Software and systems engineering

Paulo Borba
Informatics Center
Federal University of Pernambuco

phmb@cin.ufpe.br ◈ twitter.com/pauloborba

To do before class

- Watch video
- Read related parts of chapters 7 and 10 in the textbook
- Send questions and opinions through slack

Project management I, time and scope

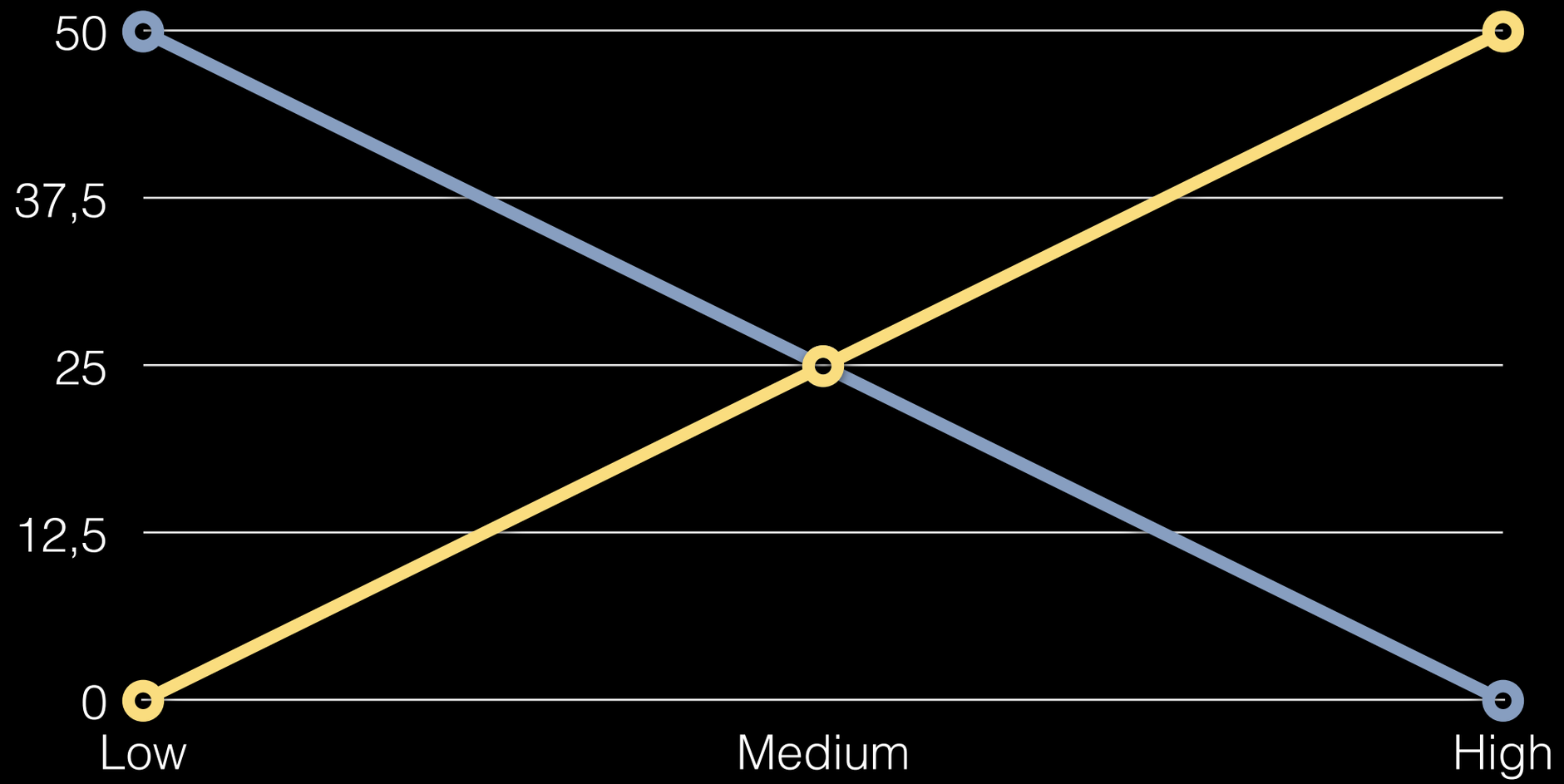
To deliver the required software and system, we have to specify, implement and test the needed features…

That often is a lot of work, carried on by a number of people, so we better get organized!

We need to control interdependent dimensions:

- scope (+)
- quality (+)
- time (-)
- costs (-)

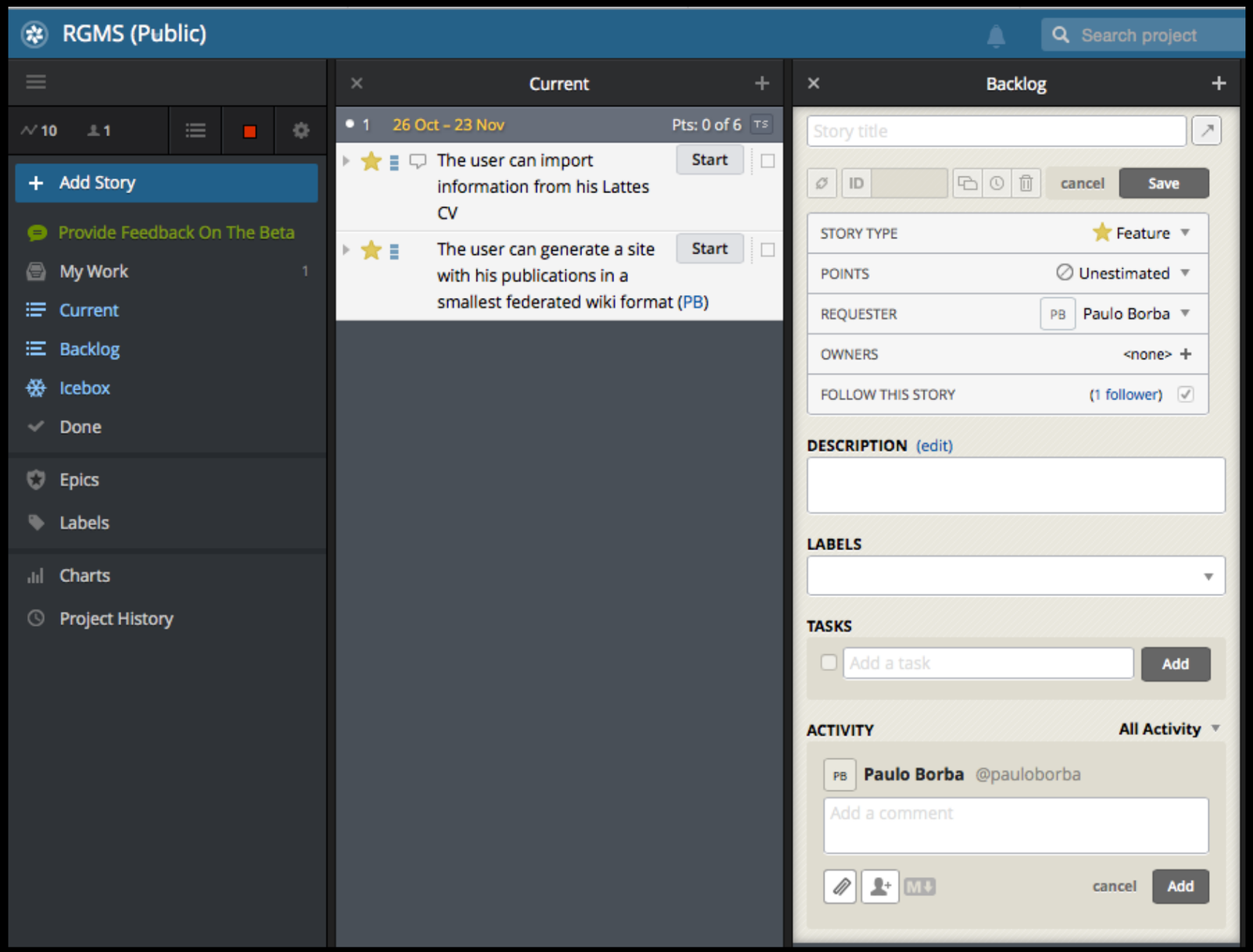
quality x cost/time
scope x cost/time



scope x quality
time x cost

Scope

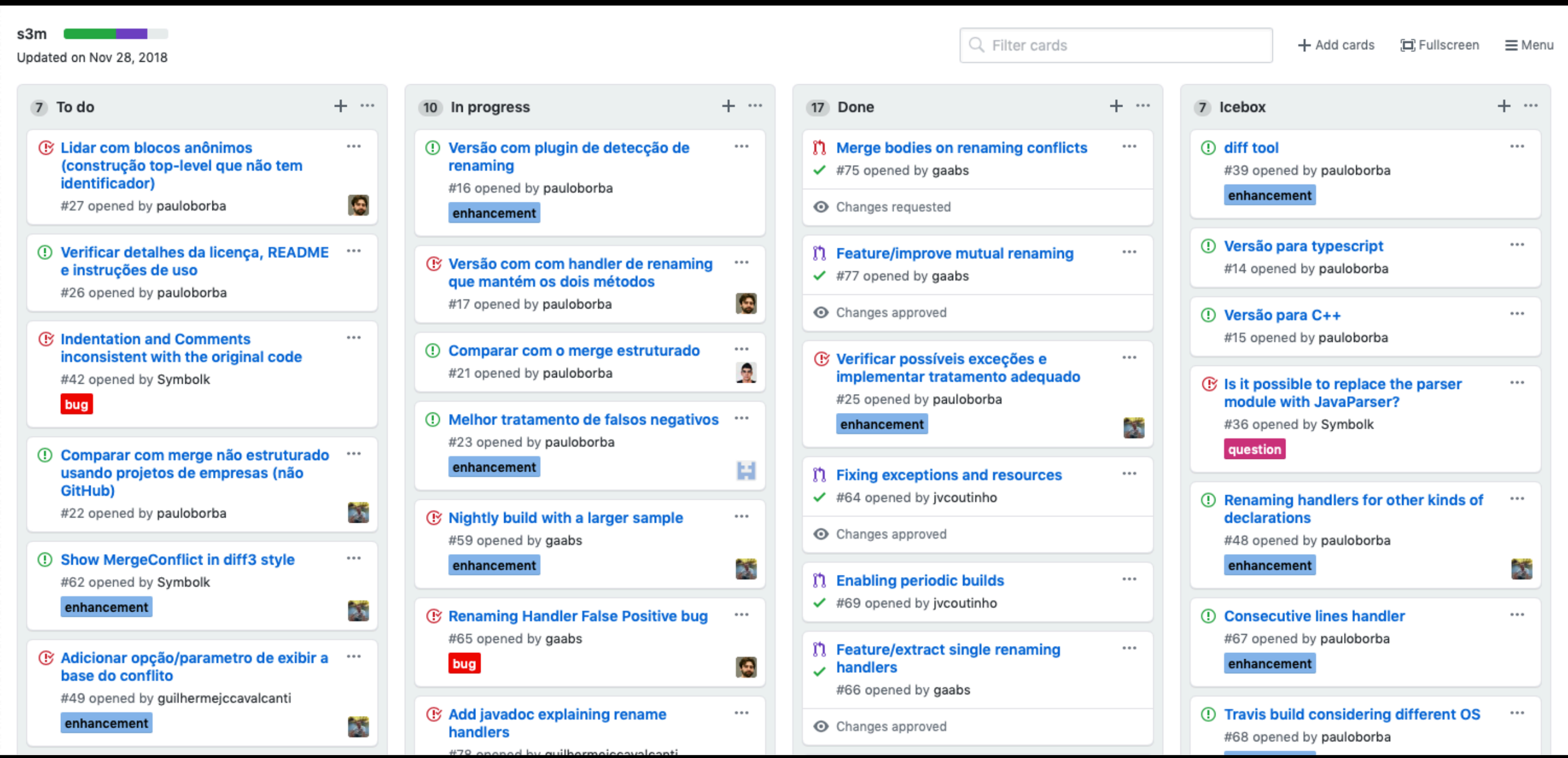
To control scope, we need to manage a list of tasks



Tasks

- Fix a bug
- Implement a new scenario or feature
- Change the implementation of an existing feature
- Refactor
- Improve performance
- Study, investigate, prepare, organize, etc.

Lists of tasks, really



For early and recurrent delivery and feedback, tasks are grouped in iterations (sprints)

feedback

feedback

feedback

feedback

Iteration 1

Iteration 2

Iteration 3

Iteration 4

Task 1

Task 2

Task 3

Task 4

Task 5

Task 6

Task 7

Task 8

Task 9

Task 10

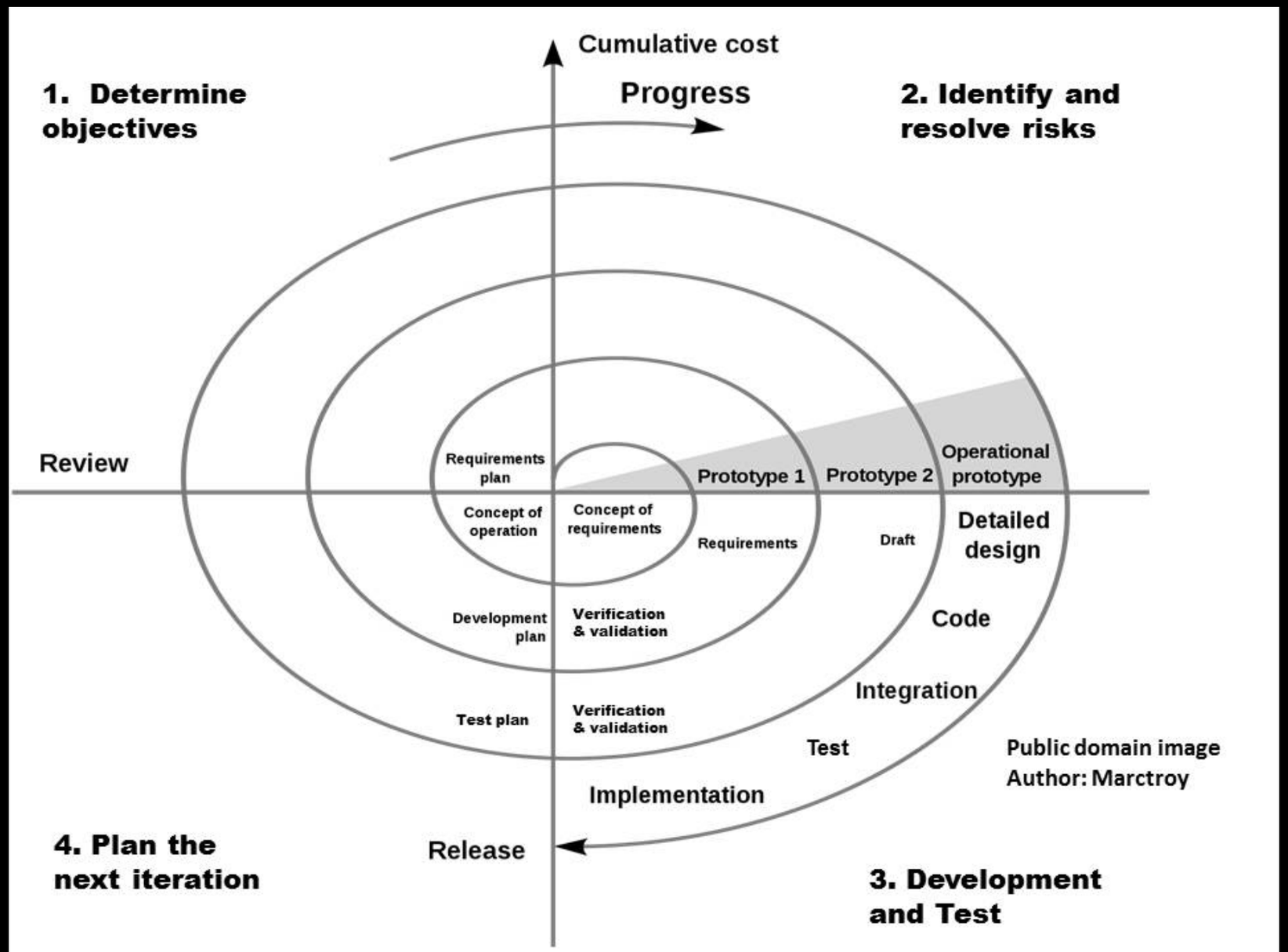
Task 11

Task 12

Iterations

- are defined by a (sprint) backlog, accordingly to stakeholders' priorities
- last from one to four weeks
- incrementally develop the system
- assign each task to a team member (owner)
- perform tasks in 4 to 16 hours

Software development models, spiral

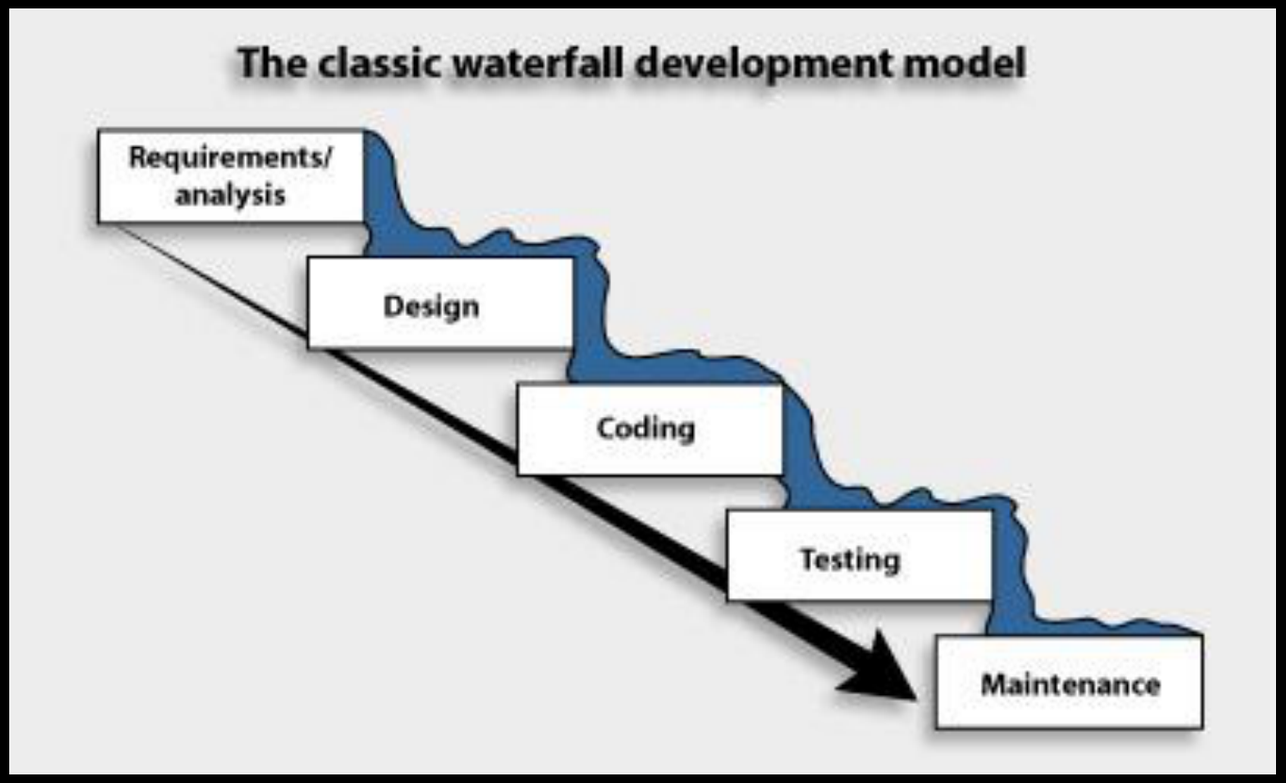


http://ultimatesdlc.com/spiral-model/?share=press-this

Diverse processes and domains

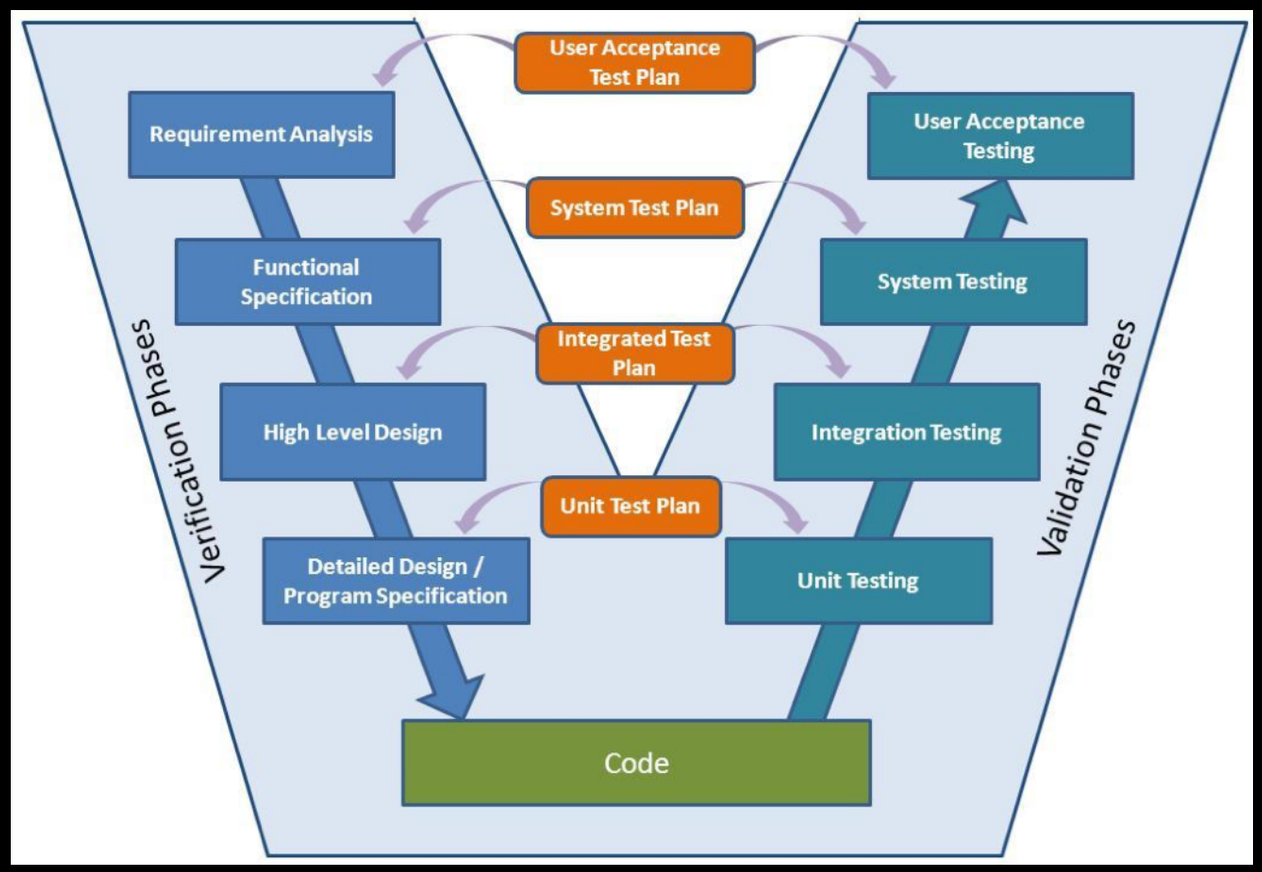
- Web systems
- Mobile applications
- Embedded software
- Startups
- Information systems
- Banks
- Google/Facebook/Microsoft/Amazon

Software development models, waterfall and V



http://04126030sasd.blogspot.com/2011/07/waterfall-model.html

misinterpreted!



http://crackmba.com/v-shaped-model/

BY MART VIRKUS

SOFTWARE DEVELOPMENT

THE NUTSHELL VIEW INTO SOFTWARE AGENCY MODELS

EXPLAINED WITH CARS

WATERFALL

TRADITIONAL, LINEAR PRODUCTION FLOW

WHAT THE CLIENT WANTS

CLIENT PLACES ORDER

HOW MUCH THEY THINK IT COSTS

DESIGN

PRODUCTION

THE "I KNOW HTML"

BECAUSE DEVELOPMENT PLANS NEVER CHANGE!

TESTING

SHIPPED & DONE!

AGILE DEVELOPMENT

TRANSPARENCY IS KEY

BUT AMOUNT OF PROBLEMS GROWS EXPONENTIALLY WITH THE SIZE OF THE CLIENT-SIDE TEAM

THE WEIRD LANGUAGES GUY

(COBOL, BASICALLY)

SMALL, INCREMENTAL CHANGES

CLIENT IS INVOLVED IN THE PROCESS

„INITIAL PLAN" GOES HERE

ADAPTABLE TO CHANGING CIRCUMSTANCES

KANBAN

TO-DO | IN ACTION | QA | DONE!

MAIL

(WAITING FOR PAYMENT.)

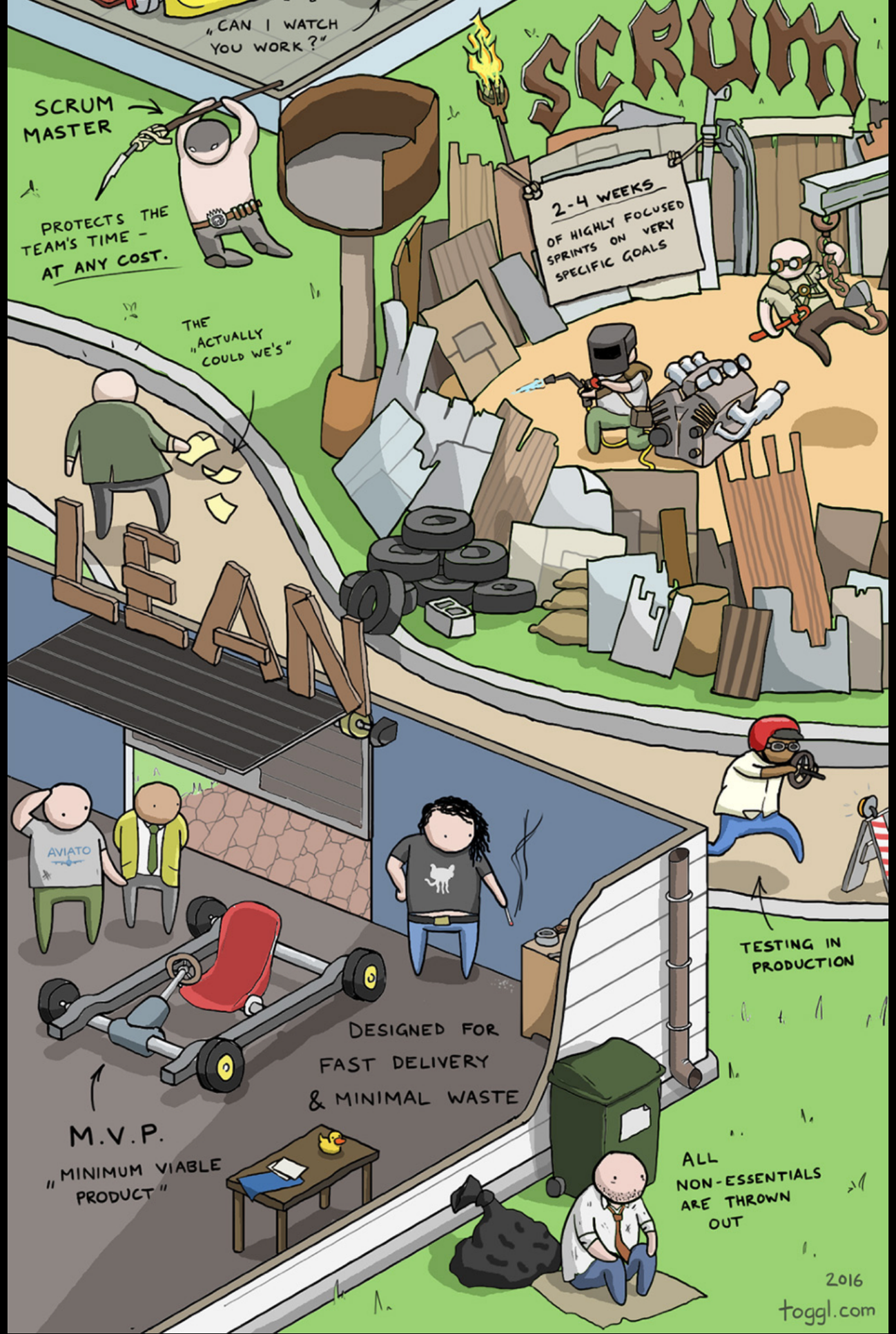
KANBAN SENSEI - MAKES SURE THE TEAM FOLLOWS THE TO-DO LIST

„CAN I WATCH YOU WORK?"

SCRUM MASTER

SCRUM AND KANBAN ARE TWO POPULAR METHODS FOR ~~ORGANISING WORK~~ AVOIDING CHAOS

SCRUM

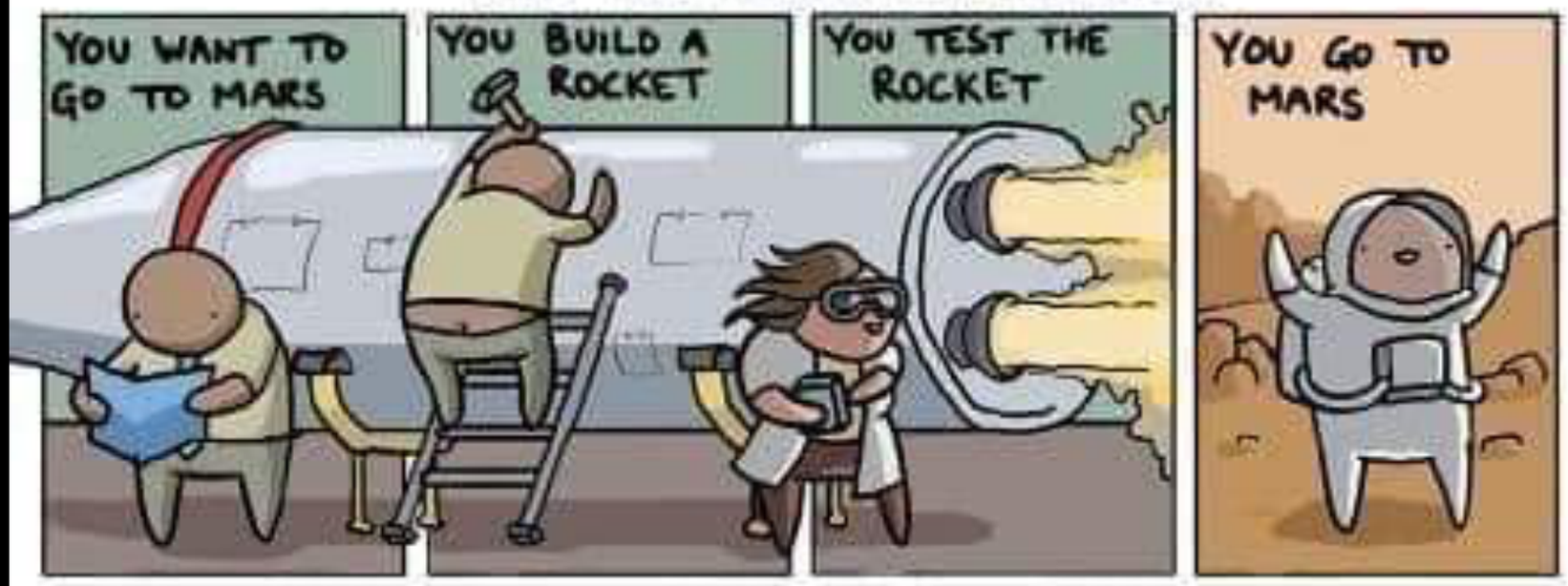


https://toggl.com/developer-methods-infographic

EXPLAINING SOFTWARE DEVELOPMENT METHODS BY FLYING TO MARS

toggl

THE WATERFALL METHOD

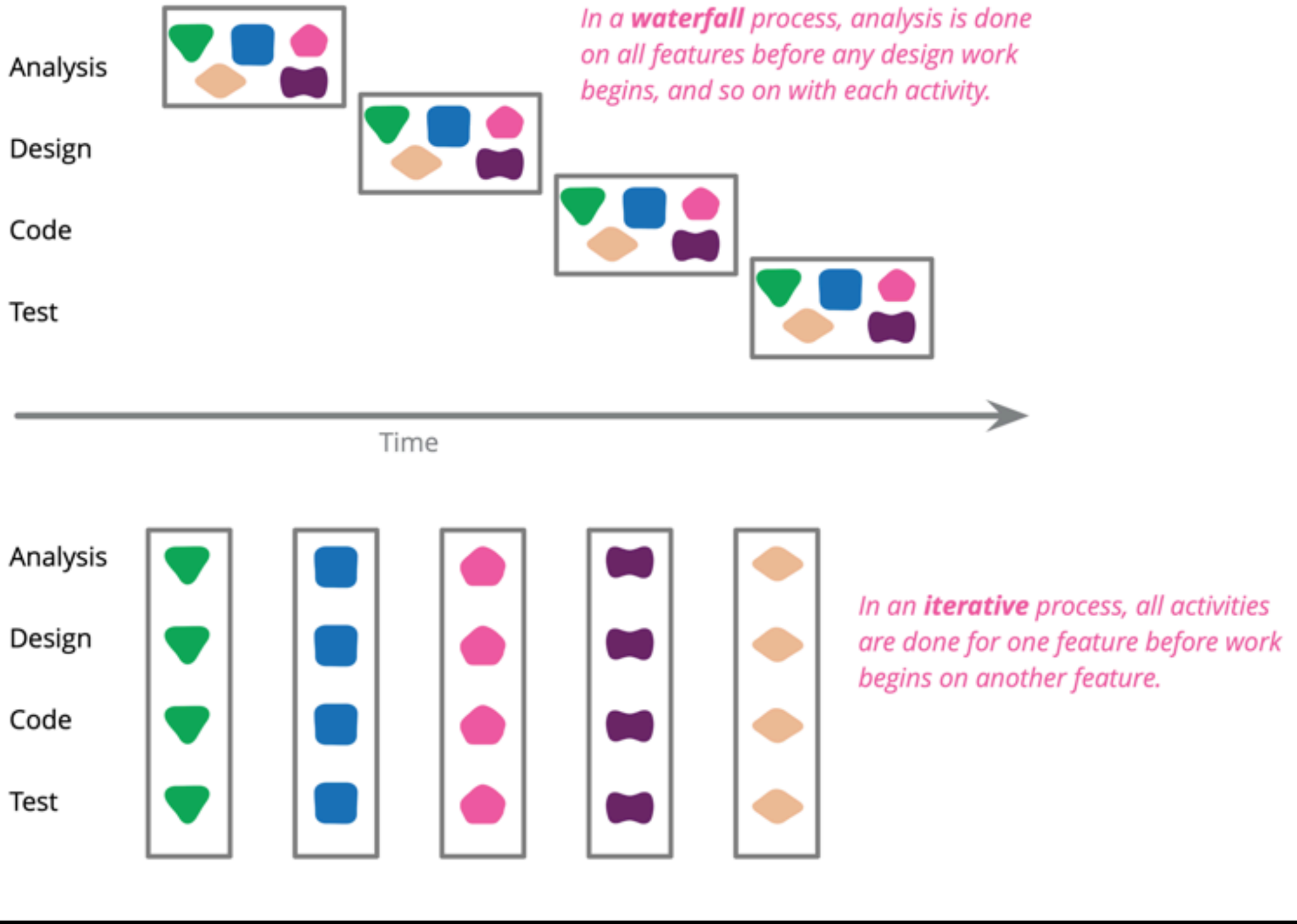


AGILE DEVELOPMENT

THE KANBAN METHOD

SCRUM

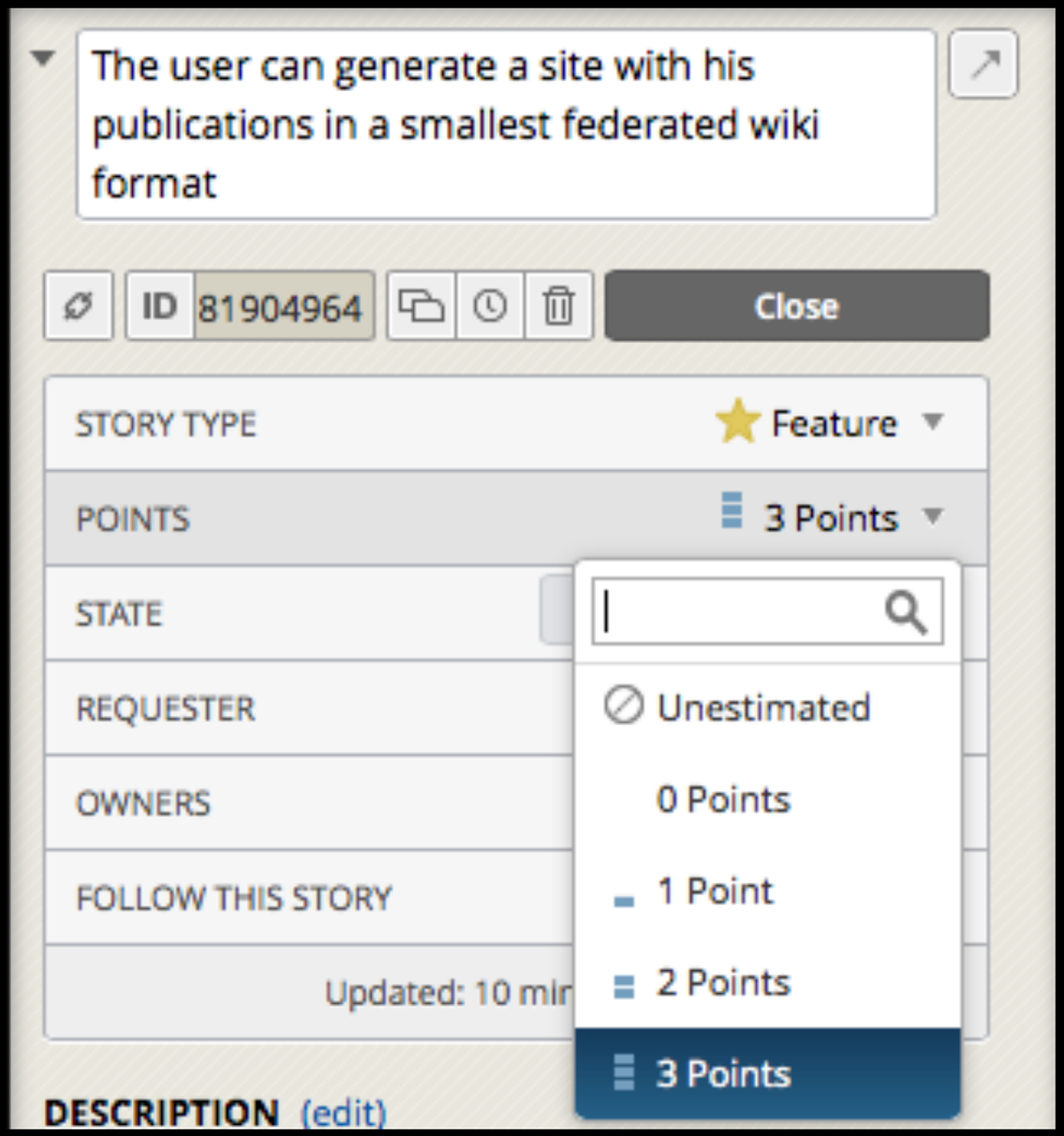
LEAN DEVELOPMENT



https://martinfowler.com/bliki/WaterfallProcess.html

Time

To control time, we need to manage task estimations…



story = task

Estimating by

- allocating points to tasks
  - starting with a simpler three-point scale
    - ground in concrete situation (work done in an ideal day)
- measuring the number of points per iteration
- finding out team velocity (the average number of points per iteration)

Iteration 1

1	3
Task 1	Task 2

Iteration 2

1	2	1	2
Task 3	Task 4	Task 5	Task 6

Iteration 3

2	2	2
Task 7	Task 8	Task 9

Iteration 4

1	1	2
Task 10	Task 11	Task 12

Focus on
development effort,
not time

Main purpose is to help make decisions, not be used as firm deadlines

Product backlog

- tasks with requester, estimation and priority
- created by the product owner (represents stakeholders interests)
- never complete, changed as needed by any team member
- priority tasks described in more detail
- epics help associate tasks to macro features

Allocating tasks in a iteration considering

- priorities (focus on business value)
- risks (more significant risks should be addressed first)
- team expertise (should match allocation)
- dependencies and potential conflicts (should be avoided)
- estimation and velocity

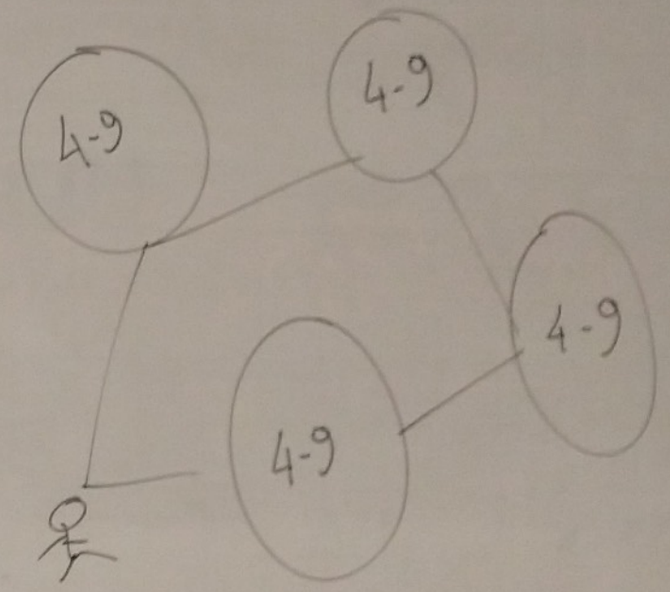
		Pontuação
A1	Cálculo média	2
A2	Auto-avaliação	3
A3	Importar Excel alunos	3
A4	Manutenção de Alunos	3
A5		2

prioridade

expertise

riscos

iteração



A1 provides A2 with class Aluno

Planning game

- negotiation between business priorities and technical priorities and estimations
- business people define
  - releases, dates, scope, priorities
- technical people define
  - estimations, process, detailed release schedule

Take notes,

now!

Hands on exercises

Project management I

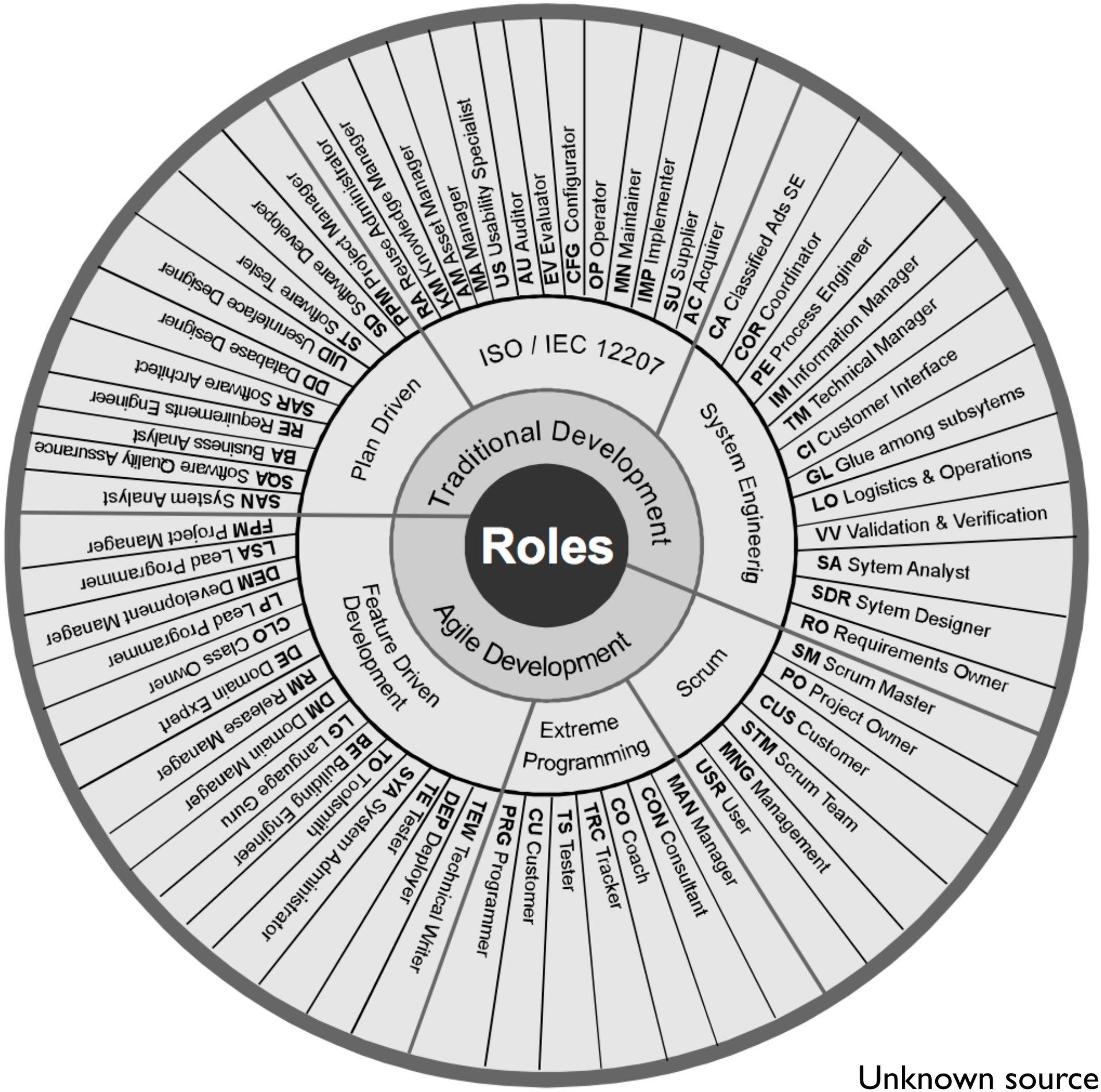
Project management II, team and collaboration

Team size

- 4 to 9 people
- common organization at Pivotal:
  - six software engineers, an interaction designer, and a product manager

Roles at Pivotal

- Interaction designers
  - requirements elicitation and validation; visual design
- Product managers
  - feature identification and prioritization, requirements specification, planning, and communicating specifications to engineers
- Software engineers
  - implement the solution and test



Unknown source

Common desired skills

- Communicate precisely
- Divide work
- Manage complexity
- Understand, choose and use proper tools
- Understand social context

Choosing people for the team and each task

- Considering a number of aspects:
  - technical,
  - experience,
  - collaboration, and
  - personality dimensions

Team dynamics are complex and managing people is difficult

Planning and increasing team awareness

- run daily (scrum) meetings (15 min)
  - done? to do? obstacles?
- planning meetings
  - defines priorities and sprint contents and deadlines
  - triage

Scrum meeting benefits and drawbacks

- information sharing, improves collaboration and coordination, team awareness, avoids conflicts, or reveal them earlier
- opportunity to discuss and solve (very quickly) problems
- team cohesion, spirit, culture and training

- boring status reporting (just drop one of the questions)
- too frequent and too long (just reduce frequency)
- disruptive, interruption (as any meeting, no solution)
- delay to start, to finish (just develop culture)

Continuously improving

- review meetings
  - discuss implemented features in a sprint, and changes to the product backlog
- retrospective meetings
  - discuss good and bad aspects of a sprint

facilitated by the Scrum master

Retrospectives at Pivotal

- emotion-based retro format with "happy" (working well), "neutral" (should monitor) and "sad" (problem to fix) faces columns
- any team member can add any topic to any column
- the team dot-votes on the topics to discuss
- topics are discussed (60 minutes) and action items are created

Minimizing team member conflicts and development effort

- Define task interfaces
- Need to change interfaces should be first discussed
- Same approach for the need to add new interfaces
- Automated testing

Broader notion of interface

Information hiding and interfaces (David Parnas, CACM 1972)

Every module in the second decomposition is characterized by its knowledge of a design decision which it hides from all others. Its interface or definition was chosen to reveal as little as possible about its inner workings.

Not necessarily public methods!

Not necessarily data representation!

Interfaces as a means to achieve independent development

Which interface should we specify if we want to independently develop features for importing student data from a file, and registering students through a form?

Large systems and collaborating teams

- Feature development and maintenance is performed by a team of developers from different component (subsystem) teams
- Component teams specialise on the functionalities of a given component, and collaborate to the development of features that require functionalities related to that component

Take notes,

now!

Hands on exercises

Project management II

Project management III, quality and cost

Quality

To control quality, we need to…

- test the system
- review artifacts
- pair programming
- measure quality attributes
- static analysis tools
- reduce technical debt

Separate environments for different activities

- Development
- Testing (integration, system, etc.)
  - Daily build and smoke test
- Execution (deployment)

Review (inspect) artifacts

- in a tool
- focus on most relevant points
- guide yourself by clear goals (checklist)
- be objective
- be kind: critique the artifact rather than its author
- be constructive

Requirements specification review

- The specification is syntactically valid and can be executed by Cucumber tools
- The specification is clear and has no English spelling and syntactic mistakes
- The commits/PR clearly convey the intention of the change, and what has been changed
- The specification conforms to the requirements specification checklist

Similarly for other software development artifacts: tests, code, design, documentation

Costs

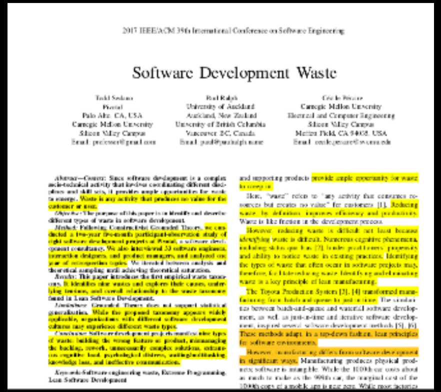
cost estimation

=

team size and cost for a given period (no promises of delivered features)

Fixing both scope and schedule is often antithetical for systems that are not similar to previously developed systems.

"The client either chooses the release date and gets only the features ready by then or chooses key features and ships the product when the features are ready."


2017 IEEE/ACM 39th International Conference on Software Engineering

Software Development Waste

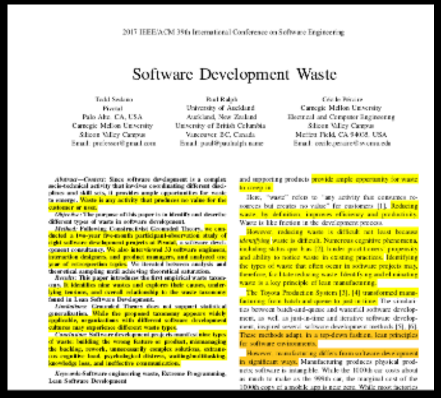
To control costs, we need to…

- control scope and quality, avoiding increased scope and quality expectations
- control time, avoiding delivery anticipation
- manage development productivity
  - team skills, tools
  - process, motivation
  - reducing waste

reducing waste

(unnecessary activity that produces no value for the stakeholders)

improves productivity, by definition


2017 IEEE/ACM 39th International Conference on Software Engineering

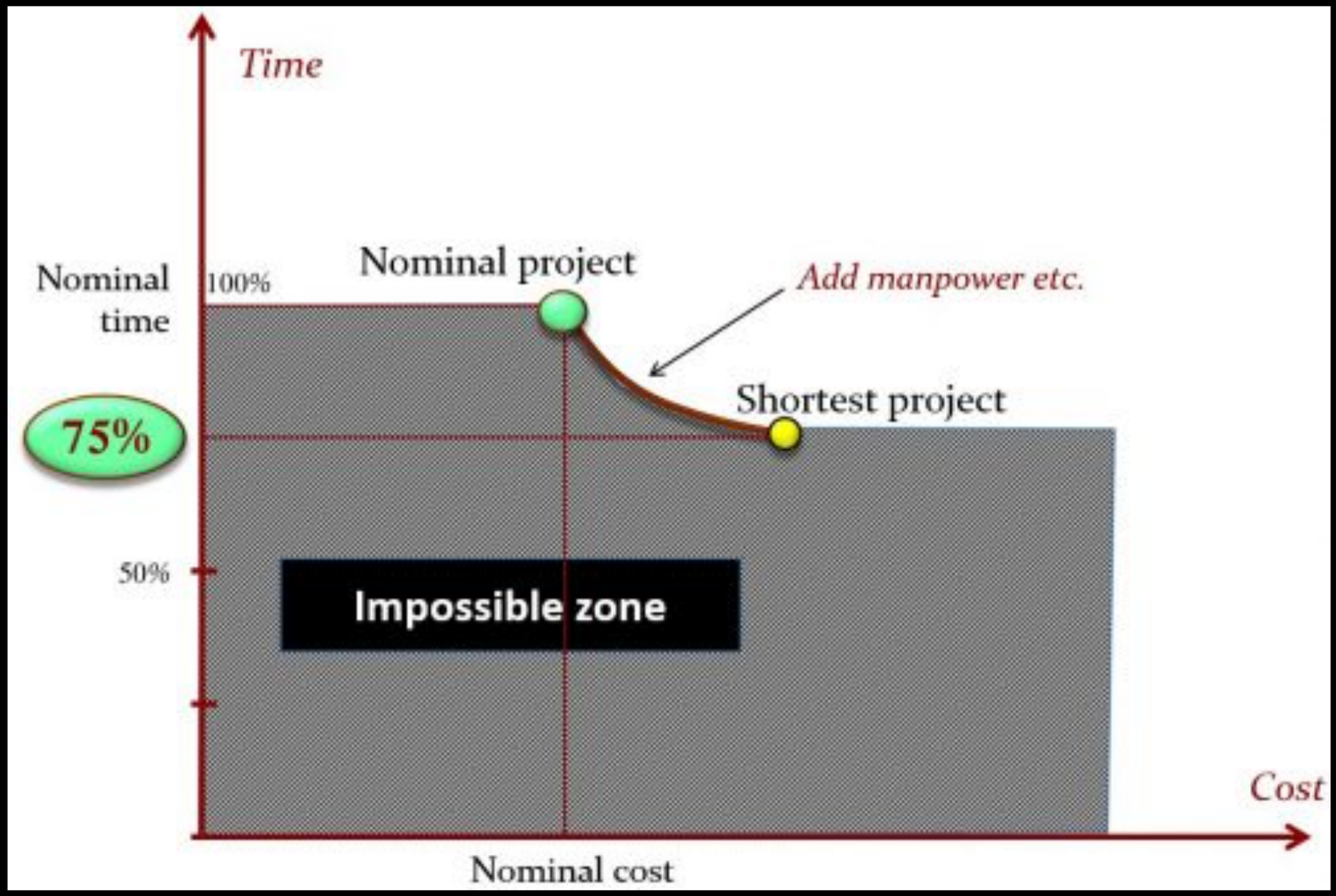
Software Development Waste

Waste Category: Psychological Distress
- Cause Property: Low team morale
  - Retro Topic: Frustrated developers
  - Retro Topic: Not managing expectations
  - Retro Topic: Negative attitudes
  - Retro Topic: Apathy
  - Retro Topic: Not knowing everyone on the team
  - Retro Topic: Project feels like it is falling apart emotionally
  - Retro Topic: Unacknowledged by management
  - Retro Topic: Messy code decreasing sense of ownership
- Cause Property: Rush mode
  - Retro Topic: Fixed set of features with a fixed timeline
  - Retro Topic: Aggressive timelines
  - Retro Topic: Shifting deadline
  - Retro Topic: Scope creep
  - Retro Topic: Repeatedly hearing “This is due today”
  - Retro Topic: Long days
  - Retro Topic: Overtime
- Cause Property: Interpersonal or team conflict
  - Retro Topic: Criticizing in public
  - Retro Topic: Difficult pairings
  - Retro Topic: Pairing fatigue
  - Retro Topic: Not listening
  - Retro Topic: Interpersonal conflict

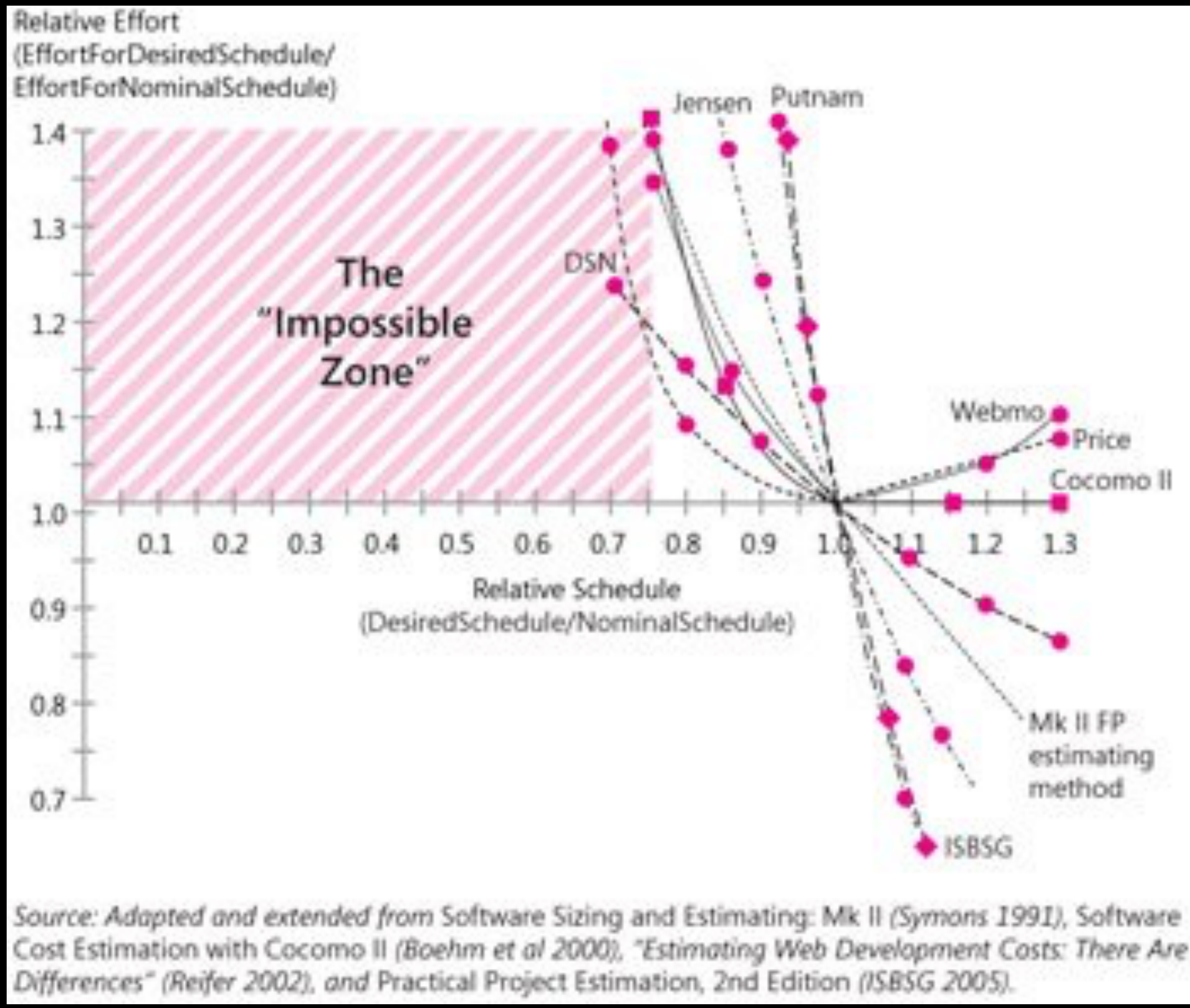
Waste	Description	Observed Causes
Building the wrong feature or product	The cost of building a feature or product that does not address user or business needs.	User desiderata (not doing user research, validation, or testing; ignoring user feedback; working on low user value features) Business desiderata (not involving a business stakeholder; slow stakeholder feedback; unclear product priorities)
Mismanaging the backlog	The cost of duplicating work, expediting lower value user features, or delaying necessary bug fixes.	Backlog inversion Working on too many features simultaneously Duplicated work Not enough ready stories Imbalance of feature work and bug fixing Delaying testing or critical bug fixing Capricious thrashing
Rework	The cost of altering delivered work that should have been done correctly but was not.	Technical debt Rejected stories (e.g. product manager rejects story implementation) No clear definition of done (ambiguous stories; second guessing design mocks) Defects (poor testing strategy; no root-cause analysis on bugs)
Unnecessarily complex solutions	The cost of creating a more complicated solution than necessary, a missed opportunity to simplify features, user interface, or code.	Unnecessary feature complexity from the user's perspective Unnecessary technical complexity (duplicating code, lack of interaction design reuse, overly complex technical design created up-front)
Extraneous cognitive load	The costs of unneeded expenditure of mental energy.	Suffering from technical debt Complex or large stories Inefficient tools and problematic APIs, libraries, and frameworks Unnecessary context switching Inefficient development flow Poorly organized code
Psychological distress	The costs of burdening the team with unhelpful stress.	Low team morale Rush mode Interpersonal or team conflict
Waiting/multitasking	The cost of idle time, often hidden by multi-tasking.	Slow tests or unreliable tests Unreliable acceptance environment Missing information, people, or equipment Context switching from delayed feedback
Knowledge loss	The cost of re-acquiring information that the team once knew.	Team churn Knowledge silos
Ineffective communication	The cost of incomplete, incorrect, misleading, inefficient, or absent communication.	Team size is too large Asynchronous communication (distributed teams; distributed stakeholders; dependency on another team; opaque processes outside team) Imbalance (dominating the conversation; not listening) Inefficient meetings (lack of focus; skipping retros; not discussing blockers each day; meetings running over (e.g. long stand-ups))

Brooks' Law: adding people to a late project delays it further

The Shortest Possible Schedule law



https://bertrandmeyer.com/2019/11/01/theorem-software-engineering/



Source: Adapted and extended from Software Sizing and Estimating: Mk II *(Symons 1991)*, Software Cost Estimation with Cocomo II *(Boehm et al 2000)*, *"Estimating Web Development Costs: There Are Differences" (Reifer 2002), and* Practical Project Estimation, 2nd Edition *(ISBSG 2005)*.

you can, within limits,
shorten delivery times
by properly bringing
people to the project

Checklist

- Define tasks and epics with clear focus, and granularity corresponding to a group of related scenarios
- Carefully estimate and measure velocity
- Daily meetings (at least virtual meetings at the slack channel)
- Plan ahead, early report delays

Take notes,

now!

Hands on exercises

Project management III

Project management research at CIn

- Team motivation: Fabio
- Software process: Alexandre Vasconcelos and Hermano
- Task scheduling: Paulo

To do after class

- Answer questionnaire (check classroom assignment), study correct answers
- Finish exercise (check classroom assignment), study correct answers
- Read, again, parts of chapters 7 and 10 in the textbook
- Evaluate classes
- Study questions from previous exams

Questions from previous exams

- Como um gerente de projetos que segue metodologias ágeis, explique brevemente o que você faria para controlar o tempo de desenvolvimento de um software sob responsabilidade da sua equipe.
- Explique brevemente o que é "velocity" e como a mesma pode ser usada para estimativas.
- Quais as vantagens de desenvolver o software em iterações ou sprints curtas?
- Explique quais as dimensões que um gerente de projeto tem que controlar, e cite duas consequências negativas de não conseguir controlá-las adequadamente?

Software and systems engineering

Paulo Borba
Informatics Center
Federal University of Pernambuco

phmb@cin.ufpe.br ◈ twitter.com/pauloborba