# Agile Processes: eXtreme Programming

Data taken from
Ganesh.Sambasivam@isoftplc.comak
SDLC 3.0 book; Google;
Scattered Notes
Course Textbook

# XP's Four Values

- **Communication**. Most projects fail because of poor communication. So implement practices that force communication in a positive way.
- **Simplicity.** Develop the **simplest product** that meets the customer's needs
- **Feedback**. Developers must obtain and value feedback from the customer, from the system, and from each other.
  - The same as standard Agile values: *value customer collaboration over contract negotiation.*
- **Courage.** Be prepared to make hard decisions that support the other principles and practices.

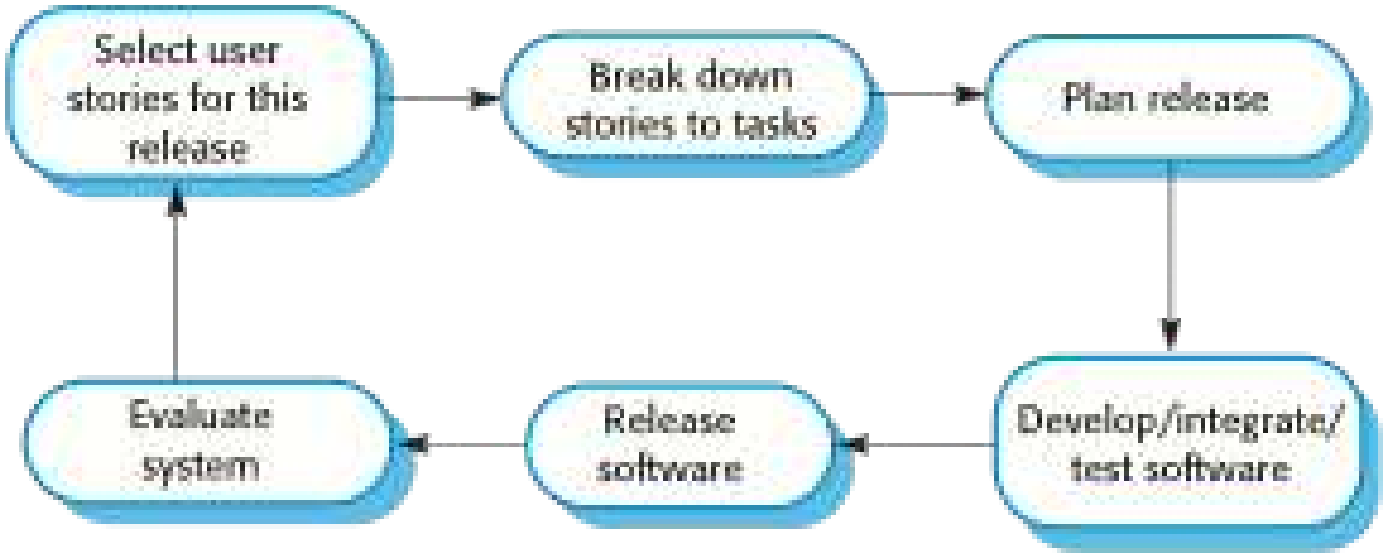
# Extreme Programming


- XP is based on these
  - four values and
  - twelve practices
  - have been extended various ways since XP's introduction
- Extreme Programming (XP) takes an 'extreme' approach to iterative development.
  - New versions may be built several times per day;
  - Increments are delivered to customers approx. every 2 weeks;
  - **All** tests must be run for every build and the build is only accepted if tests run successfully.

## XP and Agile Principles


- **Incremental development** is supported through small, frequent system releases.
- Customer involvement means **full-time** customer engagement with the team.
- People not process through **pair programming**, **collective ownership** and a **process** that avoids long working hours.
- **Change** supported through regular system releases.
- Maintaining simplicity through constant **refactoring** of code.

# The Extreme Programming Release Cycle




User Stories – coming…

## Requirements Scenarios


- In XP, a customer or user is **part** of the XP team and is responsible for making decisions on requirements.
- User requirements are expressed as **scenarios (via use cases) or user stories.**
- These (User Stories) are often written on cards and the development team break them down into **implementation tasks**.
- These tasks are the basis of schedule and cost estimates.
- The **customer** chooses the stories for inclusion in the next release based on their priorities and the schedule estimates.

# XP Fundamentals by Kent Beck

- Write unit tests **before** programming; keeping all tests running all times.
- Integrating and testing the whole system--**several times a day**.
- Producing all software in **pairs**, two programmers at one screen.
- Starting projects with **simple design.** Simple design can evolve.
- Putting a **minimal system** into production quickly and growing it in whatever directions prove most valuable.

# XP Core Practice #1- The Planning Game

- Business and development cooperate to produce **max business value** as quickly as possible.
- The planning game:
  - Business comes up with a list of desired **features**.
  - Each feature is written out as a **User Story**,
    - feature has a name, and is described in broad strokes what is required.
  - **User stories** are typically written on 4x6 cards. (You saw a variation in your book)
  - **Development estimates** how much effort each story will take, and **how much effort the team** can produce in a given time interval.
  - **Business then decides**
    - order of stories to implement,
    - And when and how often to produce a **production release** of the system.

# XP – Core Practice #2: Simple Design

- Simplest possible design to get job done.
- Requirements will change tomorrow, do what's needed to meet today's requirements

- Design in XP is not a one-time; it is an "all-the-time" activity. Have design steps in
  - release planning
  - iteration planning,
  - teams engage in quick design sessions and design revisions through refactoring,
- through the course of the entire project.

# XP – Core Practice #3: Metaphor

- Extreme Programming teams develop a common vision of how the program works, which we call the "metaphor".

- At its best, the **metaphor** is a simple evocative description of how the program works.

- XP teams use
- common system of **names** to be sure that everyone understands how the system works
- and **where to look to find the functionality** you're looking for,
- or to find the right place to **put the functionality** you're about to add.

# XP – Core Practice #4: Simple Design

- Always use the simplest possible design that gets the job done.

- The requirements will change tomorrow, so only do what's needed to meet today's requirements.

# XP – Core Practice #5: Continuous Testing

- XP teams focus on **validation** of the software at all times
- Programmers develop software by **writing tests first**, and **then code** that fulfills the requirements reflected in the tests.
- Customers provide **acceptance tests** that enable them to be **certain** that the features they need are provided.

# XP – Core Practice #6: Refactoring

- XP Team **Refactor** out any duplicate code generated in a coding session.

- Refactoring is simplified due to extensive use of **automated** test cases.

# XP – Core Practice #7: Pair Programming

- All production code is written by two programmers sitting at one machine.
  - This practice ensures that all code is reviewed as it is written and results in better Design, testing and better code.

- Some programmers object to pair programming without ever trying it.
  - It does take some practice to do well, and you need to do it well for a few weeks to see the results.
  - Ninety percent of programmers who learn pair programming prefer it, so it is recommended to all teams.
- Pairing, in addition to providing better code and tests, also serves to **communicate knowledge** throughout the team.

## Pair Programming


- In XP, programmers work in pairs, sitting together to develop code.
- This helps develop common ownership of code and spreads knowledge across the team.
- It serves as an informal review process as each line of code is looked at by more than 1 person.
- It encourages refactoring as the whole team can benefit from this.
- Measurements suggest that development productivity with pair programming is similar to that of two people working independently.

## Pair Programming


- In pair programming, programmers sit together at the same workstation to develop the software.
- Pairs are created dynamically so that all team members work with each other during the development process.
- The sharing of knowledge that happens during pair programming is very important as it reduces the overall risks to a project when team members leave.
- Pair programming is not necessarily inefficient and there is evidence that a pair working together is more efficient than 2 programmers working separately.

# Advantages of Pair Programming


- It supports the idea of collective ownership and responsibility for the system.
  - Individuals are not held responsible for problems with the code. Instead, the team has collective responsibility for resolving these problems.
- It acts as an informal review process because each line of code is looked at by at least two people.
- It helps support refactoring, which is a process of software improvement.
  - Where pair programming and collective ownership are used, others benefit immediately from the refactoring so they are likely to support the process.

# XP – Core Practice #8: Collective Code Ownership

- No single person "owns" a module.
- Any developer is expected to be able to work on any part of the codebase at any time.

# XP – Core Practice #9: Continuous Integration

- All changes are integrated into the codebase at least daily.
- Unit tests have to run 100% both before and after integration.
  - Infrequent integration leads to serious problems on a project.

- Although integration is critical to shipping good working code, the team is not practiced at it, and often it is delegated to people not familiar with the whole system.

- Problems creep in at integration time that are not detected by any of the testing that takes place on an un-integrated system.

- **Code freezes** mean that you have long time periods when the programmers could be working on important shippable features, but that those features must be held back.

# XP – Core Practice #10: 40-hour Week

- Programmers go home on time.
  - In crunch mode, up to one week of overtime is allowed.
- Multiple consecutive weeks of overtime are treated as a sign that something is very wrong with the process and/or schedule.

# XP – Core Practice #11: On-Site Customer

- Development team has **continuous access** to the customer who will actually be using the system.
- For initiatives with lots of customers, a **customer representative** (i.e. Product Manager) will be designated for Development team access.

# XP – Core Practice #12: Coding Standards

- Everyone codes to the same standards.
- The specifics of the standard are not important: what is important is that **all** of the code looks familiar, in support of **collective ownership**.

# XP on your own – Supplemental.

# XP Values – Summarized.

- XP is a values-based methodology. The **values** are Simplicity, Communication, Feedback and Courage.

- XP's core values:best summarized in the following statement by Kent Beck: **Do more of what works and do less of what doesn't**.

# Highlights of the four values itemized:

- **Simplicity** encourages:
  - Delivering the simplest functionality that meets business needs
  - Designing the simplest software that supports the needed functionality
  - Building for today and not for tomorrow
  - Writing code that is easy to read, understand, maintain and modify

# Highlights of the four values itemized:

- **Communication** is accomplished by:
  - Collaborative workspaces
  - Co-location of development and business space
  - Paired development
  - Frequently changing pair partners
  - Frequently changing assignments
  - Public status displays
  - Short standup meetings
  - Unit tests, demos and oral communication, not documentation

# Highlights of the four values itemized:

- **Feedback** is provided by:
  - Aggressive iterative and incremental releases
  - Frequent releases to end users
  - Co-location with end users
  - Automated unit tests
  - Automated functional tests
  - Courage is required to:
    - Do the right thing in the face of opposition
    - Do the practices required to succeed

# Conclusion

- Extreme Programming is not a complete template for the entire delivery organization.
- Rather, XP is a set of best practices for managing the development team and its interface to the customer.

- As a process it gives the team the ability to grow, change and adapt as they encounter different applications and business needs.
- And more than any other process we have encountered Extreme Programming has the power to **transform the entire delivery organization**, not just the development team.

# Extreme Programming Overview