CISC/CMPE 327 Software Quality Assurance

Queen's University, 2019–fall

Lecture #5

Agile development - XP

Agile Development

- A group of software development methods
  - Early and continuous delivery of software
  - Welcome changing requirements, even late in development
  - Business people and developers must work together
  - Working software is the primary measure of progress
  - Self-organizing teams produce the best architectures, requirements, and designs
  - Reflect and tune behaviour at regular intervals to become more effective

Agile Development Values

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

- Although there is value in the items on the right, agile software developers value the items on the left more

http://agilemanifesto.org

eXtreme Programming

- A Modern, Lightweight Software Process
  - Extreme Programming, or XP, is a modern lightweight process suitable for small to medium-sized software projects
  - Designed to adapt well to the observed realities of modern software production
    - short timelines
    - high expectations
    - severe competition
    - unclear and rapidly changing requirements

eXtreme Programming

- A Modern, Lightweight Software Process
  - Based on the idea of continuous evolution
  - Very practical, based largely on simplicity, testing
  - In spite of its brash, undisciplined, "fun" presentation, solidly based on the software disciplines and processes of the past

What's So eXtreme About It?

- Why is it called Extreme?
  - When first conceived, the idea was to take the best practices of good software development to the limit
    - if code reviews are good, review code all the time
    - if testing is good, test all the time
    - if design is good, design all the time
    - if simplicity is good, always use the simplest solution possible
    - if architecture is important, refine architecture all the time
    - if integration is important, integrate all the time
    - if short iterations are good, use shortest iterations possible
  - Clearly this can only work for relatively small projects

Great, Another Process...

- Why make a different approach?
  - XP was born from the dissatisfaction of programmers with the actual situation in most software development environments
  - Frustration with the lack of time to test adequately because of the rush to get new software and new versions out quickly

Great, Another Process...

- Why make a different approach?
  - Dissatisfaction with the lack of ongoing advice and social support for difficult technical decisions, and management blame for decisions that do not turn out well
  - Worry about lack of connection between planning and design activities and actual source code
    - Working software is the primary measure of progress
  - Worry about the communication gap between management and technical staff

eXtreme Programming Properties

- Characteristics of XP
  - In many ways, XP is a philosophy rather than just a process
  - It is characterized by:
    - continuing feedback from short cycles
    - incremental planning that evolves with the project
    - responsive flexibility in scheduling
    - heavy and continuous use of testing and test automation

eXtreme Programming Properties

- Characteristics of XP
    - emphasis on close and continuous collaboration and communication
    - use of tests and source code as primary communication media (communication at programmer's level)
    - evolutionary model from conception to retirement of system
    - emphasis on small, short-term practices that help yield high quality long-term results

Attacking Risks Before They Arise

- Addressing Risk
  - XP tries to explicitly address the greatest risks to software development projects actually observed in practice

Attacking Risks Before They Arise

- 1) Schedule Slips
  - Software isn't ready on the scheduled delivery date
  - Addressed in XP by short release cycles, frequent delivery of intermediate versions to customers, customer involvement and feedback in development of software

Attacking Risks Before They Arise

- 2) Project Cancellation
  - After several schedule slips, the project is cancelled
  - Addressed in XP by making the smallest initial release that can work, and putting it into production early, thus establishing credibility and results

Attacking Risks Before They Arise

- 3) System Defect Rate Too High, or Degrades with Maintenance
  - Software put in production, but defect rate is too high, or after a year or two of changes rises so quickly that system must be discarded or replaced
  - Addressed in XP by creating and maintaining a comprehensive set of tests run and re-run after every change, so defect rate cannot rise
  - Programmers maintain tests for each function, users maintain tests for each system feature

Attacking Risks Before They Arise

- 4) Business Misunderstood
  - Software put in production, but doesn't solve the problem it was supposed to
  - Addressed in XP by making customer an integral part of the team, so team is continually refining specification to meet expectations

Attacking Risks Before They Arise

- 5) Business Changes
  - Software put in production, but business problem it is designed for changes or is superseded by new, more pressing business problems
  - Addressed in XP using short release cycles and by having customer as an integral part of the team
  - Customer helps team continually refine specification as business issues change, adapting to new problems as they arise - programmers don't even notice

Attacking Risks Before They Arise

- 6) Featuritis
  - Software has a lot of potentially interesting features, which were fun to implement, but don't help customer make more money
  - Addressed in XP by addressing only the highest priority tasks, maintaining focus on real problems to solve

Attacking Risks Before They Arise

- 7) Staff Turnover
  - After a while, the best programmers begin to hate the same old program, get bored and leave
  - In XP, programmer make their own estimates and schedules, get to plan their own time and effort, get to test thoroughly
  - Less likely to get frustrated with impossible schedules and expectations
  - In XP, emphasis is on day to day social human interaction, pair and team effort and decisions
  - Less likely to feel isolated and unsupported

Criticisms of XP

- Introduction of XP resulted in immediate criticism
  - Insufficient software design
  - Lack of structure and documentation
  - Only as effective as the people involved
    - Agile methods like XP often require senior developers
  - Can be inefficient
  - Pair programming can be difficult and expensive, although rewarding

XP 1st ed. / XP 2nd ed.

- Second edition of Beck’s book, which we are **not** following at all, changed a lot of things
- 2nd edition subtitled “EMBRACE CHANGE”: XP applied to itself (very convenient...)
- Dropped some useful technical content (refactoring, coding standards)
- Added some other things (open plan offices...)

Summary

- eXtreme Programming
  - A new software process, programmer-centred
  - Strongly based on testing at every level
  - Designed to address usual project failure risks before they arise
  - We will revisit and attach our course material to eXtreme as we go along

Summary

- References
  - Beck, eXtreme Programming Explained, ch. 1 (1st ed.)
- Reading Assignment
  - Read Beck, eXtreme Programming Explained, ch. 2 (1st ed.)
- Rest of These Slides
  - More eXtreme Programming, the practices of XP

XP in Practice

- Outline
  - Here we look at the actual practices of the XP process, and how they can be applied in the context of our project
  - The key ideas to keep in mind at all times are:
    - metaphor
    - simplicity
    - testing
    - automation
    - collective work
    - standards

Agile Development

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

Although there is value in the items on the right, agile software developers value the items on the left more.

http://agilemanifesto.org

XP 1: The Planning Game

- Refers to the practice of having a continuous dialog between business and technical people on the project
  - Often in the form of weekly meetings, where business people bring business constraints, and technical people bring technical constraints
    - Business people bring issues of scope, priority, releases
    - Technical people being estimates, consequences, scheduling
  - Forces the project members to continually balance what is possible (the technical aspects) with what is desirable (the business aspects)
    - Unfortunately we won't really be able to practice this in the project, the closest we come is our dialog in class and email

XP 2: Small Releases

- Refers to the practice of addressing only the most pressing business requirements, and getting them addressed by releasing a new version quickly
  - Means that we should bring the first version into production as quickly a possible
  - Means that we should shrink the cycle to the next version as much as possible
  - In practice this means shrinking software cycles to a month or two instead of six months or a year
    - In our project, we will shrink to quick releases at roughly two week intervals

XP 3: Metaphor

- Refers to the practice of understanding and speaking of the system in real-world terms independent of its programmed solution
  - An example of a metaphor is the "desktop" of modern operating systems
    - The goals in building such an operating system can be understood in terms of an office desk
  - The metaphor drives the design of the architecture and interfaces of the system
    - In our project, the metaphor is "native", that is, there is a natural physical understanding of what we are doing, our front end is simply a retail console

XP 4: Simplicity

- Refers to the practice of always using the simplest possible design and code that can handle the tests
  - Do not speculate or try to guess what will be needed in the future, address only the current test suite
  - Do not implement any features that do not affect the test results
    - In our project, the simplest, smallest solution will be considered the best

XP 5: Testing

- The only required program features are those for which there is an automated test
  - Always create tests first, and treat them as the goal (specification)
  - Programmers create unit tests (tests for each method or segment of code)
  - Customers create functional (acceptance) tests that check that the product has the required functionality
    - In our project, we will create explicit tests first as we go along, beginning with assmt. #1, and program to meet them

XP 6: Refactoring

- Refers to the practice of continually looking for ways to simplify the architecture and coding of the system as new features and changes are made
  - When a new feature or change is needed, we first look to see if there is a way to rearchitect the system to make it easier or simpler to add - if so, we rearchitect first
  - Once the new feature has been added or changed, we look to see if the resulting new program can be simplified by rearchitecting or merging similar code
    - In our project, we will face changes that may require refactoring

XP 7: Pair Programming

- Refers to the practice of having all production code written with two people working together on one terminal
  - One partner works tactically on the specific part of the code (e.g. method) being coded at the moment
  - The other partner works strategically, considering higher level issues such as:
    - is this approach going to work?
    - can we simplify this by restructuring?
    - what other tests do we need to address here?
  - In our project, we will do all programming in pairs

XP 8: Collective Ownership

- Refers to the practice of having everyone responsible for the quality of the software, and no one to blame for failures of the software
  - Everyone is responsible for identifying opportunities to improve things and to act upon them at any time
  - No one owns the code, it belongs to everyone together - there is no notion of "my code", only the universal notion of "our code"
    - In our project, all team members will be collectively responsible for all parts of all phases

XP 9: Continuous Integration

- In XP, new code is always integrated and tested within a day
  - Changes are not allowed to go on without being continually tested in context to catch integration failures before they happen
    - In our project, starting with assignment #2, we will model this by testing again immediately after each day's changes

XP 10: On-site Customer

- A real customer must be a part of the development team at all times
  - Must be available to answer questions, resolve disputes, set short-term priorities based on business knowledge
    - In our project, we will model this by having the customer (me) available by email
      (not quite right, but it will have to do!)

XP 11: Coding Standards

- Project-wide conventions about the coding of programs
  - Necessary since everyone is responsible for all of the code, and may have to read or change any part of it at any time
  - Usually specifies
    - Commenting standards, e.g., every method must have a comment of the form ...
    - Naming conventions, e.g., variables representing dates will always be named ending in "Date", all constant will be named with a two letter prefix indicating their business type
  - In our project, you will be required to specify your coding standards, and they will be judged according to the clarity, readability, and consistency of your code.

Summary: XP Practices

- XP Practices
  - XP uses a set of standard practices that together form an easy to apply practical system for team development of software
  - Emphasis is on collective responsibility, continuous improvement, and high quality standards
  - We will try to apply these practices in the course project

Summary

- References
  - Beck chapter 10 (1st ed.)
- Reading Assignment
  - Beck chapters 11, 12 (1st ed.)
- Next Lecture(s)
  - Course Project
  - Thursday, Sept. 20: **no lecture**
  - Introduction to Systematic Testing
- Then
  - **Mini-Exam #1 Monday, 24 September**
  - Covers everything through **this slide**