Software Technology 04

Agile Methodologies

Why Agile?

– Agile is not new (Scrum 1995, XP 1996)

- Agile Manifesto (2001)
  - **Individuals and Interactions** over processes and tools
  - **Working Software** over comprehensive documentation
  - **Customer Collaboration** over contract negotiation
  - **Responding to Change** over following a plan

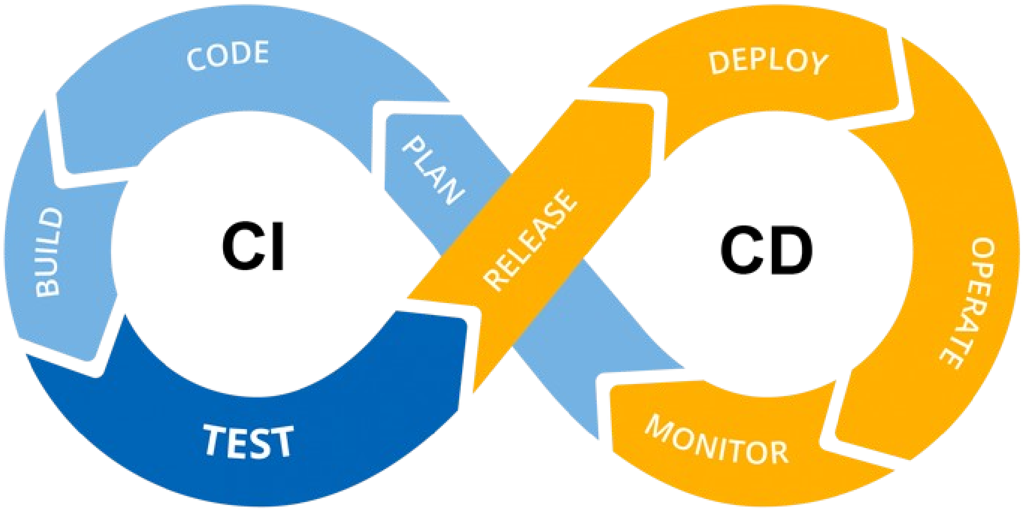
Why Agile?

We go for

- Flexible process
- Adaptive planning
- Fast response, short feedback loop

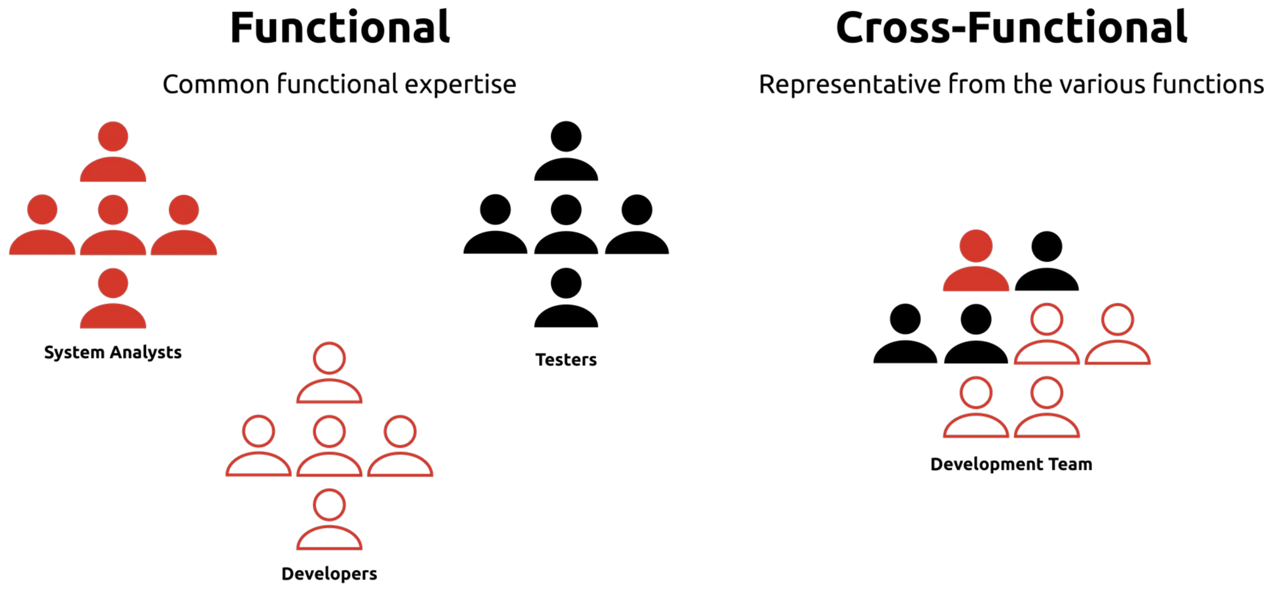
Why Agile?

- Evolutionary development
- Early delivery and continuous improvement



Why Agile?

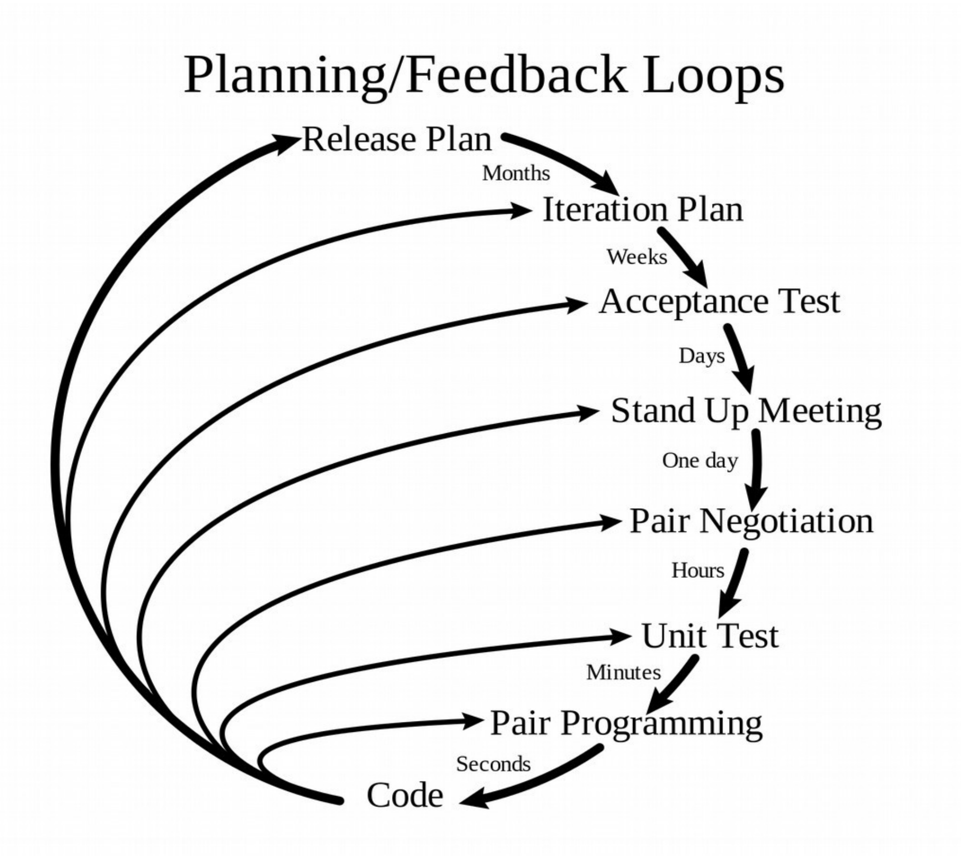
- LWPs (Lightweight Processes)
- Self-organizing cross-functional teams



XP – eXtreme Programming

Goals:

- Be responsive to changing requirements
- Improve productivity
- Improve quality
- Introduce fine-grained monitoring (frequent checkpoints)



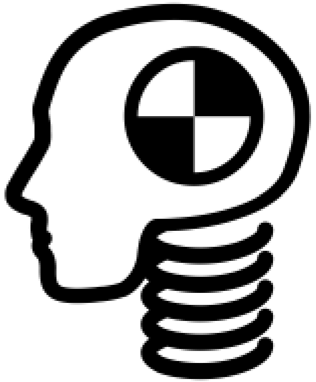
XP Coding

- Extensive **Code Reviews** or
  - May spot problems (of various levels
  - Information sharing!!!
- **Pair Programming**
  - Knowledge sharing!!!
  - Higher quality, lower error rate
  - Expensive (probably)
  - Pairs: expert-expert, expert-novice
  - Can be non-functioning (disengaged pair, no communication)

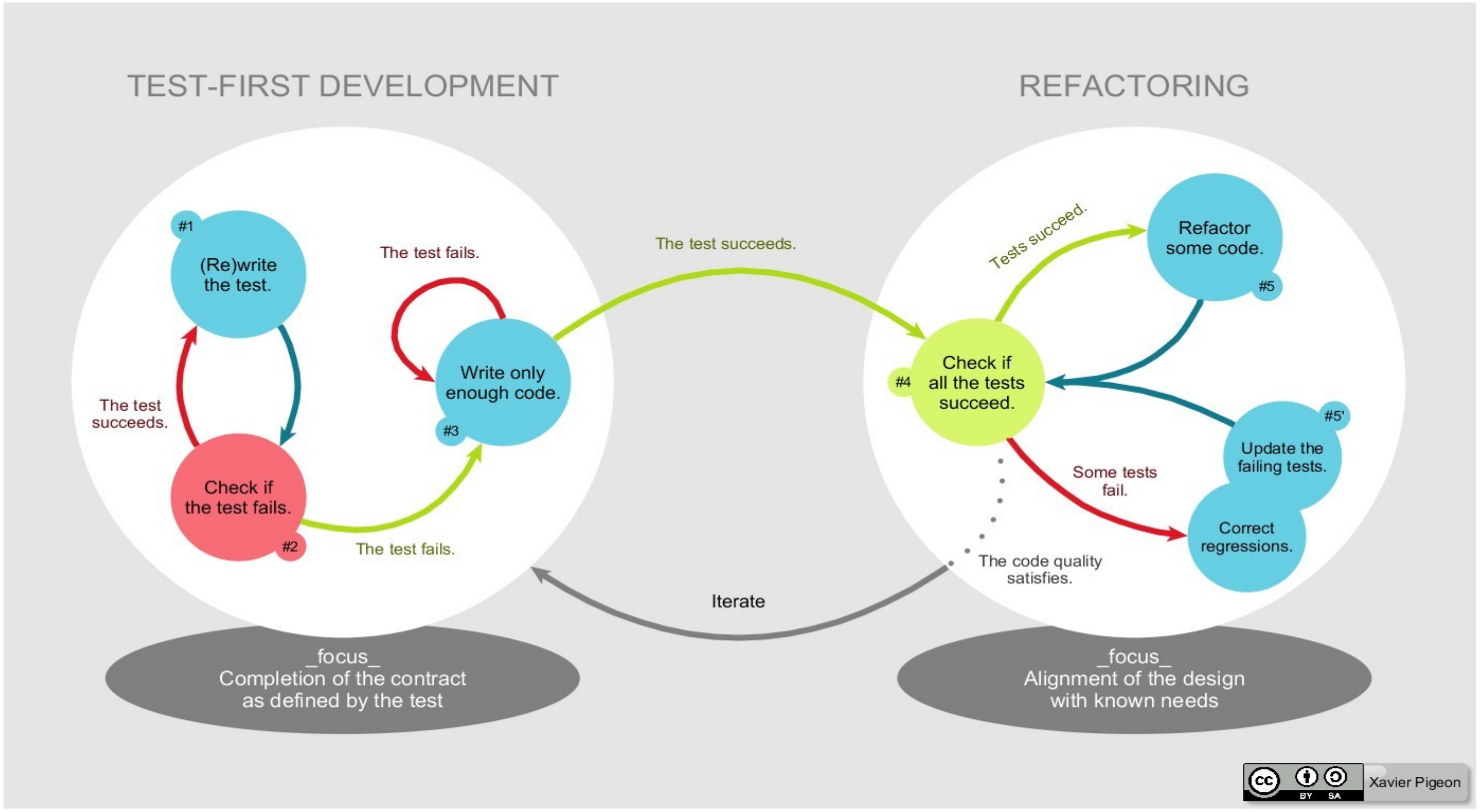
XP Testing = TDD

Test Driven Development

- Test-first approach
- Test is Documentation (no comments)
- Refactor at end (clean up mess)
- You aren’t gonna need it (KISS)
- Testing levels
- Dummy, Stub, Spy, Mock, Simulator
- Bad tests
- ATDD (Acceptance Test Driven Development)



TDD

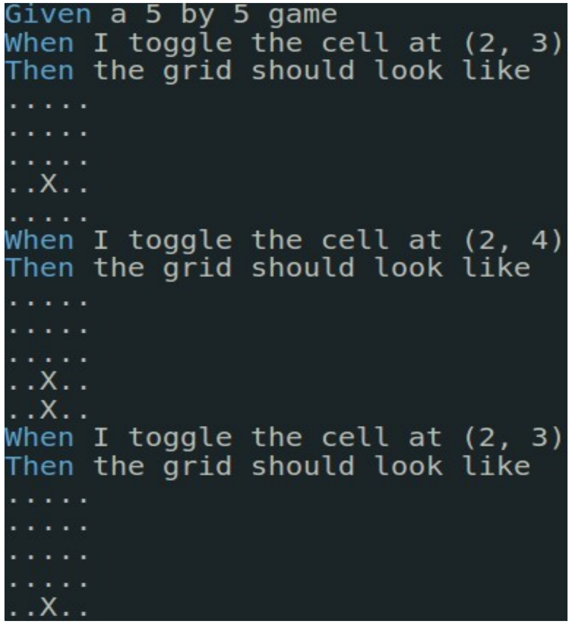

TEST-FIRST DEVELOPMENT
REFACTORING
#1
(Re)write the test.
The test fails.
The test succeeds.
Write only enough code.
#3
The test succeeds.
Check if the test fails.
#2
The test fails.
Tests succeed.
Refactor some code.
#5
#4
Check if all the tests succeed.
#5'
Update the failing tests.
Some tests fail.
Correct regressions.
The code quality satisfies.
Iterate
focus
Completion of the contract as defined by the test
focus
Alignment of the design with known needs
Xavier Pigeon

BDD

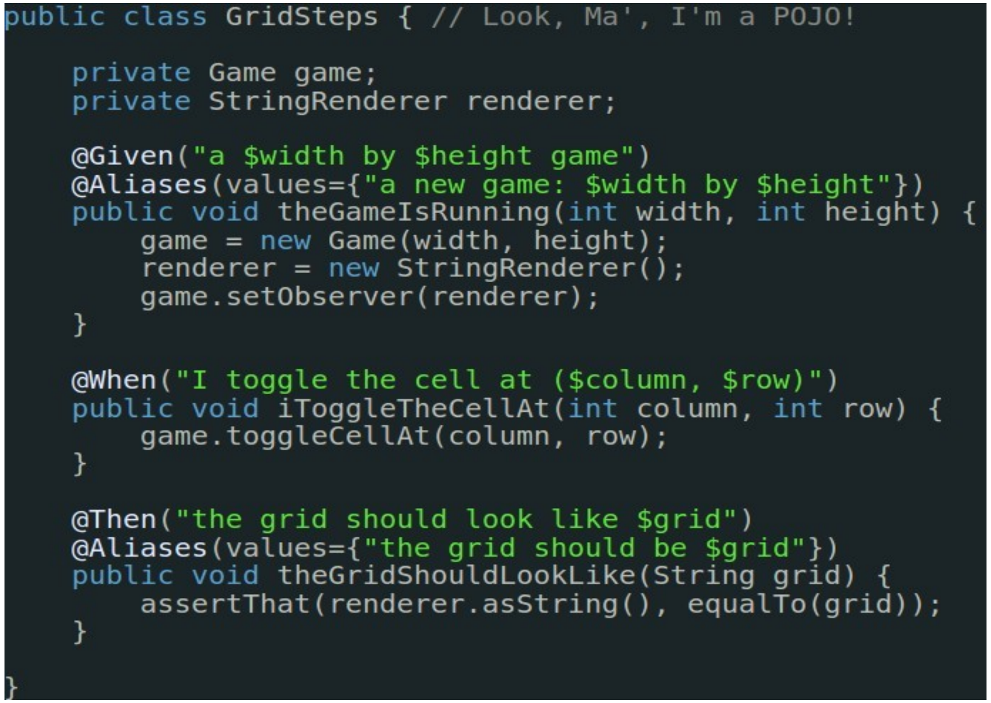
Behavior Driven Development

- BDD = ATDD + DSL
- Tooling: uses a natural language-like-DSL and auto-generates test stubs
- Test: **Acceptance Criteria**s in different **Scenario**s
- Good practice: Name tests on expected results, normal behavior (not errors or submodule names)
- Tools: Cucumber, JBehave...

BDD example DSL



```
Given a 5 by 5 game
When I toggle the cell at (2, 3)
Then the grid should look like
.....
.....
.....
..X..
.....
When I toggle the cell at (2, 4)
Then the grid should look like
.....
.....
.....
..X..
..X..
When I toggle the cell at (2, 3)
Then the grid should look like
.....
.....
.....
.....
..X..
```



```java
public class GridSteps { // Look, Ma', I'm a POJO!

    private Game game;
    private StringRenderer renderer;

    @Given("a $width by $height game")
    @Aliases(values={"a new game: $width by $height"})
    public void theGameIsRunning(int width, int height) {
        game = new Game(width, height);
        renderer = new StringRenderer();
        game.setObserver(renderer);
    }

    @When("I toggle the cell at ($column, $row)")
    public void iToggleTheCellAt(int column, int row) {
        game.toggleCellAt(column, row);
    }

    @Then("the grid should look like $grid")
    @Aliases(values={"the grid should be $grid"})
    public void theGridShouldLookLike(String grid) {
        assertThat(renderer.asString(), equalTo(grid));
    }

}
```

XP Planning Game

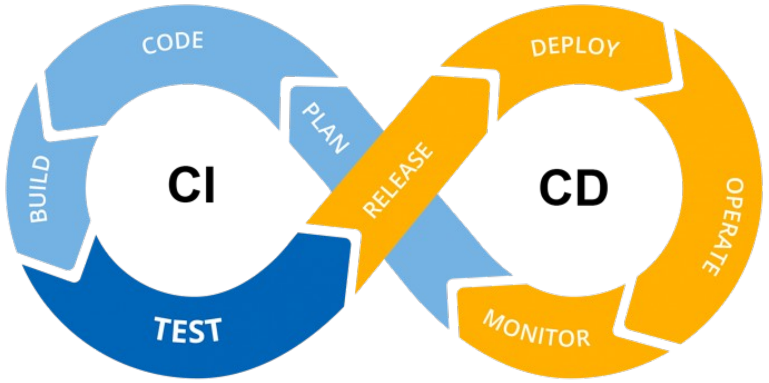
- Release Planning (~month) (w/ Customer) – *Whole Team approach*
  - Exploration phase (User Stories from Customer, estimate, split)
  - Commitment phase (Scope of next release, sort by value, risk)
  - Steering phase (plan adjustments)
- Iteration Planning (~week) (w/o Customer)
  - Exploration phase (requirements → Tasks, combine, split)
  - Commitment phase (assignment to developer)
  - Steering phase (testing, coding)

Other XP Practices

- Shared Responsibility
  - Information and Knowledge Sharing at all cost
  - Collective Code Ownership
  - Coding Standards (and enforcing tools: pep8, clang-format...)

Other XP Practices

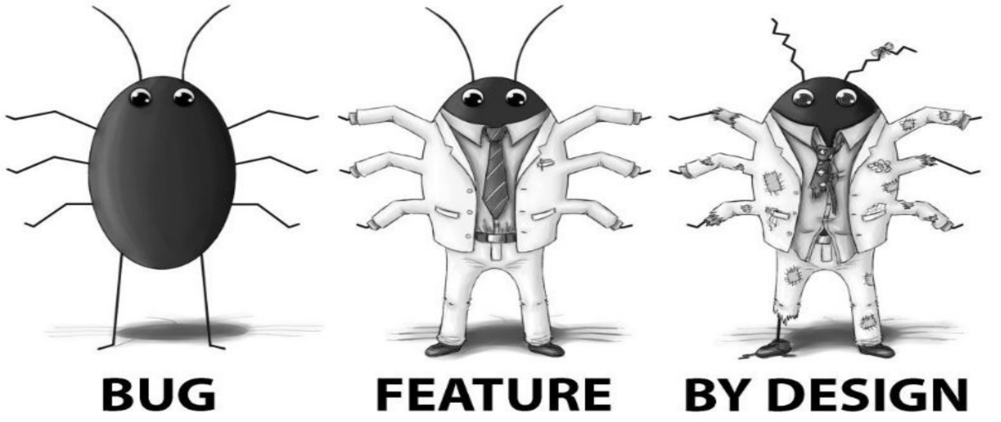
- Continuously
  - **Continuous Integration**
    - Frequent code merges
    - Automated, always running tests
    - Constant refactoring
  - Design Improvements
  - Small Releases (Release simple, release often)



Other XP Practices

- Sustainability
  - Unexpected problems minimized
  - No overtime
  - Fresh mindset (escape through refactoring)

FDD

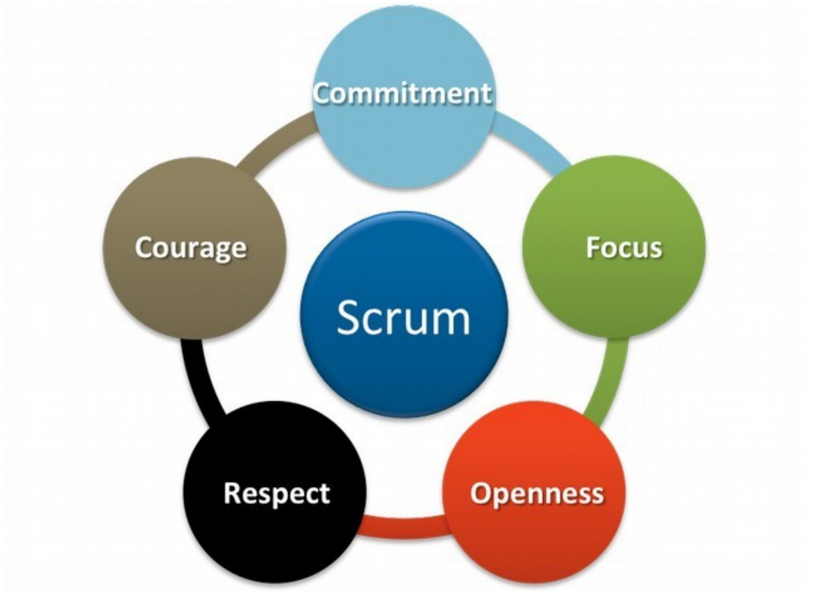


Feature Driven Development

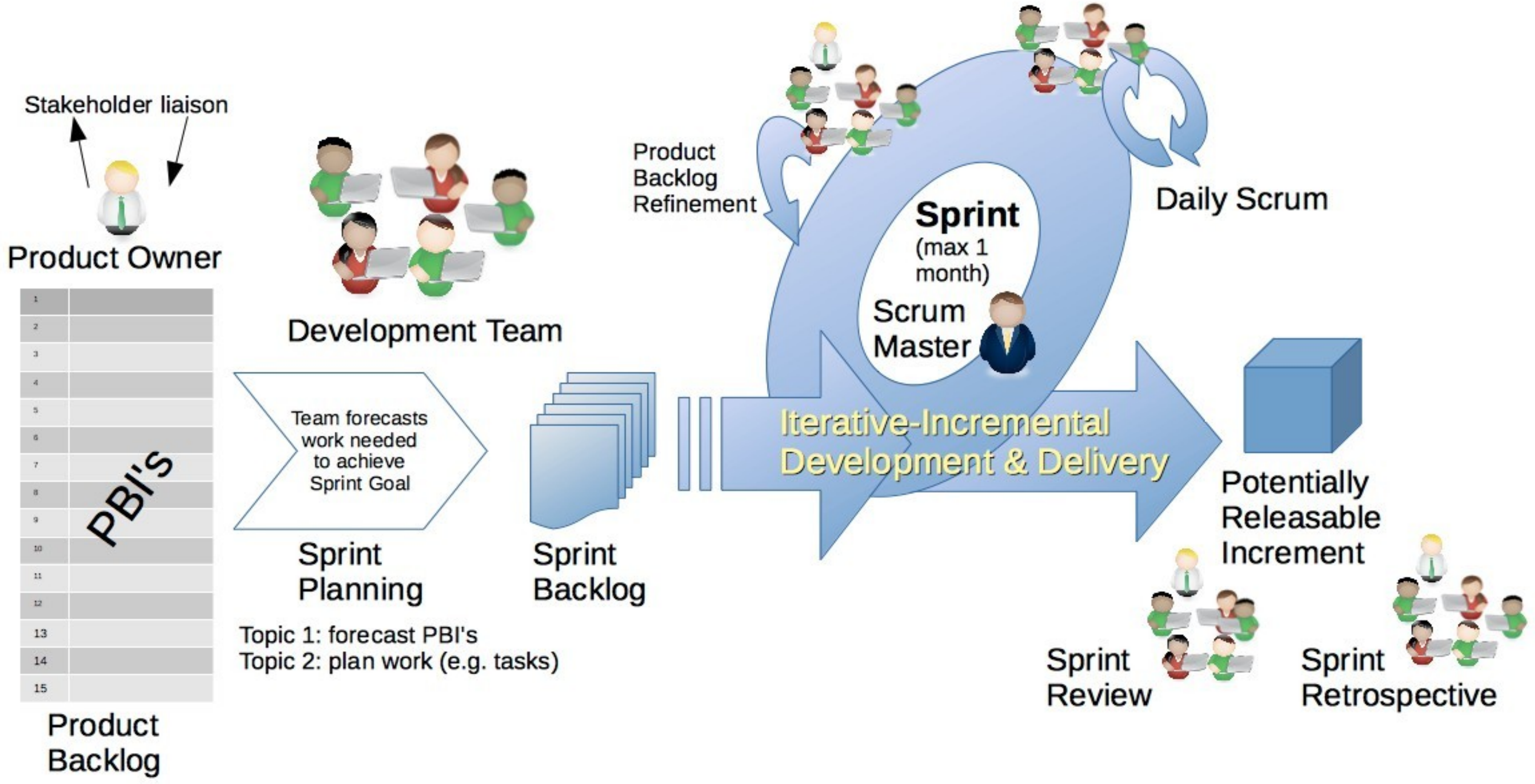
- Agile + MDD (Domain Object Modeling)
- Phases:
  - Develop overall model (walkthrough of scope)
  - Build feature list (Split longer than 2 weeks)
  - Plan by feature (Development Plan → classes, assign: one class → one developer /individual class ownership/)
  - Design by feature (chief and class owners make sequence diagrams, then class / method skeleton)
  - Build by feature (coding, testing, code review, promoting to build)
- **Feature Team**s are temporal
- Configuration Management (**Feature Switch**es)

Scrum

- Comes from Japanese manufacturing case studies in automotive and printing industries
- Cross-functional team
- Empirical / Holistic / Rugby approach → Scrum
- Goal is to increase flexibility & speed
- Decision making brought down in company structure
- Empirical process → should be
  - Transparent
  - Monitored (Inspected)
  - Adaptive
- Values are for team work

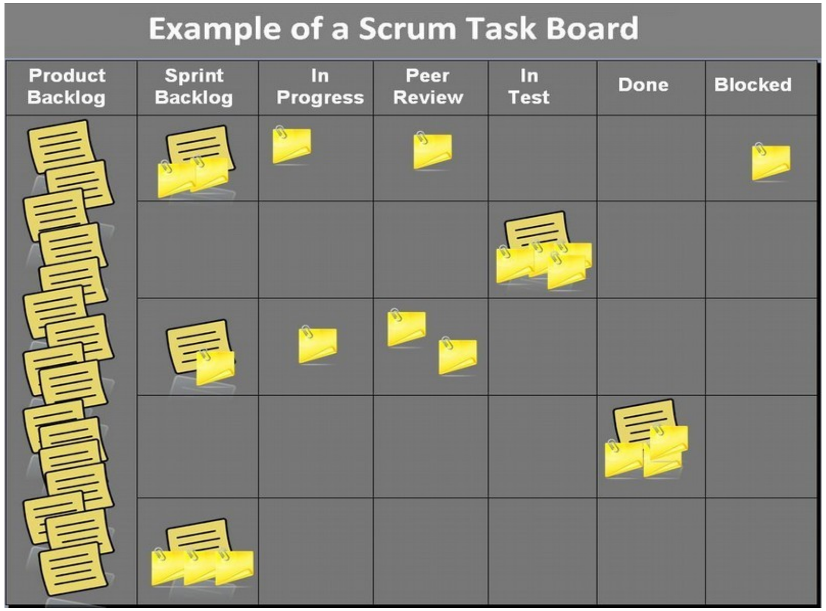


Scrum Process



Scrum Details

- Product Owner fills Product Backlog
- **Sprint**
  - Timebox 1 or 2 weeks or month
  - Emphasize on working product at the end
- Sprint **Planning**
  - Timebox 2-4 hours
  - PBIs are prioritized, selected
  - PBIs are decomposed
  - Commitment is made on filled Sprint Backlog



Scrum Details

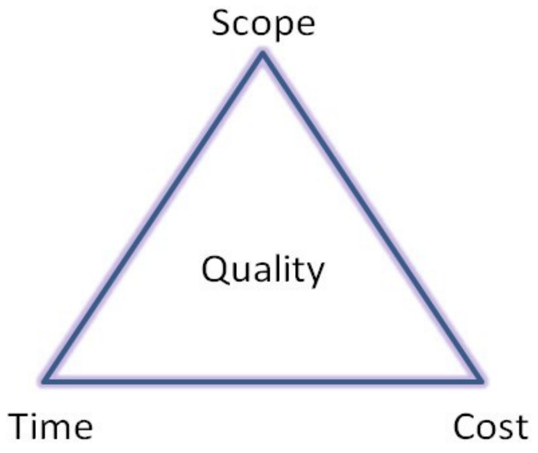
- **Stand-ups**
  - Timebox 5-10 minutes
  - Everyday: same time, same place
  - No discussion
  - Questions:
    - What did I do yesterday?
    - What will I do today?
    - Are there any problems, blocking issues? → job for Scrum Master

Scrum Details

- Sprint **Review**
  - Timebox 1 hour
  - What was completed?
  - Demo (Incomplete work doesn’t count)
- Sprint **Retrospective**
  - Timebox 1 hour
  - Questions:
    - What went well?
    - What went wrong?
  - Agreement on team process improvements

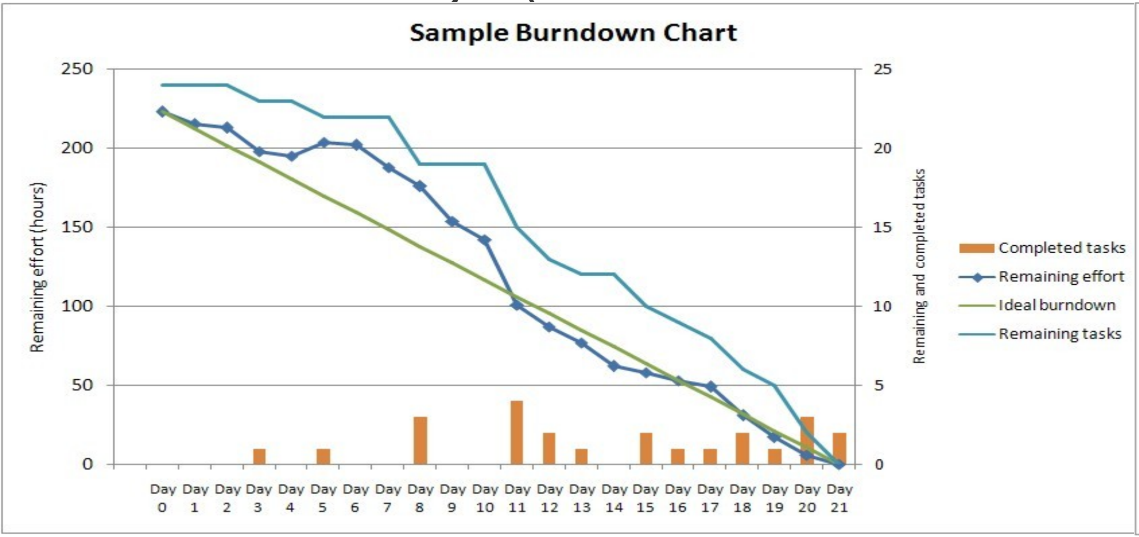
Scrum Criticism

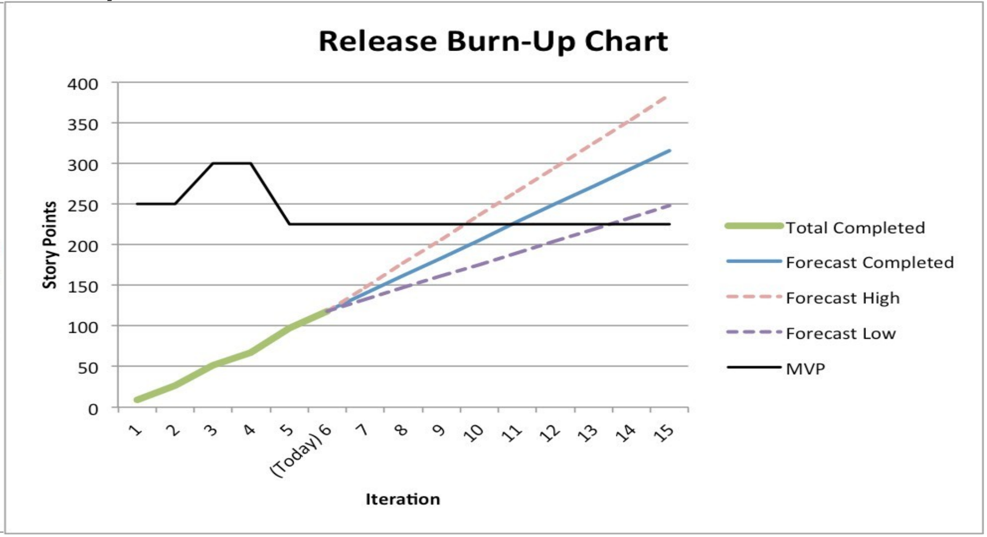
- Co-location
- Team responsibility: anyone can do anything, but cross-functional teams have specialized members
- Triple Constraint (Project Management Triangle)
  – all of them cannot be fixed



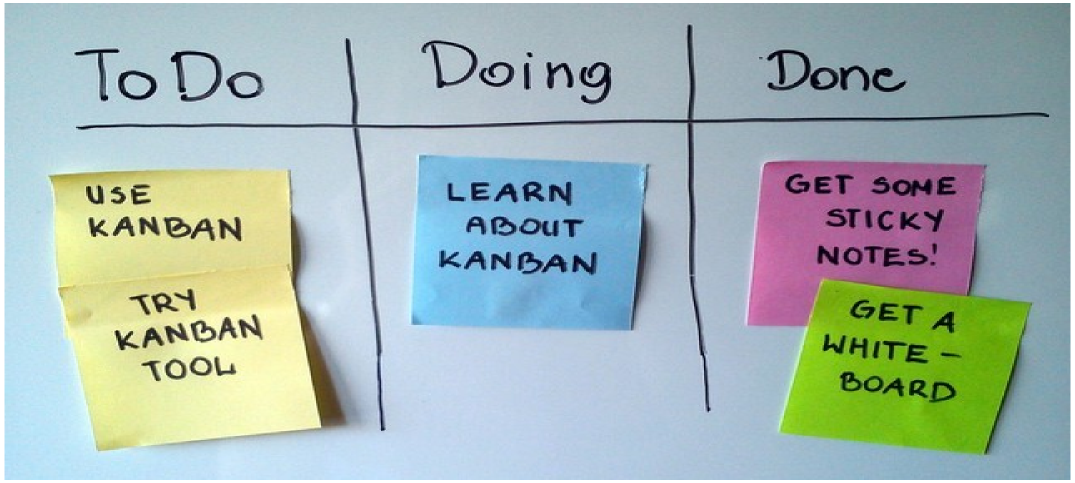
Scrum Criticism

- Measure Velocity (Agility)?
  - Statistics (commit number, fixed tickets… highly debatable)
  - Surveys (better for moral detection)



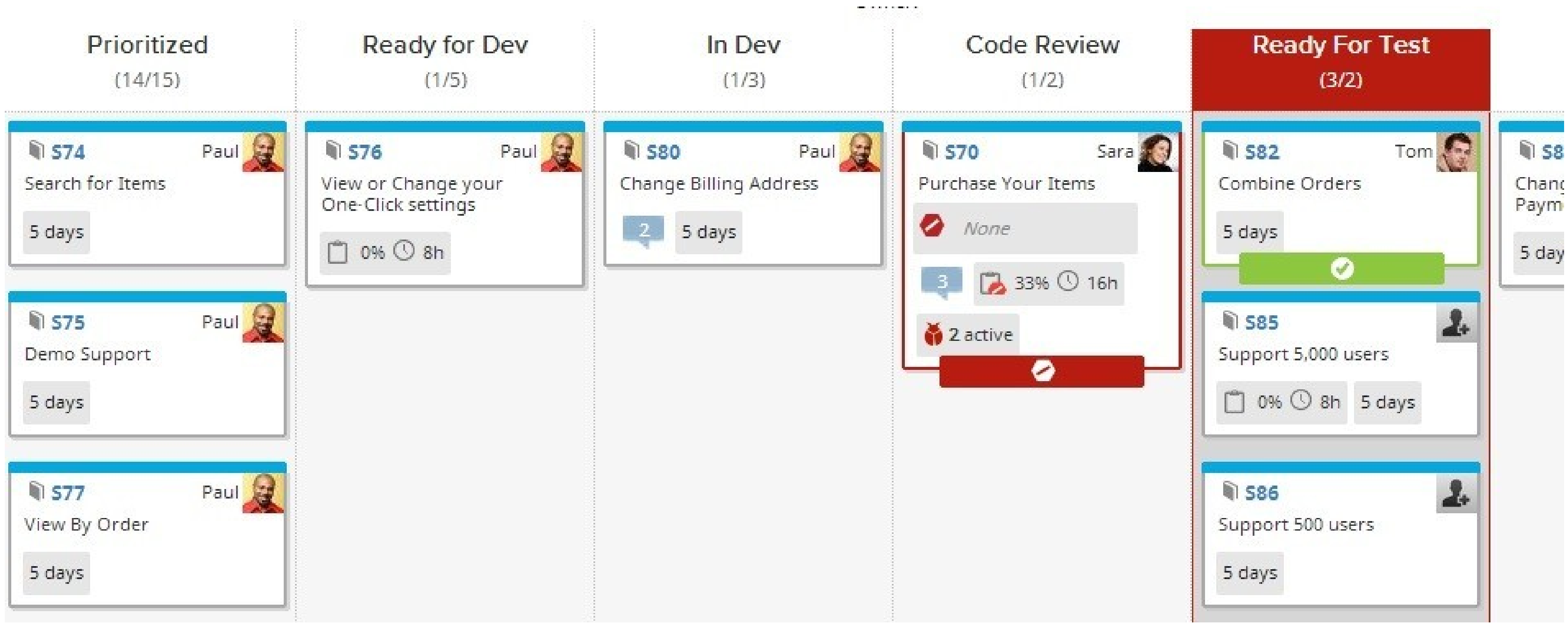


Kanban



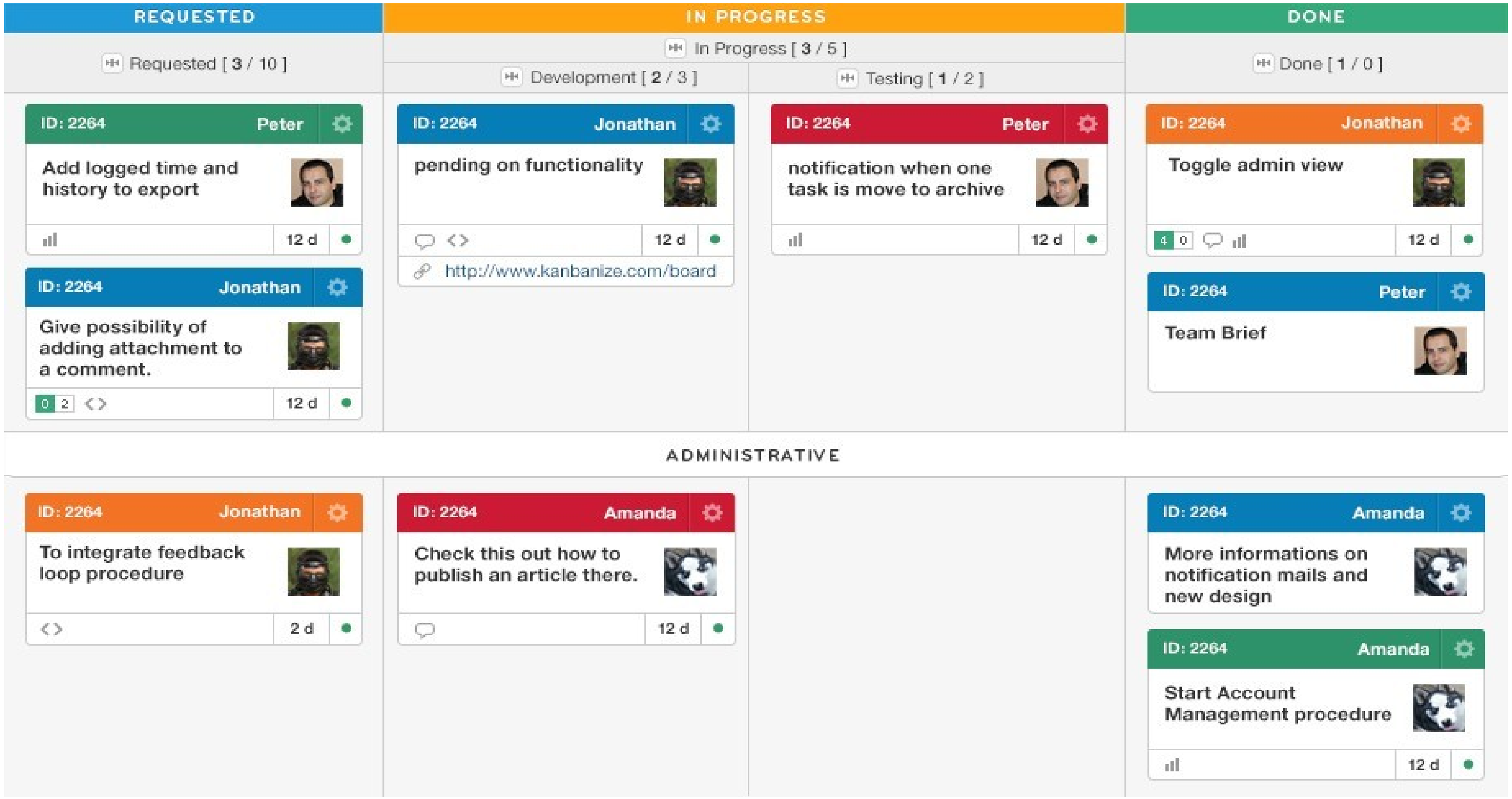
- Kanban (billboard)
- Lean manufacturing or Just-in-time manufacturing from Toyota
- A visual inventory control system for supply chain management
- Idea: software development management = supply chain management (flow management)

Kanban board



Prioritized (14/15)	Ready for Dev (1/5)	In Dev (1/3)	Code Review (1/2)	Ready For Test (3/2)	
S74 Paul Search for Items 5 days	S76 Paul View or Change your One-Click settings 0% 8h	S80 Paul Change Billing Address 2 5 days	S70 Sara Purchase Your Items None 3 33% 16h 2 active	S82 Tom Combine Orders 5 days ✓	S8 Chang Paym 5 day
S75 Paul Demo Support 5 days				S85 Support 5,000 users 0% 8h 5 days	
S77 Paul View By Order 5 days				S86 Support 500 users 5 days	

Kanban board



REQUESTED | IN PROGRESS | DONE

Requested [3 / 10]

In Progress [3 / 5]

Development [2 / 3]

Testing [1 / 2]

Done [1 / 0]

ID: 2264 Peter
Add logged time and history to export
12 d

ID: 2264 Jonathan
Give possibility of adding attachment to a comment.
0 2
12 d

ID: 2264 Jonathan
pending on functionality
12 d
http://www.kanbanize.com/board

ID: 2264 Peter
notification when one task is move to archive
12 d

ID: 2264 Jonathan
Toggle admin view
4 0
12 d

ID: 2264 Peter
Team Brief

ADMINISTRATIVE

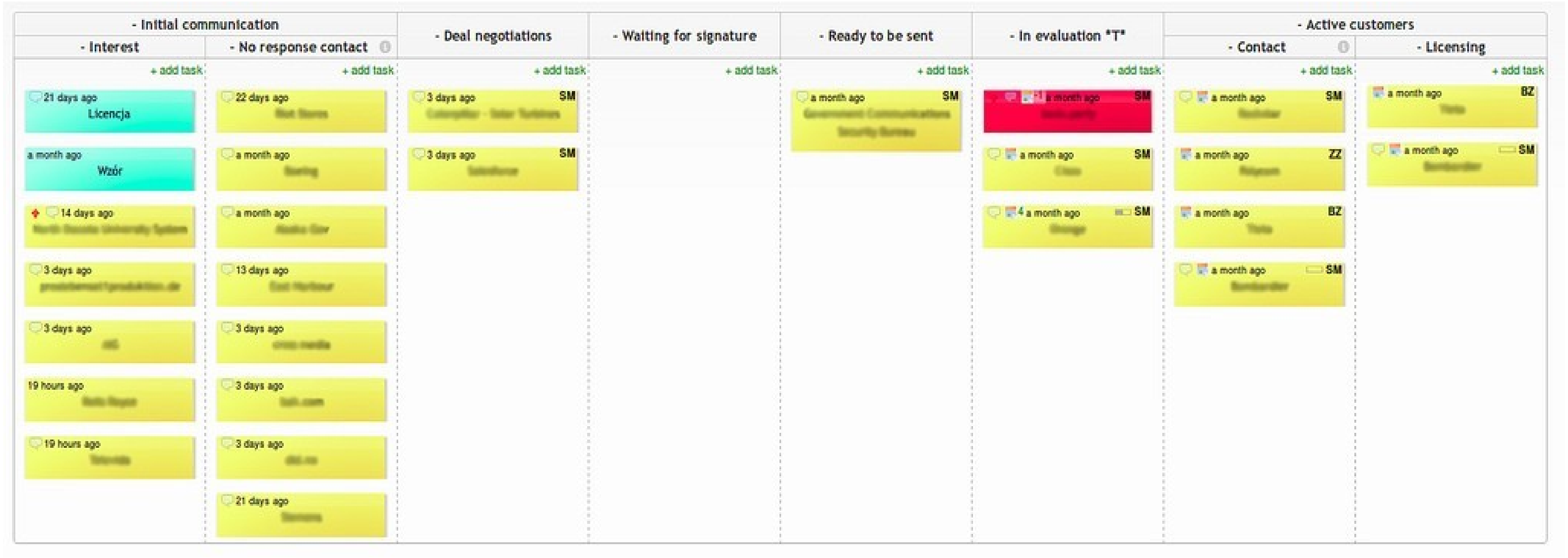
ID: 2264 Jonathan
To integrate feedback loop procedure
2 d

ID: 2264 Amanda
Check this out how to publish an article there.
12 d

ID: 2264 Amanda
More informations on notification mails and new design

ID: 2264 Amanda
Start Account Management procedure
12 d

Kanban board



- Initial communication

- Interest + add task
  - 21 days ago Licencja
  - a month ago Wzór
  - 14 days ago [redacted]
  - 3 days ago [redacted]
  - 3 days ago [redacted]
  - 19 hours ago [redacted]
  - 19 hours ago [redacted]

- No response contact + add task
  - 22 days ago [redacted]
  - a month ago [redacted]
  - a month ago [redacted]
  - 13 days ago [redacted]
  - 3 days ago [redacted]
  - 3 days ago [redacted]
  - 3 days ago [redacted]
  - 21 days ago [redacted]

- Deal negotiations + add task
  - 3 days ago SM [redacted]
  - 3 days ago SM [redacted]

- Waiting for signature + add task

- Ready to be sent + add task
  - a month ago SM [redacted]

- In evaluation "T" + add task
  - a month ago SM [redacted]
  - a month ago SM [redacted]
  - 4 a month ago SM [redacted]

- Active customers

- Contact + add task
  - a month ago SM [redacted]
  - a month ago ZZ [redacted]
  - a month ago BZ [redacted]
  - a month ago SM [redacted]

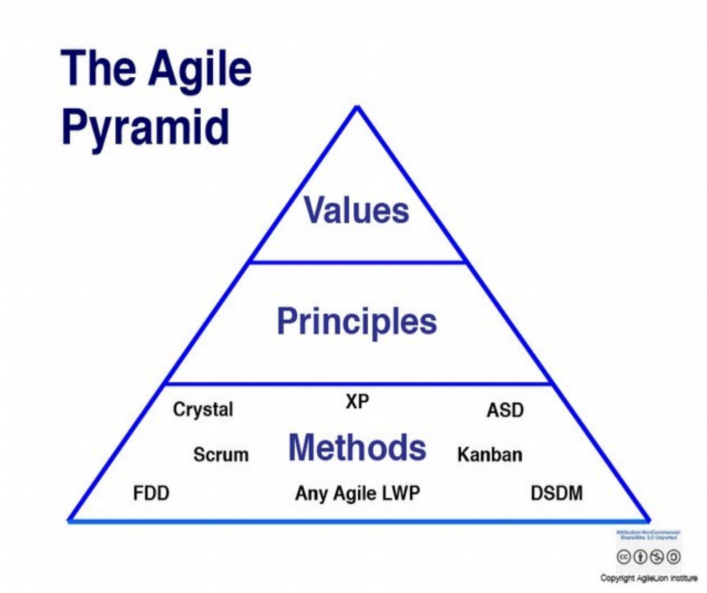
- Licensing + add task
  - a month ago BZ [redacted]
  - a month ago SM [redacted]

Lean / Kanban Practices

- Eliminate Waste
  - Useless work – Muda
  - Overburden – Muri
  - Unevenness – Mura
- Previous include
  - Task switching overhead
  - Unfinished work
  - Waiting
- Team members pull work (not pushed onto them)
- Quality is fixed (not Scope or Schedule)
- WIP Limit = Work In Progress Limit

Agile Thoughts

- Communication is important (co-located or not)
- Trust towards individuals, motivate
- Working software instead of documents
- Involve customer
- Be flexible
- Short feedback, frequent releases
- Self-organizing teams are best
- Simplicity – the art of maximizing the amount of work NOT done
- Evolve everything
- Sustainable development



Agile Criticism

- Lack of Plan vs Too Much Preparation
- Fixed time, scope, cost, quality = mission impossible
- Bad management
  - Manager(s) does not understand process (scrum master is a developer)
  - No support from other departments
  - No product owner (just a “clever” developer)
  - Assign tasks early or from outside
- No sufficient automation
- No escape from technical debt
- Too much in iteration vs Lost focus (other work)