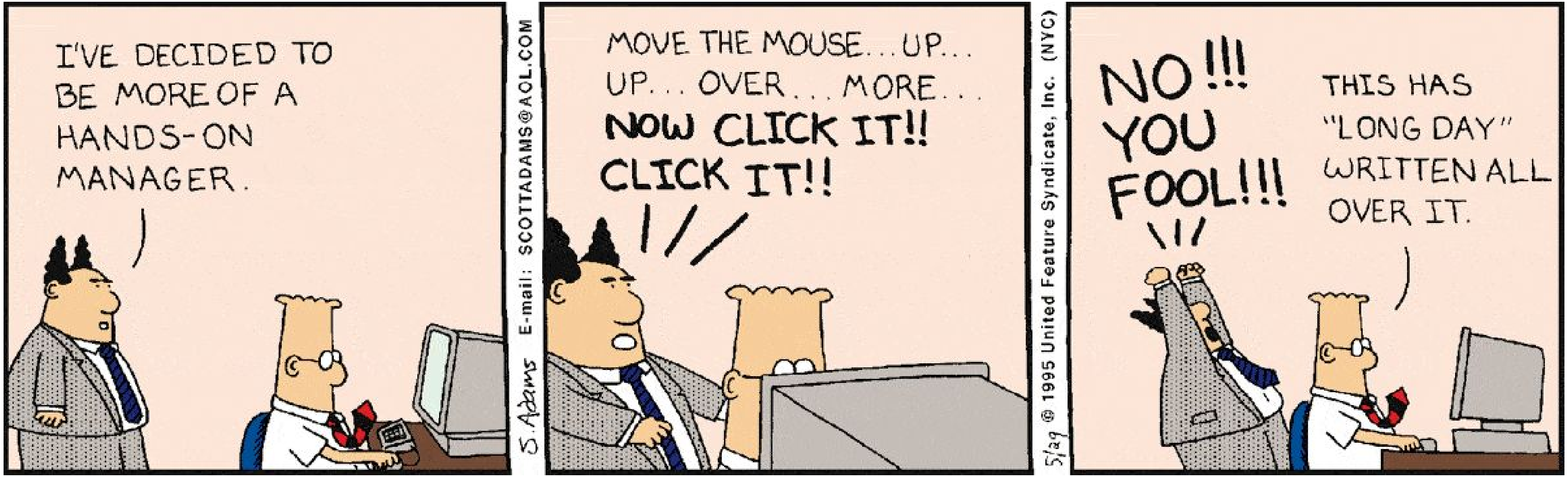

I'VE DECIDED TO BE MORE OF A HANDS-ON MANAGER.
MOVE THE MOUSE... UP... UP... OVER... MORE... NOW CLICK IT!! CLICK IT!!
NO!!! YOU FOOL!!!
THIS HAS "LONG DAY" WRITTEN ALL OVER IT.
S. Adams E-mail: SCOTTADAMS@AOL.COM
5/29 © 1995 United Feature Syndicate, Inc. (NYC)

Department of Computing

COVENTRY UNIVERSITY

302CEM

Agile Development

Mark Tyers 2016

Coventry University

Learning Outcomes

- Extreme Programming
  - To become familiar with XP principles
  - To understand how XP is used in practice
- Scrum
- Software Architecture
  - Domain-Driven Design
  - Design Patterns

Department of Computing

COVENTRY UNIVERSITY

Extreme Programming (XP)

Coventry University

Overview

XP process

XP values and principles

XP practices

XP Process 1/3

3 – 10 programmers

One or more customers on site.

Develop in weekly iterations, producing running code that is of value to the customer.

Release to end users every 2 to 5 iterations.

User Requirements

Unit of requirements gathering is the "user story"

User-visible functionality that can be developed within one iteration or less.

XP Process 3/3

Programmers work in pairs

Assignments are rotated.

Unit tests are written first

Must always run at 100%.

Stand-up meeting every day

Meeting at the end of each iteration.

Roles and Responsibilities

- **Programmer** – coding, communication, writing tests
- **Customer** – writing user stories and functional test, prioritising requirements
- **Tester** – write and run tests, broadcast results
- **Tracker** – gives feedback on estimations
- **Coach** – guide team to follow process
- **Consultant** – external member with specific technical knowledge
- **Manager** – makes decisions

XP Basic Values

Rapid feedback - Minutes, rather than days or weeks

Assume simplicity - Treat every problem as simple

Incremental change - Solve problems with a series of small changes

Embracing change - The best strategy preserves the most options while solving the most pressing problem

Quality work - Excellent or insanely excellent

THEORY

XP Practices

Planning
Small Releases
Simple design
Testing
Refactoring
Pair programming
Collective ownership
Continuous integration
On-site customer
Coding standards

Planning

Business people decide:

- scope
- priority of features
- composition of releases
- date of releases

Development team decides

- estimates
- technical consequences
- process
- detailed scheduling

THEORY

Planning Phases

Planning has three phases:

- Exploration—learn what the system will eventually do: use cases (stories), estimates
- Commitment—choose the scope and date of next release: business sorts by priority, development sorts by how well they can estimate, business sets the scope
- Steering—update the plan: iterations last one to three weeks, new uses cases as needed, re-estimate

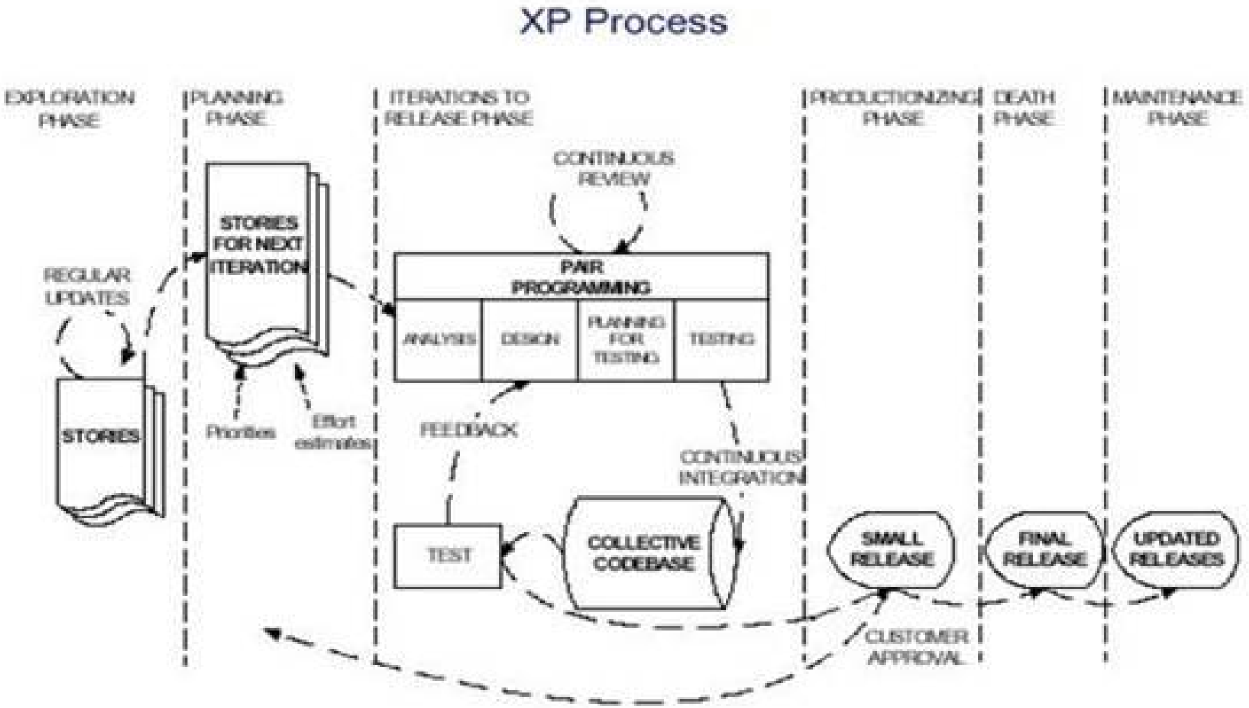
XP Planning Works

The result is what some would consider to be only a rough plan.

It works because:

- the customers were involved
- the release intervals are short, so mistakes become apparent
- the customers continue to work with the development team
- the plan is revised as needed

XP Lifecycle



THEORY

Small Releases

Select scope from a prioritized list, so even a small change has value.

Integrate continuously.

Maintain conformity to the test suite.

Keep the design simple, just sufficient for this release.

Simple Design

"The simplest thing that could possibly work.“

Start with a test, so you will know when you are done.

Design and implement enough to make the test run.

Simplify the design if the opportunity arises.

Refactoring

“Changing a software system in a way that does not alter the external behaviour of the code, while improving its internal structure.”

> As you deal with more and more use cases, refactoring the system will be necessary.
>
> This is thought of as normal, not unusual.
>
> The other parts of the XP approach make this not hard to do.
>
> Refactor to gain simplicity, reduce duplication, or communicate more clearly.

Pair Programming

All code is written by two people at one machine.

What pair programming is not:

~~One writing and one watching for syntax errors.~~

~~A tutorial session.~~

What pair programming is:

Continuous code and design review.

THEORY

Department of Computing

COVENTRY UNIVERSITY

Scrum

Coventry University

What is Scrum?

A subset of agile development

A lightweight process framework

The most popular approach used in industry

THEORY

Lightweight Process Framework

A set of practices that should be followed to be consistent with the framework

By keeping the practices simple and lightweight more time can be spent on the actual development

Time spent on paperwork is not productive

THEORY

Product Owner

The development team must be aligned to the business

Entire team must focus on a single problem domain

One member of the team must be dedicated to the product

They must have a clear understanding of the requirements of the problem domain

They are responsible for prioritising work (user stories)

They should discuss these with the team but they have to take the final decision

THEORY

Scrum Master

It is important that the team sticks to the scrum process

They will need a mentor/coach who is familiar with the techniques and ideally has some previous experience

They are also responsible for resolving any issues that are preventing the team from working effectively such as lack of resources

THEORY

Product Backlog

List of things that need to be done

Anyone in the team can add to this list

List items should be user stories

Can also include non-functional requirements such as improving performance or fixing bugs

The product owner prioritises these after consultation with the rest of the team

Priorities regularly adjusted based on current business needs

THEORY

MoSCoW Rules

User stories fit into four different priority groups:

Must Have: the top items will be essential

Should Have: next there will be the important but lower priority tasks

Could Have: next are the low priority features that will be completed eventually

Would Be Nice: the tasks at the bottom may never be implemented

THEORY

Product Sprint

A time-based unit of development

Restricted to a specific duration (timeboxed)

Duration of each sprint determined by team dynamics

Typically a week in length

Each sprint results in the delivery of working code

THEORY

Sprint Backlog

Ordered list of Product Backlog Items (User Stories)

Team agrees to complete them during the coming Sprint

Taken from the Product Backlog during the Sprint Planning Meeting

Each story assigned a Point value based on the Estimated amount of relative effort it will take to complete the story

No additions or changes until the Sprint ends.

Interrupt or Abort

Product Owner decides there is a feature of higher business value to add to the Sprint

- Triggers the <u>interruption procedure</u>
- Part of team assigned to handle this (triage)

If the interruption will have huge impact

- Product Owner aborts the Sprint
- Triggers a new planning meeting

THEORY

Sprint Planning

Regular meeting to create and plan a Sprint Goal

- Everyone in the team must attend
- Analyse top item from product backlog
- Can it be achieved in the next sprint?
- Identify 'stretch tasks'

THEORY

Clarifying Requirements

Take each item from backlog

Product owner describes it from a functional perspective

Whole team discusses and defines how it can be implemented

Captured on whiteboard or flip chart

Could be wireframes / UML or other

Clear explanation of how these will be tested

THEORY

Sprint Review Meeting

Held at the end of each sprint to improve the the Product Backlog health.

- Attended by product owner, Scrum team and client
- Scrum team demonstrates the new features to the client
- Assessed against the sprint goal
- Kept very informal
- A natural result of the sprint

THEORY

Daily Standup Meeting

Short daily meeting to share insights and create a plan for the day

All team members are required to attend and participate

Each team member answers the following three questions:

What did you do yesterday?

What will you do today?

Are there any impediments in your way?

THEORY

Addressing Problems

Daily standup not used to resolve issues

Issues are raised at the meeting

Resolved by the relevant subgroup immediately after the meeting

THEORY

Retrospection

Takes place after the sprint review meeting and takes around an hour

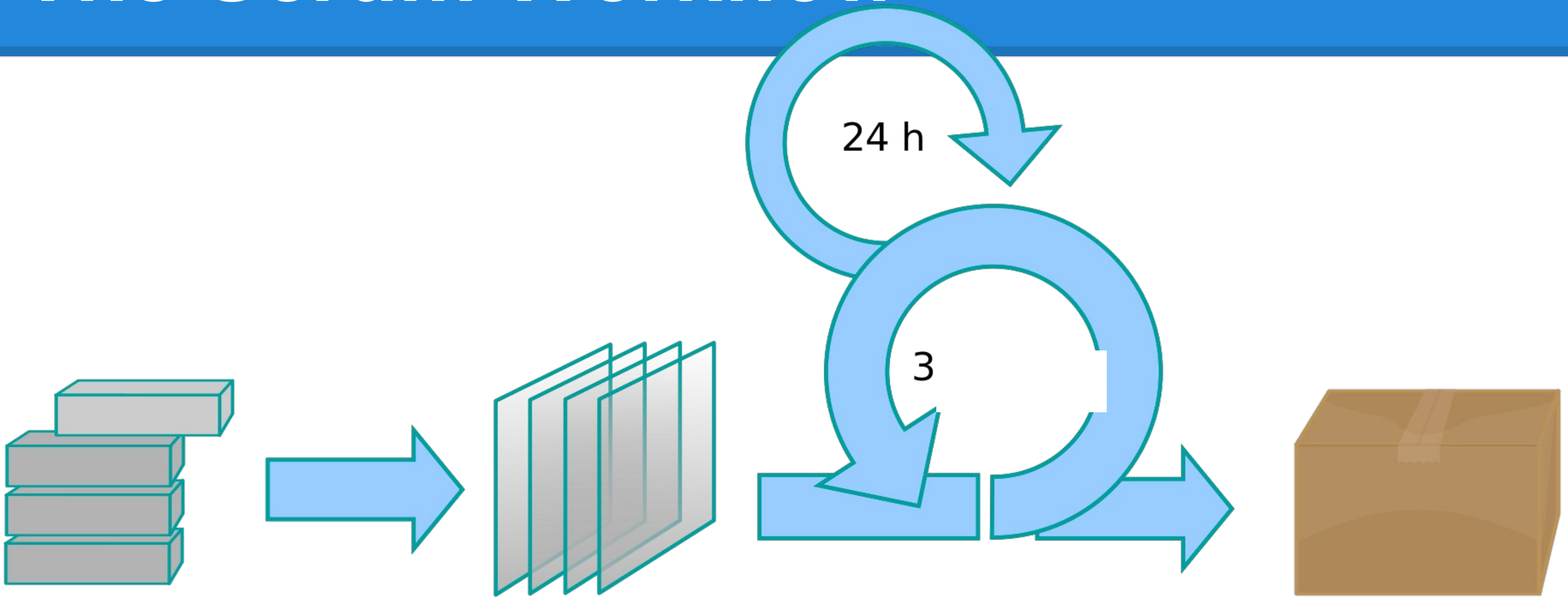
The aim is to improve the way the team do things

The entire team including ScrumMaster product owner should participate

Each member should identify specific stuff the team should:

- Start doing
- Stop doing
- Continue doing

The Scrum Workflow



24 h

3

Product Backlog

Sprint Backlog

Sprint

Working increment
of the software

wikipedia.com

THEORY

Collaborative Workspace

Develop a team workspace

Use whiteboards / sticky notes to organise team

These should be easy to see by entire team at all times

Track progress using a Kanban board

Cross reference to issue trackers

Kanban Board

Arrangement of columns to track work progress

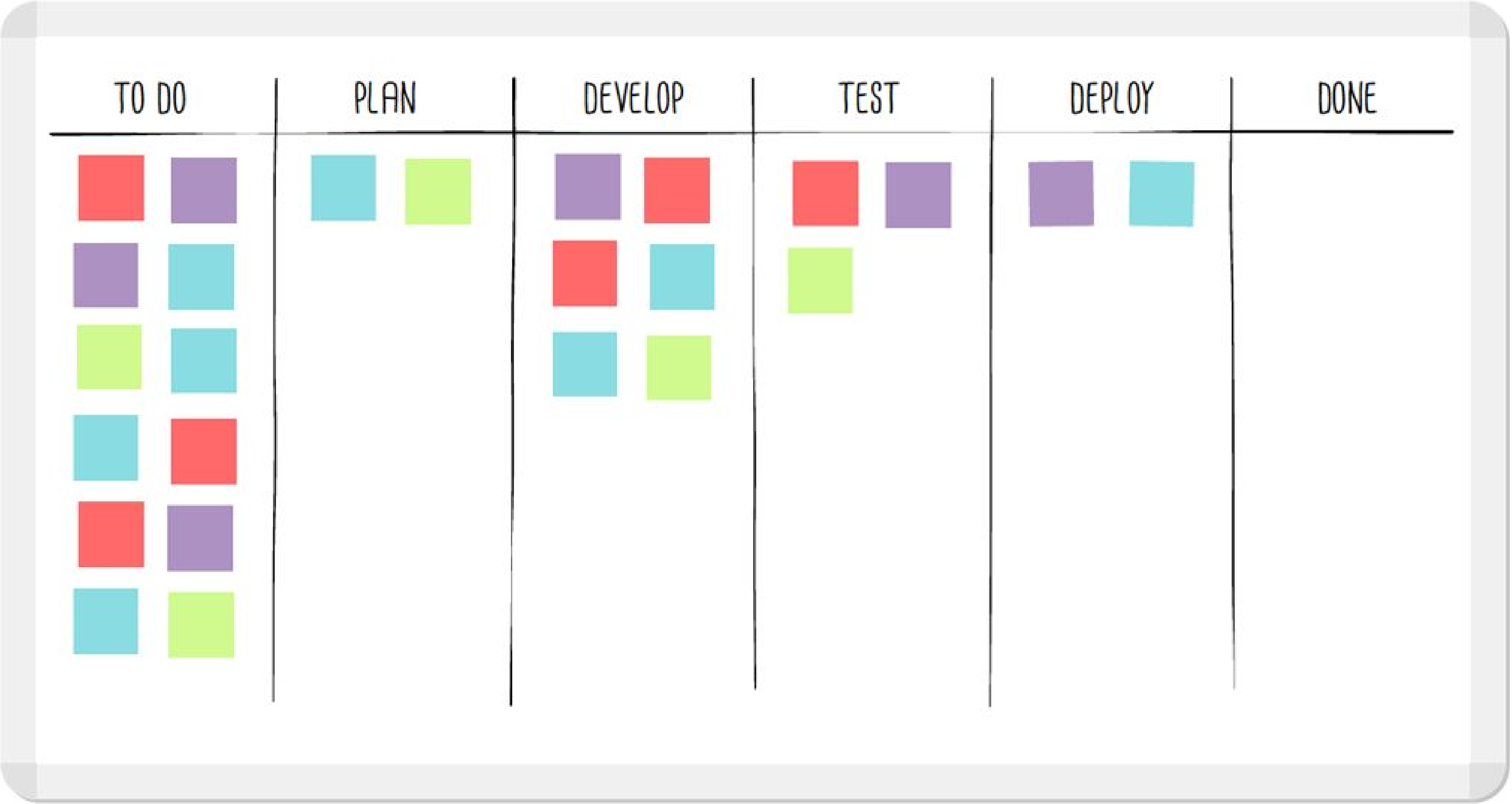
Each column represents a step in the development process

Each user story tracked across the columns

Used to define next steps to be completed

Easy to identify progress and bottlenecks

THEORY



🟪 User Story 🟩 Defect 🟦 Task 🟥 Feature

The Sprint

Implement the Product Sprint

Approach is not defined (scrum is a management process)

Team is empowered to make decisions

Timeboxed (Timeframe is fixed)

Done means done (must be 100% done)

No outside interference

Sprint Planning Meeting

Meet with client to discuss what will be completed in the next sprint.