# SEM 2:
# Agile and Extreme

Dr. Julie Greensmith

# Overview

- (Very) Briefly introduce the concepts of Agile Design and Extreme Programming
- Also briefly discuss some of the other Agile methods
- Agile Design is a design framework
  - Extreme Programming is one way to "implement" agile design

# Waterfalls and Spirals

- The methodologies we have looked at previously have been heavy top-down approaches
  - inflexible structure
  - rigid in terms of the stages performed
- Many problems with these approaches
  - tons of documentation
  - lots of waiting time for developers
  - can result in costly errors and buggy code

# Surely there has to be another way?

- Too much top-down leads to heavy handed micromanagement
  - surely the developers know best how to develop?
- No control or methodology leads to problems too
  - generation of spaghetti code
- *Is it not possible to generate code in a unified yet flexible manner?*
- Agile methods are a potential solution

# The Agile Manifesto

- This manifesto comes from the *Agile Alliance*
- "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
- While value is placed on the right hand items, the left hand items are viewed as of higher importance

# Five main principles

- The agile manifesto actually has 12 main points, but these can be condensed to the following five features:

1. Deliver Early and Often to Satisfy Customer
2. Welcome Changing Requirements
3. Face to Face Communication is Best
4. Measure Progress against Working Software
5. Simplicity is Essential

# Deliver Often = Happy Customers

- MIT Sloan Management Review published an analysis of software development practices in 2001
- Strong correlation found between quality of software system and the early delivery of a partially functioning system
  - The less functional the initial delivery the higher the quality of the final delivery!
  - Strong correlation between final quality of software system and frequent deliveries of increasing functionality: the more frequent the deliveries, the higher the final quality!
- Customers may choose to put initial/intermediate systems into production use; or they may simply review functionality and provide feedback

This is why you are developing a quick prototype for your coursework

# Embrace change

- Be flexible with changing requirements
  - Keep the customer informed of the development and changes can be made before they become too costly!
- Welcome changes even if they are late in the project
- "Developers in agile projects are not afraid of change; changes are good since it means our understanding of the target domain has increased"
- Plus, agile development practices (such as refactoring) produce systems that are flexible and easy to change

# Embrace change

# Face to Face is Best

- In an agile project, people talk to each other!
- The primary mode of communication is conversation
  - there is no attempt to capture all project information in writing
- Some “nimble documents” are still created but only if there is an immediate and significant need that they satisfy organisational policy
  - Nimble documents may be discarded, after the need has passed

# ...no matter how strange the client!

# Measure Progress

- Agile projects measure progress by the amount of software that is currently meeting customer needs
- They are 30% done when 30% of required functionality is working AND deployed
- Progress is not measured in terms of phases or creating documents
- Keeps on top of customer satisfaction and allows for more realistic and updated estimation of costs

# Keep it Simple

- This refers to the art of maximizing the amount of work *NOT* done
- Agile projects always take the simplest path consistent with their current goals
- They do not try to anticipate tomorrow's problems; *they only solve today's problems*
- High-quality work today should provide a simple and flexible system that will be easy to change tomorrow if the need arises

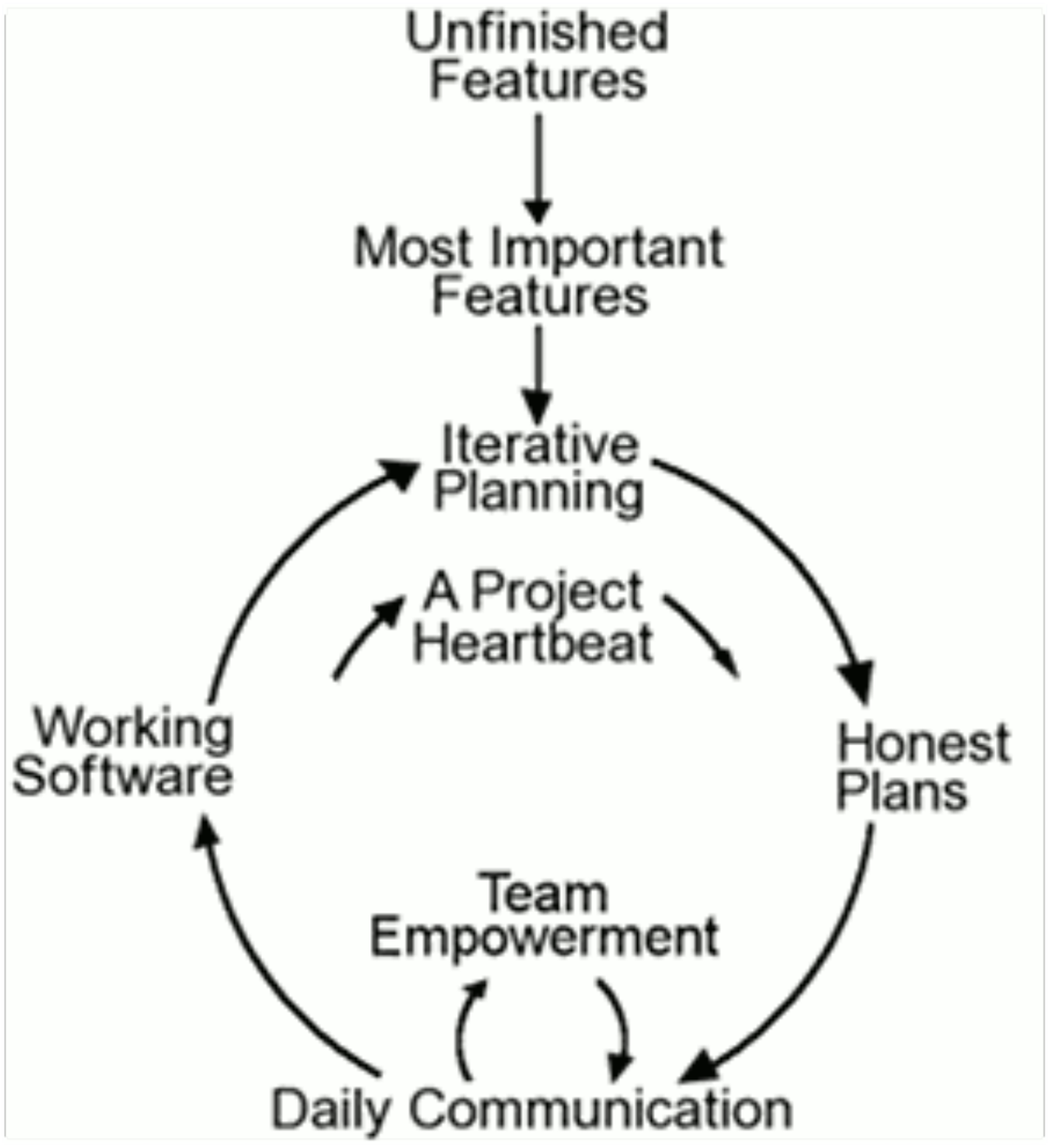
# Thats a bit abstract!

- The agile ethos is a bit abstract from the process of actually applying the principles in a sound methodology
  - different people interpret the manifesto in different ways
  - two popular aproaches are Extreme Programming and Agile Scrums, and Test Driven Development (TDD)

# Extreme Programming (XP)

- An application of agile principles to a working methodology
- Give the basic principles of XP and good practices
- Show how you can use this to help you complete the coursework!


LES
ALPES360C


Unfinished Features
Most Important Features
Iterative Planning
A Project Heartbeat
Working Software
Honest Plans
Team Empowerment
Daily Communication

# The Customer is in the Team

- The "customer" is made a member of the development team
- A customer representative should be "in the same room" or at most 100 feet away from the developers
  - seriously, the customer actually 'lives' with the developers during the working day, not joking
- "Release early; Release Often" delivers a working system to the customer so reassured about investment
- Customer representative provides continuous feedback to the developers

# User stories

- Instead of a long requirements gathering process construct stakeholder *stories*
- We need just enough detail to estimate how long it might take to develop software to support this story
- Avoid too much detail, since the requirement will most likely change
  - start at a high level, deliver working functionality and iterate based on explicit feedback

# Do you remember the syntax for a user story?

# User stories

- User stories are not documented in detail
  - we work out the scenario with the customer “face-to-face”
- We give this scenario a name
- The name is written on an index card or post-it
- Developers then write an estimate on the card based on the detail they got during their conversation with the customer
- The index card becomes a “token” which is then used to drive the implementation of a requirement based on its priority and estimated cost

# Testing from your stories

- As with system testing from requirements, you can start to write your test cases based on the functions and actions specified in the user stories
- Testing in agile and XP is at its core.
- If you dont pass the tests, features are not ‘signed off’ until they are 100% test compliant
- Unit tests are written before the code is implemented - the stories are the plan for this

# Simplicity in Design

- Encouraged to implement the simplest design possible
- Glue simple bits of code together
- “Do the simplest thing that could possibly work, then make it elegant or optimal later” - refactor
- Don't build a giant super-efficient object, sorted and hashed and linked together, if an Array will do the job!!
- Makes it easier to test and produces code which is more understandable

# Pairs Programming

- All production code is written by pairs of programmers working together at the same workstation on the same code
  - One member drives the keyboard and writes code and test cases; the second watches the code, looking for errors and improvements
- The roles will switch between the two frequently
  - this facilitates distribution of knowledge about the state of the code throughout the entire team
- Pairs programming does not degrade efficiency of a team, yet it significantly reduces the defect rate!

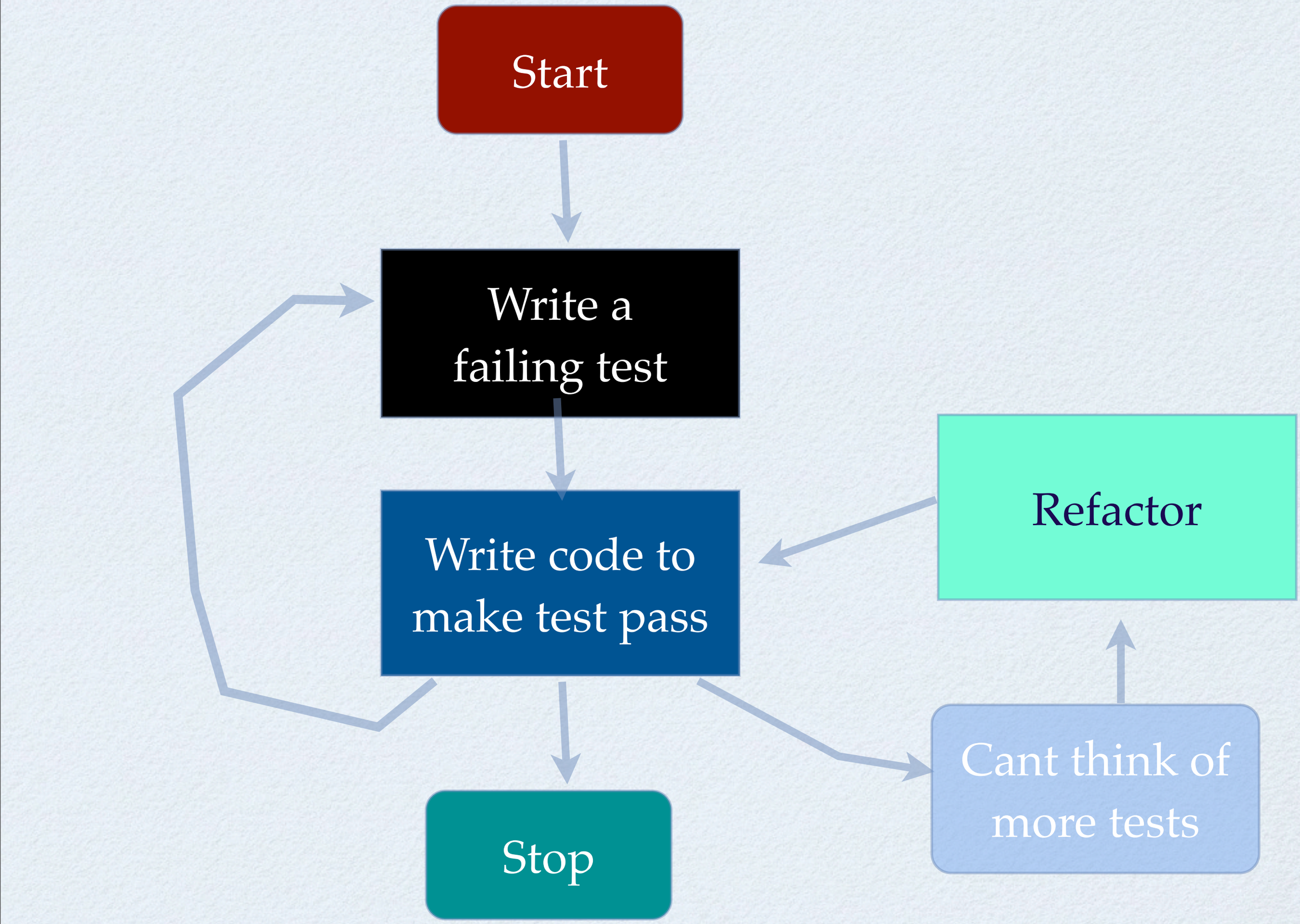

i.justrealized.com

# Using XP

- To do this coursework well as a team use proper XP inspired pairs programming
  - I promise it will speed up your development
  - avoid version control issues in development
  - neither one of you have done all the coding (or all the writing)
  - [and if you follow the next couple of points too it will help]

# Test Driven Development

- All production code is written in order to make failing test cases pass
- First, we write a test case that fails since the required functionality has not yet been implemented
- Then, we write the code that makes that test case pass
- Iteration between writing tests and writing code is very short (minutes) As a result, a very complete set of test cases is written for the system
  - not developed after the fact, integral to the process


Start
Write a failing test
Write code to make test pass
Refactor
Stop
Cant think of more tests

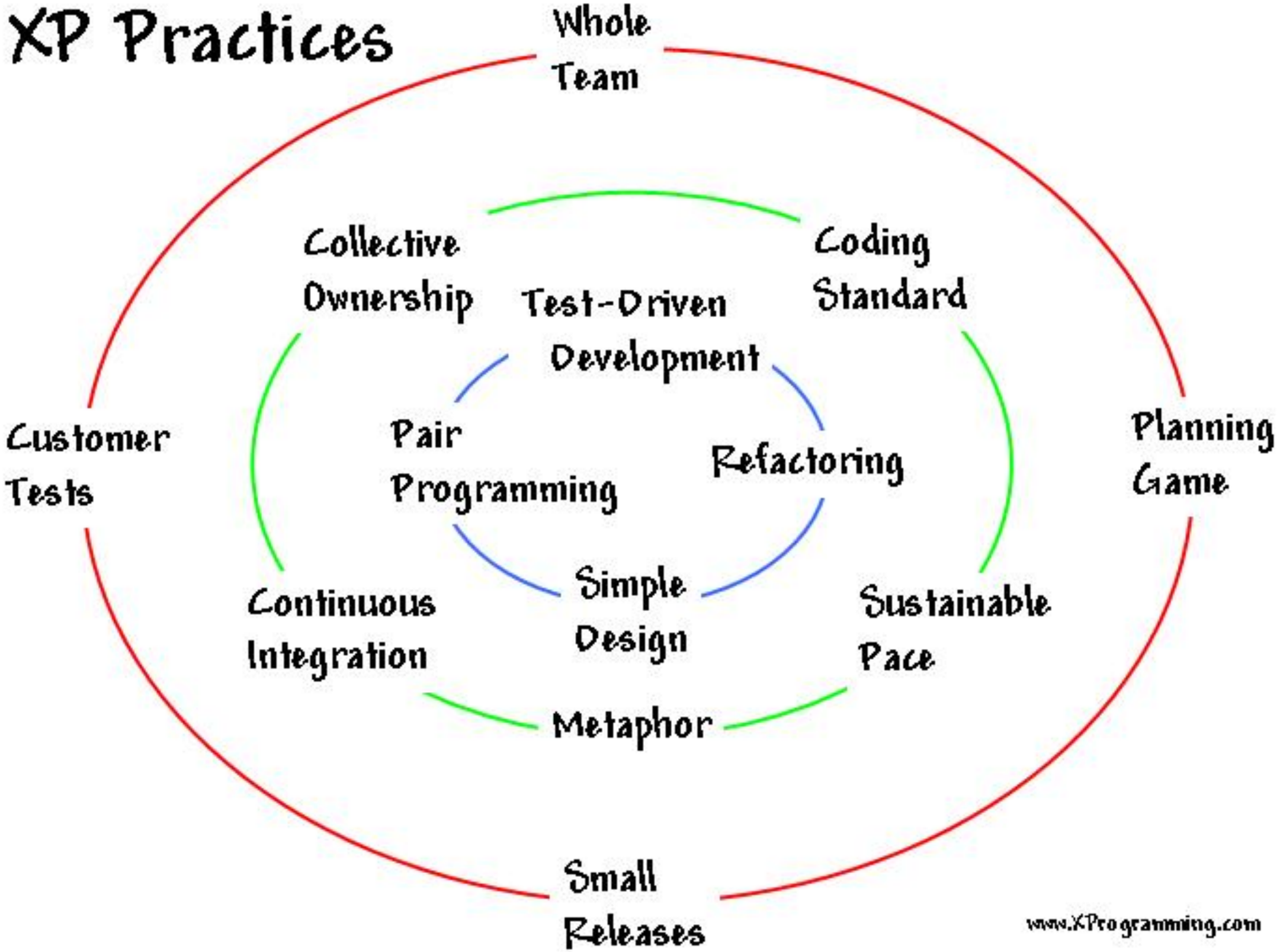
# Lightweight Documentation

- You write the code so that you do not need to keep massive documents for everything
  - classes and variable names should be self explanatory
  - have a “nimble document”
- Keep any documentation to a minimum and make sure the client is involved in the creation of the documentation

# And because you're a team...

- Easy to have good communication as you are all placed in a small team in the same physical environment
  - 3-10 people in one XP team
- Fewer mistakes are made as you can have better trust with your co-workers
- Reduces stress and allows you to produce a better performance

# XP Practices



# *If Extreme Programming and Agile methods are so good then why doesnt everyone do it?*

- These are now becoming the predominant methodologies in software engineering
- Even monolithic companies are using it
  - realising the efficiency advantages
  - due to the recession software teams are a lot smaller
  - customers are expecting "on-demand" results, don't have the time or money to wait a year before they see a product

- I have just done a very cursory look at agile and xp
- Have a look at the following links:
  - http://www.extremeprogramming.org/
  - http://www.agile-process.org/
- Most importantly, *give it a go!* You will be better for it!!!

# Summary

- Introduced the concepts of Agile Design and Extreme Programming
- Agile Manifesto: major components
- Agile Design is a design framework
  - Extreme Programming is one way to “implement” agile design