


**SE350: PRINCIPLES OF OBJECT ORIENTED DESIGN**

**Jane Cleland-Huang**  
 Office: CDM 836  
 Hours: Tuesday/Thursday 4.40pm-5.30pm  
 Phone: 312-362-8863  
 Email: [jhuang@cs.depaul.edu](mailto:jhuang@cs.depaul.edu)

**DePaul University**  


**CoEST**  
 Center of Excellence for Software Traceability

## IT Today in the USA

### Standish Report



Spend > \$250  
Billion per year in  
175,000 projects.

Avg cost per large  
company is  
\$2,322,000

Med: \$1,331,000

Small: \$434,000

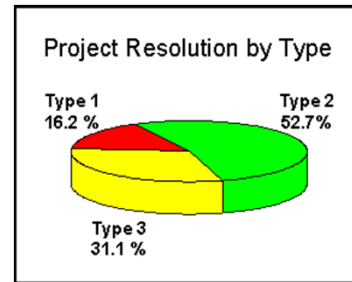
## Chaos Report

### Standish Research Group Report

**Project Success:** Type 1. The project is completed on-time and on-budget, with all features and functions as initially specified.

**Project Challenged:** Type 2. The project is completed and operational but over-budget, over the time estimate, and offers fewer features and functions than originally specified.

**Project Impaired:** Type 3.  
The project is canceled at some point during the development cycle.  
(Are ALL impaired projects failures???)



3

## Chaos Report

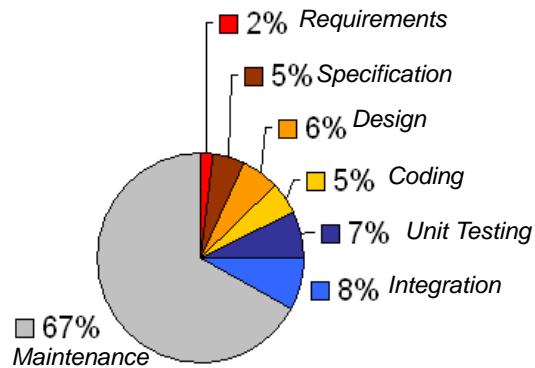
### Standish Research Group Report

Project Success Factor	% of Responses
1. User Involvement	15.90%
2. Executive Management Support	13.90%
3. Clear Statement of Requirements	13.00%
4. Proper Planning	9.60%
5. Realistic Expectations	8.20%
6. Smaller Project Milestones	7.70%
7. Competent Staff	7.20%
8. Ownership	5.30%
9. Clear Vision & Objectives	2.90%
10. Hard-Working, Focused Staff	2.40%
Other	13.90%

4

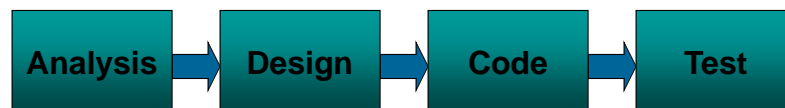
## Traditional Phases

- ▶ Requirements
- ▶ Analysis (specification)
- ▶ Design
- ▶ Implementation
  - Coding
  - Unit testing
  - Integration
- ▶ Post delivery maintenance
- ▶ Retirement



5

## Linear Sequential Models (Classic, Waterfall)

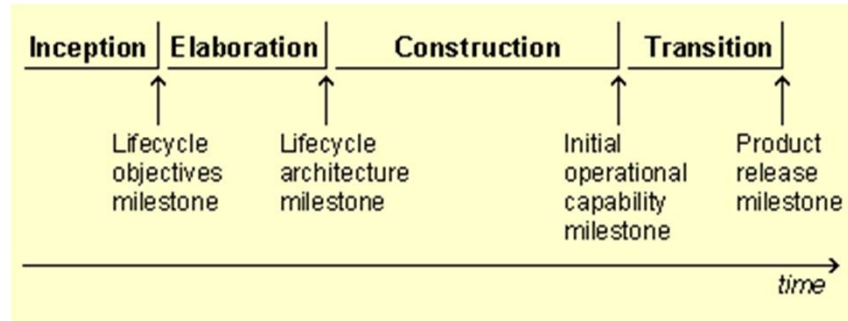


### Assumptions

- All requirements are knowable upfront.
- Requirements have no unresolved, high-risk implications.
- Requirements will not change much during development or evolution.
- Requirements are compatible with all key system stakeholder's expectations, including users, customers, developers, maintainers, investors.
- The right architecture for implementing the requirements is well understood.
- There is enough calendar time to proceed sequentially.

6

## The RUP Phases (The Rational Unified Process)



### Phases:

Define the primary engineering activities and the amount of each activity that should be occurring at each stage of the project.

### Milestones (phase-gates)

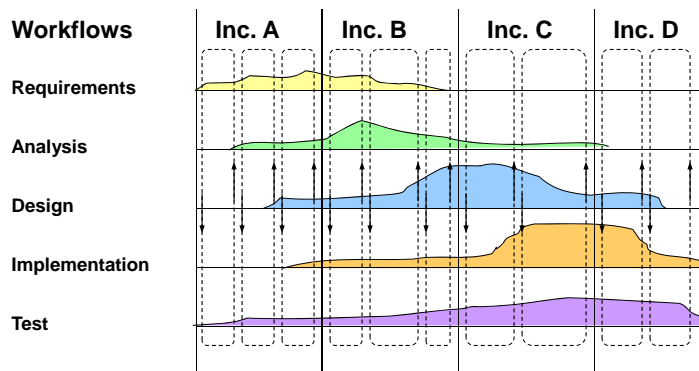
Act as go/no-go decision points between phases.

Require stakeholders to decide whether to proceed to next phase.

7

## Iterations and Increments

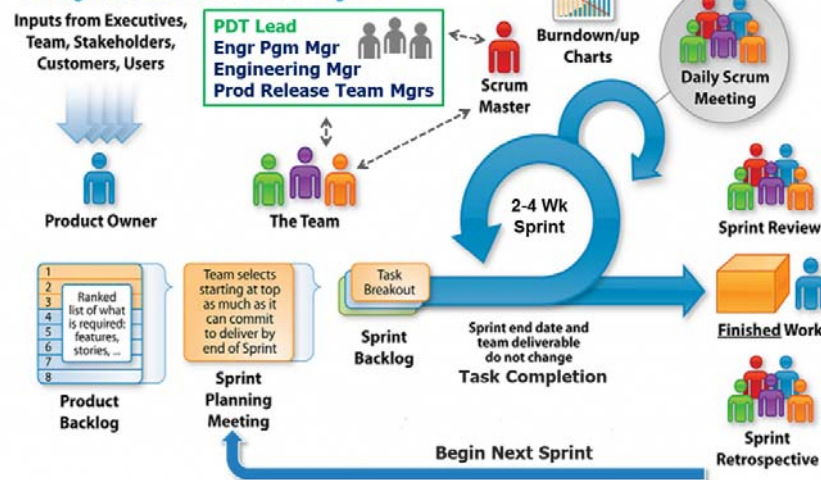
- There is iteration within each increment.
- All five workflows may be included in EACH iteration of each increment – but in varying proportions.



8

# Agile

## The Agile: Scrum Framework at a glance



9

## Manifesto for Agile Development

"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.

That is, while there is value in the items on the right, we value the items on the left more"

<http://www.agilemanifesto.org>

## eXtreme Programming

1. Customer team member
2. User stories
3. Short cycles
4. Acceptance tests
5. Pair programming
6. Test-driven development
7. Collective ownership
8. Continuous integration



9. Sustainable pace
10. Open workspace
11. The planning game
12. Simple design
13. Refactoring
14. Metaphor

## User Stories

- ▶ A **short** description of the behavior of the system from the customer's perspective.

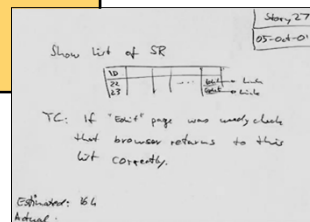
### *Add New User*

*A new user needs to get access to the system. They contact the system administrator, who records their first name, last name, phone number and email address. The new user's username is based on their last name and first letter of their first name, and must be unique. The new user can be marked as an administrator.*

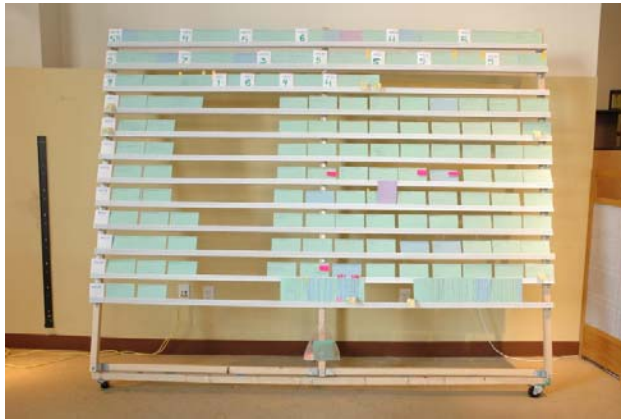
**1 week**

Developers estimate time to develop a user story as 1, 2, or 3 weeks.

Sufficient detail to make a low-risk estimate of how long it will take to complete. Often written on index cards.



## Tracking



Used to track the progression of the project and to determine what to do next....

## Tracking



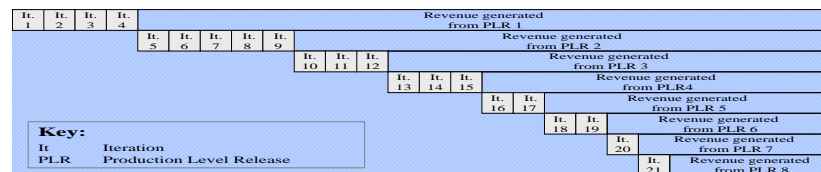
Recently completed user stories are tracked by completion date and time taken.



Old stories are moved to an "old stories" board.

## Release Planning

- To discover small units of functionality that make good business sense and can be released into the customer's environment early.
- The release plan specifies:
  - Which user stories are going to be implemented for each system release
  - Dates for those releases.
- A single system release is composed of multiple iterations.



## Project Velocity

$$\text{PV in story weeks} = \sum \text{Time estimated for the story}$$

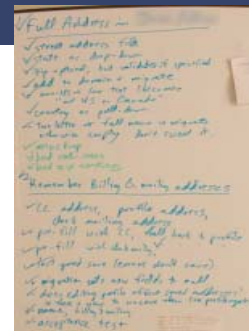
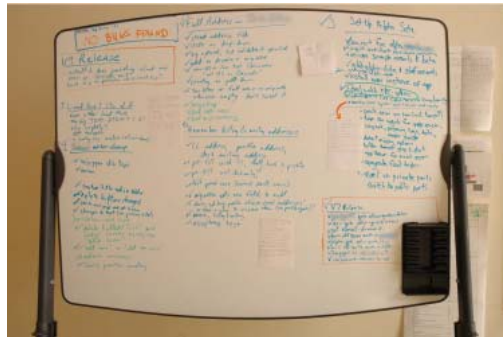
All stories completed  
in the iteration

- Don't bother to consider # of programmers or skill level. This is a rough estimation.
- Project velocity tells you how many story points you can allocate to the next iteration.
- The customer gets to pick stories that equal the project velocity.



## Once an iteration starts

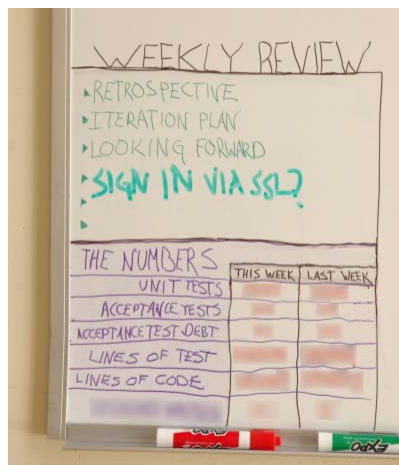
- The user stories and failed tests are translated into the programming tasks that will support them.



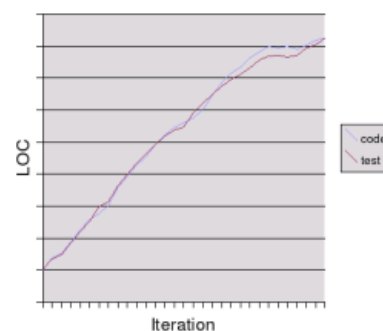
Programmers self-select tasks and perform a more detailed analysis to recalculate effort.

**Project Velocity is recalculated.**

## Post Iteration Review



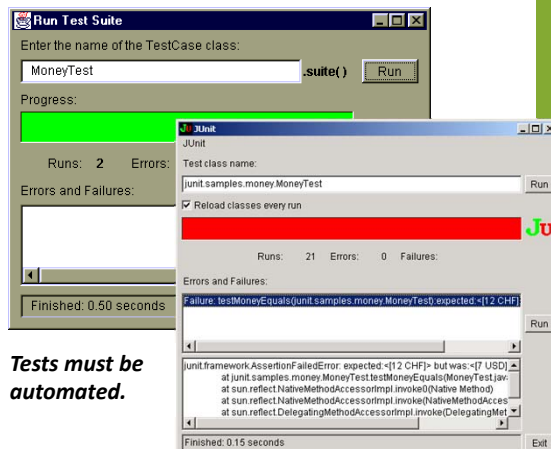
Production Vs Test Code



- Check that the project is on task.
- Plan the next iteration.

## Test-driven Development

- Unit tests are written before the code.
- Enough production code is written to allow the test to compile, but not pass.
- Test is executed to ensure that it fails. (ie is it a good testcase.)
- Task is completed.
- Test is rerun until it passes.



*Tests must be automated.*

## Pairs Programming

- **What is pair programming?** “Two programmers working collaboratively on one algorithm, design, or programming task, sitting side by side at one computer.”
- **Big question:** What is the ROI of pairs programming?



Cockburn, Alistair and Williams, Laurie  
*The Costs and Benefits of Pair Programming*  
 in *Extreme Programming Examined*.  
 Addison Wesley, 2001.

## Refactoring

- ▶ Code tends to rot!
- ▶ XP reverses degradation through frequent refactoring.



### Definition

“Making a series of tiny transformations that improve the structure of the system without affecting its behavior” (Robert Martin).

- ▶ Following refactoring – rerun tests.
- ▶ Keep the system working whilst transforming its design.
- ▶ Refactoring done continuously.

## eXtreme Programming

1. Customer team member ✓
2. User stories ✓
3. Short cycles ✓
4. Acceptance tests ✓
5. Pair programming ✓
6. Test-driven development ✓
7. Collective ownership
8. Continuous integration ✓



9. Sustainable pace
10. Open workspace
11. The planning game ✓
12. Simple design ✓
13. Refactoring ✓
14. Metaphor

## SCRUM Meetings

During a *Sprint*, *Scrum Meetings* are held daily to determine:

1. What items were completed since the last *Scrum Meeting*.
2. What issues or blocks have been found that need to be resolved. (The *ScrumMaster* is a team leader role responsible for resolving the blocks.)
3. What new assignments make sense for the team to complete until the next meeting.

**Scrum Meetings** provide opportunities for team socialization and a shared culture. This promotes a self-organized team structure, where the development process is evolved on a daily basis.



## SCRUM Meetings

Ask each person these three questions:

- a. What they worked on since the last *Scrum Meeting*. The *Scrum Master* logs what tasks have been completed and what remains undone.
- b. What blocks if any they found in performing their tasks within the last 24 hrs. The *Scrum Master* logs all blocks and later finds a way to resolve the blocks.
- c. What they will be working in the next 24 hrs. The *Scrum Master* helps the team members choosing the appropriate tasks to work on with the help of the Architect. Because the tasks are schedule on a 24 hr basis the tasks are typically small (*Small Assignments*).

The *Scrum Meeting* becomes the equivalent of a *thermometer* that constantly samples the team's temperature.



## Sprint

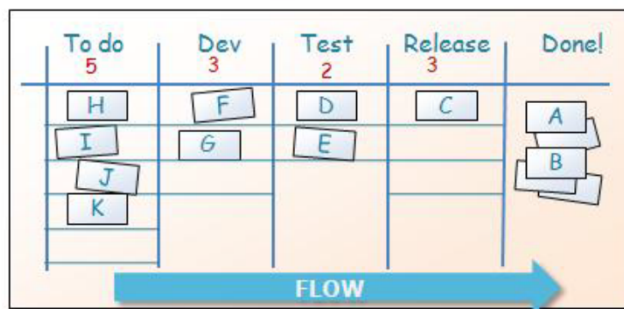
At the end of each *Sprint*, there is a *Demo* to:

1. Show the customer what's going on (**EngageCustomer**)
2. Give the developer a sense of accomplishment (**CompensateSuccess**)
3. Integrate and test a reasonable portion of the software being developed (**EngageQA**)
4. Ensure **real progress** – reduction of backlog, not just the production of more papers / hours spent (**NamedStableBases**)

After gathering and reprioritizing leftover and new tasks, a new *Backlog* is formed and a new *Sprint* starts..



## Kanban



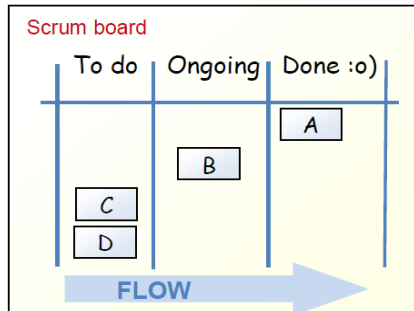
- **Visualize the workflow**
  - Split the work into pieces, write each item on a card and put on the wall
  - Use named columns to illustrate where each item is in the workflow.
- **Limit WIP** (work in progress) – assign explicit limits to how many items may be in progress at each workflow state.
- **Measure the lead time** (average time to complete one item, sometimes called “cycle time”), optimize the process to make lead time as small and predictable as possible.

KANBAN

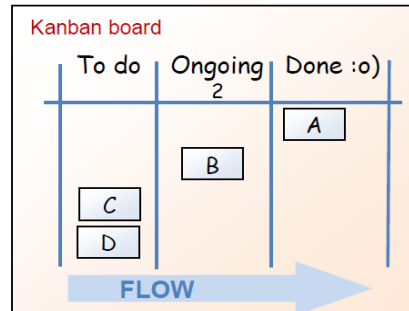
26

## WIP (Work in Progress) Limits

## PROCESSES



**Scrum WIP is limited per unit of time.** Scrum teams usually measure *velocity* – how many items (or “story points”) get done per iteration. Once the team knows their velocity, that becomes their WIP limit.



**Kanban WIP is limited per workflow state** e.g. in this project at most 2 items may be in the workflow state “Ongoing” at any given time. WIP could also be measured per state using story points.