

---

# **Agile Software Development**

## **Mastering Agility – Part II (Unit 6)**

# UNIT VI:

---

- ✧ Deliver Value: Exploit Your Agility,
- ✧ Only Releasable Code Has Value, Deliver Business Results,
- ✧ Deliver Frequently,
- ✧ Seek Technical Excellence :Software Doesn't Exist,
- ✧ Design Is for Understanding,
- ✧ Design Tradeoffs, Quality with a Name, Great Design,
- ✧ Universal Design Principles,
- ✧ Principles in Practice, Pursue Mastery

# Agile Methods - Commonalities

---

- ✧ Agile methods do share common values and principles
- ✧ From several different values and principles, we can form five themes:
  - Improve the Process
  - Rely on People
  - Eliminate Waste
  - Deliver Value
  - Seek Technical Excellence

# Deliver Value

---

- ✧ Your software only begins to have real value when it reaches users
- ✧ That's why successful agile projects deliver value early, often, and repeatedly

# Deliver Value

---

- ✧ Exploit Your Agility
- ✧ Only Releasable Code Has Value
- ✧ Deliver Business Results
- ✧ Deliver Frequently

# Exploit Your Agility

---

- ✧ Agility requires you to work in small steps, not giant leaps
- ✧ A small initial investment of time and resources, properly applied, begins producing quantifiable value immediately

# Exploit Your Agility – In Practice

---

- ✧ XP exploits agility by removing the time between taking an action and observing its results, which improves your ability to learn from this feedback
- ✧ This is especially apparent when the whole team sits together
- ✧ Developing features closely with the on-site customer allows you to identify potential misunderstandings and provides nearly instant responses to questions

# Exploit Your Agility – In Practice

---

- ✧ XP allows changes in focus through short work cycles
- ✧ The short work unit of iterations and frequent demos and releases create a reliable rhythm to make measured process adjustments
- ✧ Slack provides spare time to make small but necessary changes within an iteration



# Only Releasable Code Has Value

---

- ✧ Having code that meets customer needs perfectly has little value unless the customer can actually use it.
- ✧ Until your software reaches the people who need it, it has only potential value
- ✧ Delivering actual value means delivering real software
- ✧ Unreleasable code has no value
- ✧ Working software is the primary measure of your progress

# Only Releasable Code Has Value – Contd.

---

- ✧ Only code that you can actually release to customers can provide real feedback on how well you're providing value to your customers
- ✧ That feedback is invaluable.

# Only Releasable Code Has Value

## – In Practice

---

- ✧ The most important practice is that of “done done,” where work is either complete or incomplete
- ✧ This unambiguous measure of progress immediately lets you know where you stand
- ✧ Test-driven development produces a safety net that catches regressions and deviations from customer requirements
- ✧ A well-written test suite can quickly identify any failures that may reduce the value of the software

# Deliver Business Results

---

- ✧ Agile teams value working software over comprehensive documentation.
- ✧ Documentation is valuable - communicating what the software must do and how it works is important
- ✧ But your first priority is to meet your customer's needs
- ✧ The primary goal is always to provide the most valuable business results possible

# Deliver Business Results - In Practice

---

- ✧ XP encourages close involvement with actual customers by bringing them into the team, so they can measure progress and make decisions based on business value every day
  - Customer's vision provides answers to the questions most important to the project
  - XP approaches its schedule in terms of customer value
  - The team works on stories phrased from the customer's point of view and verifiable by customer testing
  - Iteration demo shows the team's current progress to stakeholders

# Deliver Frequently

---

- ✧ Delivering working, valuable software frequently makes your software more valuable.
- ✧ Delivering working software as fast as possible enables two important feedback loops
  - One is from actual customers to the developers
    - Where the customers use the software and communicate how well it meets their need
  - The other is from the team to the customers
    - Where the team communicates by demonstrating how trustworthy and capable it is

# Seek Technical Excellence

---

- ✧ Technical excellence includes personal accountability in which developers are committed to learning and improvement, and a commitment to the team and working together to get projects completed.
- ✧ Test-driven development, in particular, calls for code to be written in stages; complete one small piece of it, test it, modify it, and then move on to the next small piece of it. This leads us into the benefits of technical excellence.



**SPECIFICATION BY EXAMPLE**



**CONTINUOUS  
INTEGRATION**



**CONTINUOUS DELIVERY**



**TEST AUTOMATION**



**TECHNICAL  
EXCELLENCE**



**ARCHITECTURE  
& DESIGN**



**ACCEPTANCE  
TESTING**



**THINKING ABOUT TESTING**

**CODE**

**CLEAN CODE**



**TEST-DRIVEN DEVELOPMENT**



**UNIT TESTING**

<http://less.works> CC BY-ND



# Seek Technical Excellence

---

- ✧ Software Doesn't Exist
- ✧ Design Is for Understanding
- ✧ Design Trade-offs
- ✧ Quality with a Name
- ✧ Great Design
- ✧ Universal Design Principles
- ✧ Principles in Practice
- ✧ Pursue Mastery

# Seek Technical Excellence

---

- ✧ What we actually write (program) is not software
- ✧ It is a very detailed specification for a program that writes the software for you
  - This special program translates your specification into machine instructions,
  - Then directs the computer's operating system to save those instructions as magnetic fields on the hard drive
- ✧ The specification is the source code
- ✧ The program that translates the specification into software is the compiler

# Design Is for Understanding

---

- ✧ If source code is design, then what is design?
- ✧ Why do we bother with all these UML diagrams and CRC cards and discussions around a whiteboard?
- ✧ All these things are abstractions - even source code
- ✧ The reality of software's billions of evanescent electrical charges is inconceivably complex, so we create simplified models that we can understand
- ✧ Some of these models, like source code, are machine-translatable
- ✧ Others, like UML, are not - at least not yet

# Design Trade-offs

---

- ✧ Earlier, programmers (assembly language) had to take tough decisions between space (memory) and speed

## With an optimizing compiler

- ✧ C is just as good as assembly language.
- ✧ C++ adds virtual method lookups—requiring more memory and an extra level of indirection.
- ✧ Java and C# add a complete intermediate language that runs in a virtual machine atop the normal machine.

# Quality with a Name

---

- ✧ If we're not balancing speed/space trade-offs, what are we doing?
- ✧ Actually, there is one trade-off that we make over and over again
- ✧ Java, C#, and Ruby demonstrate that we are often willing to sacrifice computer time in order to save programmer time and effort

# Quality with a Name – Contd.

---

- ✧ Some programmers flinch at the thought of wasting computer time and making “slow” programs.
- ✧ However, wasting cheap computer time to save programmer resources is a wise design decision
- ✧ Programmers are often the most expensive component in software development

# Quality with a Name – Contd.

---

- ✧ If software design's only real trade-off is between machine performance and programmer time
- ✧ Then the definition of “good software design” becomes crystal clear:
  - *A good software design minimizes the time required to create, modify, and maintain the software while achieving acceptable runtime performance*

# Great Design

---

- ✧ Equating good design with the ease of maintenance is not a new idea, but stating it this way leads to some interesting conclusions
  - Design quality is people-sensitive
    - Relies so heavily on programmer time, it's very sensitive to which programmers are doing the work.
    - A good design takes this into account
  - Design quality is change-specific
    - A genuinely good design correctly anticipates the changes that actually occur



# Great Design – Contd.

---

## ✧ Great designs

- Modification and maintenance time are more important than creation time
  - A good design focuses on minimizing modification and maintenance time over minimizing creation time
- Design quality is unpredictable
  - If a good design minimizes programmer time, and it varies depending on the people doing the work and the changes required, then there's no way to predict the quality of a design

# Great Design – Contd.

---

✧ Thus, great designs

- Are easy to modify by the people who most frequently work within them
- Easily support unexpected changes
- Are easy to maintain
- Prove their value by becoming steadily easier to modify over years of changes and upgrades

# Universal Design Principles

---

- ✧ The Source Code Is the (Final) Design
- ✧ Don't Repeat Yourself (DRY)
- ✧ Be Cohesive
- ✧ Decouple
- ✧ Clarify, Simplify, and Refine
- ✧ Fail Fast
- ✧ Optimize from Measurements
- ✧ Eliminate Technical Debt

# Principles in Practice

---

- ✧ These universal design principles provide good guidance, but they don't help with specific languages or platforms
- ✧ That's why you need design principles for specific languages

# Pursue Mastery

---

- ✧ A good software design minimizes the time required to create, modify, and maintain the software while achieving acceptable runtime performance
- ✧ This definition, and the conclusions it leads to, are the most important things one should keep in mind when considering a design.

# Pursue Mastery

---

- ✧ The same is true of agile software development.
- ✧ Ultimately, what matters is success, however you define it
- ✧ The practices, principles, and values are merely guides along the way
- ✧ Learn what the principles mean.
- ✧ Break the rules, experiment, see what works, and learn some more.
- ✧ Share your insights and passion, and learn even more

# Start by following the practices rigorously

---

- ✧ Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- ✧ Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- ✧ Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- ✧ Business people and developers must work together daily throughout the project.
- ✧ Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- ✧ The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- ✧ Working software is the primary measure of progress.
- ✧ Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- ✧ Continuous attention to technical excellence and good design enhances agility.
- ✧ Simplicity—the art of maximizing the amount of work not done—is essential.
- ✧ The best architectures, requirements, and designs emerge from self-organizing teams.
- ✧ At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.