RUP

vs. Agile/XP

Yegor Bugayenko

Lecture #3 out of 16
80 minutes

The slidedeck was presented by the author in this YouTube Video

- Software Development Methodologies
- Rational Unified Process (RUP) and Co.
- Agile, Kanban, Scrum, XP
- Qualities of Good Design
- Books, Venues, Call-to-Action

Chapter #1:

Software Development Methodologies

Methodologies:

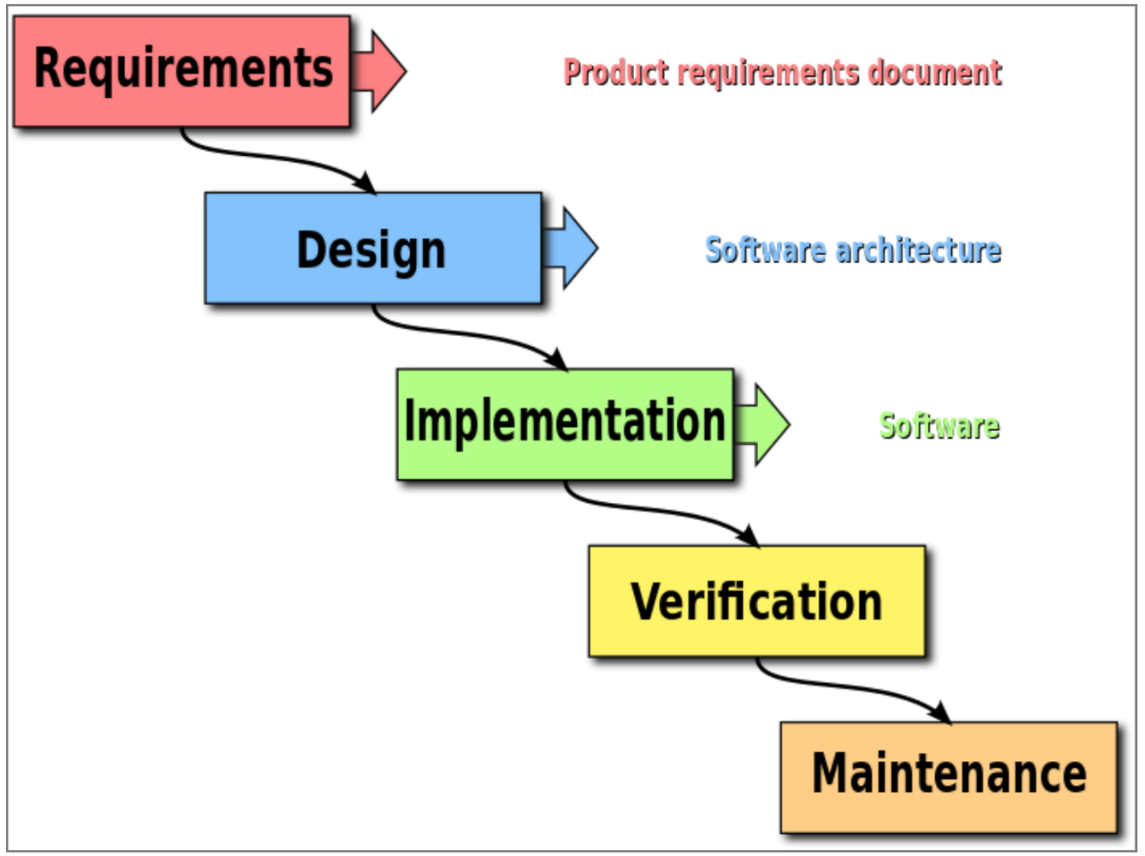
Waterfall / SDLC

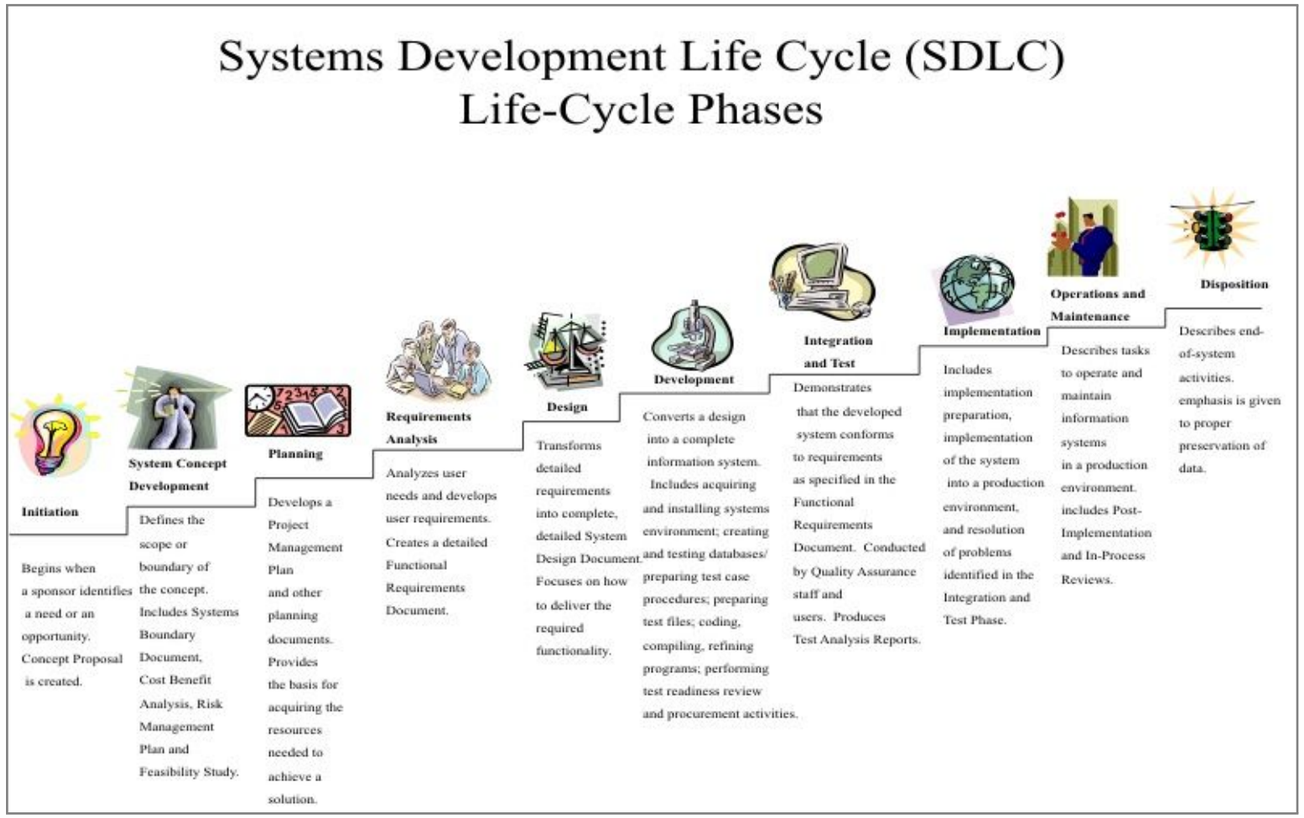
Rational Unified Process (RUP)

Microsoft Solutions Framework (MSF)

Information Technology Infrastructure Library (ITIL)

Agile & Co.

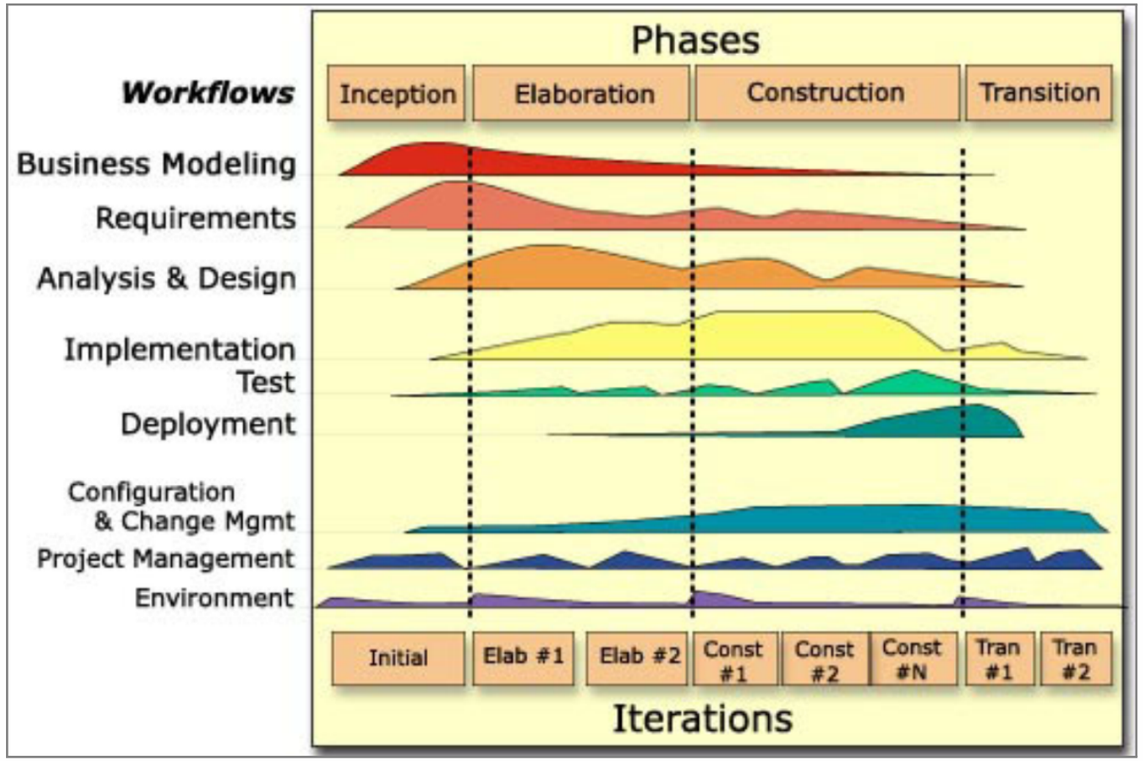

Requirements
Product requirements document
Design
Software architecture
Implementation
Software
Verification
Maintenance


Systems Development Life Cycle (SDLC)
Life-Cycle Phases
Initiation
Begins when a sponsor identifies a need or an opportunity. Concept Proposal is created.
System Concept Development
Defines the scope or boundary of the concept. Includes Systems Boundary Document, Cost Benefit Analysis, Risk Management Plan and Feasibility Study.
Planning
Develops a Project Management Plan and other planning documents. Provides the basis for acquiring the resources needed to achieve a solution.
Requirements Analysis
Analyzes user needs and develops user requirements. Creates a detailed Functional Requirements Document.
Design
Transforms detailed requirements into complete, detailed System Design Document. Focuses on how to deliver the required functionality.
Development
Converts a design into a complete information system. Includes acquiring and installing systems environment; creating and testing databases/ preparing test case procedures; preparing test files; coding, compiling, refining programs; performing test readiness review and procurement activities.
Integration and Test
Demonstrates that the developed system conforms to requirements as specified in the Functional Requirements Document. Conducted by Quality Assurance staff and users. Produces Test Analysis Reports.
Implementation
Includes implementation preparation, implementation of the system into a production environment, and resolution of problems identified in the Integration and Test Phase.
Operations and Maintenance
Describes tasks to operate and maintain information systems in a production environment. includes Post-Implementation and In-Process Reviews.
Disposition
Describes end-of-system activities. emphasis is given to proper preservation of data.

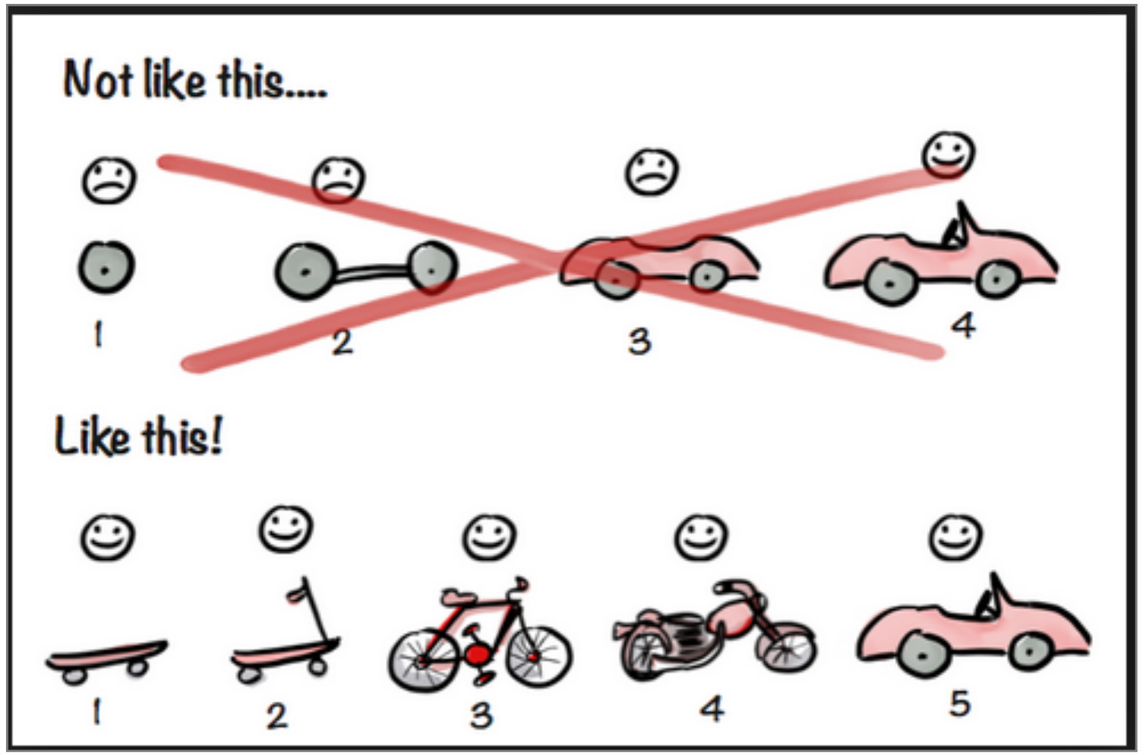
Chapter #2:

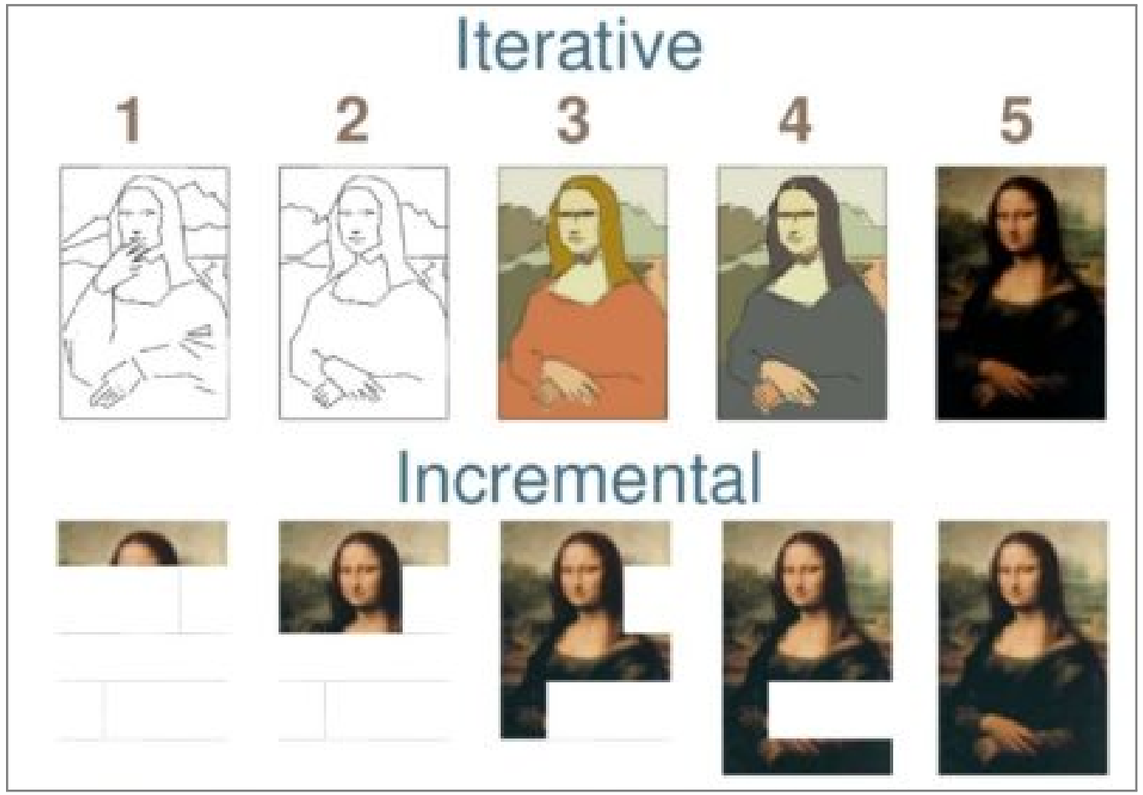
Rational Unified Process (RUP) and Co.

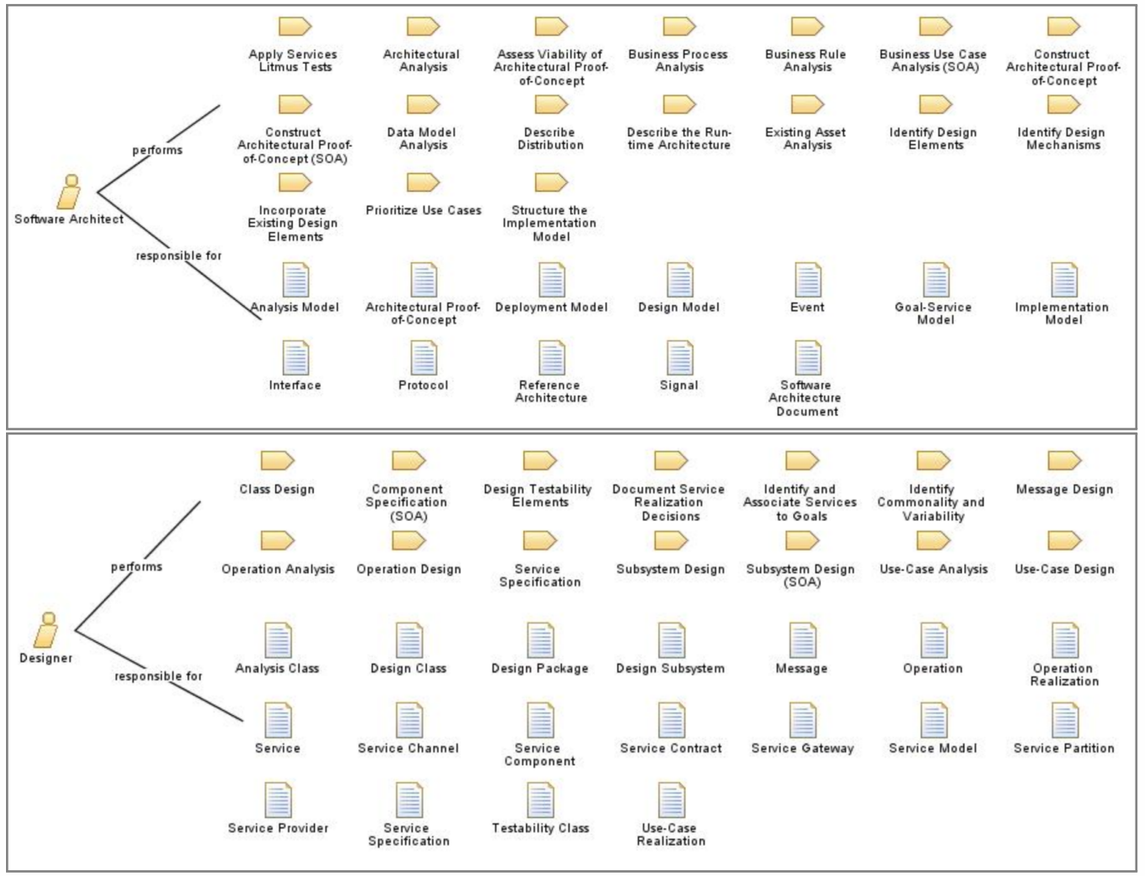
Iterative & Incremental



Iterative:




Iterative
1
2
3
4
5
Incremental


Software Architect
performs
responsible for
Apply Services Litmus Tests
Architectural Analysis
Assess Viability of Architectural Proof-of-Concept
Business Process Analysis
Business Rule Analysis
Business Use Case Analysis (SOA)
Construct Architectural Proof-of-Concept
Construct Architectural Proof-of-Concept (SOA)
Data Model Analysis
Describe Distribution
Describe the Run-time Architecture
Existing Asset Analysis
Identify Design Elements
Identify Design Mechanisms
Incorporate Existing Design Elements
Prioritize Use Cases
Structure the Implementation Model
Analysis Model
Architectural Proof-of-Concept
Deployment Model
Design Model
Event
Goal-Service Model
Implementation Model
Interface
Protocol
Reference Architecture
Signal
Software Architecture Document
Designer
performs
responsible for
Class Design
Component Specification (SOA)
Design Testability Elements
Document Service Realization Decisions
Identify and Associate Services to Goals
Identify Commonality and Variability
Message Design
Operation Analysis
Operation Design
Service Specification
Subsystem Design
Subsystem Design (SOA)
Use-Case Analysis
Use-Case Design
Analysis Class
Design Class
Design Package
Design Subsystem
Message
Operation
Operation Realization
Service
Service Channel
Service Component
Service Contract
Service Gateway
Service Model
Service Partition
Service Provider
Service Specification
Testability Class
Use-Case Realization

Professional Certification Program from IBM
www.ibm.com/certify

IBM

In recognition of the commitment to achieve professional excellence,
this certifies that

Yegor Bugaenko

has successfully completed the program requirements as an

IBM Certified Specialist
Rational Unified Process
v2003

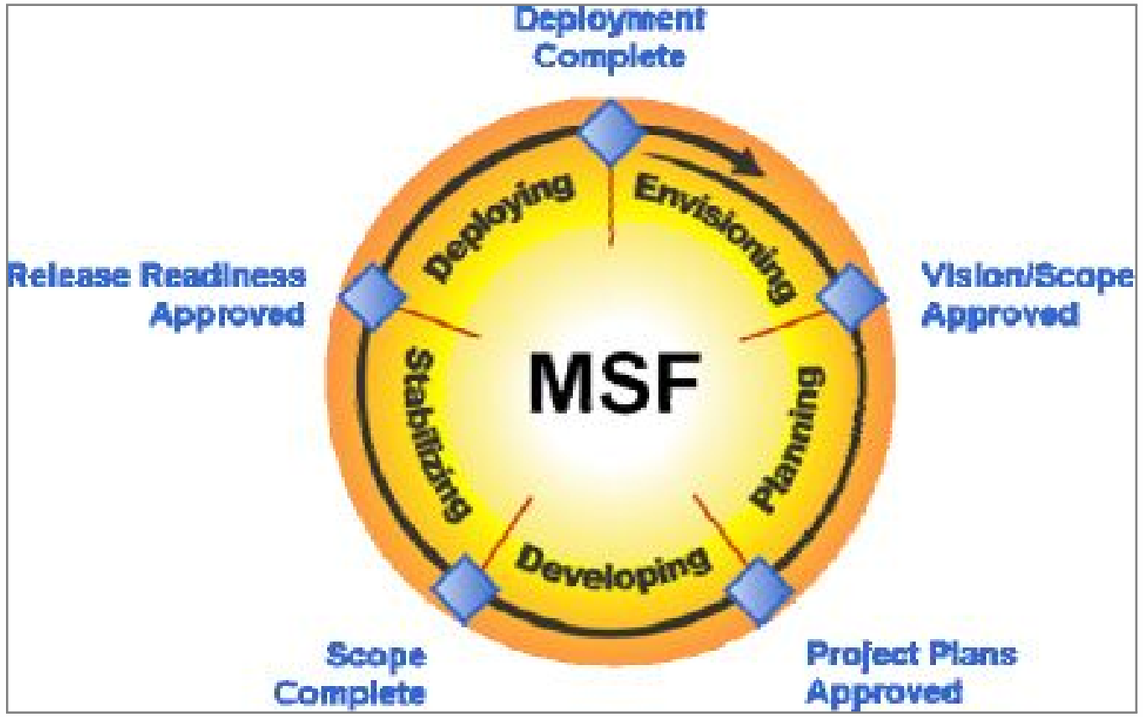
Awarded on December 21, 2006
IBM congratulates you on your achievement.

Certified for IBM Rational. software

IBM Software Group
Senior Vice President and Group Executive
Steve Mills

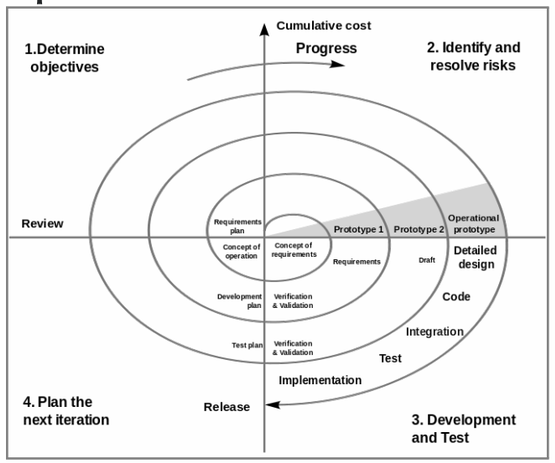
IBM Software Group
Rational Software
General Manager
D. Sabbah

Microsoft Solutions Framework (MSF)

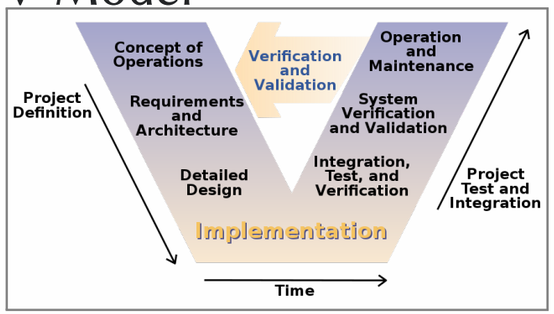


Information Technology Infrastructure Library (ITIL)

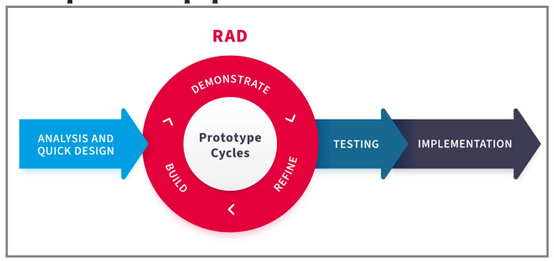
Spiral



V-Model

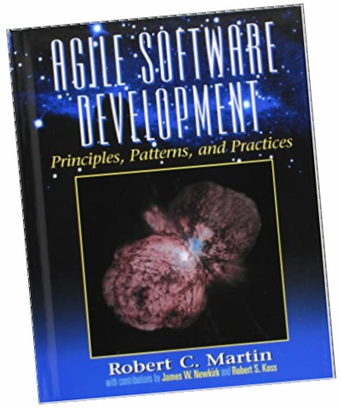


Rapid Application Development



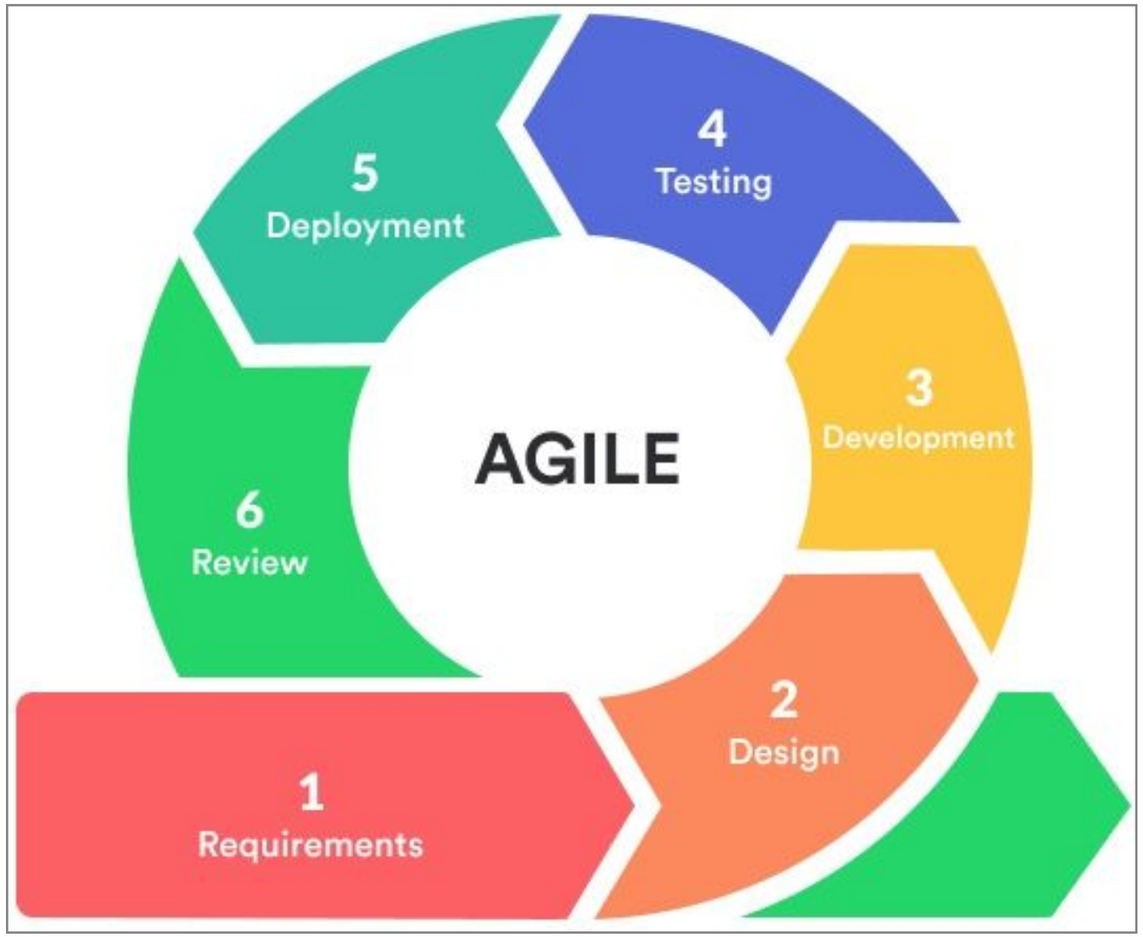
Chapter #3:

Agile, Kanban, Scrum, XP

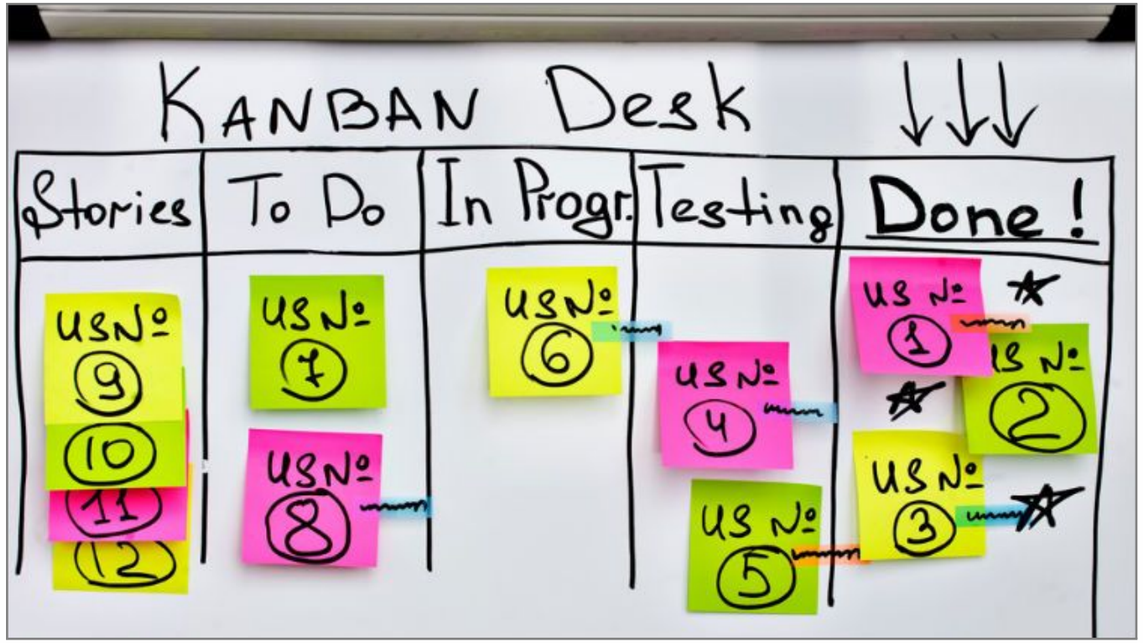


"In an agile team, the big picture evolves along with the software. With each iteration, the team improves the design of the system so that it is as good as it can be for the system as it is now. The team does not spend very much time looking ahead to future requirements and needs."

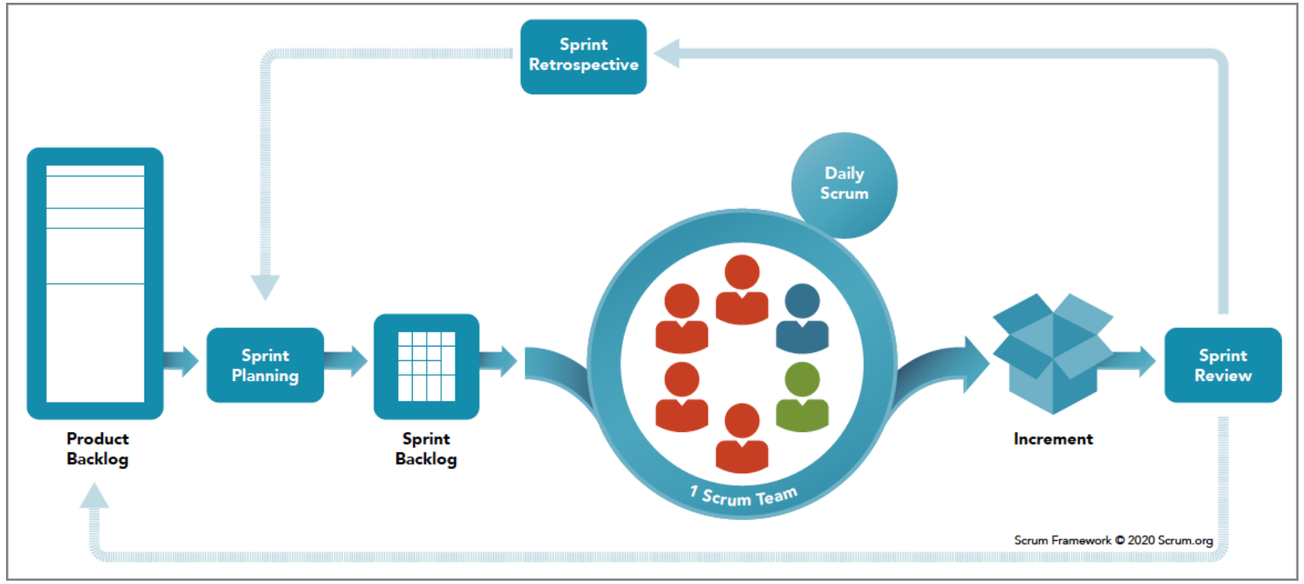
— Robert C. Martin. *Agile Software Development, Principles, Patterns, and Practices.* Prentice Hall, 2002. doi:10.5555/515230

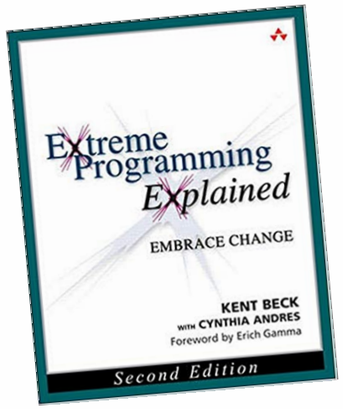

AGILE
1
Requirements
2
Design
3
Development
4
Testing
5
Deployment
6
Review

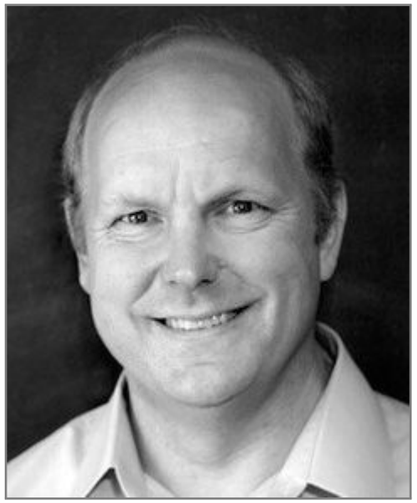
Kanban


KANBAN Desk
Stories
To Do
In Progr.
Testing
Done !
US №
9
10
11
12
7
8
6
4
5
1
2
3

Scrum Framework



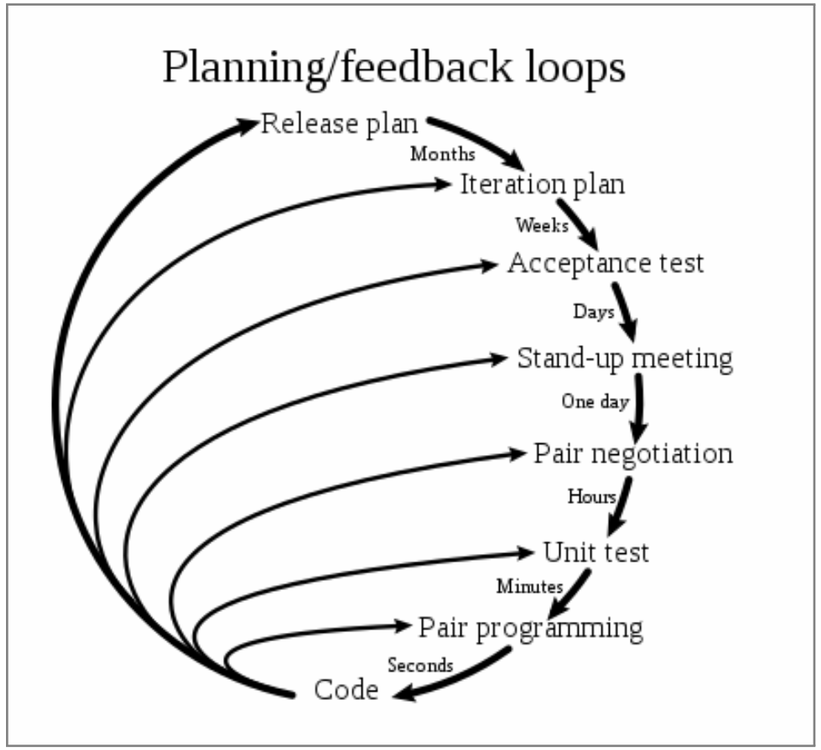




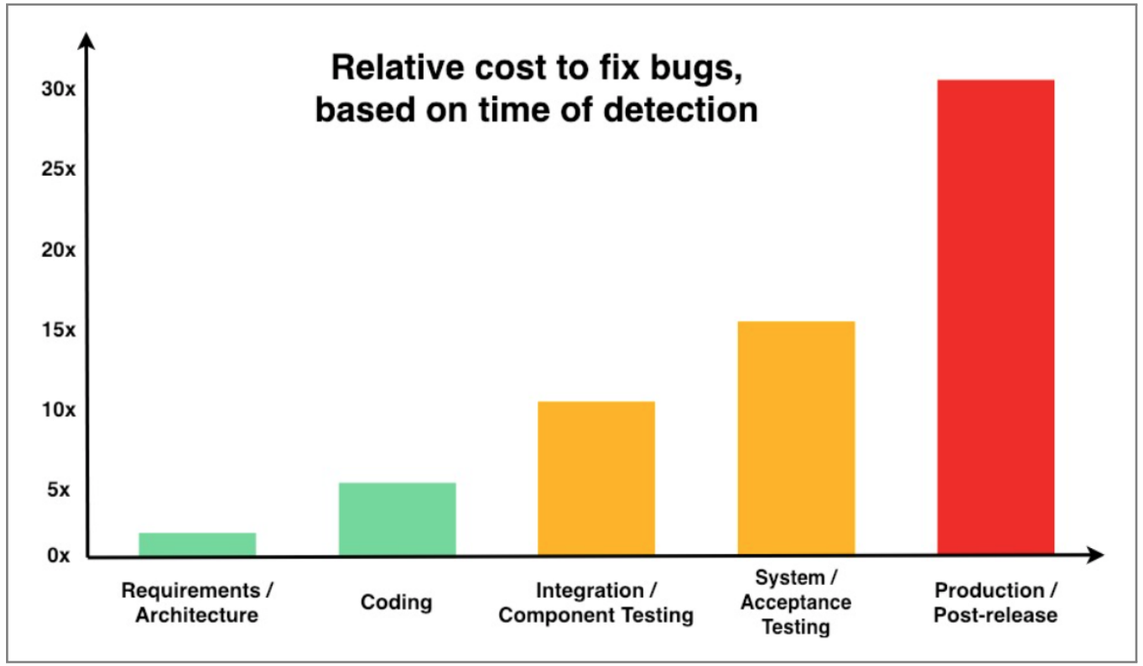
"We will continually refine the design of the system, starting from a very simple beginning. We will remove any flexibility that doesn't prove useful."

— Kent Beck. *Extreme Programming Explained: Embrace Change.* Addison-Wesley, 2000. doi:10.5555/318762

eXtreme Programming (XP)



Cost of Bugs


Relative cost to fix bugs,
based on time of detection
30x
25x
20x
15x
10x
5x
0x
Requirements / Architecture
Coding
Integration / Component Testing
System / Acceptance Testing
Production / Post-release

Chapter #4:

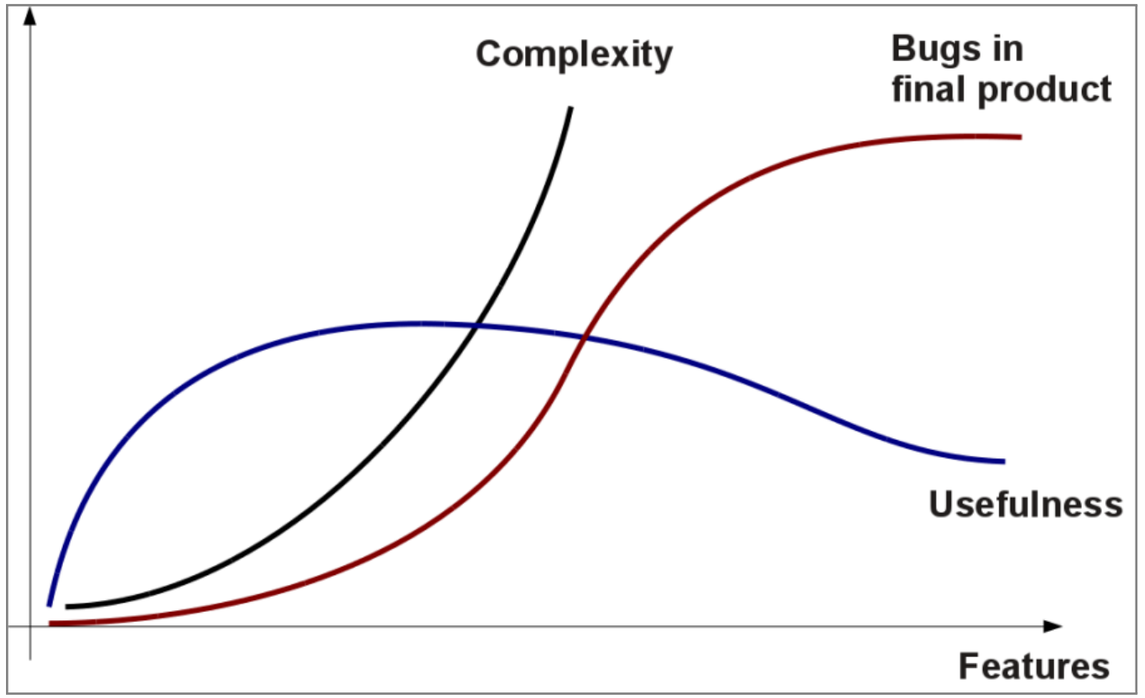
Qualities of Good Design

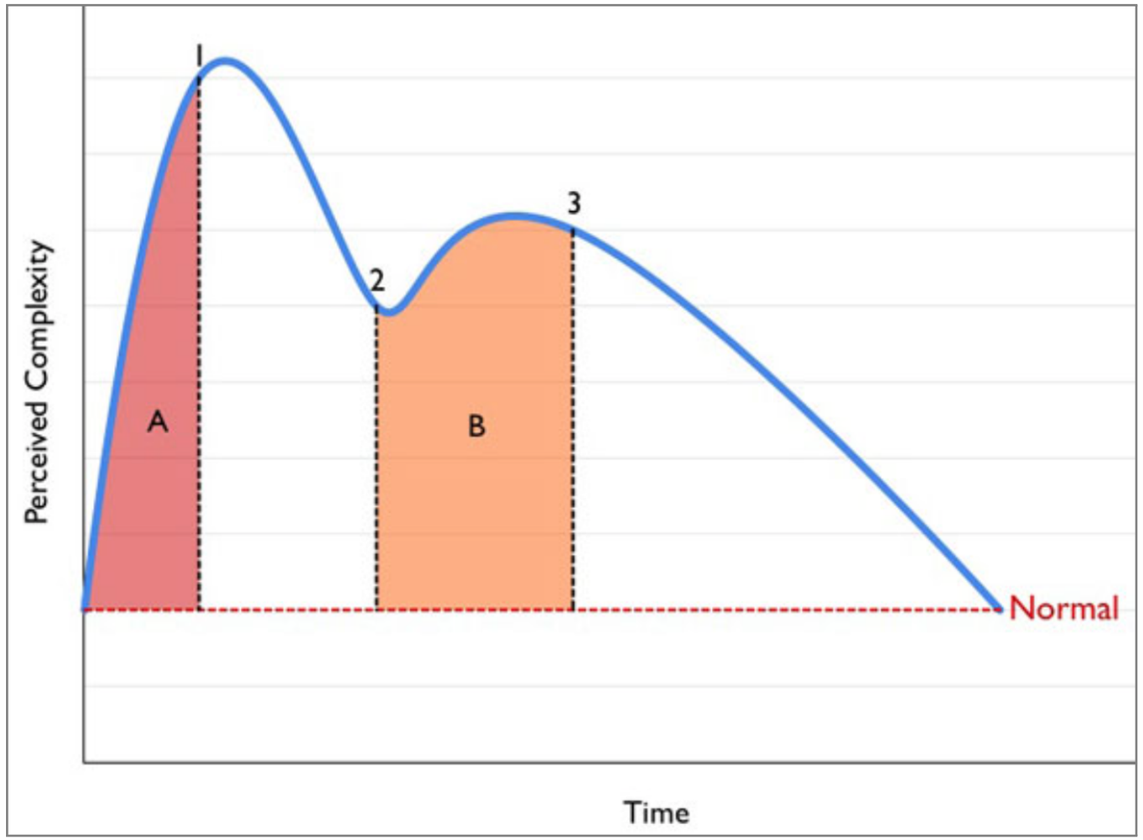
It Must Be Simple

"We should make the smallest possible investment in the design before getting payback for it" – Kent Beck

"A design contains needless complexity when it contains elements that aren't currently useful. This frequently happens when developers anticipate changes to the requirements, and put facilities in the software to deal with those potential changes." – Robert Martin

"Our failure to master the complexity of software results in projects that are late, over budget, and deficient in their stated requirements. We often call this condition the software crisis, but frankly, a malady that has carried on this long must be called normal." – Grady Booch


Complexity
Bugs in
final product
Usefulness
Features


Perceived Complexity
Time
1
2
3
A
B
Normal

“The main virtue of an architect is the ability to reduce complexity. Thus, a good architect would never be proud of a complex diagram. Instead, they would be proud of a simple and easy-to-understand drawing with a few rectangles that perfectly explain an entire multi-tier application. That is what is really difficult to do. That’s where a true architectural mind shines.”
— me.

It Must Not Repeat Itself

"When there is redundant code in the system, the job of changing the system can become arduous. Bugs found in such a repeating unit have to be fixed in every repetition. However, since each repetition is slightly different from every other, the fix is not always the same." – Robert Martin

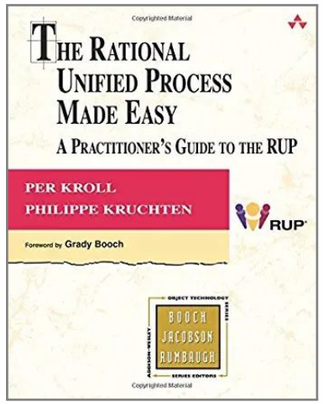
It Must Be Modular

"A design is immobile when it contains parts that could be useful in other systems, but the effort and risk involved with separating those parts from the original system are too great." – Robert Martin

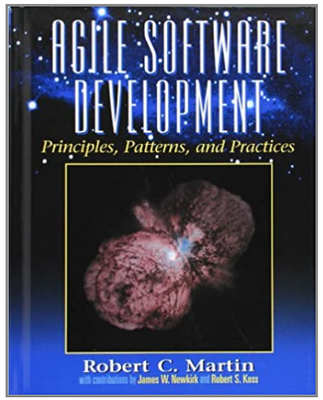
"A design is rigid if a single change causes a cascade of subsequent changes in dependent modules. The more modules that must be changed, the more rigid the design." – Robert Martin

Chapter #5:

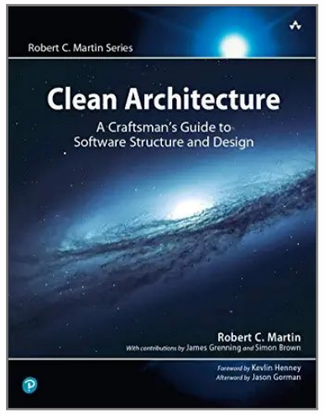
Books, Venues, Call-to-Action



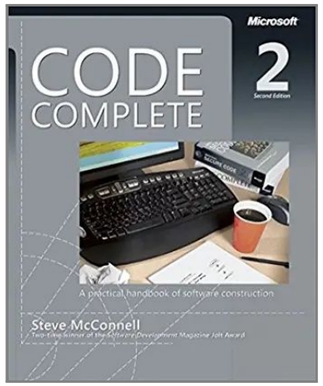
Per Kroll and Philippe Kruchten. *The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP.* Addison-Wesley, 2003



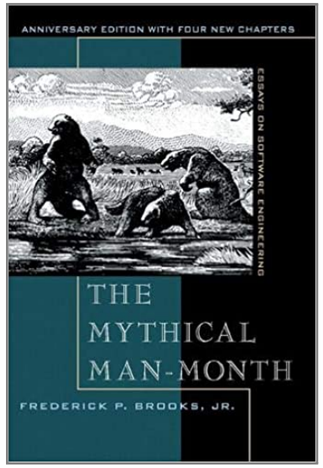
Robert C. Martin. *Agile Software Development, Principles, Patterns, and Practices.* Prentice Hall, 2002. doi:10.5555/515230



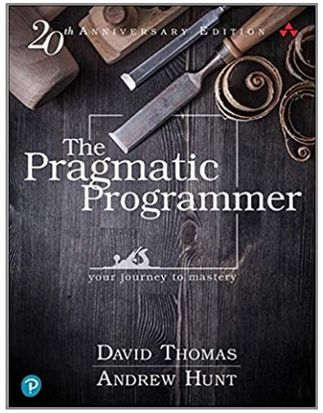
Robert C. Martin. Clean Architecture, 2017



Steve McConnell. *Code Complete*. Pearson Education, 2004. doi:10.5555/1096143



Frederick P. Brooks. *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley, 1978. doi:10.5555/540031



Andrew Hunt and Dave Thomas. *The Pragmatic Programmer: From Journeyman to Master*. Pearson Education, 1999. doi:10.5555/320326

Where to publish:

IEEE International Conference on Software Engineering (ICSE)

Call to Action:

Break down the development of your app into four interactions and specify the functionality to be delivered by the end of each of them.

Still unresolved issues:

- How to restrict design automatically?
- How to decompose the task of design?
- How to coordinate design process by robots?
- How to enforce good design?

Bibliography

Kent Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley, 2000. doi:10.5555/318762.

Frederick P. Brooks. *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley, 1978. doi:10.5555/540031.

Andrew Hunt and Dave Thomas. *The Pragmatic Programmer: From Journeyman to Master*. Pearson Education, 1999. doi:10.5555/320326.

Per Kroll and Philippe Kruchten. *The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP*. Addison-Wesley, 2003.

Robert C. Martin. *Agile Software Development, Principles, Patterns, and Practices*. Prentice Hall, 2002. doi:10.5555/515230.

Robert C. Martin. Clean Architecture, 2017.

Steve McConnell. *Code Complete*. Pearson Education, 2004. doi:10.5555/1096143.