Agile Software Development

Lecture 4: Let's Wrap up Agile Fundamentals

Mahmoud El-Gayyar

elgayyar@ci.suez.edu.eg

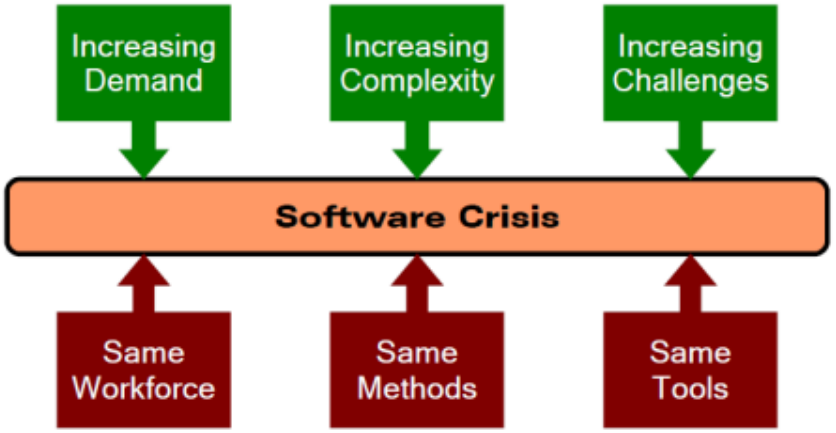
Slides are a modified version of the slides by
Prof. Kenneth M. Anderson
Acknowledgment to Dr. Waleed Ghalwash

Outline

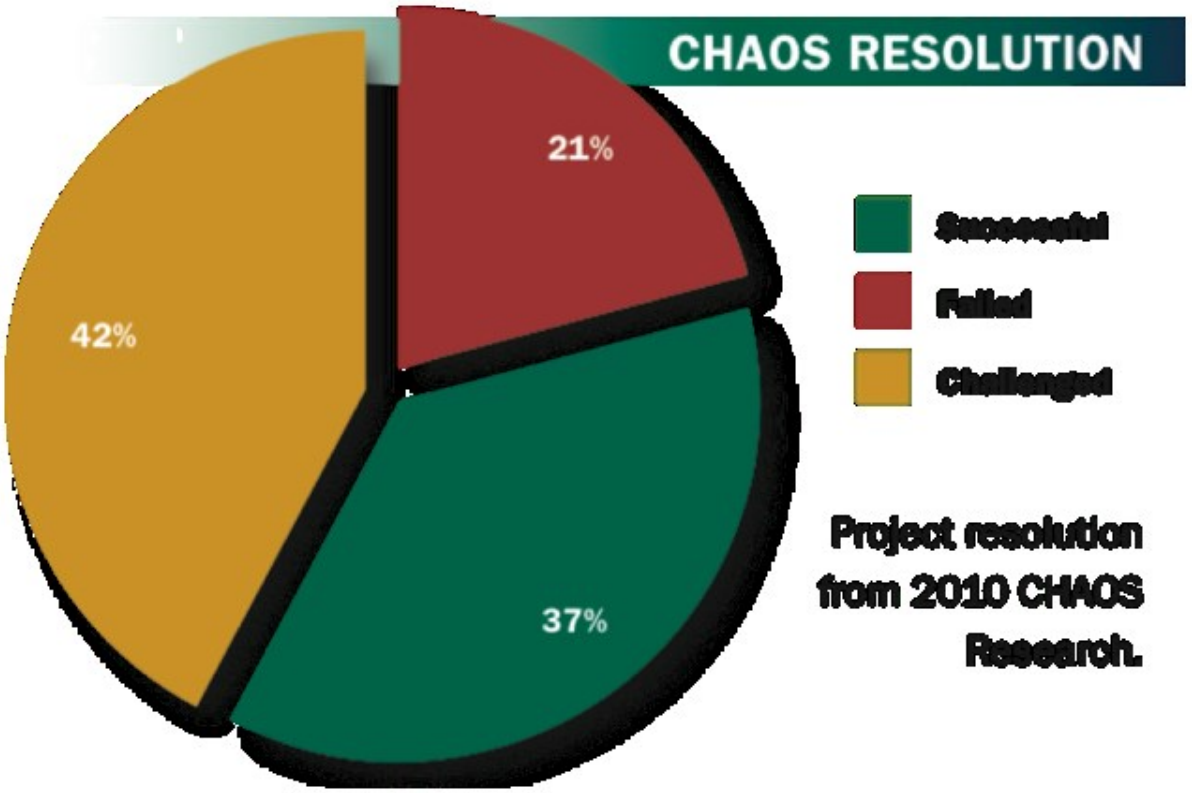
- *Software Crisis*
- *Agile Concepts*
- *SCRUM*
- *Extreme Programming*

Software Crisis

- *a term used in the early days of computing science.*
- *used to describe the impact of rapid increases in computer power and the complexity of the problems that could be tackled.*
- *In essence, it refers to the difficulty of writing correct, understandable, and verifiable computer programs.*



Software Success Rate



63% OF SOFTWARE PROJECTS NOT SUCCESSFUL

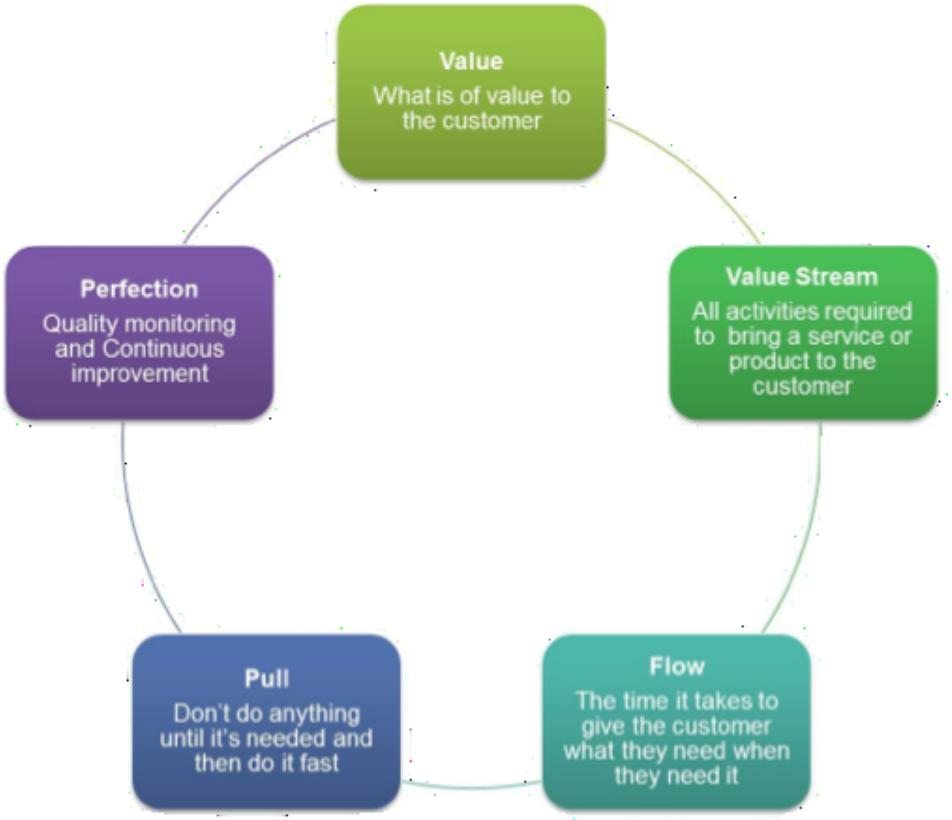
Why software projects fail?

- *Unrealistic Schedules*
- *Inappropriate Staffing*
- *Changing Requirements During Development*
- *Poor-Quality Work*
- *Believing in Magic*

LEAN THINKING

Lean Principles

- *Eliminate waste*
- *Amplify learning*
- *Decide as late as possible (defer commitment)*
- *Deliver fast*
- *Empower the team*
- *Build quality in*
- *Optimize the whole*



Eliminate Waste

- *Unclear requirements*
- *Unnecessary code and functionality*
- *Waiting in line (patch work)*
- *insufficient testing*
- *Empower the team*
- *Bureaucracy*
- *Task Switching*

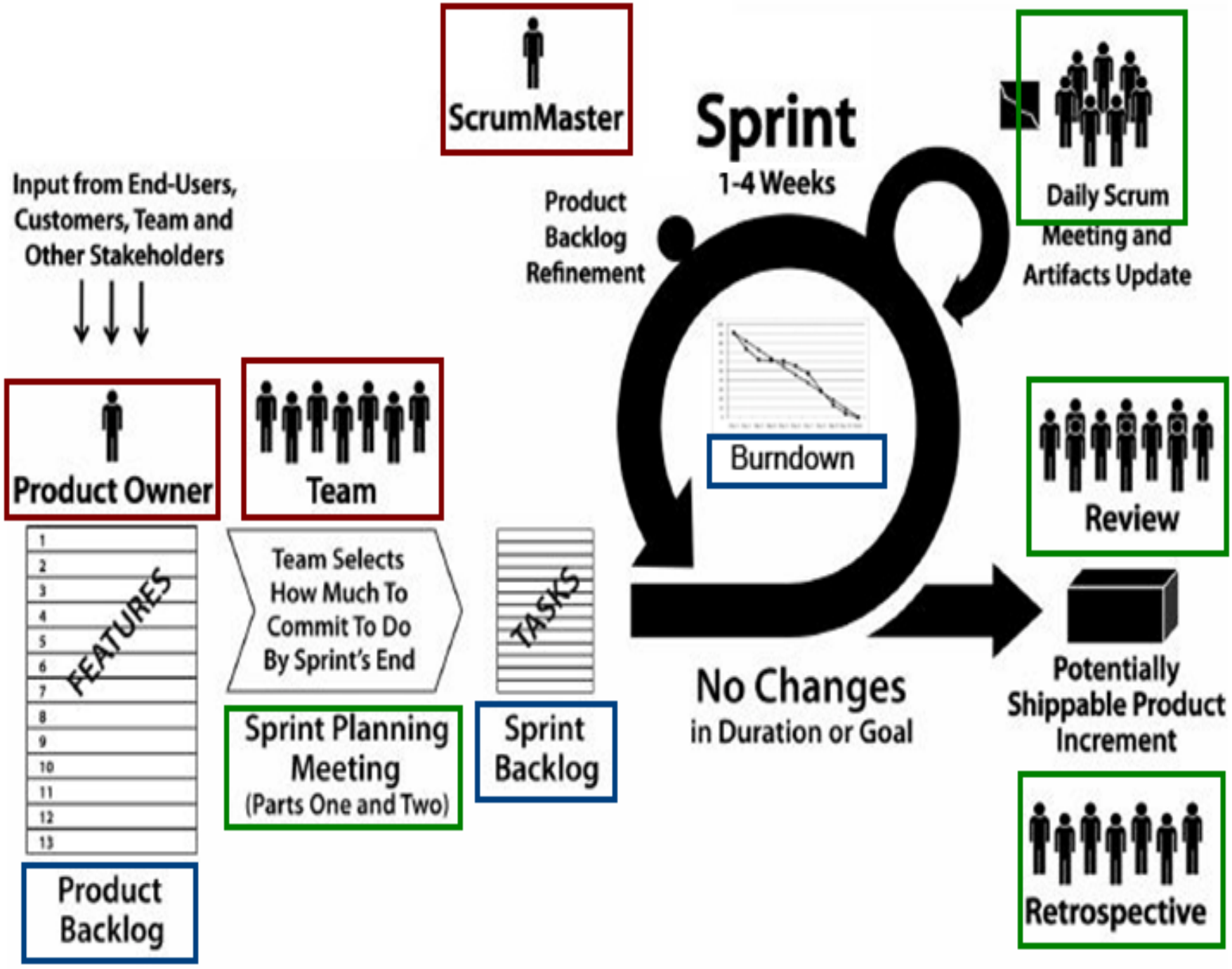
Defer Commitment

- *Apply the 80-20 rule to software development*
  - 80% of the users use only 20% of requirement
  - 80% of the ROI is produced by 20% of the effort.

AGILE MANIFESTO

- INDIVIDUALS & INTERACTIONS OVER PROCESSES & TOOLS
- WORKING SOFTWARE OVER COMPREHENSIVE DOCUMENTATION
- COSTUMER COLABORATION OVER CONTRACT NEGOTIATION
- RESPONDING TO CHANGE OVER FOLLOWING A PLAN

SCRUM
(PROJECT MANAGEMENT)


ScrumMaster
Sprint
1-4 Weeks
Daily Scrum
Meeting and
Artifacts Update
Input from End-Users,
Customers, Team and
Other Stakeholders
Product
Backlog
Refinement
Burndown
Product Owner
Team
Review
1
2
3
4
5
6
7
8
9
10
11
12
13
FEATURES
Team Selects
How Much To
Commit To Do
By Sprint's End
TASKS
No Changes
in Duration or Goal
Potentially
Shippable Product
Increment
Sprint Planning
Meeting
(Parts One and Two)
Sprint
Backlog
Retrospective
Product
Backlog

eXtreme Programming

eXtreme Programming

- *XP takes commonsense software engineering principles and practices to **extreme levels***
- *For instance "Testing is good?", then*
  - "We will test every day" and "We will write test cases before we code"
- *As **Kent Beck** says extreme programming takes certain practices and "sets them at 11 (on a scale of 1 to 10)"*

XP Practices

- *The best way to describe XP is by looking at some of its practices*
  - There are fourteen standard practices

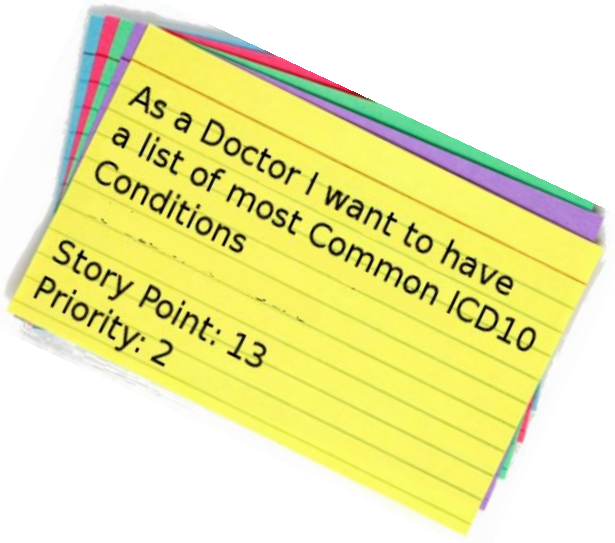
Customer Team Member
User Stories
Short Cycles
Acceptance Tests
Pair Programming
Test-Driven Development
Collective Ownership
Continuous Integration
Sustainable Pace
Open Workspace
The Planning Game
Simple Design
Refactoring
Metaphor

1. Customer Team Member

- *The "**customer**" is made a member of the development team*
  - The customer is the person or group who defines and prioritizes features
  - A customer representative should be "in the same room" or at most 100 feet away from the developers.
  - "Release early; Release Often" delivers a working system to the client organization; in between, the customer representative provides continuous feedback to the developers

2. User Stories

- *We need to have requirements*
- *XP requirements come in the form of "**user stories**" or **scenarios***
- *We need just enough detail to estimate how long it might take 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



3. Short Cycles

- *An XP project delivers working software every two weeks that addresses some of the needs of the customer*
  - At the end of each iteration, the system is demonstrated to the customer in order to get feedback

4. Acceptance Tests

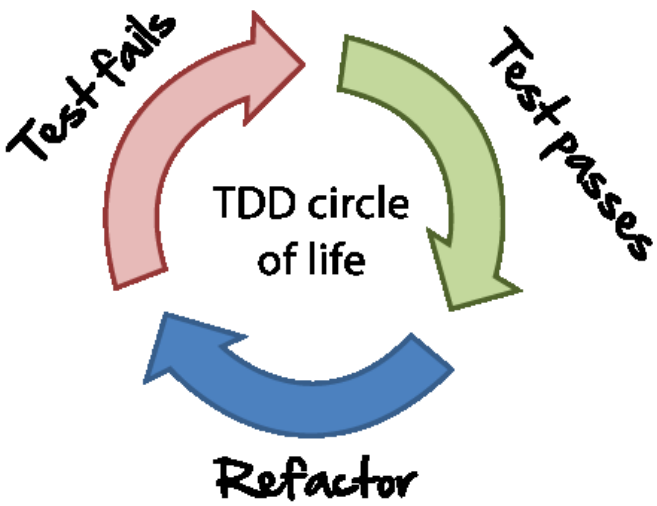
- *Details of a user story are captured in the form of acceptance tests specified by the customer*
  - The tests are written before a user story is implemented
  - They are written in a scripting language or testing framework that allows them to be run automatically and repeatedly
  - These tests are run several times a day each time the system is built

5. Pair Programming

- *Code is written by pairs of programmers*
  - working together at the same workstation
  - One member (**Driver**) drives the keyboard and writes code and test cases; the second watches the code, looking for errors and possible improvements (**Navigator**)
  - The roles will switch between the two frequently
  - Pair membership changes once per day; so that each programmer works in two pairs each day
    - *this facilitates distribution of knowledge about the state of the code throughout the entire team*

6. Test Driven Development

- *All production code is written in order t make failing test cases pass*
  - First, we write a test case that fails since th required functionality has not yet bee implemented
  - Then, we write the code that makes that test cas pass
  - Iteration between writing tests and writing code is very short; on the order of minutes
  - As a result, a very complete set of test cases is written for the system;



7. Collective Ownership

- *A pair has the right to check out/improve ANY module*
  - Developers are never individually responsible for a particular module or technology

8. Continuous Integration

- *Developers check in code and integrate it into the larger system several times a day*
- *Entire system is built every day; e.g. if the final result is a web site, they deploy the web site on a test server.*
- *This avoids the problem of **cutting integration testing** to "save time and money"*

9. Sustainable Pace

- *A software project is not a sprint; it's a marathon*
  - A team that leaps off the starting line and races as fast as it can will burn out long before the finish line
  - The team must instead "run" at a sustainable pace
- *An XP rule is that a team is not allowed to work overtime*
  - This is also stated as "40 hour work week"

10. Open Workspace

- *The team works together in an open room*
  - There are tables with workstations
  - There are whiteboards on the walls for the team members to use for status charts, task tracking, UML diagrams, etc.
- *"War room" environments can double productivity*

11. The Planning Game

12. Simple Design

- *An XP team makes their designs as simple and expressive as they can be*
  - They narrow focus to current set of stories and build the simplest system that can handle those stories

13. Refactoring

- *XP teams fight "code rot" by employing refactoring techniques constantly*
- *By "constantly" we mean every few hours versus "at the end of the project", "at the end of the release", or "at the end of the iteration"*

14. Metaphor

- *The big picture that ties the whole system together*
  - Vocabulary that crystallizes the design in a team member's head
  - Example : Windows Desktop
  - Example: network traffic analyzer, every 30 minutes, system polled dozens of network adapters and acquired monitoring data; Each adaptor provides block of data composed of several variables
    - *Metaphor: A toaster toasting bread*
    - *Data Block = "Slices"*
    - *Network analyzer = "The Toaster"*
    - *Slices are raw data "cooked" by the toaster*

Benefits of XP

- Customer Focus
- Emphasis on teamwork and communication
- Programmer estimates before implementation
- Emphasis on responsibility for quality
- Continuous measurement
- Incremental development
- Simple design
- Frequent redesign via refactoring
- Frequent testing
- Continuous reviews via pair programming

Criticisms of XP

- *Code centered vs. Design centered*
  - Hurts when developing large systems
- *Lack of design documentation*
  - Limits XP to small systems
- *Producing readable code is hard*
- *Code is not good documentation*
- *Lack of transition support*
  - how do you switch from waterfall or other process?

Key Points

- *Agile Development main ideas*
- *SCRUM*
- *eXtreme Programming*