Software Engineering and Architecture
Lecture 1: Introduction, agile methods & scrum

Olivier Liechti
olivier.liechti@heig-vd.ch

MSE | MASTER OF SCIENCE IN ENGINEERING

MSE | MASTER OF SCIENCE IN ENGINEERING

Hello.

LOTARIS

novaccess

AVALIA

MSE | MASTER OF SCIENCE IN ENGINEERING

Today.

Introduction, agile mindset, scrum
15h00 - 16h00

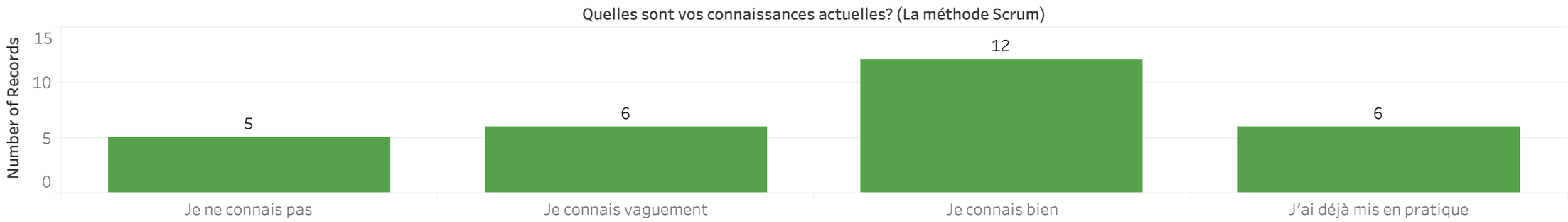
Introduction to Docker
16h15 - 17h25

9 students (25%) ???

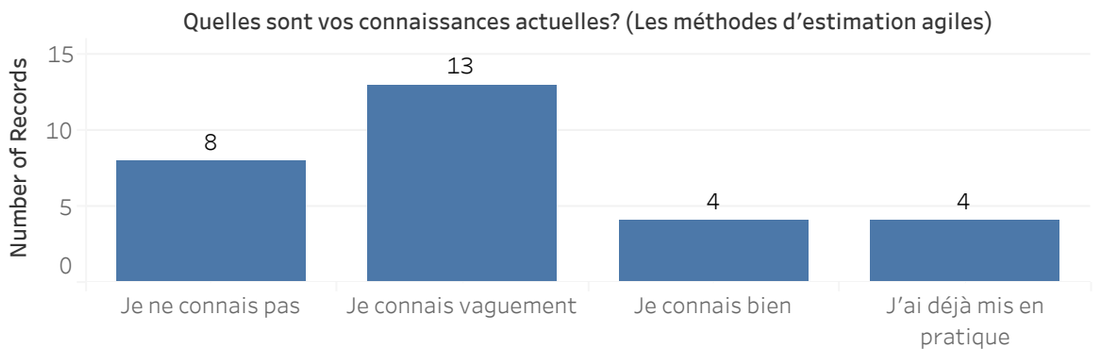
MSE | MASTER OF SCIENCE IN ENGINEERING

47% or 62%???

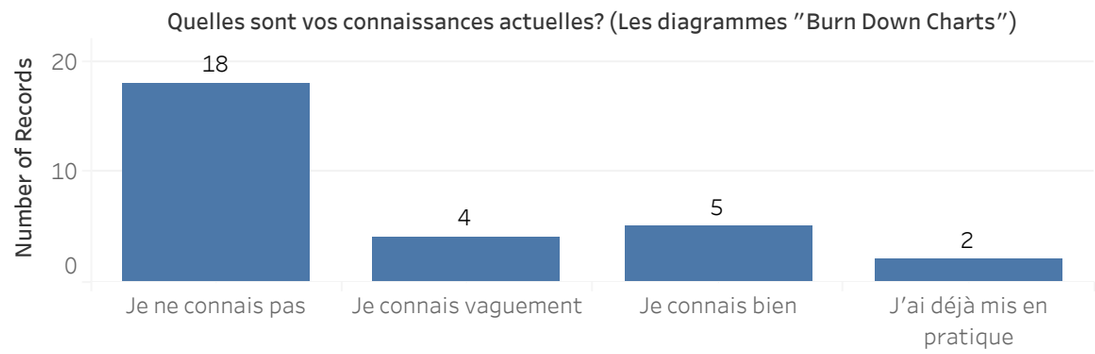
Scrum



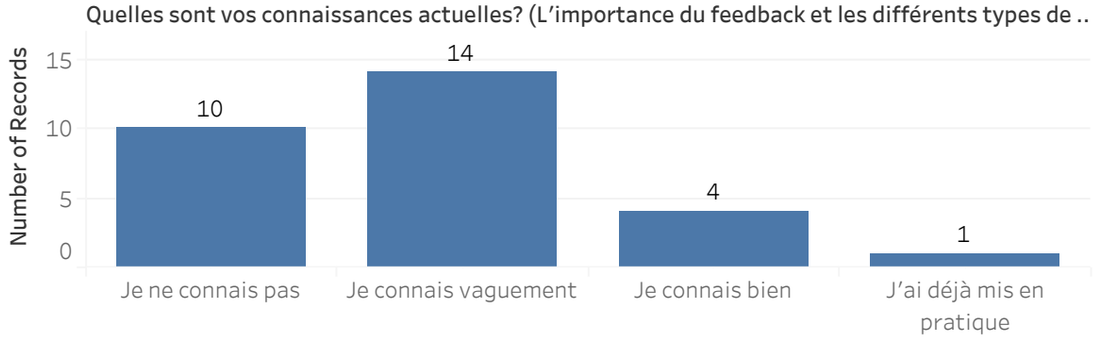
Estimations



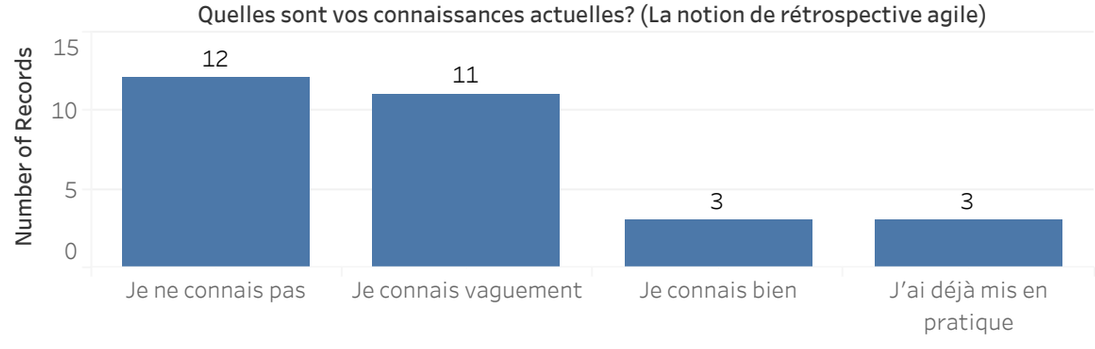
Burn Down



Feedback

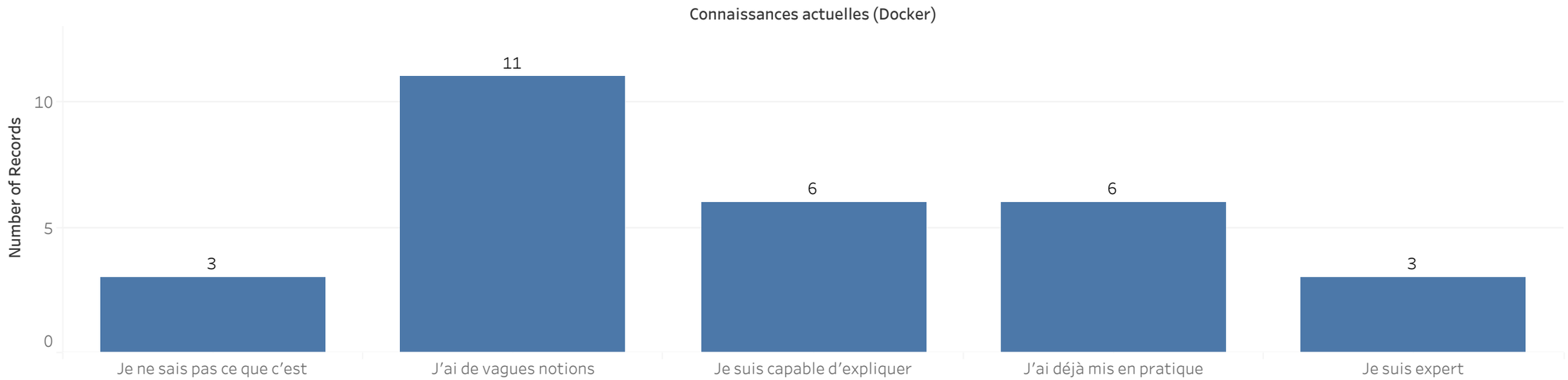


Retrospectives

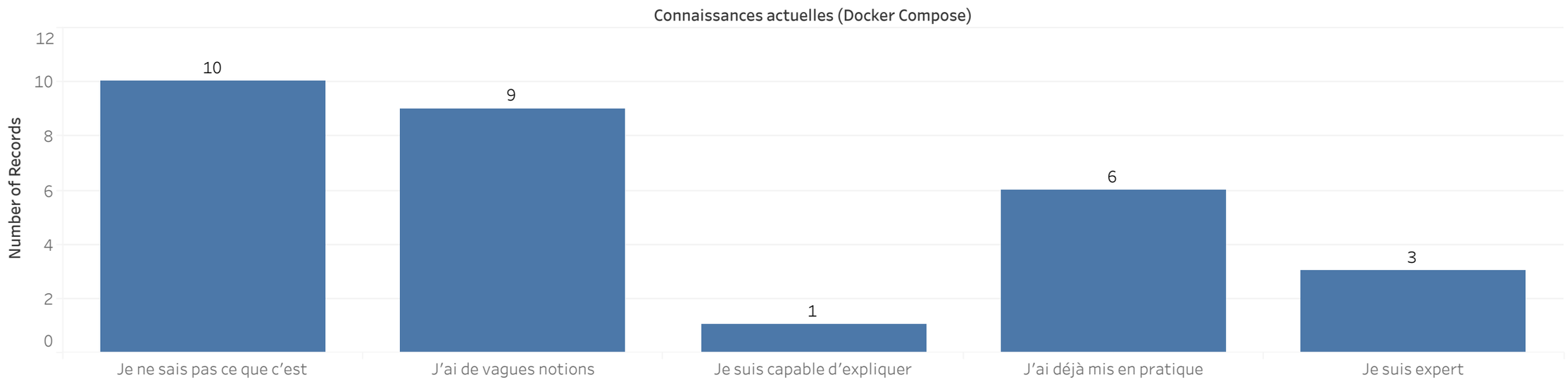


60km/h - 100km/h ???

Docker



Docker Compose



7 responses

Integration et livraison continue, microservices
Je ne comprends pas pourquoi ce cours est obligatoire pour toutes les options.
Connaitre les best practices pour créer un pipeline de livraison continue avec les technos les plus utilisées (Git + Jenkins + Docker + Cloud?)
Je ne connais pas grand chose dans ces domaines car j'ai fait un bachelor en génie électrique et n'ai jamais eu de cours de software engineering avant. Mais ce cours est obligatoire pour moi, et je pense qu'il peut m'être utile pour mon travail.
Les meilleurs pratiques pour la construction d'un logiciel. Comment bien nommer les package et les découper. Comment définir si le code est de la qualité, comment définir des barrières de qualité.
Mise en pratique concrète des différents outils et pratiques
Que ca soit pas trop dur pour les telecom

Course Introduction

MSE | MASTER OF SCIENCE IN ENGINEERING

Part 1 (biberstein)

software architecture (5 weeks)

Part 2 (liechti)

agile methods + software evolution (9 weeks)

Trends

cloud, IoT, scale, “lean”, micro-services, DevOps

Values, methods & techniques

agile mindset, continuous delivery, agile testing

Tools

maven, Jenkins, JUnit, Cucumber, Selenium, docker

Week	Theory	OO Reengineering	Practice
#1	Agile, Scrum		Intro to Docker
#2	Software evolution	Introduction	Specify and implement a micro-service
#3	Continuous delivery (1)	*Setting Directions*	Intro to Jenkins & Travis
#4	Continuous delivery (2)	*First Contact*	Our first build pipeline
#5	Agile testing (1)	*Initial Understanding*	Intro to Cucumber
#6	Agile testing (2)	*Detailed Model Capture*	Add tests to pipeline
#7	Agile metrics	*Tests: Your Life Insurance!*	Add Sonar to pipeline
#8	Continuous improvement	*Migration Strategies I*	Add non-functional tests
#9	Wrap-up	*Migration Strategies II*	

http://scg.unibe.ch/download/oorp/

https://docs.google.com/spreadsheets/d/14ZY6xSNyXjTvdPUKJ3Ked2H0MwxqKeqGrAXlImovMGg/edit?usp=sharing

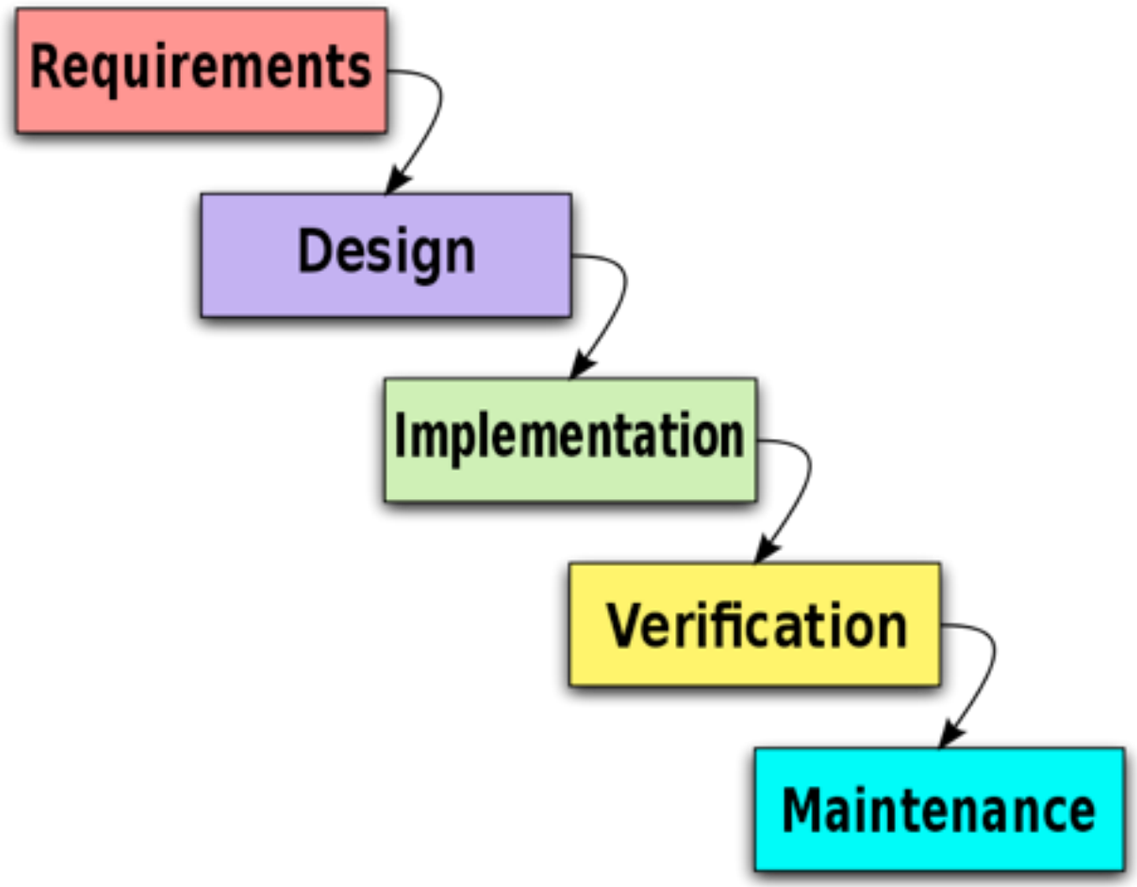
https://docs.google.com/presentation/d/1AMcviruGNDQtEF_fsg9tQ0W-zzHT_zbLBCKSPLsQmTo/edit?usp=sharing

Agile mindset, methods & practices

“Traditional” methodologies: waterfall

http://www.flickr.com/photos/44799719@N00/330191406/sizes/l/#cc_license

months or years



http://en.wikipedia.org/wiki/Waterfall_model

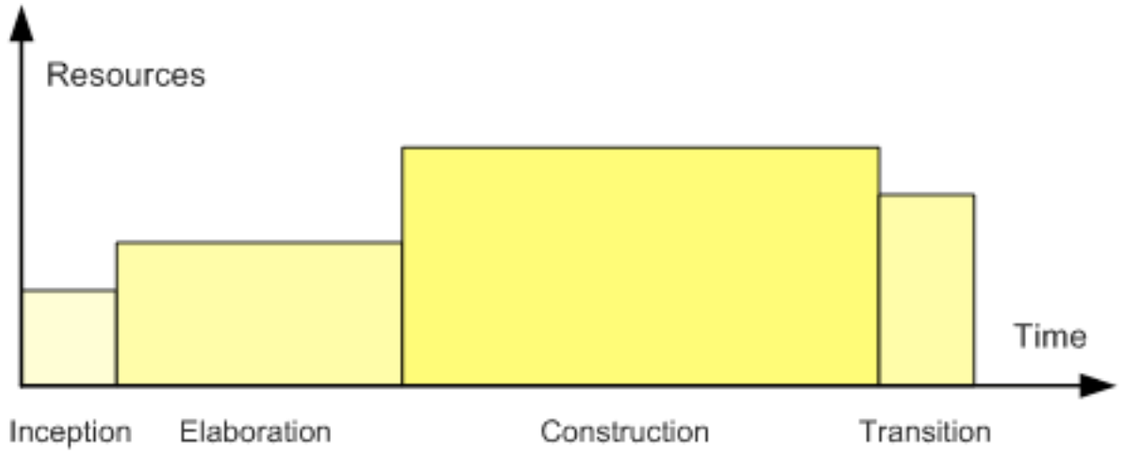
Build software to create value

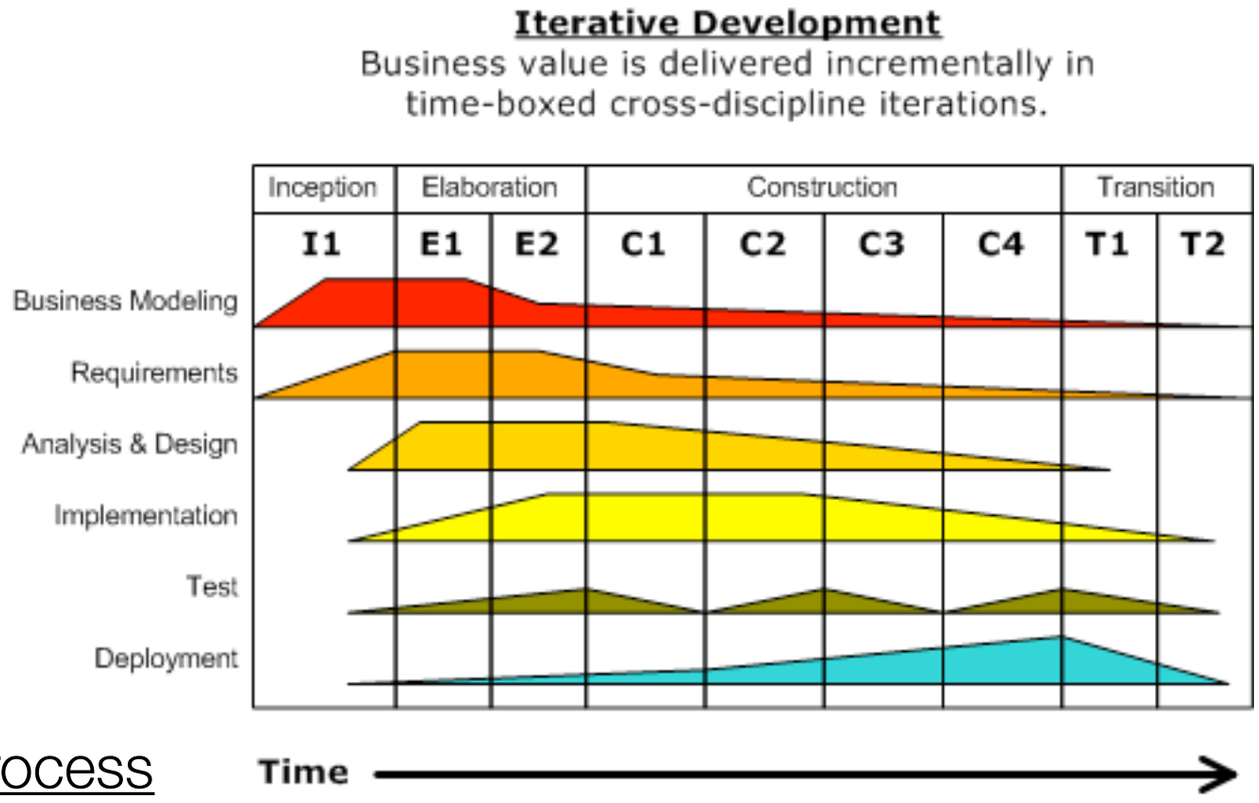
- **Let's assume that:**
  - The customer is able to completely specify requirements at the beginning of the project.
  - We are able to deliver a system that fulfils 100% of these requirements.
- **Question:**
  - Can we be sure that the project is successful?
  - Is there still a risk of failure?

What do you think?

“Traditional” methodologies: Unified Process (~2000)

- **Principles**
  - Iterative and incremental
  - Use case driven
  - Architecture centric
  - Risk focused
- **Issues**
  - Can be “process” and “artifacts” heavy.
  - Is there a “transition” phase?





http://en.wikipedia.org/wiki/Unified_Process

The Agile Manifesto

- February 2001
- “Representatives from
  - Extreme Programming, SCRUM, DSDM,
  - Adaptive Software Development, Crystal
  - Feature-Driven Development, Pragmatic Programming,
  - and others sympathetic to the need for an alternative to documentation driven, heavyweight software development processes”

Set of values and principles shared by the agile community.

The Agile Manifesto

Manifesto for Agile Software 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.

Kent Beck
Mike Beedle
Arie van Bennekum
Alistair Cockburn
Ward Cunningham
Martin Fowler
James Grenning
Jim Highsmith
Andrew Hunt
Ron Jeffries
Jon Kern
Brian Marick
Robert C. Martin
Steve Mellor
Ken Schwaber
Jeff Sutherland
Dave Thomas

The Agile Manifesto

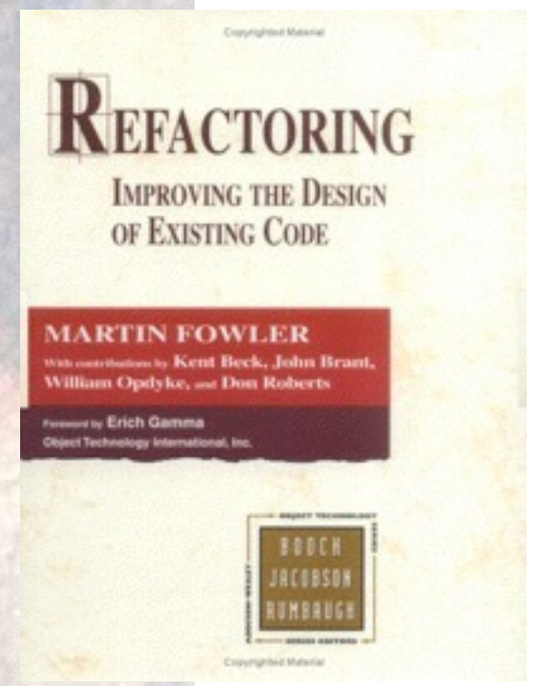
Manifesto for Agile Software Development

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.

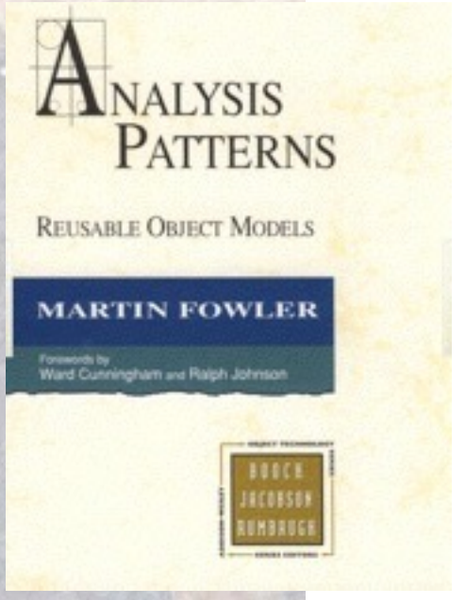
Kent Beck
Mike Beedle
Arie van Bennekum
Alistair Cockburn
Ward Cunningham
Martin Fowler
James Grenning
Jim Highsmith
Andrew Hunt
Ron Jeffries
Jon Kern
Brian Marick
Robert C. Martin
Steve Mellor
Ken Schwaber
Jeff Sutherland
Dave Thomas



JUnit

JUnit

WikiWikiWeb



The Agile Manifesto - Principles (1)

- Our highest priority is to **satisfy the customer** through **early** and **continuous** delivery of **valuable software**.
- **Welcome changing requirements**, even late in development. Agile processes harness change for the customer's **competitive advantage**.
- **Deliver working software frequently**, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

The Agile Manifesto - Principles (2)

- **Business** people and **developers** must **work together daily** throughout the project.
- Build projects around **motivated individuals**. Give them the **environment** and **support** they need, and **trust them to get the job done**.
- The most efficient and effective method of conveying information to and within a development team is **face-to-face conversation**.

The Agile Manifesto - Principles (3)

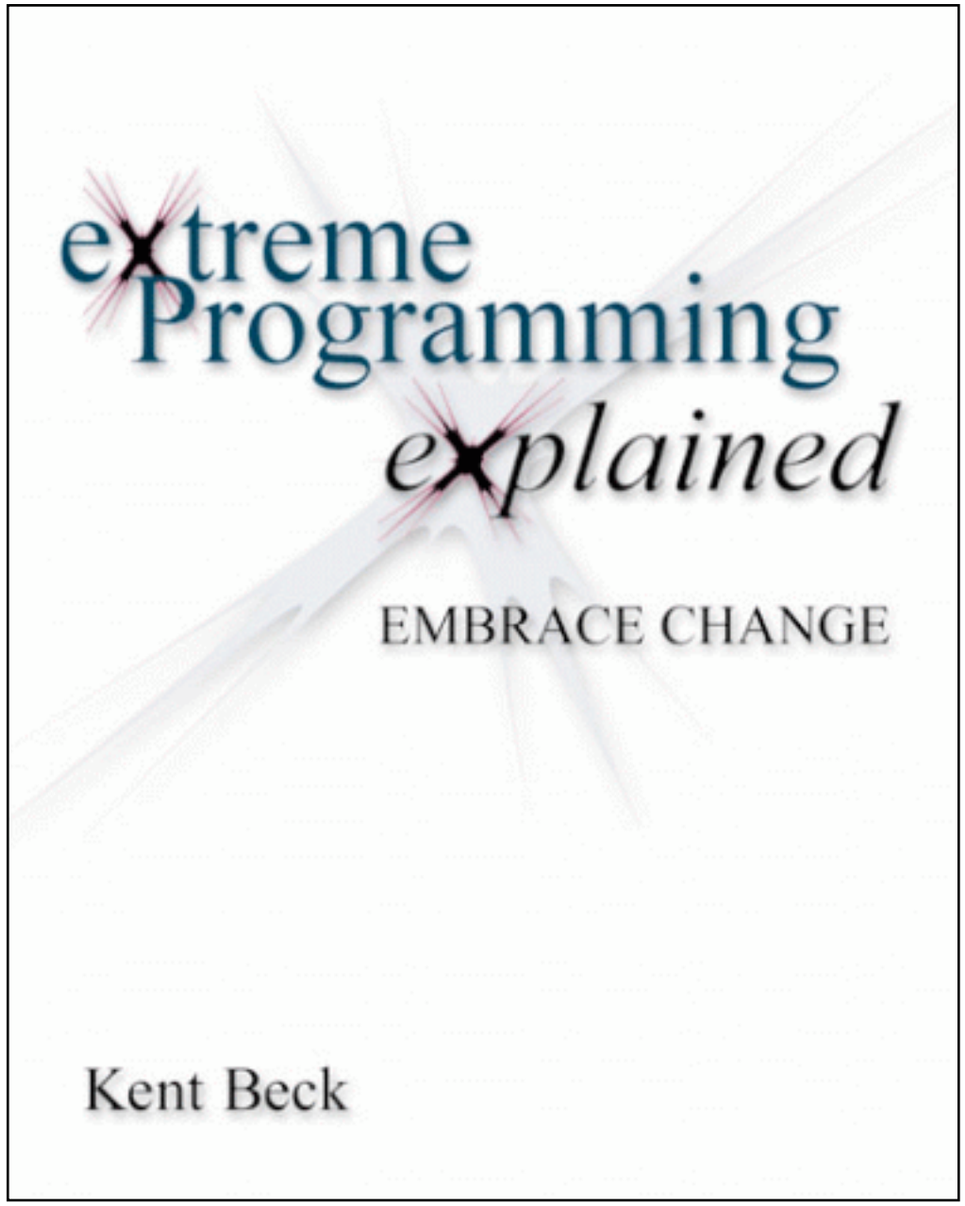
- Working software is the primary **measure of progress**.
- Agile processes promote **sustainable development**. The sponsors, developers, and users should be able to **maintain a constant pace indefinitely**.

The Agile Manifesto - Principles (4)

- **Continuous attention** to **technical excellence** and good design enhances agility.
- **Simplicity** - the art of maximizing the amount of work not done - is essential.
- The best architectures, requirements, and designs emerge from **self-organizing teams**.
- At regular intervals, **the team reflects on how to become more effective**, then **tunes** and **adjusts** its behavior accordingly.

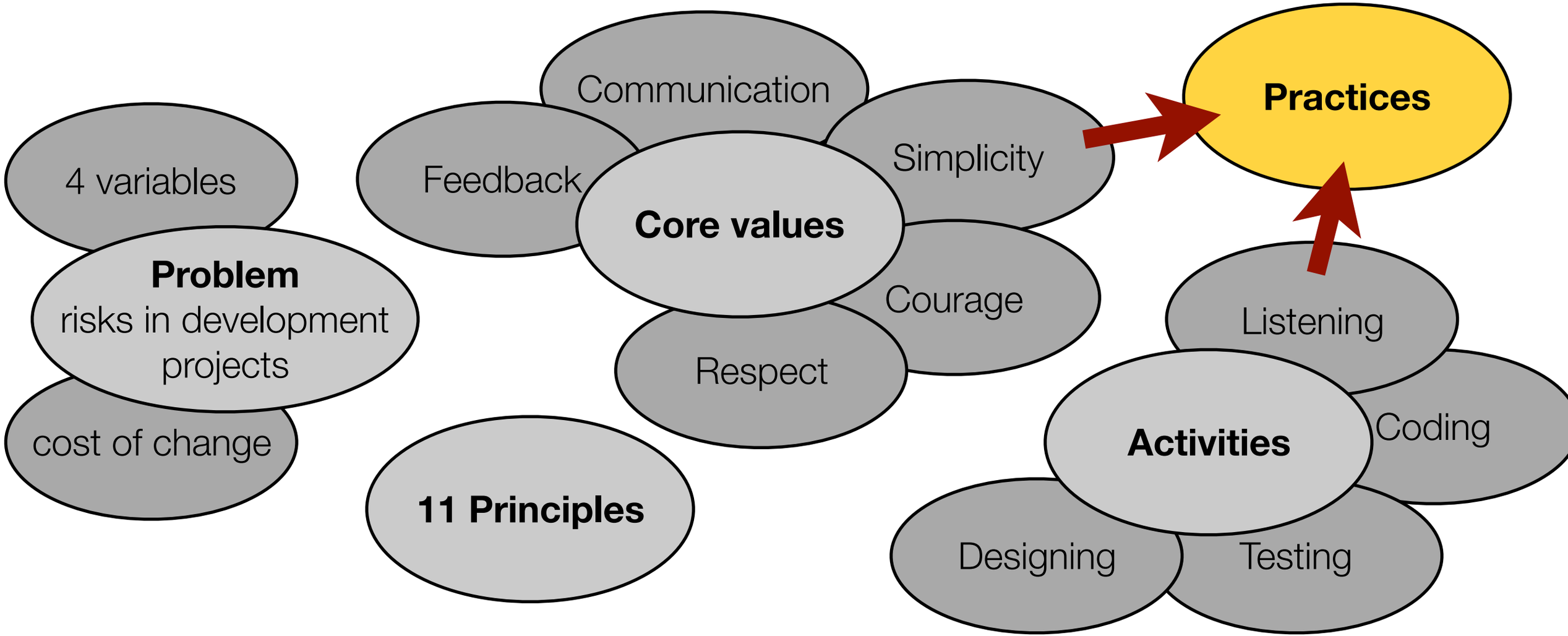
The XP reference book

- Most of the content for these slides comes from the first edition of the eXtreme Programming reference book.



eXtreme Programming (XP)

- Kent Beck
- Chrysler Comprehensive Compensation System (C3), 1996
- “eXtreme Programming explained - embrace change”, published in 1999, 2nd edition in 2004



The Values of XP: **simplicity**

- We will do what is needed and asked for, but no more.
- **This will maximize the value created for the investment made to date.**
- We will take small simple steps to our goal and mitigate failures as they happen.
- We will create something we are proud of and maintain it long term for **reasonable costs**.

http://www.extremeprogramming.org/values.html

The Values of XP: **communication**

- Everyone is **part of the team** and we **communicate face to face daily**.
- We will **work together on everything** from requirements to code.
- We will create the best solution to **our problem** that we can together.

http://www.extremeprogramming.org/values.html

The Values of XP: **feedback**

- We will take every iteration **commitment** seriously by delivering **working software**.
- We **demonstrate our software early and often** then **listen carefully** and make any changes needed.
- We will talk about the project and **adapt our process** to it, not the other way around.

http://www.extremeprogramming.org/values.html

The Values of XP: **respect**

- Everyone gives and feels the respect they deserve as a **valued team member**.
- **Everyone contributes value even if it's simply enthusiasm**.
- Developers **respect the expertise of the customers** and vice versa.
- Management respects our right to **accept responsibility and receive authority** over our own work.

http://www.extremeprogramming.org/values.html

The Values of XP: **courage**

- We will **tell the truth about progress** and estimates.
- We **don't document excuses** for failure because we plan to succeed.
- We don't fear anything because **no one ever works alone**.
- We will **adapt to changes** when ever they happen.

http://www.extremeprogramming.org/values.html

XP First Edition: work practices

- The Planning Game
- Small releases
- Metaphor
- Simple design
- Testing
- Refactoring
- Pair programming
- Collective ownership
- Continuous integration
- 40 hours week
- On-site customer
- Coding standards

XP Second Edition: 13 primary practices

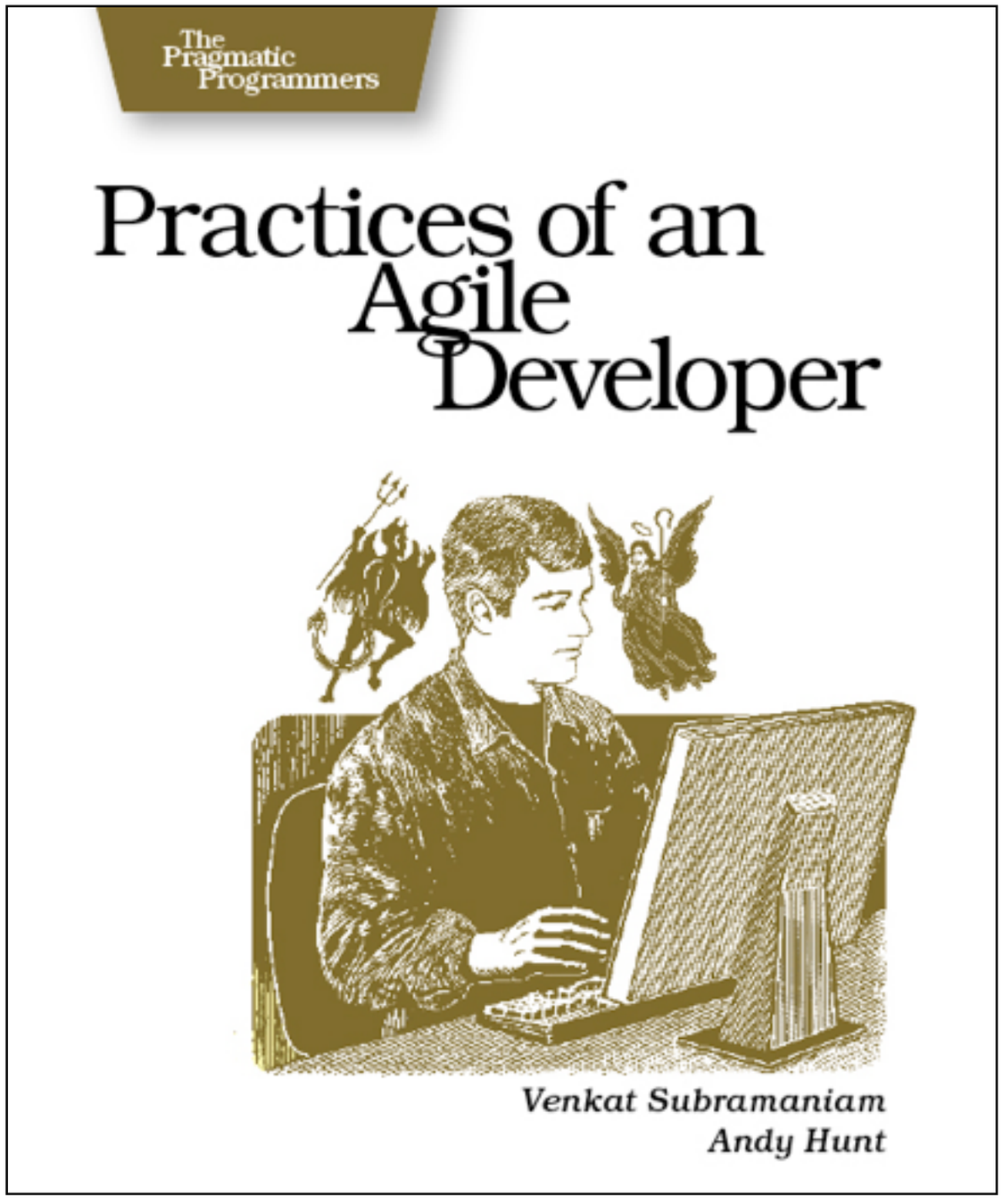
- Sit Together
- Whole Team
- Informative Workspace
- Energized Work
- Pair Programming
- Stories
- Weekly Cycle
- Quarterly Cycle
- Slack
- Ten-Minute Build
- Continuous Integration
- Test-First Programming
- Incremental Design

XP Second Edition: 11 corollary practices

- Real Customer Involvement
- Incremental Deployment
- Team Continuity
- Shrinking Teams
- Root-Cause Analysis
- Shared Code
- Code and Tests
- Single Code Base
- Daily Deployment
- Negotiated Scope Contract
- Pay-Per-Use

Practices of an Agile Developer

- This is a collection of agile work practices, with examples.
- Beginning agility (4)
- Feeding agility (5)
- Delivering what users want (9)
- Agile feedback (6)
- Agile coding (8)
- Agile debugging (5)
- Agile collaboration (8)
- Epilogue: moving to agility (5)



No silver bullet

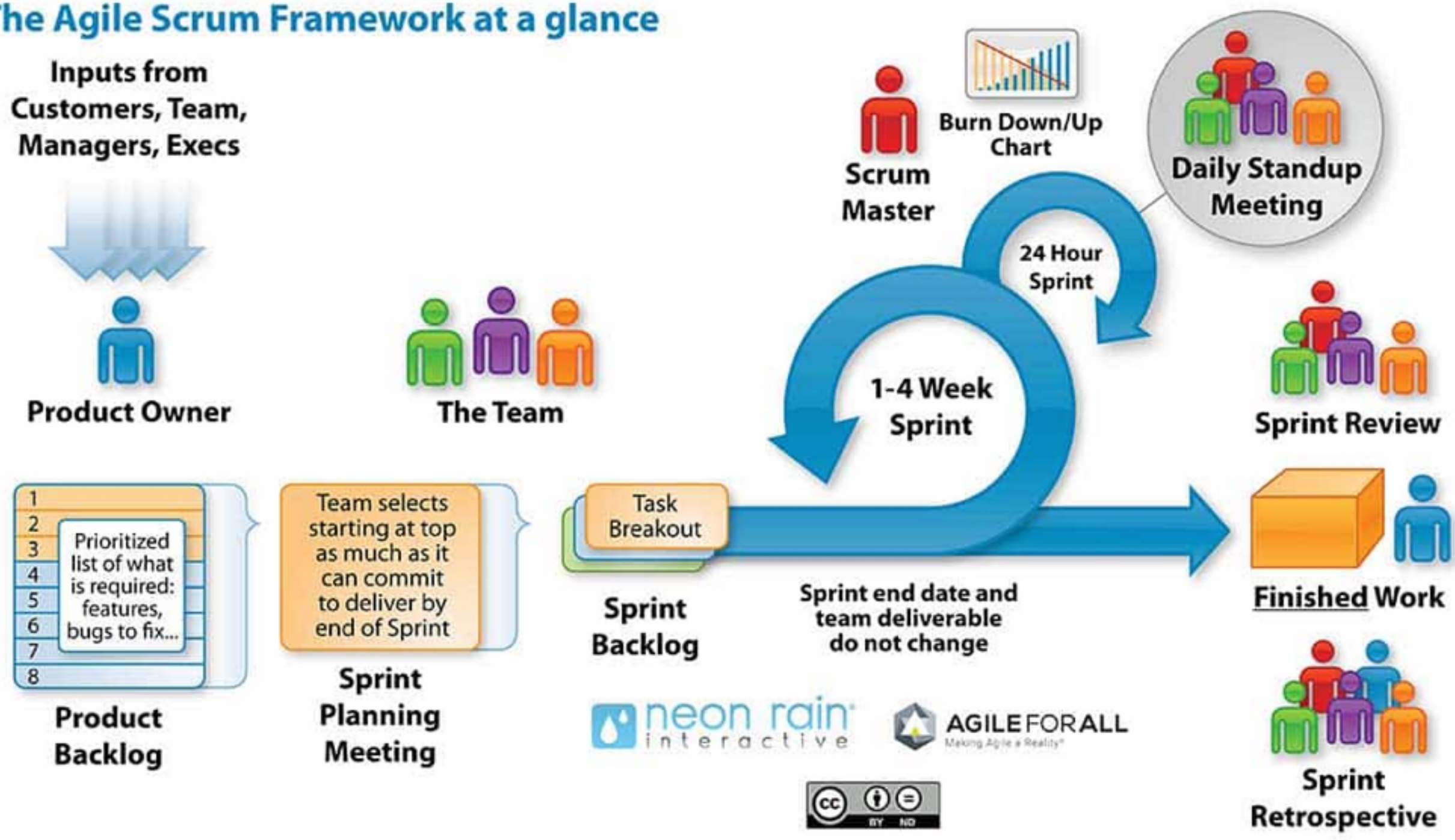
- There is **no “magic” process** that would work exactly the same way for every project, in every environment.
- Agile methodologies and XP describe **core values and key principles** that you need to integrate and **customize** in your particular **context**.
- Agile teams need to **continuously reflect** on their work.
- XP looks like it is **less “formal”** than traditional methodologies. But while there are certainly less roles, less workflows and less artifacts, XP requires **a lot of discipline** to work well.

© Scott Adams, Inc./Dist. by UFS, Inc.

MASTER OF SCIENCE
IN ENGINEERING

Introduction to Scrum

The Agile Scrum Framework at a glance



https://www.neonrain.com/agile-scrum-web-development/

Scrum roles

- **The Product Owner (PO)**
  - Must understand how value can be created
  - Defines requirements and manages priorities
  - Multi-faceted role
- **The Team**
  - Self-organized & responsible
  - Feature teams vs component teams
- **The Scrum Master**
  - Ensures that Scrum rules are respected
  - Removes impediments & protect the team

Scrum cycles

- **The Release cycle (~2-3 months)**
  - The company (e.g. marketing) needs to have a medium / long term vision for the product (the roadmap).
  - We need to have a high-level view of what features will be available when (e.g. “In Q3 2015, we will have payment capabilities in the mobile app”).
- **The Sprint cycle (~2 weeks)**
  - The PO and the Team agree on a precise scope, which **cannot change** during the sprint.
  - At the end of the sprint, value has been added. All features must have been designed, implemented, tested, documented (ideally deployed).
- **The Daily cycle (1 day)**
  - Team Members coordinate themselves to work on tasks, based on their priorities.

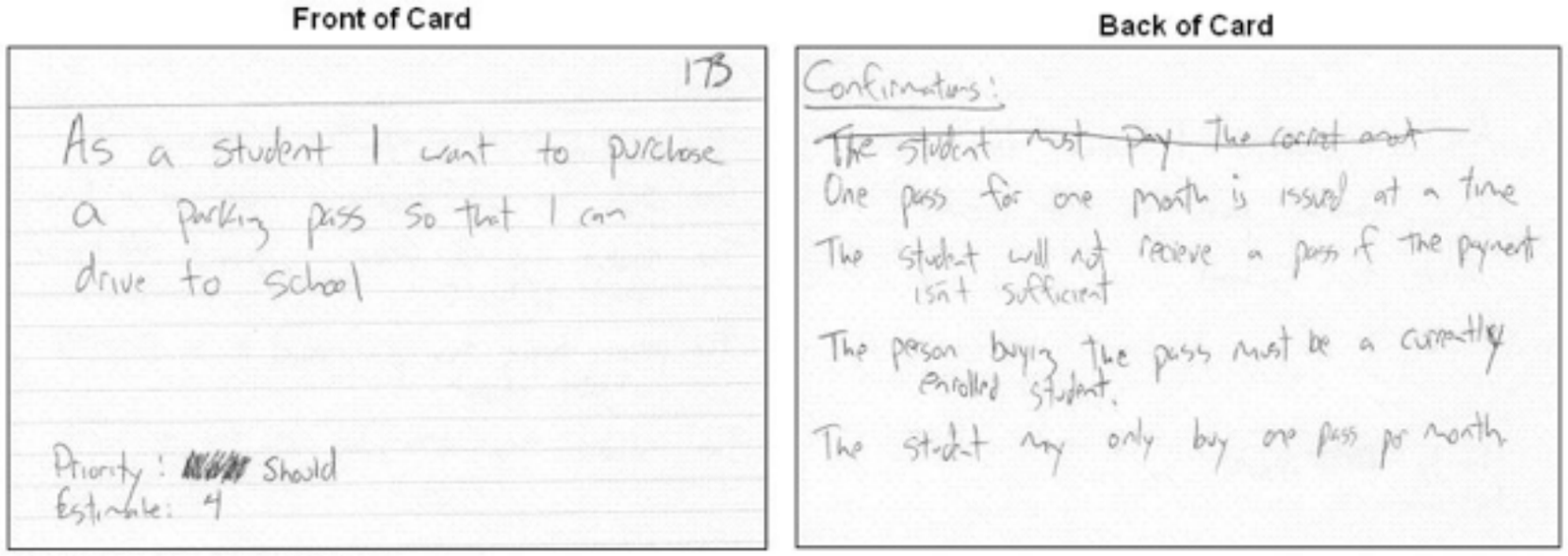
Scrum ceremonies

- **Sprint Planning**
  - Define the “theme” for the sprint
  - Review priorities (top of the product backlog)
  - Estimate what can be achieved during the sprint and make a commitment
- **Daily Scrum Meeting**
  - Review progress and identify issues early (what did I do yesterday, what do I plan to do today, what is blocking me?)
- **Sprint Review**
  - Demo what was achieved during the sprint (review of the software)
- **Sprint Retrospective**
  - Reflection by the team on the evolving work practices (continuous improvement)

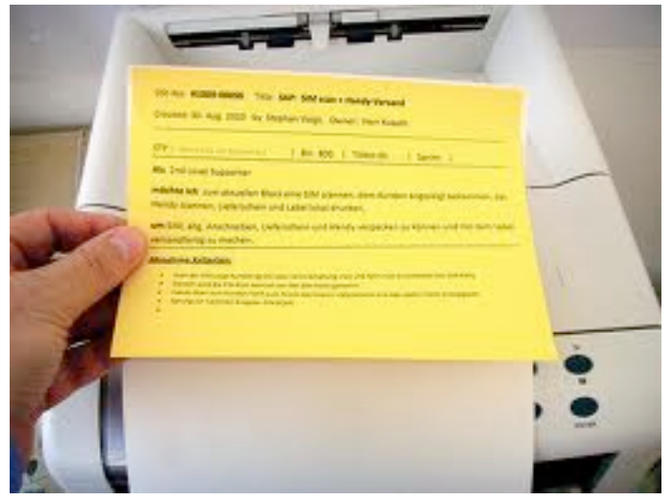
Scrum artifacts

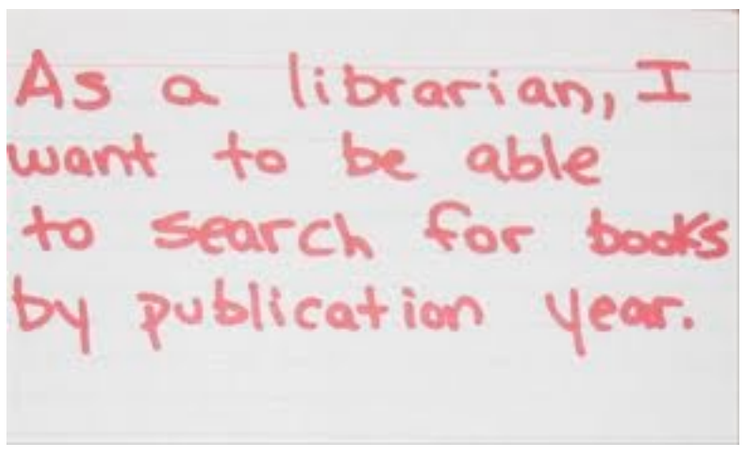
- **User stories**
  - Concise way to capture requirements; goal is to prompt conversations.
- **The Product Backlog**
  - Prioritized list of user stories and epics
- **The Sprint Backlog**
  - Prioritized list of tasks
- **The Burndown Chart**
  - Measures the progress of the team during the sprint
- **The Task Board**
  - Measures the progress of the team during the sprint

User Stories

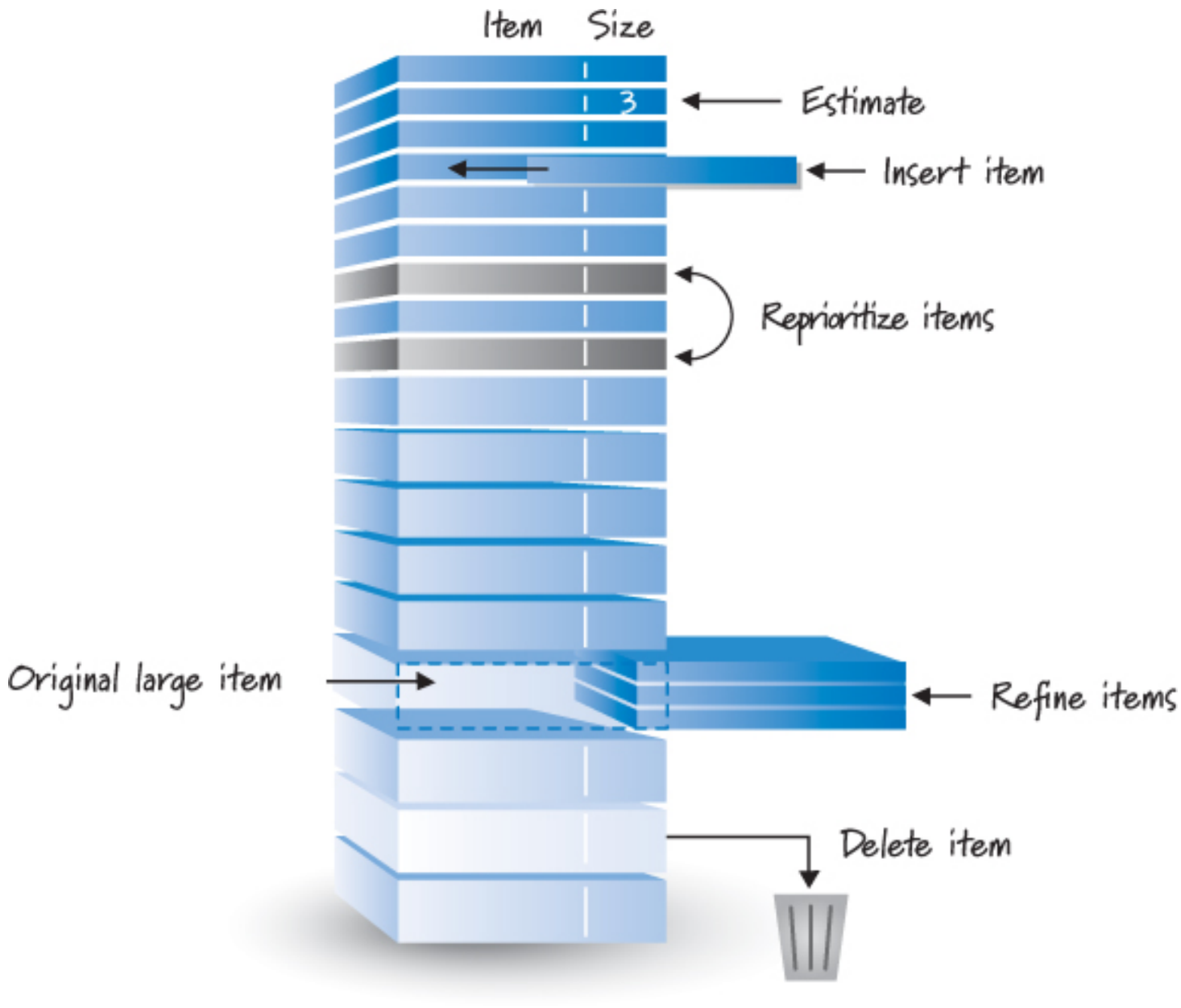


Copyright 2005-2009 Scott W. Ambler





Product Backlog



http://www.informit.com/articles/article.aspx?p=1928232

Sprint Backlog

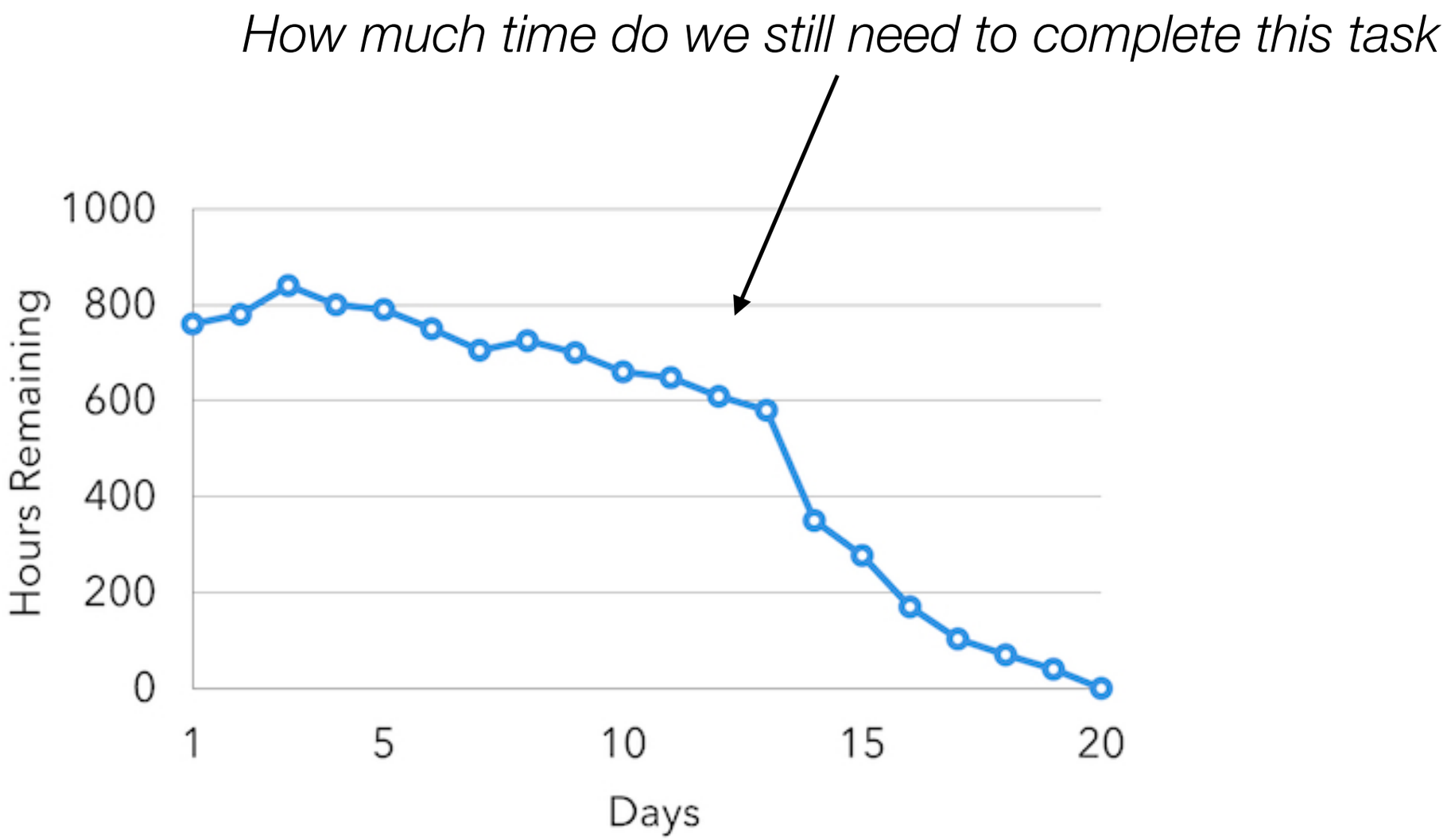
Every story (estimated in relative sizes) is split in small, concrete tasks (estimated in hours)

How much time do we still need to complete this task

User Story	Tasks	Day 1	Day 2	Day 3	Day 4	Day 5	...
As a member, I can read profiles of other members so that I can find someone to date.	Code the ...	8	4	8	0		
	Design the ...	16	12	10	4		
	Meet with Mary about ...	8	16	16	11		
	Design the UI	12	6	0	0		
	Automate tests ...	4	4	1	0		
	Code the other ...	8	8	8	8		
As a member, I can update my billing information.	Update security tests	6	6	4	0		
	Design a solution to ...	12	6	0	0		
	Write test plan	8	8	4	0		
	Automate tests ...	12	12	10	6		
	Code the ...	8	8	8	4		

http://www.mountaingoatsoftware.com/agile/scrum/sprint-backlog

Burndown Chart



http://www.mountaingoatsoftware.com/agile/scrum/sprint-backlog

Task Board

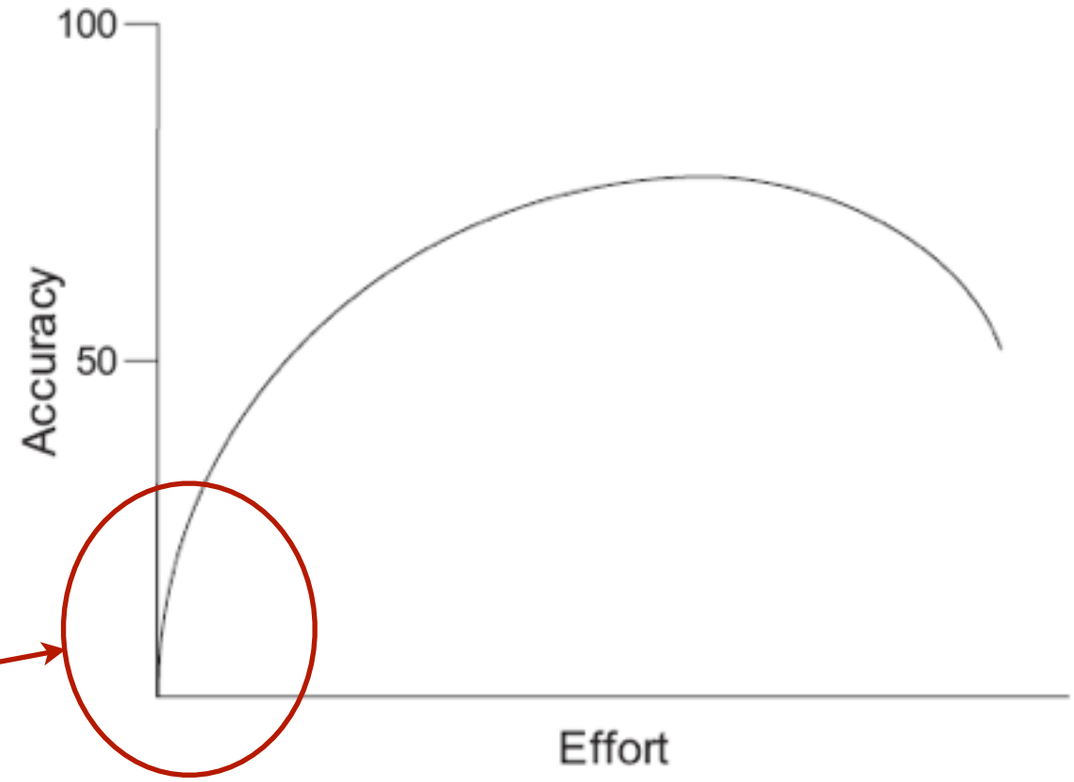

PROJECT/TEAM: Awesome Scrum Team
© Liza Wood
socketsandlightbulbs.com
Backlog
To-Do
In Progress
In Review/QA
Done!
User Story 1
User Story 2
User Story 3
User Story 4
User Story 5
User Story 6
User Story 7

Agile approaches to estimation

http://www.flickr.com/photos/bb_matt/306544780/sizes/m/#cc_license

Techniques for Estimating

- “Prediction is very difficult, especially about the future” — Niels Bohr, Danish physicist.
- The more effort we put in something, the better the result. Right?
- Agile teams choose to be closer to the left of the Effort-Accuracy-Graph [Cohn06]



http://www.mountaingoatsoftware.com/system/hidden_asset/file/15/aep_sample.pdf

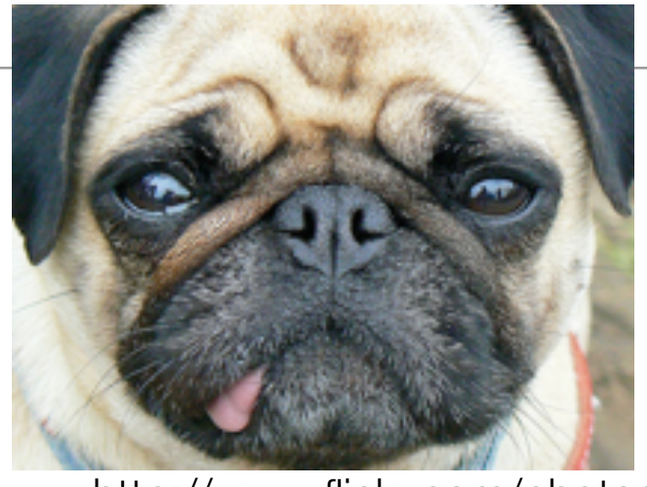
Let’s try to estimate...

- What is the weight of a giraffe?

http://www.flickr.com/photos/badjonni/527455832/sizes/m/#cc_license

Let’s estimate

- **Sort these animals by increasing weight:**
  - a giraffe
  - an elephant
  - a fly
  - a cow
  - a rhinoceros
  - a dog
  - a tyrannosaurus
  - a cat
  - a frog
  - a horse
  - a lion
  - a bull



http://www.flickr.com/photos/e3000/2104850919/sizes/m/#cc_license

http://www.flickr.com/photos/badjonni/527455832/sizes/m/#cc_license

http://www.flickr.com/photos/digitalart/2101765353/sizes/s/#cc_license

Observations

- It is easier to estimate in **relative** than in absolute terms.
- For instance, even if I don't know how much a dog, a giraffe and an elephant weight, I may "guesstimate" that an elephant is about twice as heavy as a giraffe, and that a giraffe is as heavy as 30 dogs.
- The more animals I consider, the more correct the relative weights are likely to be.
- If I know that a dog weights "5 points", then I can infer that a giraffe is about "150 points" and an elephant is about "300 points".
- If I then measure the absolute weight of one (or ideally several) animals, I can get a pretty good idea of the absolute weight of all animals.
- In agile terms, translating relative values into absolute values is based on the notion of velocity.

http://www.flickr.com/photos/e3000/2104850919/sizes/m/#cc_license

http://www.flickr.com/photos/digitalart/2101765353/sizes/s/#cc_license

Techniques for estimating

- **Absolute** vs. **relative** estimation
- Selecting a **unit** (story points, ideal days, etc.)
- Selecting a **scale**
- **Collecting metrics over time** - computing the velocity of the Team

Estimating with “story points”

- **Key ideas:**
  - estimating relative size is easier than estimating absolute size;
  - story points are not a direct measure of time (vs. hours or days);
  - story points measure the relative time to develop a set of features.
- **Story points are not a direct measure of time:**
  - If we estimate that a particular user story has a weight of “10”, we mean that it will require twice the effort of another story that has a weight of “5”.
  - Iteration after iteration, we look at how many “story points” can be delivered in one iteration.
  - This gives us the velocity of the Team.
  - We can use that value to estimate when we will be finished with a release (or whether we should drop some features from the release).

Estimating with “ideal days”

- Key ideas:
  - in a typical day, how much time can a Team Member really spend on the project-related tasks (vs. coordination, administrative tasks, etc.)?
  - How do you take this overhead time in your estimates?
    - An ideal day represents the number of hours that could theoretically be devoted to the project, if there was nothing else to do.
- How does it work:
  - Estimates are done in ideal days
  - Collection of metrics over time tell us how many ideal days fit within one week (elapsed time)
  - We can thus determine how many iterations are needed to complete a given set of features.

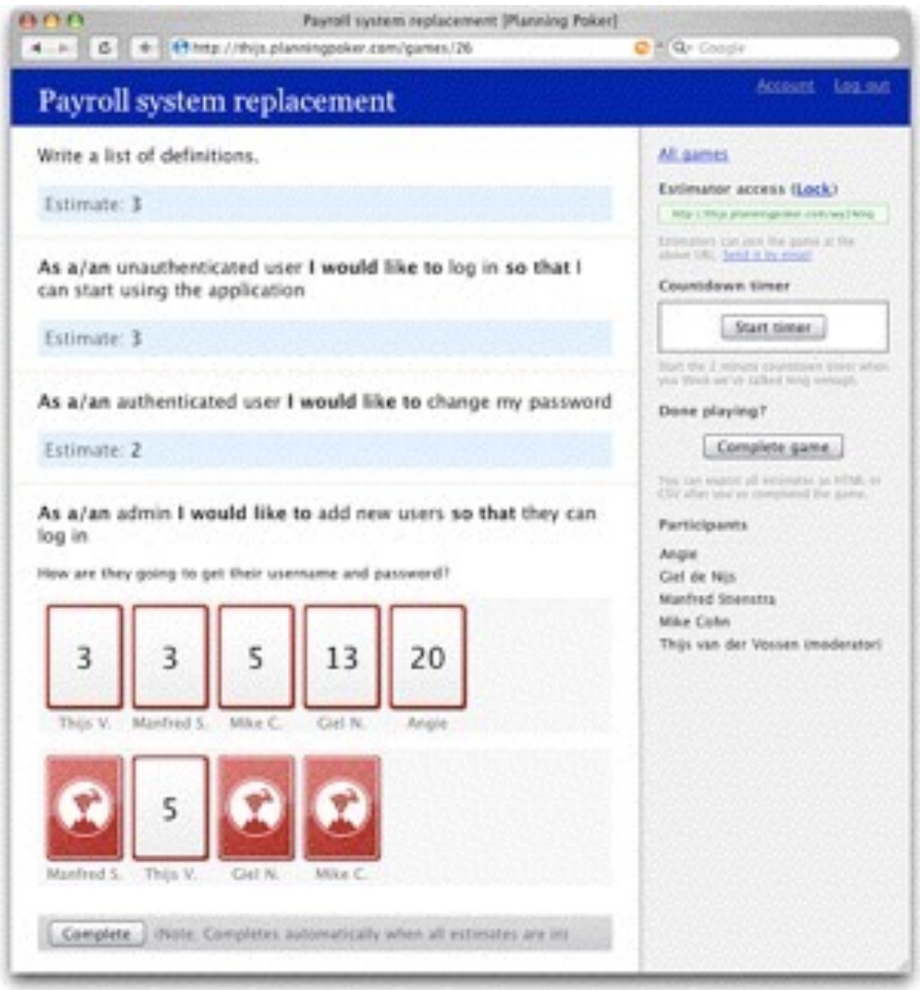
The Planning Poker

- **An iterative approach to estimating**
  - Each estimator is given a deck of cards, each card has a valid estimate written on it (number of story points, ideal days, etc.).
  - The customer/product owner reads a story, which is then briefly discussed.
  - Each estimator selects a card with his or her estimate for the “size” of the story.
  - All participants turn their cards at the same time, so that everyone sees all estimations.
  - The team discusses the differences, which is useful to identify things that were not obvious and to remove ambiguities.
  - People redo estimations until they converge.

The Planning Poker

- Planning Poker combines expert opinion, analogy, and disaggregation (Grenning J. 2002. Planning Poker, www.objectmentor.com/resources/articles/PlanningPoker.zip)

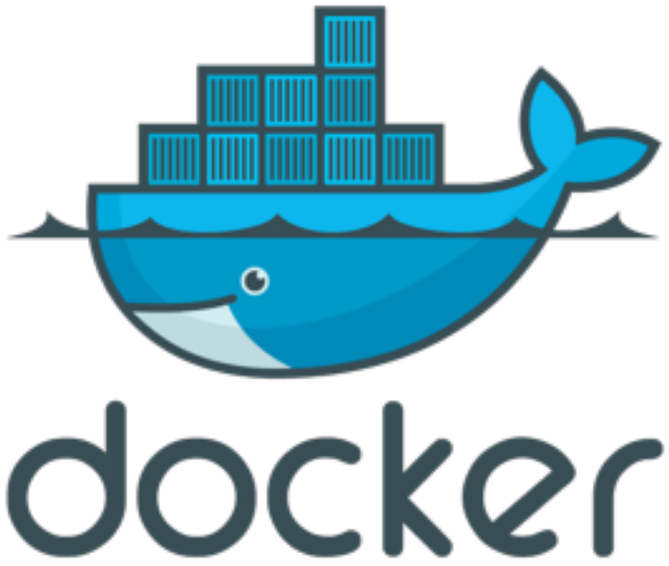
- Online Planning Poker
  www.planningpoker.com

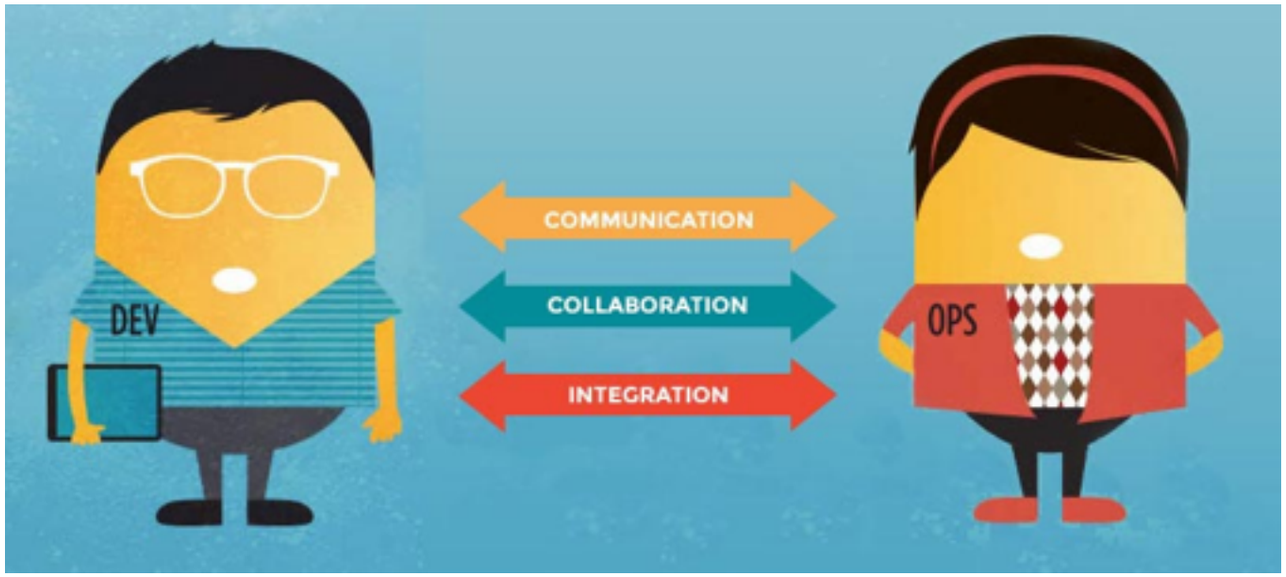


Introduction to Docker

heig-vd

Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

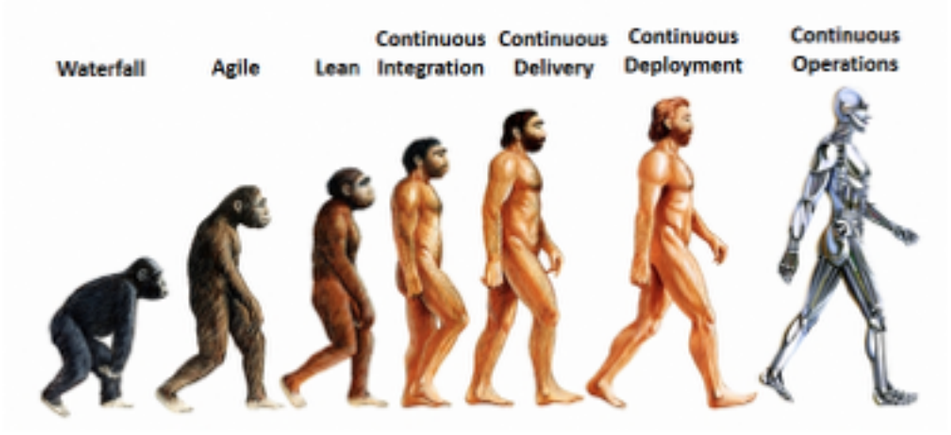


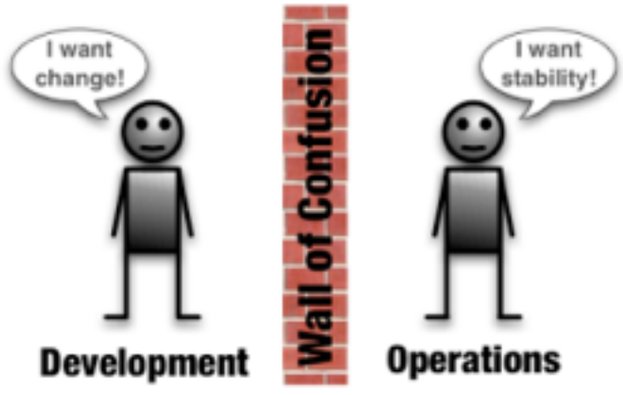


heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

DevOps

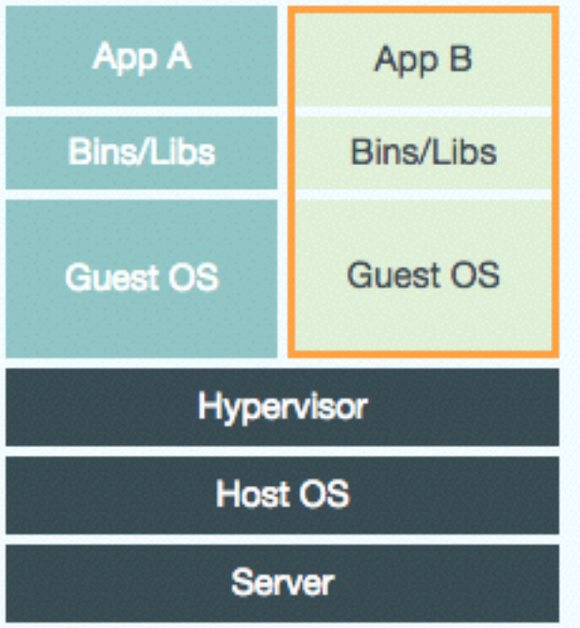
DevOps Movement





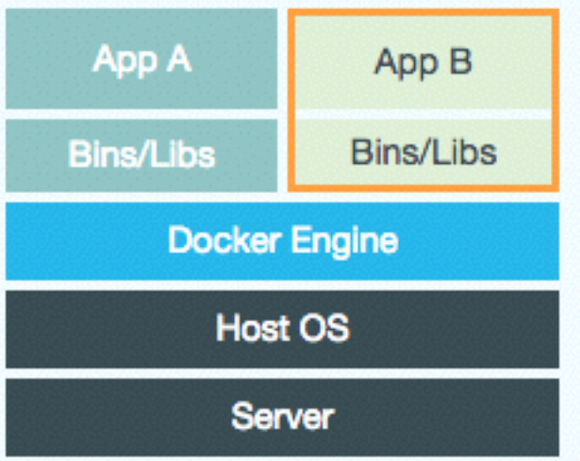
heig-vd

d'Ingénierie et de Gestion
e Vaud



Virtual Machines

Each virtualized application includes not only the application - which may be only 10s of MB - and the necessary binaries and libraries, but also an entire guest operating system - which may weigh 10s of GB.



Docker

The Docker Engine container comprises just the application and its dependencies. It runs as an isolated process in userspace on the host operating system, sharing the kernel with other containers. Thus, it enjoys the resource isolation and allocation benefits of VMs but is much more portable and efficient.

https://www.docker.com/whatisdocker/

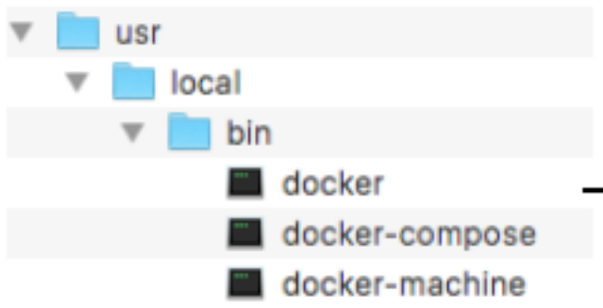
Installing Docker

heig-vd
HAUTE ÉCOLE D'INGÉNIERIE ET DE GESTION DU CANTON DE VAUD
www.heig-vd.ch

- Docker is based on the **Linux Containers** (LXC) technology. If you run a linux distribution on your laptop, you will use Docker natively. Just be careful with the version you are using.
- If you run **Windows** or **Mac OS X**, you have different options. Behind the scenes, you will need a **Linux VM**. The tools that you will install will hide this VM (more or less), but it will still be here.
- On the Docker website, you will be encouraged to install **Docker for Windows** or **Docker for Mac**. This might not be the best choice, and you might want to go for the (legacy) **Docker Toolbox** and **Docker Machine** toolset.
- If you run Windows, be aware that you cannot run **Docker for Windows** and **VirtualBox** at the same time. You will need to enable/disable **Hyper V** and reboot to switch between environments. Docker Toolbox does not have this issue.
- Personally, I still use **Docker Machine** and not Docker for Mac. Reasons: performance (file system), ability to have different environments, the Linux VM is less hidden (has its own IP address). In my demos / webcasts, you will see 192.168.99.100 and not localhost.

heig-vd
HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

Docker Toolbox installation




```
DOCKER_HOST=tcp://xxx.xxx.xxx.xxx:2376
DOCKER_MACHINE_NAME=default
DOCKER_TLS_VERIFY=1
DOCKER_CERT_PATH=$HOME/.docker/machine/machines/default
```

manages

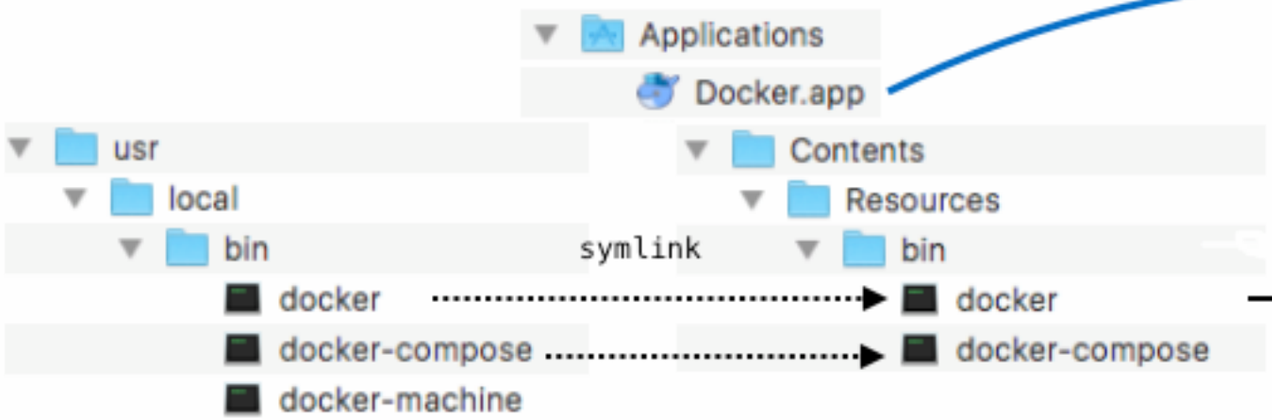
default

IP xxx.xxx.xxx.xxx

HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD

Docker for Mac installation

manages




```
unset ${!DOCKER_*}
```

```
/var/run/docker.sock
```

docker.local

HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

Docker image

Docker container

Docker containers (1)

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

- You can think of a container as a **lightweight** virtual machine. Each container is isolated from the others and has its own IP address.
- Each container is created from a docker image (one could say that a container is a **running instance of an image**).
- There are **commands** to start, list, stop and delete containers.

```
# Start a container (more on this later)
$ docker run

# List running containers
$ docker ps

# List all containers
$ docker ps -a

# Delete a container
$ docker rm

# Display logs produced by a container
$ docker logs
```

Docker containers (2)

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

- With Docker, the philosophy is to have **one application-level service per container** (it is not a strict rule).
- With Docker, the philosophy is also to **run several (many) containers on the same machine**.
- If you think of a typical **Web Application infrastructure**, you would have one or more containers for the apache web server, one container for the database, one container for the reverse proxy, etc.
- With Docker, containers tend also to be **short-lived**. Each container has an **entry point**, i.e. a command that is executed at startup. When this command returns, the container is stopped (and will typically be removed).
- **If a container dies, it should not be a big deal**. Instead of trying to fix it, one will create a new one (from the same image).

Docker images (1)

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

- **A Docker image is a template**, which is used to create containers.
- Every image is **built from a base image** and adds its own configuration and content.
- With Vagrant, we use a file named Vagrantfile to configure and provision a Vagrant box. With Docker, we use a file name **Dockerfile** to create an image. The file contains statements (FROM, RUN, COPY, ADD, EXPOSE, CMD, VOLUME, etc.)
- Just like the community is sharing Vagrant boxes, **the community is sharing Docker images**. This happens on the Docker Hub registry (https://registry.hub.docker.com/).

Docker images (2)

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

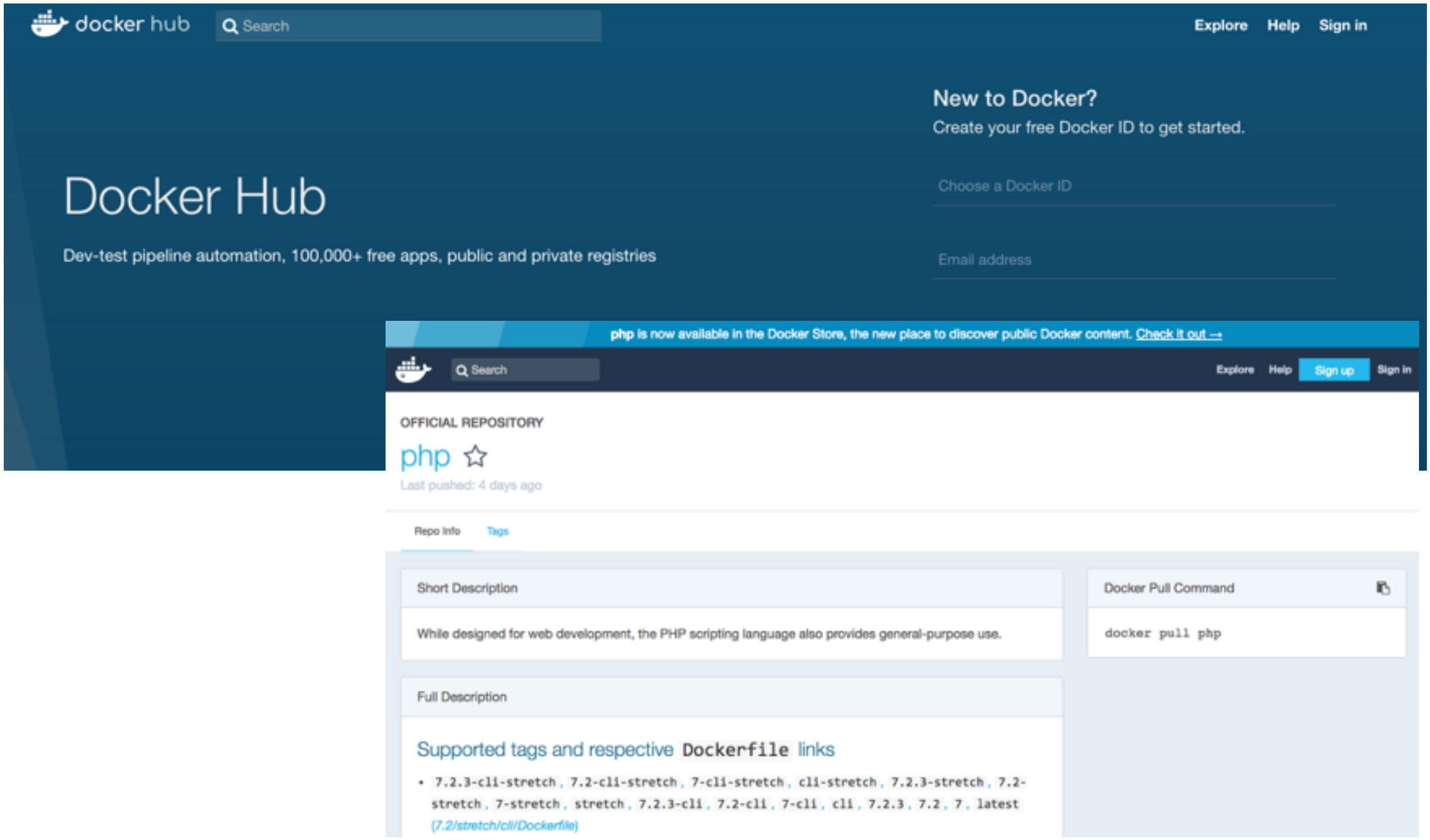
- Here is an example for a Dockerfile (used for first experiments, does not

```
# This image is based on another image

FROM node:carbon

# For information: who maintains this Dockerfile?

MAINTAINER Olivier Liechti

# When we create the image, we copy files from the host into
# the image file system. This is NOT a shared folder!

COPY file_system /opt/res/

# With RUN, we can execute commands when we create the image. Here,
# we install the PM2 process manager

RUN npm install -g pm2@0.12.9
```

```
# Create an image from this Dockerfile
$ docker build -t heigvd/res-demo .

# Execute /bin/bash in a new container, created from the image
$ docker run -i -t heigvd/res-demo /bin/bash
```

Docker Hub

HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch



heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Host (your laptop running Windows or Mac OS X) 10.192.116.213

Linux VM (e.g. 192.168.99.100

docker0
bridge
172.17.0.1

172.17.0.6
/opt/res
Docker
container

172.17.0.2
/opt/res
Docker
container

172.17.0.4
/opt/res
Docker
container

172.17.0.3
/opt/res
Docker
container

Sniffing UDP traffic

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Using tcpdump on the Docker host

```
$ sudo tcpdump -i docker0 udp -A
```

```
E..J..@...E.........q&..6.e{"timestamp":1427800010160,"temperature":null}
11:06:50.362733 IP 172.17.0.8.58225 > 239.255.22.5.9907: UDP, length 46
E..J..@...E.........q&..6.e{"timestamp":1427800010362,"temperature":null}
11:06:50.565816 IP 172.17.0.8.58225 > 239.255.22.5.9907: UDP, length 46
E..J..@...E.........q&..6.e{"timestamp":1427800010565,"temperature":null}
11:06:50.768966 IP 172.17.0.8.58225 > 239.255.22.5.9907: UDP, length 46
E..J..@...E.........q&..6.e{"timestamp":1427800010768,"temperature":null}
11:06:50.970691 IP 172.17.0.8.58225 > 239.255.22.5.9907: UDP, length 46
E..J..@...E.........q&..6.e{"timestamp":1427800010970,"temperature":null}
11:06:51.172537 IP 172.17.0.8.58225 > 239.255.22.5.9907: UDP, length 46
E..J..@...E.........q&..6.e{"timestamp":1427800011172,"temperature":null}
11:06:51.374546 IP 172.17.0.8.58225 > 239.255.22.5.9907: UDP, length 46
E..J..@...E.........q&..6.e{"timestamp":1427800011374,"temperature":null}
11:06:51.578663 IP 172.17.0.8.58225 > 239.255.22.5.9907: UDP, length 46
E..J..@...E.........q&..6.e{"timestamp":1427800011578,"temperature":null}
```

Demo 1

HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

https://github.com/SoftEng-HEIGVD/Teaching-Docker-SimpleJavaServer

Demo 2

HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

https://github.com/SoftEng-HEIGVD/Teaching-Docker-UDP-sensors