CI/CD

Agile methodologies

Aleix Llusà Serra

Master's degree in Machine Learning and Cybersecurity for Internet Connected Systems
Universitat Politècnica de Catalunya
https://epsem.upc.edu

4 de novembre de 2026



This work is licensed under a Creative Commons "Attribution-ShareAlike 4.0 International" license.

1 Agile methodologies

2 Planning

3 Tasks

4 Final remarks

5 References

Agile manifesto (philosophy)

The 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.

© 2001 https://agilemanifesto.org and Twelve Principles of Agile Software https://agilemanifesto.org/principles.html

Agile development

Definition

It's about simplicity and feedback, communication and trust. It's about delivering value. (Shore i Warden 2008)

Definition

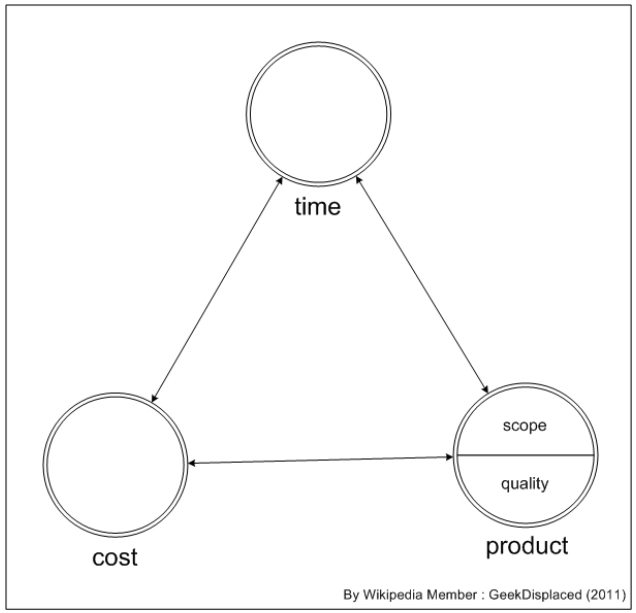
Rather than trying to plan for every eventuality, embrace the possibilities that change brings you. (Shore i Warden 2008)

Software phases

1. Planning
2. Analysis
3. Design
4. Coding
5. Testing
6. Deployment

Agile development

The project management triangle:



Project: cost and deadline

- Traditional lifecycle: phased-based, do not start next phased until last is completed
- Agile: All phases iterated in parallel

Be warned: every project always has some disappointment associated.

Agile development

- Mix customers (not end-users) and developers
- Adaptive, not predictive
- Delivering value and decreasing costs
- Release most valuable first
- Release new versions frequentlly
- Change direction when needed or when discovered
- Cancel bad tasks: the art of maximizing the amount of work not done
- Trust the team, self-organising teams

Agile methods

- Extreme Programming (XP)
- Scrum
- ...

Exercise: Gitlab plan and track

https://docs.gitlab.com/ee/topics/plan_and_track.html

Example: https://gitlab.com/groups/gitlab-org/

Roles

The team members and the project community, by Shore i Warden (2008)

- On-site customers: What stakeholders will find valuable
- Programmers/developers, designers, and architects: How to design and program the software
- Testers: Where defects are likely to hide
- Operations: Deploy, monitor, secure in production
- Product manager/Product owner: Why the software is important
- Project manager: How to interact with the rest of the company
- Domain experts: The rules the software should follow
- Interaction designers: How the software should behave
- Graphic designers: How the user interface should look
- Coach: Where to improve work habits
- Stakeholder: end users, purchasers, managers, and executives
- Executive sponsor, senior management...

Team size

Shore i Warden (2008):

- Large teams incur extra communication and process overhead, reducing individual productivity
- 12 people turns out to be a natural limit for team collaboration
- 6 programmers, 4 customers, 1 tester, and 1 project manager
- Small team: 4 programmers, 1 product manager (also project manager, customer, and tester)

Agile practices

- Version control
- Coding standards
- Weekly demos
- Daily stand-up meeting
- Informative workspace
- Work in pairs

Shore i Warden (2008) Quiz `https://www.jamesshore.com/v2/books/aoad1/assess_your_agility` or `https://www.oreilly.com/library/view/the-art-of/9780596527679/ch04s03.html`

Extreme Programming (XP)

- Test-driven development
- Always capable of deployment
- Requirements, design, and testing are good: so do them every day
- Quick feedback: easier to correct bugs while developing
- Iterations: weekly working on stories
- Technical Debt (is it easy to reimplement instead of modify?): refactor

Extreme Programming (XP)

Shore i Warden (2008):

- Team agreement
- Sit together
- Keep code clean
- Show progress every week
- Iteration planning
- Incremental design and architecture
- 10 minute build

DevOps

- Break down the wall between development, operations and security
- Cross-functional teams: also DevSecOps, with database admins...
- Make software resilient and stable
- DevOps features/tasks must also be in the planning
- Create stories for monitoring, scaling, securing the software on production

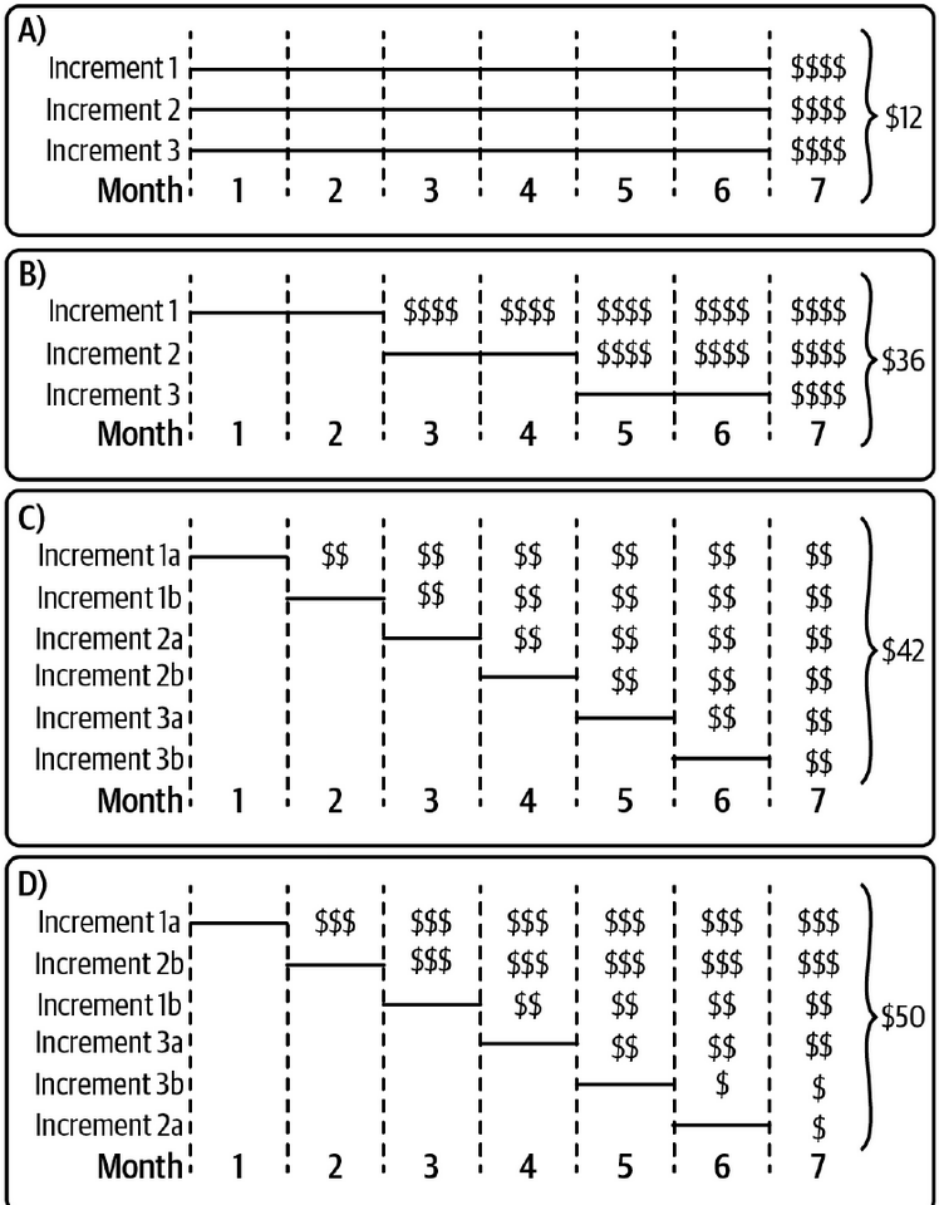
Exercise: relaxing reading

Understanding XP (Shore i Warden 2008)
“I’m eager to get started,” said Pat. “I took a software engineering course in school, and they taught us about the software development lifecycle. That made a lot of sense. There was a bit about XP, but i sounded like i was mostly about working in pairs and writing tests first. Is that right?”
“Not exactly,” said Kim

Planning

- Vision statement: Why the project exists?
  - What should accomplish
  - Why is valuable
  - Success criteria (proof-of-concept, beta, payed service, profits…)
- Release planning: roadmap, plan for implementation and for learning
  - Scopeboxed: features to release without date
  - Timeboxed: release date without knowing specific features
  - Release date + features?: software development is risky and uncertain

Planning on value



(Shore i Warden 2008)

Table 8-1. *Realistic example of focusing on value*

	Scenario A	Scenario B
Total Cost	$4.312 million	$4.712 million
Revenue	$5.600 million	$7.800 million
Investment	$2.760 million	$1.640 million
Payback	$1.288 million	$3.088 million
Net Present Value @ 10%	$0.194 million	$1.594 million
Internal Rate of Return	12.8%	36.3%

A) Increments in parallel

B) one increment at a time

C) smaller releases

D) highest-value first

Stories

- Future discussion notes
- Be brief
- Just a reminder for the next meeting
- For planning with customers
- So: customer-centric value instead of developer-centric
- One story: releasable
- Usability centric not required: As a <role> I can <what> to <why>

E.g: Put demo data on stage environment. Support 1000 transactions per minute.

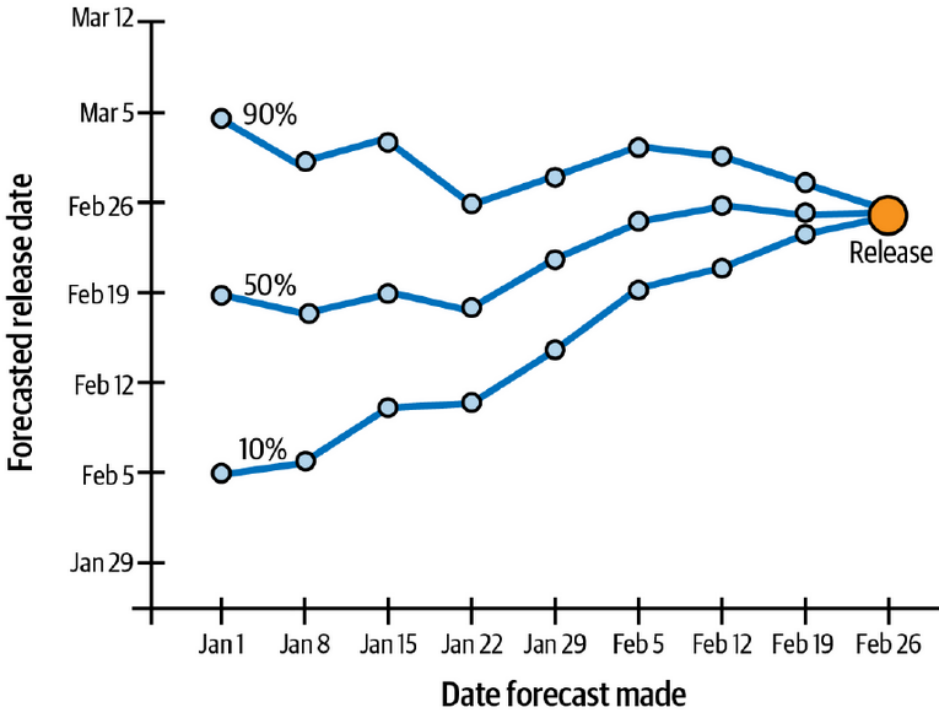
Release planning

- List of prioritised stories
- Highest value and lowest cost first
- Programmers: estimate (costs)
- Customers: prioritise (value)

Release risk management forecasting

Risk adjustment rules of thumb (Shore i Warden 2008)

Likelihood	Low-risk-team	High-risk team
10% (almost impossible)	1	1
50% (coin toss)	1.4	2
90% (very likely)	1.8	4



Risk management: new requirements, sick, vacations, uncertainty, computer crash, unexpected difficulty in technology, unexpected important bugs, more use than expected.... Negative thinking wanted.

Iteration planning

- From the prioritised release plan work in iterations
- Timeboxed every week or two. Then it communicates feedback and progress (or reveals problems)...(Hint: why not starting on wednesday?)
- Are you firefighting? Try daily iterations.
- Only plan achievable iterations: plan stories by adding estimated time and comparing to the last iteration velocity
- Then each story should be broken down into engineering tasks by programmers (one task, one commit to VCS)
- At the end of the iteration: continuous deployment at least to a demo

Capacity/Velocity

- How much the team can accomplish in an iteration?
- Just add the last iteration's completed estimations (or count finished stories if they are the same size)
- Accuracy does not matter: you are consistently inaccurate at estimating
- The last iteration capacity shows how much overhead you have (e.g. bug fixing)
- You don't need to add the time expended in meetings nor refactoring nor other minor tasks:
  next iteration = capacity + slack (time for minor tasks)
- Capacity is a prediction tool, not a productivity measure
- Capacity tends to stabilise along time, unless there are holidays

Relaxing reading

How to Win the Planning Game (Shore i Warden 2008)

Stories types

- New funcionalities
- Documentation
- Bugs
- Nonfunctional: performance, scalability, stability…
- Operation, monitoring and security
- Spike: research to size other stories
- Clean-up, refactor: however incremental design is better

Minimum viable product (MVP)

- Minimum marketable feature (MMF): smallest set of functionality valuable
- Work in incremental design: from specific to abstraction and generalisation
- Test driven development is needed
- Be obsessed with simplicity and simple design
- Only develop to fulfil your current requirements
- Refactor when you see it
- Do not anticipate changes
- Do not think about extensibility
- Trust that a simple project is more ready for any future change

What is a good design?

- Do you need a lot of extra documentation or source code provides understandable meaning: e.g. assembler vs high languages
- Waste computer resources (cheap) in favour of programmer resources (expensive)?
- Over years and lot of changes is it still easy to modify?

Definition

A good software design minimizes the time required to create, modify, and maintain the software while achieving acceptable runtime performance (Shore i Warden 2008)

Definition

Modification and maintenance times are more important than creation time (Shore i Warden 2008)

Final remark

About the principles in practice (Agile principles or others), consider them in context:

Definition

Exercise healthy scepticism. Ask yourself, «When is this not true?» and «What is the author assuming?» (Shore i Warden 2008)

Exercise the pragmatism and only adopt the methodologies your team wants to adopt.

References



Shore, James i Shane Warden (2008). The Art of Agile Development. O'Reilly. ISBN: 9780596527679. URL: https://discovery.upc.edu/permalink/34CSUC_UPC/8e3cvp/alma991005248477306711.