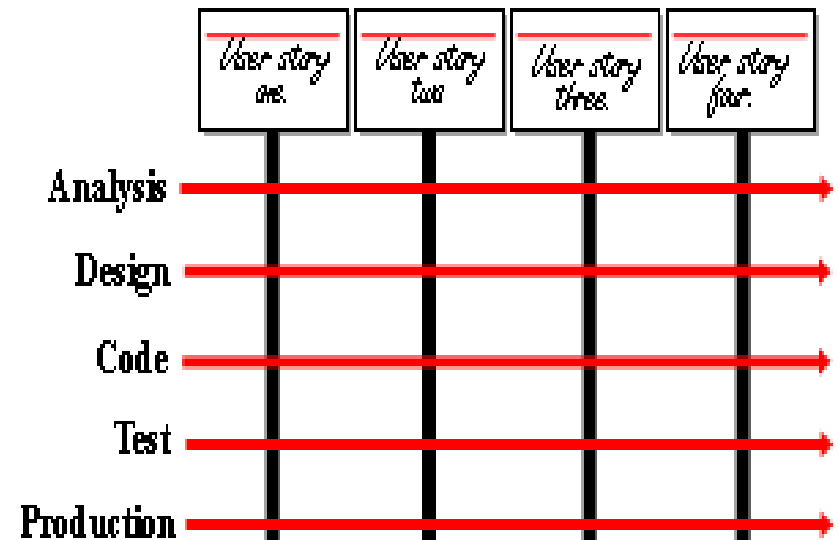
- Developers receive constant feedback by working in pairs, constant testing and continuous integration
- XP team receives daily feedback on progress and obstacles through daily stand-up meetings
- Customers get feedback on progress with user acceptance scores and demonstrations at the end of each iteration
- XP developers deliver value to the customer through producing working software progressively at a “steady heartbeat” and receive customer feedback and changes that are “gladly” accepted.

XP vs Waterfall Life-Cycle

Traditional software development is linear, with each stage of the lifecycle requiring completion of the previous stage.



Waterfall Life Cycle



XP Life Cycle

Extreme Programming (XP) turns the traditional software development process sideways, flipping the axis of the previous chart, where we visualise the activities, keeping the process itself a constant

What happens in XP Planning ?

Key steps in XP Planning Game:

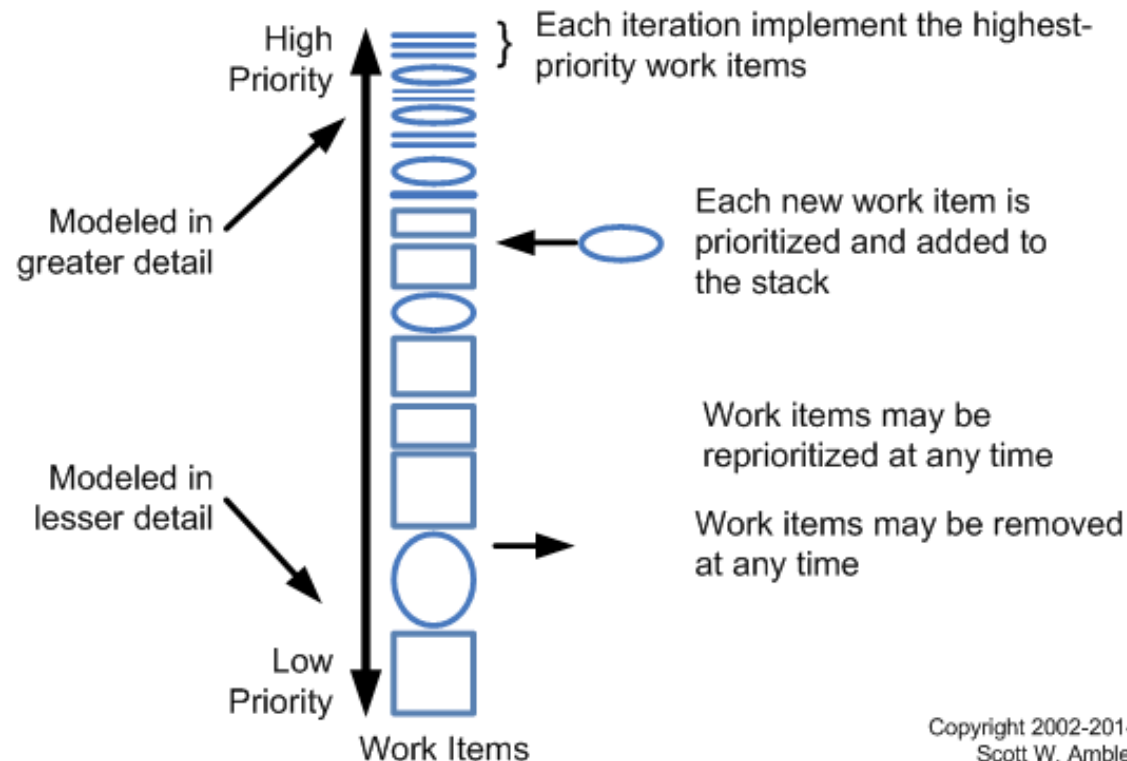
- Initial Exploration
- Release Plan
- Iteration Planning
- Task Planning

The XP Planning Game: Initial Exploration

- Developer and customer have **conversations** about the system-to-be and identify significant features (not “all” features...customers will discover more)
- Each feature broken into one or more **user stories**
- Developers *estimate* the user story in *user story points* based on team’s velocity (becomes more accurate through iterations)
 - Stories that are too large or too small are difficult to estimate. An **epic** story should be split into pieces that aren't too big.
 - Developers complete a certain number of stories each week. Sum of the estimates of the completed stories is a metric known as **velocity**
 - Developers have a more accurate idea of **average velocity** after 3 or 4 weeks, which is used to provide better estimates for ongoing iterations.

User Story and Planning

- User-stories not only capture the user's vision but also impact the **planning process** in two key areas; **estimating** and **scheduling**



The XP Planning Game: Release Plan

- Negotiate a **release date** (6 or 12 or 24 months in the future)
 - Customers specify which user stories are needed and the order for the planned date (business decisions)
 - Customers can't choose more user stories than will fit according to the current project velocity
 - Selection is crude, as initial velocity is inaccurate. RP can be adjusted as velocity becomes more accurate
- Use the **project velocity** to plan
 - by **time**: **compute** #user stories that can be implemented before a given date (multiply number of iterations by the project velocity
 - by **scope**: how long a set of stories will take to finish divide the total weeks of estimated user stories by the project velocity

The XP Planning Game: Iteration Planning

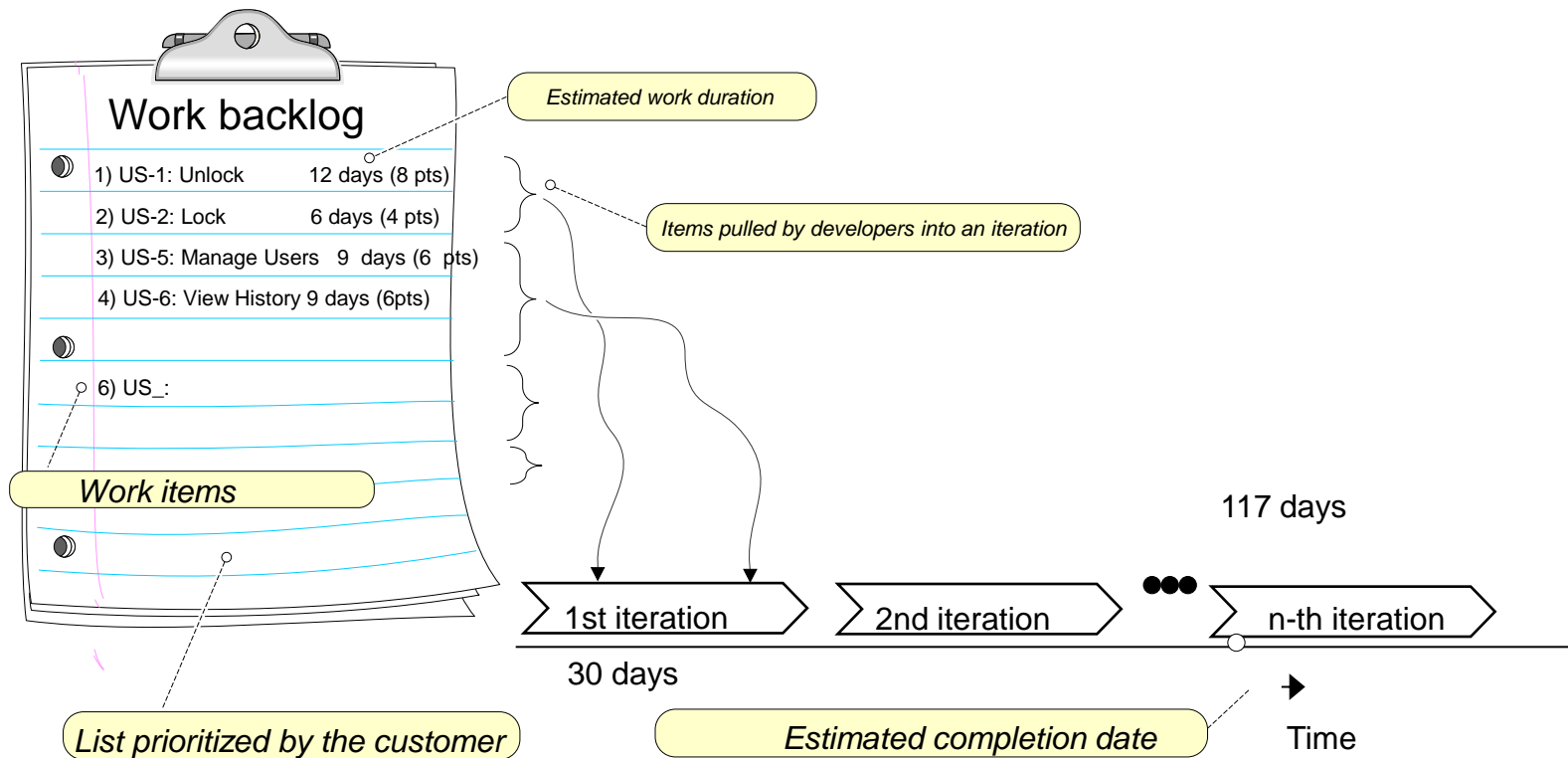
- ❖ Use the release plan to create **iteration plans**
- ❖ Developers and customers choose an iteration size: typically 1 or 2 weeks
- ❖ Customers prioritise user stories from the release plan in the first iteration, but must fit the current velocity
- ❖ Customers cannot change the stories in the iteration once it has begun (can change or reorder any story in the project except the ones in the current iteration)
- ❖ The iteration ends on the specified date, even if all the stories aren't **done**. Estimates for all the completed stories are totalled, and velocity for that iteration is calculated
 - *The planned velocity for each iteration is the measured velocity of the previous iteration.*
- ❖ Defining “done” - **A story is not done until *all* its acceptance tests pass**

The XP Planning Game: Task Planning

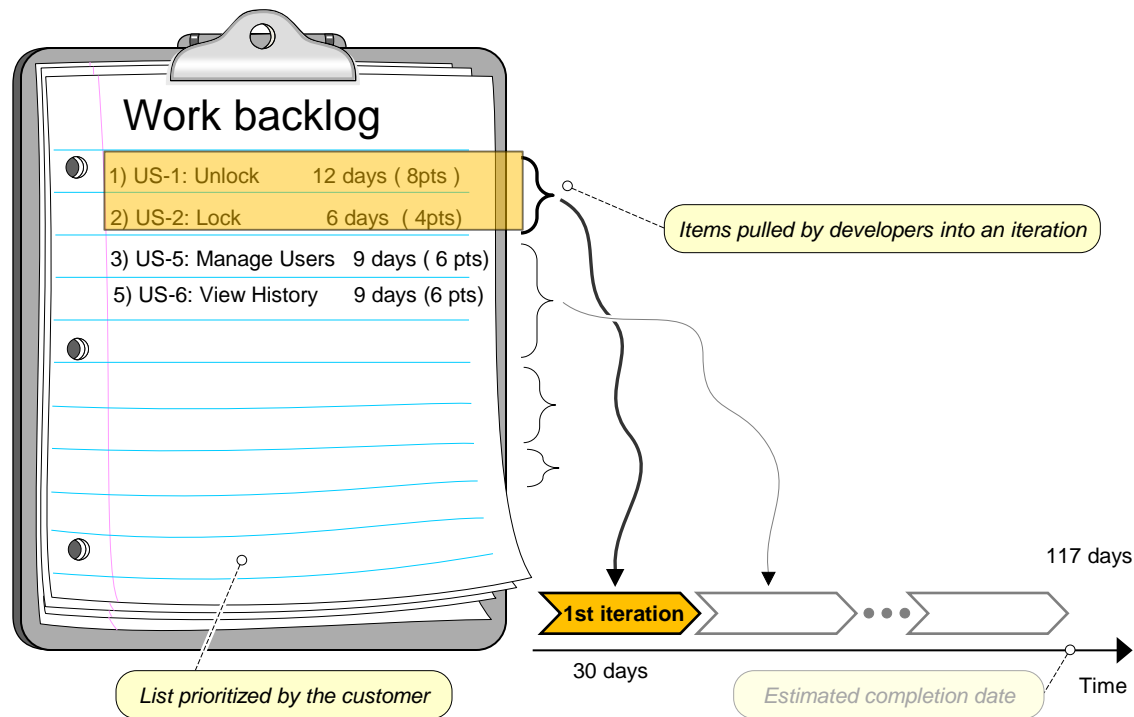
- ❖ Developers and customers arrange an **iteration planning meeting** at the beginning of each iteration
 - Customers choose user stories for the iteration from the release plan but **must** fit the current project velocity
 - User stories are broken down into programming tasks and order of implementation of user stories within the iteration is determined (***technical decision***)
 - Developers may sign up for any kind of tasks and then estimate how long task will take to complete (*developer's budget – from previous iteration experience*)
 - Each task estimated as 1, 2, 3 (or even ½) days of ideal programming days. Tasks < 1 day grouped together, tasks > 3 days broken down
 - Project velocity is used again to determine if the iteration is over-booked or not
 - Time estimates in ideal programming days of the tasks are **summed** up, and this must not exceed the project velocity (initial or from the last iteration).
 - *If the iteration has too much - the customer must choose user stories to be put off until a later iteration (snow plowing). If the iteration has too little then another story can be accepted.*
 - **The velocity in task days (iteration planning) overrides the velocity in story weeks ([release planning](#)) as it is more accurate.**
 - Team holds a meeting halfway through iteration to track progress

Agile Project Effort Estimation for case-study

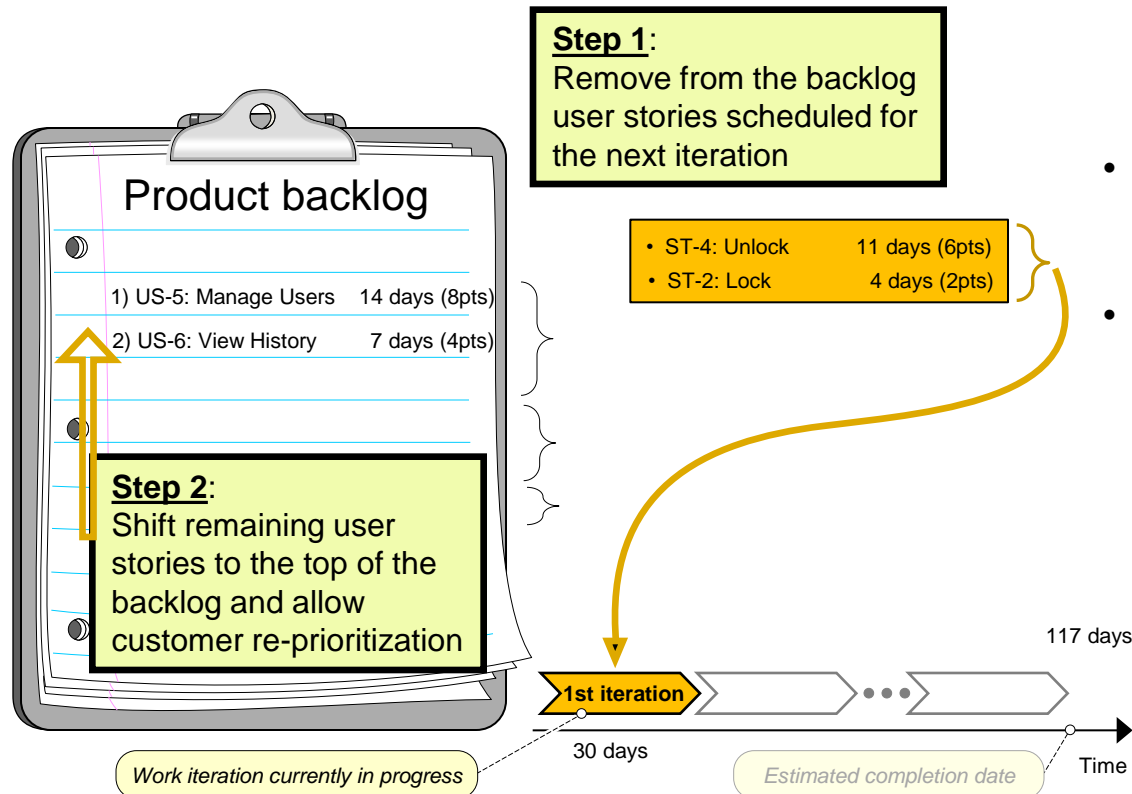
Total project effort is estimated based on the cumulative story points of all user-stories



Agile Prioritization of Work



Update Product Backlog between iterations



- Items pulled by developers into an iteration are not subject to further customer prioritization
- Developers have a **steady goal** until the end of the current iteration
- Customer has **flexibility** to change priorities in response to changing market forces

Project Velocity

- Size points assigned to each user story
- Total work size estimate:
 - Total size = $\sum (\text{points-for-story } i), \quad i = 1..N$
- Estimate **project velocity** (= team's productivity) - estimated from the number of user-story points that the team can complete in a particular iteration (enables customers to obtain an idea of the cost of each story, its business value and priority)
(e.g., if 42 points' worth of stories are completed during the previous week, the velocity is 42)
- Estimate the project duration
 - Project duration = Total Work Size / Project Velocity

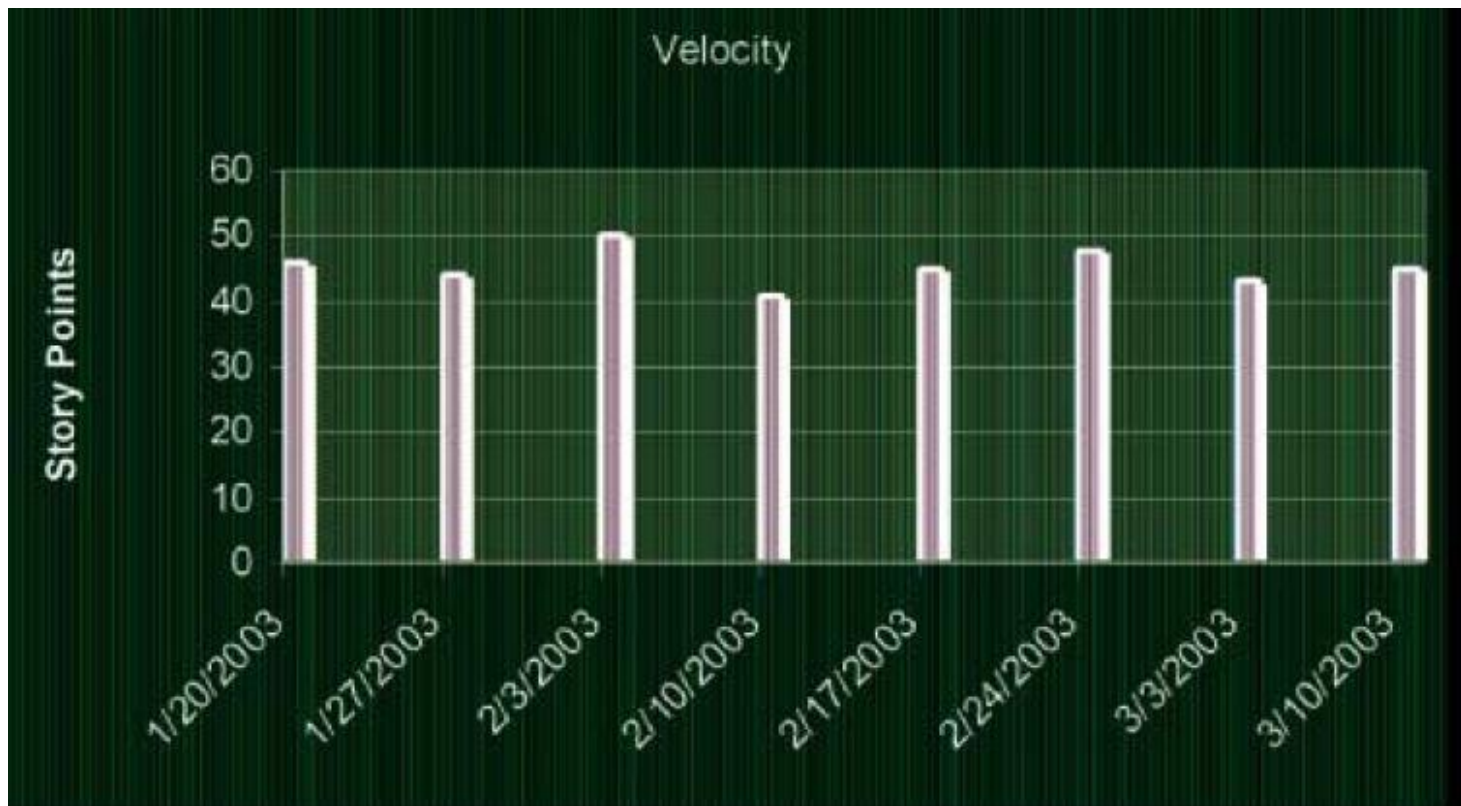
Project Tracking

The recording the results of each iteration and use those results to predict what will happen in the next iteration

- Tracking the total amount of work done during each iteration is the key to keep the project running at a **sustainable, steady pace**
- XP teams use a **velocity chart** or **burn-down chart** to track the project velocity which shows how many story points were completed (passed the user acceptance tests)
- These tools provide a reliable project management information for XP teams

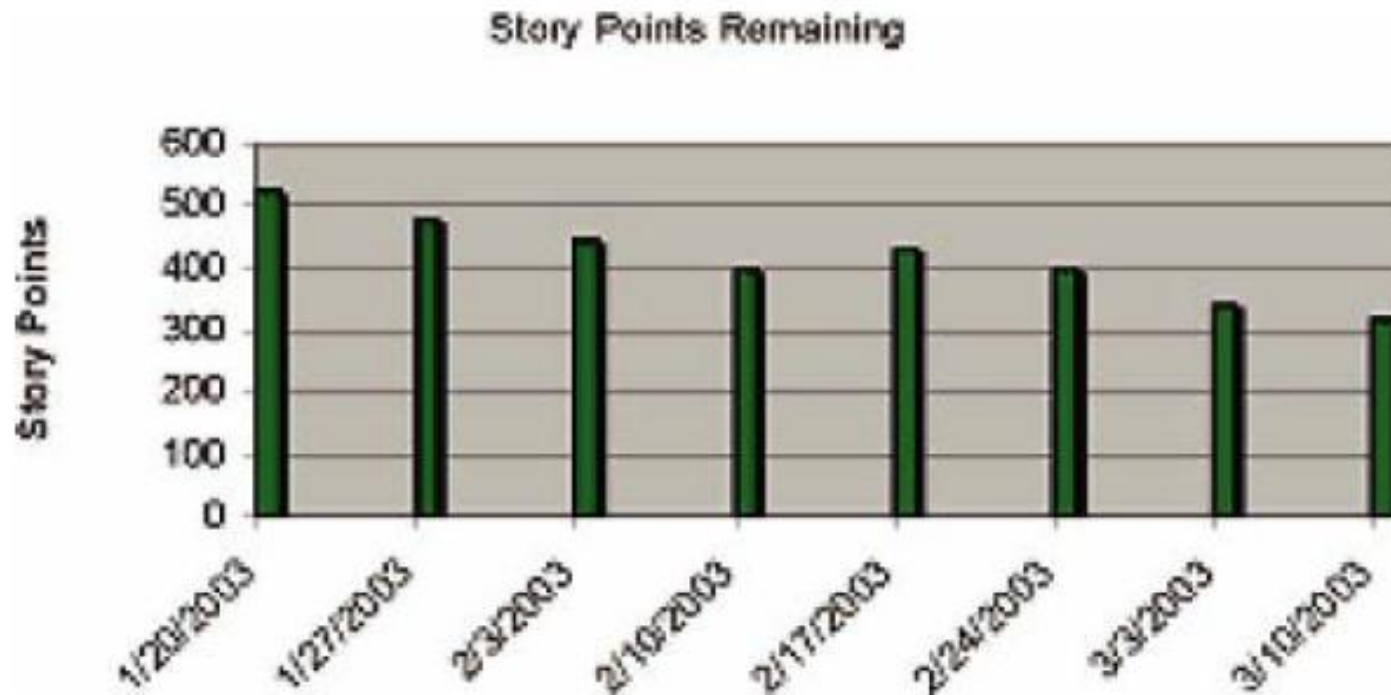
Project Tracking using Velocity Chart

- XP teams use a **velocity chart** to track the project velocity which shows how many story points were completed (passed the user acceptance tests)
- Average project velocity in this example is approximately 42 story points



Project Tracking Using Burn-Down Chart

- ❖ A **burn-down** chart shows the week-by-week progress
 - The slope of the chart is a reasonable predictor of the end-date
 - The difference between the bars in the burn-down chart does not equal the height of the bars in the velocity chart as new stories are being added to the project. (may also indicate that the developers have re-estimated the stories)



Agile drawbacks

- Daily stand up meetings, close collaboration – not ideal for development outsourcing, clients and developers separated geographically, or business clients who simply don't have the manpower, resources
- Emphasis on modularity, incremental development, and adaptability – not suited to clients desiring contracts with firm estimates and schedule
- Reliance on small self-organized teams makes it difficult to adapt to large software projects with many stakeholders with different needs and neglects to take into account the need for leadership while team members get used to working together.
- Lack of comprehensive documentation can make it difficult to maintain or add to the software after members of the original team turn over
- Agile development – Need highly experienced software engineers who know how to both work independently and interface effectively with business users.

When should XP be used?

- ❖ Useful for problem domains where requirements change, when customers do not have a firm idea of what they want
- ❖ XP was set up to address project risk. XP practices mitigate risk and increase the likelihood of success (e.g. a new challenge for a software group to be delivered by a specific date)
- ❖ XP ideally suitable for project group sizes of 2-12
- ❖ XP requires an extended development team comprising managers, developers and customers all collaborating closely
- ❖ XP also places great emphasis on *testability* and stresses creating automated unit and acceptance tests
- ❖ XP projects deliver greater productivity, although this was not aimed as the goal of XP

Which methodology?

- What does the customer want?
 - **need software yesterday with the most advanced features at the lowest possible cost !**
- No one methodology is the best fit
- Successful software development - understand all three processes in depth and take the parts of each that are most suited to your particular product and environment.
- Stay agile in your approach through constant re-evaluation and revising the development process
- SaaS (Software as a Service) and Web 2.0 applications that require moderate adaptability are likely to be suited to agile style
- Mission-critical applications such as military, medical that require a high degree of predictability are more suited to waterfall