Information systems development & agile methods

Management Information Systems course

Antti Salovaara

http://people.aalto.fi/antti_salovaara

Department of Design, Aalto ARTS

Learning objectives

Understand why software development is iterative (in many levels)

Know basics about the main software development process models

- Waterfall
- Iterative waterfall
- Agile

Introduction to

the *error-prone* nature of programming

and

***iteration* as its primary solution**

Exact Instructions Challenge - THIS is why my kids hate me. | Josh Darnit
https://www.youtube.com/watch?v=cDA3_5982h8

Lessons from the video

It is very easy to make errors (“bugs”) in programming

- Testing and “debugging” take a lot of time (50% of time is normal)

The bug-prone nature of programming makes scheduling and software project management very difficult

Software complexity

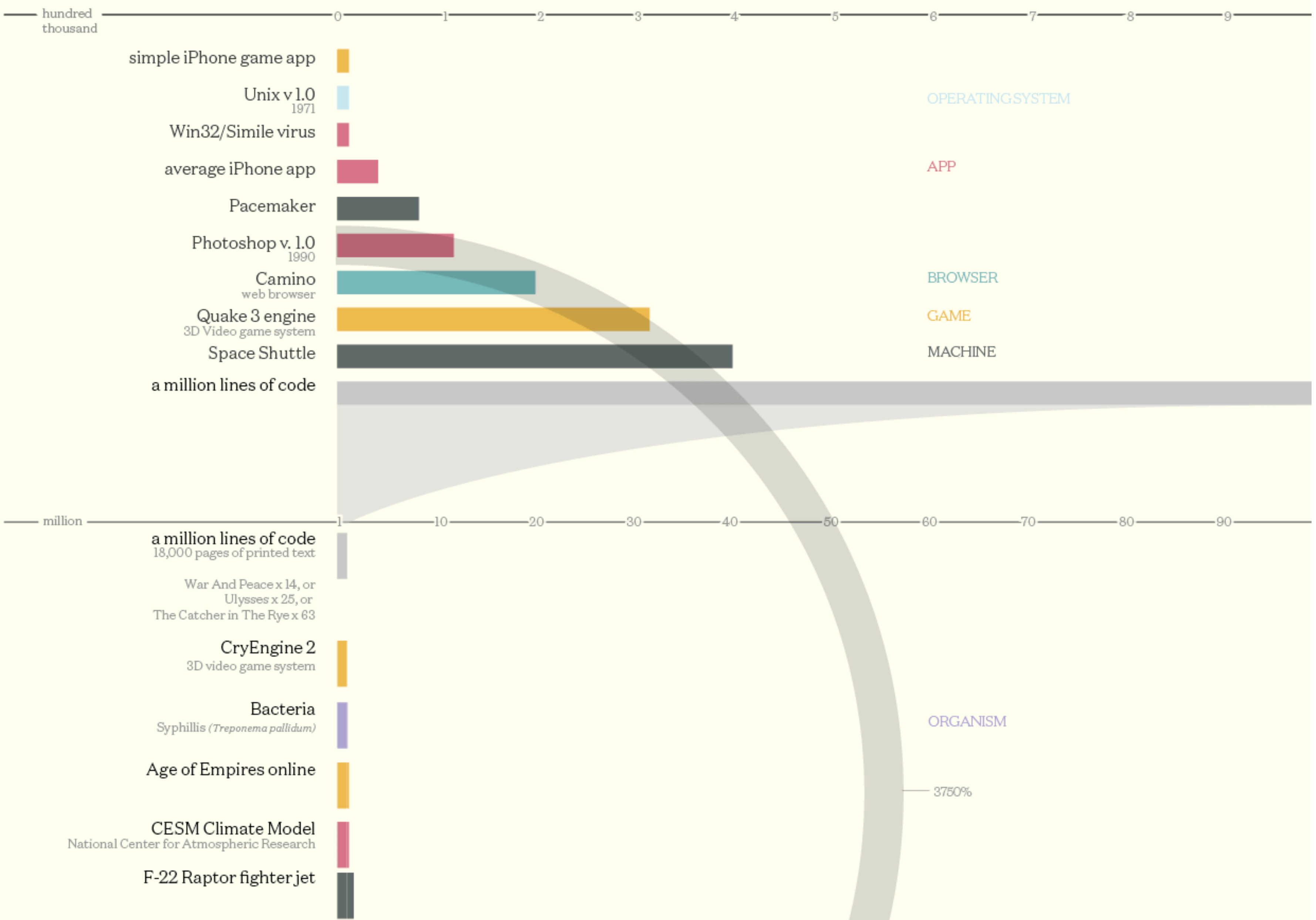
	Lines of code:
QuikSort algorithm for list sorting	20
Average iPhone app	15,000
World of Warcraft (server side)	5,000,000
Google Chrome browser	8,000,000
Boeing 787	14,000,000
Microsoft Office 2013	44,000,000
Large Hadron Collider @ CERN	50,000,000
Facebook	62,000,000
All Google internet services in 2015	2,000,000,000

http://www.wired.com/2015/09/google-2-billion-lines-codeand-one-place/
http://www.informationisbeautiful.net/visualizations/million-lines-of-code/

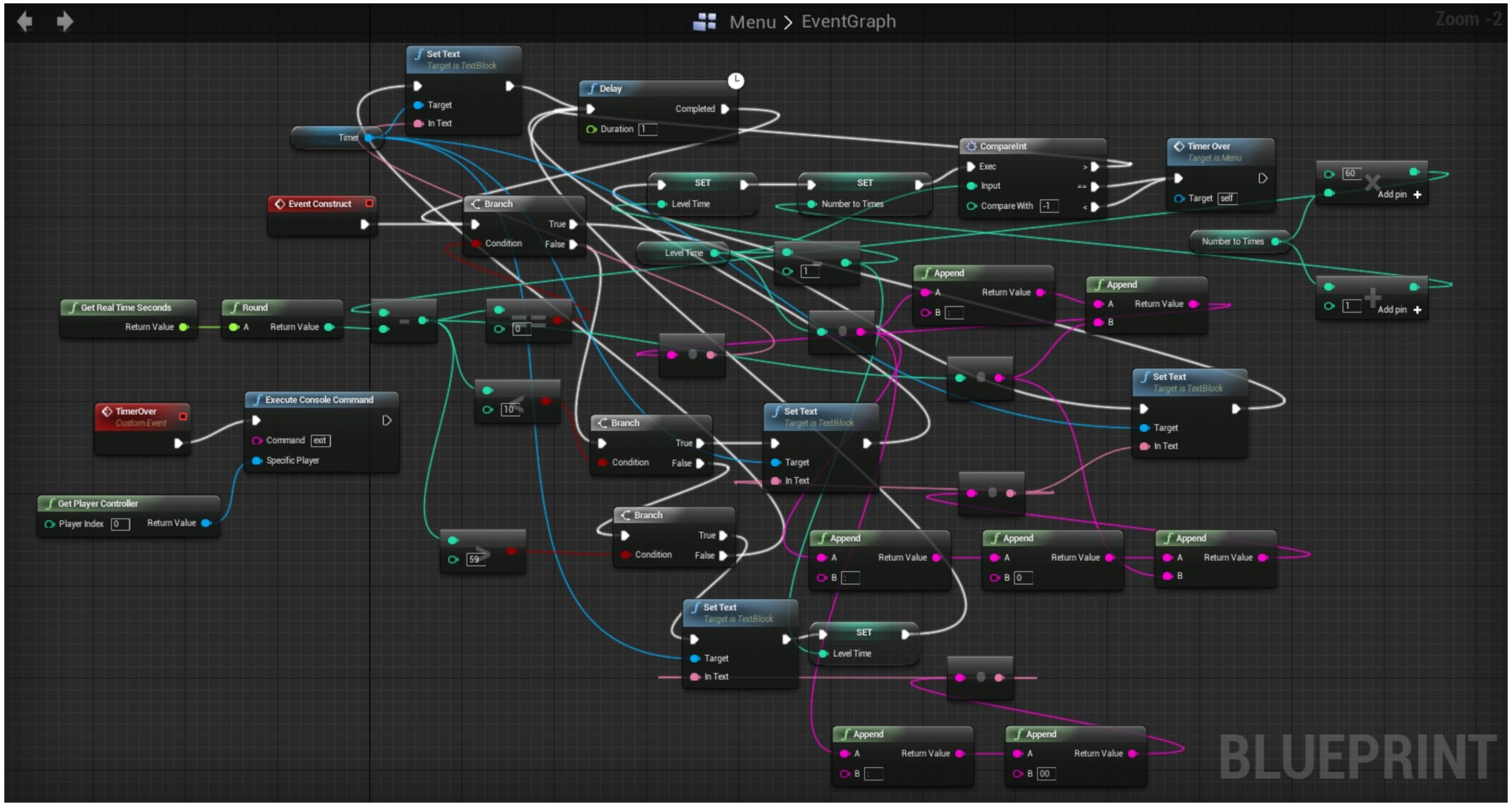
Codebases

Millions of lines of code

http://www.informationisbeautiful.net/visualizations/million-lines-of-code/



What can happen when new features are added and the code base grows: “Spaghetti code”



“A visual representation of Spaghetti code using nodes in Unreal Engine 4.”
https://medium.com/@BitBandit/the-quest-to-avoid-spaghetti-code-c8913e1a5527

The ninety-ninety rule

“

The first 90 percent of the code accounts for the first 90 percent of the development time.

The remaining 10 percent of the code accounts for the other 90 percent of the development time.

”

– Tom Cargill, Bell Labs

What can be done to bugs and spaghetti?

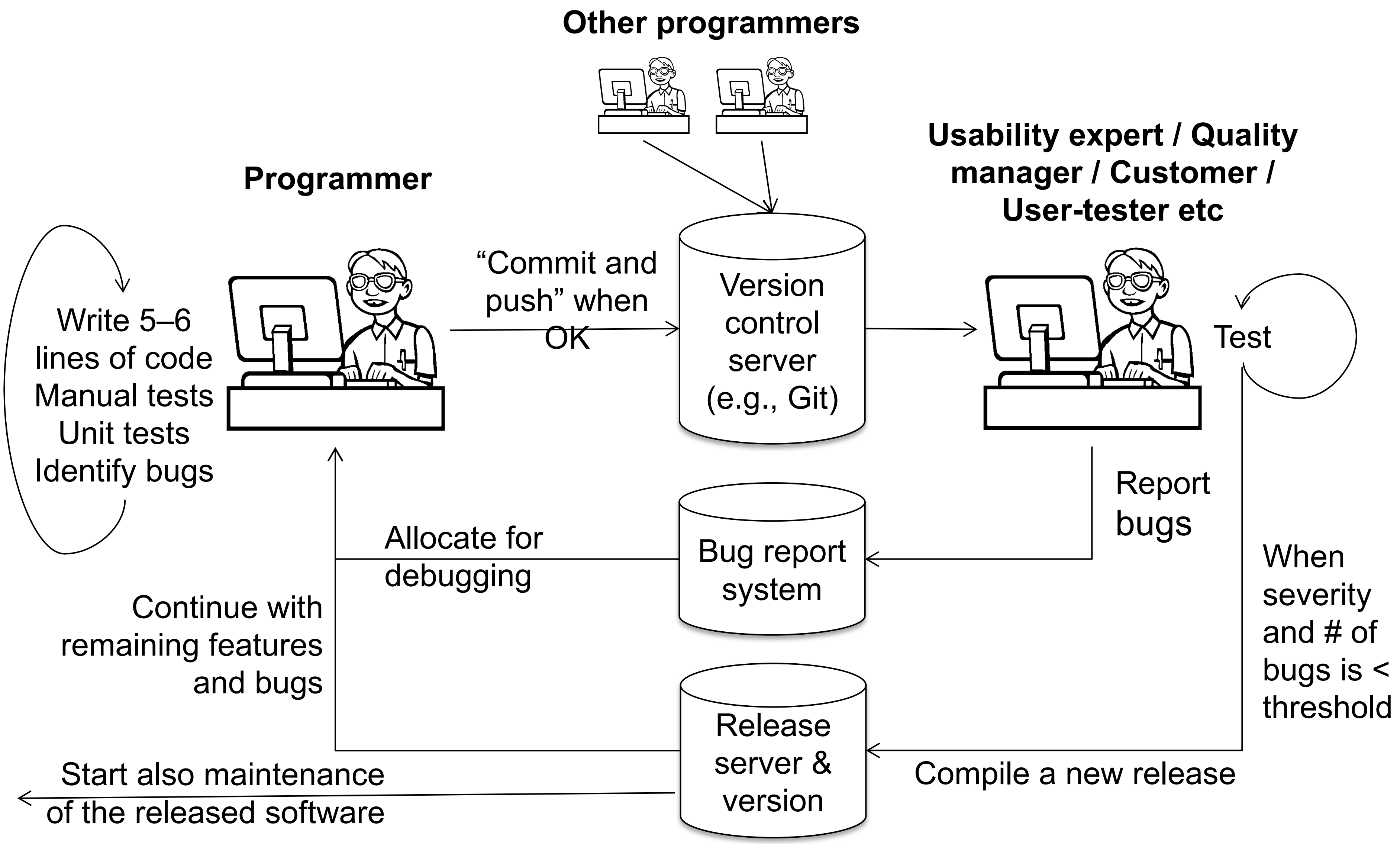
Spaghetti

Ravioli

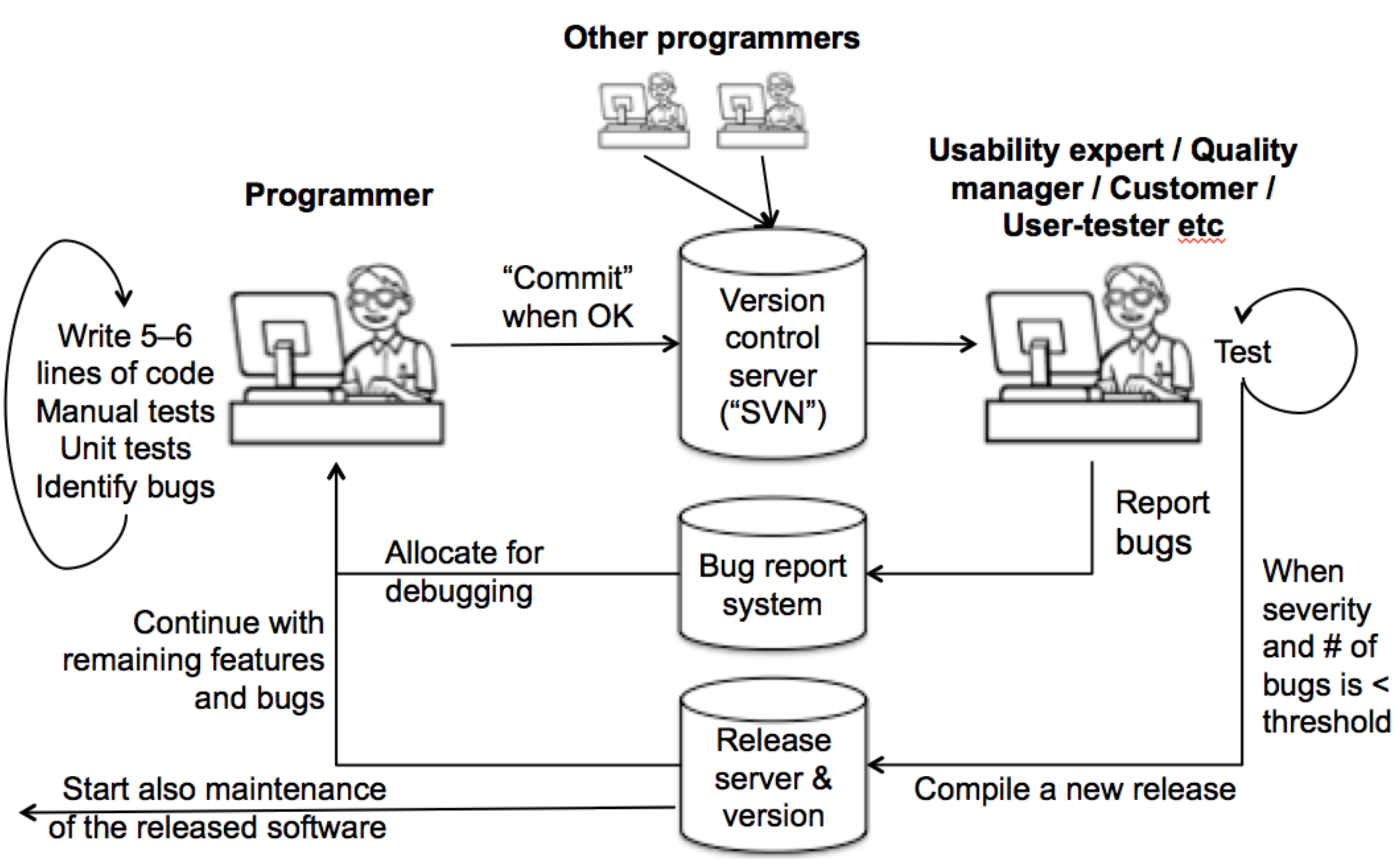
Methods and solutions:

- **Modularity:** Organize software into reusable modules (objects and classes, libraries, functions, etc.)
  - "Ravioli code"
- **Iterative** working style
  - E.g. write only 5 lines. Then test them before you continue
- **Test-driven development**: "unit tests"
  - = short code fragments that automatically test your program code
  - Unit tests can be written before the actual program code

Iteration in practice in a small project


Other programmers
Programmer
Usability expert / Quality manager / Customer / User-tester etc
Write 5–6 lines of code
Manual tests
Unit tests
Identify bugs
"Commit and push" when OK
Version control server (e.g., Git)
Test
Report bugs
Allocate for debugging
Bug report system
When severity and # of bugs is < threshold
Continue with remaining features and bugs
Release server & version
Compile a new release
Start also maintenance of the released software

Other involved stakeholders


Other programmers
Programmer
Usability expert / Quality manager / Customer / User-tester etc
Write 5–6 lines of code
Manual tests
Unit tests
Identify bugs
"Commit" when OK
Version control server ("SVN")
Test
Report bugs
Bug report system
Allocate for debugging
Continue with remaining features and bugs
When severity and # of bugs is < threshold
Release server & version
Compile a new release
Start also maintenance of the released software

Main software development models

Differences in planning processes

Pair discussion **5 mins** + general discussion **5 mins**

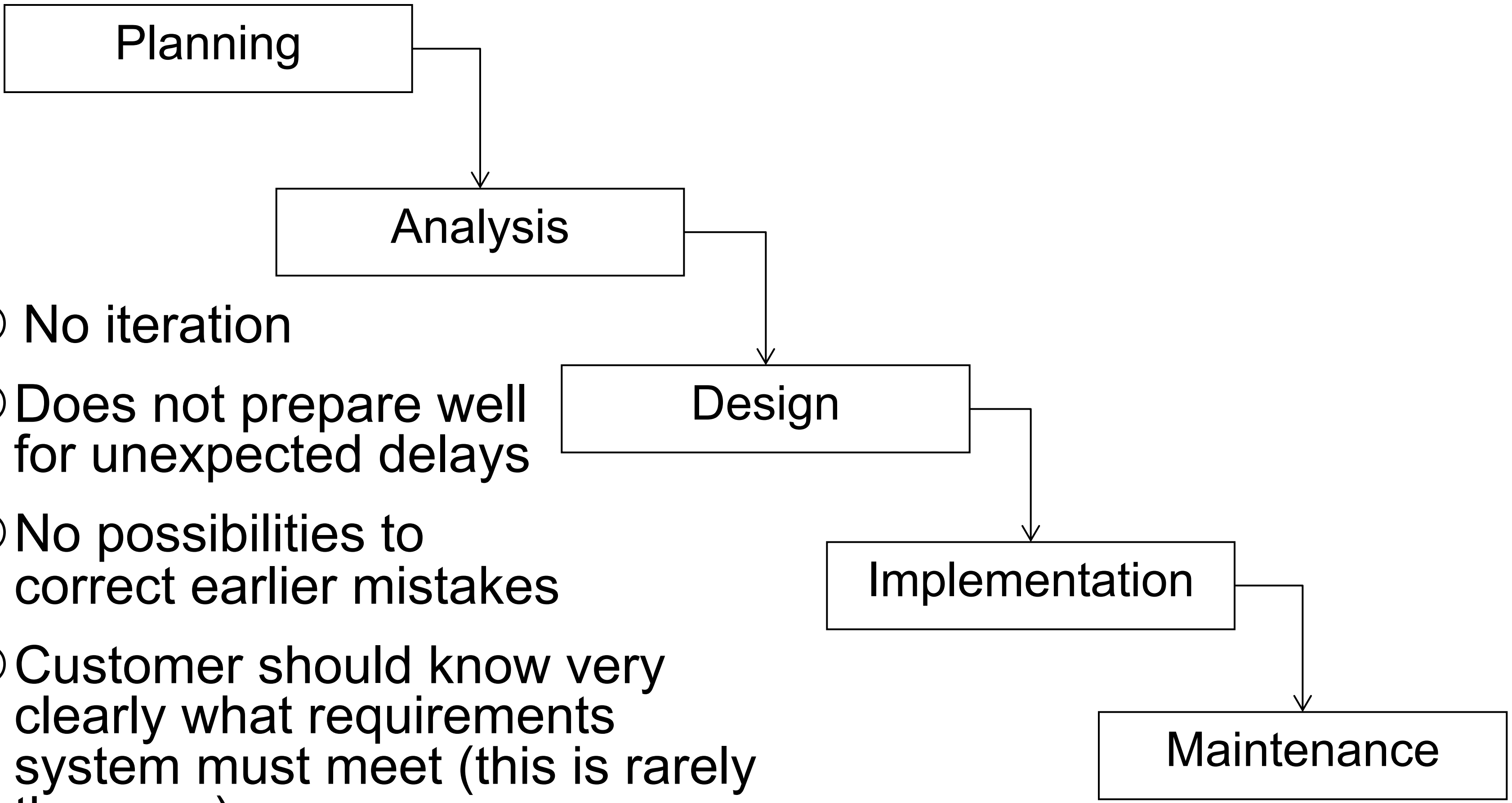
What process would be good for…

vs.

… development of a popular menu for a new restaurant?

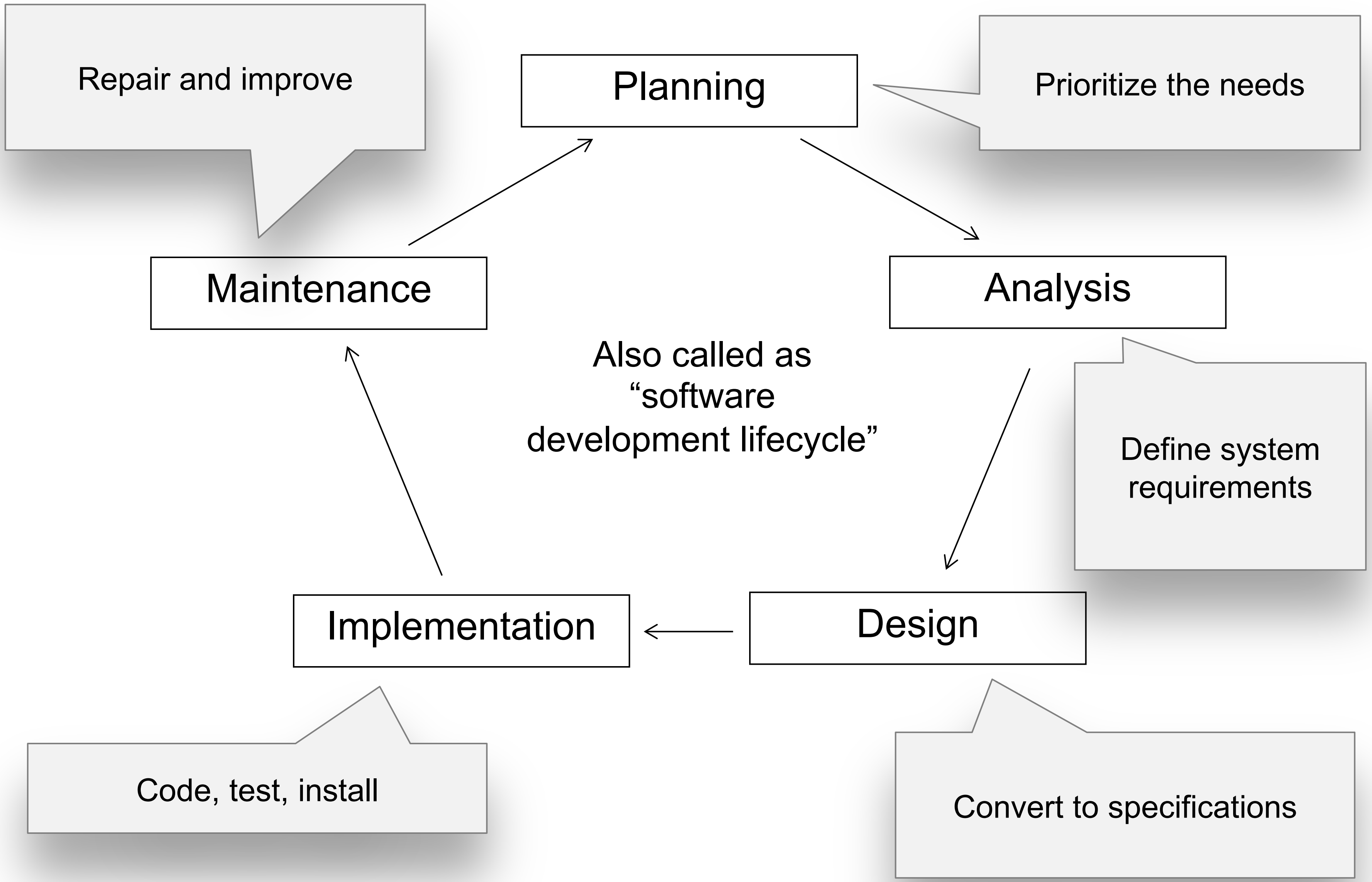
… building a bridge?

Traditional waterfall



- ☹ No iteration
- ☹ Does not prepare well for unexpected delays
- ☹ No possibilities to correct earlier mistakes
- ☹ Customer should know very clearly what requirements system must meet (this is rarely the case)
- ☹ During the project it is difficult to see how good the product will be

Iterative waterfall



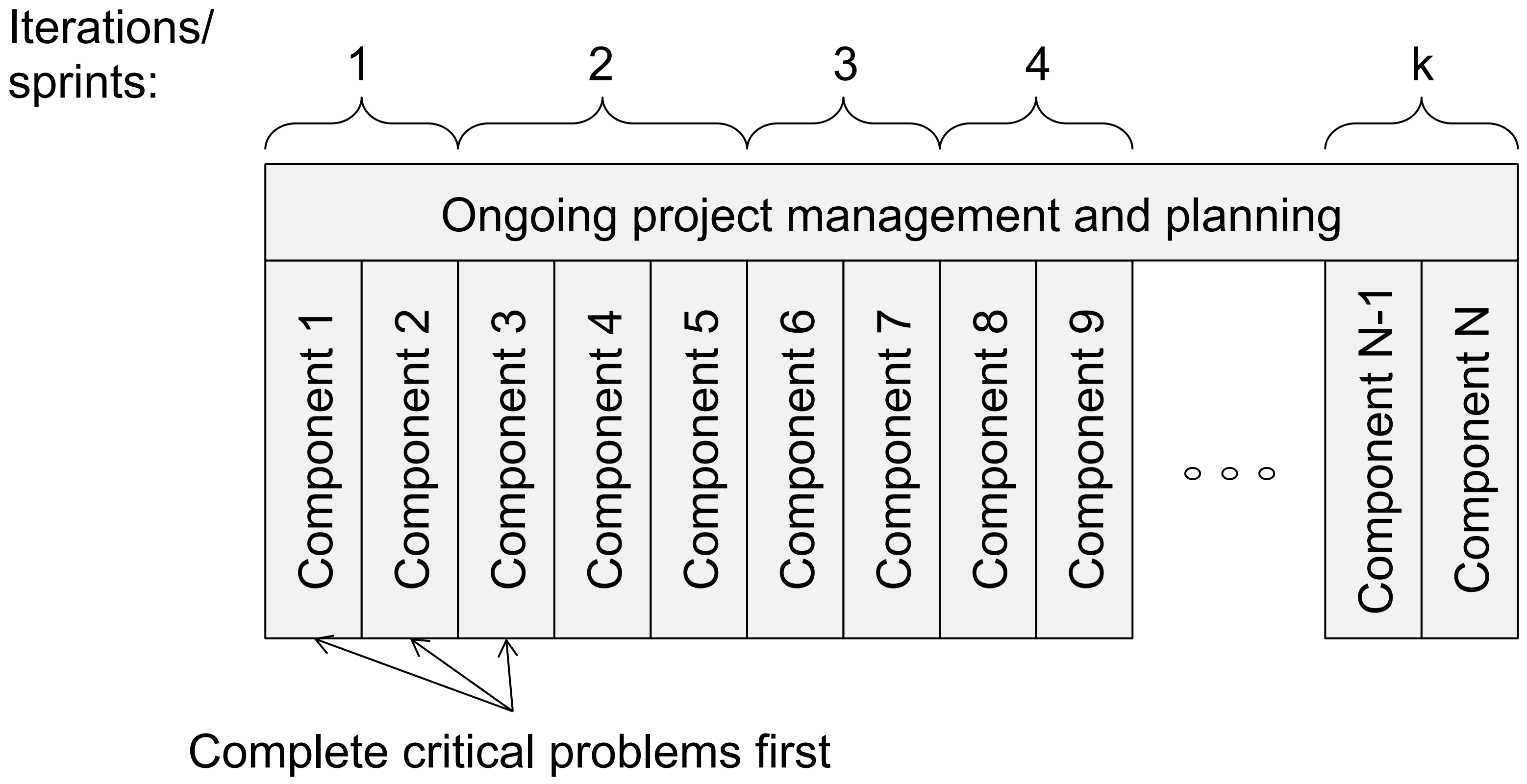
Agile development

1. Customer satisfaction by **early and continuous delivery** of valuable software
2. **Welcome changing requirements**, even in late development
3. Working software is delivered 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 principal 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

http://www.agilemanifesto.org/principles.html

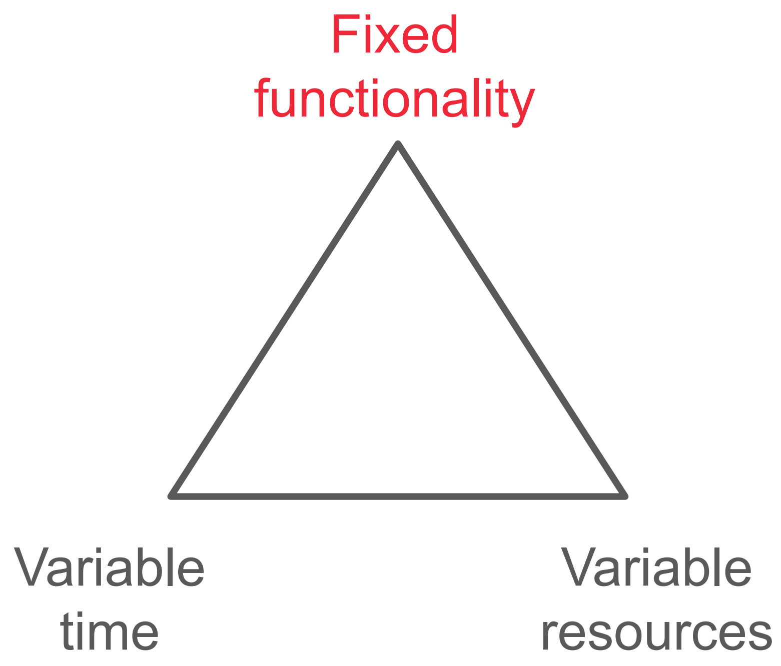
Vertical systems development

Every iteration focuses on creating functional components



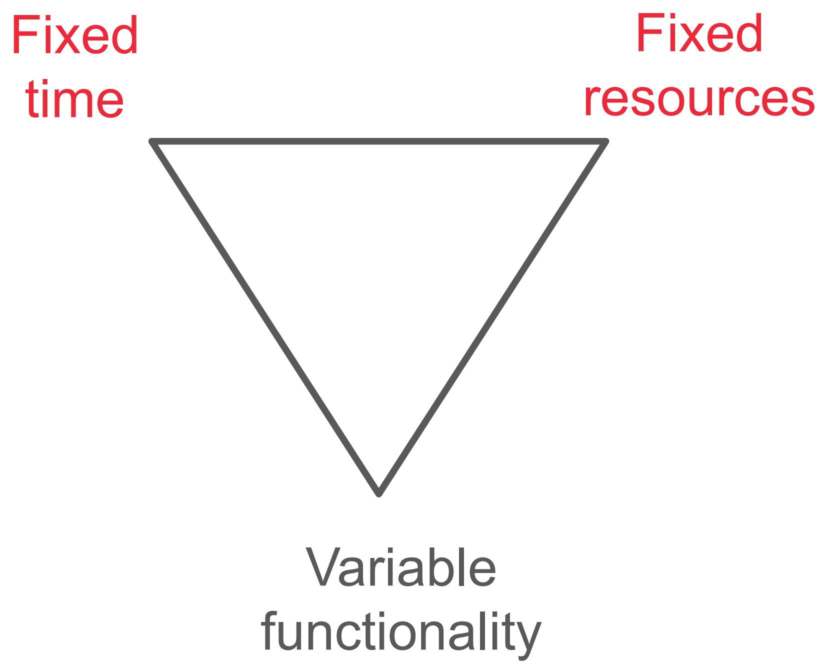
Timeboxing

Fixed feature set approach (e.g., in waterfall):



➔ Overruns in time and costs ☹

Agile timeboxed approach:



➔ Agility + support for learning

A story from one project

“ One state bureau summarized its recent experiences of an agile project.

In its end, they wanted to know, out of curiosity, **what percentage of their original plans had changed during the project.**

When they did this comparison, they noticed that in the end, only **14% of the work had been correctly specified in the beginning**.

All the other work had been identified during the project.

On the other hand, of all the originally specified projects, the **customer finally wanted to have only 23%**.

It decided that it did not need the other parts, and those parts were therefore left undone. ”

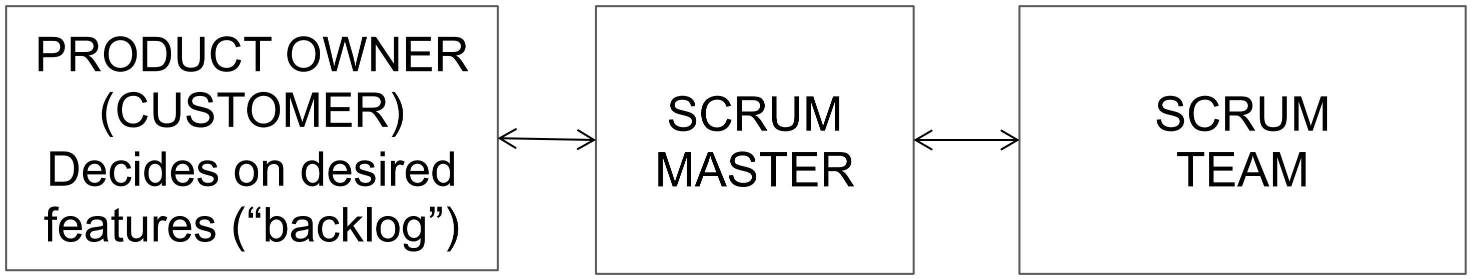
Translated from text that was published in:
http://reaktor.com/blog/moderneja-tietojarjestelmia-ei-voi-hankkia-kuten-koulujen-pulpetteja/

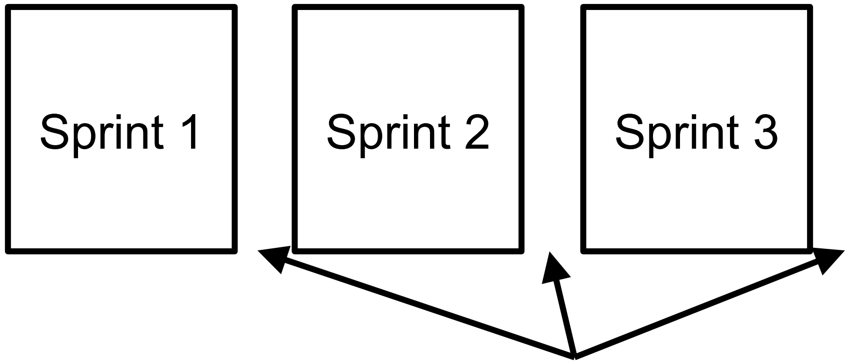
Scrum

Scrum teams are :

Self-organizing

Cross-functional: include all the necessary experts


PRODUCT OWNER (CUSTOMER) Decides on desired features ("backlog")
SCRUM MASTER
SCRUM TEAM

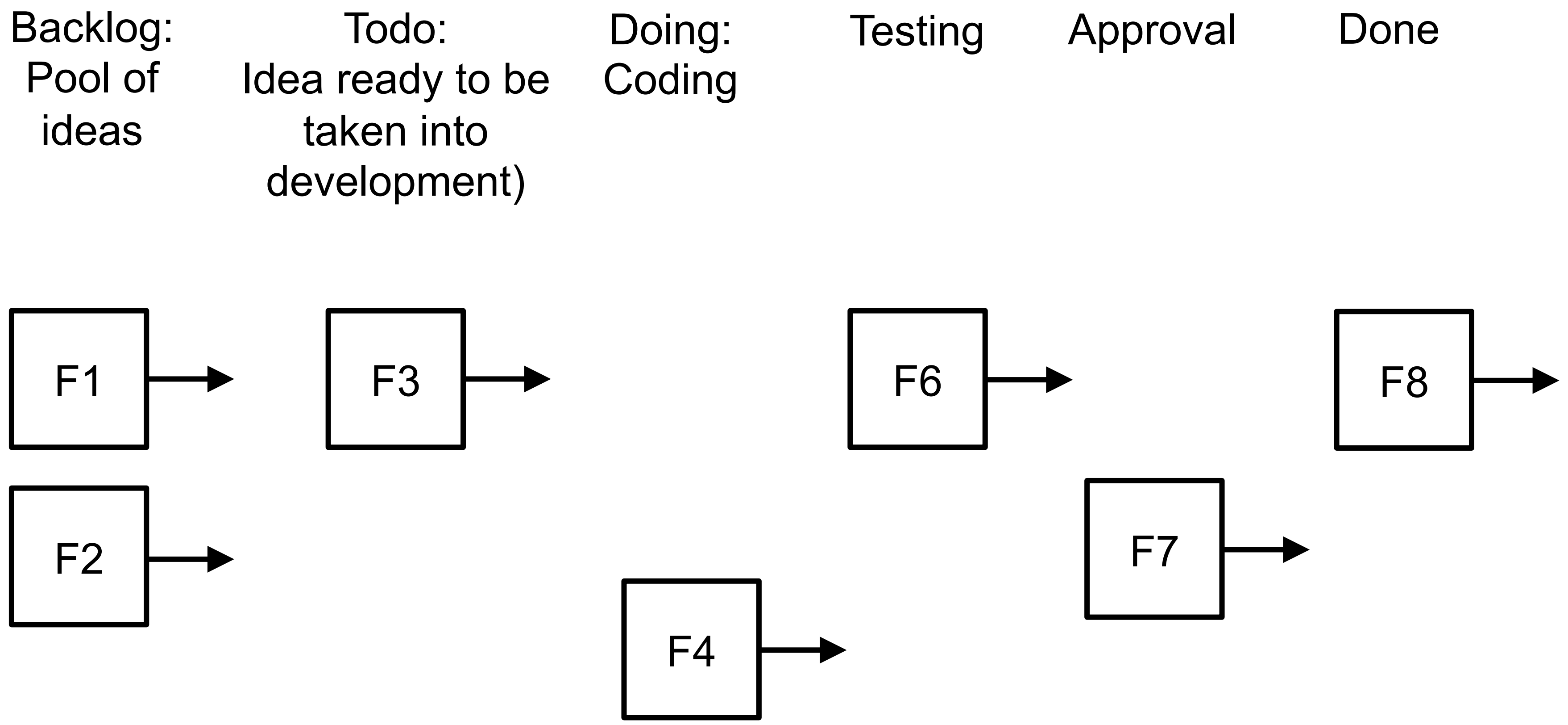


Each sprint is 1–4 weeks

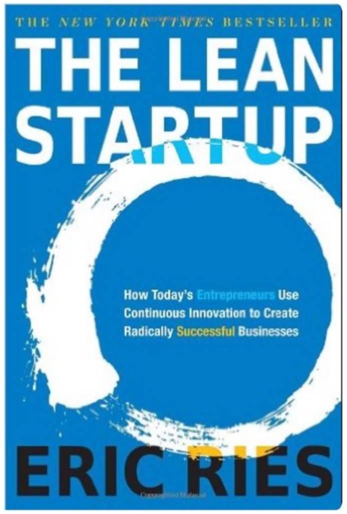
Customer decides what functionality will be added next

Kanban

Instead of sprints, every feature has its own progress status



Lean


THE NEW YORK TIMES BESTSELLER
THE LEAN STARTUP
How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses
ERIC RIES

Is a business model that fits well to the agile approach and startups

Characteristics:

- Favors experimentation over elaborate planning
- Validated learning over unverified assumptions
- Iterative design over up front planning

Key concepts:

- "Learn fast, fail fast"
- **Minimum viable products (MVPs):** the simplest possible product that can be launched to market to accomplish validated learning
- **Pivots:** when MVP shows that your assumption was incorrect, change your business plan

Benefits and problems of agile approach

Benefits:

- Transparency for customer
- Faster discovery of hidden problems
- Planning for late-stage changes
- Earlier shipping / Earlier killing
- Suitability for complex markets

Problems:

- Some organizations are hostile to an idea of lack of upfront planning
- Less emphasis on documentation
- Not suitable for safety-critical systems
- Requires good, committed developers
- Hard to make work on large projects

Spot possible problems in a vendor’s proposal

Company has the following project proposal and seeks offers:

- We need a web shop for customers, and a sales management system for tracking the orders
- Company has no existing system
- Technical feasibility assessment: low tech familiarity in company, low structure (current processes have developed bottom-up)
- Project to be completed in 1 year

The IT services firm proposes:

- Months 1-3: requirements analysis, process modeling
- Months 4-6: logical design of processes and databases
- “Feature freeze” after 6th month
  - Any additional system features to be negotiated separately
- Months 7-8: sales tracking implementation
  - Using IT firm’s own software package
- Months 9-10: web shop and sales tracking interfaces for users
- Months 11-12: sales tracking integration to rest of the system, user testing, installation

Project manager, 3 programmers

Spot possible problems in a vendor's proposal…

A perfect example of waterfall model

Feature freeze + separate billing for additional features
=> no incentive for developing good software

Reuse may be beneficial, but it may also create a lock-in

The IT services firm proposes:

- Months 1-3: requirements analysis, process modeling
- Months 4-6: logical design of processes and databases
- "Feature freeze" after 6th month
  - Any additional system features to be negotiated separately
- Months 7-8: sales tracking implementation
  - Using IT firm's own software package
- Months 9-10: web shop and sales tracking interfaces for users
- Months 11-12: sales tracking integration to rest of the system, user testing, installation

Project manager, 3 programmers

Differences in planning processes

Agile?

… development of a popular menu for a new restaurant?

vs.

Waterfall?

… building a bridge?

Learning objectives

Understand why software development is iterative (in many levels)

Know basics about the main software development process models

- Waterfall
- Iterative waterfall
- Agile