Team project

Short introduction: methodologies

WYDZIAŁ
MATEMATYKI
i INFORMATYKI
Uniwersytet Łódzki

Piotr Fulmański

Goals

Goals

What we will talk about and why

1. Do we need any control on software development?
2. First attempts – waterfall model.
3. Is it good or bad?
4. How you can think about this task in different way.
5. Less stiffness, more flexibility.
6. SCRUM – holy grail of software development?
7. SCRUM is not the only one.

My private thoughts you don't have to agree with

My private thoughts you don't have to agree with

Is it good or bad?

Is it [methodology] good or bad? The mos honest and probably correct answer is: it depends.

In general, I treat all methodologies as a necessary evil, or more precisely as a design savoir-vivre. Some rules are simply better than none at all. Thanks to this, everyone knows what, where, when and how. However, I am a staunch opponent of subordinating everything without exception to a chosen methodology. The argument: "*Because this is how it is (not) done in XYZ and that's it.*" is unacceptable to me. Methodology is there to facilitate management, not to rule us. I always ask myself "*So what?*", "*Why do I need this?*", "*What does it give to me?*" and further actions depend on the answers to them.

My private thoughts you don't have to agree with

Is it good or bad?

"Scrum? Agile? I co z tego?"

https://www.youtube.com/watch?v=mi3QAKuOIu4

Long, long time ago...

Long, long time ago...
Waterfall model

Waterfall model – one of several types of software development processes defined in software engineering.

The waterfall model is a breakdown of development activities into **linear sequential *phases***, meaning they are passed down onto each other, where each phase depends on the deliverables of the previous one and corresponds to a specialization of tasks.

The approach is typical for certain areas of engineering design where the process is highly iterative – parts of the process often need to be repeated many times before another can be entered. [1]

Long, long time ago...
Waterfall model

Waterfall model is attributed to:

- The first known presentation describing use of such phases in software engineering was held by Herbert D. Benington at the Symposium on Advanced Programming Methods for Digital Computers on 29 June 1956.
- Although the term "waterfall" is not used in the paper, the first formal detailed diagram of the process later known as the "waterfall model" is often cited as a 1970 article by Winston W. Royce in the article „Managing the Development of Large Software Systems”.
- The earliest use of the term "waterfall" may have been in a 1976 paper by Bell and Thayer [1].

Long, long time ago...

Waterfall model phases

1. System and software requirements: captured in a product requirements document
2. Analysis: resulting in models, schema, and business rules
3. Design: resulting in the software architecture
4. Coding: the development, proving, and integration of software
5. Testing: the systematic discovery and debugging of defects
6. Operations: the installation, migration, support, and maintenance of complete systems

The waterfall model maintains that one should move to a phase only when its preceding phase is reviewed and verified.

Long, long time ago...

Waterfall model

Mostly attributed to Winston W. Royce, however he:

- pointed out the major flaws stemming from the fact that testing only happened at the end of the process, which he described as being "risky and invites failure";
- he criticized rigid adherence to the phases.

Long, long time ago...

Waterfall model – supporting arguments

Cost reduction

Time spent early in the software production cycle can reduce costs at later stages. For example, a problem found in the early stages (such as requirements specification) is cheaper to fix than the same bug found later on in the process (by a factor of 50 to 200).

Long, long time ago...
Waterfall model – supporting arguments

High maintainability

It places emphasis on documentation (such as requirements documents and design documents) as well as source code. In less thoroughly designed and documented methodologies, **knowledge is lost if team members leave before the project is completed, and it may be difficult for a project to recover from the loss**. If a fully working design document is present, new team members or even entirely new teams should be able to familiarise themselves by reading the documents.

Long, long time ago...

Waterfall model – supporting arguments

Easy to understand and explain

The waterfall model provides a structured approach; the model itself progresses linearly through discrete, easily understandable and explainable phases and thus is easy to understand; it also provides easily identifiable milestones in the development process.

Long, long time ago...

Waterfall model – supporting arguments

Foresightedness

Simulation can play a valuable role within the waterfall model. By creating computerized or mathematical simulations of the system being developed, teams can gain insights into how the system will perform before proceeding to the next phase. Simulations allow for testing and refining the design, identifying potential issues or bottlenecks, and making informed decisions about the system's functionality and performance.

Long, long time ago...

Waterfall model – supporting arguments

In common practice, waterfall methodologies result in a project schedule with 20–40% of the time invested for the first two phases, 30–40% of the time to coding, and the rest dedicated to testing and implementation.

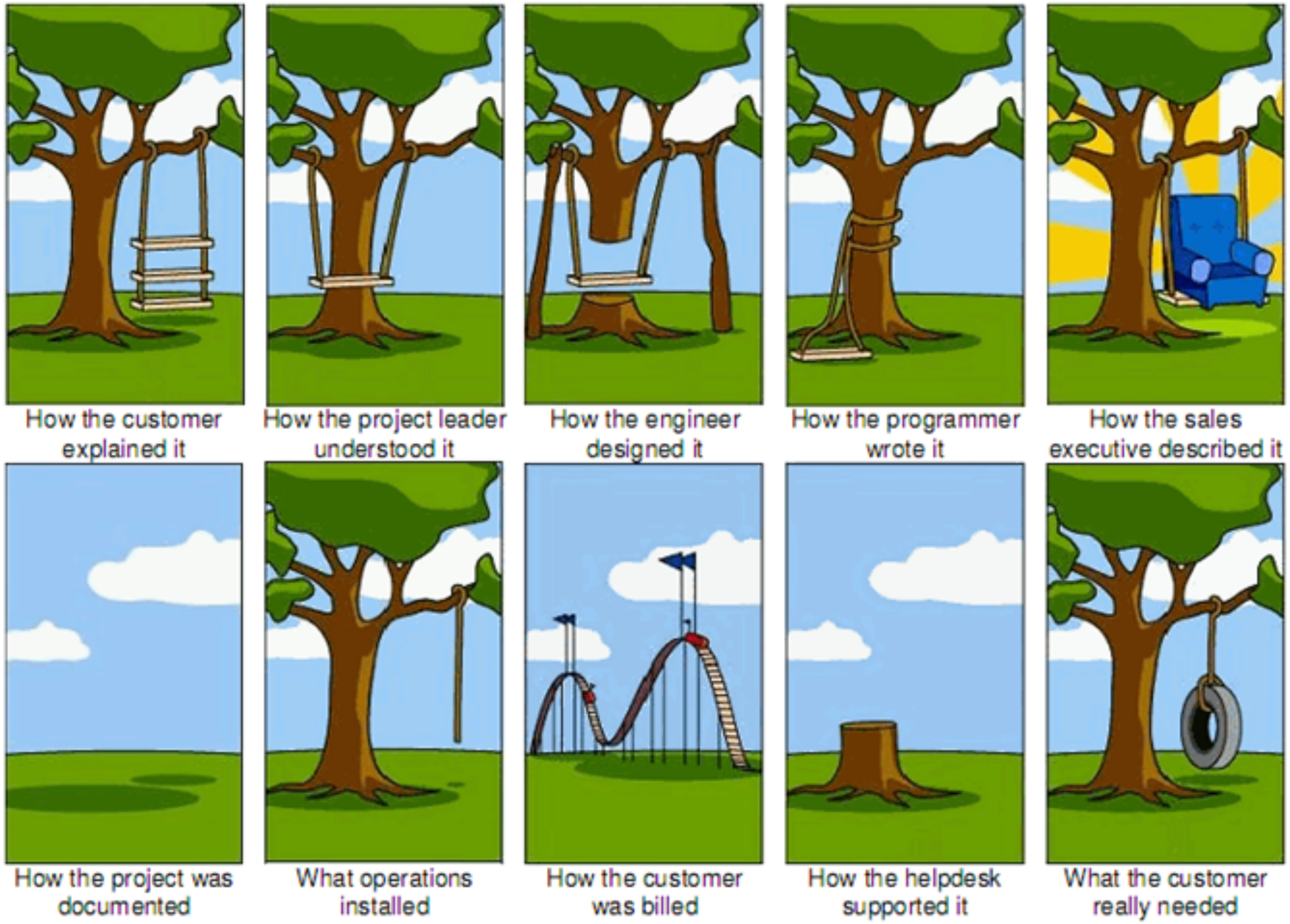
Long, long time ago...

Waterfall model – criticism

Inability to self-define

Clients may not know exactly what their requirements are before they see working software and so change their requirements, leading to redesign, redevelopment, and retesting, and increased costs.

Long, long time ago...
Waterfall model – criticism


How the customer explained it
How the project leader understood it
How the engineer designed it
How the programmer wrote it
How the sales executive described it
How the project was documented
What operations installed
How the customer was billed
How the helpdesk supported it
What the customer really needed

Long, long time ago...

Waterfall model – criticism

Prediction is very difficult, especially if it’s about the future [2,3]

Niels Bohr [nels boa]

Designers may not be aware of future difficulties when designing a new software product or feature, in which case it is better to revise the design than persist in a design that does not account for any newly discovered constraints, requirements, or problems. Implementing any non-trivial system will almost inevitably expose issues and edge cases that the systems analyst did not consider.

Long, long time ago...
Waterfall model – criticism

Prediction is very difficult, especially if it’s about the future [2,3]

Niels Bohr [nels boa]

Designers may not be aware of future difficulties when designing a **NEW** software product or feature, in which case it is better to revise the design than persist in a design that does not account for any newly discovered constraints, requirements, or problems. Implementing any non-trivial system will almost inevitably expose issues and edge cases that the systems analyst did not consider.

Long, long time ago...

Waterfall model – criticism

Prediction is very difficult, especially if it’s about the future [2,3]

Niels Bohr [nels boa]

The model can only be used if the requirements are clear and transparent, because each iteration is time-consuming and requires a large investment in improvement. At the same time, it is used in normal engineering practice.

Long, long time ago...

Waterfall model – criticism

Non-flexibility (non-agile)

Inflexible division into successive disjoint iterative phases. Moving to the next phase possible after completing the previous one.

Iterative and incremental development

Iterative and incremental development

A sparrow in the hand is better than the pigeon on the roof

It's better to have something quickly (right now) than to have everything without knowing when.

Two stories:

- home renovation [5, p. 155];
- the Trident missile – a submarine-launched ballistic missile.

Iterative and incremental development

A sparrow in the hand is better than the pigeon on the roof

The basic idea behind this method is to develop a system through repeated cycles (iterative) and in smaller portions at a time (incremental), allowing software developers to take advantage of what was learned during development of earlier parts or versions of the system. Learning comes from both the development and use of the system, where possible key steps in the process start with a simple implementation of a subset of the software requirements and iteratively enhance the evolving versions until the full system is implemented.

Iterative and incremental development

A sparrow in the hand is better than the pigeon on the roof

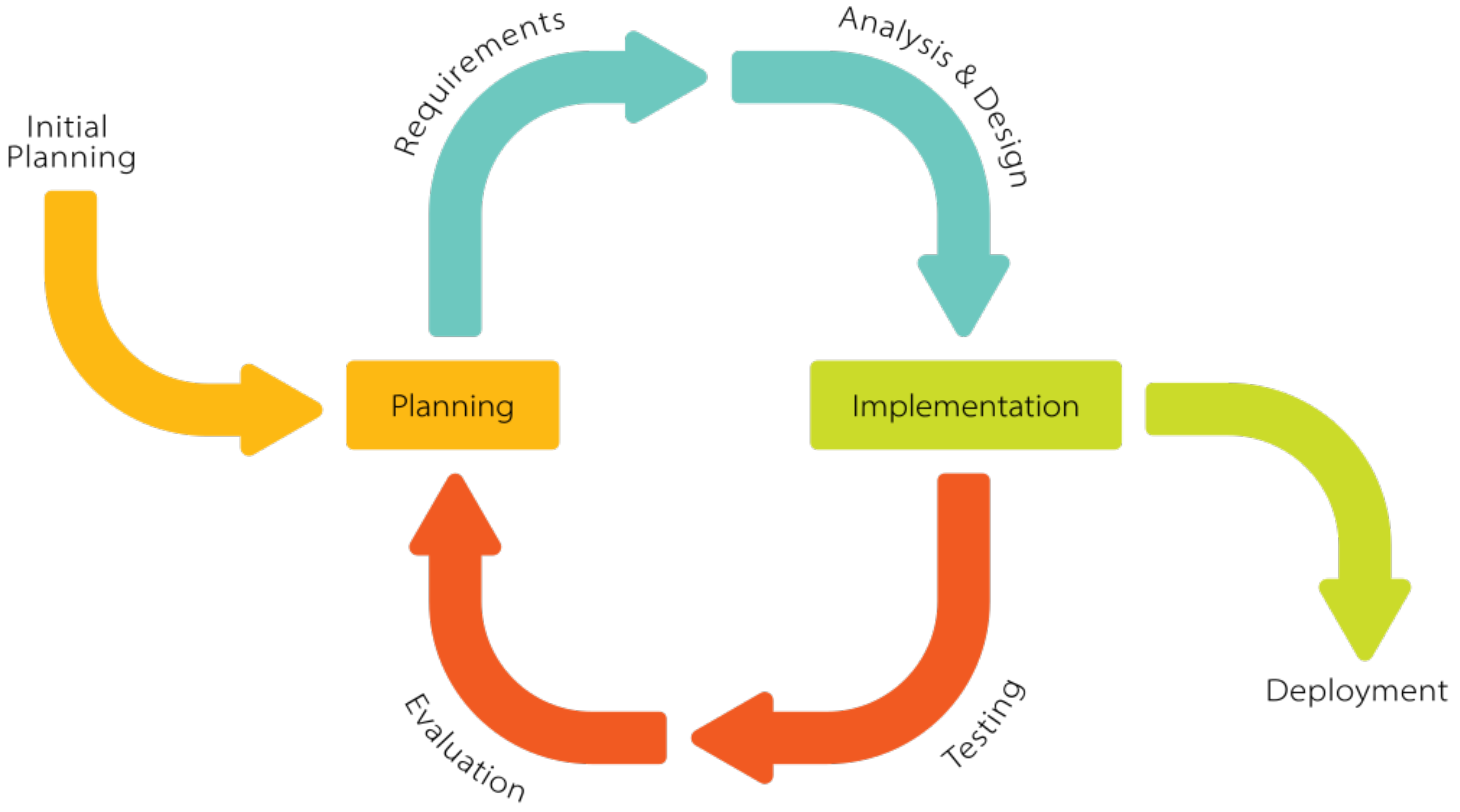


Image: [4]

At each iteration, design modifications are made and new functional capabilities are added.

Iterative and incremental development

Supporting arguments

- No need to define all requirements in advance (at the beginning we define what we will achieve, hoping that we will be able to specify all requirements at the stage of testing the implemented fragments).
- Frequent contacts with the client (shortened breaks compared to the waterfall model).
- Early use of parts of the system (functionalities) by the client.
- Possibility of flexible response to delays in the implementation of a fragment (acceleration of work on other parts without delaying the entire project undertaking).

Iterative and incremental development

Criticism

- Potential difficulties with cutting out a subset of fully independent functions.
- Additional cost related to independent implementation of system fragments.
- Consequence of above problems: necessity to implement skeletons (interface compatible with the target system) – additional workload (cost), risk of failure to detect errors in the testing phase.

Agile software development

Agile software development

Less stiffness, more flexibility

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value [7]:

- **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.

Agile software development

Less stiffness, more flexibility

Twelve principles of the Agile Manifesto [8]:

1. Customer satisfaction by early and continuous delivery of valuable software.
2. Welcome changing requirements, even in late development.
3. Deliver working software frequently (weeks rather than months).
4. Close, daily cooperation between business people and developers.
5. Projects are built around motivated individuals, who should be trusted.
6. Face-to-face conversation is the best form of communication (co-location).
7. Working software is the primary measure of progress.
8. Sustainable development, able to maintain a constant pace.
9. Continuous attention to technical excellence and good design.
10. Simplicity—the art of maximizing the amount of work not done – is essential.
11. Best architectures, requirements, and designs emerge from self-organizing teams.
12. Regularly, the team reflects on how to become more effective, and adjusts accordingly.

Agile software development
Less stiffness, more flexibility

When performing tasks in the software production process based on the agile programming methodology, the following stages are distinguished:

- planning,
- designing,
- programming,
- testing,
- release (implementation and deployment of the system),
- feedback.

The above stages create a cycle repeated until the completion of a given task.

Agile software development
Less stiffness, more flexibility

Very, very important:

It is very important to note that subsequent cycles are to serve the purpose of:

- possible correction of the prepared task based on the customer's information
- or flexible introduction of possible changes in the customer's requirements, if such have appeared in the form of feedback.

Subsequent cycles **are not intended** to endlessly correct errors in a given task resulting from the omission or imprecise execution of the planning stage, including the collection and analysis of requirements from the customer.

Agile software development

Criticism

- An excessive focus on rapid delivery at the expense of quality.
- Lack of sufficient documentation.
- Difficulty in forecasting project time and cost.

SCRUM – the holy grail of software development?

SCRUM

SCRUM – the holy grail of software development?

No, not at all. SCRUM is just one of many agile methodologies you can choose from. Probably the most popular at this moment.

SCRUM

Scrum is an agile team collaboration framework commonly used in product development.

Is it the best?

No, not at all. SCRUM is just one of many agile methodologies you can choose from. Probably the most popular in software development at this moment.

SCRUM
In short

- User requirements are usually collected in the form of (verifiable) **user stories**.
- Scrum prescribes for teams to **break work into goals** to be completed within time-boxed iterations, called ***sprints***.
- Each sprint is **no longer than one month** and commonly lasts two weeks.
- A team is a self-organizing body by definition, **there is no way to assign tasks to individual team members**. They choose the tasks to perform, according to common agreements, skills, or other preferences.
- The scrum team assesses progress in time-boxed, **stand-up** meetings of up to 15 minutes (hence their name: stand-ups, nowadays very often called **daily** scrums).
- At the end of the sprint, the team holds two further meetings:
  - one **sprint review** to demonstrate the work for stakeholders and solicit feedback,
  - and one internal sprint **retrospective**.
- A person in charge of a scrum team is typically called a scrum master.

SCRUM

Team and responsibilities

Typically, a Scrum team consists of less than 10 people. It is good if it is interdisciplinary and consists of people with different skills. The main roles in the project are played by:

- Scrum Master,
- Product Owner
- and Developers.

SCRUM

Team and responsibilities

Scrum Master

Some scrum master responsibilities include coaching, objective setting, problem solving, oversight, planning, backlog management, and communication facilitation. Scrum master's responsibility is not to manage the project (Scrum master is not a project manager). Scrum master's role is also to educate and coach teams about scrum theory and practice.

SCRUM

Team and responsibilities

Product Owner

Each scrum team has one product owner. The product owner focuses on the business side of product development and spends the majority of time liaising with stakeholders and the team. The role is intended to primarily represent the product's stakeholders, the voice of the customer.

Product owners manage the product backlog and are responsible for maximizing the value that a team delivers.

They do not dictate the technical solutions of a team.

SCRUM

Team and responsibilities

Developers

The term *developer* or *team member* refers to anyone who plays a role in the development and support of the product and can include researchers, architects, designers, programmers, etc.

SCRUM
User stories

User stories are short and concise descriptions of the functionality to be implemented in the project. They are written from the perspective of the user of the system and describe what task the user would like to perform to achieve a specific goal:

As a ... I want to ...

SCRUM
User stories

1. Identify user role
2. Describe user goal
3. Describe user action
4. Add details
5. Add acceptance criteria (measurable and unambiguous criteria to determine if the task has been completed)
6. Estimate priority (assign priority)
7. Add an estimate (value the task)
8. Plan the task (assign to sprint)

SCRUM

More about Scrum

- Scrum, `https://boringowl.io/tag/scrum`
- Scrum — co to za metodyka?, `https://interviewme.pl/blog/scrum`
- Czym jest Scrum i jak zacząć, `https://www.atlassian.com/pl/agile/scrum`
- Przewodnik po Scrumie (Scrum Guide), `https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-Polish.pdf`

SCRUM is not the only one

SCRUM is not the only one

KANBAN, JIT, Lean management (Toyota Production System, TPS)

Kanban (jp., meaning signboard) is a scheduling system for lean manufacturing (also called just-in-time manufacturing, abbreviated JIT). Taiichi Ohno, an industrial engineer at Toyota, developed kanban after observing supermarkets after World War II and to apply the idea of shelf-stocking techniques to the factory floor to improve manufacturing efficiency. In a supermarket, customers generally retrieve what they need at the required time – no more, no less.

The system takes its name from the cards that track production within a factory.

One of the main benefits of kanban is to establish an upper limit to work in process inventory to avoid overcapacity.

The basic Kanban principle in software development is visualization – presentation of subsequent stages of processes on a board (wall or electronic). Then writing down the tasks on cards and placing them in the appropriate columns.

Another one is to limit work in progress – setting the maximum allowable number of tasks that can be in a given column. Little's law is used for this purpose.

SCRUM is not the only one

KANBAN, JIT, Lean management (Toyota Production System, TPS)

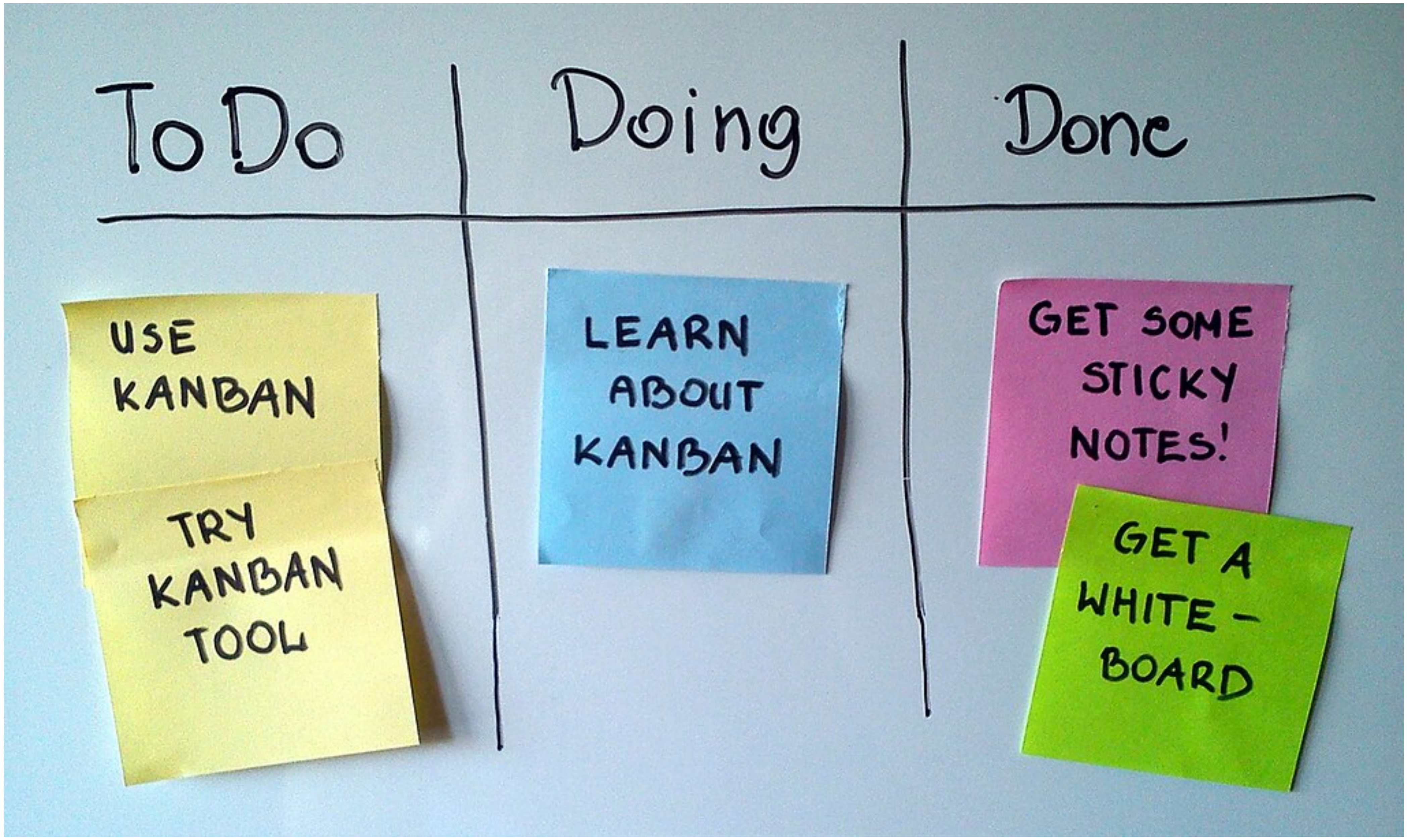
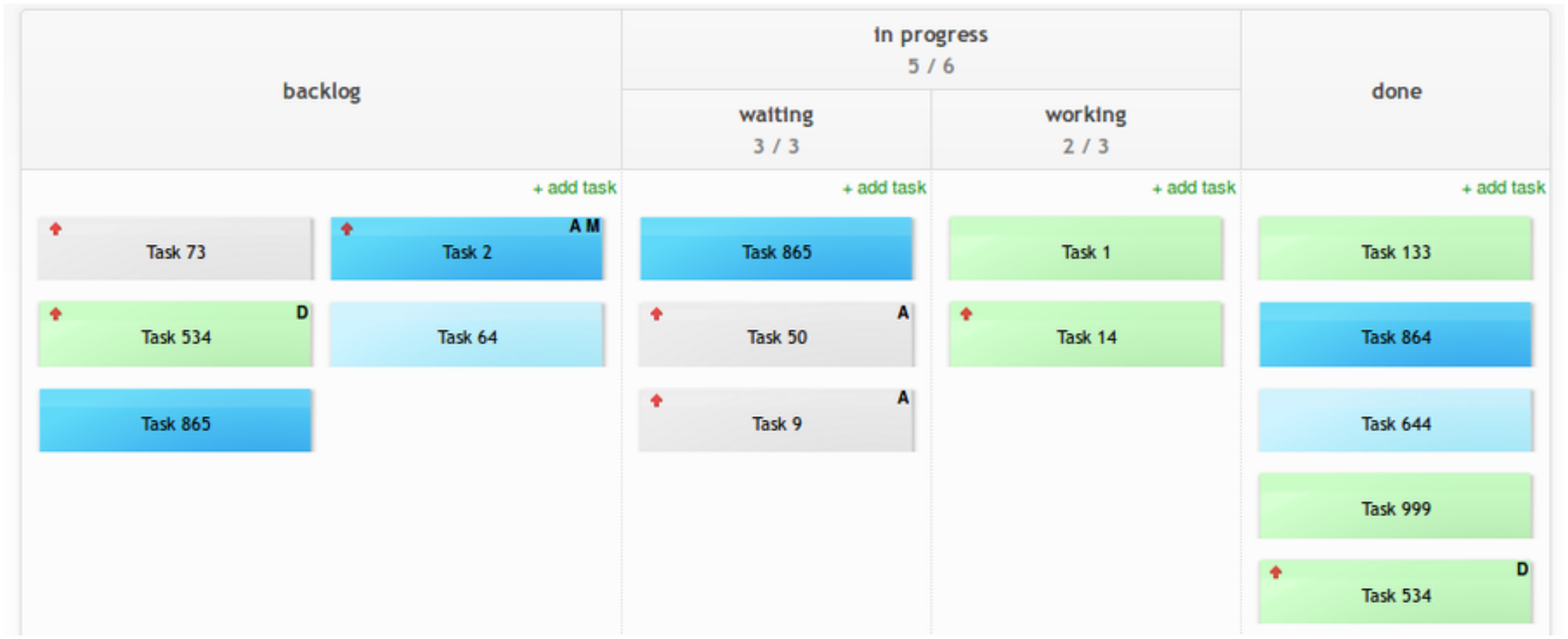


Image: [9]

SCRUM is not the only one

KANBAN, JIT, Lean management (Toyota Production System, TPS)



backlog		in progress 5 / 6		done
		waiting 3 / 3	working 2 / 3	
	+ add task	+ add task	+ add task	+ add task
Task 73	A M Task 2	Task 865	Task 1	Task 133
D Task 534	Task 64	A Task 50	Task 14	Task 864
Task 865		A Task 9		Task 644
				Task 999
				D Task 534

Image: [10]

Bibliography

Bibliography

1. Engineering design process, `https://en.wikipedia.org/wiki/Engineering_design_process`, retrieved 2024-09-30
2. Niels Bohr (cytaty), `https://pl.wikiquote.org/wiki/Niels_Bohr`, retrieved 2024-09-30
3. Quote Origin: It’s Difficult to Make Predictions, Especially About the Future, `https://quoteinvestigator.com/2013/10/20/no-predict/`, retrieved 2024-09-30
4. Image by Krupadeluxe - Own work, CC BY-SA 4.0, `https://commons.wikimedia.org/wiki/File:Iterative_Process_Diagram.svg`, retrieved 2024-09-30
5. Stephen Denning, *Radykalna rewolucja w zarządzaniu*, Helion 2012.
6. Mariusz Chrapko, *SCRUM. O zwinnym zarządzaniu projektami*, Helion, 2013.
7. Kent Beck; James Grenning; Robert C. Martin; Mike Beedle; Jim Highsmith; Steve Mellor; Arie van Bennekum; Andrew Hunt; Ken Schwaber; Alistair Cockburn; Ron Jeffries; Jeff Sutherland; Ward Cunningham; Jon Kern; Dave Thomas; Martin Fowler; Brian Marick, *Manifesto for Agile Software Development*, Agile Alliance, 2001, `http://agilemanifesto.org`, retrieved 2024-10-01
8. Kent Beck; James Grenning; Robert C. Martin; Mike Beedle; Jim Highsmith; Steve Mellor; Arie van Bennekum; Andrew Hunt; Ken Schwaber; Alistair Cockburn; Ron Jeffries; Jeff Sutherland; Ward Cunningham; Jon Kern; Dave Thomas; Martin Fowler; Brian Marick, *Principles behind the Agile Manifesto,* Agile Alliance, 2001. Archived from the original on 14 June 2010, `https://web.archive.org/web/20100614043008/http://www.agilemanifesto.org/principles.html`, retrieved 2024-10-01
9. Jeff.lasovski, `https://commons.wikimedia.org/wiki/File:Simple-kanban-board-.jpg` (Creative Commons Attribution-Share Alike 3.0 Unported license), retrieved 2024-09-30
10. Bossarro, `https://commons.wikimedia.org/wiki/File:Basic-team-kanban-board.png` (Creative Commons Attribution-Share Alike 4.0 International license), retrieved 2024-09-30

WYDZIAŁ
MATEMATYKI
i INFORMATYKI
Uniwersytet Łódzki