

RUP

vs. Agile/XP

YEGOR BUGAYENKO

Lecture #3 out of 16
80 minutes

The slidedeck was presented by the author in this [YouTube Video](#)

All visual and text materials presented in this slidedeck are either originally made by the author or taken from public Internet sources, such as web sites. Copyright belongs to their respected authors.

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:

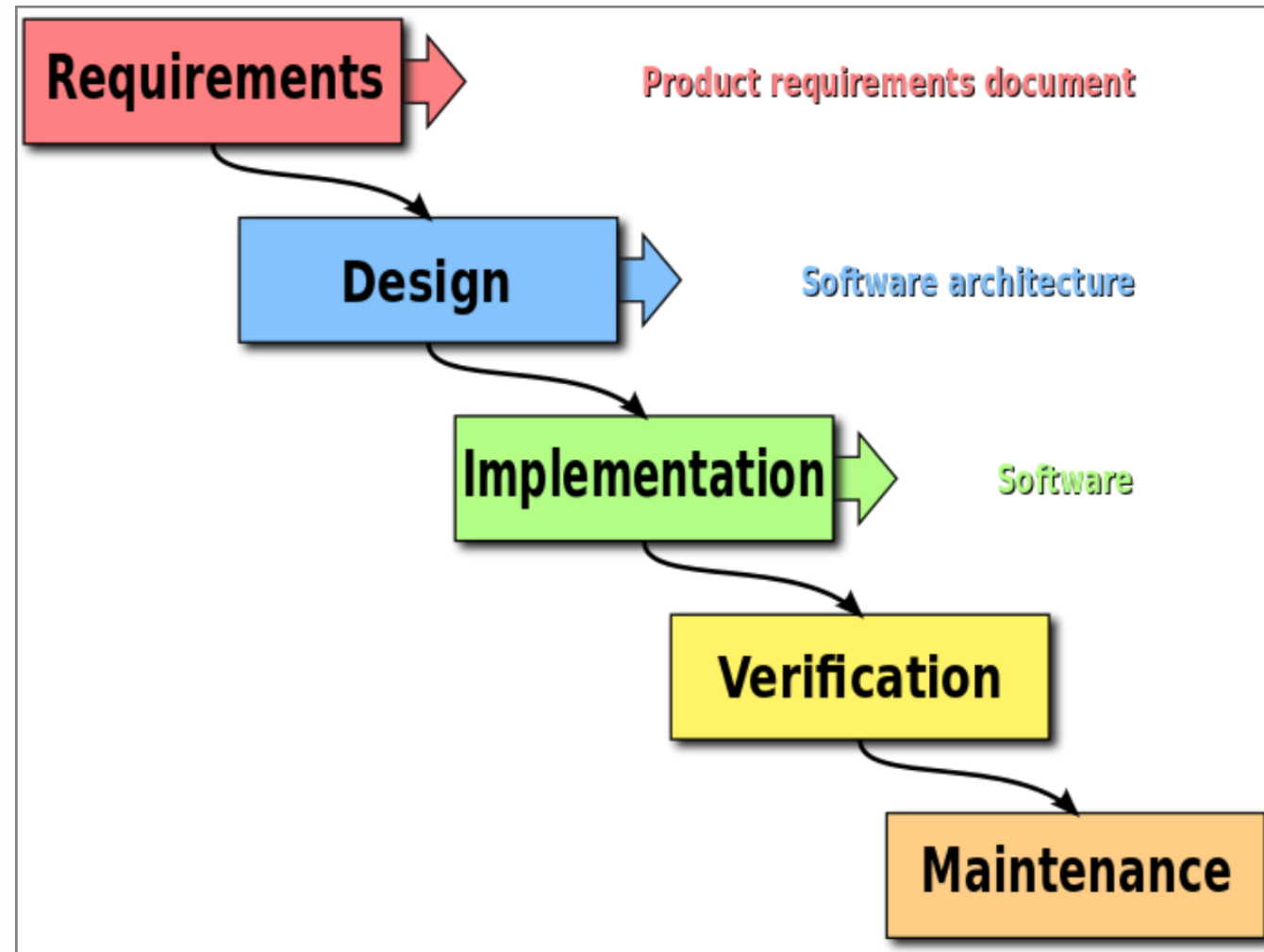
Waterfall / SDLC

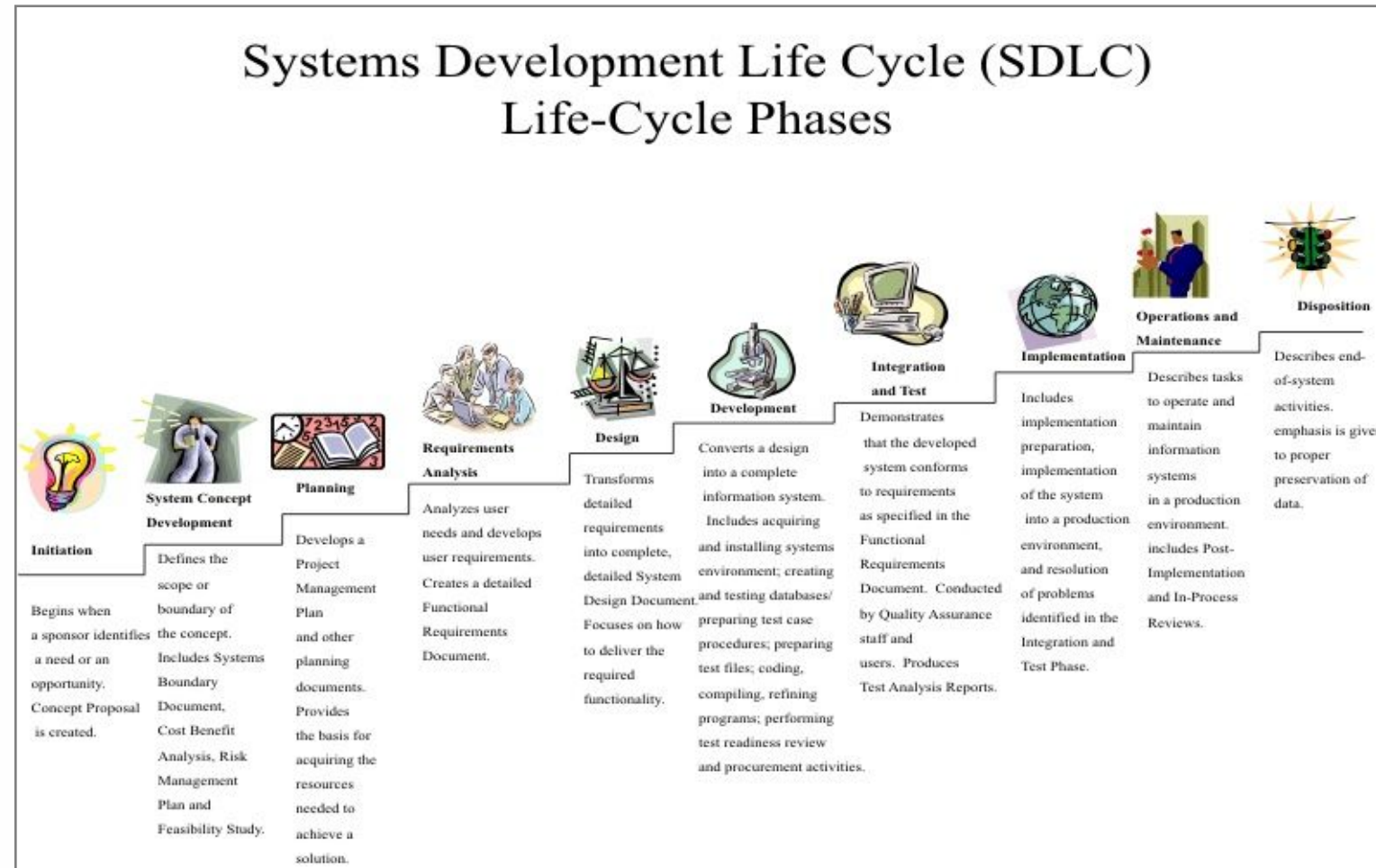
Rational Unified Process (RUP)

Microsoft Solutions Framework (MSF)

Information Technology Infrastructure Library (ITIL)

Agile & Co.



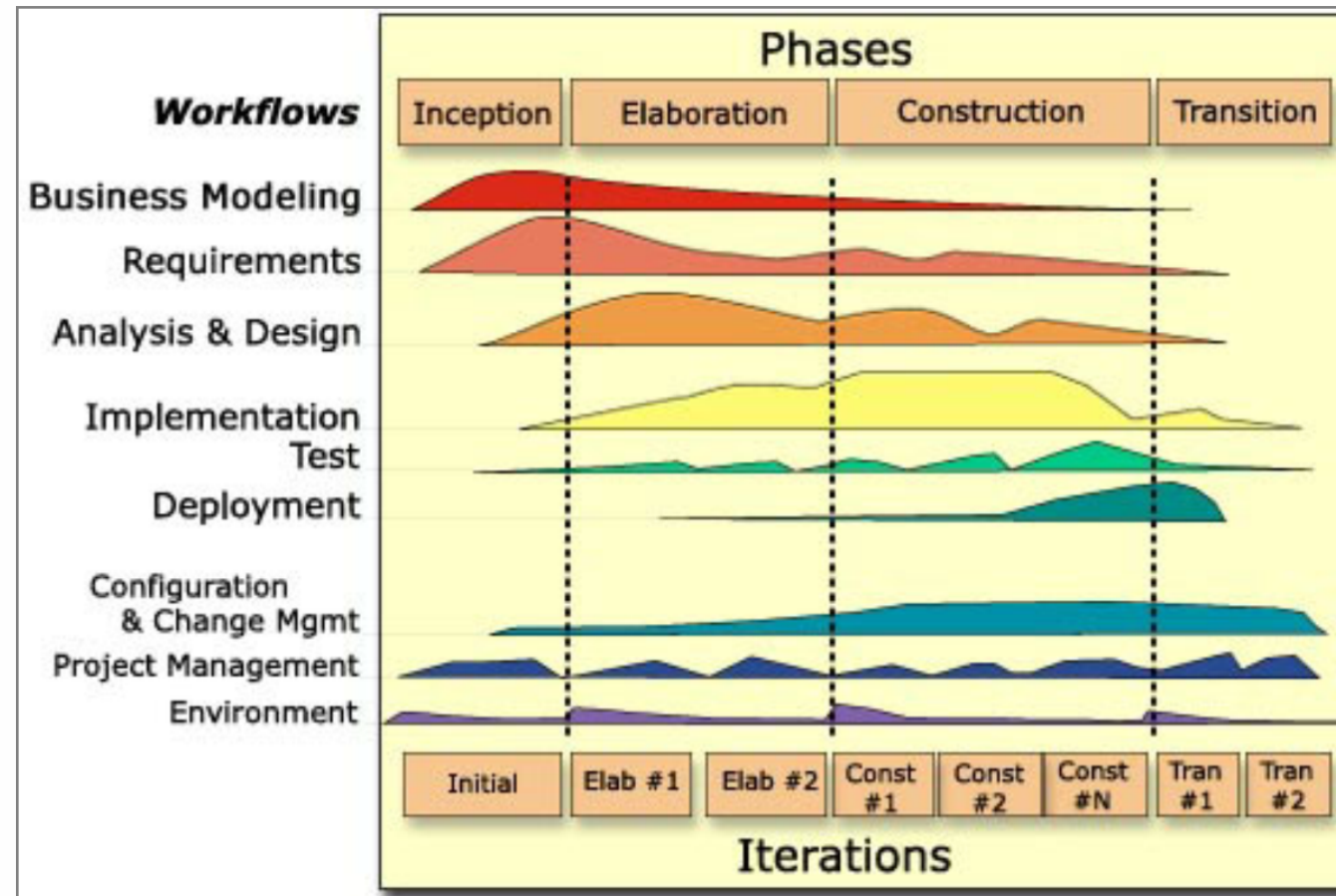


Chapter #2:

Rational Unified Process (RUP) and Co.

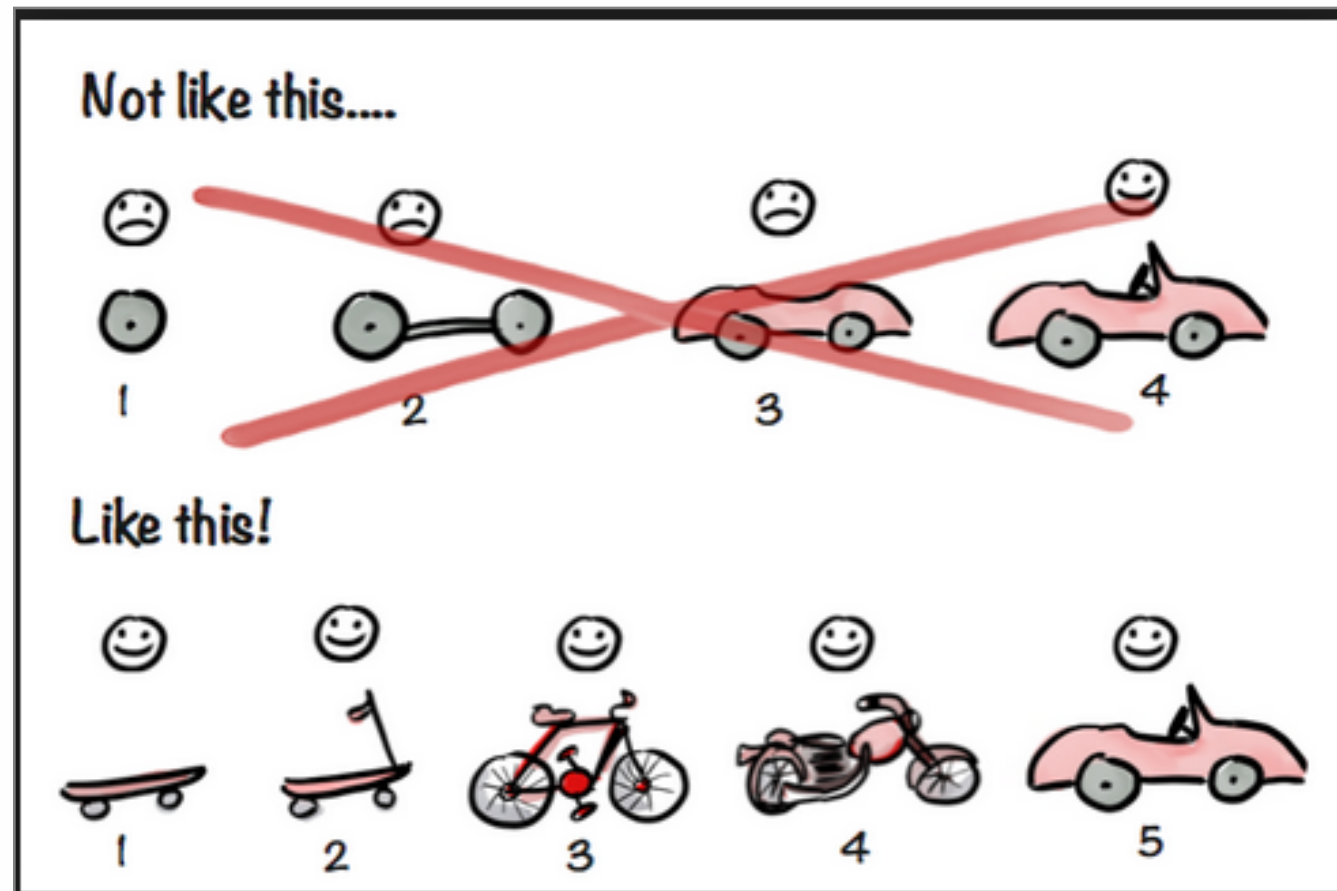
[[Incremental](#) Iterative Roles Certification MSF ITIL]

Iterative & Incremental

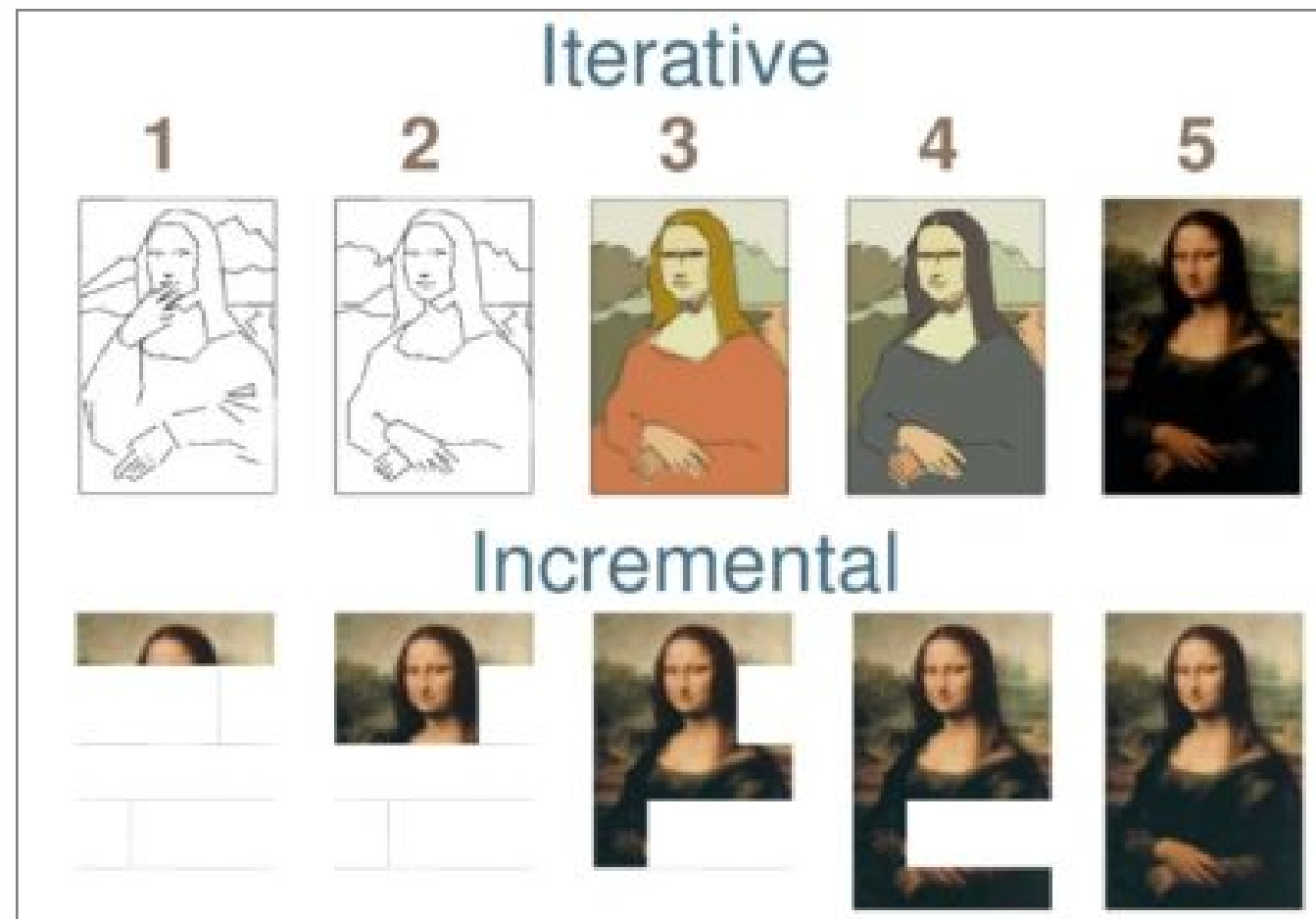


[Incremental [Iterative](#) Roles Certification MSF ITIL]

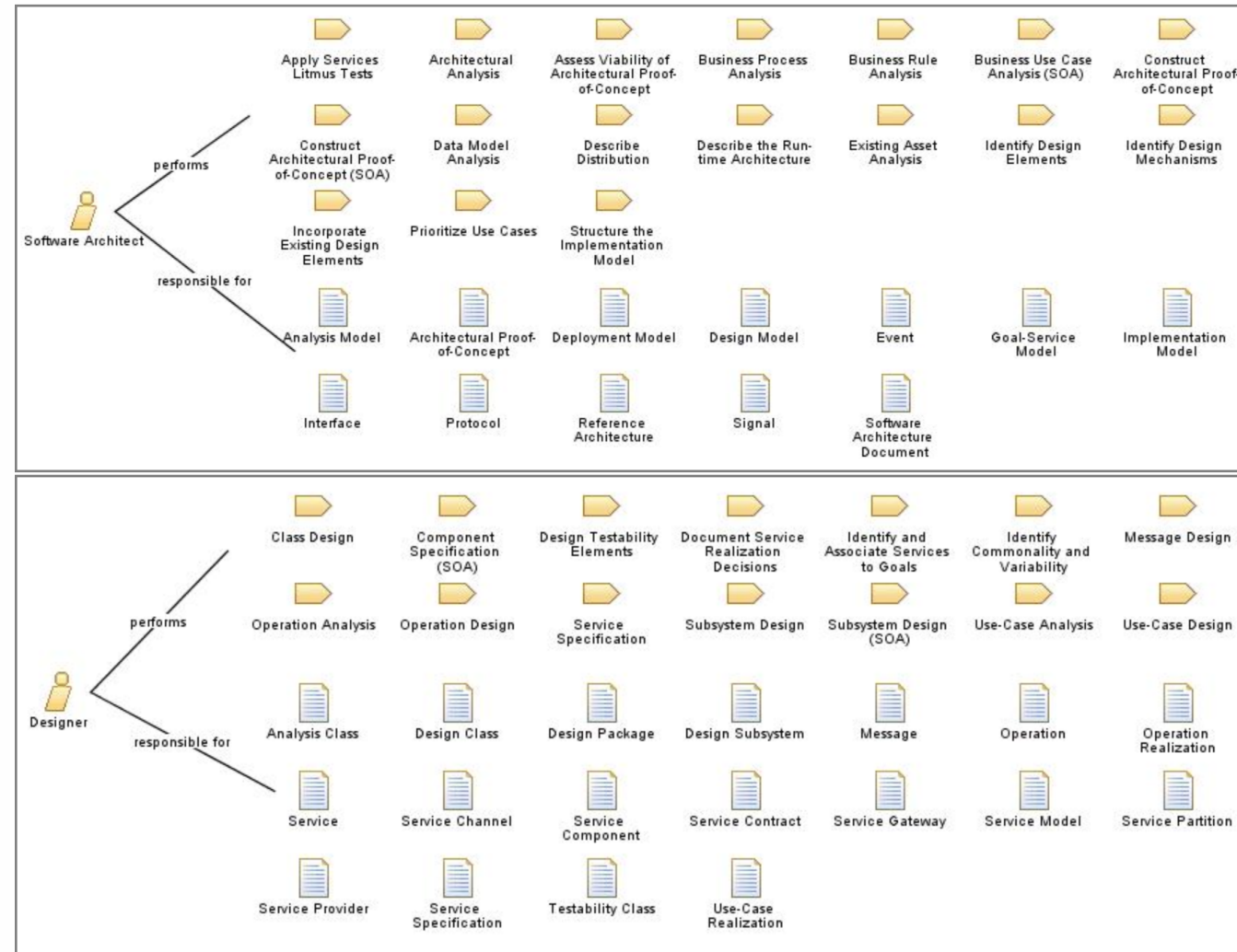
Iterative:



[Incremental [Iterative](#) Roles Certification MSF ITIL]



[Incremental Iterative Roles Certification MSF ITIL]



[Incremental Iterative Roles Certification MSF ITIL]



Microsoft Solutions Framework (MSF)

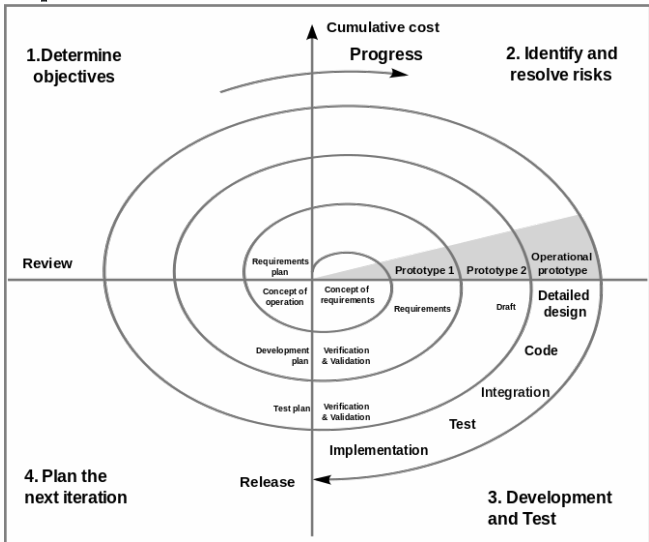


Information Technology Infrastructure Library (ITIL)

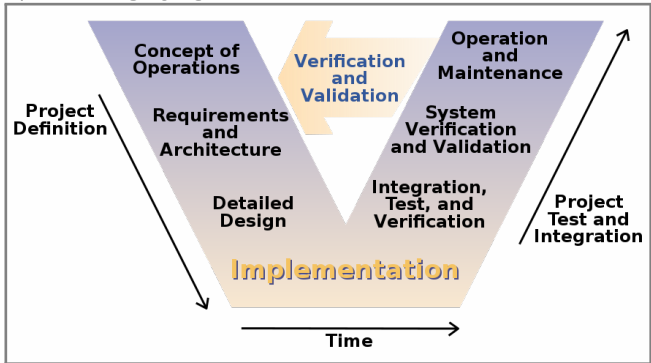


[Incremental Iterative Roles Certification MSF ITIL]

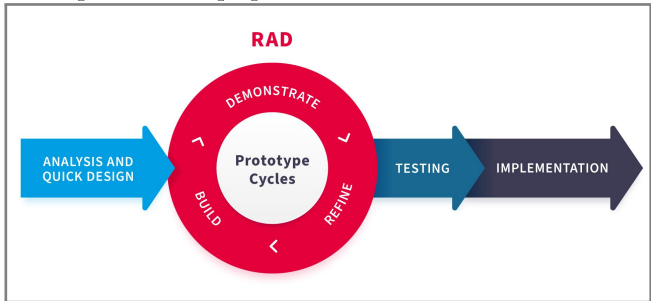
Spiral



V-Model



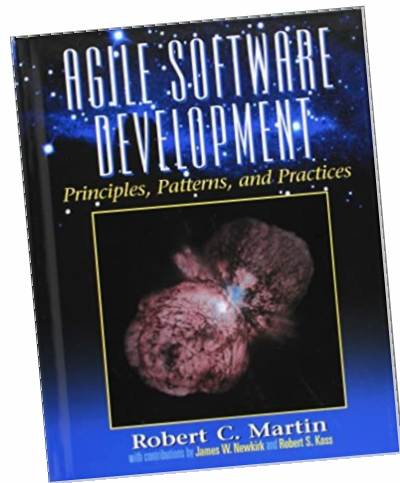
Rapid Application Development



Chapter #3:

Agile, Kanban, Scrum, XP

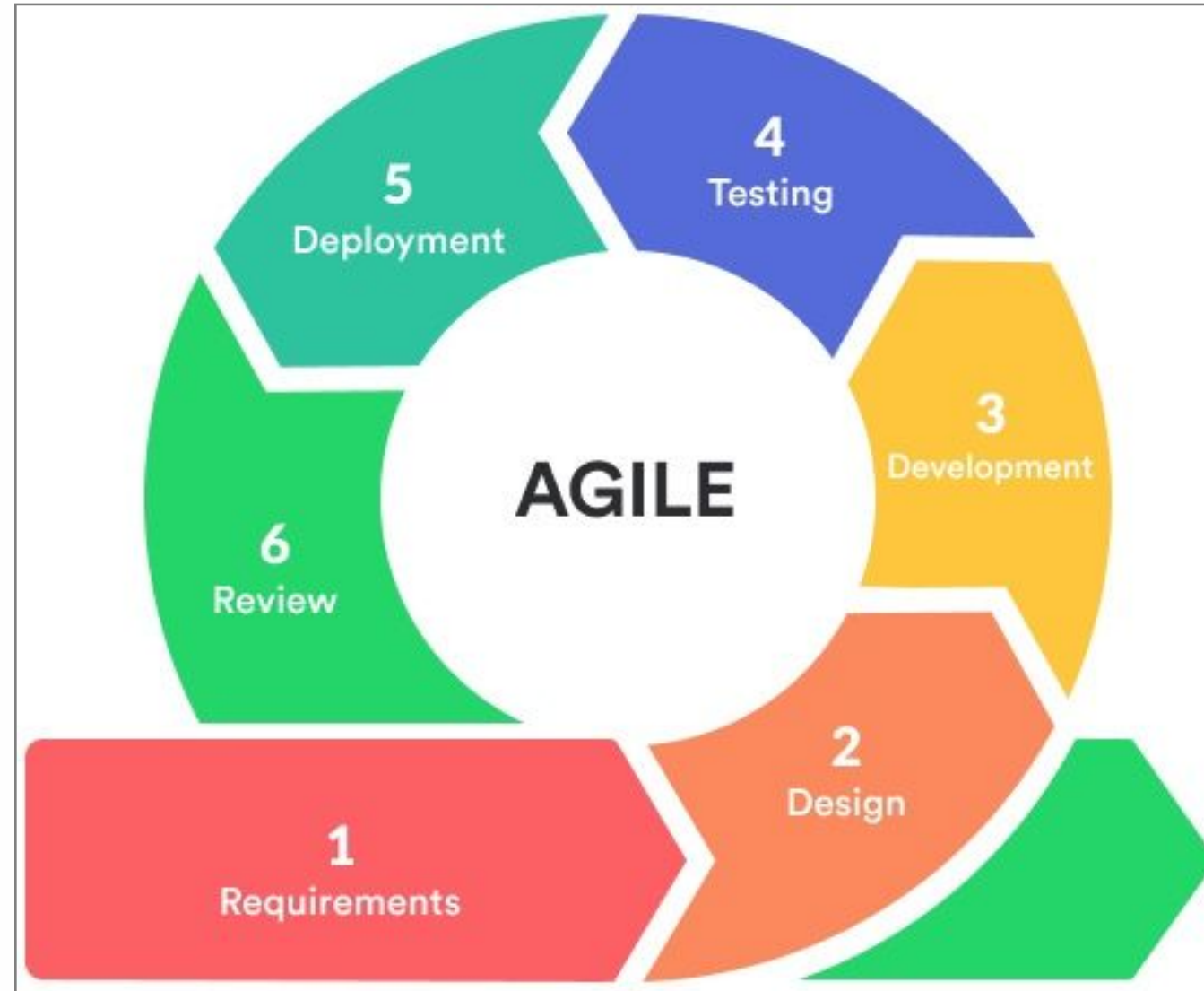
[Agile Kanban Scrum XP Cost]



“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.”

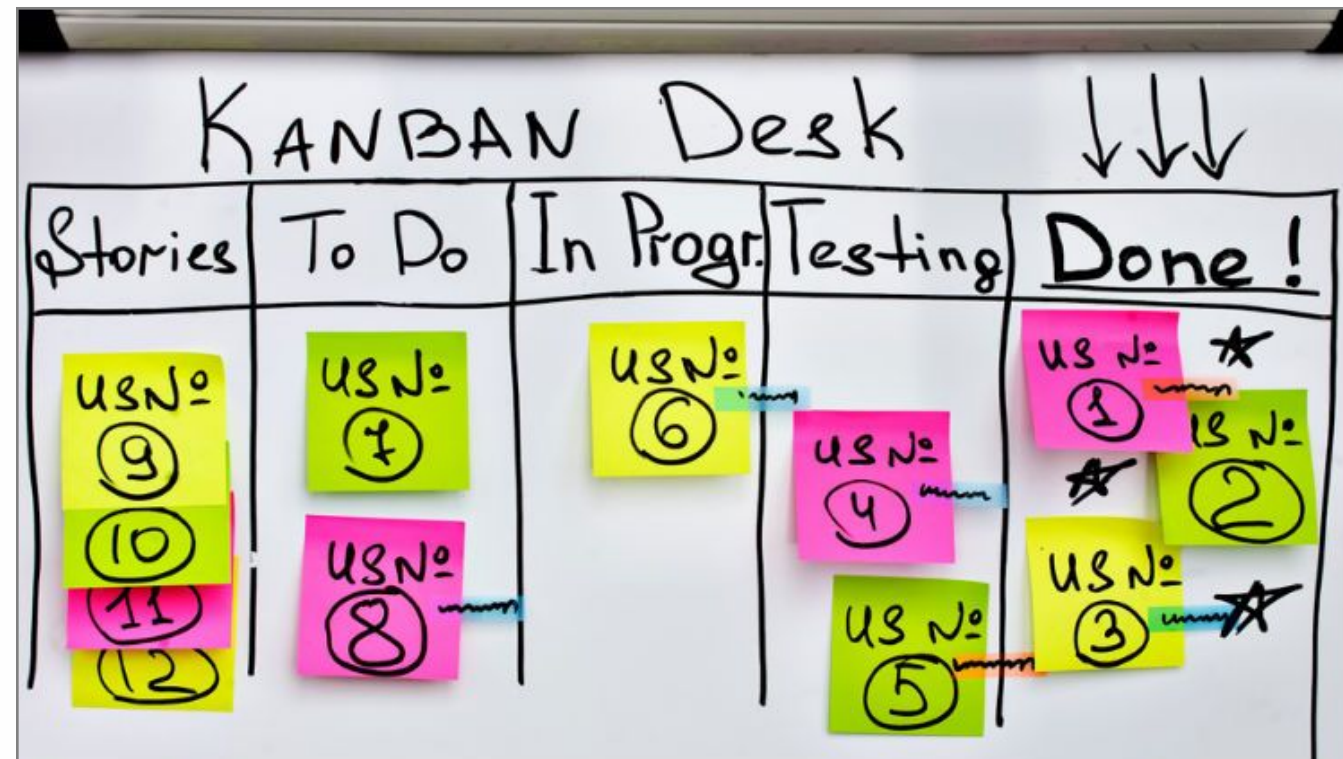
— Robert C. Martin. *Agile Software Development, Principles, Patterns, and Practices*. Prentice Hall, 2002. doi:[10.5555/515230](https://doi.org/10.5555/515230)

[[Agile](#) Kanban Scrum XP Cost]



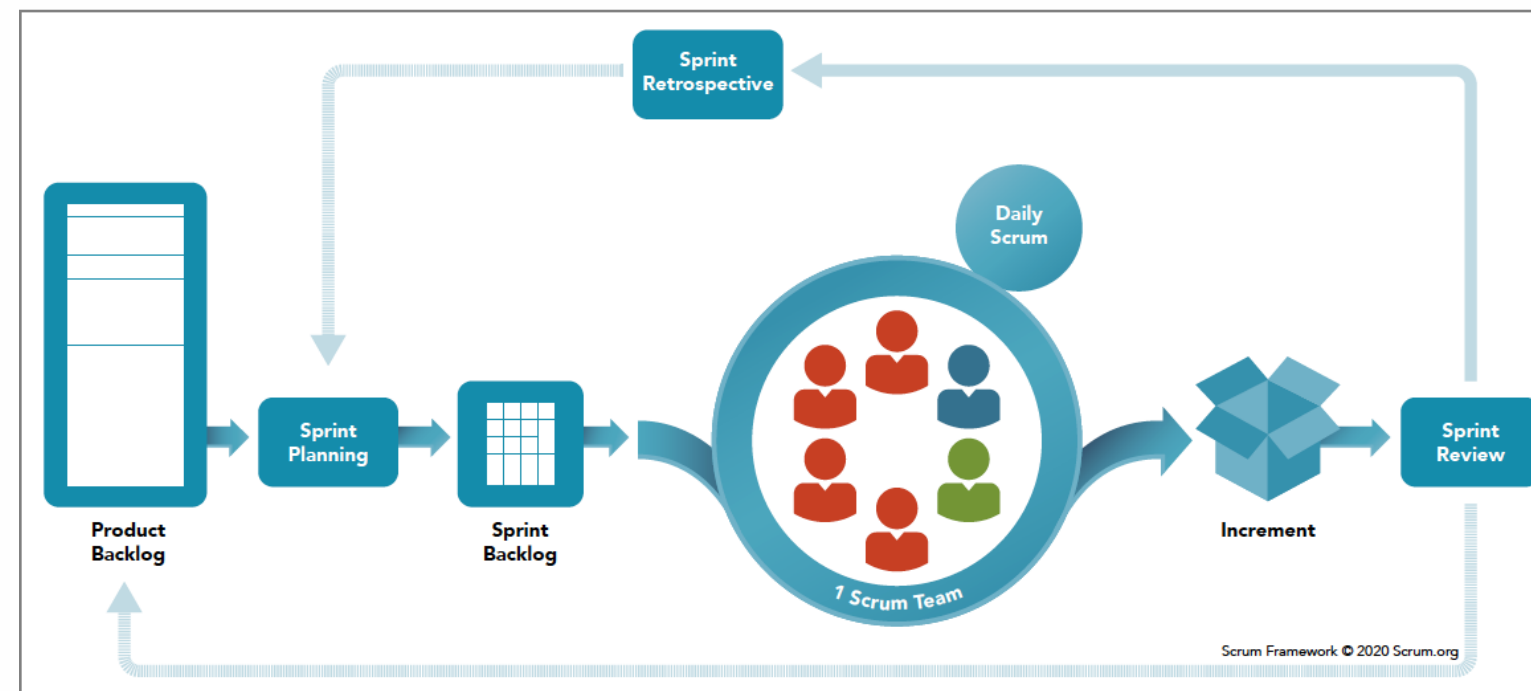
[Agile Kanban Scrum XP Cost]

Kanban

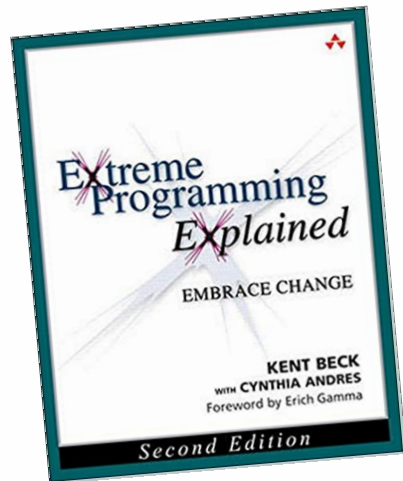


[Agile Kanban Scrum XP Cost]

Scrum Framework



[Agile Kanban Scrum XP Cost]

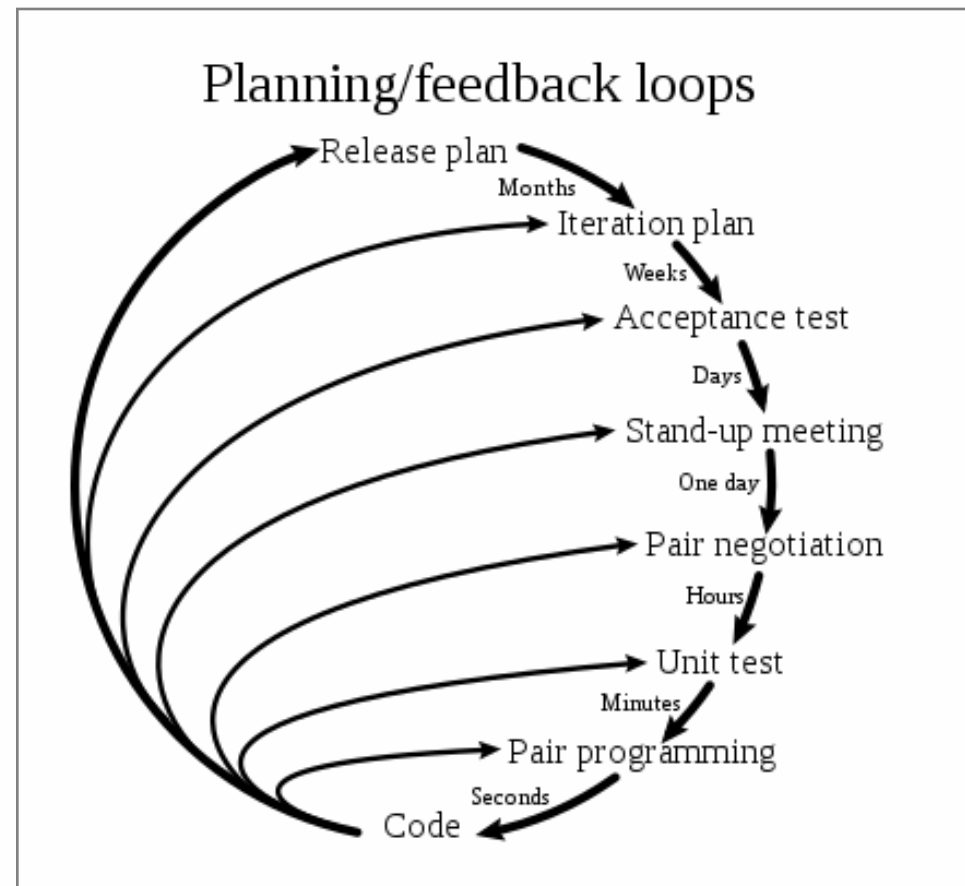


“We will continually refine the design of the system, starting from a very simple beginning. We will remove any flexibility that doesn’t prove useful.”

— Kent Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley, 2000. doi:[10.5555/318762](https://doi.org/10.5555/318762)

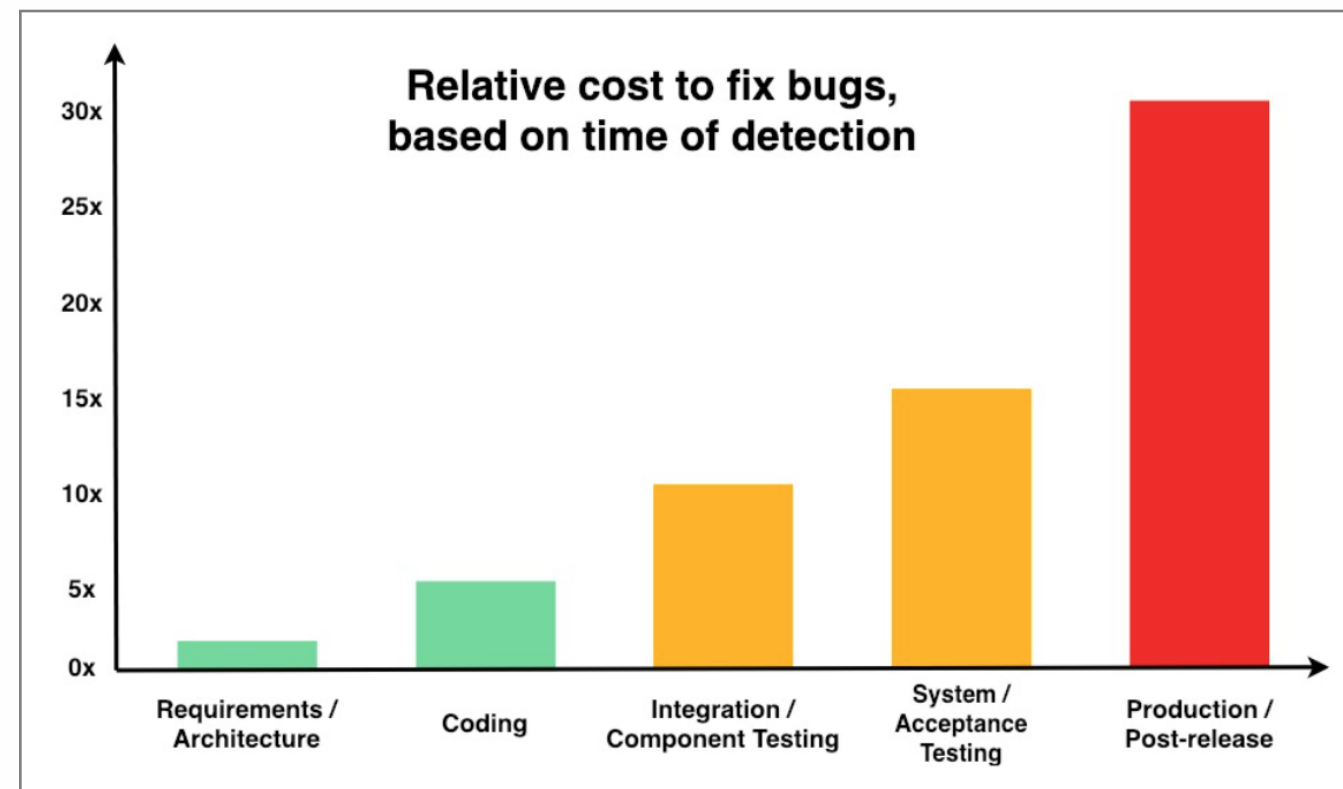
[Agile Kanban Scrum XP Cost]

eXtreme Programming (XP)



[Agile Kanban Scrum XP [Cost](#)]

Cost of Bugs



Chapter #4:

Qualities of Good Design

[Complexity Duplication Immobility]

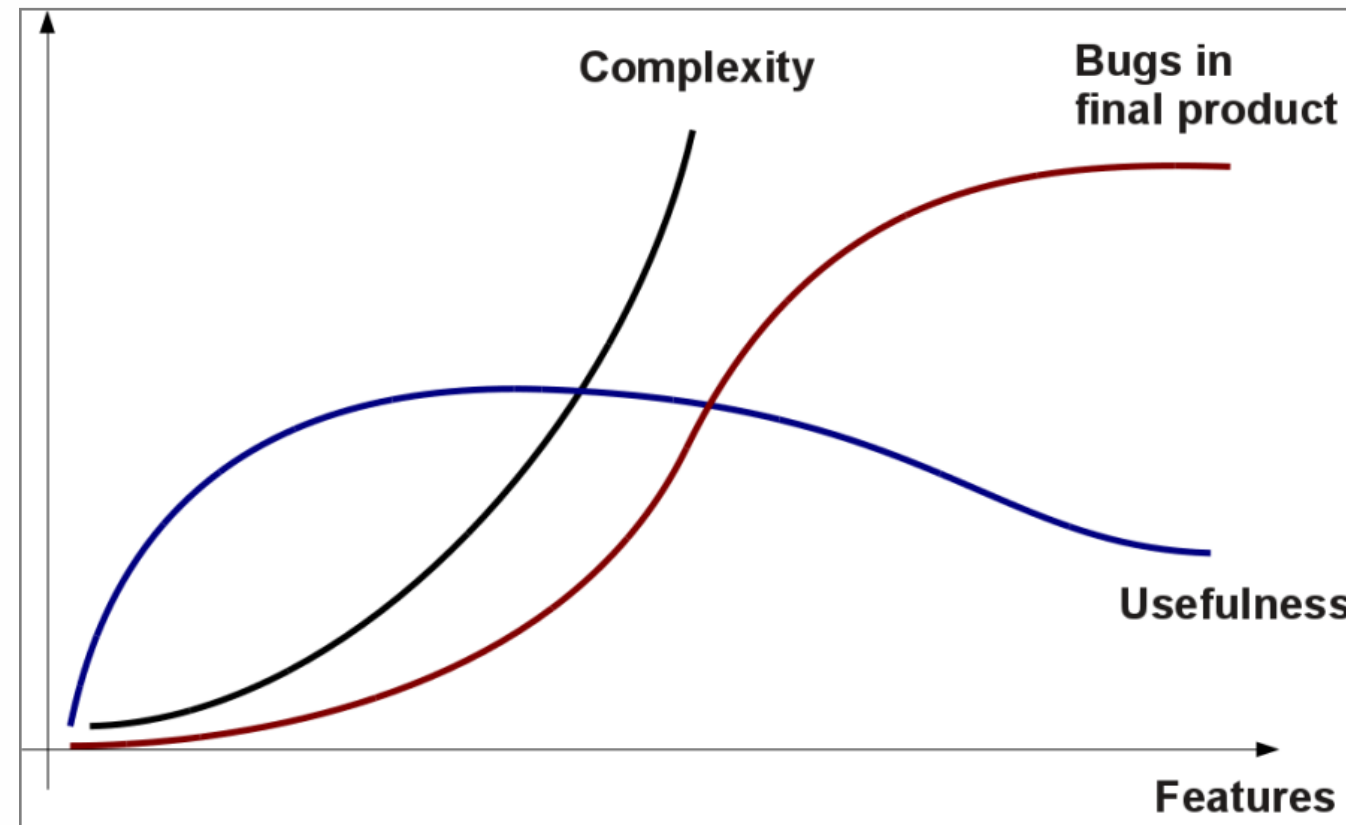
It Must Be Simple

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

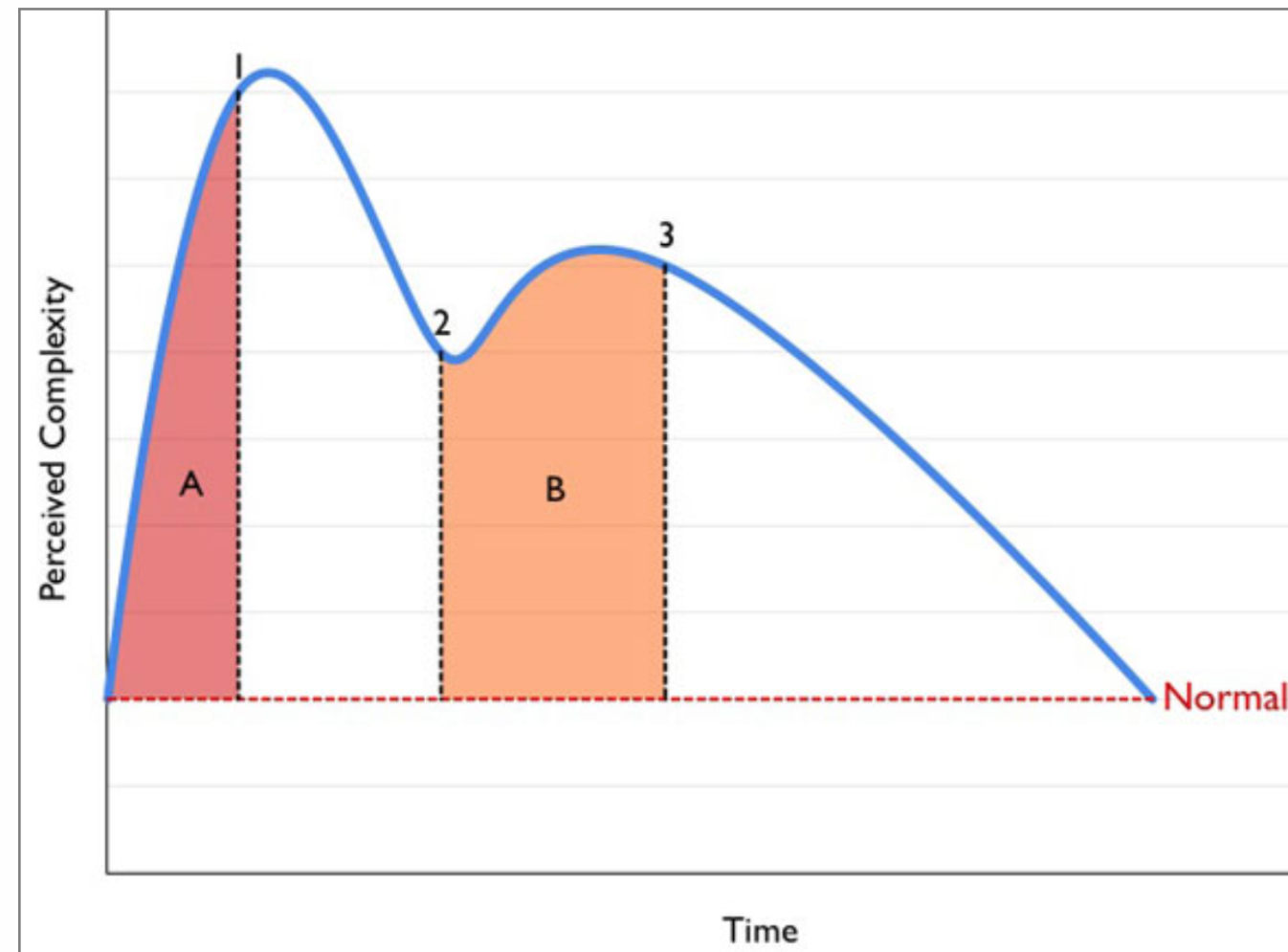
“A design contains needless complexity when it contains elements that aren’t currently useful. This frequently happens when developers anticipate changes to the requirements, and put facilities in the software to deal with those potential changes.” – Robert Martin

“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](#) Duplication Immobility]



[[Complexity](#) Duplication Immobility]



[[Complexity](#) Duplication Immobility]

“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

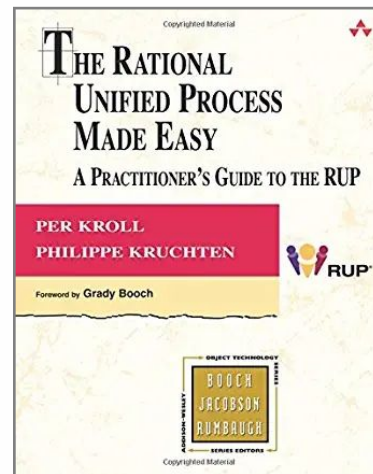
It Must Be Modular

“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

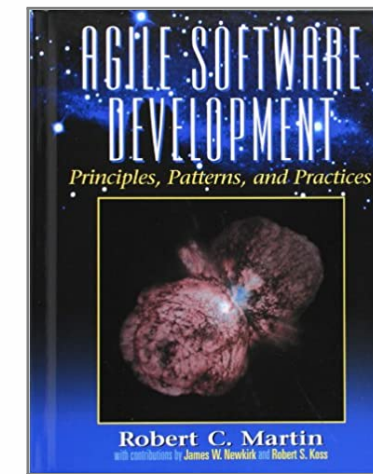
“A design is rigid if a single change causes a cascade of subsequent changes in dependent modules. The more modules that must be changed, the more rigid the design.” – Robert Martin

Chapter #5:

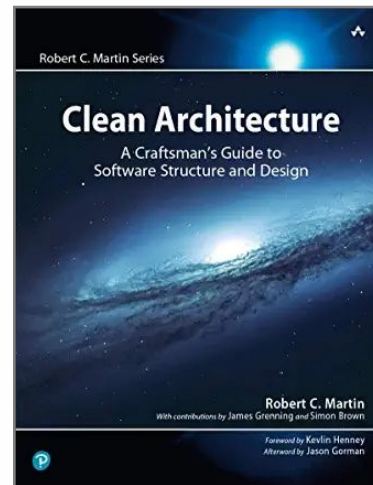
Books, Venues, Call-to-Action



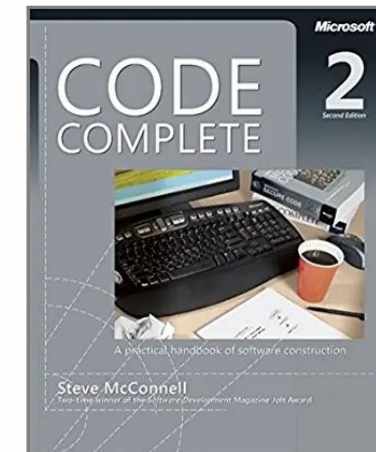
Per Kroll and Philippe Kruchten. *The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP*. Addison-Wesley, 2003



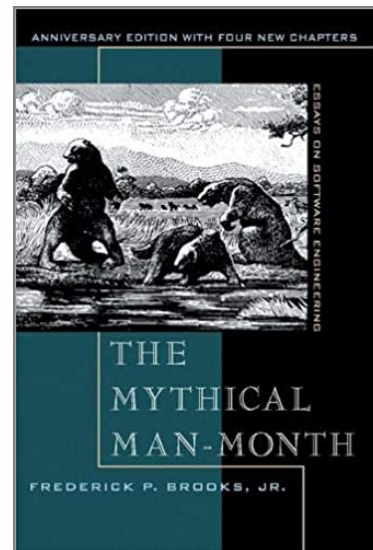
Robert C. Martin. *Agile Software Development, Principles, Patterns, and Practices*. Prentice Hall, 2002.
doi:[10.5555/515230](https://doi.org/10.5555/515230)



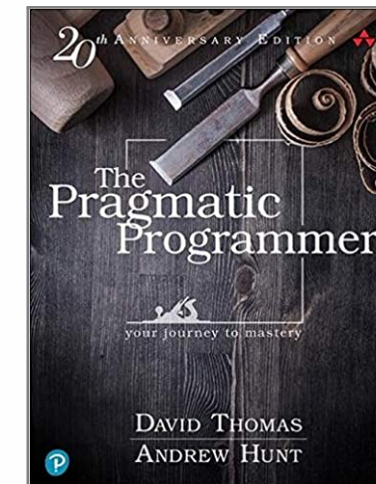
Robert C. Martin. Clean Architecture, 2017



Steve McConnell. *Code Complete*. Pearson Education, 2004. doi:[10.5555/1096143](https://doi.org/10.5555/1096143)



Frederick P. Brooks. *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley, 1978.
doi:[10.5555/540031](https://doi.org/10.5555/540031)



Andrew Hunt and Dave Thomas. *The Pragmatic Programmer: From Journeyman to Master*. Pearson Education, 1999. doi:[10.5555/320326](https://doi.org/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](https://doi.org/10.5555/318762).

Frederick P. Brooks. *The Mythical Man-Month: Essays on*

Software Engineering. Addison-Wesley, 1978.
doi:[10.5555/540031](https://doi.org/10.5555/540031).

Andrew Hunt and Dave Thomas. *The Pragmatic Programmer: From Journeyman to Master*. Pearson Education, 1999.
doi:[10.5555/320326](https://doi.org/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](https://doi.org/10.5555/515230).

Robert C. Martin. *Clean Architecture*, 2017.

Steve McConnell. *Code Complete*. Pearson Education, 2004.
doi:[10.5555/1096143](https://doi.org/10.5555/1096143).