Software Engineering

FEUP-MIEIC-ESOF-2019-20

Ademar Aguiar, Filipe Correia

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO


In short, software is eating the world
— Marc Andreessen —
AZ QUOTES

SOFTWARE IS EATING THE WORLD

1980	1990	2000	2010	2017
1. IBM	1. IBM	1. General Electric	1. Exxon Mobil	1. Apple
2. AT&T	2. Exxon	2. Exxon Mobil	2. Apple	2. Alphabet
3. Exxon	3. General Electric	3. Pfizer	3. Microsoft	3. Microsoft
4. Standard Oil of Indiana	4. Philip Morris	4. Citigroup	4. Berkshire Hathaway	4. Facebook
5. Schlumberger	5. Royal Dutch Petrol	5. Cisco Systems	5. General Electric	5. Amazon


Mercedes Class S
close to 100 million lines of code, executing in 70 to 100 electronic control unit based on microprocessors, in a vehicular network
S SK 5319
http://spectrum.ieee.org/transportation/systems/this-car-runs-on-code

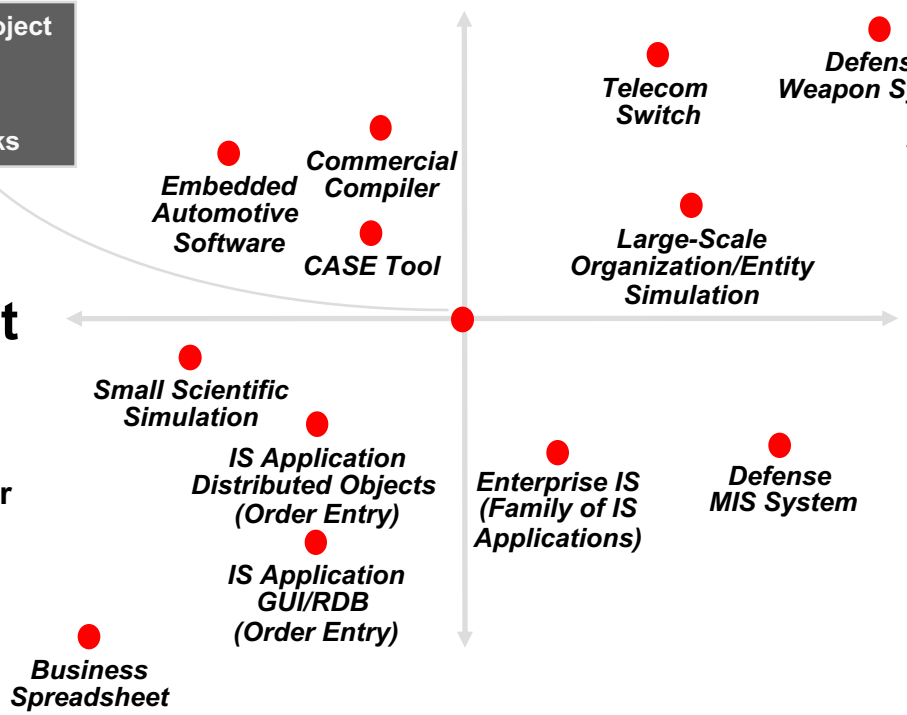
Airbus A380
120 million lines of code (*avionics* and onboard support systems)

https://www.computerweekly.com/news/2240233948/Airbus-Interview-research-innovation-and-the-future-of-aviation

Higher technical complexity

- Embedded, real-time, distributed, fault-tolerant
- Custom, unprecedented, architecture reengineering
- High performance

An average software project
- 5-10 people
- 3-9 month duration
- 3-5 external interfaces
- Some unknowns & risks



Lower management complexity

- Small scale
- Informal
- Single stakeholder
- "Products"

Higher management complexity

- Large scale
- Contractual
- Many stakeholders
- "Projects"

Lower technical complexity

- Mostly 4GL, or component-based
- Application reengineering
- Interactive performance

Walker Royce

Historical episodes of software issues

- 1983 Soviet nuclear false alarm incident - G8
- Air Control Failure - G7
- Airbus A380 - G5
- Amazon UK Christmas 2014 G6
- Apple's GOTO bug - G8
- Ariane 5 Explosion (1997) - Grupo 1
- Fiat Chrysler Software Recall (2016) - G7
- Ford Motor Company and Oracle G6
- Hartford Coliseum Collapse (1978) - G5
- Heartbleed (2014) - G8
- Heathrow terminal 5 opening 2008 - G7
- HSBC Online Banking Glitch - G6
- Inmates Released Early - Grupo 1
- Intel Pentium's FDIV bug - G1
- IPO Facebook G6
- IPv4 address exhaustion - G2
- Japanese Hitomi Spacecraft - G4
- Jeep Cherokee Steering Attacks - G1
- Knight's 440$ Million Glitch - G4
- Mariner 1 Launch Failure (1961) - G7
- Medical Machine Kills (1985) - Grupo 2
- Millennium Bug - G1
- MIM Patriot failure in Saudi Arabia - G7
- Mt Gox bitcoin hack - G8
- Multidata Systems (2000) - G4
- NASA's Mars Climate Orbiter - G3
- Nissan's Airbag Software Malfunction (2013) - G2
- North American Blackout (August 14, 2003) - Grupo 3
- Northeast Blackout of 2003 - G5
- Pentium Fails Long Division (1993) - Grupo 2
- Phobos 1 Drone Malfunction - G3
- Swedish Social Security Agency - G2
- Tesla driver killed in crash with Autopilot active - G5
- The 2003 Northeast Blackout - G2
- The Collapse of the Hartford Civic Center - G3
- The Meltdown Attack - G5
- The Morris Worm - G8
- The Patriot Missile Failure - G4
- Therac 25(1985 1987) - G7
- Toyota´s electronic throttle control system failure - G1
- Uber - False Positive - G6
- US Prisoners Released Early - G1
- USS Yorktown Incident - G8
- WannaCry Ransomware Cyber Attack - G6
- Y2K bug (2000) - G2
- Yahoo! Data Theft - G8

www.tricentis.com/resources/software-fail-watch-5th-edition/#

Explore Tricentis: qTest | Tosca | Flood | RPA

Support Community Company CIO Corner Blog English

TRICENTIS

Products Solutions Services Resources

Contact Us

Free Trials Demos

WHITE PAPERS

Software Fail Watch: 5th Edition

White Paper

The Software Fail Watch is an analysis of software failures found in a year's worth of English language news articles. The result is an extraordinary reminder of the role software plays in our daily lives, the necessity of software testing in every industry, and the far-reaching impacts of its failure.

The 5th Edition of the Software Fail Watch identified 606 recorded software failures, impacting half of the world's population (3.7 billion people), $1.7 trillion in assets, and 314 companies. And this is just scratching the surface—there are far more software defects in the world than we will likely ever know about.

LOSSES FROM SOFTWARE FAILURES (USD)

1,715,430,778,504

ONETRILLIONSEVENHUNDREDFIFTEENBILLIONFOURHUNDREDTHIRTYMILLIONSEVENHUNDREDSEVENTY-EIGHTTHOUSANDFIVEHUNDREDFOUR

Download Now

*Salutation: Mr.

*Email Address

*First Name:

*Last Name:

*Company Name:


NOUN
RSET
0 1 2 3
KEY
REL

Why do software projets fail so often?

- **Unrealistic** or unarticulated project **goals**
- Inaccurate **estimates** of needed resources
- Badly defined system **requirements**
- Poor **reporting** of the project's **status**
- Unmanaged **risks**
- Poor **communication** among customers, developers, and users
- Use of **immature technology**
- Inability to handle the project's **complexity**
- Sloppy development practices
- Poor project management
- Stakeholder politics
- Commercial pressures

http://spectrum.ieee.org/computing/software/why-software-fails

What is Software Engineering

- "Research, design, develop, and test operating systems-level software, compilers, and network distribution software for medical, industrial, military, communications, aerospace, business, scientific, and general computing applications"—Bureau of Labor Statistics
- "the systematic application of scientific and technological knowledge, methods, and experience to the design, implementation, testing, and documentation of software"—The Bureau of Labor Statistics—IEEE *Systems and software engineering - Vocabulary*
- "The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software"—IEEE *Standard Glossary of Software Engineering Terminology*

- "an engineering discipline that is concerned with all aspects of software production"— Ian Sommerville
- "the establishment and use of sound engineering principles in order to economically obtain software that is reliable and works efficiently on real machines"—Fritz Bauer
- “a systematic approach to design, construct, and maintain software more ‘citius, vilis, and bonus’ — Ademar Aguiar

The History of Software Engineering

Grady Booch

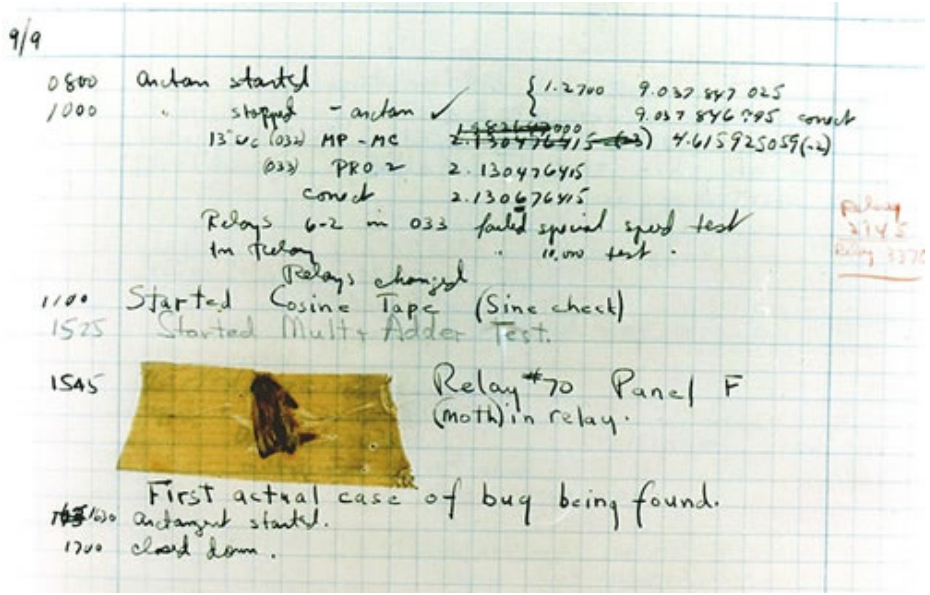
IBM Fellow & Chief Scientist for Software Engineering

Email: gbooch@us.ibm.com
Twitter: @grady_booch
Web: computingthehumanexperience.com

V1.0

The First Recorded Software Bug (1945)

- On September 9, 1945, U.S. Navy officer Grace Hopper found a **moth** between the relays on the Harvard Mark II computer she was working on.
- In those days computers filled (large) rooms and the warmth of the internal components attracted moths, flies and other flying creatures. Those creatures then **shortened circuits** and caused the computer to malfunction.



Grace Hopper

The First (?) Documented Software Development Process (1956)

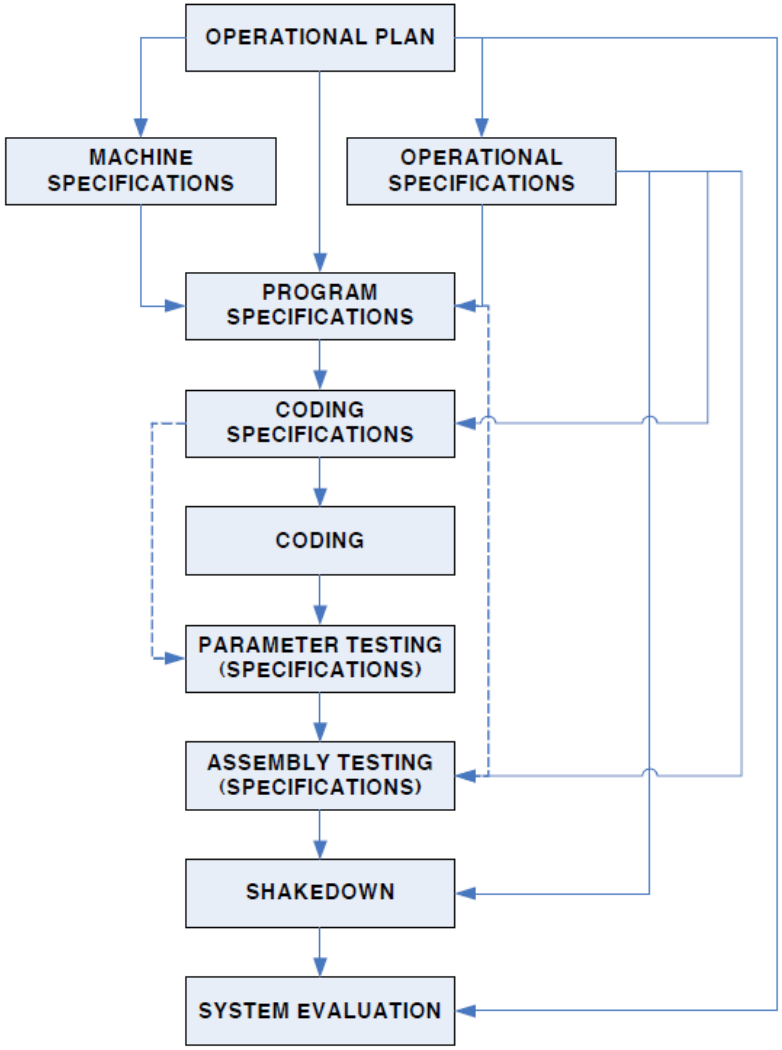


Figure 1. The SAGE Software Development Process (1956)

SAGE: Semi-automatic Ground Environment Air Defense System

Margaret Hamilton: SAGE & Apollo (1960's)

- **Margaret Hamilton** was Director of the Software Engineering Division of the MIT Instrumentation Laboratory, which developed on-board flight software for the Apollo space program.
- **"The Heart and Soul of Apollo: Doing it Right the First Time"**
- From 1961 to 1963, she worked on the Semi-Automatic Ground Environment (SAGE) Project at Lincoln Lab,
- On November 22, 2016, she was awarded the Presidential Medal of Freedom by U.S. President Barack Obama

Hamilton in 1969, standing next to listings of the software she and her MIT team produced for the Apollo project

Edgar Dijkstra: Go To Statement Considered Harmful (1968)

For a number of years I have been familiar with the observation that the quality of programmers is a decreasing function of the density of **go to** statements in the programs they produce. More recently I discovered why the use of the **go to** statement has such disastrous effects, and I became convinced that the **go to** statement should be abolished from all "higher level" programming languages (i.e. everything except, perhaps, plain machine code). At that time I did not attach too much importance to this discovery; I now submit my considerations for publication because in very recent discussions in which the subject turned up, I have been urged to do so.

In Communications of the ACM, 1968

Edgar Dijkstra, ACM Turing Award winner, 1972

First Software Engineering Conference (1968)

- Since the late 1950s and early 1960s, programmers debated what engineering might mean for software.
- Two conferences on software engineering sponsored by the NATO Science Committee in 1968 and 1969 as a response to the software crisis marked the official start of the profession of software engineering and gave the field its initial boost
- See also: http://en.wikipedia.org/wiki/History_of_software_engineering

SOFTWARE ENGINEERING

Report on a conference sponsored by the

NATO SCIENCE COMMITTEE

Garmisch, Germany, 7th to 11th October 1968

Chairman: Professor Dr. F. L. Bauer

Co-chairmen: Professor L. Bolliet, Dr. H. J. Helms

Editors: Peter Naur and Brian Randell

Waterfall

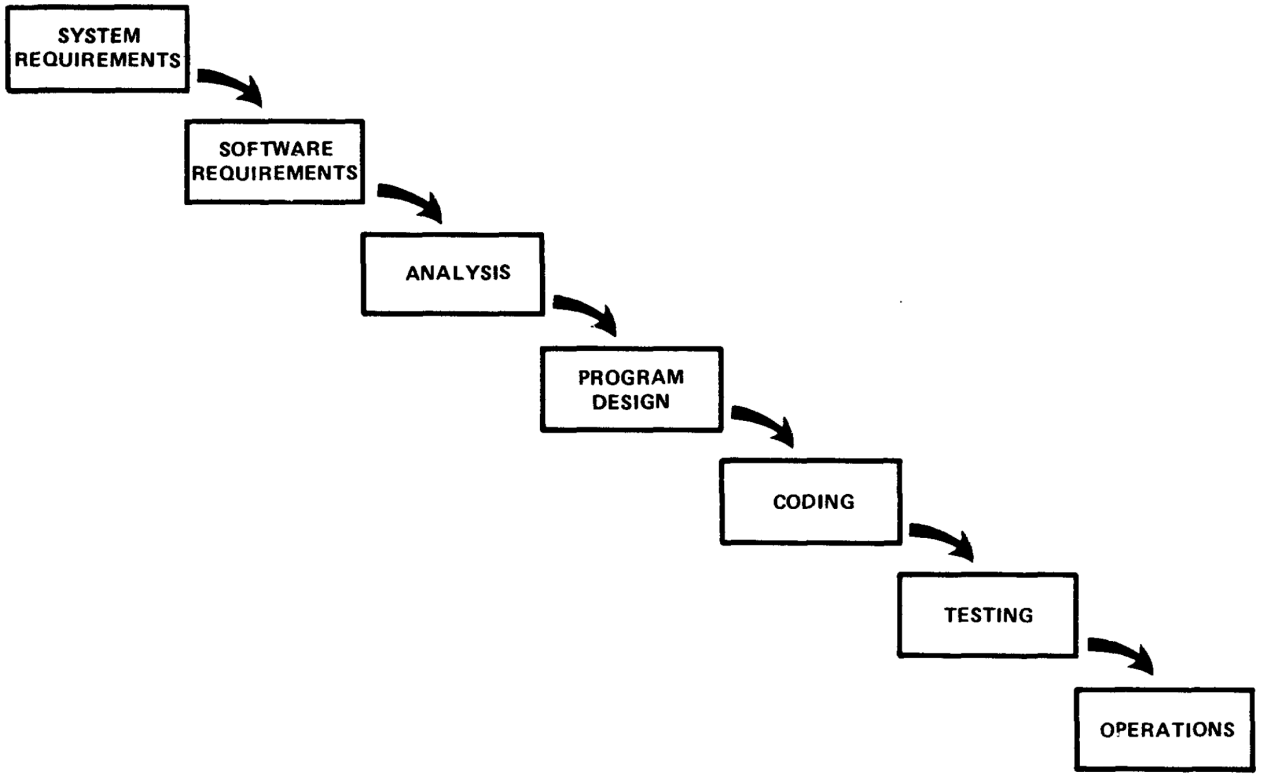


Figure 2. Implementation steps to develop a large computer program for delivery to a customer.

Iterative

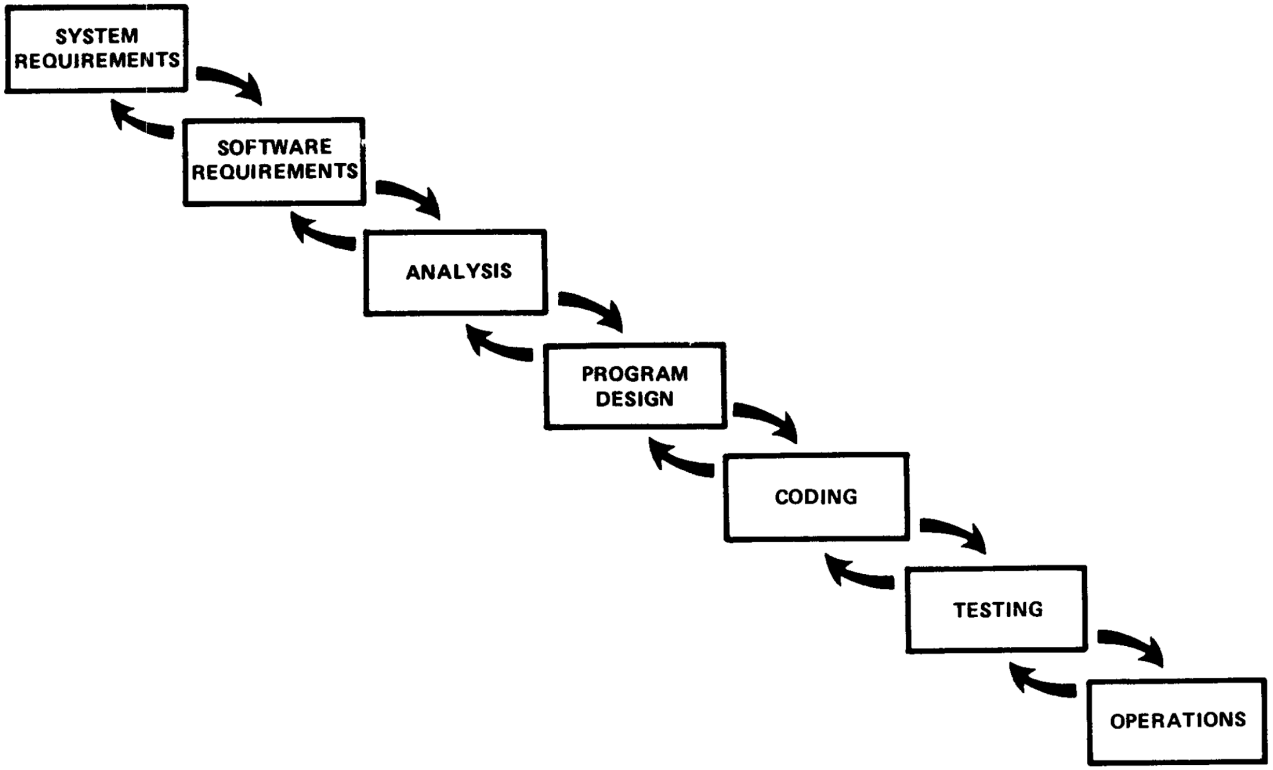


Figure 3. Hopefully, the iterative interaction between the various phases is confined to successive steps.

Spiral

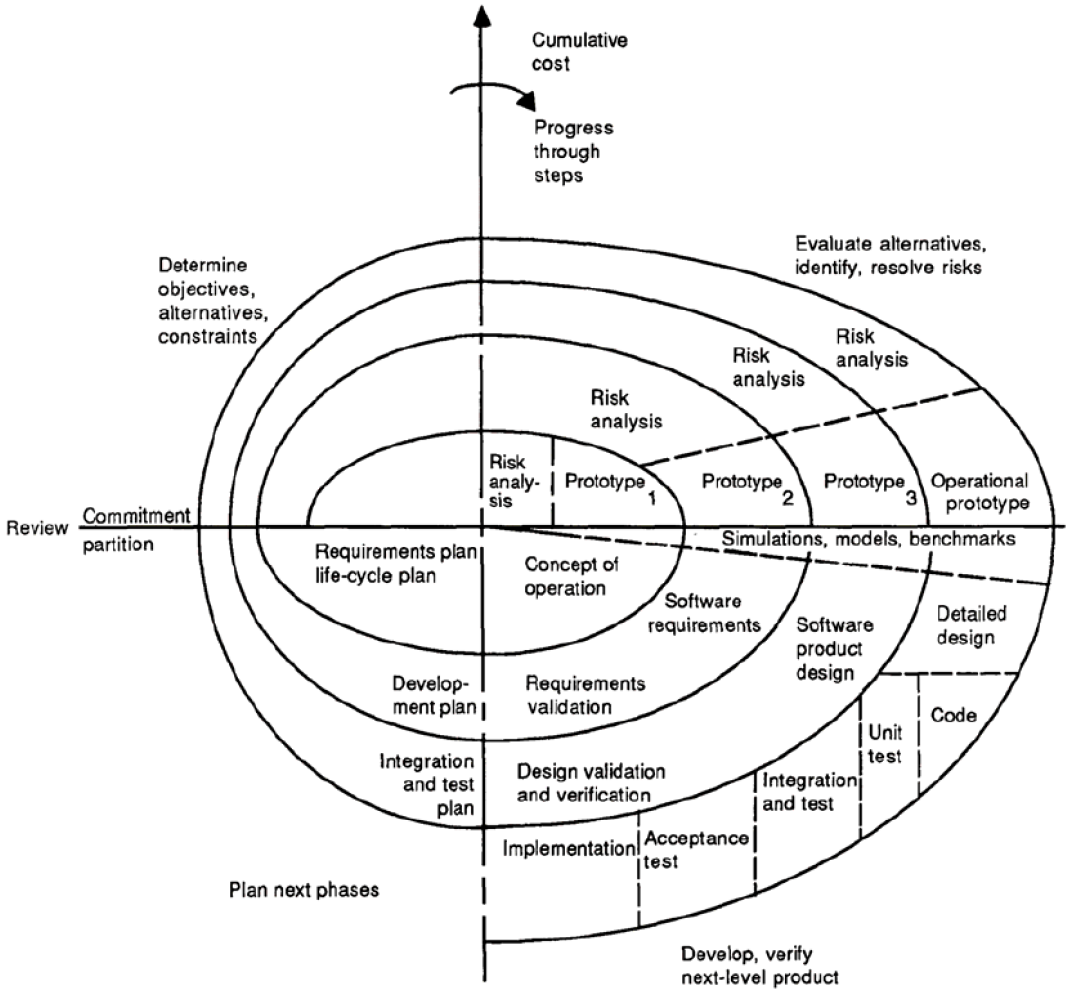
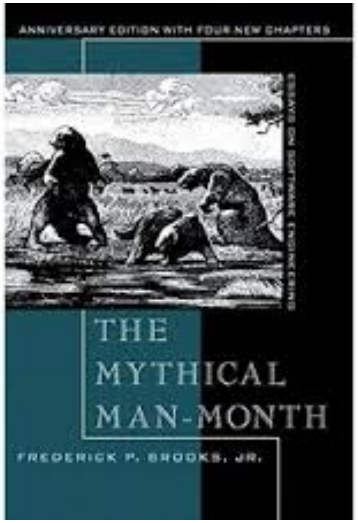


Figure 2. Spiral model of the software process.

Fred Brooks: The Mythical Man-Month (1975)



for there is a probability distribution for the delay that will be encountered, and "no delay" has a finite probability. A large programming effort, however, consists of many tasks, some chained end-to-end. The probability that each will go well becomes vanishingly small.

The Man-Month

The second fallacious thought mode is expressed in the very unit of effort used in estimating and scheduling: the man-month. Cost does indeed vary as the product of the number of men and the number of months. Progress does not. *Hence the man-month as a unit for measuring the size of a job is a dangerous and deceptive myth.* It implies that men and months are interchangeable.

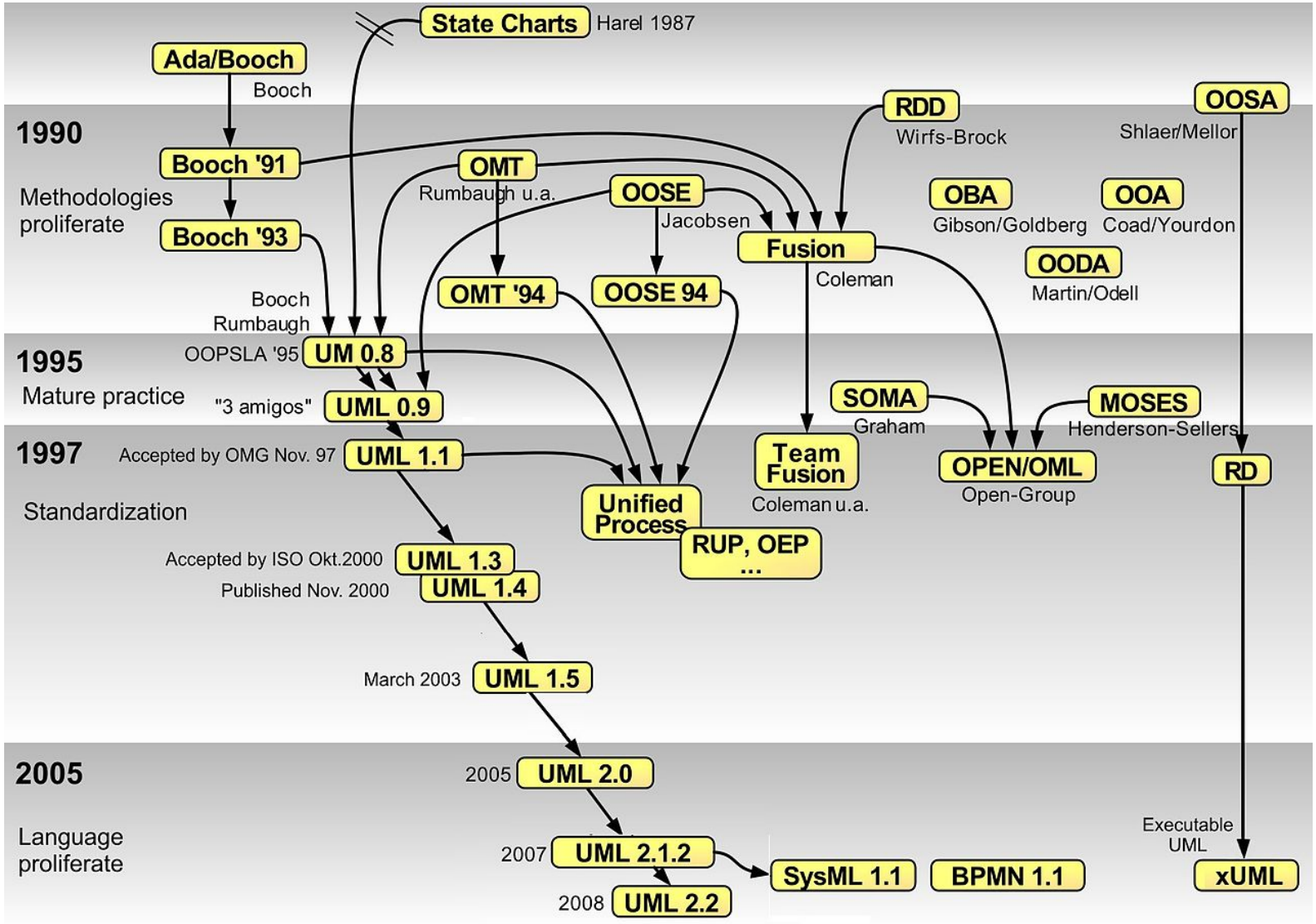
Men and months are interchangeable commodities only when a task can be partitioned among many workers *with no communication among them* (Fig. 2.1). This is true of reaping wheat or picking cotton; it is not even approximately true of systems programming.

Fred Brooks: No Silver Bullet (1986)

- “No Silver Bullet – Essence and Accident in Software Engineering”, Fred Brooks, 1986
- “there is no single development, in either technology or management technique, which by itself promises even one order of magnitude [tenfold] improvement within a decade in productivity, in reliability, in simplicity."
- "we cannot expect ever to see two-fold gains every two years" in software development, as there is in hardware development (Moore's law).

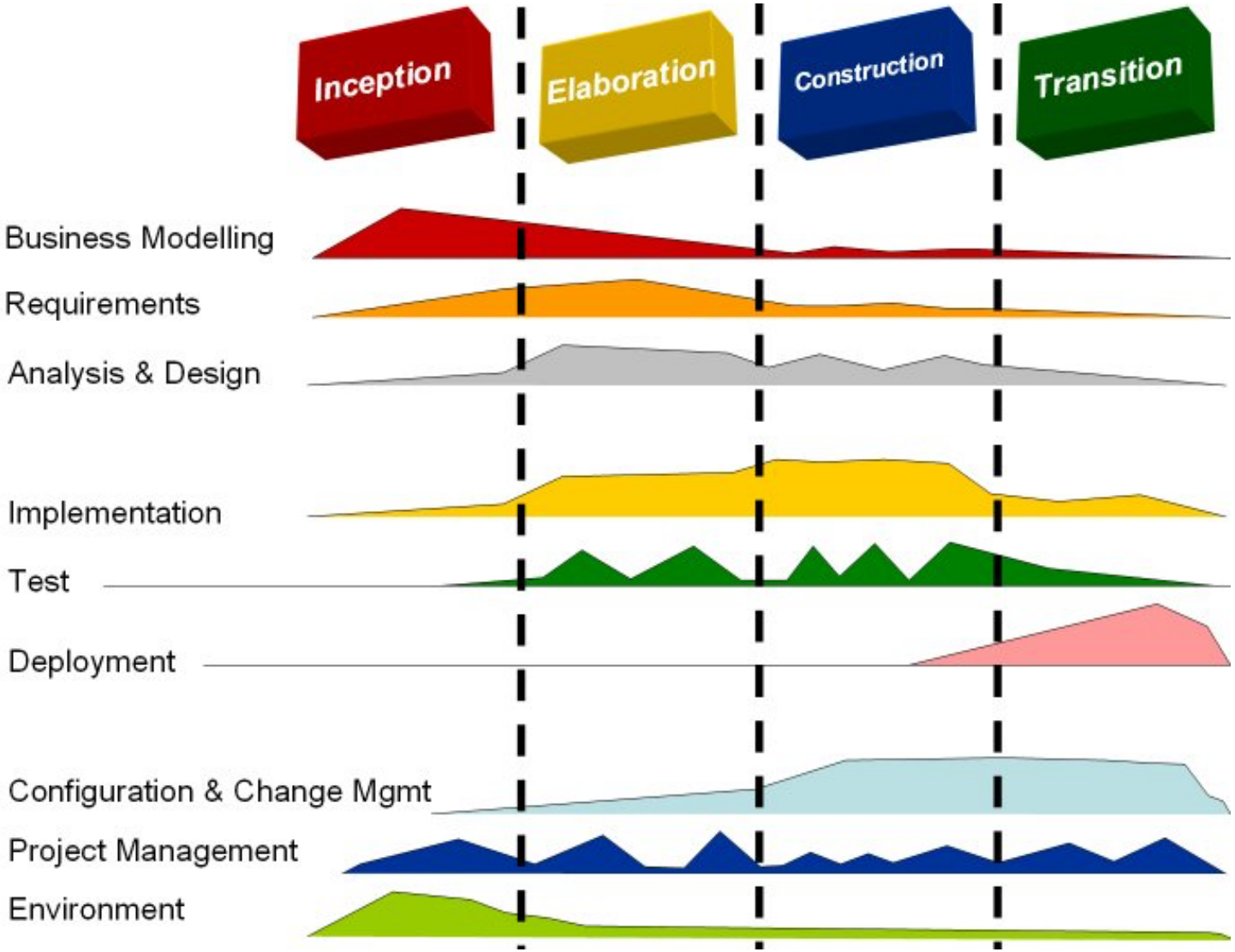
Fred Brooks,
ACM Turing Award winner,
1999

The Last Method War (1990-1994)

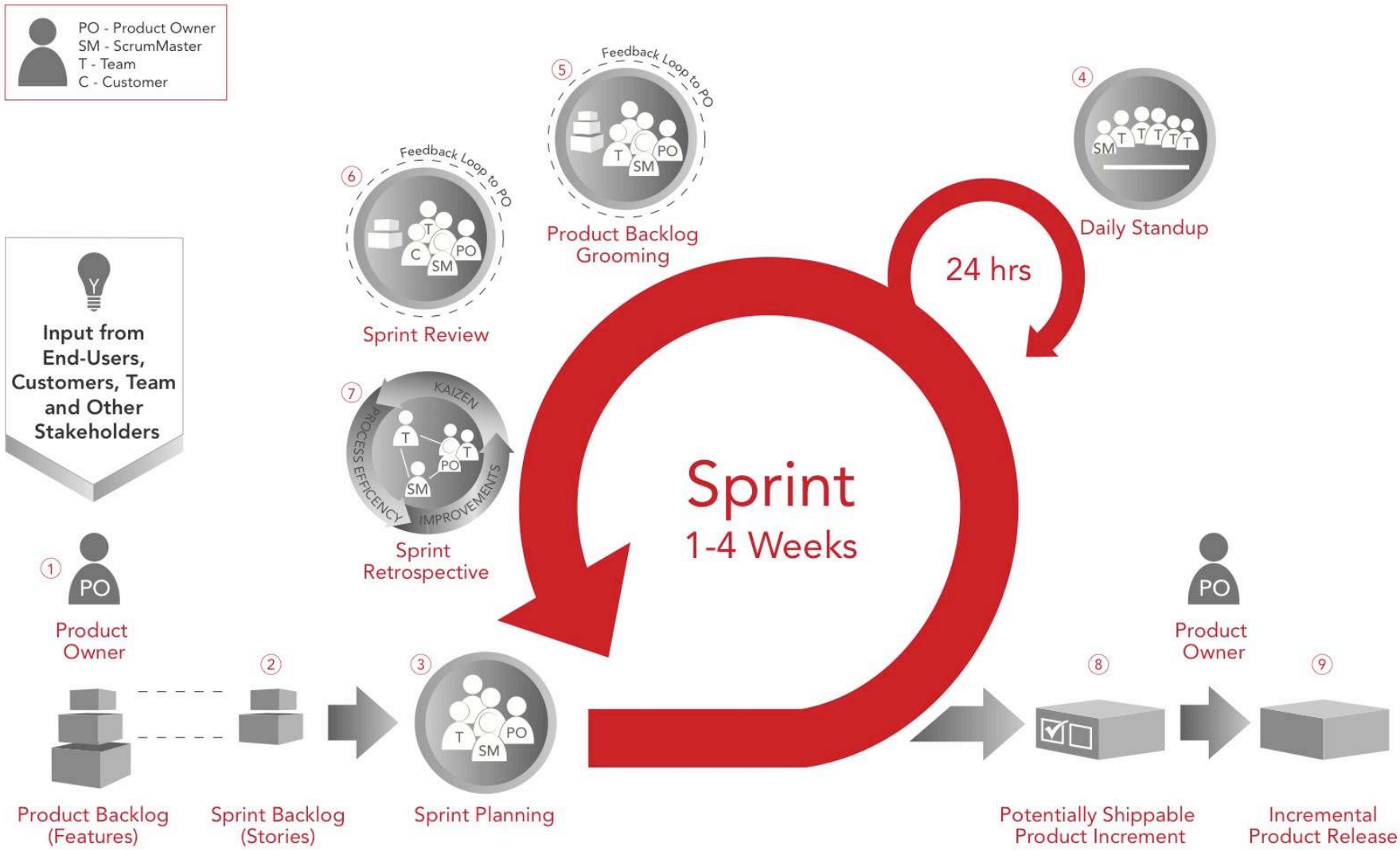


https://en.wikipedia.org/wiki/Unified_Modeling_Language

Rational Unified Process

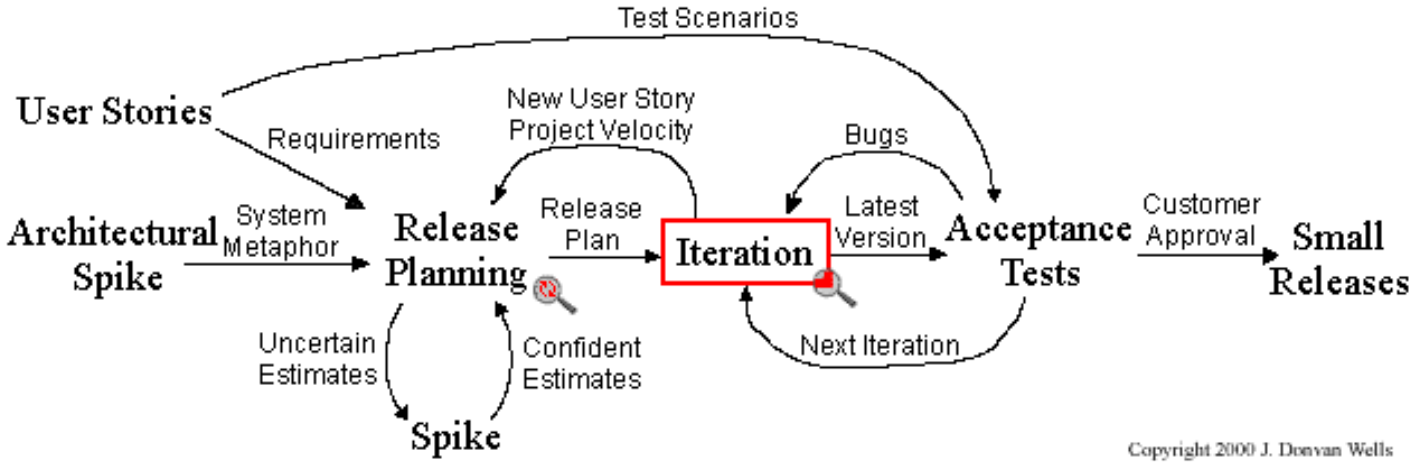


How Scrum Works

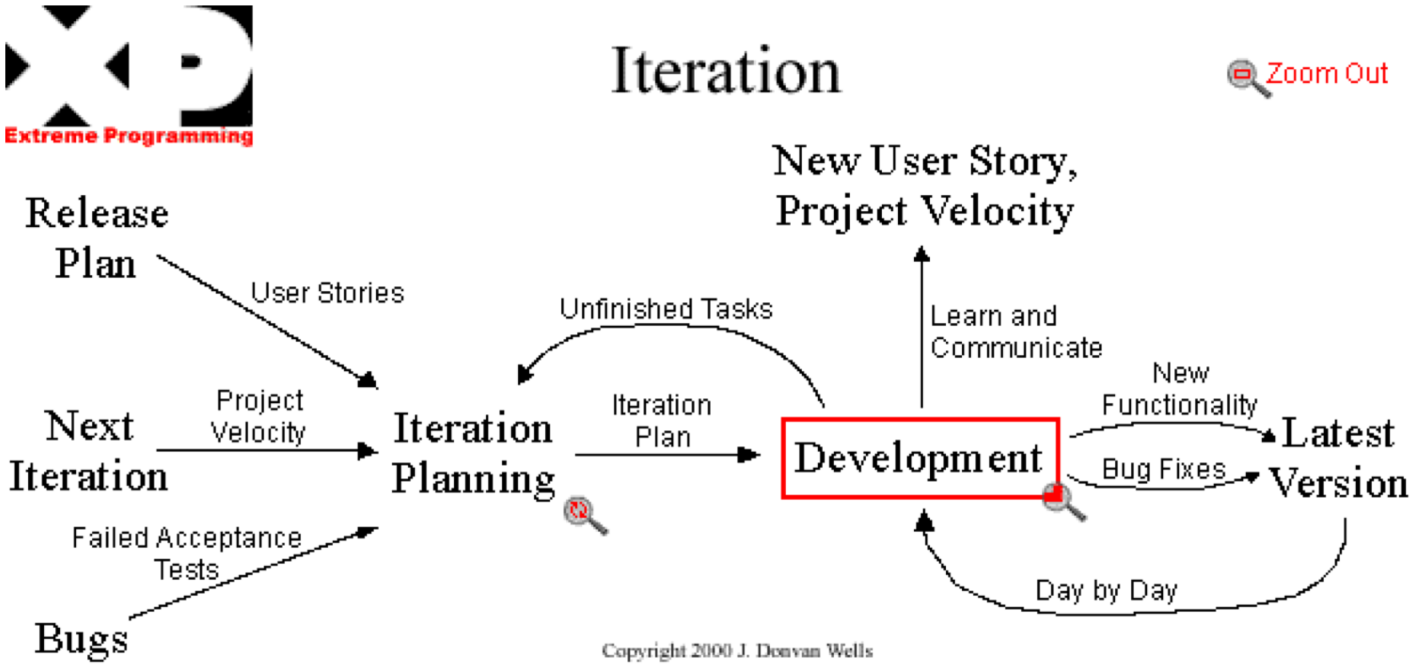


scruminc.

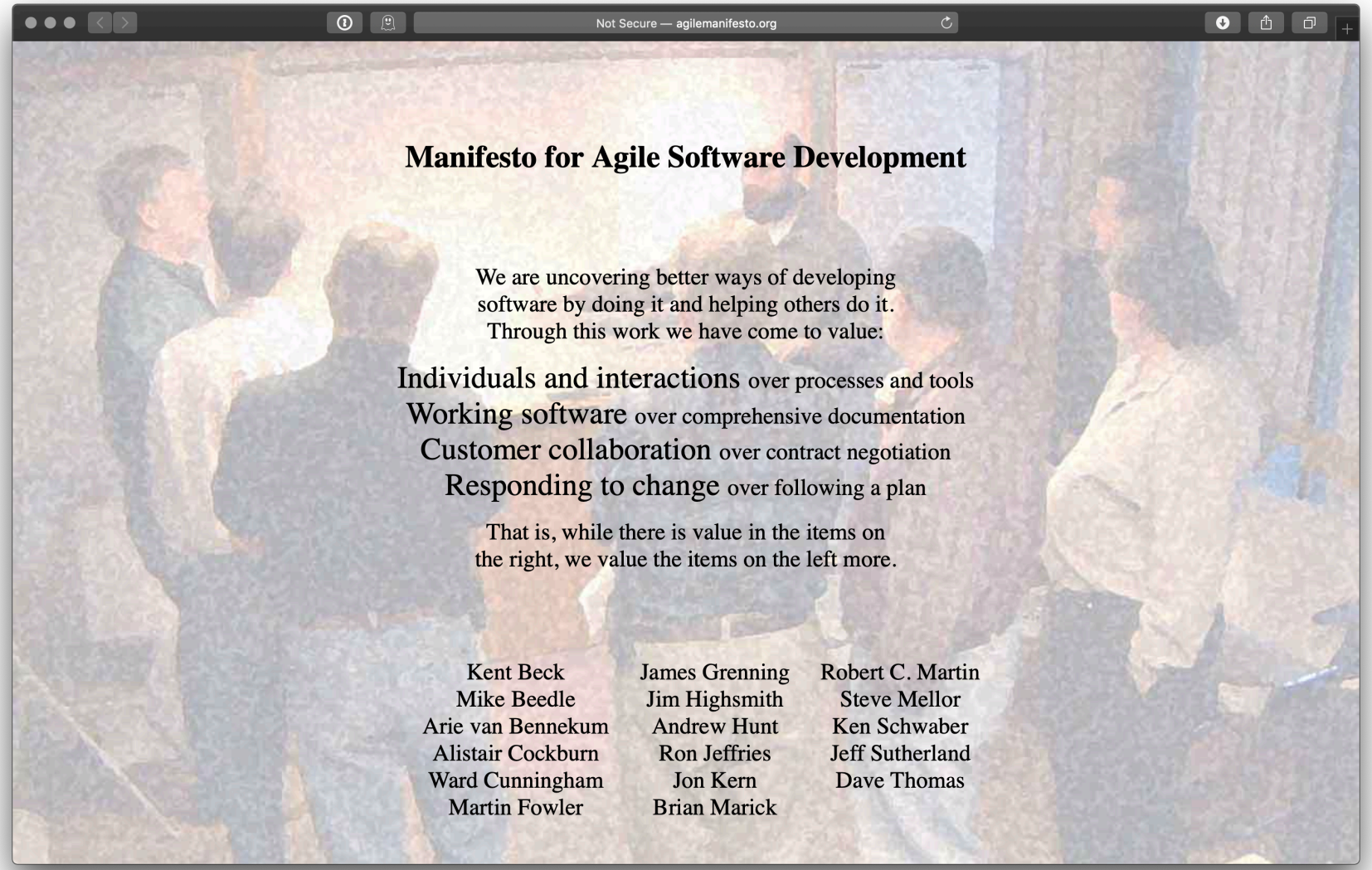
Extreme Programming Project



Iteration

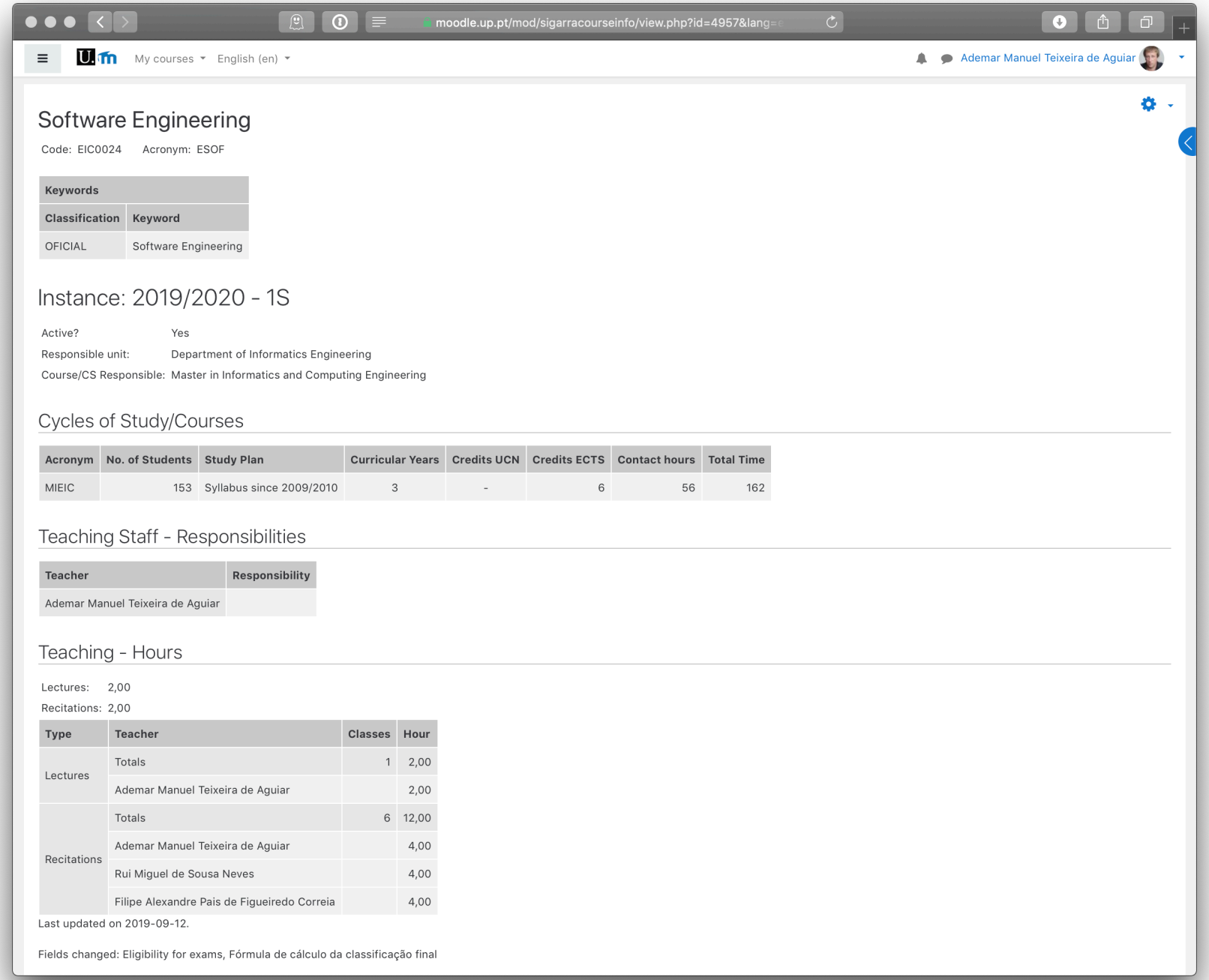


Agile Manifesto (2001)



http://agilemanifesto.org/

Course structure and organization



Software Engineering

Code: EIC0024 Acronym: ESOF

Keywords	
Classification	**Keyword**
OFICIAL	Software Engineering

Instance: 2019/2020 - 1S

Active? Yes
Responsible unit: Department of Informatics Engineering
Course/CS Responsible: Master in Informatics and Computing Engineering

Cycles of Study/Courses

Acronym	No. of Students	Study Plan	Curricular Years	Credits UCN	Credits ECTS	Contact hours	Total Time
MIEIC	153	Syllabus since 2009/2010	3	-	6	56	162

Teaching Staff - Responsibilities

Teacher	Responsibility
Ademar Manuel Teixeira de Aguiar	

Teaching - Hours

Lectures: 2,00
Recitations: 2,00

Type	Teacher	Classes	Hour
Lectures	Totals	1	2,00
	Ademar Manuel Teixeira de Aguiar		2,00
Recitations	Totals	6	12,00
	Ademar Manuel Teixeira de Aguiar		4,00
	Rui Miguel de Sousa Neves		4,00
	Filipe Alexandre Pais de Figueiredo Correia		4,00

Last updated on 2019-09-12.

Fields changed: Eligibility for exams, Fórmula de cálculo da classificação final

Course strategy

- Introductory (1st cycle) course on software engineering
- Theoretical and practical nature
  - Key concepts overviewed in lectures
  - Key activities, practices and tools used in practical classes, applied to a group project.
  - Detailed in subsequent labs: LBAW, LDSO, LGPR
- Breadth-first
  - In-depth elective courses on specific knowledge areas can be found in the 2nd cycle, for those that want to specialize in software engineering
    - Requirements Engineering (ERSS)
    - Architecture of Software Systems (ASSO)
    - Software Quality and Testing (TQSO)
    - Agile Methods for Software Development (MADS)

Course bibliography

Mandatory

- Software Engineering (9th edition), Ian Sommerville, Addison-Wesley, 2011, ISBN: 9780137035151
- Ivar Jacobson, Bud Lawson, Paul mcMahon, Michael Goedicke; Software Engineering Essentialized (http://semat.org/web/book)

Complementary

- Russ Miles & Kim Hamilton; Learning UML 2.0, O'Reilly, 2006. ISBN: 0-596-00982-8
- Silva, Alberto Manuel Rodrigues da; UML, metodologias e ferramentas CASE (2ª edição). ISBN: 989-615-009-5
- Humphrey, Watts S; A discipline for Software engineering. ISBN: 0-201-54610-8

Guide to the Software Engineering Body of Knowledge

Version 3.0

Editors

Pierre Bourque, École de technologie supérieure (ÉTS)
Richard E. (Dick) Fairley, Software and Systems Engineering Associates (S2EA)

Software engineering knowledge areas

Table I.1. The 15 SWEBOK KAs
Software Requirements
Software Design
Software Construction
Software Testing
Software Maintenance
Software Configuration Management
Software Engineering Management
Software Engineering Process
Software Engineering Models and Methods
Software Quality
Software Engineering Professional Practice
Software Engineering Economics
Computing Foundations
Mathematical Foundations
Engineering Foundations

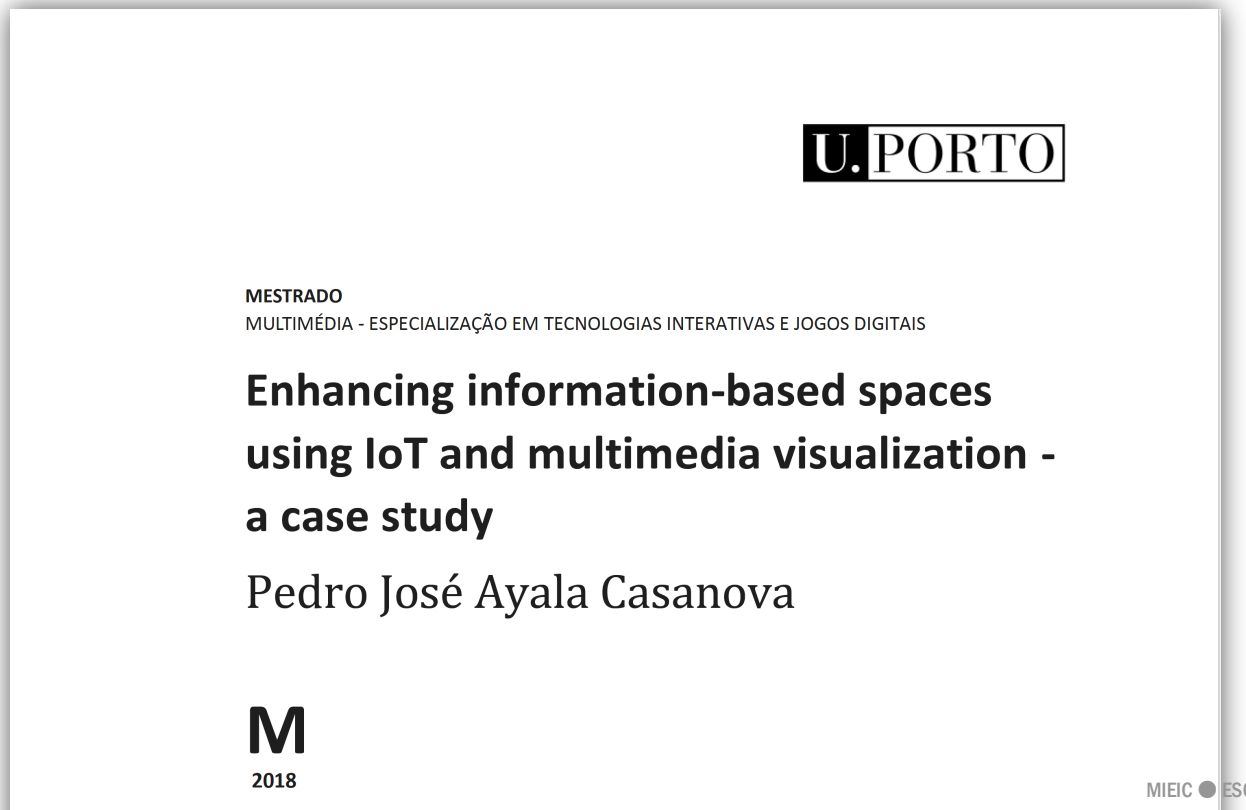
Guide to the Software Engineering Body of Knowledge Version 3.0 SWEBOK®, IEEE Computer Society

https://www.computer.org/web/swebok/v3

Table I.2. Related Disciplines
Computer Engineering
Computer Science
General Management
Mathematics
Project Management
Quality Management
Systems Engineering

Project

- **Enhancing information-based spaces using IoT and multimedia visualization**, based on Pedro Ayala thesis.
- To be an open source project by FEUP/DEI.
- Groups of 4 students



U.PORTO

MESTRADO
MULTIMÉDIA - ESPECIALIZAÇÃO EM TECNOLOGIAS INTERATIVAS E JOGOS DIGITAIS

Enhancing information-based spaces using IoT and multimedia visualization - a case study

Pedro José Ayala Casanova

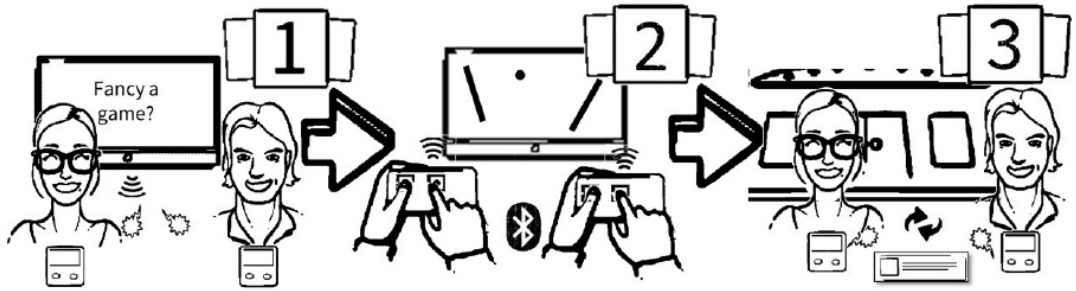
M

2018

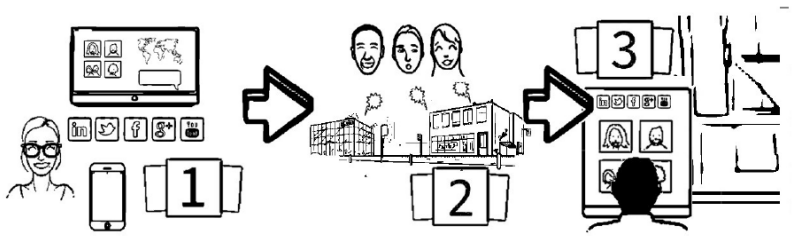
Project Vision

- Interactive Agenda

- Interactive Badges



- Social Media Visualizer



Project Technologies

- Languages
  - Python, Java, Javascript, whatever you like
- Models
  - UML
- Apps
  - Mobile, web, ...
- Devices
  - Microbit. RaspberryPi, Arduino, Drones?
- Tools
  - Code editors, UML editors, github, trello

Important dates

- 11-15.Nov.2019 – intermediate delivery of TP
- 16-20.Dec.2019 – final delivery of TP
- dd.Jan.2020 – Exam @ moodle

Grades

- Final = 65% TP + 35% Exam
- Minimum of 40% in each evaluation component.
- Final grades higher than or equal to 18 pts may require an oral examination addressing all aspects of the course.

ademar.aguiar@fe.up.pt