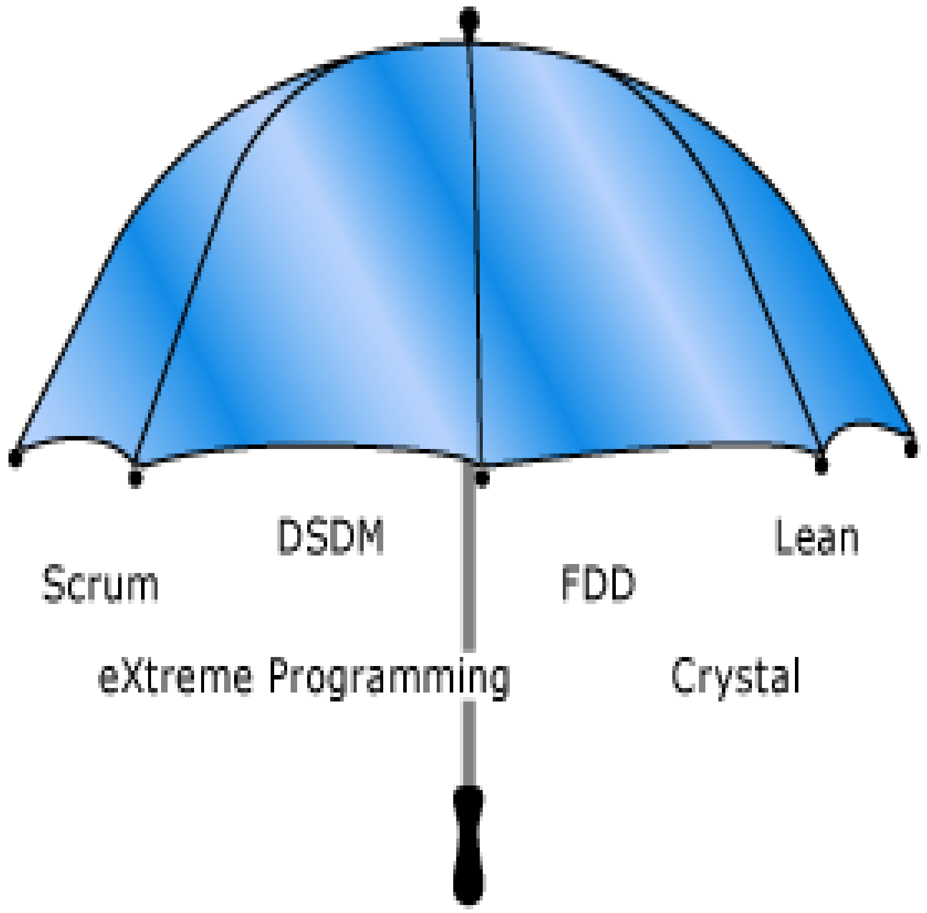
COMP 1531 Software Engineering Fundamentals

Week 02 Tuesday

XP, Scrum

Aarthi Natarajan

The Agile Umbrella

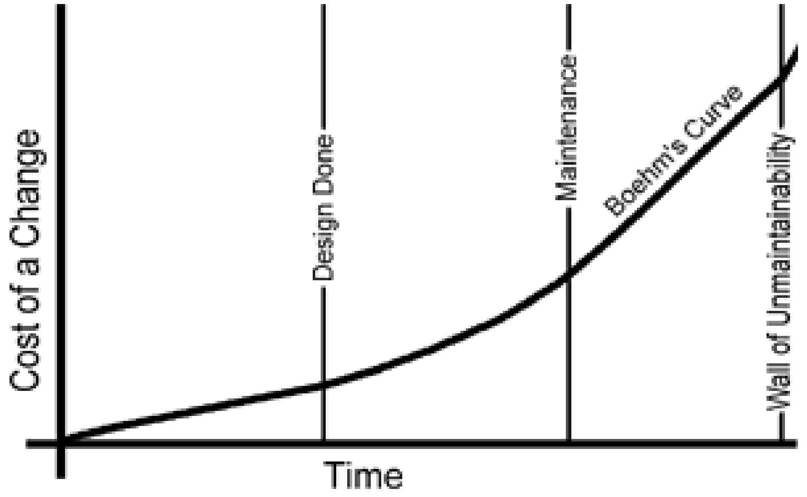


Extreme Programming (XP)

- ❖ 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 values, principles and practices to bring more flexibility to changes

Boehm’s Cost Curve

As software proceeded through its life cycle the cost of making a change became larger.



Commonly incorrectly interpreted as – Create infallible requirements documents and detailed error-free designs OR pay a huge price ! A better interpretation is: Be prepared when changes occur.

Software can rot! How?

- improper design; complexity creeps as easy code changes are preferred over difficult design changes
- code duplication during maintenance tasks
- limited project resources and cost of making vital changes exceeds resources

The Values of Extreme Programming

Communication

- Increased collaboration of customers and developers,
- Customer is a member of them team, who defines and prioritises features,
- XP engages techniques for rapid dissemination of information among developers and give them a shared view of the system that matches the customer view

Simplicity

- Simplest solution – focus on design & coding on needs of today as designing for uncertain future requirements carries a risk of spending unnecessary resources
- Maximise the value created for the investment up to date

Feedback

- Feedback from the system by writing unit tests or running periodic integration tests
- Feedback from the customer through UAT written by the customer and plan review in cycles of 2 or 3 weeks
- Feedback from the team
- Feedback is closely related to communication and simplicity

Courage

XP techniques encourage developers to:

- design and code for today and not for tomorrow,
- be comfortable with refactoring code to build better solutions
- tell the truth about progress and estimates
- don't fear anything as no one is along
- to take on problems proactively and integrate testing and changes in the development phase

Respect

XP techniques encourage developers to:

- never commit changes that break compilation or fail unit-tests
- strive for high-quality and seeking for the best design

The XP Principles: Whole Team

- Managers, developers and developers collaborate work closely as a single team, being aware of one another’s problems and collaborating to solve these problems.
- Customer of an XP team (who defines and prioritises features) can be a group of BA or marketing specialists but must be a member of the team and available to the team at all times.
- The best case is for the customer to work in the same room as the developers. Next best is if the
- Customer works within 100' of the developers. The larger the distance, the more difficult it is for the customer to be a true team member.
- If your customer cannot be close by, find a customer representative who is willing and able to stand in for the true customer and provide the daily connection.

The XP Principles: User Stories

- A ***user story*** is:
  - a mnemonic token of an ongoing conversation about a requirement and are at the heart of the planning of XP
  - Developers and customers have conversations with the customer to get a sense of the requirements and record each requirement as a user-story; a short short, simple description of a feature narrated from the perspective of the person who desires that capability

    As a < type of user >, I want < some goal > so that < some reason
  - Often written on index cards or sticky notes and arranged on walls or tables which helps to shift focus from writing about features to **planning** and **discussion (**more important than written text)
  - Serves as a planning tool to schedule the implementation of a requirement, based on its priority and estimated cost.

The XP Principles: Short Cycles

Based on the above user stories, an XP team creates a:

- **Project Release plan**
  - A release is usually three months' worth of work and represents a major delivery that can be deployed to production and often maps out the next six or so iterations
  - Consists of prioritized collections of user stories that have been selected by the customer according to a budget presented by the developers.
  - Customer may select any number of stories for the release, so long as the total of the estimate does not exceed that budget.
  - Business determines the order in which the stories will be implemented in the release. If the team so desires, it can map out the first few iterations of the release by showing which stories will be completed in which iterations.
  - And finally a break-down of stories into tasks with developer sign up for the current iteration.
- **Iteration Plan:**
  - XP project works in **short cycles,** delivering working software every two weeks.
  - An iteration plan represents a minor delivery consisting of a collection of user stories selected by the customer according to a budget established by developers. (based on task completed in previous iteration)
  - The customer may select any number of stories for the iteration so long as the total of the estimate does not exceed that budget.
  - Business cannot change the definition or priority of the user-stories once an iteration has been started.
  - During this time, the developers are free to cut the stories up into *tasks* and to develop the tasks in the order that makes the most technical and business sense.
  - At the end of the two weeks, system is demonstrated to the stakeholders.

❖ Plans are considered temporary artifacts in XP. You will be expected to re-create your plans before they expire. Every time the customer gains insight you will make a new plan. Every time you slip or get ahead of your schedule you will make a new plan.

The XP Principles: High Quality

- *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".
  - Roles can change frequently. Change pair partners so that every team member has worked with every other member of the team
  - Spread knowledge throughout the team, through pairing with members assigned to specialist tasks
  - No **collective ownership**, pair can check out any module and improve it
- *Continuous Integration*
  - Programmers check their code in and integrate several times per day
  - First one to check in wins, every one else merges
- *Sustainable pace* – Team must run at a moderate, steady pace to conserve energy and alertness (a marathon, not a sprint) and **not** allowed to work overtime
- *Open Workspace* – Teams work in an open room, typically with tables of 2 or 3 workstations, walls covered with status charts, task breakdowns, UML artifacts…
- *Refactoring* – New features, requirement changes, bug fixes cause structure of code to degrade (software rot). **Refactoring** (practice of making a series of tiny transformations to improve structure of the system without affecting the behaviour) helps to reverse software degradation

The XP Principles: High Quality

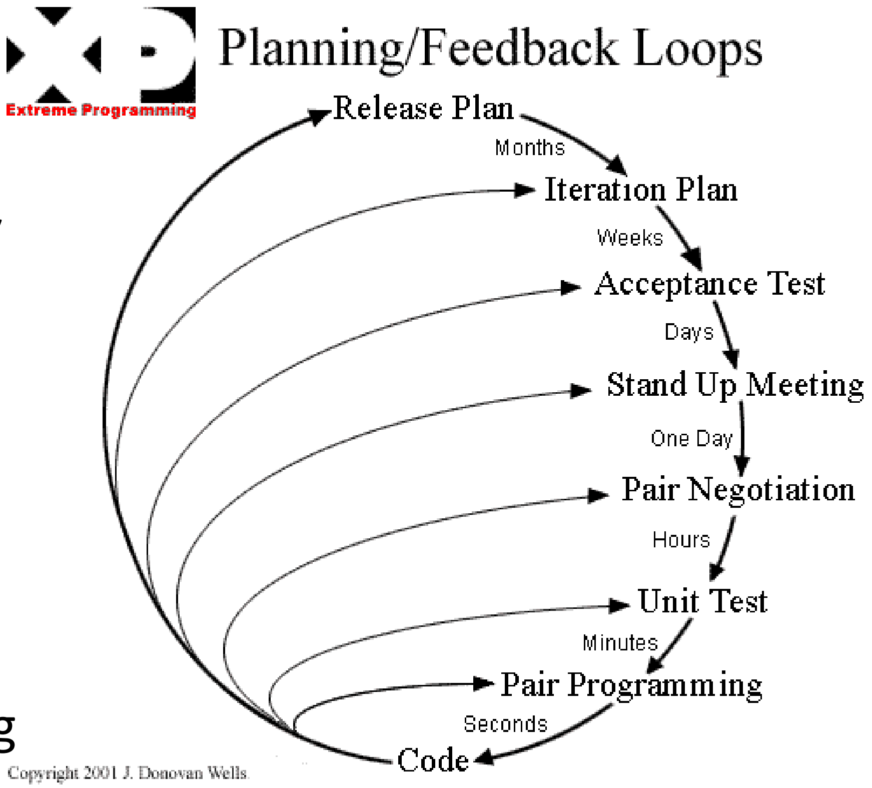
- ***Test-driven Development* (TDD)** – Detailed test coverage at two levels:
  - *Unit Testing*
    - A complete body of test-cases evolve along with the code.
    - When you write code that passes a unit test, that code by definition is “testable”.
    - Unit tests encourages developers to “decouple” modules, so that they can be test independently (**Low coupling**)
    - Unit tests facilitate “refactoring”
  - *User Acceptance Testing*
    - Written by customers or business analysts and become the “true” requirements document of the project; every detail about every feature is described in the acceptance test
    - Serve as the final authority to determine the correctness of the implementation.
    - Once an acceptance test passes, it is added to the body of passing acceptance tests and is never allowed to fail again.
    - Growing body of acceptance tests run several times and every time the system is integrated and built. If an acceptance tests fails, the build is declared a failure. Thus, once a requirement is implemented, it is never broken.

The XP Principles: Simple Design

- An XP team focuses 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

The XP Principles: Continuous Feedback

- ❖ An XP team receives intense feedback in many ways, in many levels
  - 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.



The XP Principles: System Metaphor

- One of the trickiest, poorly understood principles of XP, yet one of the most important practices of all.

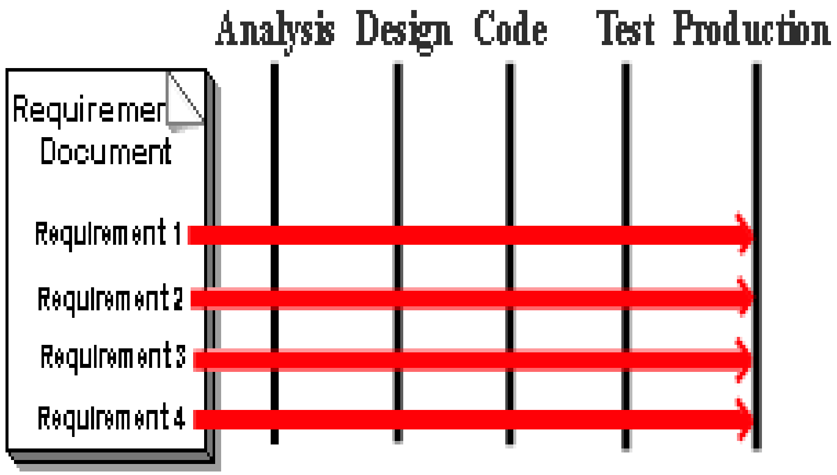
  Dictionary definition: *a figure of speech which makes an implied comparison between things which are not literally alike*

  System metaphor*: is a story that everyone – customers, programmers, and managers – can tell about how the system works."*
  Kent Beck, Extreme Programming Explained*, p. 179.*

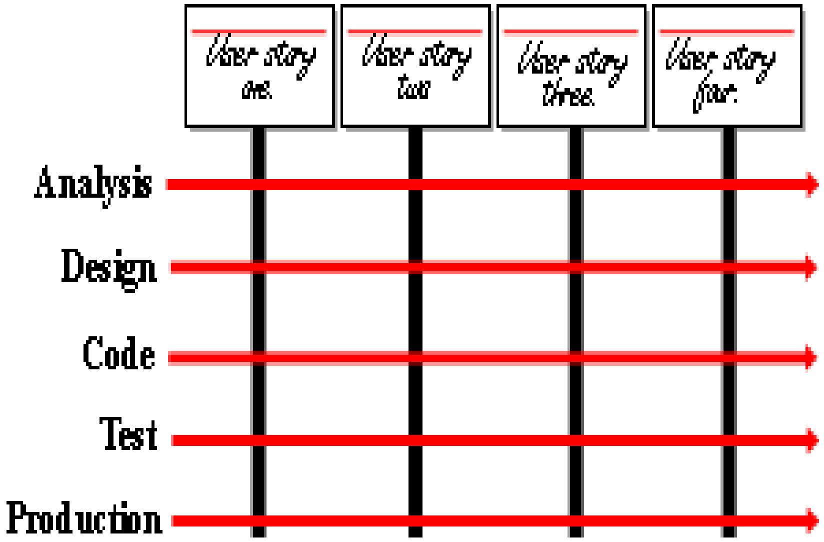
- A system metaphor in necessary for:
  - *A common vision:* Makes it easier to understand what the system is (or what it could be) and enables everyone to agree on how the system works. The metaphor suggests the key structure of how the problem and the solution are perceived.
  - *Shared vocabulary:* Often, helps to suggest a common system of names, which provide a vocabulary for elements in the system and helps define relationships between them (a powerful, specialised, shorthand vocabulary for the team)
  - *Architecture:* Shapes the system, by identifying key objects and suggesting aspects of their interfaces. It supports the static and dynamic object models of the system.
  - *Generativity:* The analogies of a metaphor can suggest new ideas about the system (both problems and solutions)

XP Programming Life-Cycle

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

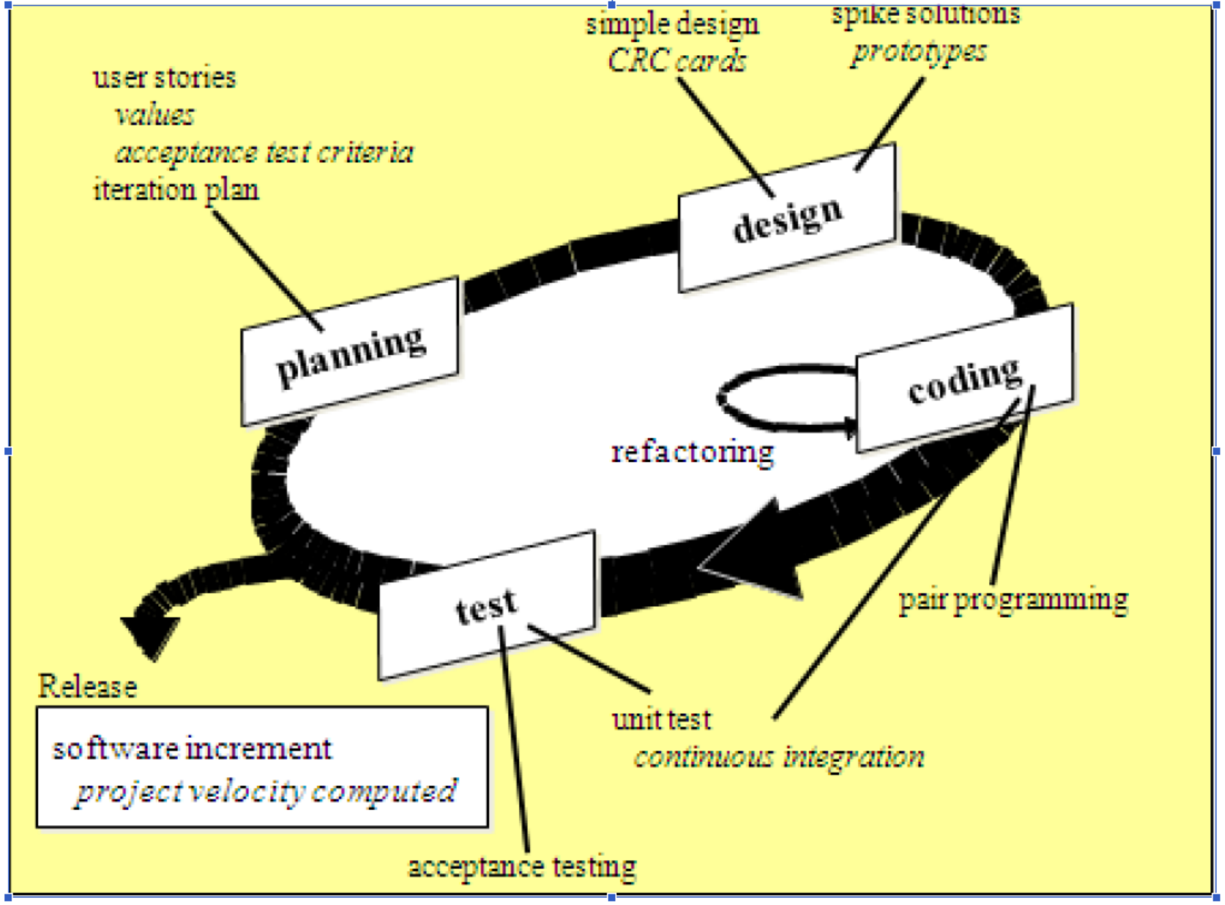


Extreme Programming (XP) turns the traditional software development process sideways, flipping the axis of the previous chart, where we visualise the activities, keeping the process itself a constant



XP Programming Life-Cycle

The Extreme Programming software development process starts with **planning**, and all iterations consist of four basic phases in its life cycle: designing, coding, testing, and listening.



© http://www.agile-process.org/

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*** captured onto index card (short description)
- Developers *estimate* the user story (relative) in *user story point*s
  - Stories that are too large or too small are difficult to estimate. Developers tend to underestimate large stories and overestimate small ones. 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*** (e.g., if 42 points' worth of stories are completed during the previous week, the velocity is 42).
  - Developers have a more accurate idea of ***average velocity*** after 3 or 4 weeks, which is used to provide better estimates for ongoing iterations. ***Tracking velocity*** is one of the most management tools in an XP project.
  - Project velocity enables customers to obtain an idea of the cost of each story, its business value and priority

The XP Planning Game: Release Plan

❖ Use a **release planning meeting** to create a **release plan**:

- Plan a **release date** (6 or 12 or 24 months in the future) and customers specify which user stories are needed and the order for the planned date (business decisions)
  - Customers cannot 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** (how many user stories can be implemented before a given date) or **scope** (how long a set of stories will take to finish)
  - When planning by time, multiply number of iterations by the project velocity to determine the # of user stories that can be completed
  - When planning by scope divide the total weeks of estimated user stories by the project velocity to determine how many iterations till the release is ready
- Negotiate a release plan all developers, customers and manager can agree to and do not underestimate
- Base philosophy of release planning: a project may be quantified by four variables; scope (how much is to be done), release (how many people are available), time (when the release is done) and quality (how good the software will be). One variable impacts the other.
- Use the release plan to create **iteration plans**

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 in the first iteration, but must fit the current velocity
- Order of implementation of user stories within the iteration is a technical decision
- 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 first iteration from the release plan but **must** fit the current project velocity
  - User stories are broken down into programming tasks, arranged on flip chart or white-board 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

XP Phases: Summary of Key Features

Planning

- User stories are written.
- Release planning creates the release schedule.
- Make frequent small releases.
- The project is divided into iterations.
- Iteration planning starts each iteration.

Managing

- Give the team a dedicated open work space.
- Set a sustainable pace.
- A stand up meeting starts each day.
- The Project Velocity is measured.
- Move people around.

eaks.

Designing

- Simplicity.
- Choose a system metaphor.
- Use CRC cards for design sessions.
- Create spike solutions to reduce risk.
- No functionality is added early.
- Refactor whenever and wherever possible.

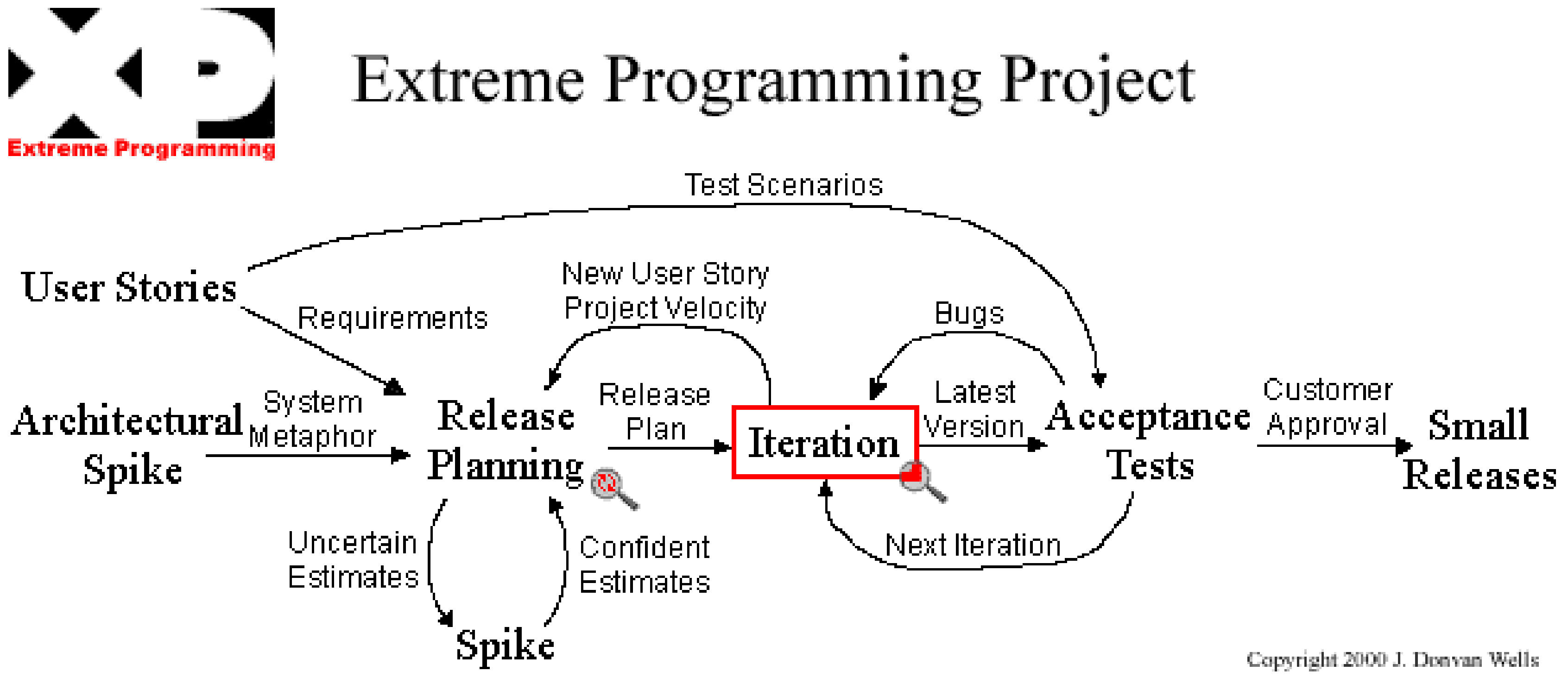
Coding

- The customer is always available.
- Code must be written to agreed standards.
- Code the unit test first.
- All production code is pair programmed.
- Only one pair integrates code at a time.
- Integrate often.
- Set up a dedicated integration computer.
- Use collective ownership.

Testing

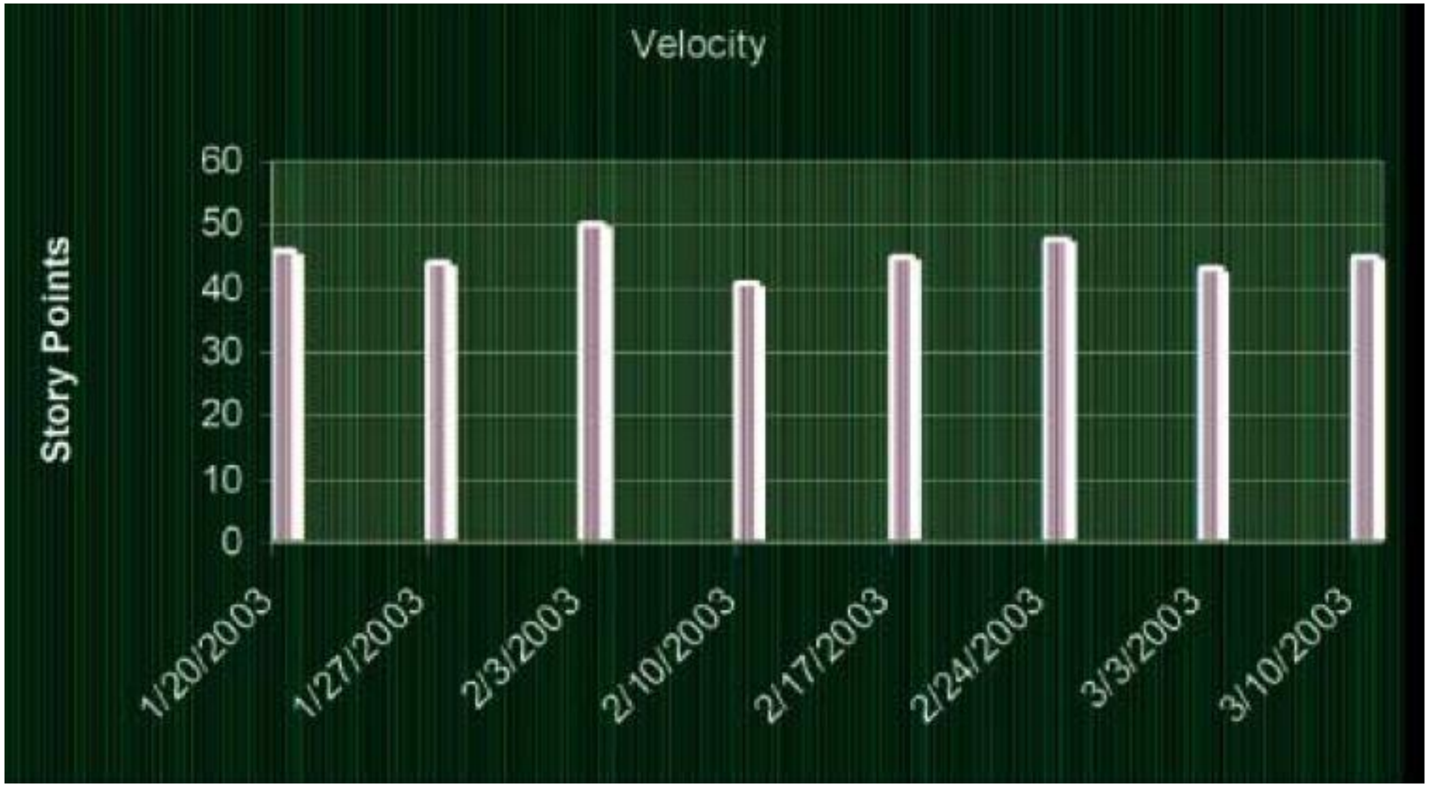
- All code must have unit tests.
- All code must pass all unit tests before it can be released.
- When a bug is found tests are created.
- Acceptance tests are run often and the score is published.

XP Project Life-Cycle Summary



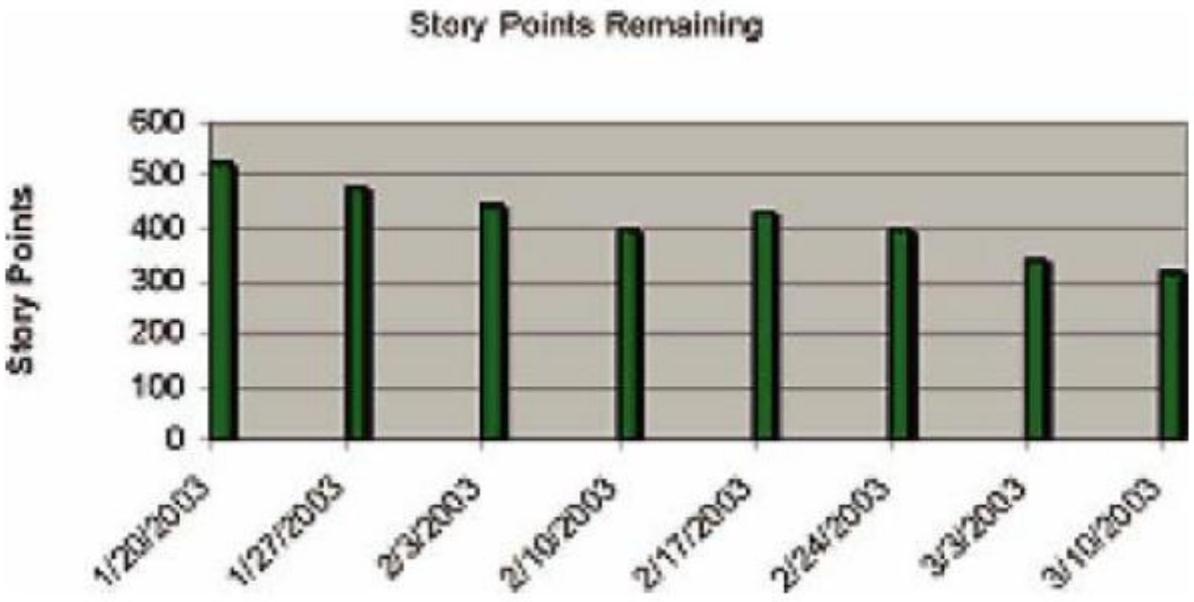
Project Tracking

- Project tracking – 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*** 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

- XP Teams also use a *burn-down* chart to show 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)



- Together, the velocity chart and the *burn-down* charts provide a reliable project management information for XP teams

When should XP be used?

- Useful for problem domains where requirements change
- When your customers do not have a firm idea of what they want or your system functionality is expected to change every few months
- 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

Useful Links

- http://www.extremeprogramming.org
- http://www.agile-process.org/change.html
- http://www.extremeprogramming.org/introduction.html
- http://www.extremeprogramming.org/values.html

Next…

Week 02 Wednesday
Requirements Engineering
The Agile Way
Aarthi Natarajan