Agile Principles and Practices

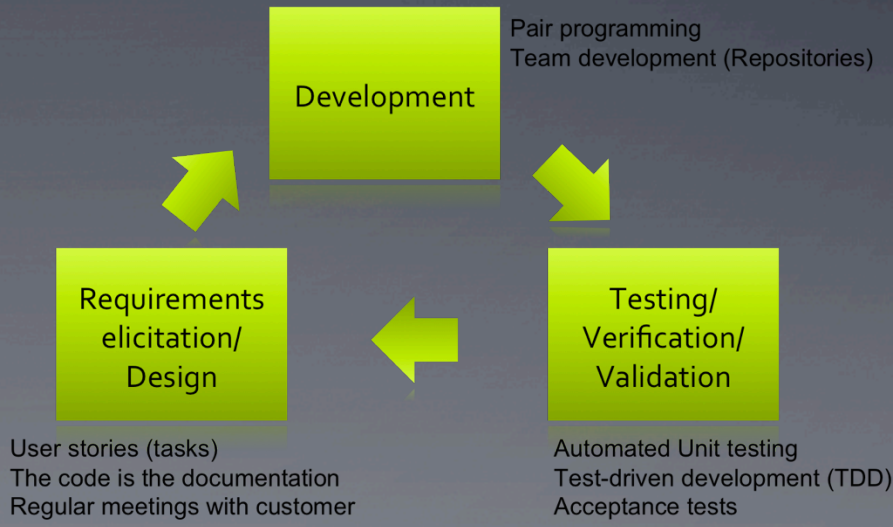
Emil Alégroth

In administrive news

- Groups on the course wiki:
- https://github.com/oerich/EDA397/wiki
- If you have not signed up, please do so in the break!
- Note that there are 11 groups! You have to scroll down to see 11. Sorry.

Recap from Tuesday

- Efficient software development requires a development process
- Traditional processes
  - Not efficient when requirements change
- Agile software development
  - Processes for efficient software development in the context of change.



http://agilemanifesto.org

- Began as a provocation: Plan-driven development did not safe the Software world…
- Now a very serious movement, well adapted in industry.
- There are a couple of established agile methods: How to integrate these values in everyday software development

Agile Principles

1. Early and continuous delivery of valuable software
2. Welcome changing requirements, even late
3. Deliver working software frequently
4. Business people and developers must work together
5. Build projects around motivated individuals
6. Face-to-face communication is most effective and efficient
7. Working software is the primary measure of progress
8. Sustainable development
9. Continuous attention to technical excellence and good design
10. Simplicity is essential
11. Self-organizing teams
12. Regular reflection

eXtreme Programming practices

- The 12 practices of XP

Planning Game

- Release planning (Customer and developers)
  - Create storycards with most important requirements
  - Commit to development and set deadline (1 week)
  - If required: adjust, add or change included requirements
- Iteration planning (Developers)
  - Break requirements into tasks
  - Assign tasks to developers and estimate time (Planning poker)
  - Integrate tasks and verify conformance to original storycard
- Aims to steer the project rather than to set a fixed delivery

Small releases

- Every release
  - … should be as small as possible
  - … should contain the most valuable business requirements
  - … has to make sense as a whole
  - … should (in a real project) be delivered every 4-8 weeks (rather than 6-12 month)
- Provide customer with confidence in the progress of the project.

Metaphor

- A story that captures the use of the system.
- Used for naming conventions and to provide domain knowledge.
- Example: Library system
  - Books can be borrowed: `class borrowBook()`
  - A book can be overdue: `private bool overdue = false;`
  - If a book is returned: `returnedBook(Book name){name.setAvailable()}`

Simple Design

- Every time new code is added ask yourself
  - Can this added functionality be developed in a simpler way?
  - If yes: Refactor the code and test that new code behaves as old code
  - Possible changes:
    - Reduce line-count/modularize code
    - Reduce/change classes and methods
    - Lower the ordo-complexity
- Keep It Simple Stupid (KISS method)
- By keeping the code simple it becomes easier to maintain and update, i.e. Change!

Testing

- All classes/methods that are productive and could possibly break should be tested automatically!
  - Unit testing: A test that tests a specific component through input to which there is an expected output
  - Visual GUI Testing: A test that executes against the graphical user interface to test the features of the system.
- Test first (Test-driven development)
  - Unit tests can be written before the code
  - Requires you to think carefully about the functionality
  - Provides test coverage

Refactoring

- Modify the code to improve its qualities
  - Simplicity of design
  - Readability
  - Modifiability
  - Testability
  - Performance
  - Etc.
- Requires good test coverage in order to ensure that changes did not break the system
- Investment for future development (Has occurred in the course in previous years)
  - Counteracts the lack of rigorous system design

Pair Programming

- Work in pairs
  - Driver: Writes the code
  - Navigator: Does real-time code review, provides strategic assistance regarding the solution, e.g. how to KISS it ;)
- Pairs should change often
  - Spread knowledge of code among the team
  - Experience/knowledge-sharing
- The most experienced person should be the navigator!
  - Often costly but an investment!

Collective Code Ownership

- Everyone is responsible for the code!
  - If you see a possible improvement, change the code!
- No such thing as HIS/HER code OR HIS/HER responsibility!
  - You work as a team
- Barriers (Experience from previous years)
  - Culture
  - Knowledge/experience

Continuous Integration

- Integrate code often! Every few hours (1 day at the most)
- After each integration regression test! (Run the automated tests)
- Automated integration helps
  - However, you are working with GIT
  - Different branches at once
  - Merge problems
  - Must be handled before integration

Sustainable pace (aka 40h week)

- In practice: 40 hours every week
- Prevents exhaustion and thereby lack of motivation
- Overtime not permitted!
- Could be used in the course if you can plan hours where you all sit and work together.

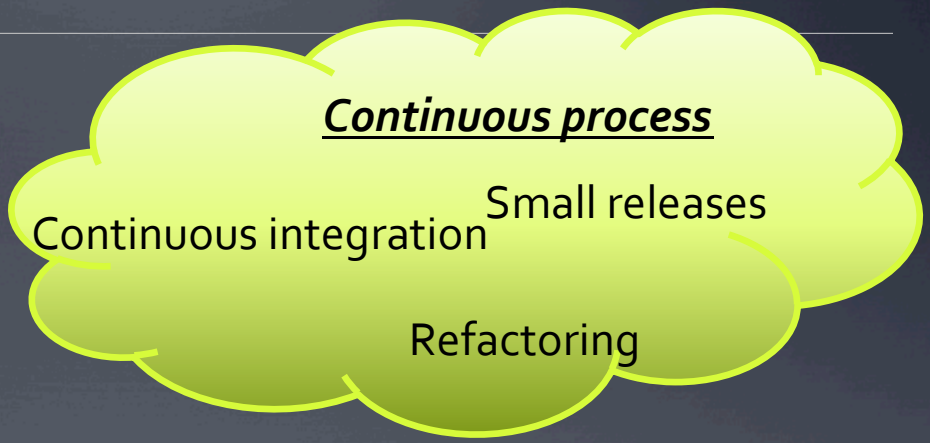
On-Site Customer

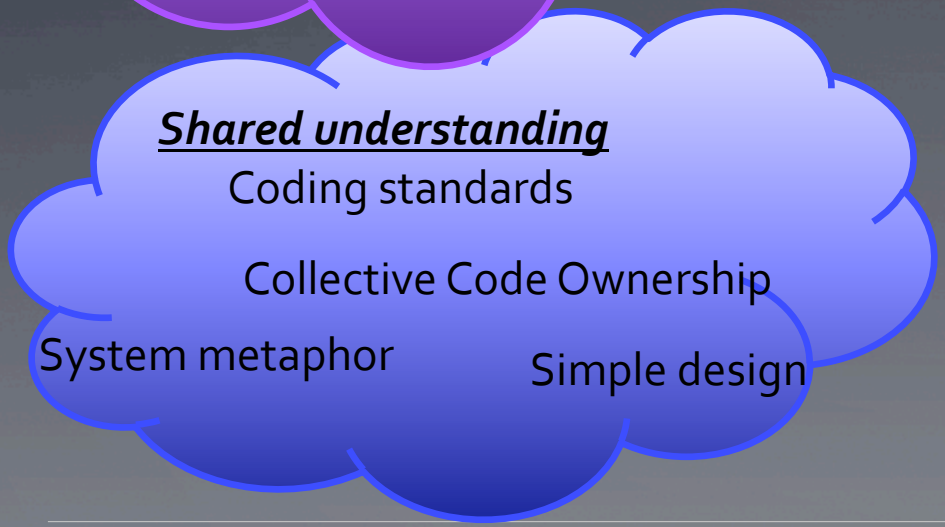
- Real customer in the room
  - Answer questions quickly (..might revise the answer later)
  - If no customer available: Customer proxy
- Quick answers are more important than correct answers!
  - Keep development moving
- In the course (and in reality) this practice is hard to achieve!
  - However, frequent acceptance tests!

Coding Standards

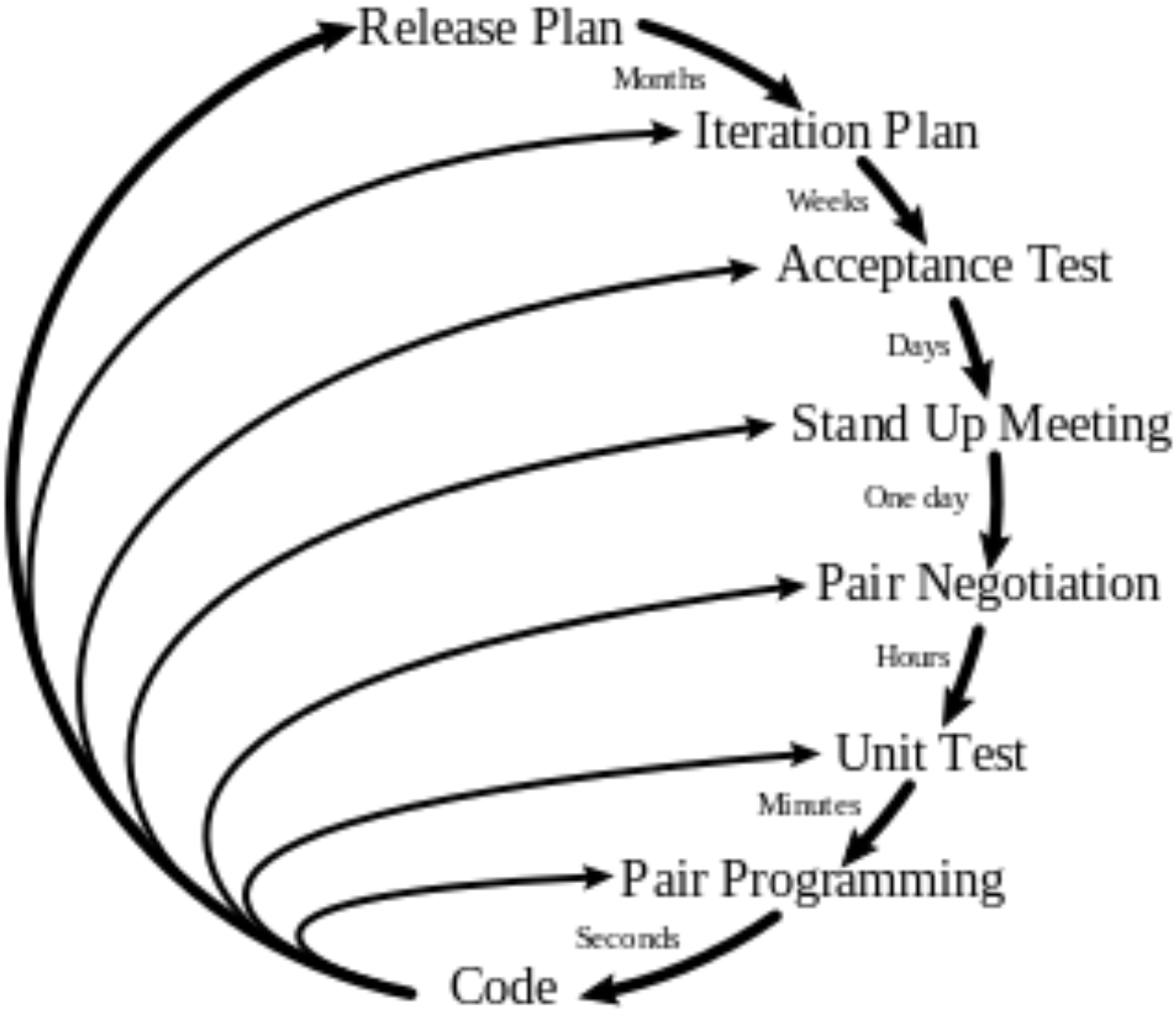
- Different people in different constellations working with the same code… MAYHEM!
- Coding standards makes sure your code is consistent
- Makes it easier to
  - Maintain/refactor the code
  - Find possible improvements
  - Etc.
- Example:
  - Public class{}
  - Or
  - Public class
  - {}

XP Practices





XP Planning/Feedback Loops



XP Principles

Core

- Rapid feedback
- Assume simplicity
- Incremental change
- Embracing change
- Quality work

Less central

- Teach learning
- Small initial investment
- Play to win
- Concrete requirements
- Open, honest communication
- Work with people's instincts, not against them
- Accepted responsibility
- Local adaptation
- Travel light
- Honest measurement

SCRUM

- Roles
  - Product manager
  - Development team
  - Scrum master
- Events
  - Sprint
  - Meetings
    - Sprint planning meeting
    - Daily scrum meeting
    - End meetings
- Artifacts
  - Product backlog
  - Spring backlog
  - Burndown chart

Scrum Roles

- Product owner (Customer representative)
  - Communicator
    - Voice of customer – Face of development team
  - Ensures that value is developed
  - Product owner != Scrum master
- Development team
  - 3-9 people
  - Self-organizing
  - Cross-functional
- Scrum master
  - Not a project manager!
  - Ensures the process is followed
    - Educate, support, plan
    - Bring coffee! Dry-cleaning, etc.

Events

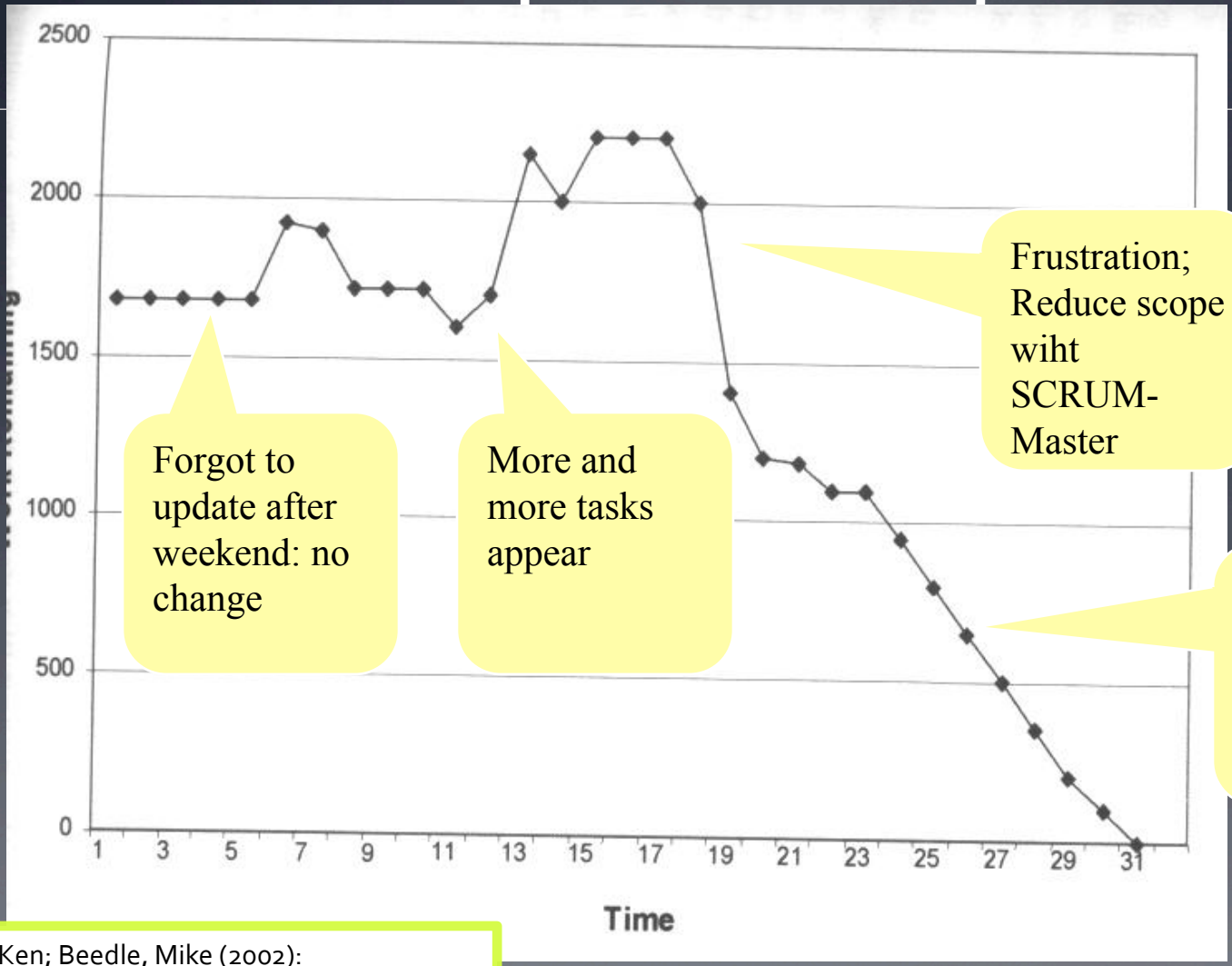
- Sprint (Iteration)
  - One to two weeks
  - Sprint planning meeting – Sprint retrospective meeting
  - End of sprint: Work/Activities are DONE!
- Sprint planning meeting
  - Select work to be done (Sprint backlog)
  - Allocate work time, etc.
- Daily scrum (stand-up meeting)
  - 15 minutes max!!!
    - What did I do yesterday?
    - What will I do today?
    - What are my problems?
- End meetings
  - Sprint review meeting
  - Sprint retrospective meeting

Artifacts

- Product backlog (Product)
  - Ordered list of
    - Features, bugs, technical work, etc.
    - Requirements!
  - Different formats (usually stories)
  - Each requirement has a cost (sometimes Fibonacci)
  - Managed by Product owner
- Sprint backlog (Iteration)
  - Subset of product backlog
  - No changes are allowed to the sprint backlog during a sprint!
- Burndown charts (Sprint and release)
  - Visualize how items from the backlog(s) are taken care of

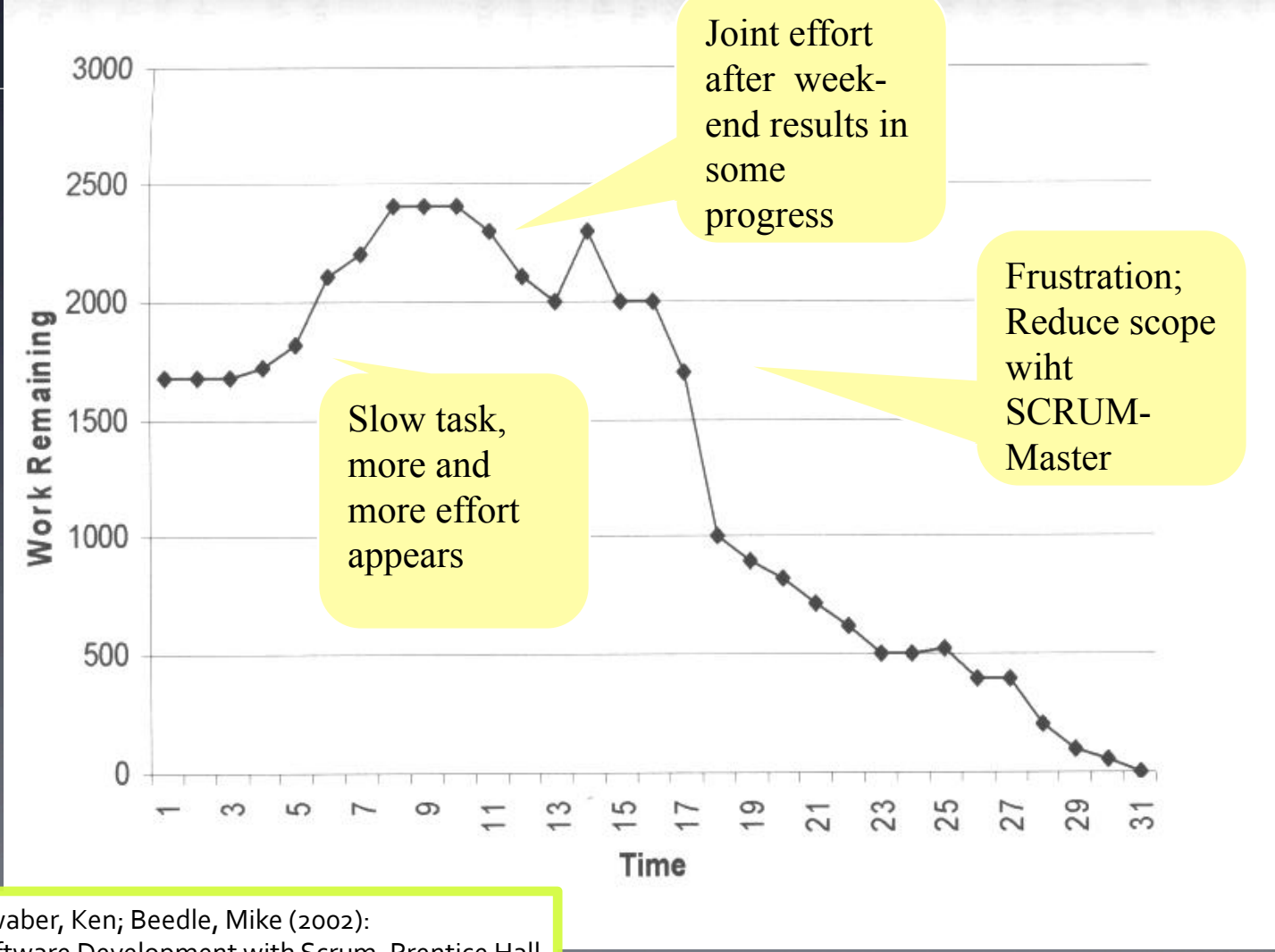
Scrumboard and burndown chart

Assess Sprint Progress



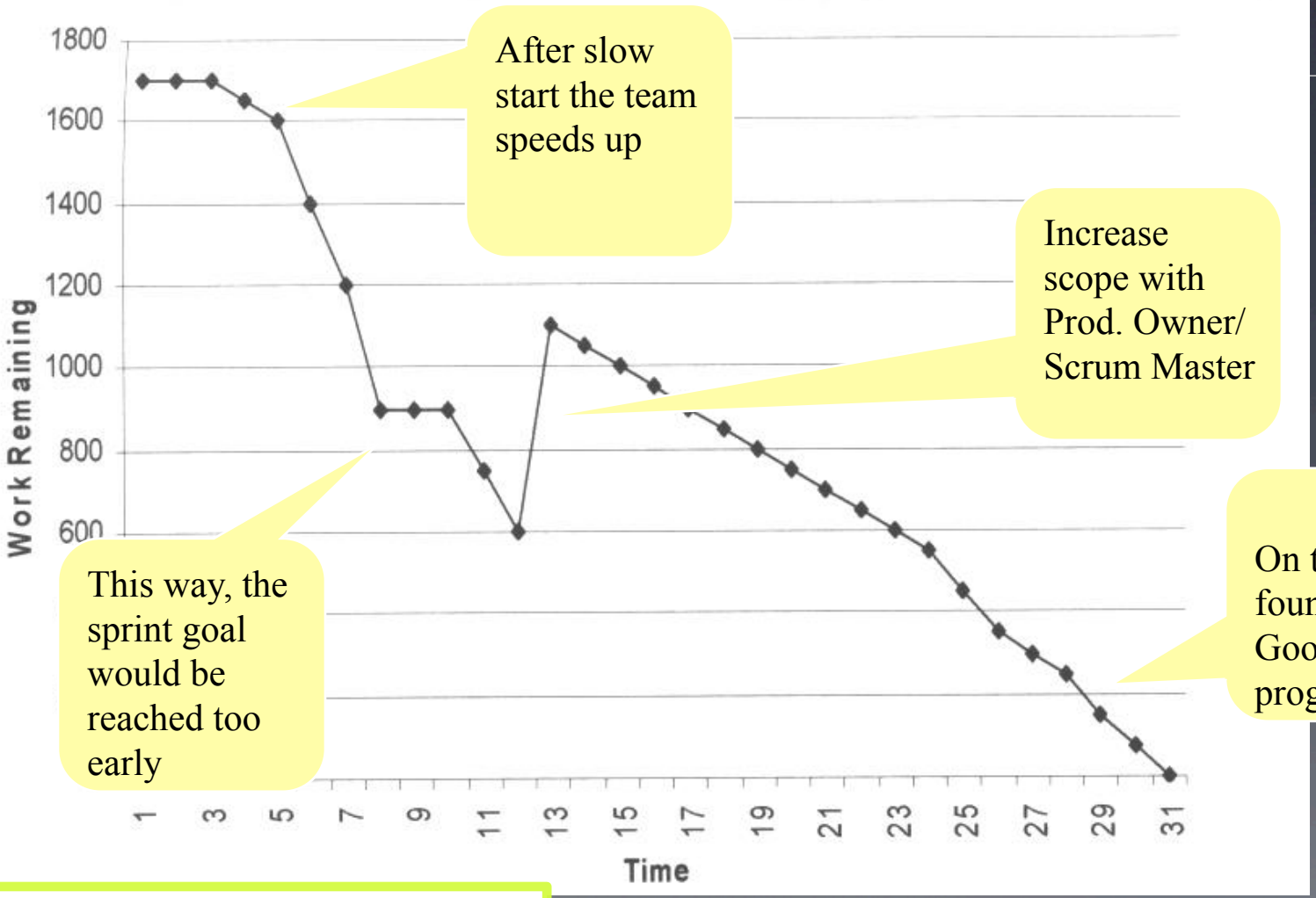
c.f. Schwaber, Ken; Beedle, Mike (2002):
Agile Software Development with Scrum. Prentice Hall.

Assess Sprint Progress



C.f. Schwaber, Ken; Beedle, Mike (2002):
Agile Software Development with Scrum. Prentice Hall.

Assess Sprint Progress

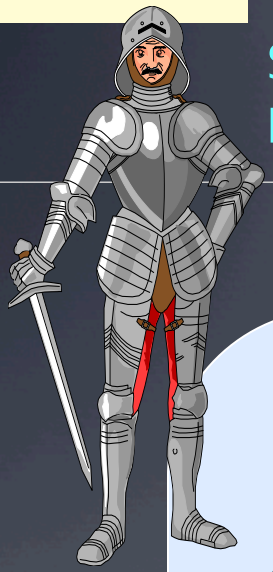


C.f. Schwaber, Ken; Beedle, Mike (2002):
Agile Software Development with Scrum. Prentice Hall.

Scrum

Product Owner

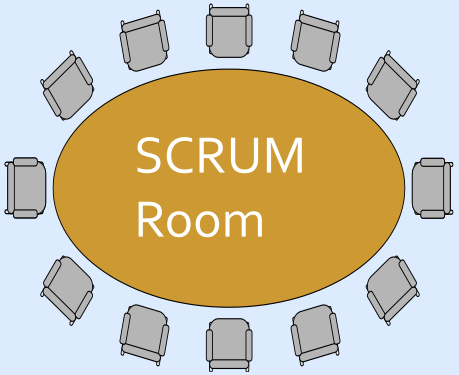
Product Backlog (prioritized)



SCRUM Master

Others

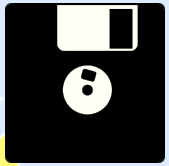
SCRUM team 7+/-2



SPRINT Backlog

SPRINT: 30 days

Increment



Daily SCRUM
15 min.

Why does SCRUM work?

- Self-organizing teams
  - Ownership
- Multi-Learning
  - Between group, organization, and individual
- Subtle control
- Constant learning
  - Experienced developers in new teams

Risk management

- Risk: Customer unhappy
  - Show working system often
- Risk: Incomplete feature set
  - Prioritize: If something is missing, it is not important
- Risk: Bad estimation
  - Daily updates during SCRUM
- Risk: Lack of experience with Development cycle
  - Test early and execute repeatedly
- Risk: Changes in performance estimation
  - No impact on Sprint

c.f. Schwaber, Ken; Beedle, Mike (2002):
Agile Software Development with Scrum. Prentice Hall.

SCRUM vs. XP

- XP is often hard to introduce
- SCRUM is easy to introduce (according to Schwaber)
- Best-practice: Combine!
  - SCRUM organizational shell: *Day-to-day management*
  - XP method of implementation
  - Shared values with XP
    - Quickly generate executable code
    - Facilitate communication

Summary SCRUM

- SCRUM is a management process
  - Can be used around XP
  - Or other approaches
- Overlap with XP, but differences exist
  - Similar values
  - Different practices
  - Partly complement each other
    - Management of development
- Not as much impact as XP, easier to introduce
- Strengths
  - Information flows not only in one direction
  - Multiple feedback cycles stabilize system development (project)

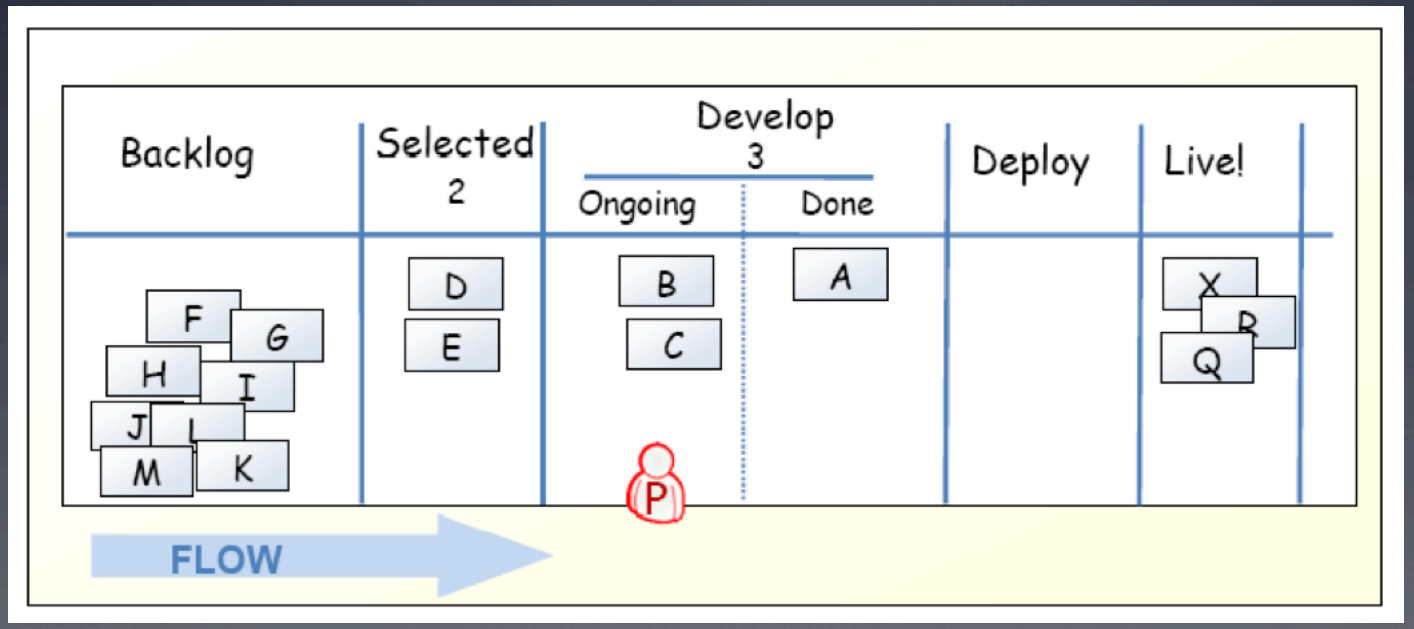
Scrum Principles

- Reflection
  - Stop and review product & process
- Self-correction
  - Based on reflection
- Visibility
  - Everything is visible (=known) for all stakeholders, e.g. plans, schedules, issues, …

Kanban core practices

- Visualize
  - Visualization of workflow allows to understand and improve it
- Limit Work-in-Progress
  - Limit the amount of workitems for each step
  - Introduce a pull-system
- Manage flow
  - Measure how workitems flow through the process and understand, if a change improves the situation
- Make policies explicit
  - Ways of working, organization, etc
- Implement feedback loops
  - Understand (as a team) how good the process is working
- Improve collaboratively, evolve experimentally
  - Whole team needs to share a theory on why (small) change helps

Kanban-Board



Limit work in progress

- Prevent context switching
  - Reduce multi-tasking
  - Perform tasks sequentially yields results sooner
    - From requirement to working feature!
- Maximize throughput
  - A fewer number of completed functions is better than many half-finished functions.
- Enhance teamwork
  - Promotes working together to get things done
  - Increases cross-functionality

WIP Strategy

- Start with some initial value
  - Small constant (1-3)
  - number of developers
  - number of testers
- Measure the cycle time
  - Average time of one piece full cycle flow
- Change limit to decrease cycle time

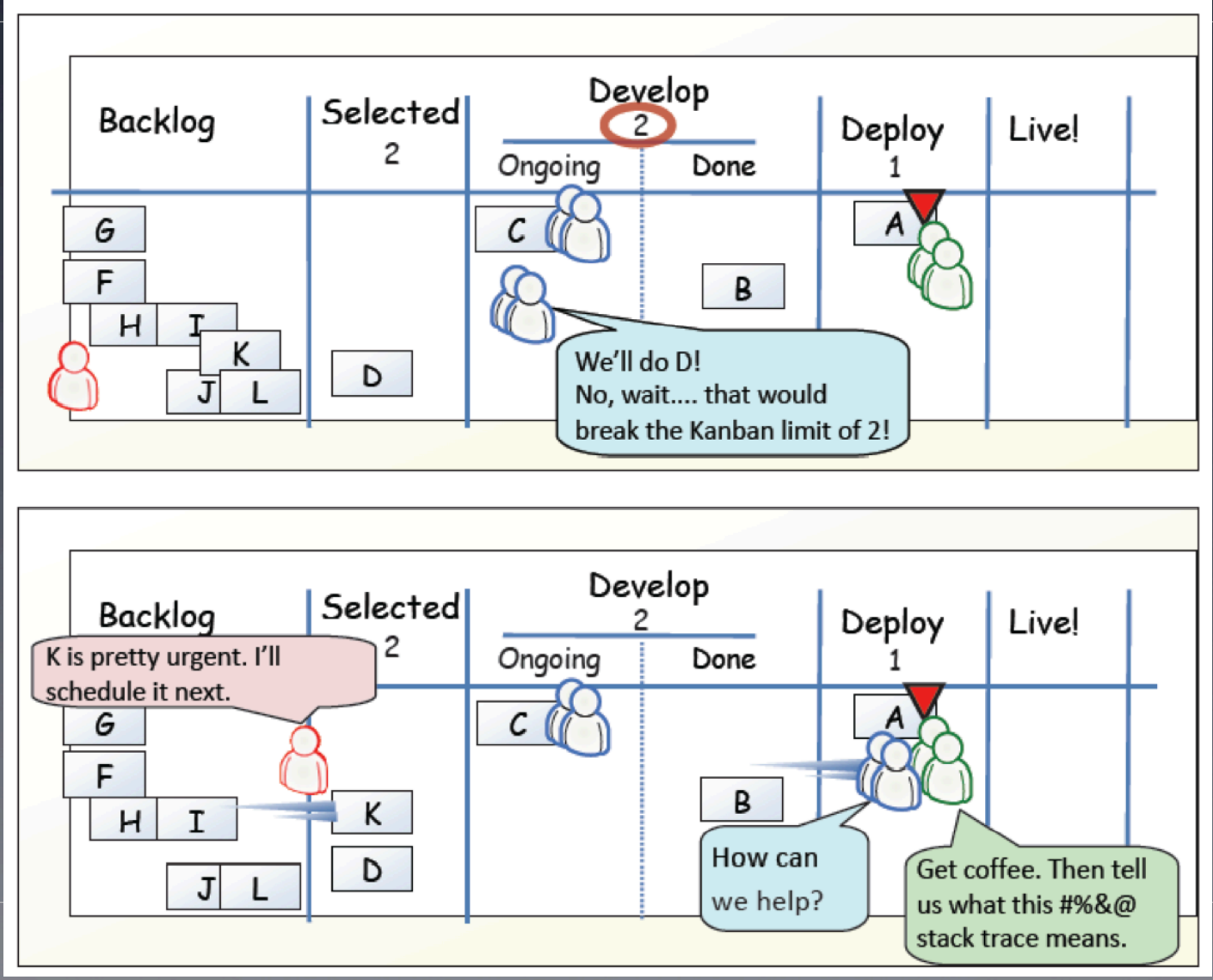
Idle Members

- Can you help progress in existing work? – Work on that.
- Don't have the right skills? – Find the bottleneck and work to release it.
- Don't have the right skills? – Pull in work from the queue.
- Can't start anything in the queue? – Check if there is any lower priority to start investigating.
- There is nothing lower priority? – Find other interesting work (refactoring, tool automation, innovation).

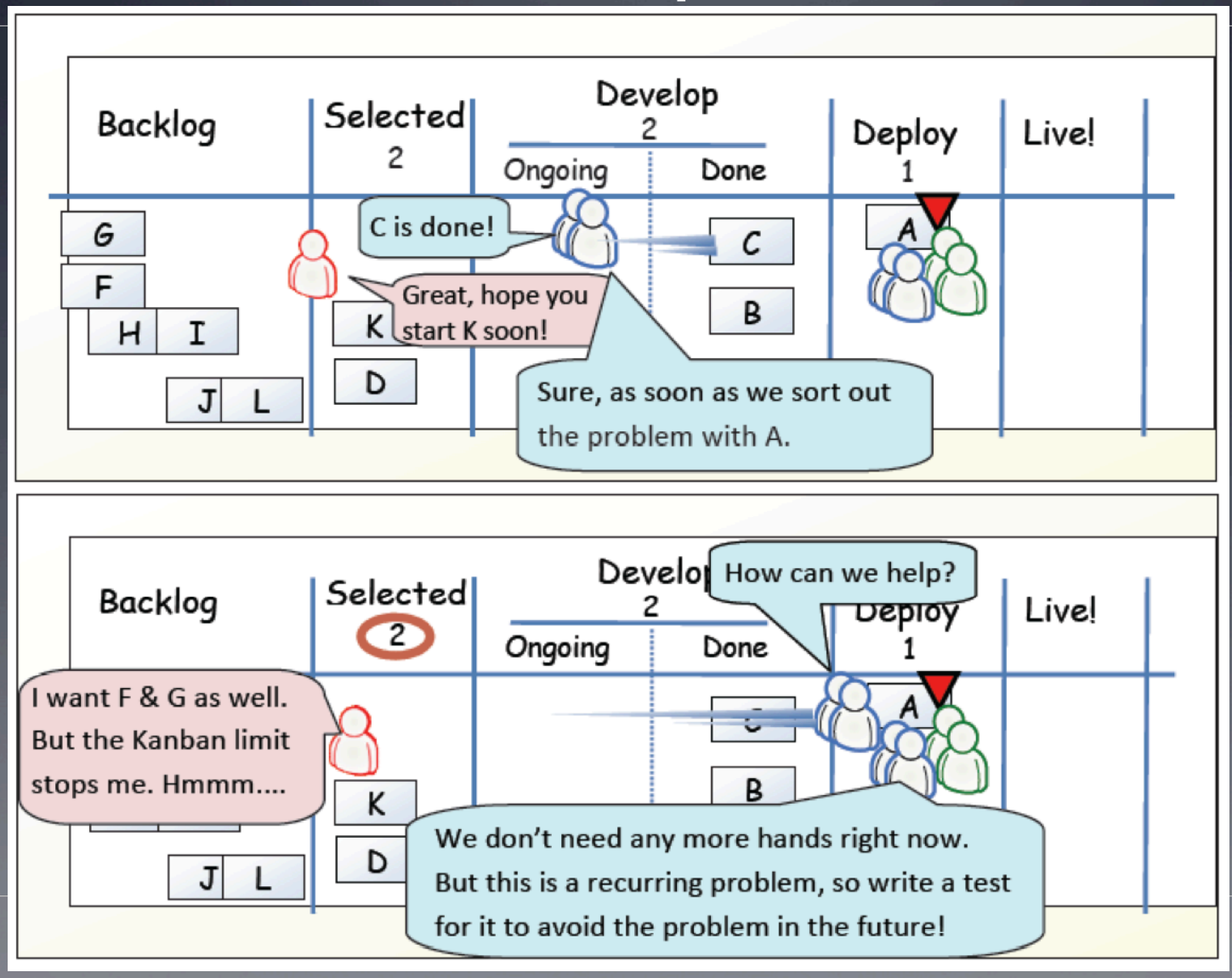
Metrics

- Stories in progress (SIP)
- When story enters stories queue set entry date (ED)
- When story enters first process step set start processing date (SPD)
- When story is done set finish date (FD)
- Cycle time (CT) = FD – SPD
- Waiting time (WT) = SPD – ED
- Throughput (T) = SIP / CT

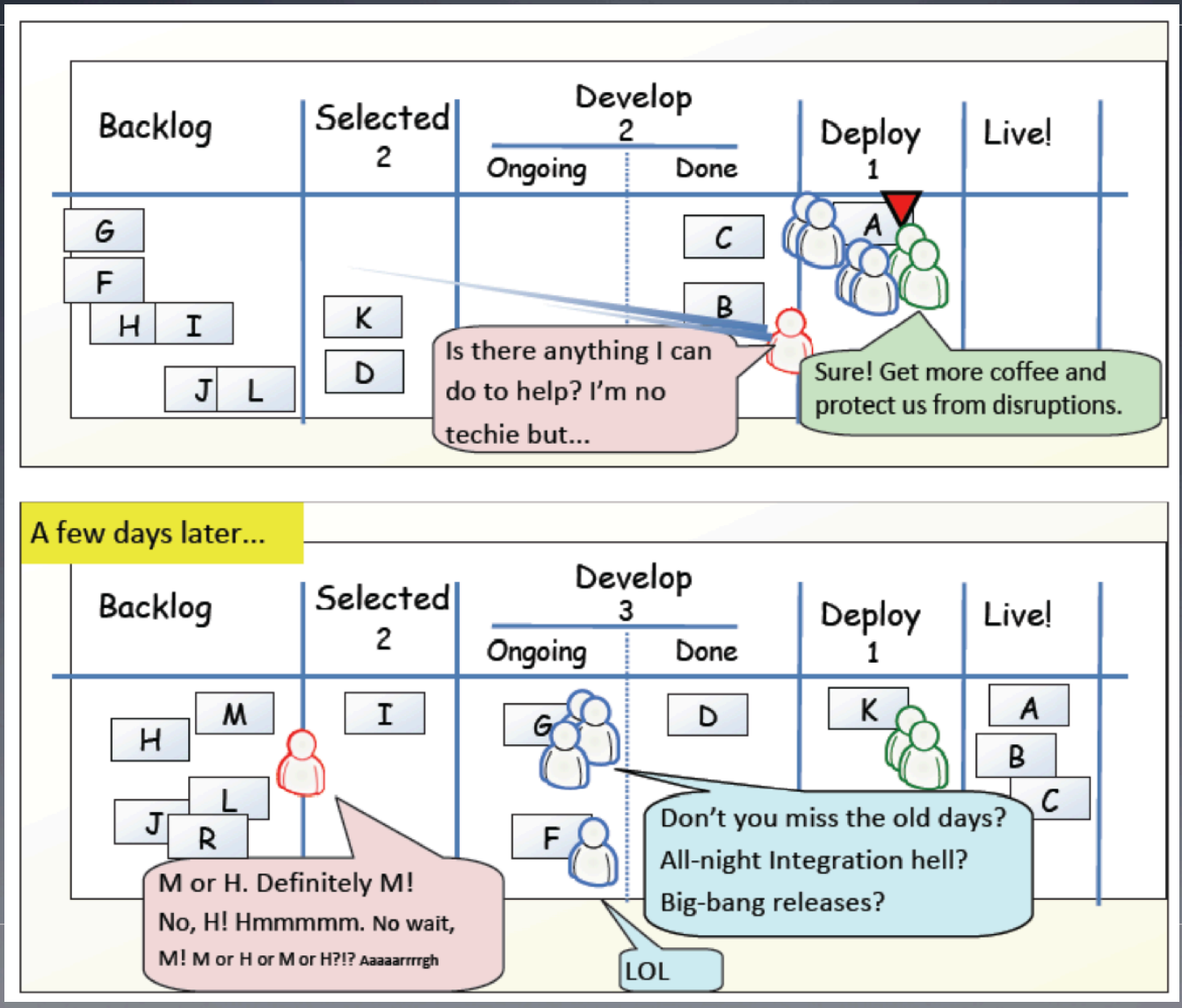
Example


Backlog
Selected
2
Develop
2
Ongoing
Done
Deploy
1
Live!
G
F
H
I
K
J
L
D
C
B
A
We'll do D!
No, wait.... that would
break the Kanban limit of 2!
Backlog
Selected
2
Develop
2
Ongoing
Done
Deploy
1
Live!
K is pretty urgent. I'll
schedule it next.
G
F
H
I
K
J
L
D
C
B
A
How can
we help?
Get coffee. Then tell
us what this #%&@
stack trace means.

Example


Backlog
Selected
2
Develop
2
Ongoing
Done
Deploy
1
Live!
G
F
H
I
J
L
K
D
C
B
A
C is done!
Great, hope you start K soon!
Sure, as soon as we sort out the problem with A.
Backlog
Selected
2
Develop
2
Ongoing
Done
Deploy
1
Live!
How can we help?
I want F & G as well. But the Kanban limit stops me. Hmmm....
K
D
J
L
B
A
We don't need any more hands right now. But this is a recurring problem, so write a test for it to avoid the problem in the future!

Example


Backlog
Selected 2
Develop 2
Ongoing
Done
Deploy 1
Live!
G
F
H
I
J
L
K
D
C
B
A
Is there anything I can do to help? I'm no techie but...
Sure! Get more coffee and protect us from disruptions.
A few days later...
Backlog
Selected 2
Develop 3
Ongoing
Done
Deploy 1
Live!
H
M
J
L
R
I
G
F
D
K
A
B
C
M or H. Definitely M! No, H! Hmmmmm. No wait, M! M or H or M or H?!? Aaaaarrrrgh
Don't you miss the old days? All-night Integration hell? Big-bang releases?
LOL

Scrum vs. Kanban

Scrum	Kanban
Timeboxed iterations prescribed.	**Timeboxed iterations optional.** Can have separate cadences for planning, release, and process improvement. Can be event-driven instead of timeboxed.
Team commits to a specific amount of work for this iteration.	**Commitment optional.**
Uses **Velocity** as default metric for planning and process improvement.	Uses **Lead time** as default metric for planning and process improvement.
Cross-functional teams prescribed.	Cross-functional teams optional. **Specialist teams allowed.**
Items must be broken down so they can be completed within 1 sprint.	No particular item size is prescribed.
Burndown chart prescribed	No particular type of diagram is prescribed

Scrum vs. Kanban

Scrum	Kanban
WIP limited indirectly (per sprint)	**WIP limited directly** (per workflow state)
Estimation prescribed	**Estimation optional**
Cannot add items to ongoing iteration.	**Can add new items whenever capacity is available**
A **sprint backlog is owned by one specific team**	A **kanban board may be shared by multiple teams** or individuals
Prescribes 3 roles (PO/SM/Team)	**Doesn't prescribe any roles**
A **Scrum board is reset** between each sprint	A **kanban board is persistent**
Prescribes a prioritized product backlog	**Prioritization is optional.**

Kanban principles

- Start with what you do now
- Agree to pursue incremental, evolutionary change
- Respect the current process, roles, responsibilities and titles
- Leadership at all levels

Agile Principles

1. Early and continuous delivery of valuable software
2. Welcome changing requirements, even late
3. Deliver working software frequently
4. Business people and developers must work together
5. Build projects around motivated individuals
6. Face-to-face communication is most effective and efficient
7. Working software is the primary measure of progress
8. Sustainable development
9. Continuous attention to technical excellence and good design
10. Simplicity is essential
11. Self-organizing teams
12. Regular reflection