IT4305: Rapid Software Development

BIT – 2nd Year

Semester 4

Learning Outcome

After successful completion of this course students will be able to:

- Obtain a firm foundation on Agile concepts and methodologies.
- Acquire understanding of the practices and application of Agile practices Scrum and XP
- Learn how to apply the Agile framework in software Development Projects

Outline of Syllabus

1. Introduction to Agile Software Development
2. Agile Principles
3. Introduction to Scrum
4. Core Concepts in Scrum
5. Scrum Roles
6. Scrum Planning
7. Sprinting
8. Alternative Approaches to Agile Software Development

References

1. Essential Scrum Practical Guide to the Most Popular Agile Process by Kenneth S. Rubin.
2. The Art of Agile Development by James Shore and Shane Warden
3. Agile and Iterative Development: A Manager's Guide by Craig Larman, Agile Software development series, Alistair Cockburn and Jim Highsmith, Series Editors
4. http://agilemanifesto.org
5. https://msdn.microsoft.com/en-us/library/hh533841.aspx

IT4305: Rapid Software Development

Extreme Programming, an agile software development process

Duration: 16 hours

Learning Outcome

- Explain the XP life cycle.
- Understands the XP team features.
- Explain pair programming and its usage.
- Understands Energized Work.
- Define Informative Workspace, Root Cause Analysis and Retrospectives.
- Describe practices that help a team and its stakeholders collaborate efficiently and effectively.
- Practice Coding standards.

Learning Outcome Cont...

- Practice Iteration demos and Reporting.
- Describe the ways that can be used to leverage the release.
- Discuss Version Controlling, Continuous integration, Collective code ownership and documentation.
- Define Release Planning and Planning Game.
- Discuss Risk Management, Iteration Planning and Slack.
- Describe Stories and Estimating.

Learning Outcome Cont...

- Describe Incremental Requirements.
- Explain and understand application of Test Driven Development.
- Understand Refactoring.
- Understand Incremental Design and Architecture.
- Understand Performance Optimization and Exploratory Testing.

Detailed Syllabus

8.1 Introduction to Extreme Programming (XP)

8.2 XP Practices

8.3 Collaboration in XP

8.4 Product Releasing in XP

8.5 Planning Process in XP

8.6 Product Development in XP

8.1: INTRODUCTION TO EXTREME PROGRAMMING XP

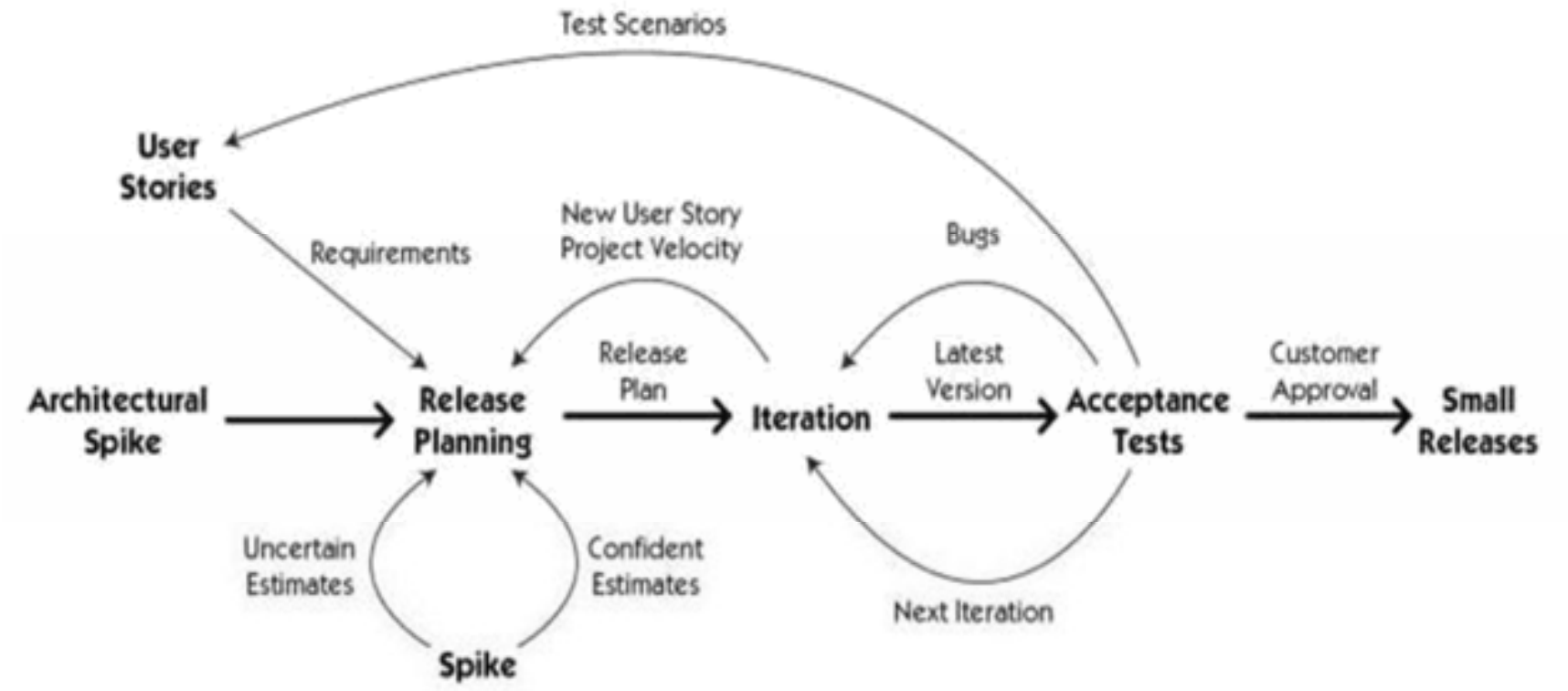
8.1: Introduction to Extreme Programming XP

- 8.1.1 Introduction
- 8.1.2 Introduction to XP Lifecycle
- 8.1.3 The XP Team
  - 8.1.3.1 The Whole Team
  - 8.1.3.2 On-Site Customers
  - 8.1.3.3 Programmers
  - 8.1.3.4 Testers
  - 8.1.3.5 Coaches
  - 8.1.3.6 Other Team Members
  - 8.1.3.7 The Project Community
  - 8.1.3.8 Filling Roles
  - 8.1.3.9 Team Size

8.1.1 Introduction

- Extreme Programming eliminates
  - analysis, design, and testing phases, along with their associated documentation.
- XP teams perform
  - significant analysis, design, testing, and coding every day
- High-bandwidth communication, cross-functional teams, and practices tuned for iterative and incremental work.
- Short, timeboxed iterations provide structure, and the team produces potentially shippable software at the end of each iteration.
- Each iteration starts with a brief planning session and ends with a product demo and retrospective.

8.1.1 Introduction

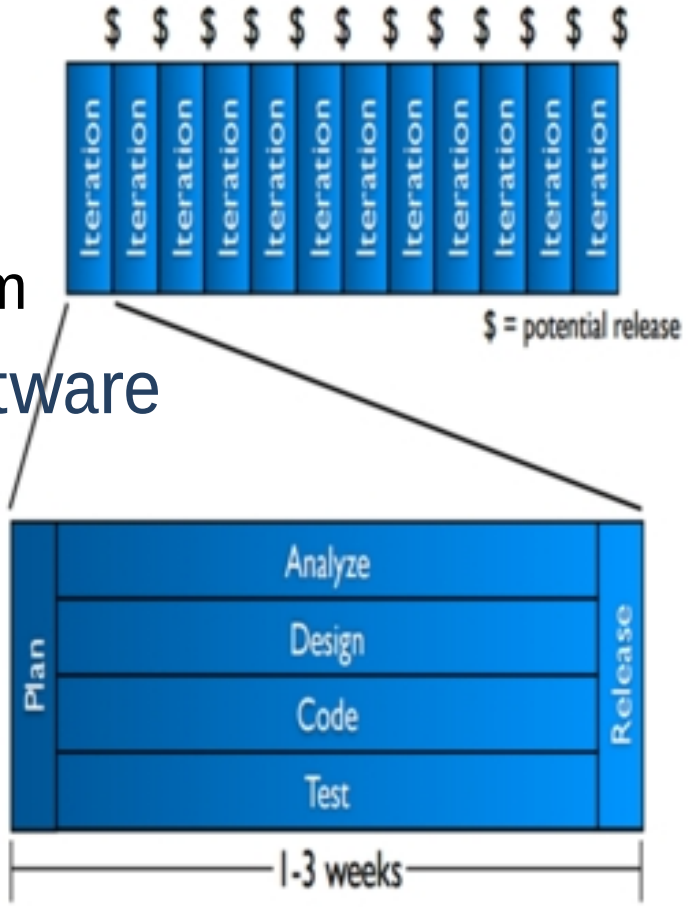


A visual process model for XP

Refer to Ref.2 pg.15-17 for a scenario

8.1.2 Introduction to XP Lifecycle

- can eliminate
  - Requirements
  - Design
  - Testing phases
  - Formal documents that go with them
- XP team produces deployable software every week
- XP does it by working in iterations: week-long incrementsof work.



Refer to Ref:2 Pg.18-21

8.1.2 Introduction to XP Lifecycle

- Every week, the team does a bit of
  - release planning
  - Design
  - Coding
  - Testing
  - They work on stories:
    - very small features
    - or parts of features, that

8.1.2 Introduction to XP Lifecycle Cont..

- Every week, the team commits to delivering four to ten stories.
- Throughout the week, they work on all phases of development for each story.
- At the end of the week, they deploy their software for internal review.

8.1.2 Introduction to XP Lifecycle - Planning

- Every XP team includes several business experts
- on-site customers
  - who are responsible for making business decisions.
  - point the project in the right direction by clarifying the project vision, creating stories, constructing a release plan, and managing risks.
- Programmers
  - provide estimates and suggestions, which are blended with customer priorities in a process called the planning game.

8.1.2 Introduction to XP Lifecycle - Planning Cont...

- customers
  - continue to review and improve the vision and the release plan to account for new opportunities and unexpected events.
- The team
  - creates a detailed plan for the upcoming week at the beginning of each iteration.
  - touches base every day in a brief stand-up meeting,
  - its informative workspace keeps everyone informed about the project status.

8.1.2 Introduction to XP Lifecycle - Analysis

- on-site customers
  - Rather than using an upfront analysis phase to define requirements sit with the team full-time.
  - may or may not be real customers depending on the type of project
  - but they are the people best qualified to determine what the software should do.
  - are responsible for figuring out the requirements for the software.
  - When programmers need information, they simply ask. Customers are responsible for organizing their work so they are ready when programmers ask for information.
  - They figure out the general requirements for a story before the programmers estimate it
  - formalize tricky or difficult to understand requirements with the assistance of testers, by creating customer tests:

8.1.2 Introduction to XP Lifecycle - Analysis Cont...

- Customers and testers
  - create the customer tests for a story around the same time that programmers implement the story.
- For the UI,
  - customers work with the team to create sketches of the application screens.
  - Some teams include an interaction designer who's responsible for the application's UI.

8.1.2 Introduction to XP Lifecycle - Design and Coding

- XP uses incremental design and architecture to continuously create and improve the design in
- small steps.
- test-driven development (TDD),
  - an activity that inextricably weaves together testing, coding, design, and architecture.
  - To support this process, programmers work in pairs,
  - ensures that one person in each pair always has time to think about larger design issues.
- use a version control system for configuration management and maintain their own automated build.

8.1.2 Introduction to XP Lifecycle - Design and Coding Cont...

- Programmers integrate their code every few hours and ensure that every integration is technically capable of deployment.
- programmers also maintain coding standards and share ownership of the code.
- The team shares a joint aesthetic for the code.
- Everyone is expected to fix problems in the code regardless of who wrote it.

8.1.2 Introduction to XP Lifecycle - Testing

- Each member of the team—programmers, customers, and testers—makes his own contribution to software quality.
- produce a handful of bugs per month in completed work.
- TDD produces automated unit and integration tests.
- Customers review work in progress to ensure that the UI works the way they expect.
- They also produce examples for programmers to automate that provide examples of tricky business rules.

8.1.2 Introduction to XP Lifecycle - Testing Cont...

- When the testers find a bug, the team conducts root-cause analysis and considers how to improve their process to prevent similar bugs from occurring in the future.
- Testers explore the nonfunctional characteristics, such as performance and stability.
- When bugs are found, programmers create automated tests to show that the bugs have been resolved.
- The team supports their quality efforts through pair programming, energized work, and iteration slack.

8.1.2 Introduction to XP Lifecycle - Deployment

- XP teams keep their software ready to deploy at the end of any iteration.
- They deploy the software to internal stakeholders every week in preparation for the weekly iteration demo.
- Deployment to real customers is scheduled according to business needs.
- when the project ends, the team may hand off maintenance duties to another team.
- In this case, the team creates documentation and provides training as necessary during its last few weeks.

8.1.3 The XP Team

- Team software development
  - How to design and program the software (programmers, designers, and architects)
  - Why the software is important (product manager)
  - The rules the software should follow (domain experts)
  - How the software should behave (interaction designers)
  - How the user interface should look (graphic designers)
  - Where defects are likely to hide (testers)
  - How to interact with the rest of the company (project manager)
  - Where to improve work habits (coach)
- XP acknowledges this reality by creating cross functional teams composed of diverse people

8.1.3 The XP Team - The Whole Team

- XP teams sit together in an open workspace.
- At the beginning of each iteration, the team meets for a series of activities:
  - an iteration demo
  - a retrospective
  - iteration planning.
- These meeting typically take two to four hours in total.
- The team also meets for daily stand-up meetings, which usually take five to ten minutes each.
- Everyone on the team plans his own work : it's as informal as somebody standing up and announcing across the shared workspace that he would like to discuss an issue.
- This self-organization is a hallmark of agile teams.

Refer to Ref.2 pg.28

8.1.3 The XP Team – On-Site Customers

- responsible for defining the software the team builds.
- most important activity is release planning.
- Customers need to evangelize the project's vision;
- identify features and stories determine how to group features into small, frequent releases; manage risks; create an achievable plan by coordinating with programmers and playing the planning game.
- responsible for refining their plans by soliciting feedback from real customers and other stakeholders.
  - One of the venues for this feedback is the weekly iteration demo

8.1.3 The XP Team – On-Site Customers Cont...

- responsible for providing programmers with requirements details upon request.
- Customers themselves act as living requirements documents, researching information in time for programmer use and providing it as needed.
  - XP uses requirements documents only as memory aids for customers.
- Help communicate requirements by creating mock-ups, reviewing work in progress, Creating detailed customer tests that clarify complex business rules.

8.1.3 The XP Team – On-Site Customers Cont...

- Typically, product managers, domain experts, interaction designers, and business analysts play the role of the on-site customer.
- One of the most difficult aspects of creating a cross-functional team is finding people qualified and willing to be on-site customers.
- A great team will produce technically excellent software without on-site customers
- Customer involvement makes a huge difference in product success

NOTE: Include exactly one product manager and enough other on-site customers for them to stay one step ahead of the programmers. As a rule of thumb, start with two on-site customers (including the product manager) for every three programmers.

8.1.3 The XP Team- Programmers

- Responsible for finding the most effective way of delivering the stories in the plan.
- Provide effort estimates, suggest alternatives, and help customers create an achievable plan by playing the planning game.
- Spend most of their time in pair programming.
- Using test-driven development, they write tests, implement code, refactor, and incrementally design and architect the application.
- Have to have the awareness of technical debt and its impact on development time and future maintenance costs.

8.1.3 The XP Team- Programmers Cont...

- Maintain a ten-minute build that can build a complete release package at any time
- Use version control and practice continuous integration, keeping all but the last few hours' work integrated and passing its tests
- At the beginning of the project establish coding standards
- Have the right and the responsibility to fix any problem they see

Refer to Ref.2 pg.33

8.1.3 The XP Team - Testers

- Help customers identify holes in the requirements and assist in customer testing.
- Use exploratory testing to help the team identify whether it is successfully preventing bugs from reaching finished code.
- Provide information about the software's nonfunctional characteristics, such as performance, scalability, and stability, by using both exploratory testing and long-running automated tests.

8.1.3 The XP Team – Testers Cont..

- When testers find bugs, they help the rest of the team figure out what went wrong so that the team as a whole can prevent those kinds of bugs from occurring in the future.
- require creative thinking, flexibility, and experience defining test plans.
- XP automates repetitive testing rather than performing manual regression testing

Refer to Ref.2 pg.34

8.1.3 The XP Team - Coaches

- XP leaders lead by example, helping the team reach its potential rather than creating jobs and assigning tasks.
- XP leaders are called coaches.
- Leadership roles dynamically switch from person to person as situations dictate.
- Help the team start their process by arranging for a shared workspace and making sure that the team includes the right people.
- They help set up conditions for energized work, and they assist the team in creating an informative workspace.

8.1.3 The XP Team – Coaches Cont...

- Work, and they assist the team in creating an informative workspace.
- Help the team interact with the rest of the organization.
- Take responsibility for any reporting needed.
- Help the team members maintain their self-discipline, helping them remain in control of challenging practices such as risk management, test-driven development, slack, and incremental design and architecture.

8.1.3 The XP Team - Other Team Members

- **Technical Writer**
- **Analyst**
- **Product Manager**
  - maintain and promote the product vision
  - sharing it with stakeholders, incorporating feedback, generating features and stories, setting priorities for release planning, providing direction for the team's on-site customers, reviewing work in progress, leading iteration demos, involving real customers, and dealing with organizational politics.

8.1.3 The XP Team - Other Team Members Cont...

- **Domain Experts**
  - spend most of their time with the team, figuring out the details of upcoming stories and standing ready to answer questions when programmers ask.
  - For complex rules, they create customer tests to help convey nuances.

8.1.3 The XP Team

- **Interaction Designers**
  - focuses on understanding users, their needs, and how they will interact with the product.
  - They interview users, create user personas, review paper prototypes with users, and observing usage of actual software.

    Graphic designers: convey ideas and moods via images and layout.
    Interaction designers: focus on the types of people using the product, their needs, and how the product can most seamlessly meet those needs
- **Business Analysts**
  - clarify and refine customer needs,
  - Help customers think of details they might otherwise forget and help programmers express technical trade-offs in business terms.

Refer to Ref.2 pg.32-37

8.1.3 The XP Team Cont...

- **Designers and architects**
  - Everybody codes and designs on an XP team.
  - Guide the incremental design and architecture efforts and by helping team members see ways of simplifying complex designs.
  - act as peers—guiding rather than dictating.
- **Technical specialists**
  - XP programmers are generalizing specialists. Although each person has his own area of expertise
  - everybody is expected to work on any part of the system that needs attention.

8.1.3 The XP Team Cont...

- **The programmer-coach**
  - helps the other programmers with XP's technical practices.
  - often senior developers and may have titles such as "technical lead" or "architect."
  - programmer-coaches also act as normal programmers and participate fully in software development.
- **The project manager**
  - usually good at coaching non-programming practices.
  - Some functional managers fit into this role as well.
  - lack the technical expertise to coach XP's programming practices
  - Project managers may also double as customers.

Refer to Ref.2 pg.32-37

8.1.3 The XP Team-The Project Community

- Organization's Human Resources and Facilities departments
- Human Resources : handles performance reviews and compensation. Their mechanisms may not be compatible with XP's team-based effort
- In order to use XP, you'll need the help of Facilities to create an open workspace

8.1.3 The XP Team - Filling Roles

- You don't have to have one person for each role—some people can fill multiple roles.
  - At a minimum, however, one person clearly designated as "product manager" (who may do other customer-y things) and one person clearly defined as "programmer-coach" (who also does programmer-y things).
  - The other roles may blend together. Product managers are usually domain experts and can often fill the project manager's shoes, too.
  - One of the customers may be able to play the role of interaction designer, possibly with the help of a UI programmer.
  - On the programming side, many programmers are generalists and understand a variety of technologies. In the absence of testers, both programmers and customers should pick up the slack.

8.1.3 The XP Team - Team Size

- Assume teams with 4 to 10 programmers (5 to 20 total team members).
- For new teams, four to six programmers is a good starting point.
- 6 programmers produces a team that also includes 4 customers, 1 tester, and a project manager, for a total team size of 12 people.
- Twelve people turns out to be a natural limit for team collaboration.
- XP teams can be as small as one experienced programmer and one product manager,
- The smallest team with full XP consists of five people: four programmers (one acting as coach)and one product manager (who also acts as project manager, domain expert, and tester).
- Starting with 10 programmers produces a 20-person team that includes 6 customers, 3 testers, and a project manager.
- You can create even larger XP teams, but they require special practices

Refer to Ref.2 pg.39

8.2: XP PRACTICES

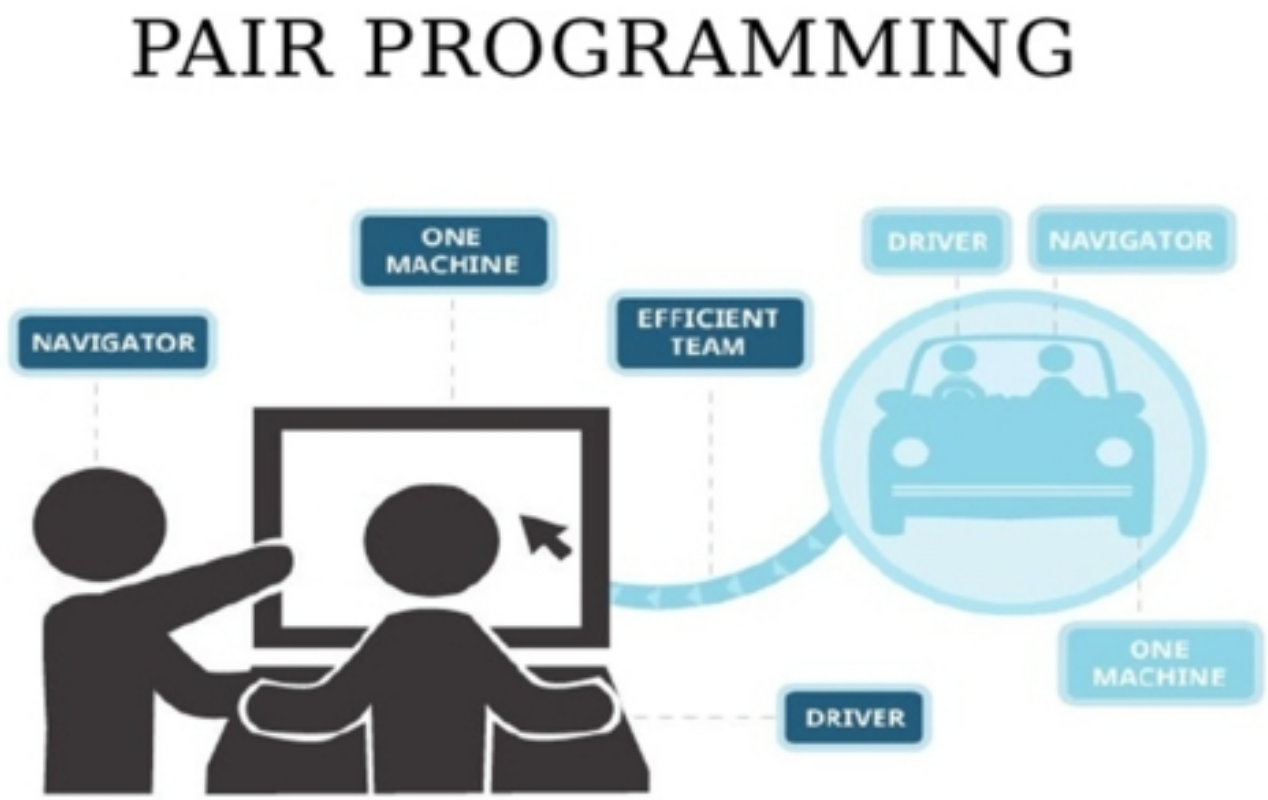
8.2 XP Practices

- 8.2.1 Introduction
- 8.2.2 Pair Programming
- 8.2.3 Energized Work
- 8.2.4 Informative Workspace
- 8.2.5 Root-Cause Analysis
- 8.2.6 Retrospectives

8.2.1 Introduction

- XP doesn't require experts. It does require a habit of mindfulness.
- Five practices to help mindful developers excel:
  - Pair programming doubles the brainpower available during coding, and gives one person in each pair the opportunity to think about strategic, long-term issues.
  - Energized work acknowledges that developers do their best, most productive work when they're energized and motivated.
  - An informative workspace gives the whole team more opportunities to notice what's working well and what isn't.
  - Root-cause analysis is a useful tool for identifying the underlying causes of your problems.
  - Retrospectives provide a way to analyze and improve the entire development process.

8.2.2 Pair Programming



8.2.2 Pair Programming Cont...

- When you pair, one person codes—*the driver*.
- The other person is *the navigator*, whose job is to think.
- You'll spread coding knowledge and tips throughout the team.
- No interrupts from others when working with a partner

When you start working on a task, ask another programmer to work with you. If another programmer asks for help, make yourself available. Never assign partners: pairs are fluid, forming naturally and shifting throughout the day. Over time, pair with everyone on the team. This will improve team cohesion and spread design skills and knowledge throughout the team.

8.2.2 Pair Programming Cont...

- It s a good idea to switch partners several times per day even if you don't feel stuck.
- When you sit down to pair together, make sure you're physically comfortable.
- The discussions may enlighten your partner as much as it does you.
- As you pair, switch roles frequently.

8.2.2 Pair Programming Cont...

- PAIRING TIPS
  - Pair on everything you'll need to maintain.
  - Allow pairs to form fluidly rather than assigning partners.
  - Switch partners when you need a fresh perspective.
  - Avoid pairing with the same person for more than a day at a time.
  - Sit comfortably, side by side.
  - Produce code through conversation. Collaborate, don't critique.
  - Switch driver and navigator roles frequently.

8.2.2 Pair Programming Cont...

- Read following sub topics from Ref :2 (pg. 77 to 78)
  - Paring Stations
  - Challenges
  - Comfort
  - Mismatched Skills
  - Communication Style
  - Tools and Key Binding
- Read Ref :2 (pg. 81) for
  - Alternatives

8.2.3 Energized Work

- Work only as many hours as you can be productive and only as many hours you can sustain.
  - Tired developers make more mistakes, which slows you down more in the long run (remove value from product).
  - If you mess with people's personal lives (by taking it over), in the long run the project will pay the consequences.

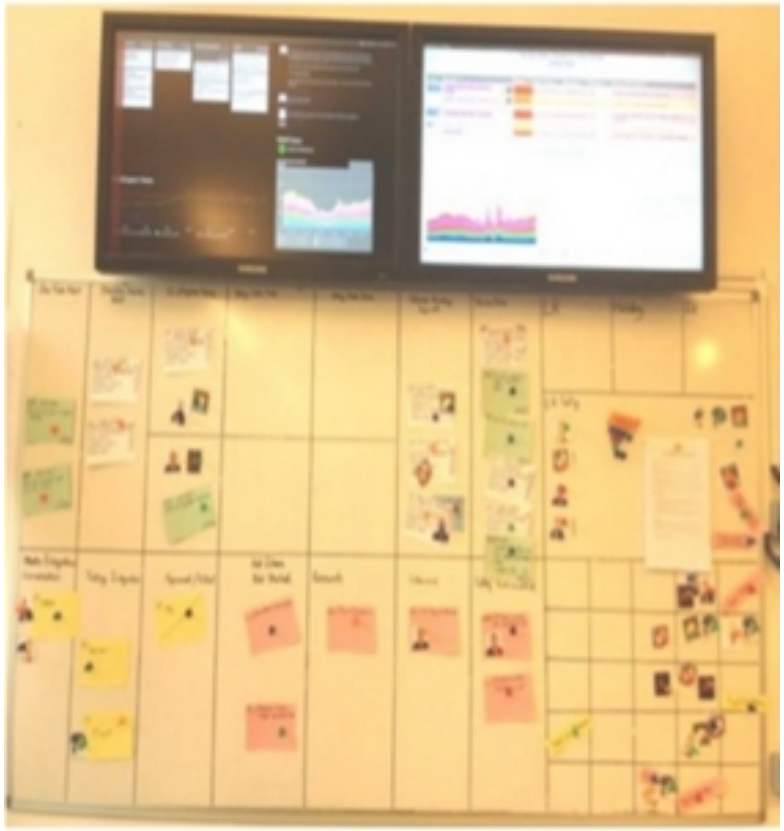
8.2.4 Informative Workspace

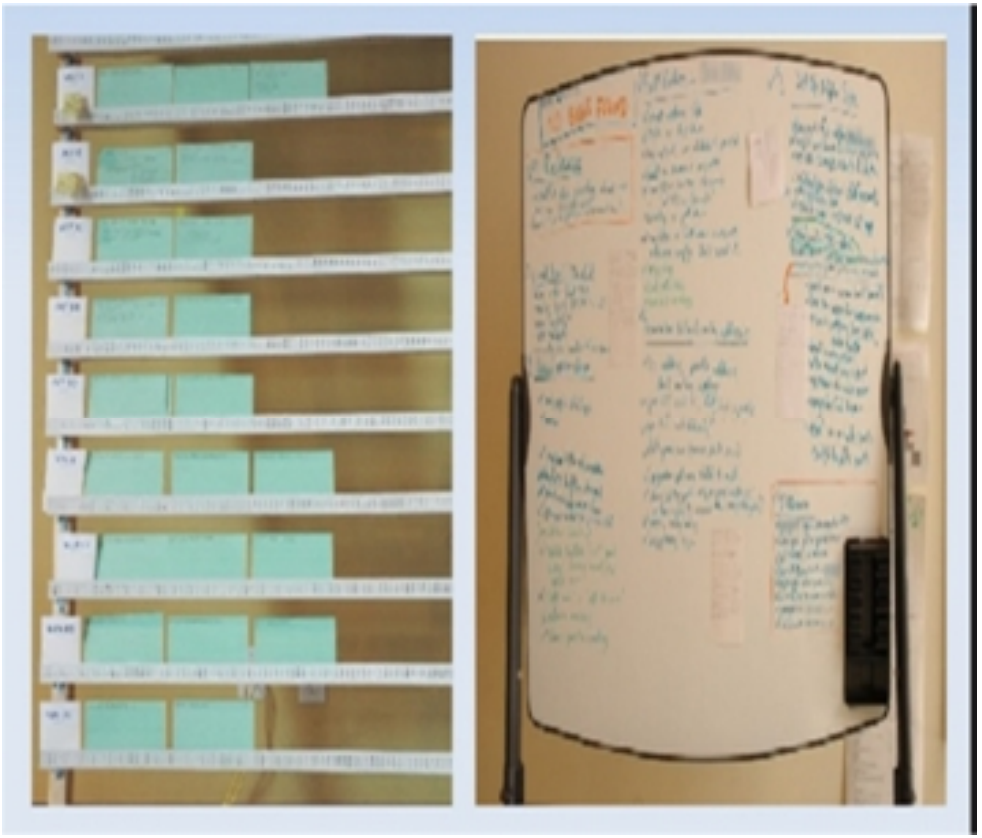
- An informative workspace broadcasts information into the room
- An informative workspace also allows people to sense the state of the project just by walking into the room.
- The agile team project room often includes the story board with
  - user story cards
  - movable from *not started* to *in progress* to *done* column
  - release and/or iteration burn down charts
  - the automated indicators showing the status of the latest unit-testing run

8.2.4 Informative Workspace Cont...

- Many teams find it useful to conduct the daily standups meeting in front of their story board.
- It is also worth noting that even if the team is not co-located .
- It is still likely to find it useful to have a dedicated meeting.
- room with the big visible charts, whiteboards and flipcharts.

8.2.4 Informative Workspace Cont...





8.2.4 Informative Workspace Cont...

- Read about the following Charts on Ref 2. pg 87-88
  - Big Visible Charts
  - Hand drawn Charts
  - Process Improvement Charts

8.2.5 Root-Cause Analysis

- "We prevent mistakes by fixing our process."
- Technique for identifying and eliminating process faults
  - First developed in the nuclear power industry
  - used in many fields.

– classic approach to root-cause analysis is to ask "why" five times.
  - Why? Because the build is often broken in source control.
  - Why? Because people check in code without running their tests.
  - Why don't they run tests before checking in? Because sometimes the tests take longer to run than people have available.
  - Why do the tests take so long? Because tests spend a lot of time in database setup and teardown.
  - Why? Because our design makes it difficult to test business logic without touching the database.

8.2.6 Retrospectives

- "We continually improve our work habits."
- **Types of Retrospectives**
  - the iteration retrospective(most common retrospective) : occurs at the end of every iteration.
- Intensive retrospectives at crucial milestones.
  - release retrospectives
  - project retrospectives
  - surprise retrospectives (conducted when an unexpected event changes your situation)

Refer to Ref.2 pg.94-100

8.2.6 Retrospectives Cont...

- **How to Conduct an Iteration Retrospective**
  - Everyone on the team should participate
  - The process
    - **Step 1: The Prime Directive**
    - **Step 2: Brainstorming**
    - **Step 3: Mute Mapping**
    - **Step 4: Retrospective Objective**

  Ref 2: pg 95 – 97
- **Retrospective**
  - sharing ideas gives the team a chance to grow closer,
  - coming up with a specific solution gives the team a chance to improve.

8.3: COLLABORATION IN XP

8.3 Collaboration in XP

- 8.3.1 Introduction
- 8.3.2 Trust
- 8.3.3 Sit Together
- 8.3.4 Real Customer Involvement
  - 8.3.4.1 Contraindications
  - 8.3.4.2 Alternatives
- 8.3.5 Ubiquitous Learning
- 8.3.6 Stand-up Meetings
- 8.3.7 Coding Standards
- 8.3.8 Iteration Demo
- 8.3.9 Reporting

8.3.1 Collaboration in XP - Introduction

- 8 practices to help your team and its stakeholders collaborate efficiently and effectively:
  - Trust
  - Sitting together
  - Real customer involvement
  - A ubiquitous language
  - Stand-up meetings
  - Coding standards
  - Iteration demos
  - Reporting

8.3.2 Collaboration in XP - Trust

- a series of group dynamics
  - "Forming, Storming, Norming, and Performing"
- Trust is essential for the team to perform well.
- need to trust that taking time to help others won't make you look unproductive.
- need to trust that you'll be treated with respect when you ask for help or disagree with someone.
- The organization needs to trust the team

Refer to Ref.2 pg.103

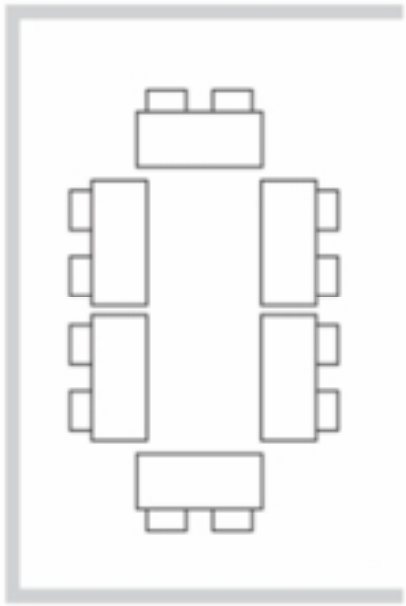
- strategies for generating trust in your XP team.
  - **Team Strategies**
    - **Customer-Programmer Empathy**
    - **Programmer-Tester Empathy**
    - **Eat Together**
    - **Team Continuity**
  - **Organizational Strategies**
    - **Show Some Hustle**
    - **Deliver on Commitments**
    - **Manage Problems**
    - **Respect Customer Goals**
    - **Promote the Team**
    - **Be Honest**

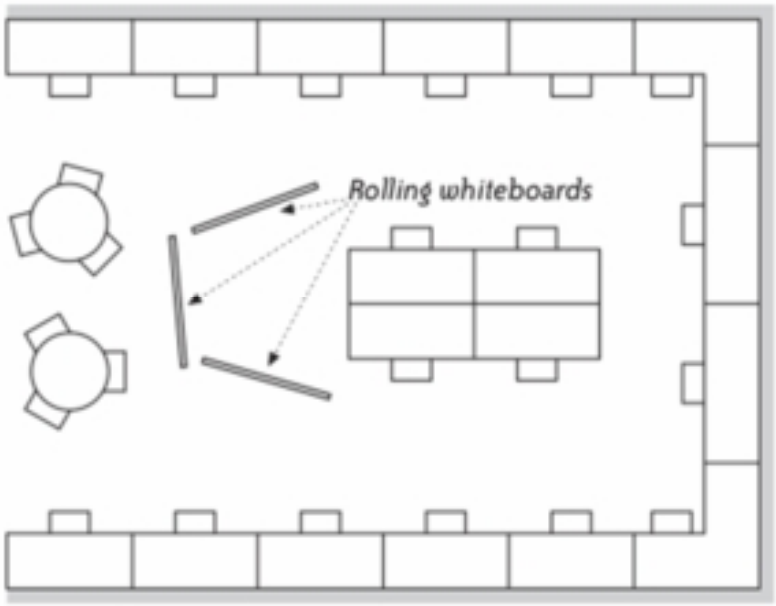
8.3.3 Collaboration in XP – Sit Together

- Why?
  - **Accommodating Poor Communication**
  - **A Better Way**
    - you need only turn your head and ask. You get an instant response
- Exploiting Great Communication
  - eliminates waste caused by waiting for an answer
- **Secrets of Sitting Together**
  - **Encourage interruptions**

Refer to Ref 2: pg 113-117

- A sample workspace





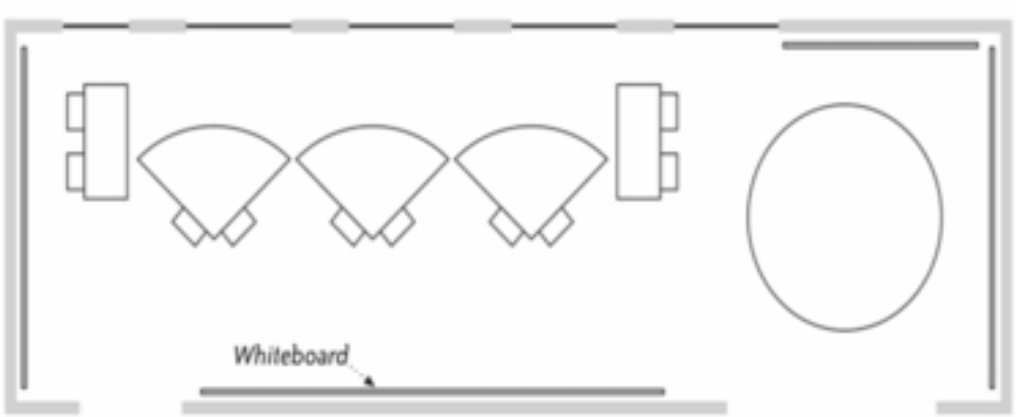
pairing stations,

a series of cubbies for personal effects.

Nonprogrammers worked in cubbies close to the pairing stations

Refer to Ref 2: pg 118-119

- A small workspace



a table for meetings

charts and whiteboards on dividers

a pod of half-cubicles small conference rooms

Refer to Ref 2: pg 118-119

8.3.4 Collaboration in XP – Real Customer Involvement

- Customer involvement is a key part of XP where the customer is part of the development team.
- The role of the customer is:
  - To help develop stories that define the requirements
  - To help prioritize the features to be implemented in each release
  - To help develop acceptance tests which assess whether or not the system meets its requirements.

8.3.4.1 Contraindications

- Danger of involving real customers is
  - they won't necessarily reflect the needs of all your customers.
  - they don't steer you toward creating software that's only useful for them.
  - Customer desires inform the vision and may even change it, but ultimately the product manager holds final responsibility for product direction.
  - End-users often think in terms of improving their existing way of working
- If innovation is important to your project,
  - give innovative thinkers such as a visionary product manager or interaction designer, prominent role on your team.

8.3.4.2 Alternatives

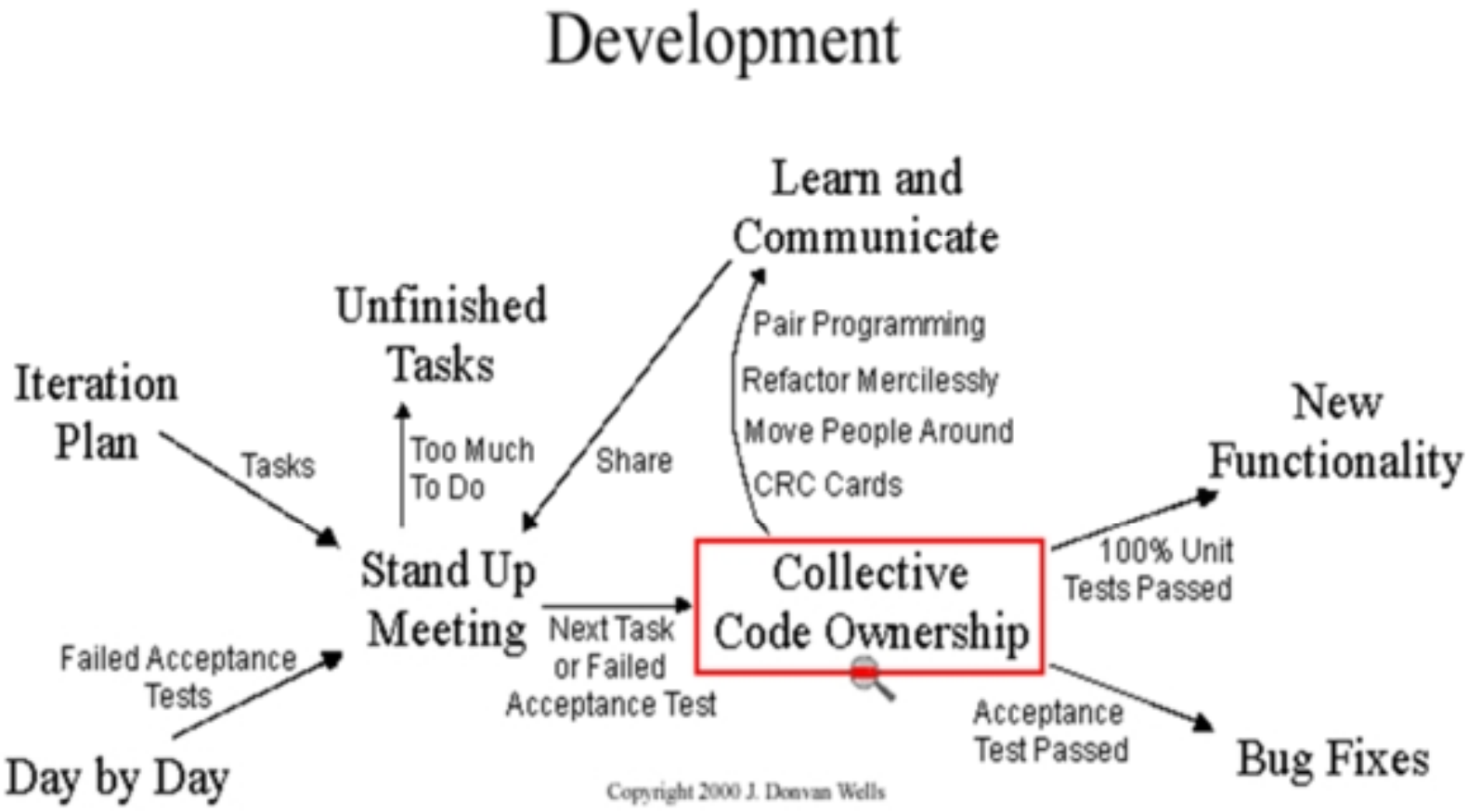
- Real customer involvement is not crucial.
- In the absence of real customer involvement, be sure to have a visionary product manager
- feedback from real customers is always informative
- It's especially useful when you've deployed software to them; their reaction to working software gives you valuable information about how likely you are to reach the greatest levels of success.

8.3.5 Collaboration in XP – Ubiquitous Language

- Try describing the business logic in your current system to a nonprogrammer domain expert.
  - design your code to use the language of the domain
    - name your classes, methods, and variables *anything*.
      - programmers aren't necessarily experts in the areas for which they write software.
      - The people who are experts in the problem domain are rarely qualified to write software.

8.3.6 Collaboration in XP – Stand-up Meetings

- Goal: Identify items to be accomplished for the day and raise issues
- Everyone attends,
  - including the customer
- Not a discussion forum
- Take discussions offline
- Everyone gets to speak 15 minutes


Development
Learn and Communicate
Unfinished Tasks
Pair Programming
Refactor Mercilessly
Move People Around
CRC Cards
Iteration Plan
Tasks
Too Much To Do
Share
New Functionality
Stand Up Meeting
Collective Code Ownership
100% Unit Tests Passed
Failed Acceptance Tests
Next Task or Failed Acceptance Test
Acceptance Test Passed
Bug Fixes
Day by Day
Copyright 2000 J. Donvan Wells

8.3.7 Collaboration in XP – Coding Standards

- Code must be formatted to agreed coding standards.
- Coding standards keep the code consistent and easy for the entire team to read and refactor.
- Code that looks the same encourages collective ownership

8.3.7 Collaboration in XP – Iteration Demo

- Produce working software every week,
- demonstrate to stakeholders, Invite anyone who's interested to the demo.
- take about ten minutes.
- The product manager often conducts the session.
  - describe the features scheduled, their value, and any unexpected changes. Build trust by being honest, not defensive, about changes.
  - At the end of each demo, ask your executive sponsor two questions: "Is our work to date satisfactory?" and "May we continue?“
  - Conduct demos at the same place and time each week. When schedule problems occur, a regular demo makes it easier to face reality.

8.3.7 Collaboration in XP – Reporting

- A vision statement,
  - weekly product demos, release and iteration plans, and a burn-up chart are a normal byproduct of your work. Share them as a matter of course.
- Other reports take extra time.
  - They're technically waste, but may be necessary to help build trust in your team.
- Roadmap and status emails summarize your release plan and demos.
- Productivity, throughput, and defect reports help management understand your effectiveness.

8.3.7 Collaboration in XP – Reporting Cont...

- Time usage reports help explain your velocity.
- Avoid reporting lines of code, numbers of stories, and velocity. They're misleading at best.
- Avoid sharing code quality metrics, too: they require expert interpretation.

8.4: PRODUCT RELEASING IN XP

8.4: Product Releasing in XP

8.4.1 “Done Done”

8.4.2 No Bugs

8.4.3 Version Control

8.4.4 Ten Minute Build

8.4.5 Continuous Integration

8.4.6 Collective Code Ownership

8.4.7 Documentation

8.4.1 Product Releasing in XP – "Done Done"

- A story is only complete when on-site customers can use it as they intended.
- completed stories should be
  - Tested , Coded , Designed, Integrated , Builds , Installs, Migrates , Reviewed, Fixed

  Accepted (customers agree that the story is finished)
  - i.e. The build script builds, installs, and migrates data for the story.
  - Bugs have been identified and fixed and customers have reviewed the story and agree it's complete.

8.4.1 Product Releasing in XP – "Done Done" Cont...

- To achieve this result, make progress on everything each day.
- Use test-driven development to combine testing, coding, and design.
- Keep the build up to date and integrate continuously. Demonstrate progress to your customers and incorporate their feedback as you go.

8.4.2 Product Releasing in XP-No Bugs

- Rather than fixing bugs, agile methods strive to prevent them.
  - Test-driven development structures work into easily-verifiable steps.
  - Pair programming provides instant peer review, enhances brainpower, and maintains self-discipline.
  - Energized work reduces silly mistakes. Coding standards and a "done done" checklist catch common errors.
  - On-site customers clarify requirements and discover misunderstandings.
  - Customer tests communicate complicated domain rules. Iteration demos allow stakeholders to correct the team's course.

8.4.2 Product Releasing in XP-No Bugs Cont...

- Simple design, refactoring, slack, collective code ownership, and fixing bugs early eliminates bug breeding grounds.

- Exploratory testing discovers teams' blind spots, and root-cause analysis allows teams to eliminate them.

8.4.3 Product Releasing in XP-Version Control

- To support collective ownership, use a concurrent model of version control.
  - Support time travel by storing tools, libraries, documentation, and everything else related to the project in version control.
  - Keep the entire project in a single repository.
  - Avoid long-lived branches, particularly for customized versions; they'll cripple your ability to deliver on a timely schedule.
    - Instead, use configuration files and build scripts to support multiple configurations.
- Keep your repository clean: never check in broken code. All versions should build and pass all tests.
- "Iteration" versions are ready for stakeholders;
- "release" versions are production-ready.

8.4.4 Product Releasing in XP-Ten-Minute Build

- Build, test, and deploy your entire product at any time with the push of a button.
- Your build should be comprehensive but not complex.
  - Make it compile source code, run tests, configure registry settings, initialize database schemas, set up web servers, launch processes, build installers, and deploy.
  - Your IDE won't do all this, so learn to use a dedicated build tool.
  - Make sure your build works when disconnected from the network, too.
- Builds should be fast. If not, look at your tests.
- End-to-end integration tests are the typical culprit. Replace them with faster, more maintainable unit tests.

8.4.5 Product Releasing in XP- Continuous Integration

- Keep code integrated and build release infrastructure with the rest of the application.
  - The ultimate goal is to be able to deploy all but the last few hours of work at any time.
- Integrate every few hours and keep your build, test, and other release infrastructure up to date.
  - Each integration should get as close to a real release as possible.
- Prefer synchronous integration, in which you wait for the integration to succeed, to asynchronous integration, in which a tool tests the integration for you.
  - Synchronous integration requires fast builds, but ensures that they never break.

8.4.6 Product Releasing in XP- Collective Code Ownership

- With collective code ownership, everyone shares responsibility for code quality.
- Anyone can make necessary changes anywhere, and everyone is expected to fix problems they find.
- Take as much pride in the *team's* code as in your *own* code.
- When working with unfamiliar code, ask the local expert to pair with you.
  - If he's unavailable, infer high-level class responsibilities and method behaviors from their names, and rely on unit tests for further documentation and as your safety net.
- As you work, help the next person by taking opportunities to refactor.

8.4.6 Product Releasing in XP-Documentation

- Projects use three main types of documents:
  - work-in-progress; product; hand-off.
- Work-in-progress documents
  - communicate, and other forms of communication may replace them. High-bandwidth communication replaces some of these documents
  - Test-driven development creates executable low-level design specifications. Customer tests describe high-level behavior, and a ubiquitous language further clarifies intent.

8.4.6 Product Releasing in XP- Documentation Cont...

- Product documents have business value
  - schedule them with stories.
- Handoff documents
  - Best and most accurate at the end of the project.
  - Set aside time after delivery to create them
  - consider conducting an incremental handoff using pair programming and collective ownership.

8.5: PLANNING PROCESS IN XP

8.5: Planning Process in XP

8.5.1 Introduction

8.5.2 Vision

8.5.3 Release Planning

8.5.4 Planning Game

8.5.5 Risk Management

8.5.6 Iteration Planning

8.5.7 Slack

8.5.8 Stories

8.5.9 Estimating

8.5.1 Planning Process in XP- Introduction

- ***Vision*** reveals where the project is going and why it's going there.
- ***Release Planning*** provides a roadmap for reaching your destination.
- ***The Planning Game*** combines the expertise of the whole team to create achievable plans.
- ***Risk Management*** allows the team to make and meet long-term commitments.
- ***Iteration Planning*** provides structure to the team's daily activities.
- ***Slack allows*** the team to reliably deliver results every iteration.
- ***Stories*** form the line items in the team's plan.
- ***Estimating enables*** the team to predict how long their work will take.
- 0

8.5.2 Planning Process in XP-Vision

- Every project needs a single vision
  - the product manager must unify, communicate, and promote that vision.
- Distance between visionaries and the product manager increases error and waste.
  - If you only have one visionary, the best approach is for him to be product manager. Alternatively, use the visionary's protogé.
- Some projects have multiple visionaries. They need to combine their ideas into a unified vision. The product manager facilitates discussion and consensus.
- Document the vision with *what success is*, *why it's important*, and *how you know* you've achieved it. Post it prominently and involve your visionaries in planning and demos.

8.5.3 Planning Process in XP-Release Planning

- Maximize your return on investment by:
  - working on one project at a time;
  - releasing early and often;
  - adapting your plans;
  - keeping your options open; and
  - planning at the last responsible moment.
- Use timeboxing to control your schedule. Set the release date, then manage scope to meet that date. This forces important prioritization decisions and makes the endpoint clear.

8.5.3 Planning Process in XP-Release Planning Cont...

- Prioritized Minimum Marketable Features (MMFs) and stories form the body of your plan. Demonstrate your progress as you develop and use that feedback to revise your plan.
- To minimize rework, develop the details of your requirements at the last responsible moment.

8.5.4 Planning Process in XP-The Planning Game

- Customers have the most information about *value*:
  - what best serves the organization.
  - Programmers have the most information about *cost*: what it takes to implement and maintain those features
  - every decision to *do* something is a decision to *not do* something else.
- The planning game brings together customers and programmers so that they may maximize value while minimizing costs.

8.5.4 Planning Process in XP-The Planning Game Cont...

- Anybody may create stories. Programmers estimate the stories, and customers prioritize them. Programmers and customers may question each others' decisions, but each group has final say over its area of expertise.
- The end result is a single prioritized list.

8.5.5 Planning Process in XP- Risk Management

- Risk management allows you to make and meet commitments.
- Manage common risks with risk multipliers.
  - Manage project-specific risks by brainstorming disasters and their causes.
  - Create mitigation and contingency plans for serious risks and define unambiguous triggers for taking action.

8.5.5 Planning Process in XP- Risk Management Cont...

- Combine risk multipliers with project-specific risks to project your chances of meeting commitments, and report those probabilities as "commitments" and "stretch goals."
  - Lower your risk and improve projections by including slack in every iteration and getting stories "done done".
- Remember that success is more than meeting commitments. Sometimes it's better to delay delivery and create a better product.

8.5.5 Planning Process in XP- Iteration Planning

- Iterations are timeboxed to one week and follow a strict schedule:
  - Plan iteration
  - Commit to delivering stories
  - Develop stories
  - Prepare release
  - Demonstrate release
  - Hold retrospective
- To plan, measure the velocity of the previous iteration (total the estimates of "done done" stories). Select stories from the release plan that match the velocity. It shouldn't take long.

8.5.5 Planning Process in XP- Iteration Planning Cont...

- Assuming programmers are your constraint, they brainstorm and estimate engineering tasks. Ask the on-site customer about detailed requirements when necessary. Compare the task estimates to last iteration's to confirm the plan's feasibility.

- Post the stories and tasks prominently and mark them when complete.

8.5.5 Planning Process in XP- Risk Management

- Risk management allows you to make and meet commitments.
- Manage common risks with risk multipliers.
  - Manage project-specific risks by brainstorming disasters and their causes.
  - Create mitigation and contingency plans for serious risks and define unambiguous triggers for taking action.
- Combine risk multipliers with project-specific risks to project your chances of meeting commitments, and report those probabilities as "commitments" and "stretch goals."
  - Lower your risk and improve projections by including slack in every iteration and getting stories "done done".

8.5.5 Planning Process in XP- Risk Management Cont...

- Remember that success is more than meeting commitments. Sometimes it's better to delay delivery and create a better product.

8.5.6 Planning Process in XP- Slack

- *deliver on our iteration commitments.*
- project plans are also too important to be disrupted by the slightest provocation.
- schedule no work on the last day or two of your iteration.
- useful, important work that isn't time-critical—work you can set aside in case of an emergency.
  - useful, important work that isn't time-critical—work you can set aside in case of an emergency. *Paying down technical debt* fits the bill perfectly.
- Each refactoring should address a specific, relatively small problem. Sometimes you'll fix only part of a larger problem—that's okay as long as it makes the code better.

8.5.7 Planning Process in XP- Stories

- Stories may be the most misunderstood entity in all of XP. They're not requirements. They're not use cases.
- *Stories* are for planning
- A story is a placeholder for a detailed discussion about requirements.
- two important characteristics
  - Stories represent *customer value* and are written in the customers' terminology. They describe an end-result that the customer values, not implementation details.
  - Stories have clear *completion criteria*. Customers can describe an objective test that would allow programmers to tell when they've successfully implemented the story.

8.5.7 Planning Process in XP- Stories Cont...

- Story Cards : Write stories on index cards.
- Customer-Centricity :Write them from the on-site customers' point of view
- Splitting and Combining Stories : Split large stories; combine small ones.
- Special Stories : Documentation Stories, "Non-Functional" Stories, Bug Stories, Spike Stories

8.5.7 Planning Process in XP- Estimating

- *provide reliable estimates.*
- consider estimating to be a black art

With no history, the first plan is the hardest and least accurate (fortunately, you only have to do it once)

How to start estimating:

- Begin with the stories that you feel the most comfortable estimating.
- Intuitively imagine how long it will take you.
- Base other estimates on the comparison with those first stories.

Spike Solutions:

- Do a quick implementation of the whole story.
- Do not look for the perfect solution!
- Just try to find out how long something takes

Keys to effective story estimation:

- Keep it simple
- Use what happened in the past ("Yesterday's weather")
- Learn from experience

Comparative story estimation:

- One story is often an *elaboration* of a closely related one
- Look for stories that have *already* been implemented
- Compare *difficulties*, not implementation time
  - "twice as difficult", "half as difficult"
- *Discuss* estimates in the team. Try to find an agreement.
- *"Optimism wins":* Choose the more optimistic of two disagreeing estimates.

8.6: PRODUCT DEVELOPMENT IN XP

8.6: Product Development in XP

8.6.1 Introduction

8.6.2 Incremental Requirements

8.6.3 Customer Tests

8.6.4 Test-Driven Development

8.6.5 Refactoring

8.6.6 Simple Design

8.6.7 Incremental Design and Architecture

8.6.8 Spike Solutions

8.6.9 Performance Optimization

8.6.1 Product Development in XP - Introduction

- ***Incremental Requirements*** allow the team to get started while customers work out requirements details.
- ***Customer Tests*** help communicate tricky domain rules.
- ***Test-Driven Development*** allows programmers to be confident that their code does what they think it should.
- ***Refactoring*** enables programmers to improve code quality without changing its behavior.
- ***Simple Design*** allows the design to change to support any feature request, no matter how surprising.

8.6.1 Product Development in XP – Introduction Cont...

- ***Incremental Design and Architecture*** allows programmers to work on features in parallel with technical infrastructure.
- ***Spike Solutions*** use controlled experiments to provide information.
- ***Performance Optimization*** uses hard data to drive optimization efforts.
- ***Exploratory Testing*** enables testers to identify gaps in the team's thought processes.

8.6.2 Product Development in XP - Incremental Requirements

- *We define requirements in parallel with other work.*
- XP doesn't have a requirements phase
- The Living Requirements Document
  - the on-site customers sit with the team. They're expected to have all of the information about requirements at their fingertips.
- Work Incrementally
  - Work on requirements *incrementally*, in parallel with the rest of the team's work.
- Vision, Features, and Stories
  - Start by clarifying your project vision, then identify features and stories as described in **8.5**
    - Rough Expectations, Mock-Ups, Customer Tests, and Completion Criteria, Customer Review

- Many changes will be minor and the programmers will be able to fix them as part of their iteration slack.
- If there are major changes, however, the programmers may not have time to fix them in the current iteration.
  - Create story cards for these changes. Before scheduling such a story into your release plan, consider whether the value of the change is worth its cost.

8.6.3 Product Development in XP – Customer Tests

- make sure that the programmers understand the domain rules well enough to code them properly in the application.*Customer tests* help customers communicate their expertise.
- Describe
  - At the beginning of the iteration, look at your stories and decide whether there are any aspects that programmers might misunderstand.
- Demonstrate
  - Tables are often the most natural way to describe this information, but you don't need to worry about formatting. Just get the examples on the whiteboard.

8.6.3 Product Development in XP – Customer Tests Cont...

- Develop
  - When you've covered enough ground, document your discussion so the programmers can start working on implementing your rules.
- One of the most common mistakes in creating customer tests is describing what happens in the user interface rather than providing examples of business rules.

8.6.4 Product Development in XP – Test Driven Development

- *produce well-designed, well-tested, and well-factored code in small, verifiable steps.*
- *TDD*, is a rapid cycle of testing, coding, and refactoring.
- When adding a feature, a pair may perform dozens of these cycles, implementing and refining the software in baby steps until there is nothing left to add and nothing left to take away.
- When used properly, it also helps improve your design, documents your public interfaces, and guards against future mistakes.

8.6.4 Product Development in XP – Test Driven Development Cont...

- TDD isn't perfect, of course. TDD is difficult to use on legacy codebases.
- Go through
  - Why TDD Works, How to Use TDD
  - Unit Tests
  - Focused Integration Tests
  - End-to-End Tests
  - TDD and Legacy Code

8.6.5 Product Development in XP - Refactoring

- the process of changing the design of your code without changing its behavior
- also reversible; sometimes one form is better than another for certain cases.
- Go thro'
  - Reflective Design
  - Analyzing Existing Code
  - How to Refactor
  - Refactoring in Action

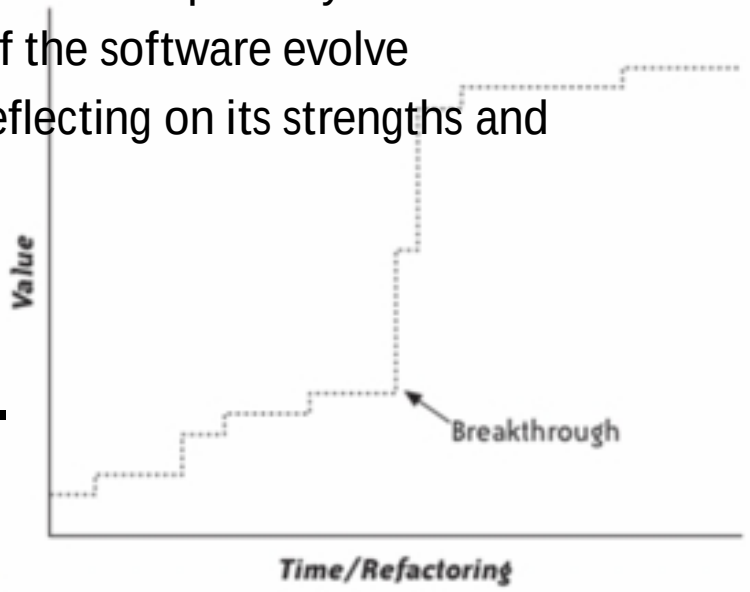
8.6.6 Product Development in XP – Simple Design

- *easy to modify and maintain Design.*
- A simple design is clean and elegant, not something you throw together with the least thought possible.
  - some points to keep in mind as you strive for simplicity: (Go thro' them)
    - You Aren't Gonna Need It (YAGNI)
    - Once and Only Once
    - Self-Documenting Code
    - Isolate Third-Party Components
    - Limit Published Interfaces
    - Fail Fast

8.6.7 Product Development in XP - Incremental Design and Architecture

- every week, programmers should finish four to ten customer-valued stories.
- Every week, customers may revise the current plan and introduce entirely new stories—with no advance notice. This regimen starts with the first week of the project.
- How it works
    - start by creating the simplest design that could possibly work
    - incrementally add to it as the needs of the software evolve
    - continuously improve the design by reflecting on its strengths and weaknesses.
- Continuous Design
  - Breakthroughs happen at all levels

of the design, from methods to architectures.



Refer to Ref 2: pg 323-333 122

8.6.8 Product Development in XP – Spike Solutions

- a technical investigation
- It's a small experiment to research the answer to a problem.
- usually to create a small program or test that demonstrates the feature in question.
- Scheduling
  - are performed on the spur of the moment. You see a need to clarify a small technical issue, and you write a quick spike to do so.
  - are performed on the spur of the moment. You see a need to clarify a small technical issue, and you write a quick spike to do so.

8.6.9 Product Development in XP – Performance Optimization

- Measure the performance of the entire system, make an educated guess about what to change, then re-measure. If the performance gets better, keep the change. If it doesn't, discard it. Once your performance test passes, stop—you're done.

- How to Optimize
  - run your test suite one more time. Then integrate. If you're adding new code, such as a cache, use test-driven development to create that code. If you're removing or refactoring code, you may not need any new tests, but be sure to run your test suite after each change.

8.6.9 Product Development in XP – Performance Optimization cont...

- When to Optimize
  - two major drawbacks : often leads to complex, buggy code, and it takes time away from delivering features

- If your tests start to take too long, go ahead and optimize until you meet a concrete goal, such as five or ten minutes. Keep in mind that the most common cause of a slow build is too much emphasis on end-to-end tests, not slow code.