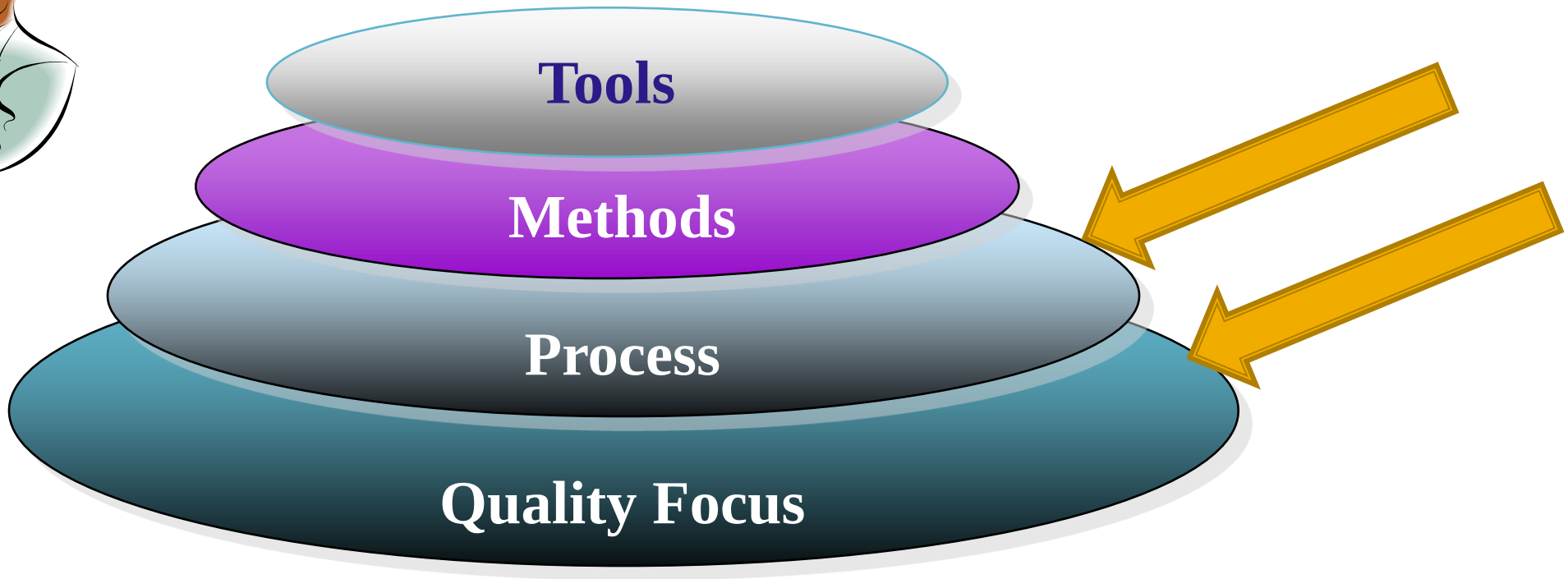
Software Process

Jianyong Wang(王建勇)
Department of Computer Science and Technology
Tsinghua University, Beijing, China

A Layered Technology



Outline

- Evolution of Software Process Model
- Agile Methods
- Tools for Agility

History of Software Process Model

Alistair Cockburn
One of the initiators of the agile movement

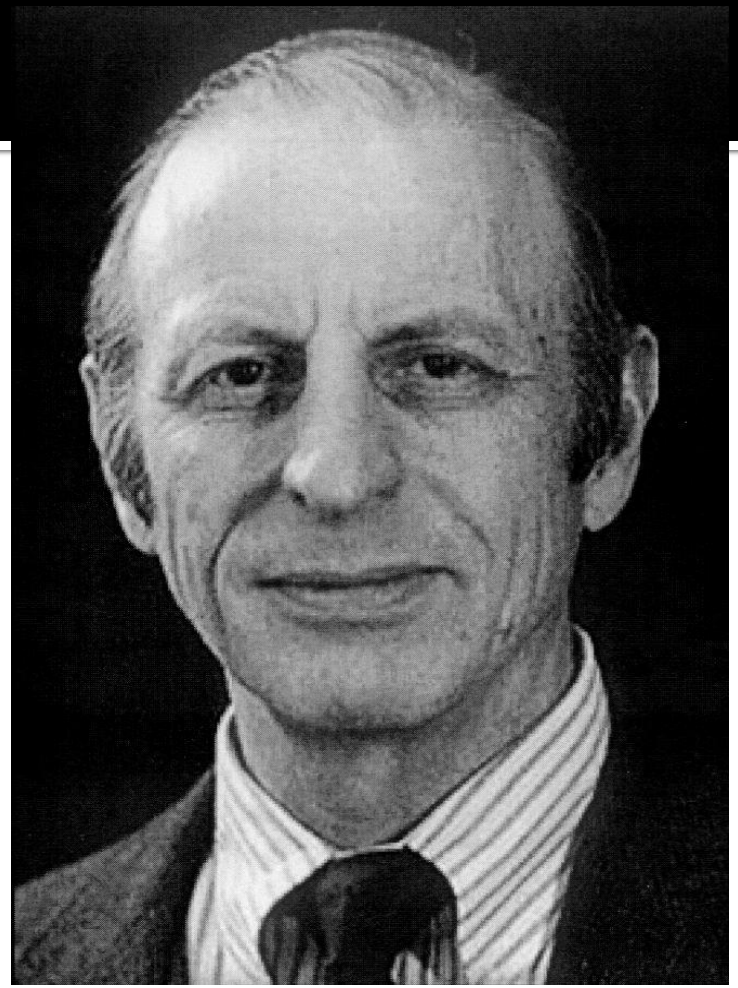
Software Development is Resource-limited cooperative game of invention and communication

Doing Engineering is Creating a trade-off solution in the face of conflicting demands.

The Process Premise

"The quality of a software system is governed by the quality of the process used to develop and evolve it."

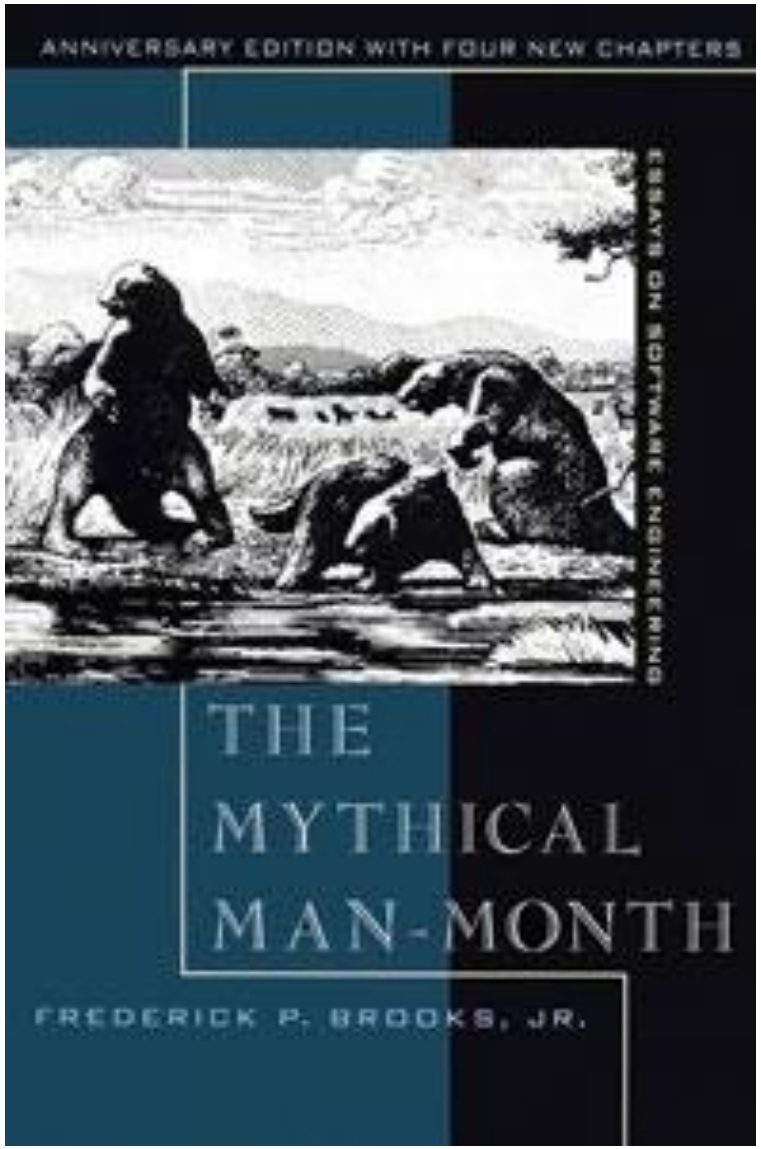
-- Watts Humphrey



"Father of software quality"

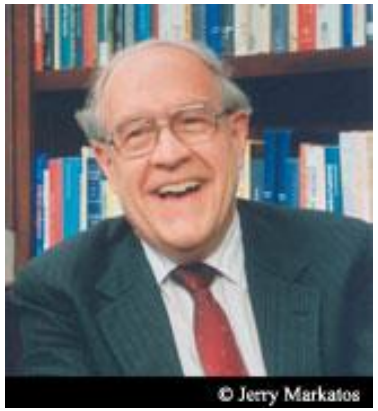
Founder of Software Engineering Institute at CMU , which developed Capability Maturity Model

When software development process was ad hoc…



“软件人员太像皇帝新衣故事中的裁缝了。当我来检查软件开发工作时，所得到的回答好象对我说：我们正忙于编织这件带有魔法的织物。只要等一会儿，你就会看到这件织物是极其美丽的。

但是我什么也看不到，什么也摸不到，也说不出任何一个有关的数字，没有任何办法得到一些信息说明事情确实进行得非常顺利，而且我已经知道许多人最终已经编织了一大堆昂贵的废物而离去，还有不少人最终什么也没有做出来。”



-- F.D. Brooks, Manager of OS/360
“The Mythical Man-Month”, 1974

Brooks' project, OS/360:

- **was late (over 5000 person year, 61-64)**
- **cost many times more than estimate (over $500M)**

Software Process Spectrum

- **Waterfall**
- Incremental
- **Prototyping**
- Spiral

Process Models

Light-weight Process

- Agile

- CMM
- SPICE

Ad Hoc development

Heavy-weight Process

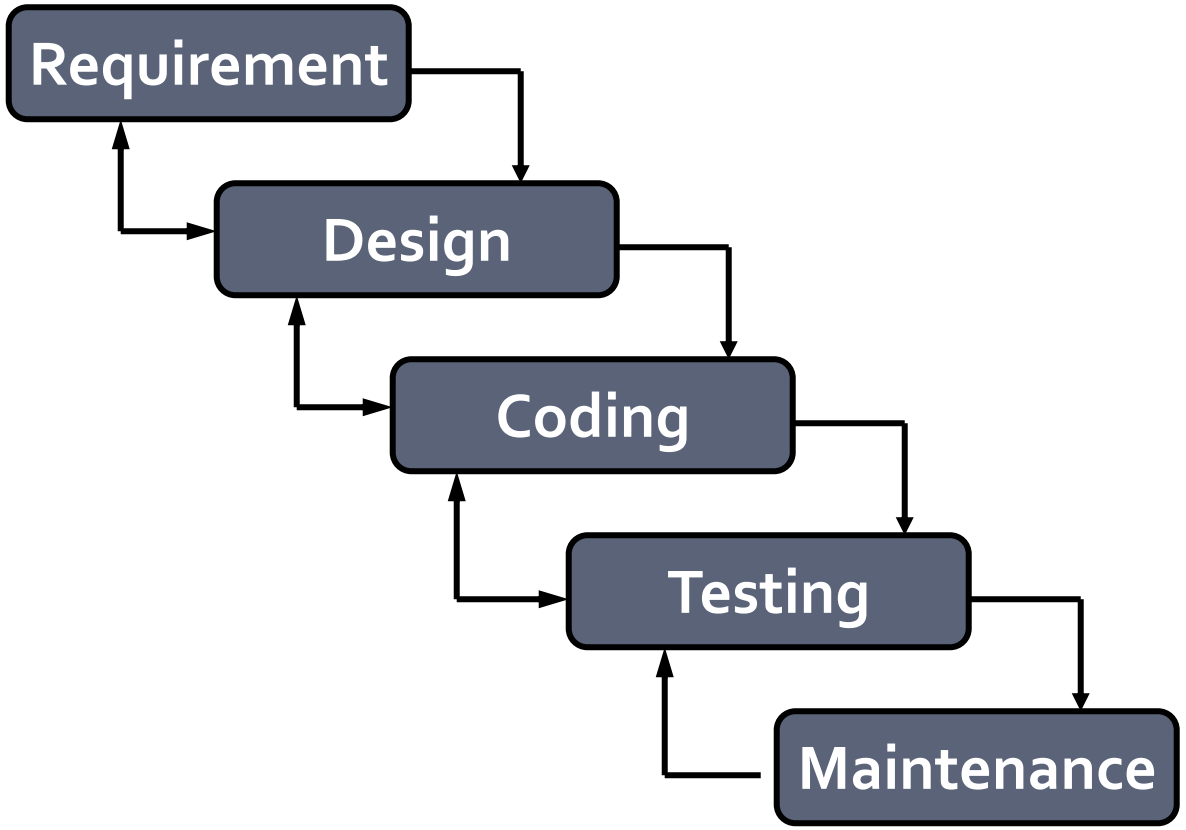
~1968: bring engineering discipline to software

- Why can't building software be like building a bridge? Safe, predictable cost & quality…
  - (though 90% infrastructure projects late/over $)
- "Plan-and-Document"
  - Before coding, project manager makes plan
  - Write detailed documentation all phases of plan
  - Progress measured against the plan
  - Changes to project must be reflected in documentation and possibly to plan

1st Development Process: Waterfall (1970), 5 phases

1. Requirements analysis & specification
2. Architectural design
3. Implementation & Integration
4. Verification
5. Operation & Maintenance

- Complete one phase before start next one
  - Why? Earlier catch bug, cheaper it is
  - Extensive documentation/phase for new people



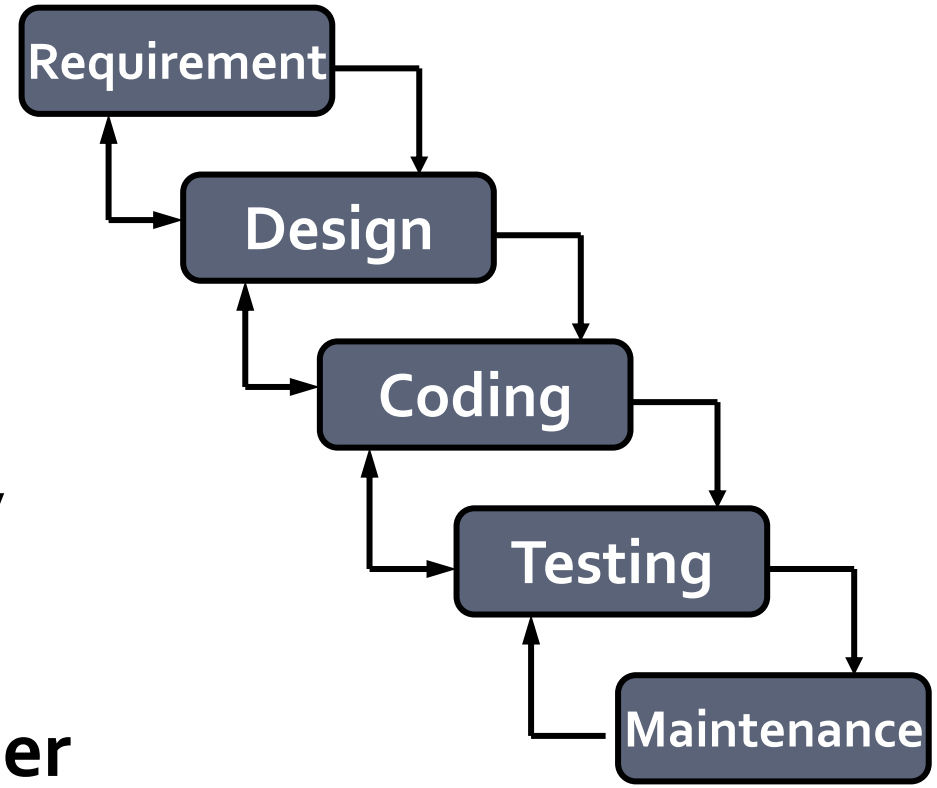
Was this a successful approach for SW development?

- ~420KLOC, ≤1 bug in last 3 versions
- 260 SW engineers working for one of only 4 dev shops to receive highest "maturity rating" from SW Engineering Institute@CMU
- Rigorous adherence to waterfall
  - ex: 6 KLoC change = 2500 pages of docs

They Write the Right Stuff. FastCompany, 1996.
Photo: Flickr user Matthew Simantov, CC-BY-SA

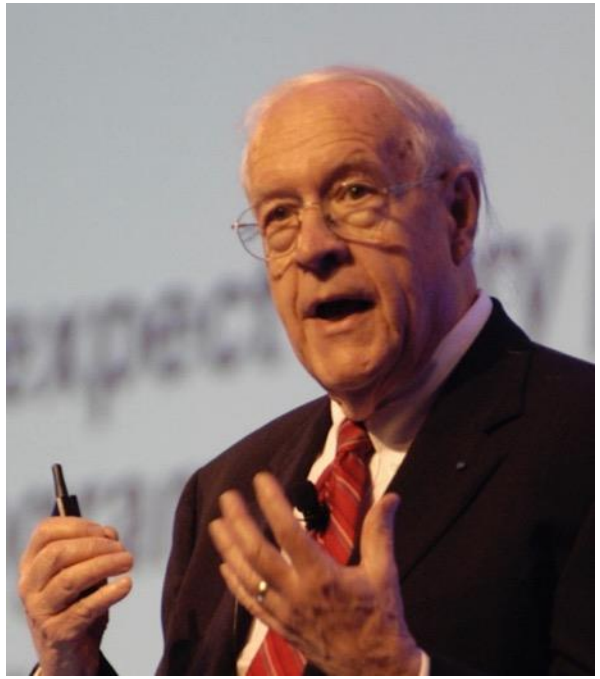
Was this a successful approach for SW development?

- ***And the users exclaimed with a laugh and a taunt: "It's just what we asked for, but not what we want." —Anonymous***
- **Software's strength is its evolvability— its ability to change and adapt— but that's a poor fit for "big design up front" (BDUF) or "top down" approaches**
- **Lengthy @ each step, bounced back to earlier steps when problems are encountered.**



How to deal with change?

- Software customers and end-users usually find it very difficult to express their real requirements.
- Hard to predicate how a system will affect working practices
- *"Plan to throw one [implementation] away; you will, anyhow."*
  - Fred Brooks, Jr. (1999 Turing Award winner)
- Often after build first one, developers learn right way they should have built it



(Photo by Carola Lauber of SD&M www.sdm.de. Used by permission under CC-BY-SA-3.0.)

Evolutionary Process Model – Prototyping

- Can we build software effectively without careful planning and documentation?
- Evolutionary Process Model: Enable software engineers to develop increasingly, iteratively more complete versions of the software
- Prototyping as the methods for requirement analysis
  - Throw-away prototyping
  - Evolutionary prototyping
- How to avoid “just hacking”?

Agile Manifesto

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

Individual 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. "

"Manifesto for Agile Software Development",
K. Beck, et al., 2001.

"The New Methodology"

by Martin Fowler

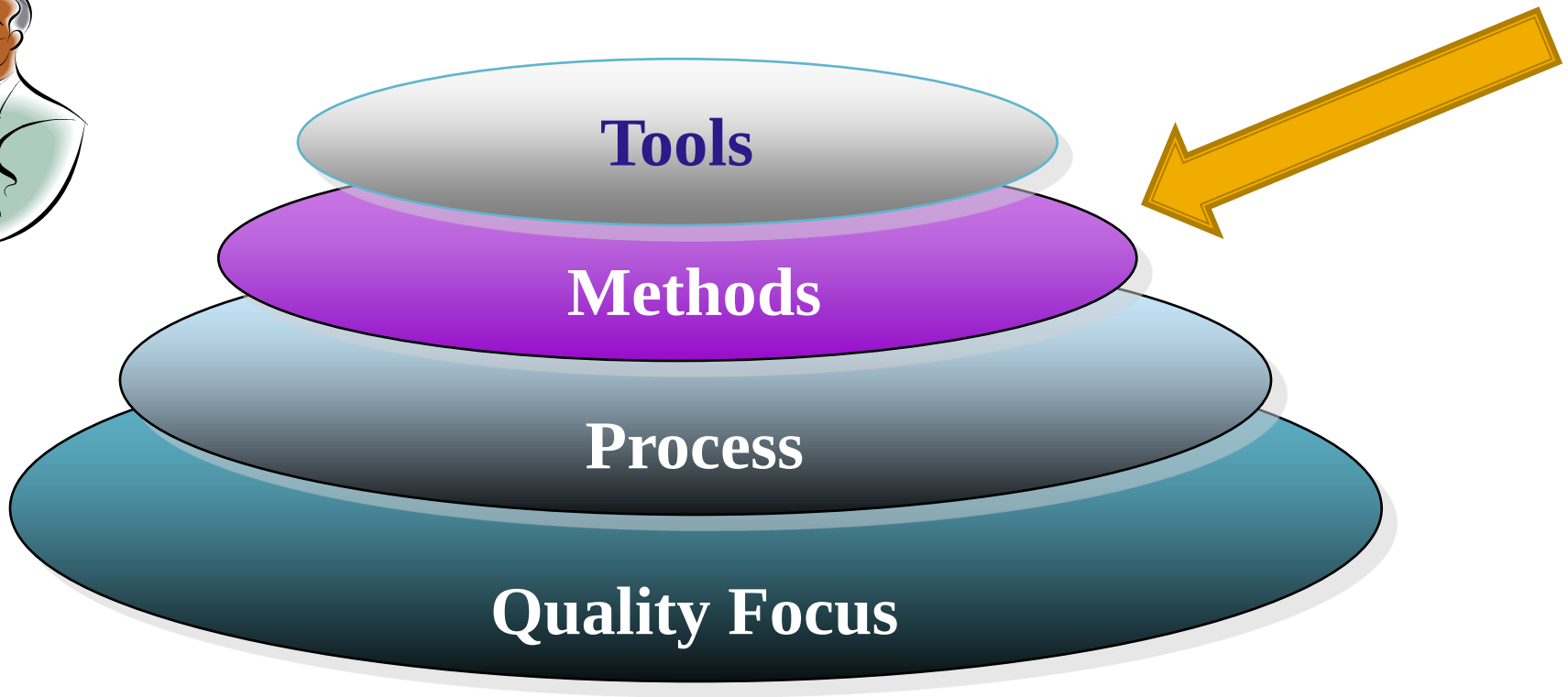
- **Adaptive rather than predictive**
  - **Heavy methods** tend to try to plan out a large part of the software process in great detail for a long span of time, this works well until things change. So their nature is to resist change.
  - **The light methods**, however, welcome change. They try to be processes that adapt and thrive on change, even to the point of changing themselves.
- **People-oriented rather than process-oriented**
  - They explicitly make a point of trying to **work with peoples' nature rather than against them** and to emphasize that **software development should be an enjoyable activity**.

Agile Pioneer,
Author of "Refactoring" Book

12 AGILE PRINCIPLES BEHIND THE AGILE MANIFESTO

1 Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.	2 Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.	3 Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4 Business people and developers must work together daily throughout the project.	5 Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.	6 Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
7 Working software is the primary measure of progress.	8 The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.	9 Continuous attention to technical excellence and good design enhances agility.
10 Simplicity – the art of maximizing the amount of work not done – is essential.	11 The best architectures, requirements, and designs emerge from self-organizing teams.	12 At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

A Layered Technology



Outline

- Evolution of Software Process Model
- Agile Methods
- Tools for Agility

Agile Methods

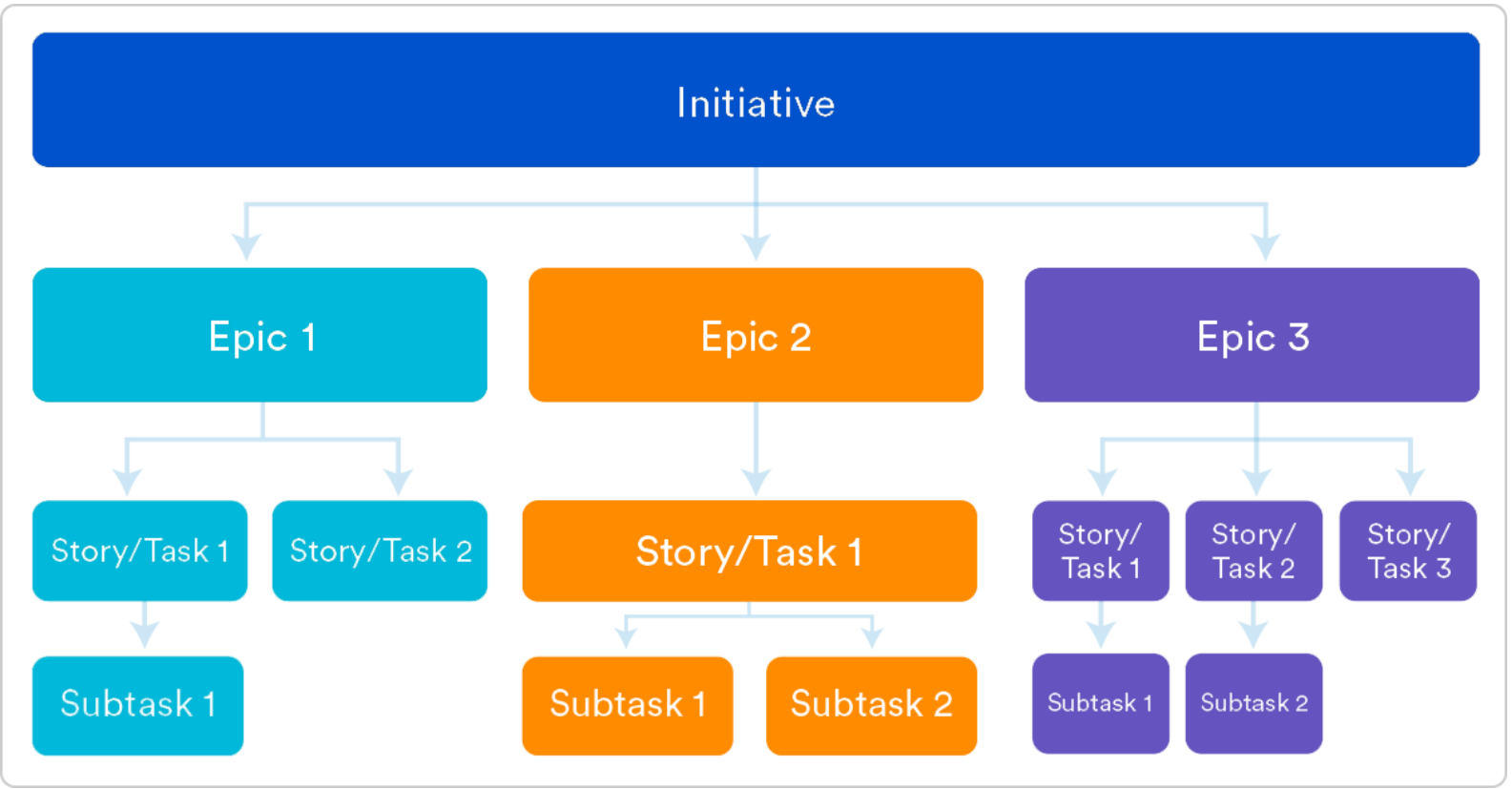
- Agile methods:
  - Scrum
  - Extreme Programming
  - Adaptive Software Development (ASD)
  - Dynamic System Development Method (DSDM)
  - …
- Agile Alliance
  - A non-profit organization promotes agile development

Agile with Scrum Process

In rugby, the term "scrum" refers to a method of restarting play where teams interlock arms, lower their heads, and push against the other team in order to gain possession of the ball.

The idea of a scrum is teamwork. All players are interlocked and move together as a team toward the same goal. No individual player stands alone.

Initiatives, Epics, and Stories



- **Stories**, also called "user stories," are short requirements or requests written from the perspective of an end user. Completed within one or two-week sprint.

- An **epic** is a large body of work that can be broken down into a number of smaller stories. Epics are almost always delivered over a set of sprints, within a month or a quarter. 2-3 epics per quarter. Scope are flexible: Its user stories can be added/removed.
- **Initiative**s are collections of epics that drive toward a common goal. Completed in multiple quarters to year

用户故事表达：卡片

作为一名<角色>，
我可以<活动>，
使得<业务价值>。

作为一名消费者（<角色>）

我希望能够查看我的日常

能源使用量（<我利用系统

做什么>），从而我可以

降低能源支出和使用率

（<我收到的业务价值>）。

验收标准

- 每10秒钟读取DecaWatt计量表的数据，在门户上显示15分钟的增量，在室内屏上显示每一读数；
- 随着新数据的产生读取KiloWatt计量表，并且每小时显示在门户上，在每次读取后显示在室内屏上；

……

良好的用户故事 INVEST

Independent 独立性

故事不依赖于其它故事，可以被**单独**地开发、测试甚至交付。

Negotiable 可协商

故事不是详尽的需求说明书（系统应该完成的事），而是可协商的**意向表达**。

Valuable 有价值

聚焦与由**用户**规定的**价值**，而非功能分解结构

Estimable 可估算

故事相对而言更**容易估算**，可以快速确定实现相关功能的工作量。

Small 小型

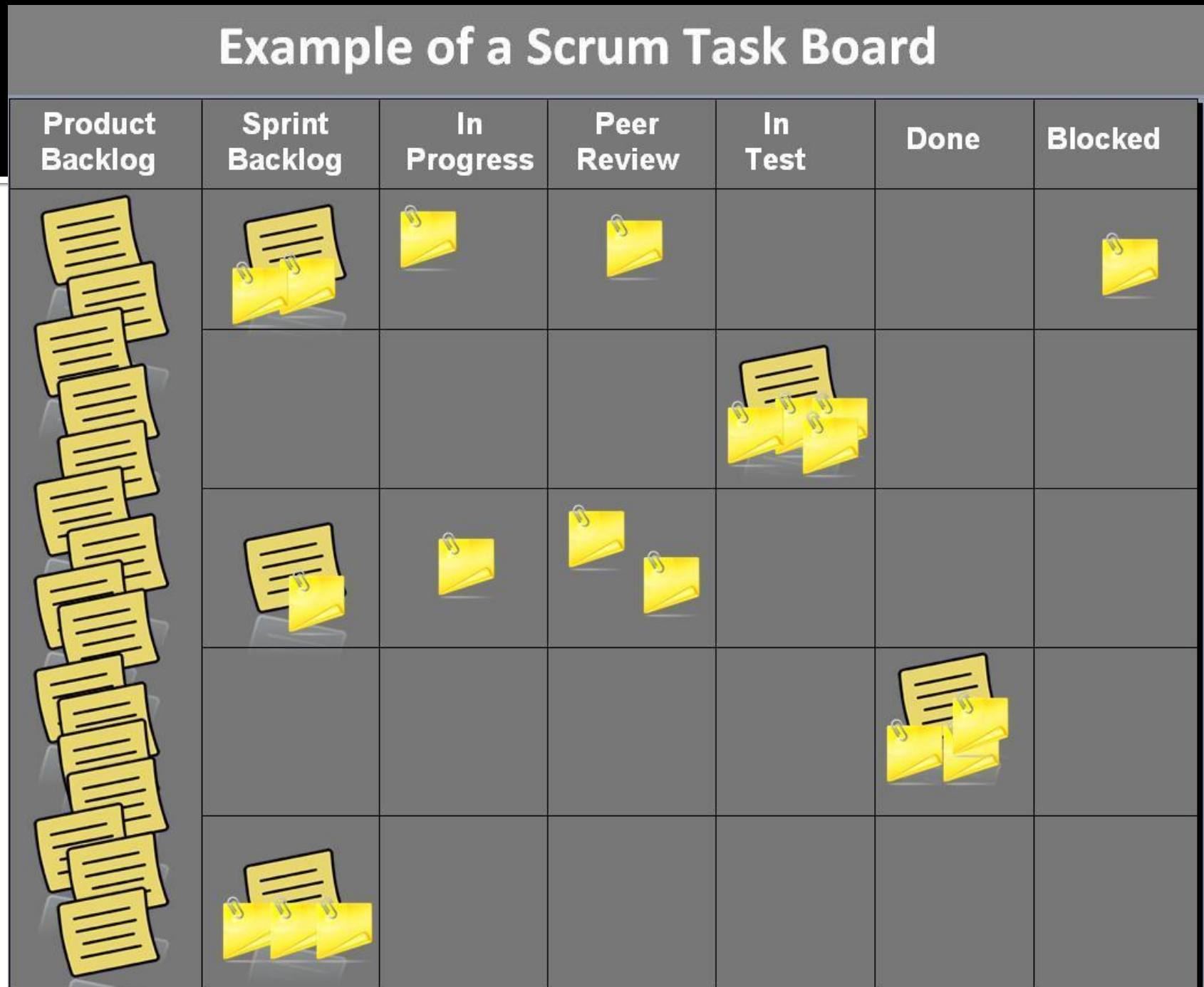
故事代表有**价值功能**的**小型增量**，可以在一到两周内交付。

Testable 可测试

清晰可测试

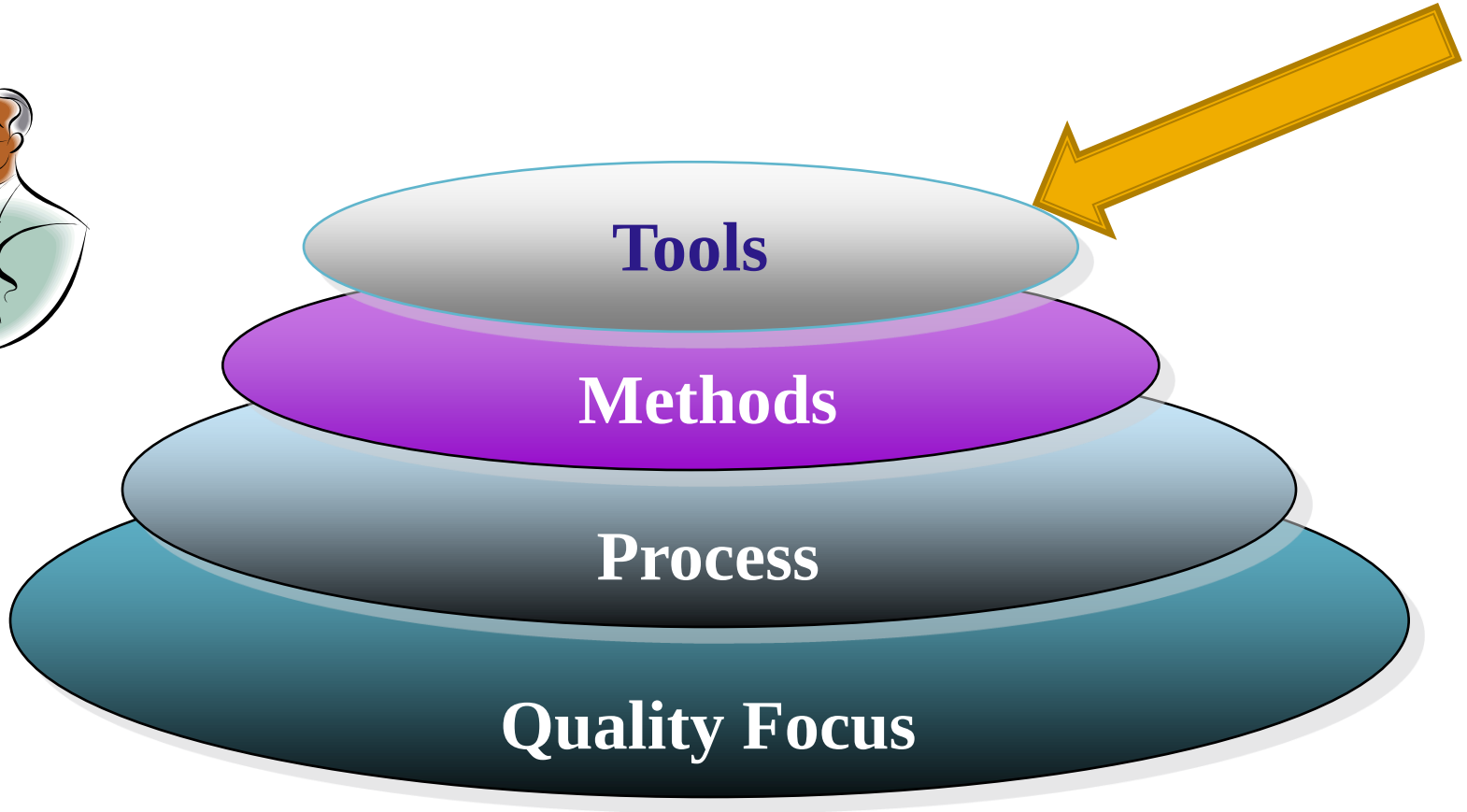
Product Backlogs:

1. User Stories
2. Technical Stories
3. Spikes
   - Experimental stories
4. Defects




Product Owner
Scrum Master
Scrum Team Members
Stakeholders
Sprint Review Meeting
Daily Scrum
Review
Implementation
Retrospect
S01-Core features (Web)
[1 of 1]
2018-08-16 [14 days left]
5/7
35%
Scrum Board
Sprint Report
Impediment Log
Burndown Chart
Sprint Retrospective Meeting
Identify Project Vision
Identify Business Goals
Manage Epics
Manage User Stories
Create / Switch Sprint
Planning
Project Retrospective Meeting
Project Vision
Prioritized Use Cases
Prioritized Epics
Prioritized User Stories
Sprint Planning Meeting
Sprint Backlog

A Layered Technology

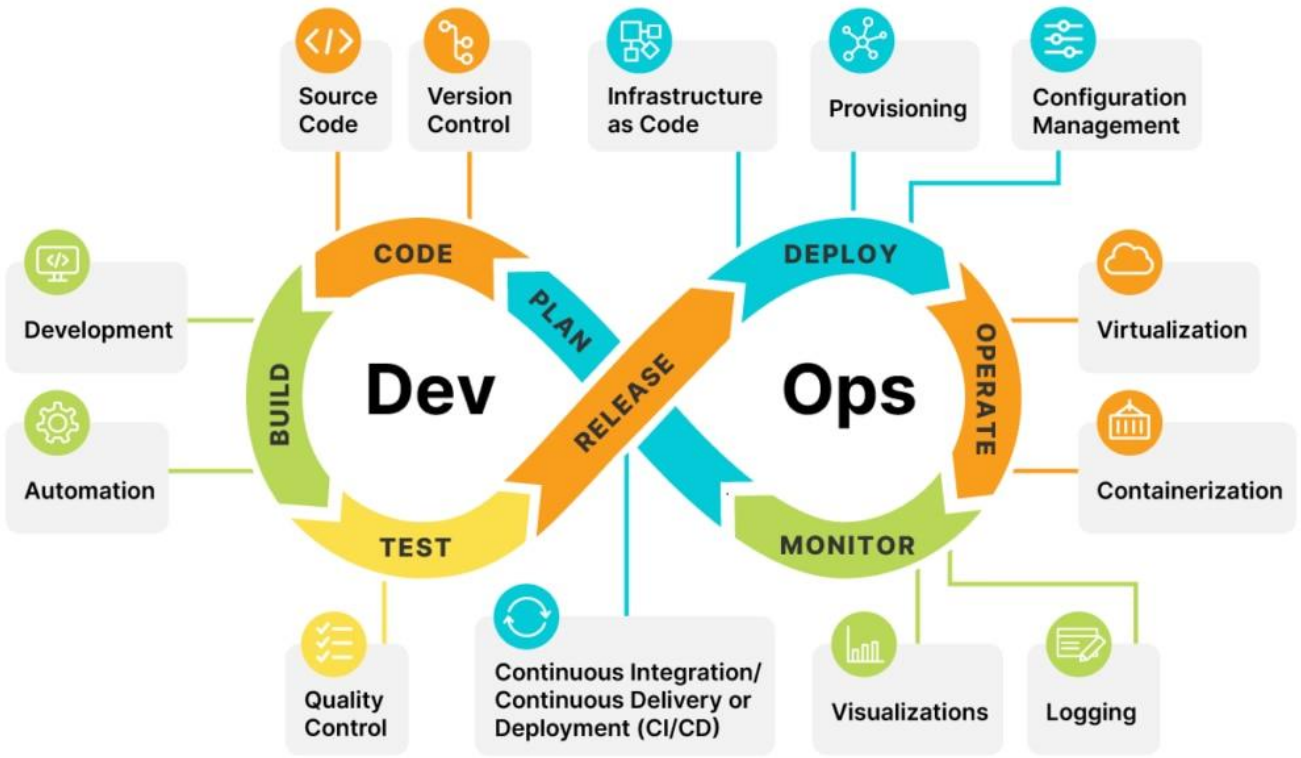


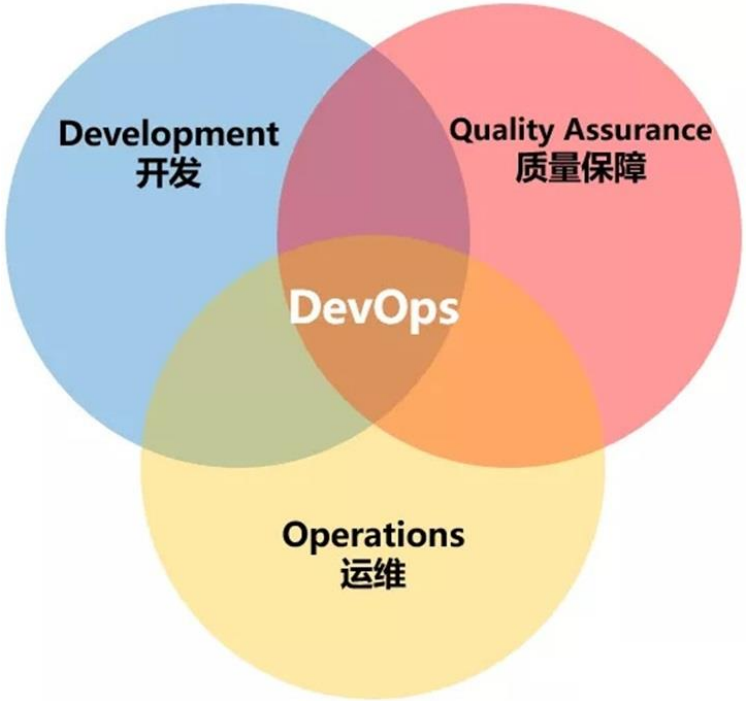
Outline

- Evolution of Software Process Model
- Agile Methods
- Tools for Agility

GitLab – a DevOps Tool/Platform

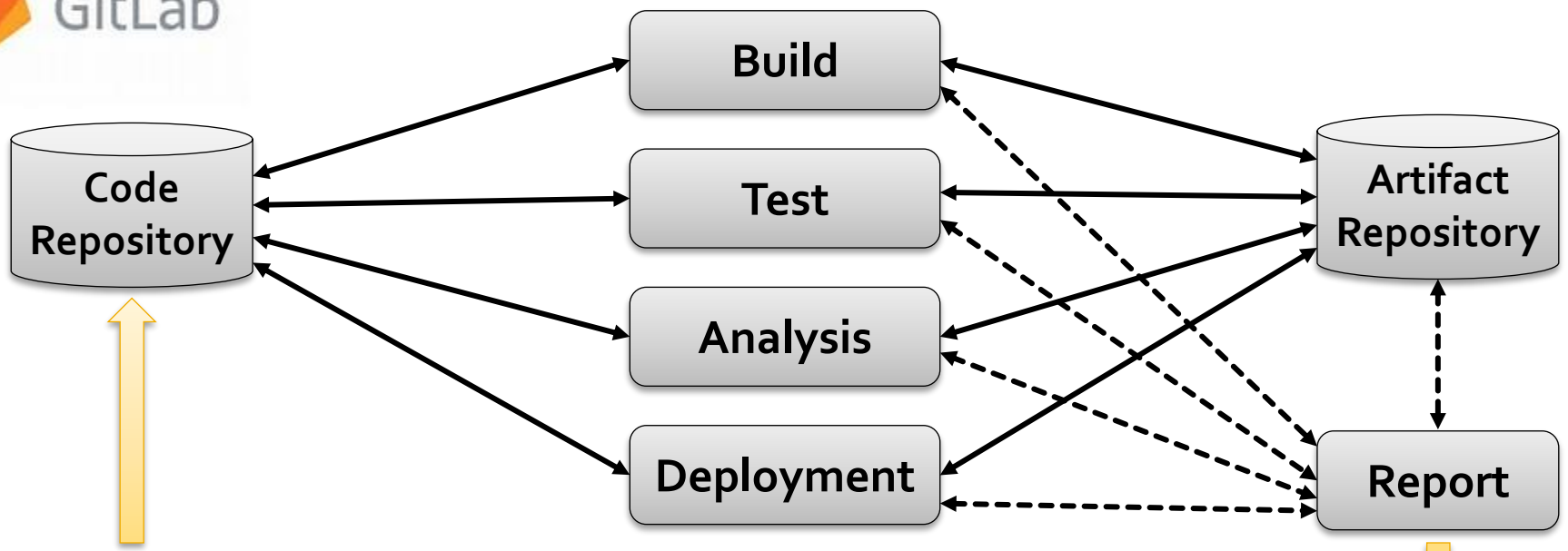
DevOps (Development + Operations)





- DevOps是一组过程、方法与系统的统称，用于促进**开发**、**技术运营**和**质量保障（QA）**部门之间的沟通、协作与整合
- DevOps就是让开发人员和运维人员更好地沟通合作，通过自动化流程来使得软件整体过程更加快捷和可靠

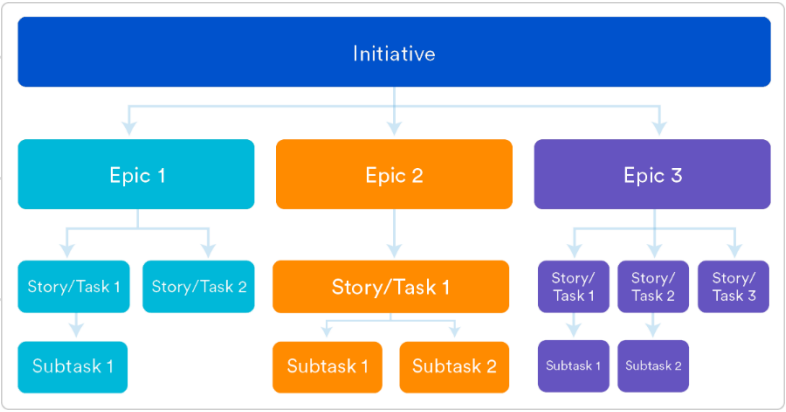
GitLab



- Allow multiple integrations per day
- Each integration is automated built (tested, integrated) to generate a immediate deployable system

Mapping Agile artifacts to GitLab features

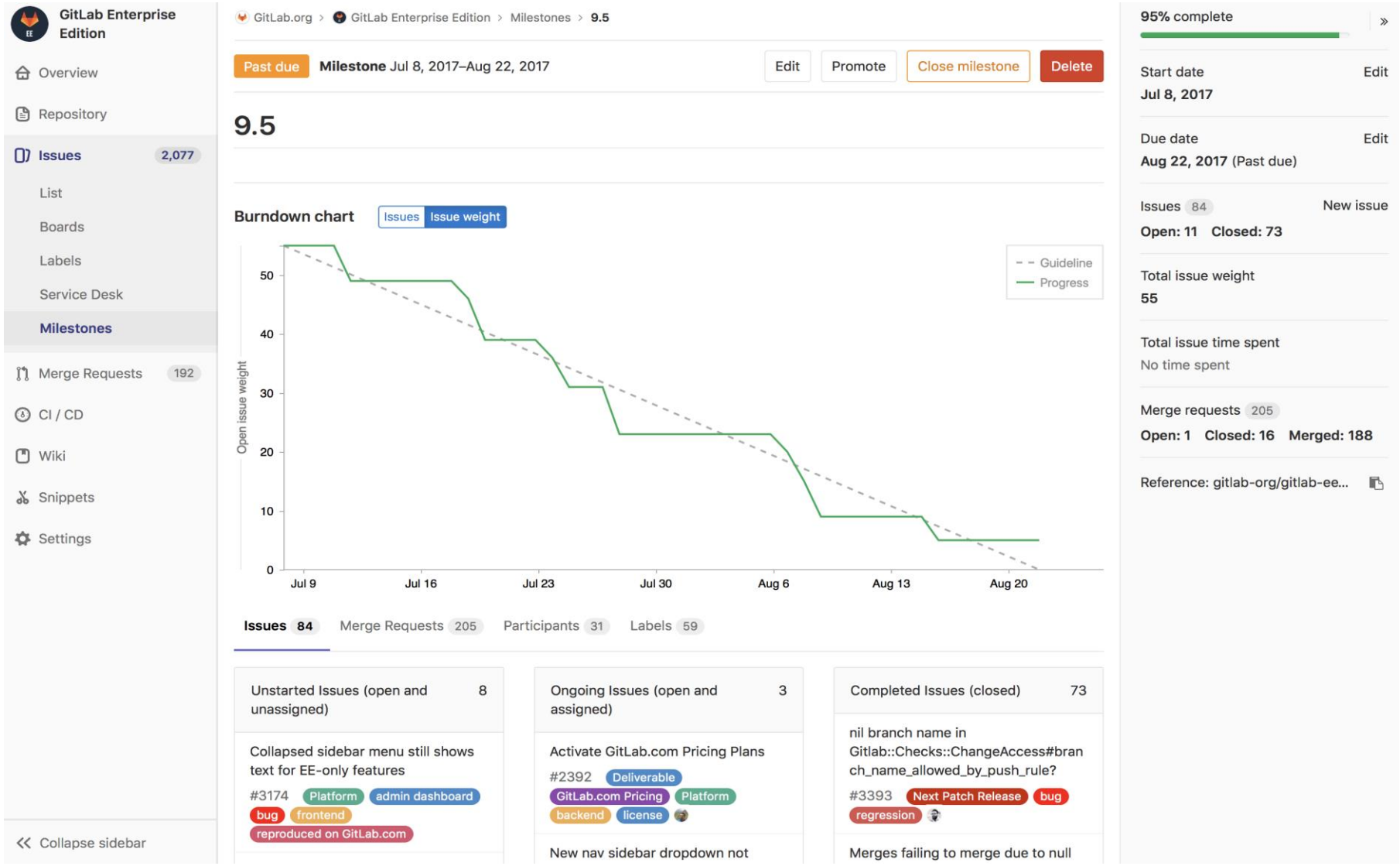
Agile artifact	GitLab feature
User story	Issues
Task	Task lists
Epic	Epics
Points and estimation	Weights
Product backlog	Issue lists and prioritized labels
Sprint/iteration	Milestones
Burndown chart	Burndown charts
Agile board	Issue boards



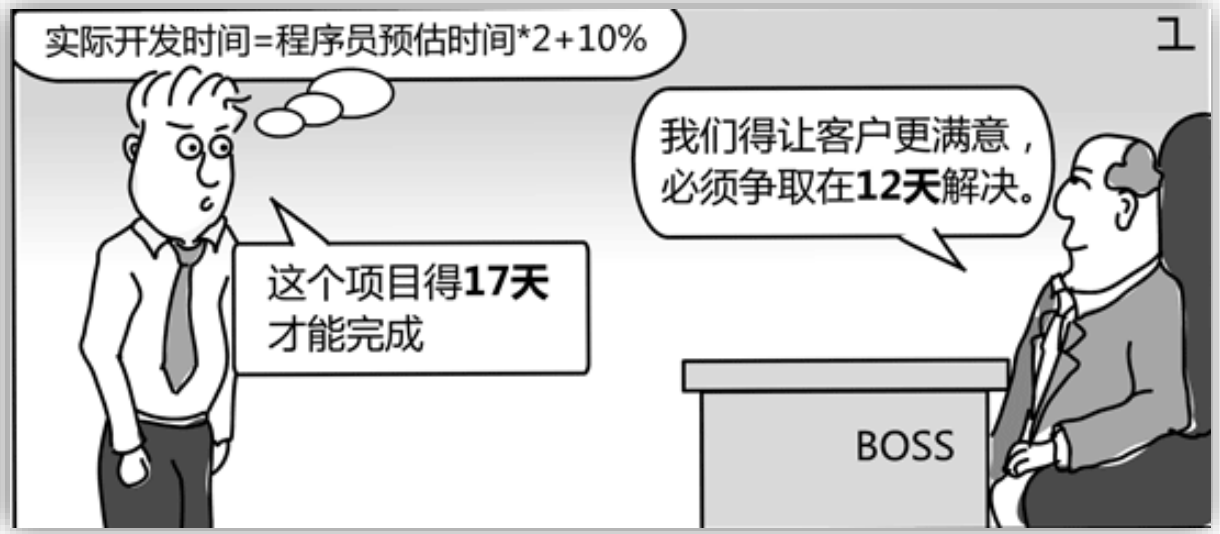
https://about.gitlab.com/blog/2018/03/05/gitlab-for-agile-software-development/

Burndown Chart – a workload management tool

Burndown chart




一
预估这个项目完成
需要多久？
项目经理
8天就够了
高级程序员
没有我2个晚上
搞不定的代码。
普通程序员

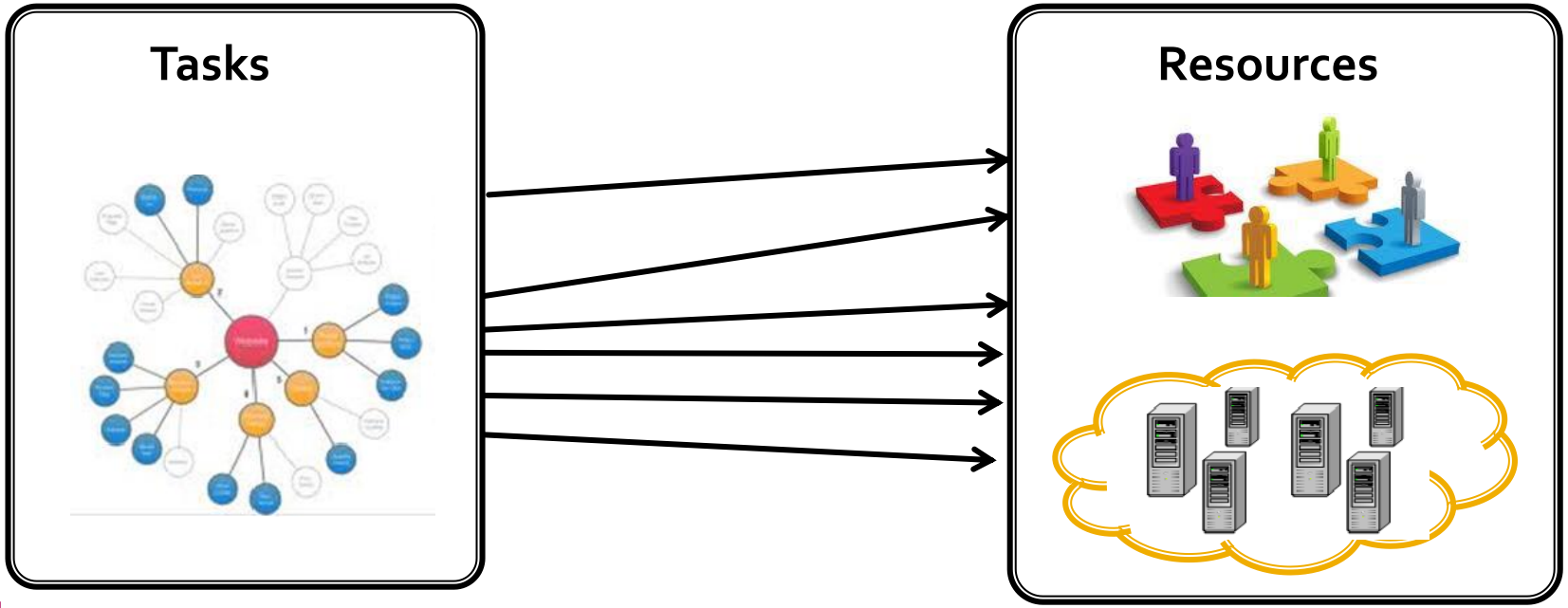

实际开发时间=程序员预估时间*2+10%
这个项目得17天
才能完成
我们得让客户更满意，
必须争取在12天解决。
BOSS


二
我们是敏捷开发的行业领头羊，
只需要8天，
你再也找不到比我们
更靠谱的乙方了。
销售


三
进度怎么样了，
开发是否完成？
8天后……


四
别烦我们，正忙着呢，
一个大BUG。

Scheduling: Tasks vs Resources



Question:

"How does a project get to be a year late?"

Answer:

"One day at a time."

Scheduling in Agile Process

- Not to worry about **critical path** within an iteration because of the daily standups

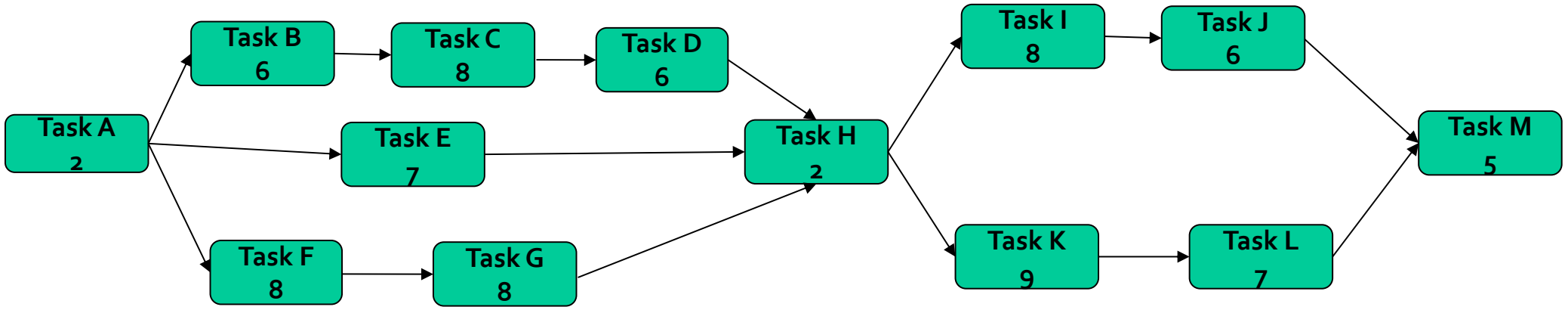
- Rolling lookahead planning: teams (on projects with multiple teams) conclude their **sprint planning meetings** by taking a five minute look ahead at the next 1-3 iterations, identifying dependencies between teams.

- On a very complex project:
  - Try first to write mostly independent product backlog items. If a project's product backlog items are independent and can be developed in any sequence then critical path issues disappear.

  - Drawing a task network diagram to see the relationships among the next 3-6 months worth of user stories, and conduct critical path analysis

Task Network Diagram

- Depicts project task length, sequence, concurrency, and dependency
- Critical path is the longest-duration path through the network.
  - The activities lie on it cannot be delayed without delaying the project.
  - To accelerate the project, it is necessary to reduce the total time required for the activities in the critical path



Critical Path: A-B-C-D-H-K-L-M

CLASS EXERCISE

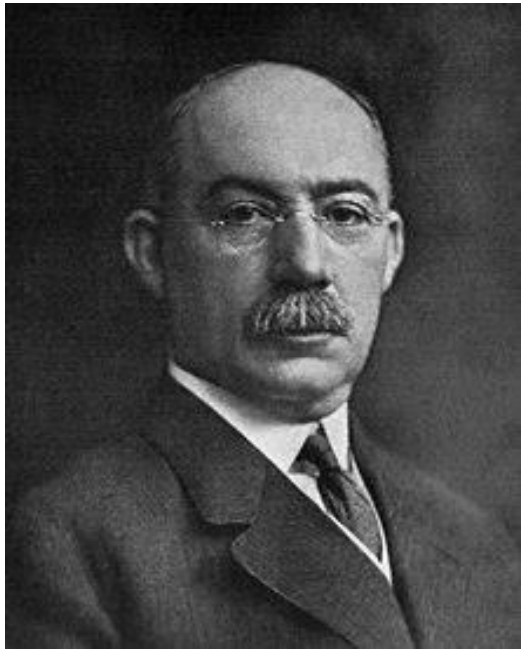
Timeline chart:

Task #	Task Name	Duration	Start	Finish	Pred.	4/1	4/8	4/15	4/22	4/29	5/6	5/13	5/20	5/27	
A	Establish increments	3	4/1		None										
B	Analyze Inc One	3			A										
C	Design Inc One	8			B										
D	Code Inc One	7			C										
E	Test Inc One	10			D										
F	Install Inc One	5			E										
G	Analyze Inc Two	7			A, B										
H	Design Inc Two	5			G										
I	Code Inc Two	4			H										
J	Test Inc Two	6			E, I										
K	Install Inc Two	2			J										
L	Close out project	2			F, K										

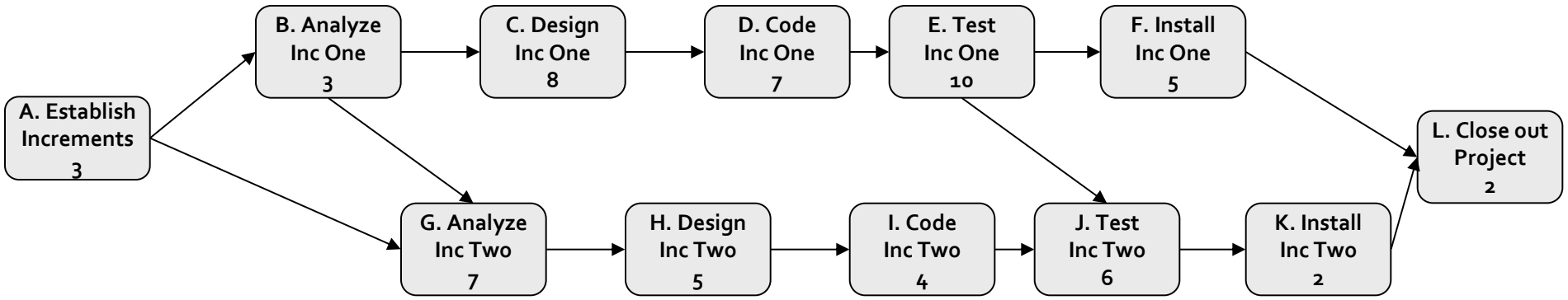
SOLUTION

Timeline chart:

Task #	Task Name	Duration	Start	Finish	Pred.
A	Establish increments	3	4/1	4/3	None
B	Analyze Inc One	3	4/4	4/6	A
C	Design Inc One	8	4/7	4/14	B
D	Code Inc One	7	4/15	4/21	C
E	Test Inc One	10	4/22	5/1	D
F	Install Inc One	5	5/2	5/6	E
G	Analyze Inc Two	7	4/7	4/13	A, B
H	Design Inc Two	5	4/14	4/18	G
I	Code Inc Two	4	4/19	4/22	H
J	Test Inc Two	6	5/2	5/7	E, I
K	Install Inc Two	2	5/8	5/9	J
L	Close out project	2	5/10	5/11	F, K



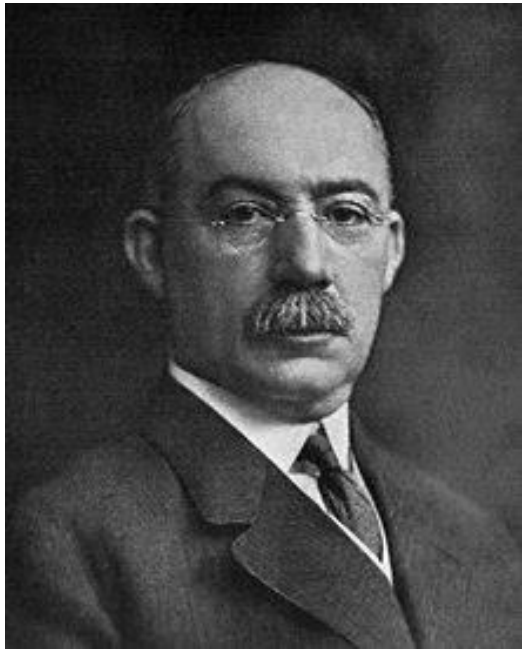
Henry Gantt, (1861-1919)
inventor of the Gantt chart



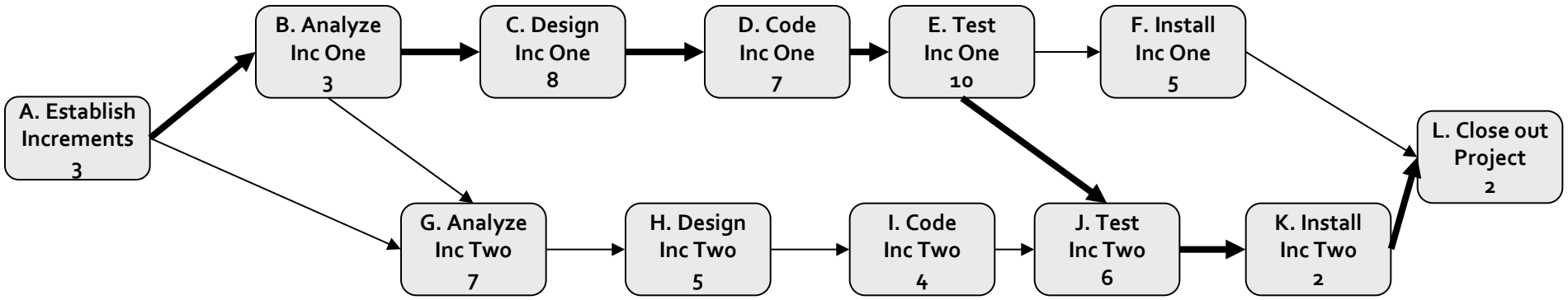
SOLUTION

Timeline chart:

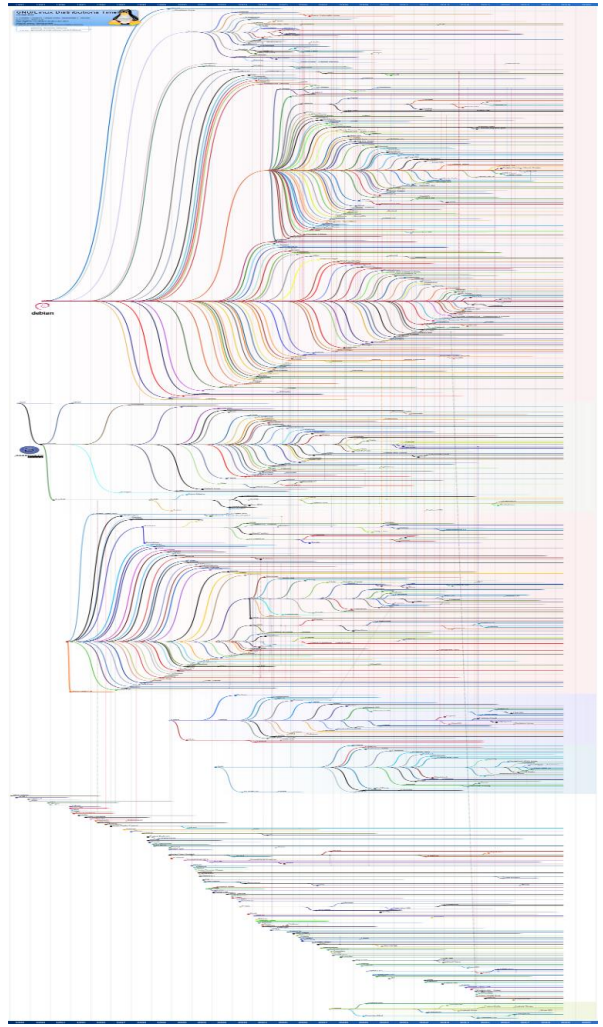
Task #	Task Name	Duration	Start	Finish	Pred.
A	Establish increments	3	4/1	4/3	None
B	Analyze Inc One	3	4/4	4/6	A
C	Design Inc One	8	4/7	4/14	B
D	Code Inc One	7	4/15	4/21	C
E	Test Inc One	10	4/22	5/1	D
F	Install Inc One	5	5/2	5/6	E
G	Analyze Inc Two	7	4/7	4/13	A, B
H	Design Inc Two	5	4/14	4/18	G
I	Code Inc Two	4	4/19	4/22	H
J	Test Inc Two	6	5/2	5/7	E, I
K	Install Inc Two	2	5/8	5/9	J
L	Close out project	2	5/10	5/11	F, K



Henry Gantt, (1861-1919)
inventor of the Gantt chart



Git – a Version Control Tool



Terminology: Release vs. Version vs. Revision

- Many naming schemes exist. A 3-digit scheme is quite common.

A.B.C

  - A: the release number from customer perspective
    - The formal distribution of an approved version.
  - B: the version number from developer perspective
    - The formal distribution of an approved version.
  - C: the revision number from developer perspective
    - Change to a version that corrects only errors in the design/code, but does not affect the documented functionality.

Version Control Tools

SCCS & RCS (70s-80s)

CVS (1986)

Subversion (SVN,2001)

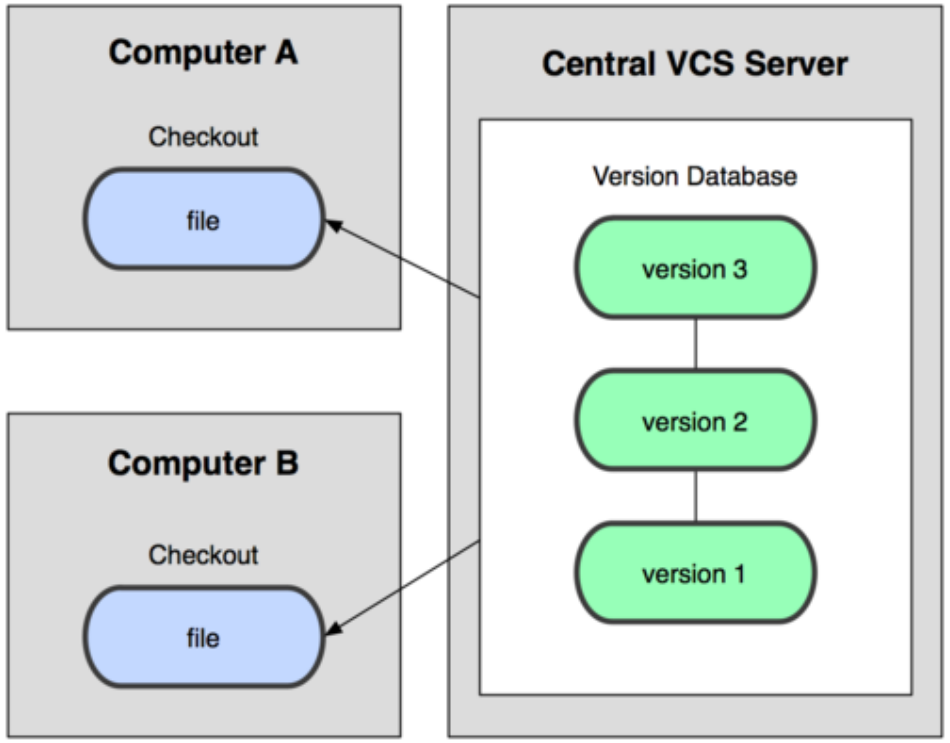
Git (2005)

Image © TheSun.au

Git uses a distributed model

Came out of Linux development community
Linus Torvalds, 2005

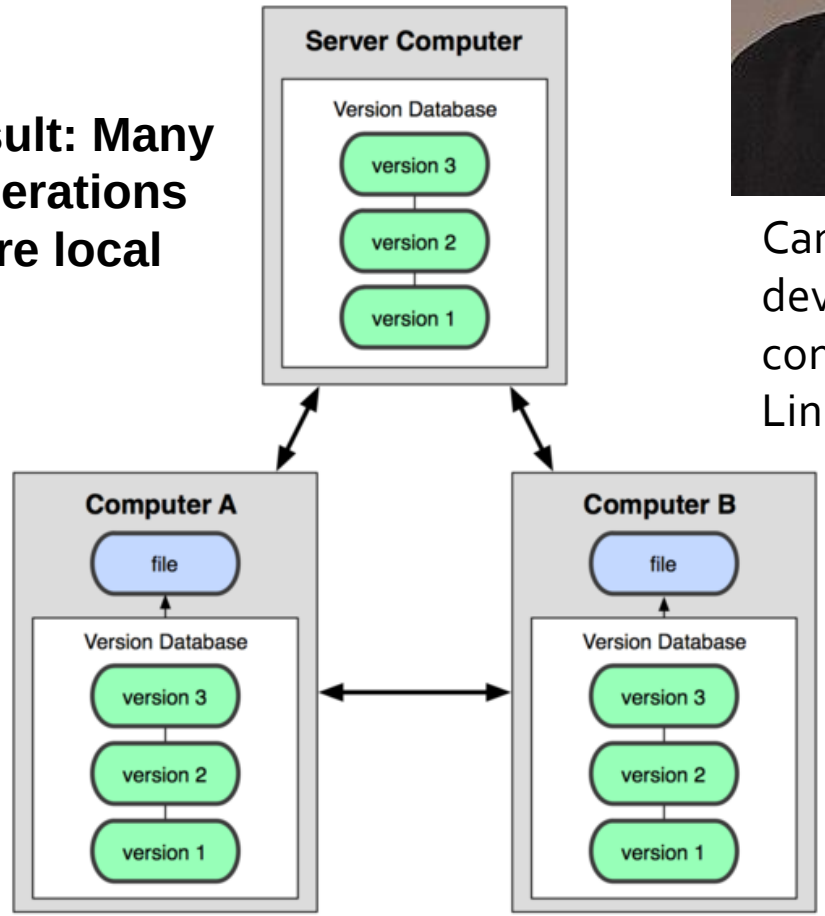
Centralized Model



(CVS, Subversion, Perforce)

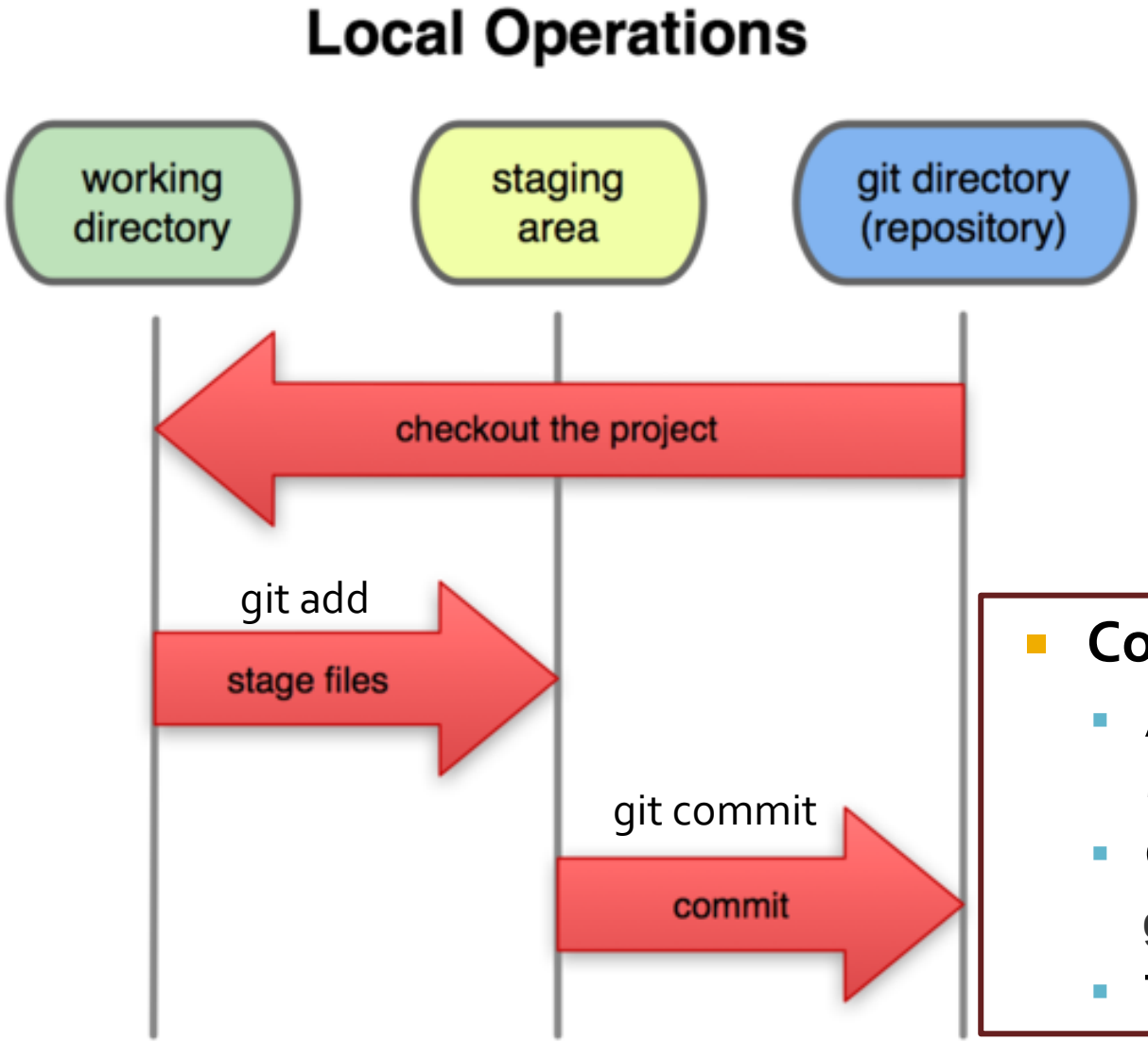
Distributed Model

Result: Many operations are local



Git

A Local Git project has three areas

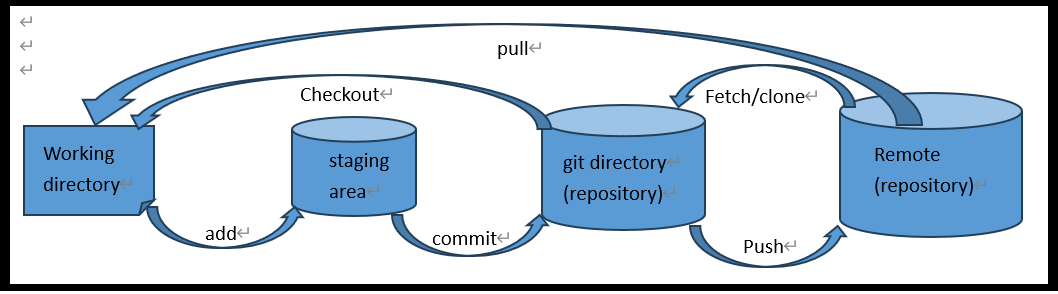


- **Working directory**, Unmodified/modified files
- **Staging area**, Staged files
- **Git directory (repository)**, Committed files

- **Commit**
  - A commit message is important to identify what is being changed and, more importantly, why.
  - Cross-link issues

    `git commit -m "this is my commit message. Ref #xxx"`
  - Trigger Build/Test

Git Basics



Setting up a Git Repository

Git init

Create a new repository and place a project under revision control.

Git clone

Create a copy of an existing repository.

Git config

Set configuration options for Git installation.

Recording Snapshots

Git add

Move changes from the working directory to the staging area, to prepare a snapshot before committing it to the official history.

Git commit

Take the staged snapshot and commit it to the project history.

Inspecting a Git Repository

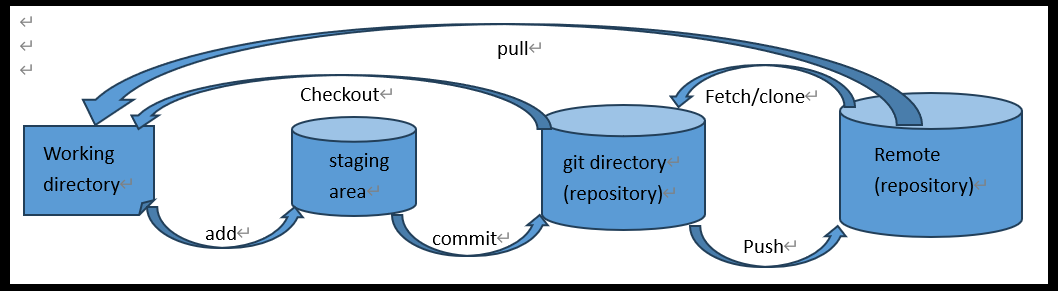
Git status

Display the state of the working directory and the staged snapshot.

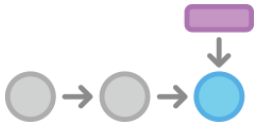
Git log

Explore project revision history.

Git Basics

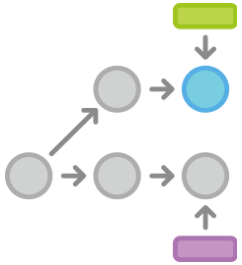


Git Branches



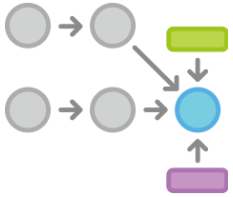
Git branch

Create isolated development environments within a single repository.



Git checkout

Check out old commits and file revision, and navigate existing branches.



Git merge

Integrate changes from divergent branches.

Remote repository

Git remote

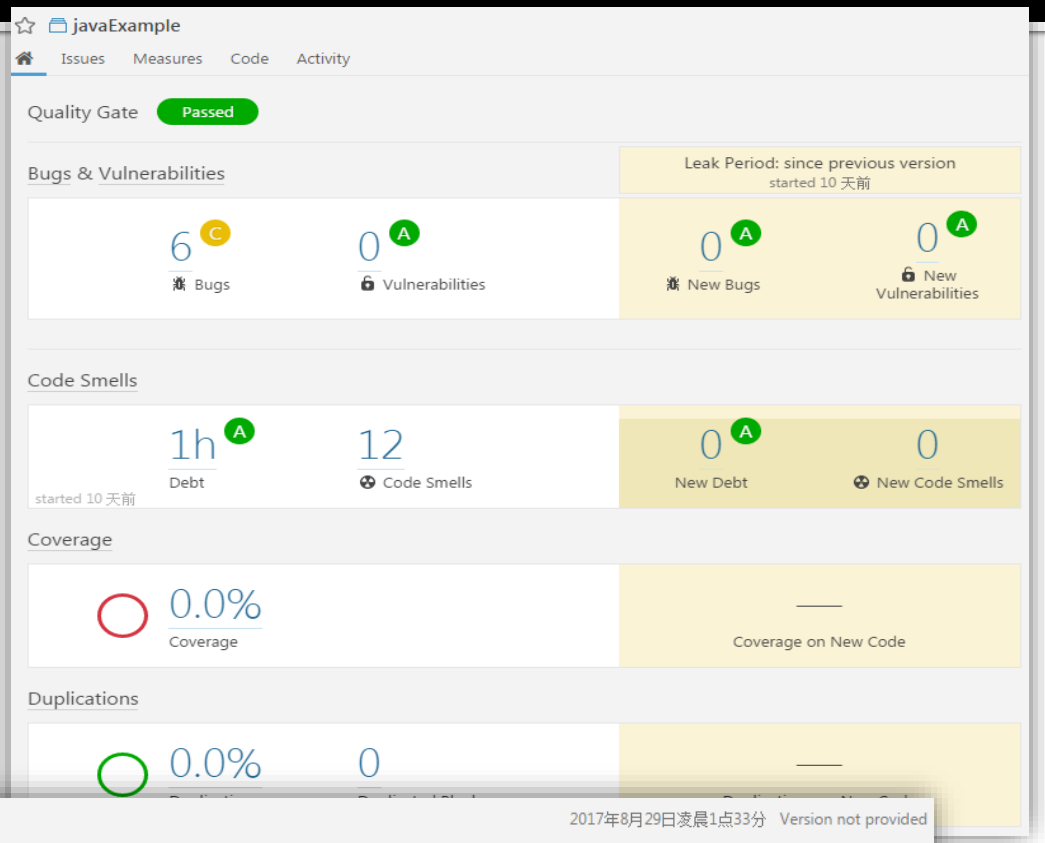
Git fetch

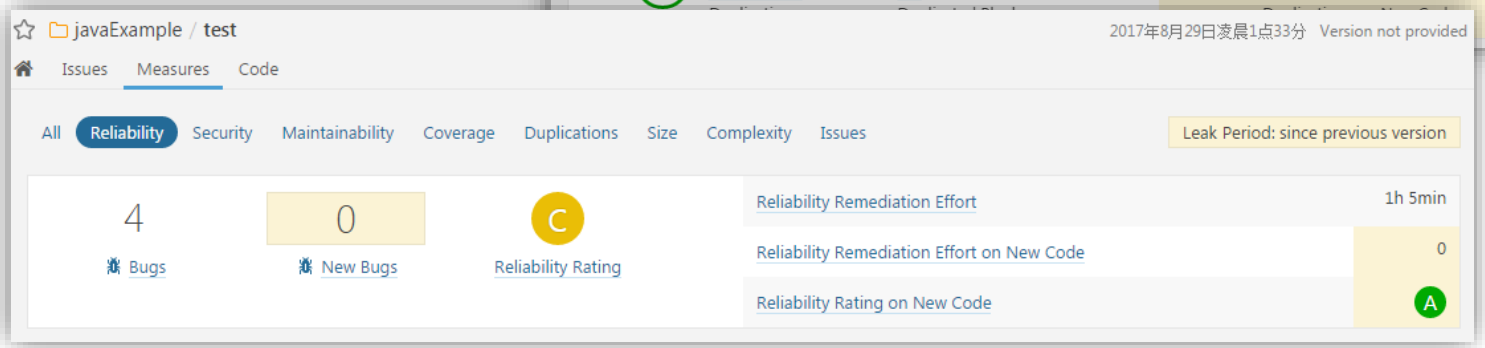
Git pull

Git push

Sonarqube – a Code Quality Tool

- Continuous inspection
  - Automatic review with static analysis of code quality
- Health
  - Duplicated code
  - Coding standards
  - Code complexity
  - Comments
  - Bugs
  - Security vulnerabilities
  - ……





Thank you!