CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Object-oriented Programming Project

Introduction and software development overview

Dr. Alex Gerdes

TDA367/DIT212 - HT 2018

Introduction

- Teaching team:
  - Lecturer, course responsible, and examiner:
    - Alex Gerdes, alexg@chalmers.se, room EDIT 6479
  - Course assistants:
    - Joel Hultin
    - Mazdak Farrokhzad
    - Robert Krook
    - Adam Waldenberg
    - Ayberk Tosun
- Contact information on the course website
- Student representatives!

Context

- A combined Chalmers and GU course
- Prerequisites: TDA548/DIT012 and TDA552/DIT953 (or similar)
- Course position:

Expected learning outcomes

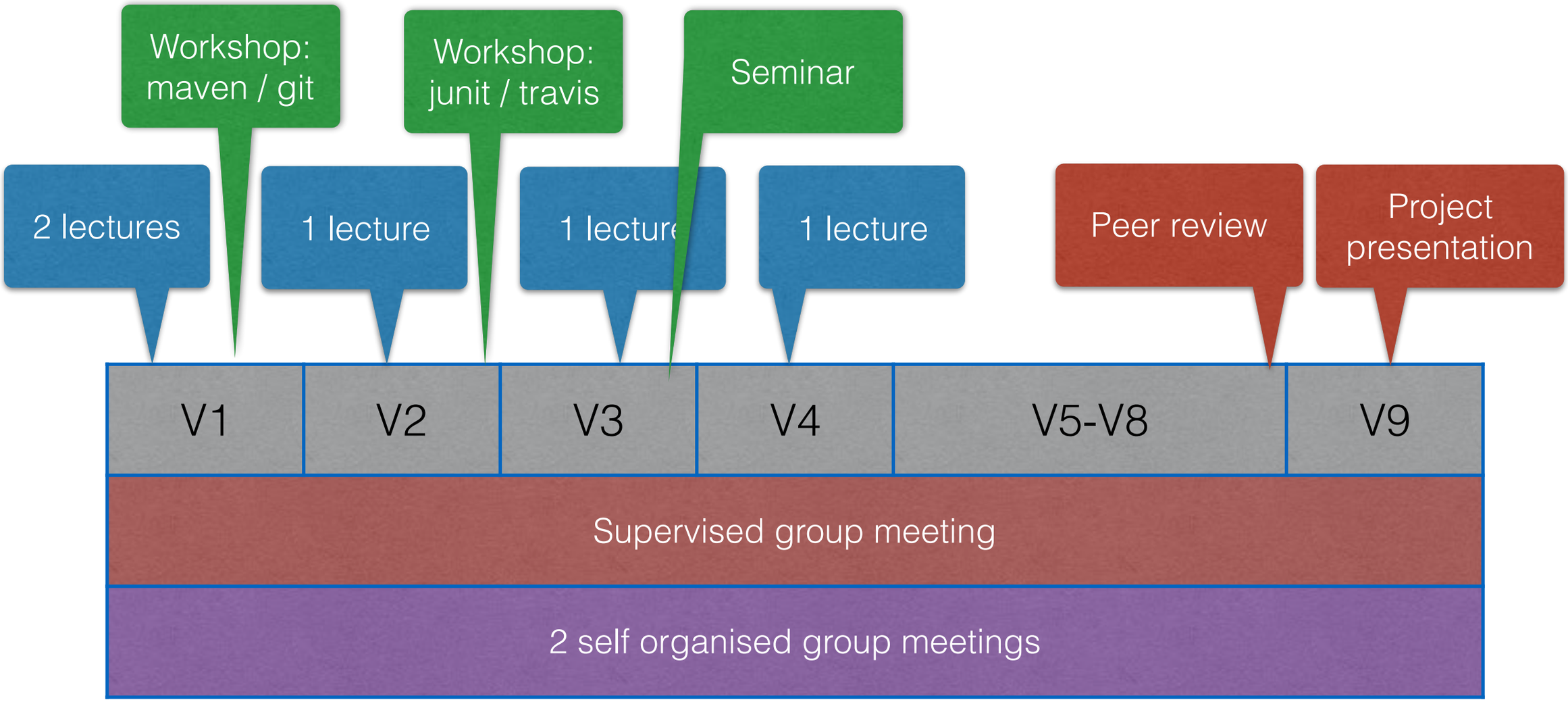
- Knowledge and understanding
  - Explain development methods in technical projects, especially software projects.
  - Describe the phases of a development project: problem identification, problem description, user analysis, specification, design, implementation, testing, evaluation, etc.
  - Describe basic concepts in software development, e.g. requirements and development process.
- Skills and abilities
  - Run a small scale object-oriented software project in a group according to predefined development process and schedule.
  - Write a report of the project, containing a basic requirements specification and the design.
  - Apply skills from previous programming courses, such as design principles and patterns, in the design and development phase of the project.
  - Use modern software development tools, such as testing frameworks, 'build automation', and version management.
- Judgement and approach
  - Reflect on good object-oriented design and implementation in the project.

Course content

- What are we going to do?
  - Lectures
  - Workshops
  - Seminar
  - Project

- Project:
  - Work together in a group to build an application of your choice following a (predefined) agile software process
  - Expected application type is a standalone or mobile with a GUI
  - Examples:
    - Chat application
    - Mail application
    - (preferably not a 2D-game, but not forbidden)

Course organisation



Lectures

- Just a few lectures (compared to other courses)
  - Software Development overview (this lecture)
  - User stories and related topics
  - Analysis
  - Design
  - Implementation
- Purpose: introduce concepts
- (not many new concepts, exercise what you learned in the first year)

Workshops

- Topics
  - Version control: Git
  - Build automation: Maven
  - Unit testing: JUnit
  - Continuous integration: Travis

- Pointers to tutorials + some instructions

- Purpose: learn about tools and how to use them

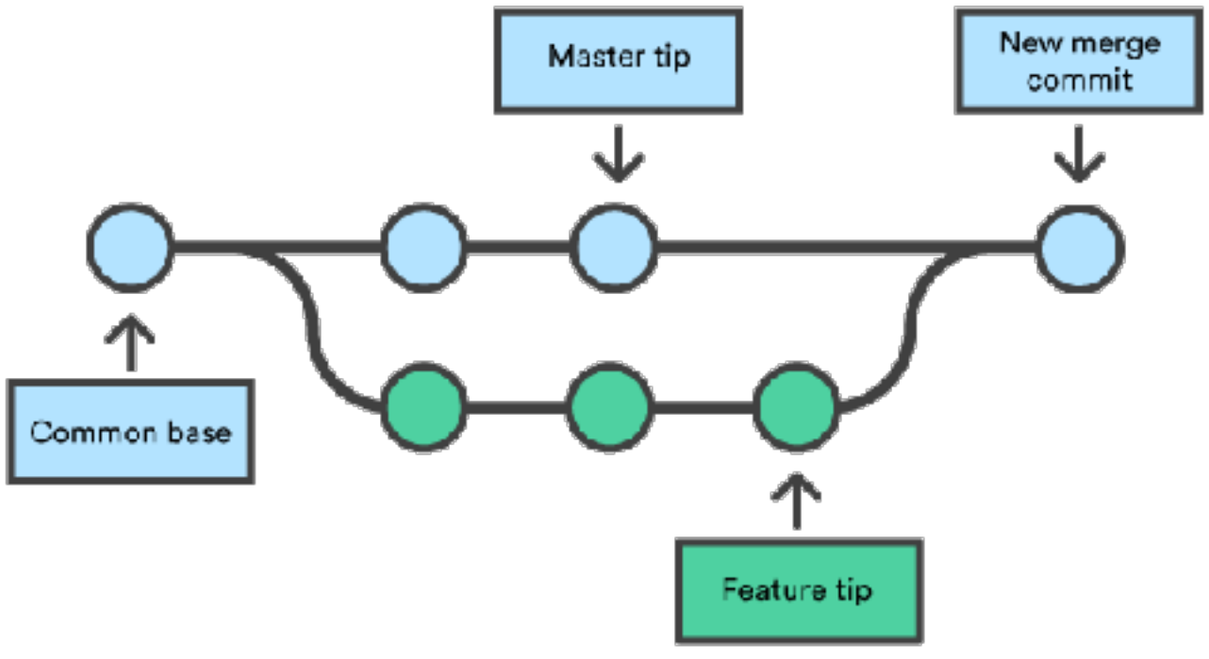
- Other possible useful tools:
  - Project management: Google Sheets, Trello
  - Communication: Slack

Build automation

- Automate all steps to build, test, package, and deploy software
- Modern software development is highly dependent on libraries
- First thing to investigate if some need: Is there a library?
  - Potentially very complex to handle because of transitive dependencies (libraries depend on other libraries, that depends on ...)
- By using build automation software to handle libraries/APIs needed by our application we greatly reduce the problems!
- We accept: Maven and Gradle

Version control: Git

- Git is mandatory
- Hosting may be on GitHub or BitBucket, as long as assistants and I have access
- Must use Git in a disciplined way
- All members should use a (the same) workflow (more in workshop)



Groups

- Size: 4 or 5 persons
- Register group via link on course website,
- Do it as soon as possible
  - deadline: Wednesday 5 September at 13:00
- Find team members with same ambition level, important!
- Not necessarily same skill level
- Can’t find a group: contact me

Group meetings

- With assistant:
  - 1 per week, approximately 1 hour
  - Help with process, advice in design, documentation, etc.
  - *Not a bug fixer!*
  - The group is responsible, you are supposed to push
  - Any problems: contact me
  - **Mandatory and on time!**
- Self organised:
  - At least 2 times per week
  - Documented: agendas on GitHub
  - Mandatory!

Examination

- Project presentation
  - Demo run application
  - Technical walkthrough
  - Answer questions (from audience)
  - Act as opponents for other group
    - ~ 20 min./group
- End report
  - Integration of documents written during project
  - Peer review of other group’s design and code
- Group gets a grade, which is adjusted for each individual

Course website

- The course website contains updated and relevant information:
  - Latests news (also announced via a Google-group, subscribe!)
  - Schedule
  - Slides (contains last year’s slides now, will be updated after/ just before lecture)
  - Contact information
  - Project description
  - Lots of pointers to relevant information
  - …
- Check it regularly!

Ask questions!

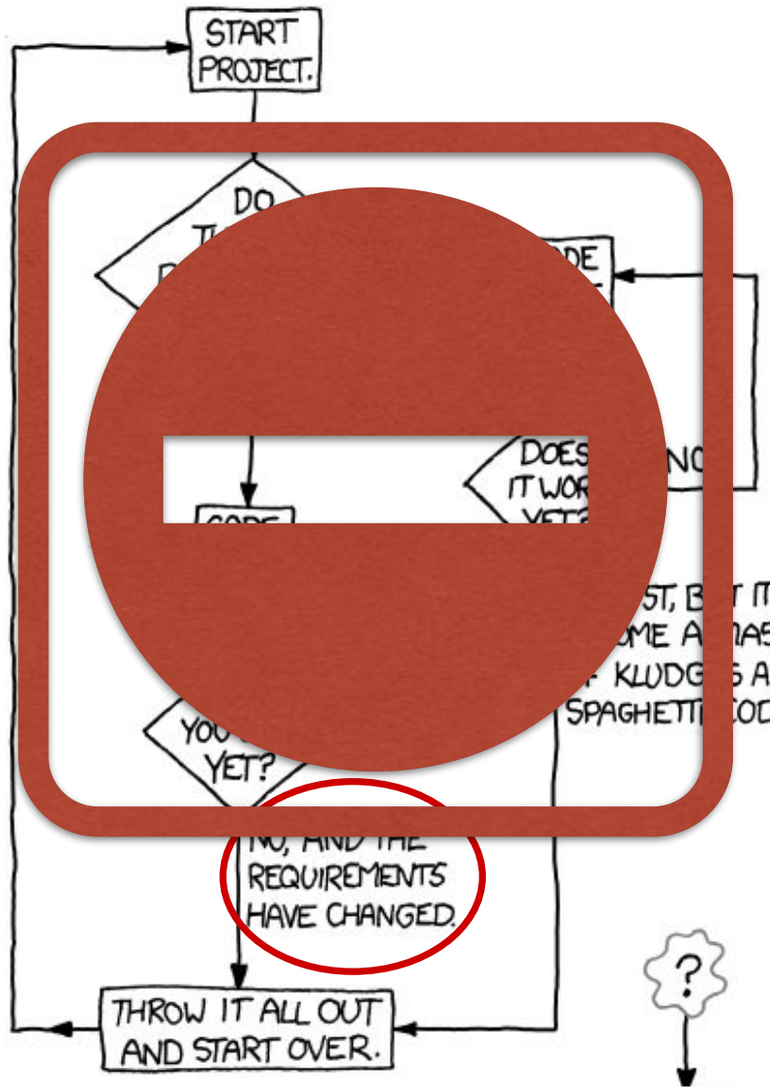
Reflect!

It is going to be fun!!!

Software development

Software development

- Software development is different from classical engineering
- Software development
  - Is often very complex because
    - many stakeholders
    - concepts or specifications not well defined
  - Is a young engineering discipline, somewhat of an art
  - Is in between very informal (dynamic/chaotic)
  - Is normally a group task
  - Is highly dependent on communication
- How to construct a high quality application fulfilling its intent?
  - Unsolved problem... but many impressive applications constructed



Waterfall model

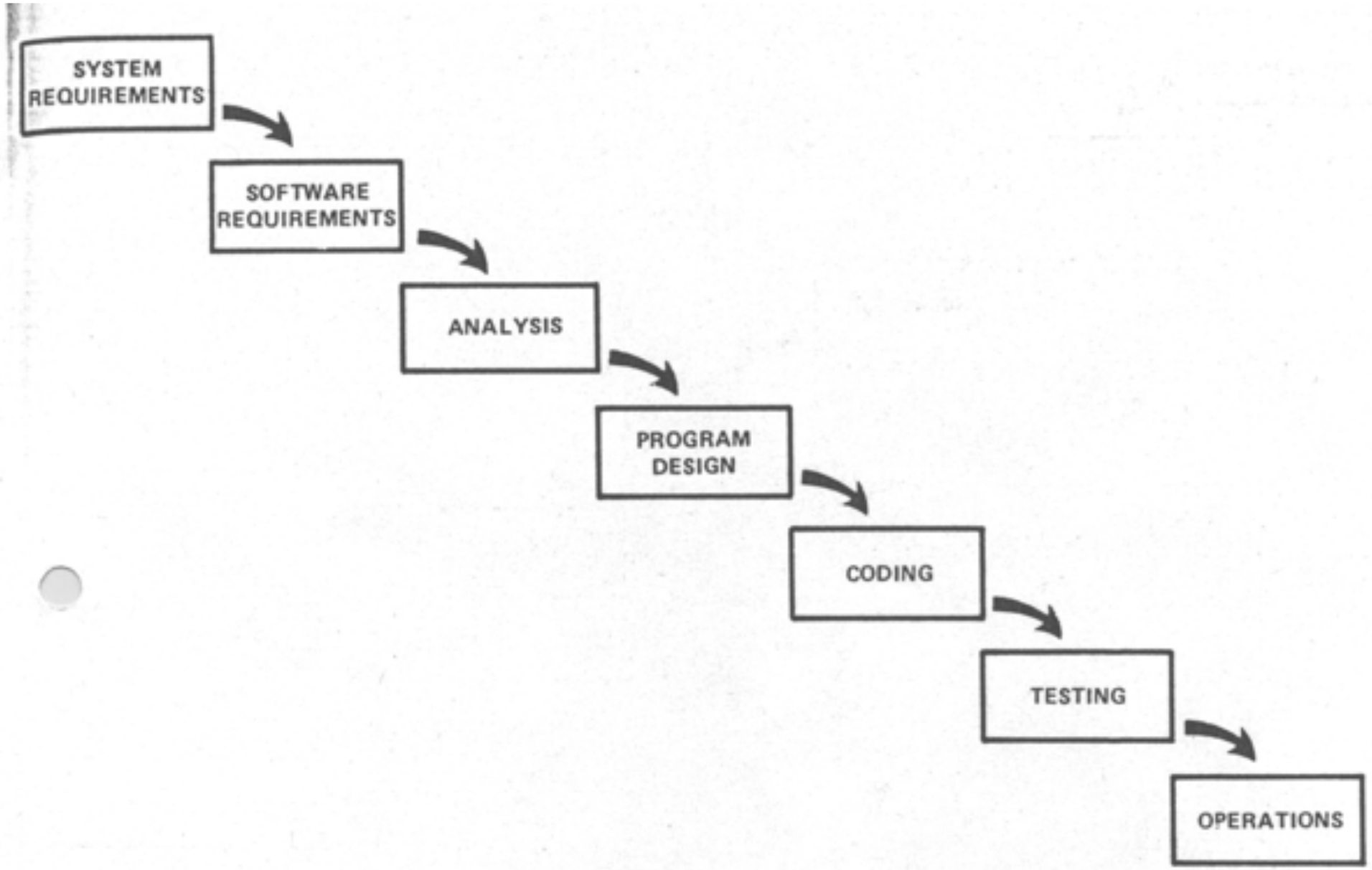


Figure 2. Implementation steps to develop a large computer program for delivery to a customer.

[Royce, 1970]: https://dl.acm.org/citation.cfm?id=41801

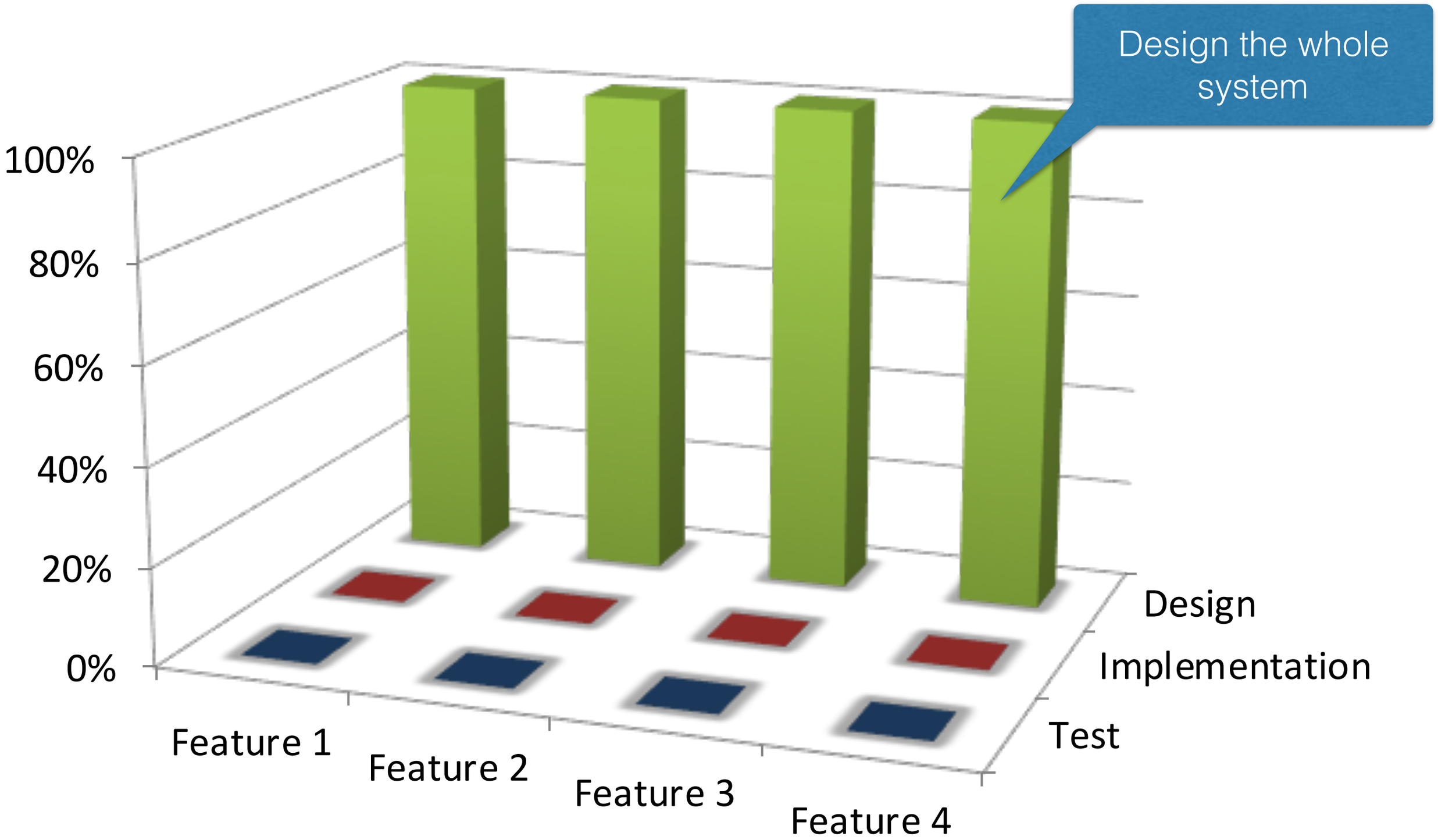
Waterfall model

- Phases
  - Requirements analysis and definition
  - System and software design
  - Implementation and unit testing
  - Integration and system testing
  - Operation and maintenance
- Main principle: One phase has to be completed before the next phase can be started
- Changes are hard due to implied dependencies between artefacts from different phases

Waterfall model - restrictions

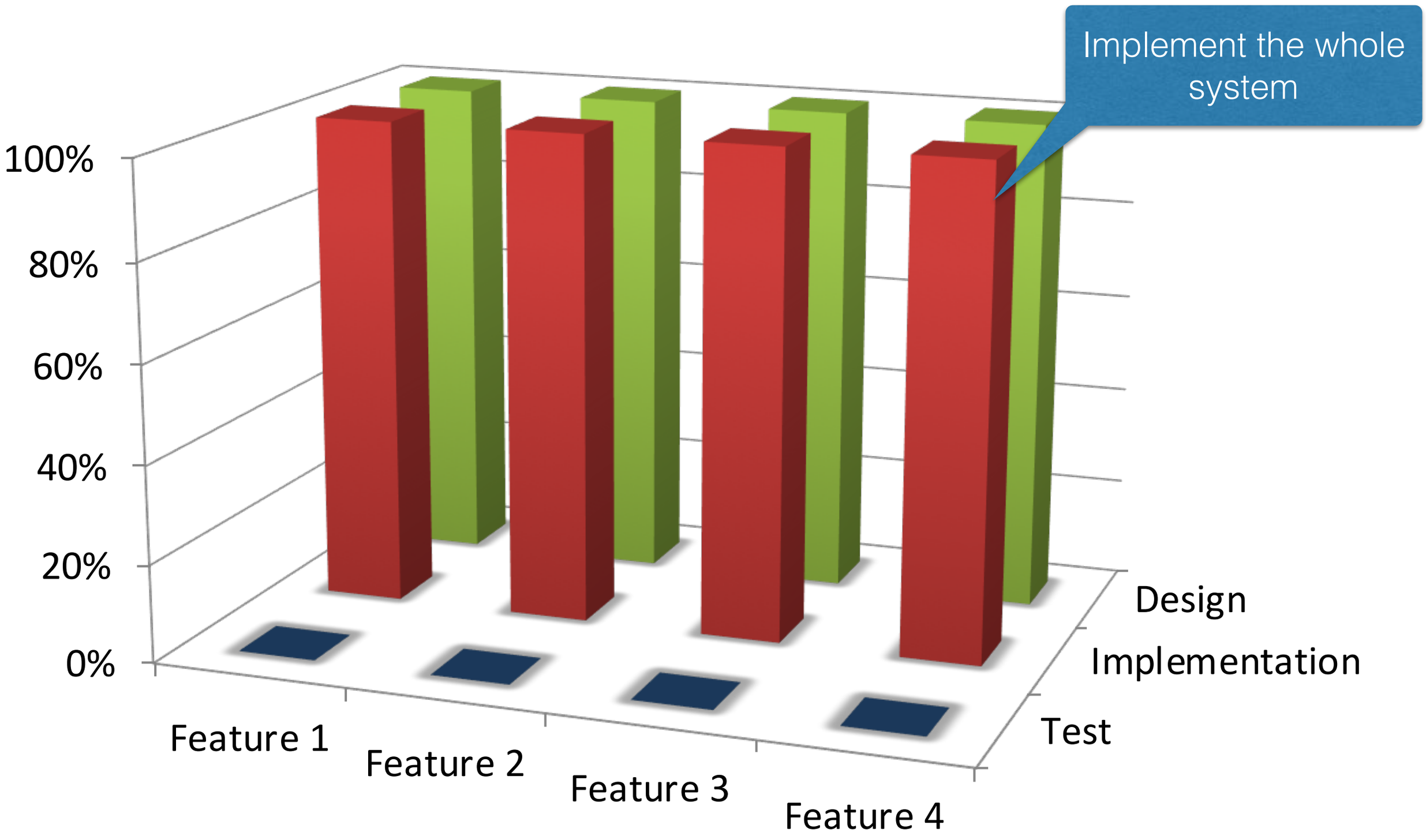
- Customer's change requests cannot be adapted easily by the ongoing development process
- The process should only be applied when requirements are pretty complete, well understood by all stakeholders (customer, project manager, developers, testers, ...), and changes are not expected
- Only a few projects fulfil all these preconditions at the beginning
- Waterfall model is used for large engineering projects, which might be spread over several development sites

Waterfall model



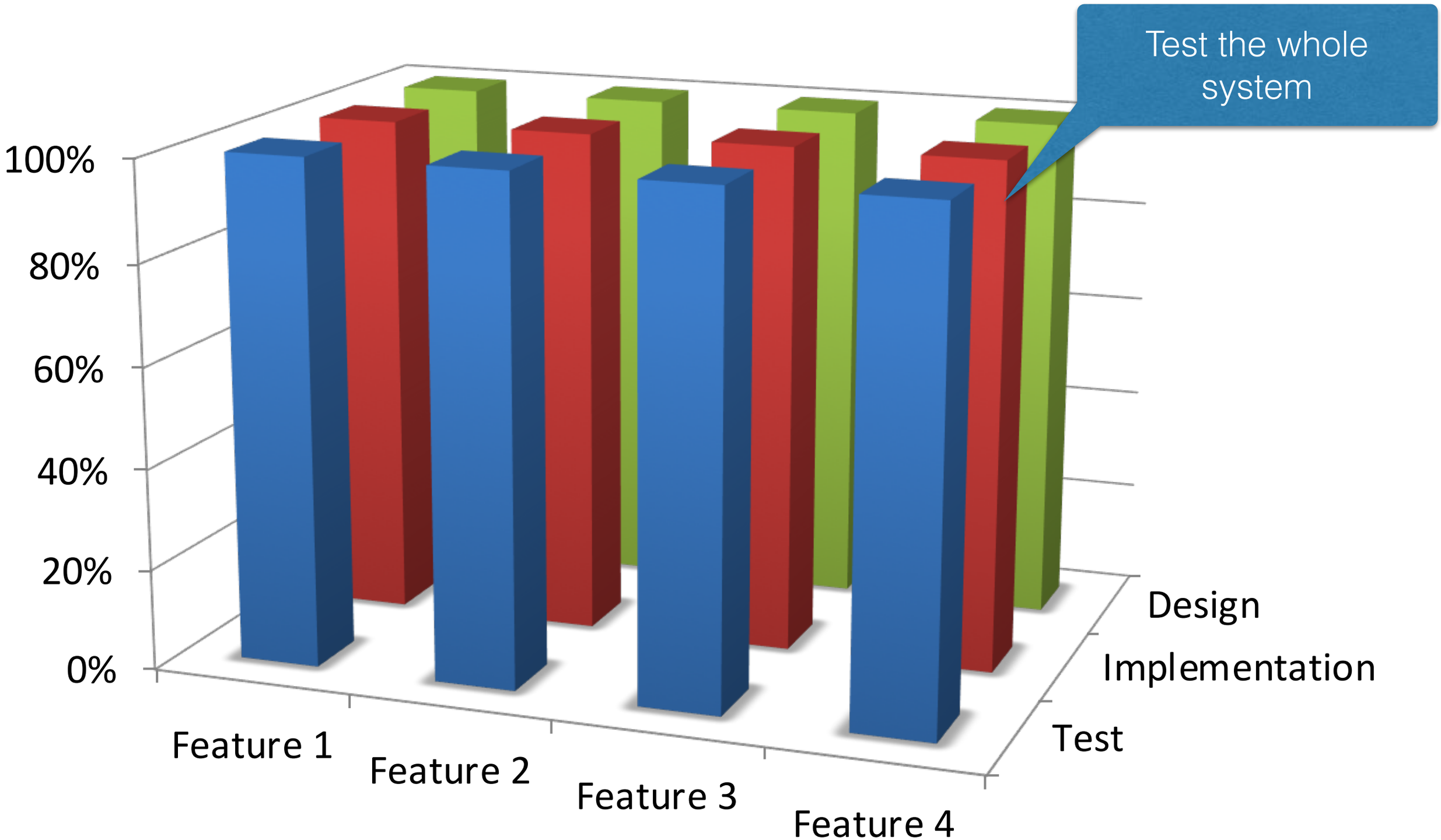
src: https://availagility.co.uk/2009/12/22/fidelity-the-lost-dimension-of-the-iron-triangle/

Waterfall model



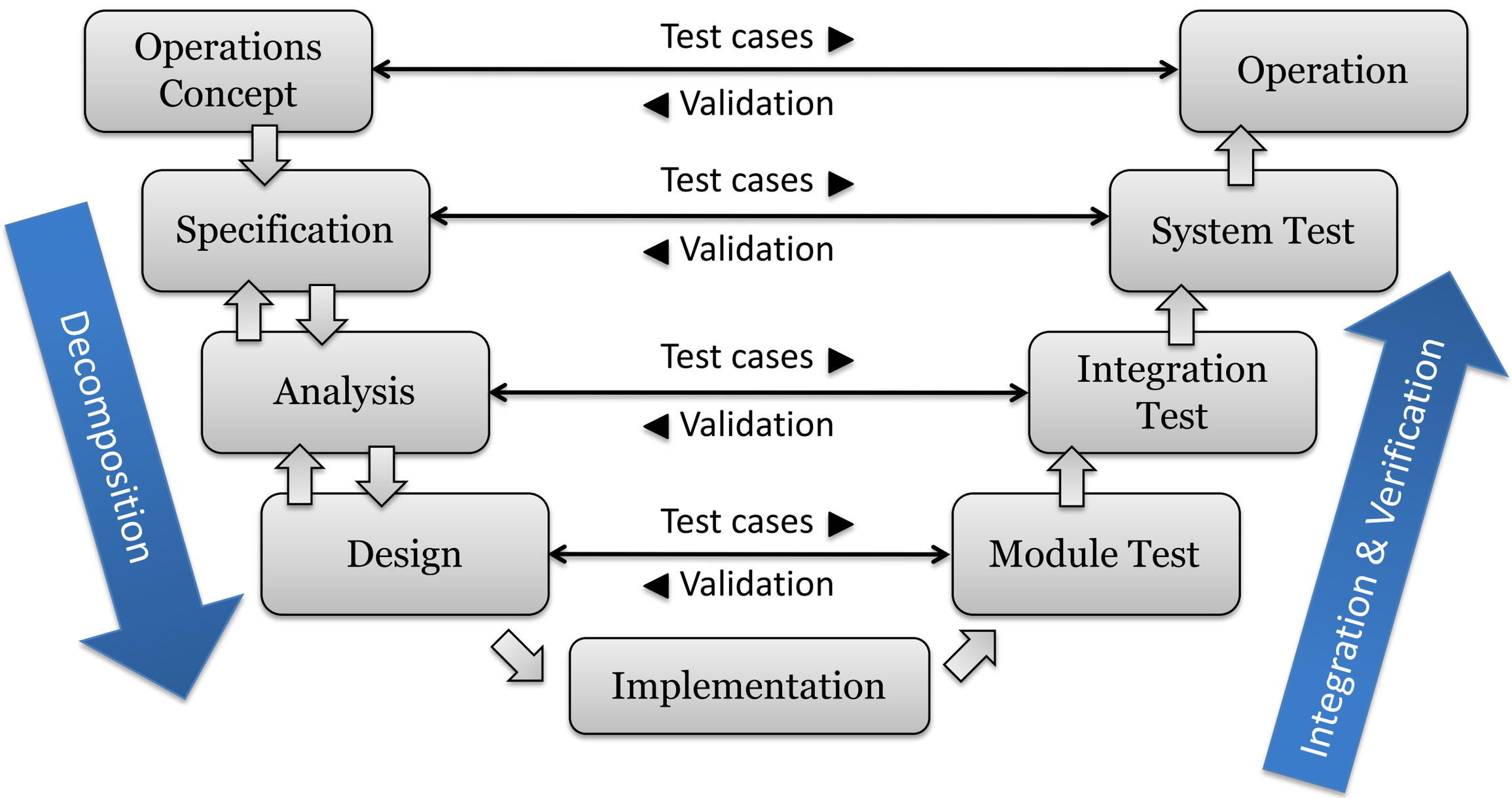
src: https://availagility.co.uk/2009/12/22/fidelity-the-lost-dimension-of-the-iron-triangle/

Waterfall model



src: https://availagility.co.uk/2009/12/22/fidelity-the-lost-dimension-of-the-iron-triangle/

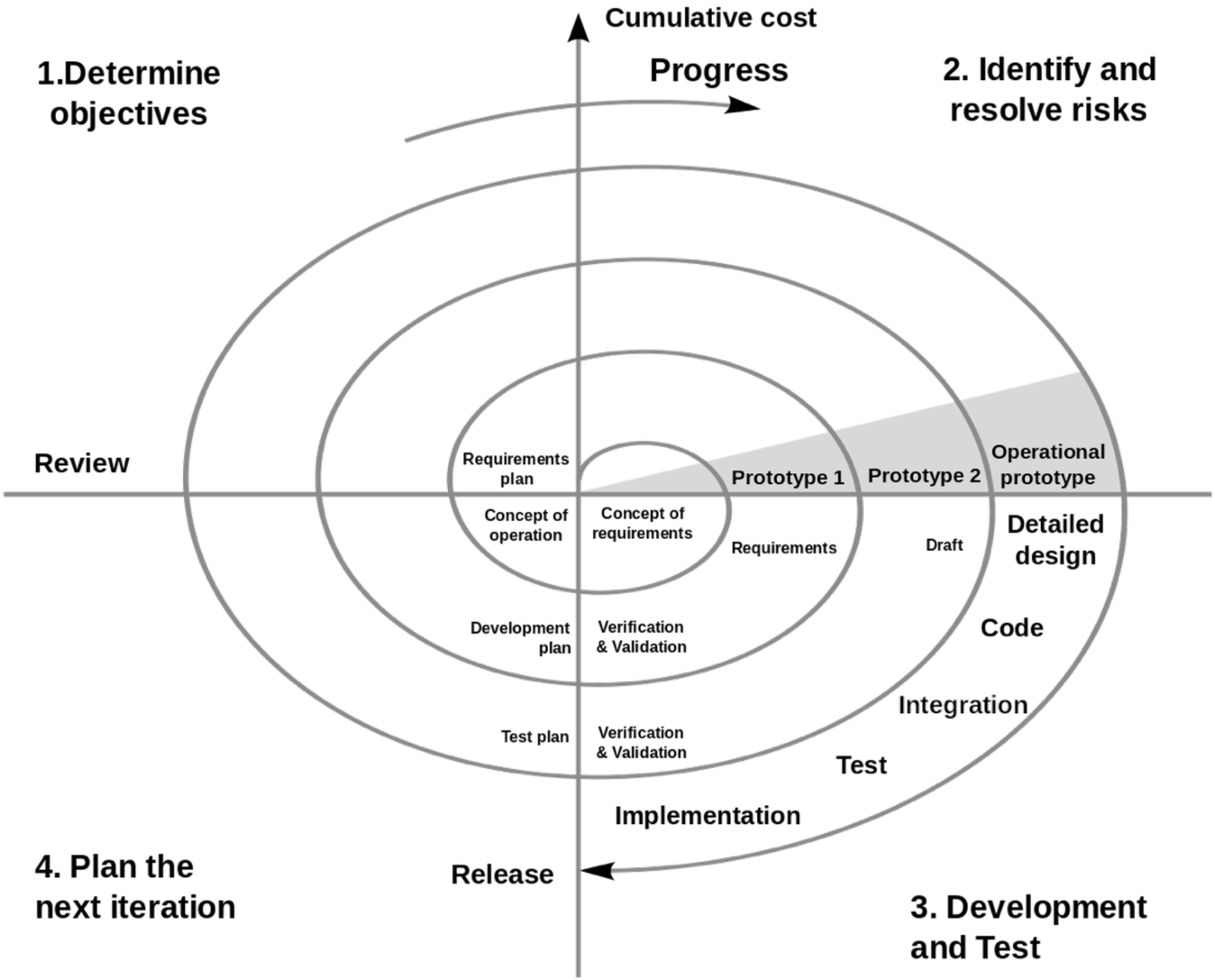
V-model


Operations Concept
Test cases ▶
◀ Validation
Operation
Specification
Test cases ▶
◀ Validation
System Test
Analysis
Test cases ▶
◀ Validation
Integration Test
Design
Test cases ▶
◀ Validation
Module Test
Implementation
Decomposition
Integration & Verification

V-model

- Applicability
  - Automotive industry
  - Medical and other high-criticality devices
  - Integrating systems and software development
  - Governmental projects with strict documentation standards
  - Hierarchical development projects with subcontractors
  - Large engineering projects spread over several development sites
- Restrictions
  - Change requests due to modified requirements could not be adapted easily
  - Sequential development

Spiral model

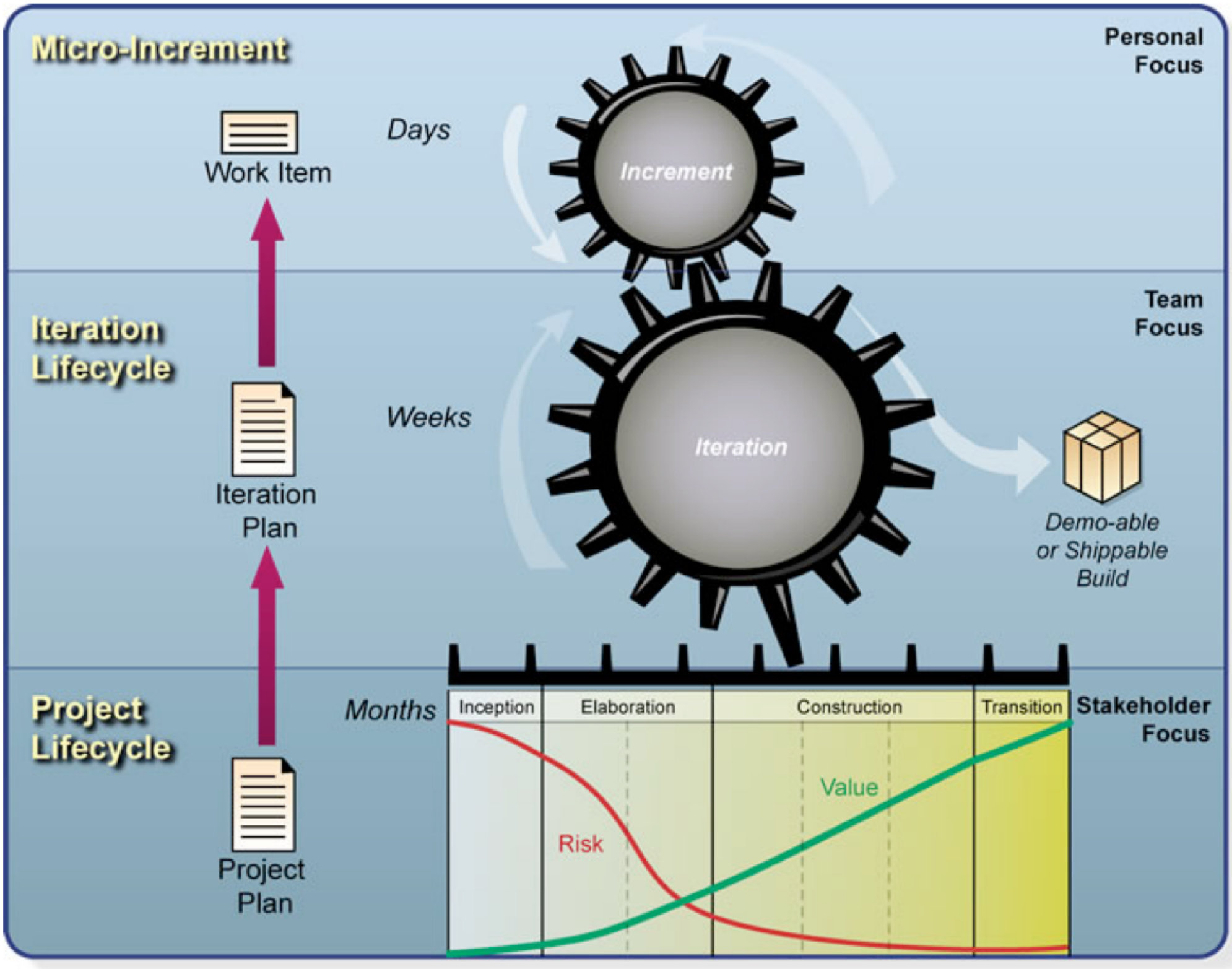


http://commons.wikimedia.org/wiki/File:Spiral_model_%28Boehm,_1988%29.svg

Spiral model

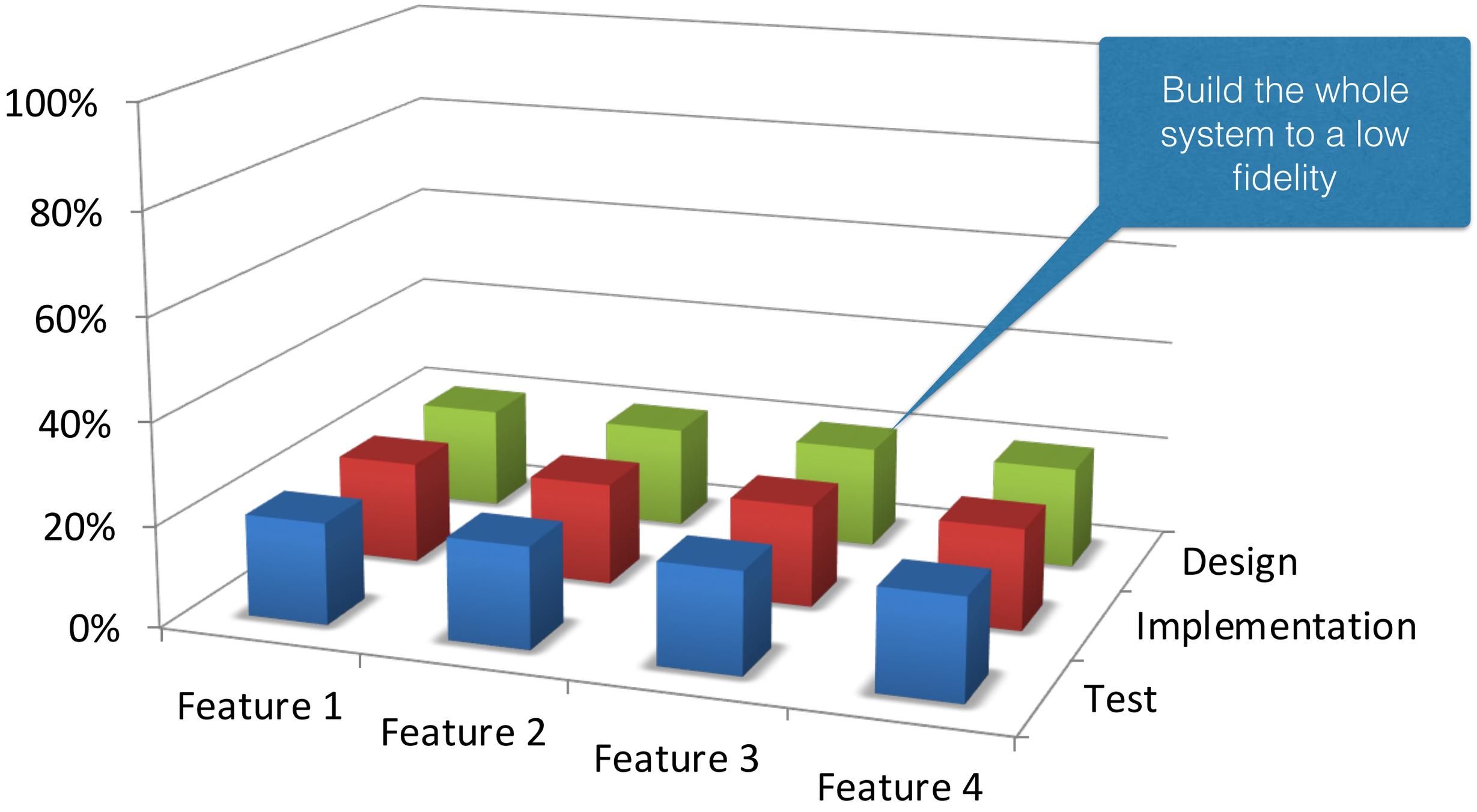
1. Objective setting
   - Specific objectives for the phase are identified
2. Risk assessment and reduction
   - Risks are assessed and activities put in place to reduce the key risks
3. Development and validation
   - A development model for the system is chosen which can be any of the generic models (“process model generator” [Boehm, 2000])
4. Planning
   - The project is reviewed and the next phase of the spiral is planned

Incremental / iterative model



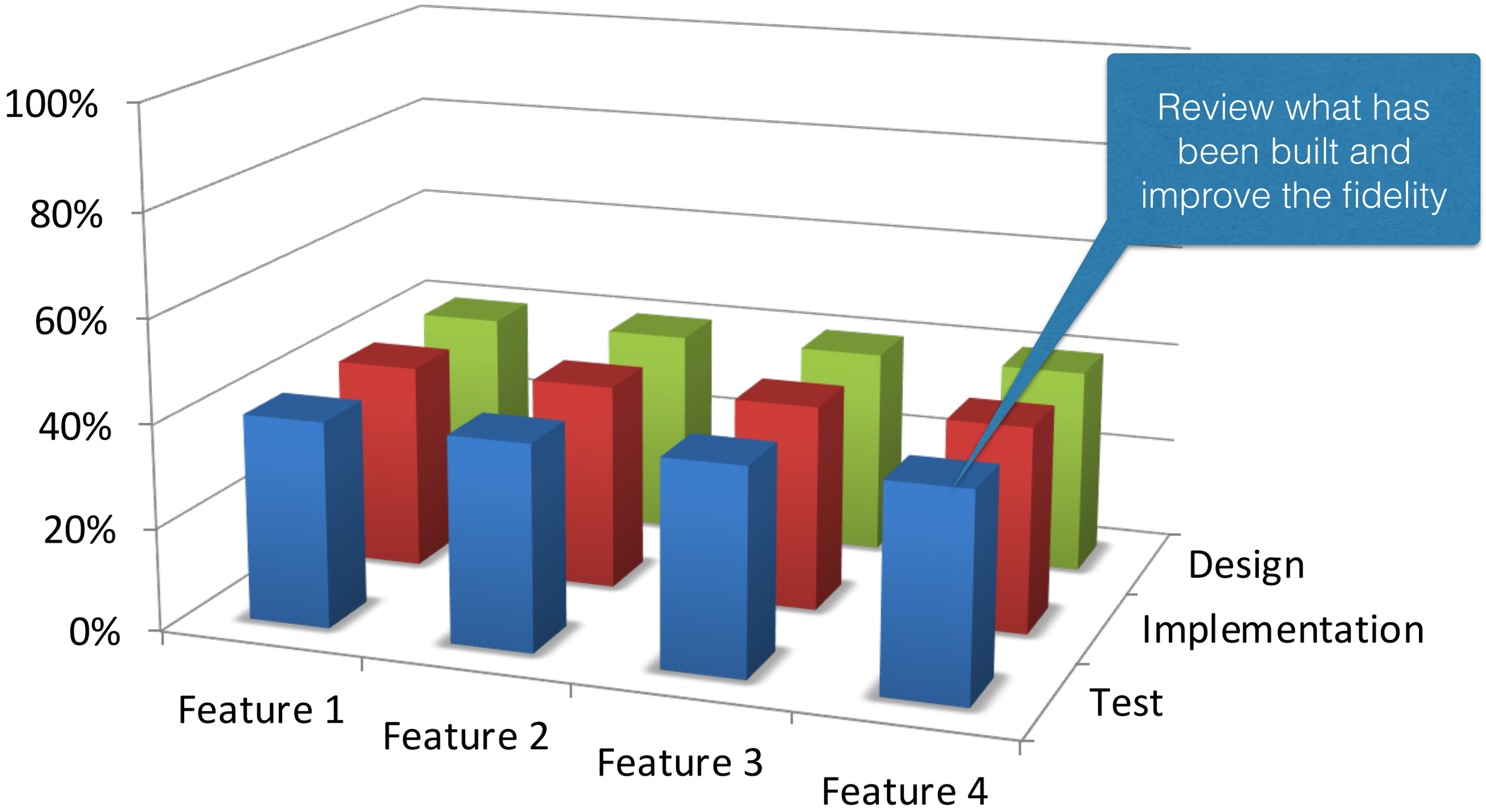
src: http://epf.eclipse.org/wikis/openup/

Iterative – Iteration 1



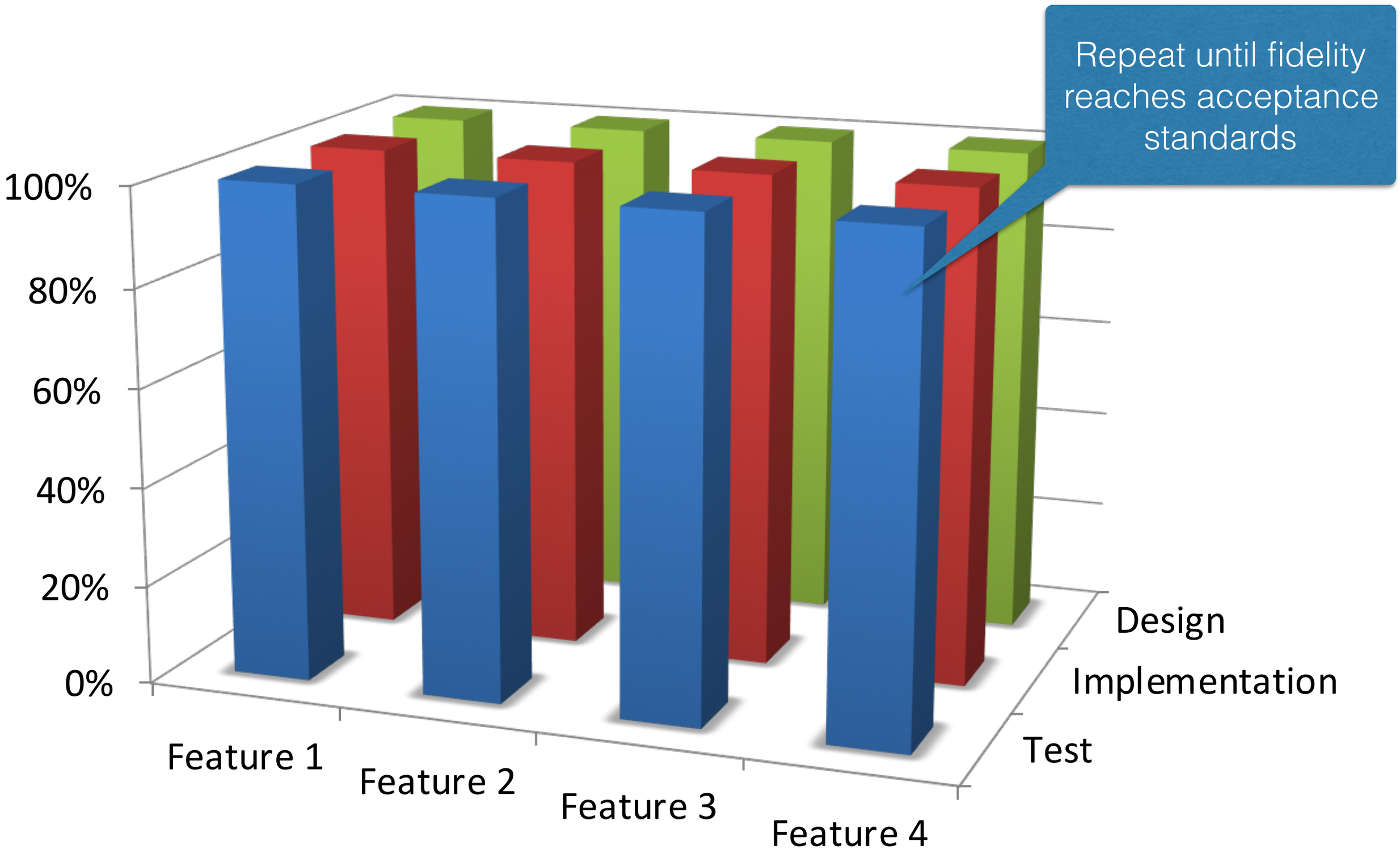
src: https://availagility.co.uk/2009/12/22/fidelity-the-lost-dimension-of-the-iron-triangle/

Iterative – Iteration 2



src: https://availagility.co.uk/2009/12/22/fidelity-the-lost-dimension-of-the-iron-triangle/

Iterative – Iteration 3



src: https://availagility.co.uk/2009/12/22/fidelity-the-lost-dimension-of-the-iron-triangle/

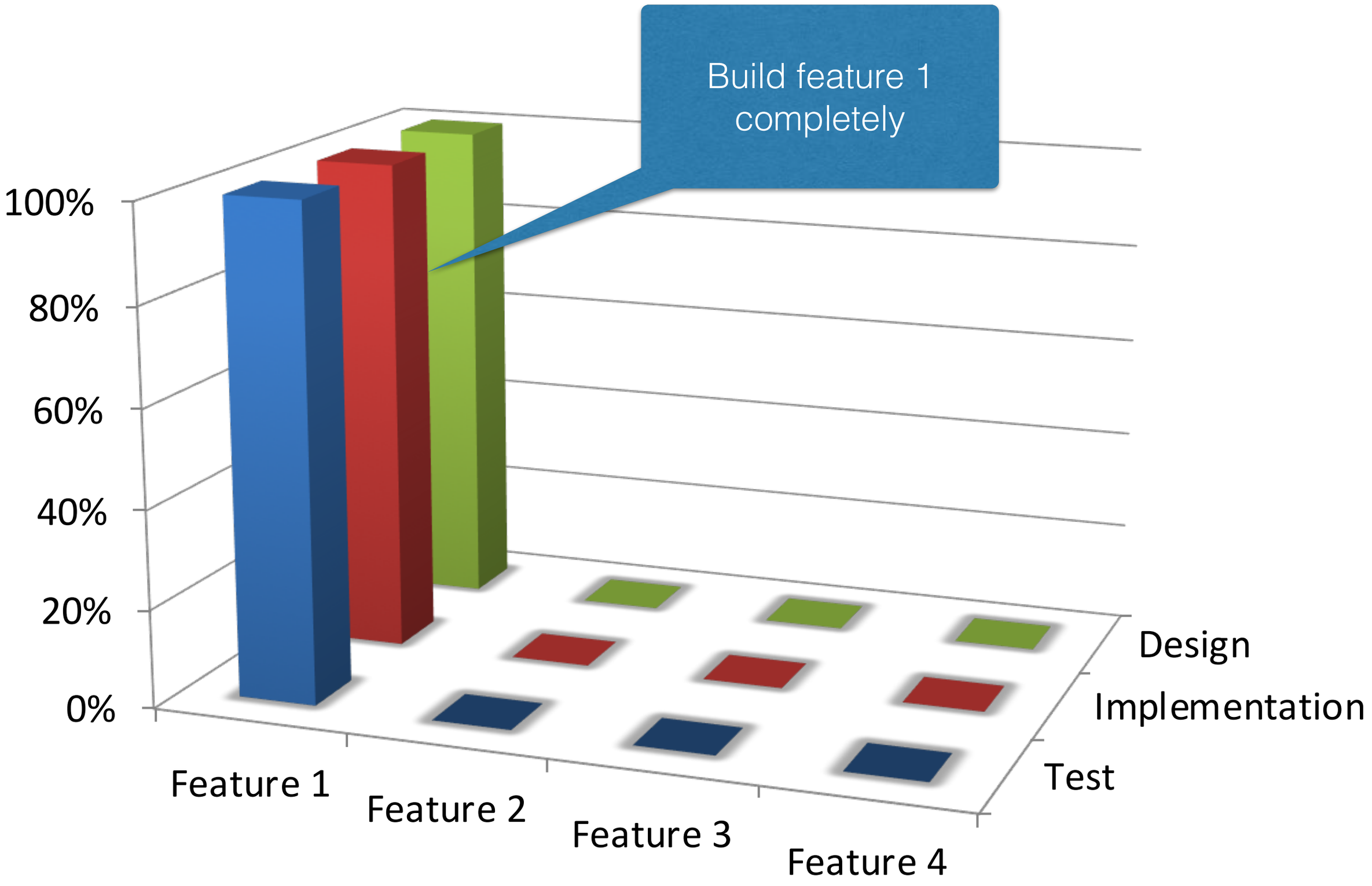
Iteration Models

- System requirements always evolve during a project
- Iterations are part of larger development projects
- Iterations require additional time for refactoring of the architecture and the code!

That's the "evolution" part!

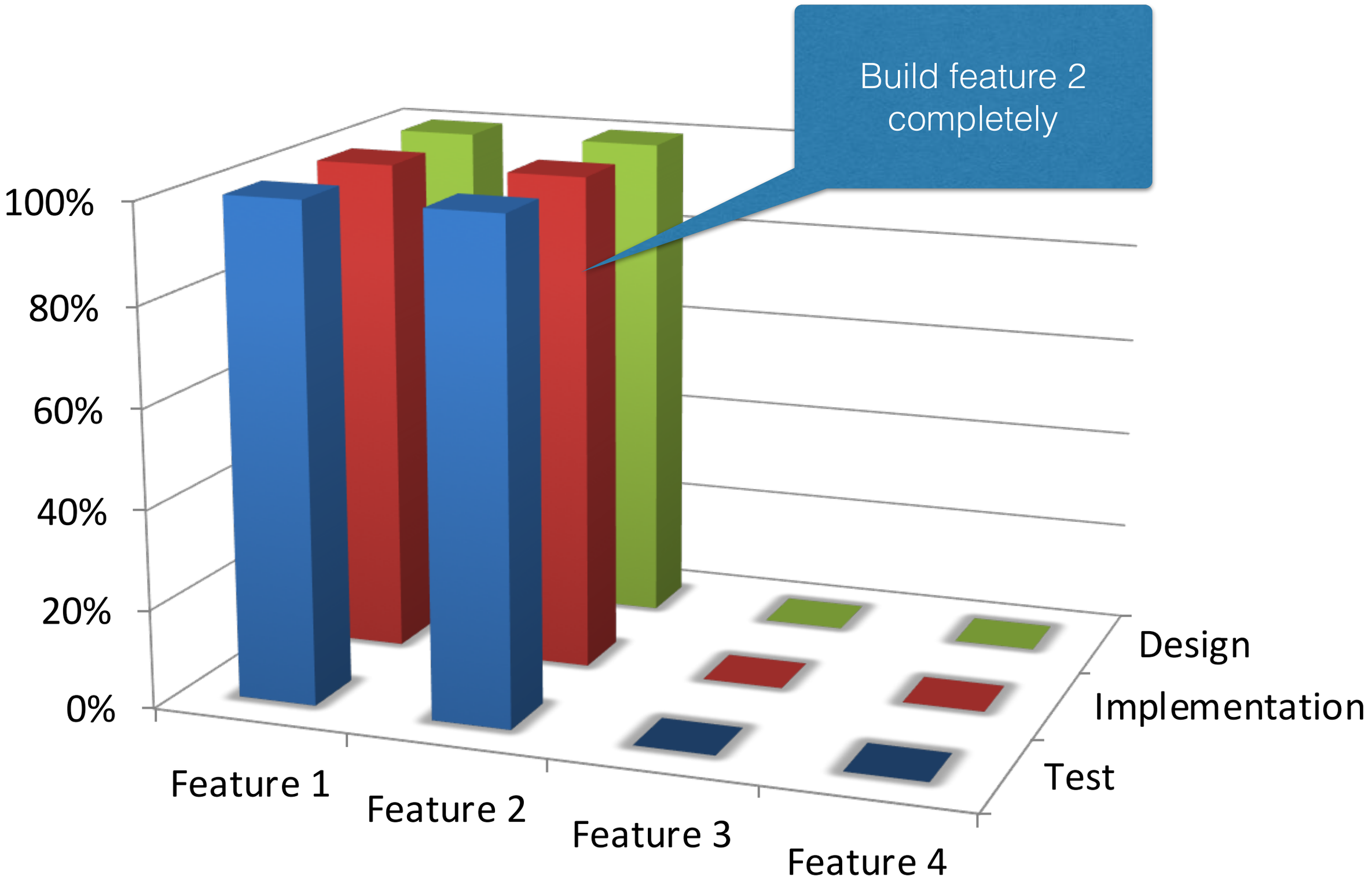
- Pure iteration models do not prescribe delivery after each iteration!

Incremental – increment 1



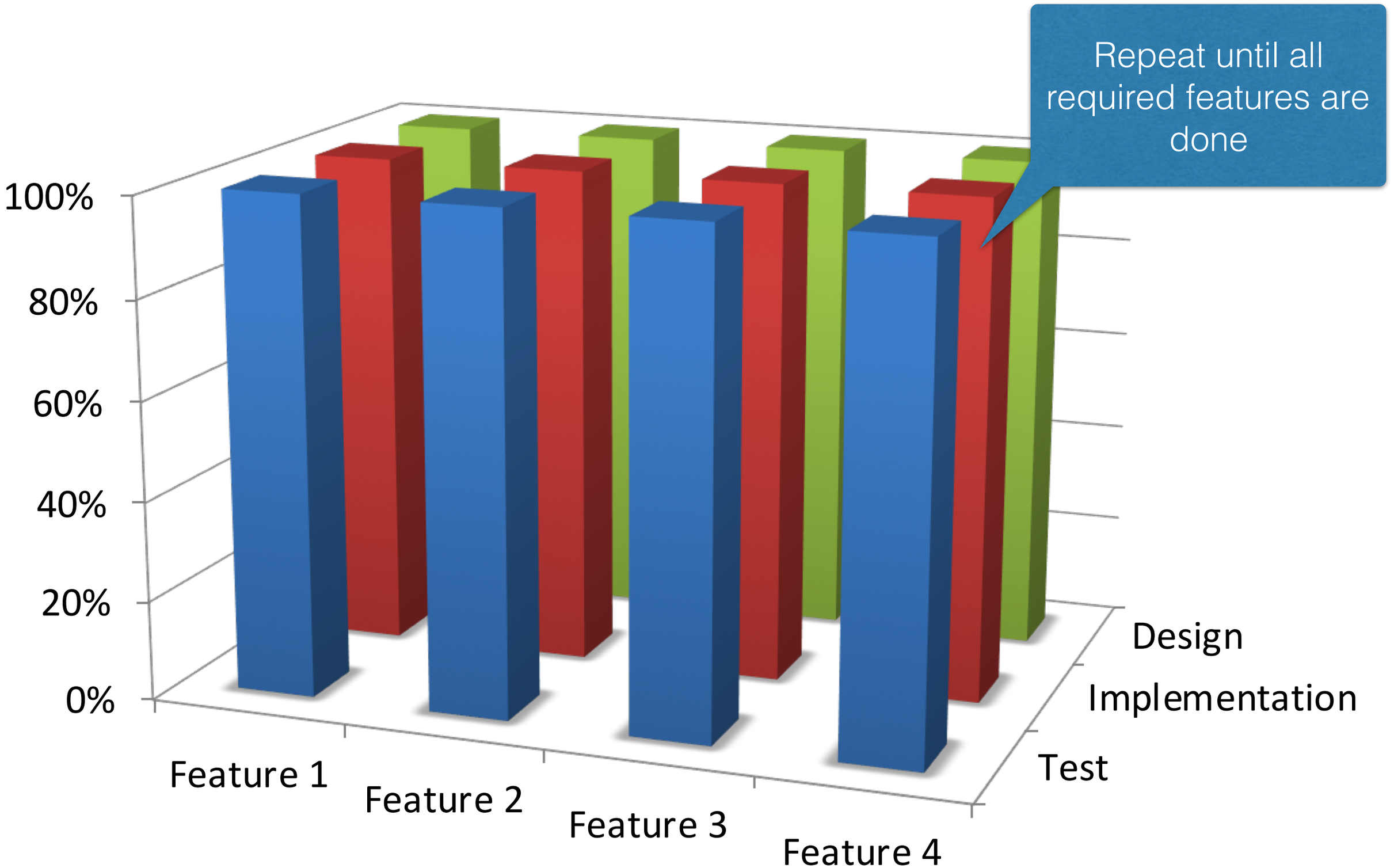
src: https://availagility.co.uk/2009/12/22/fidelity-the-lost-dimension-of-the-iron-triangle/

Incremental – increment 2



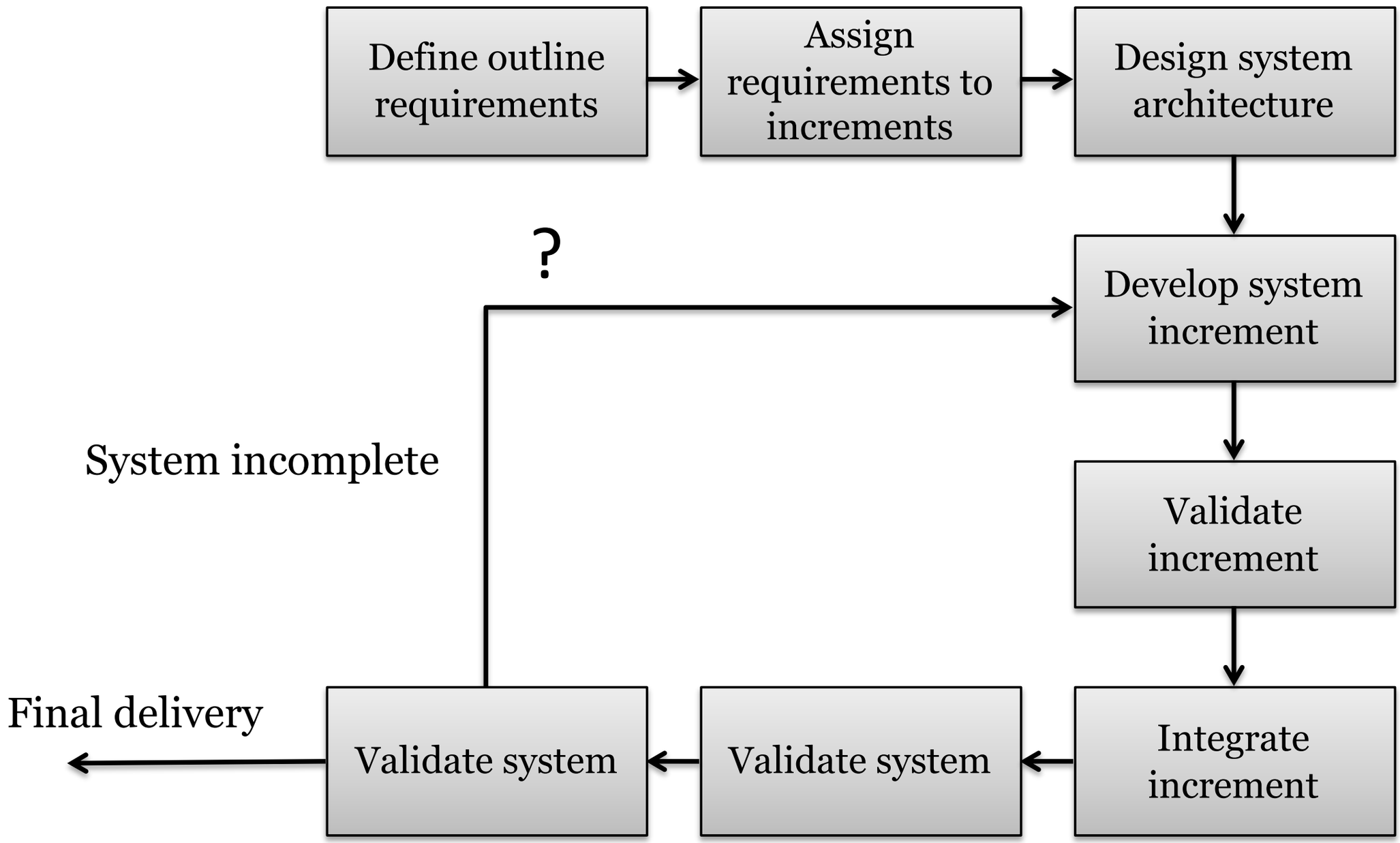
src: https://availagility.co.uk/2009/12/22/fidelity-the-lost-dimension-of-the-iron-triangle/

Incremental – increment *n*

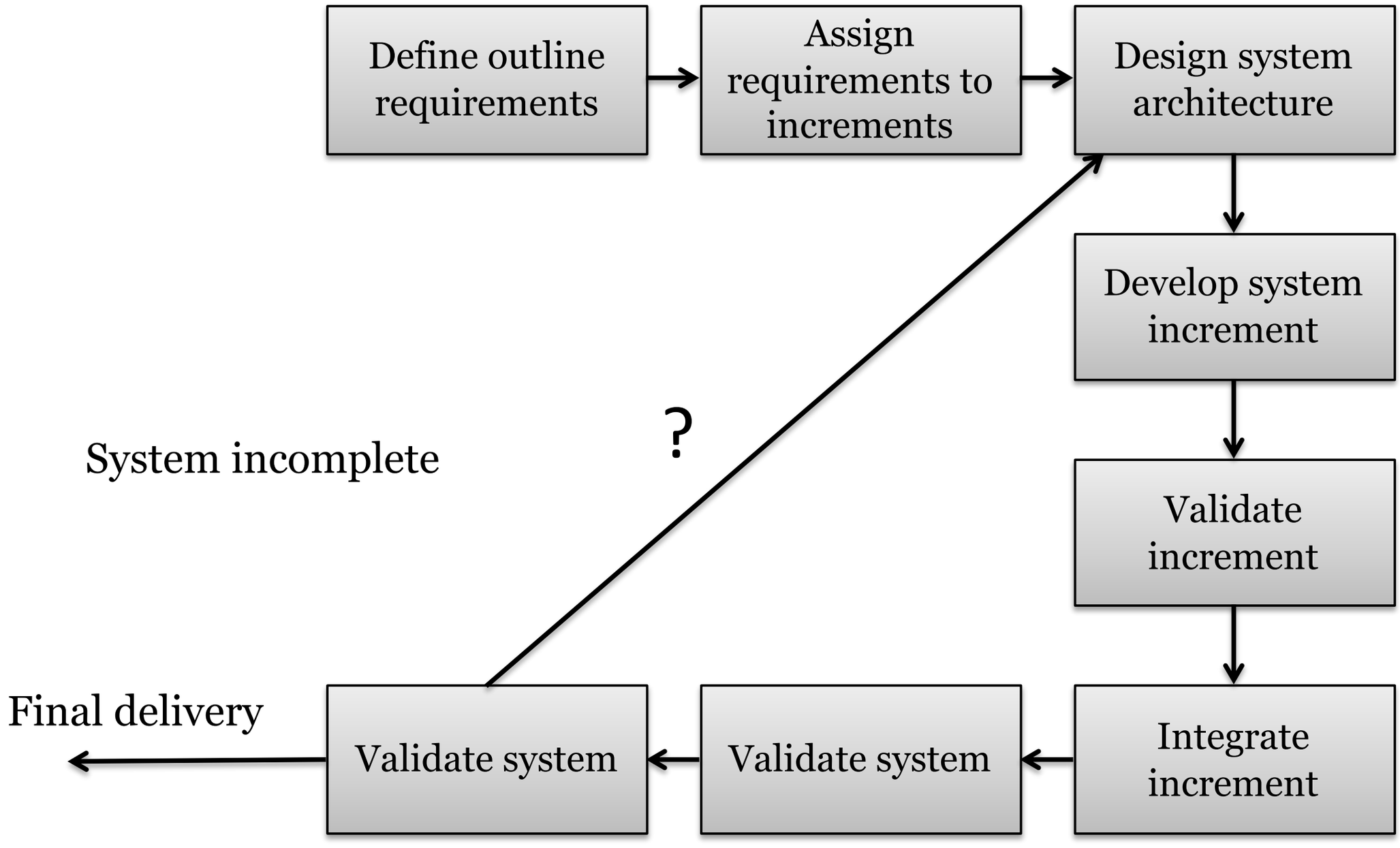


src: https://availagility.co.uk/2009/12/22/fidelity-the-lost-dimension-of-the-iron-triangle/

Incremental delivery



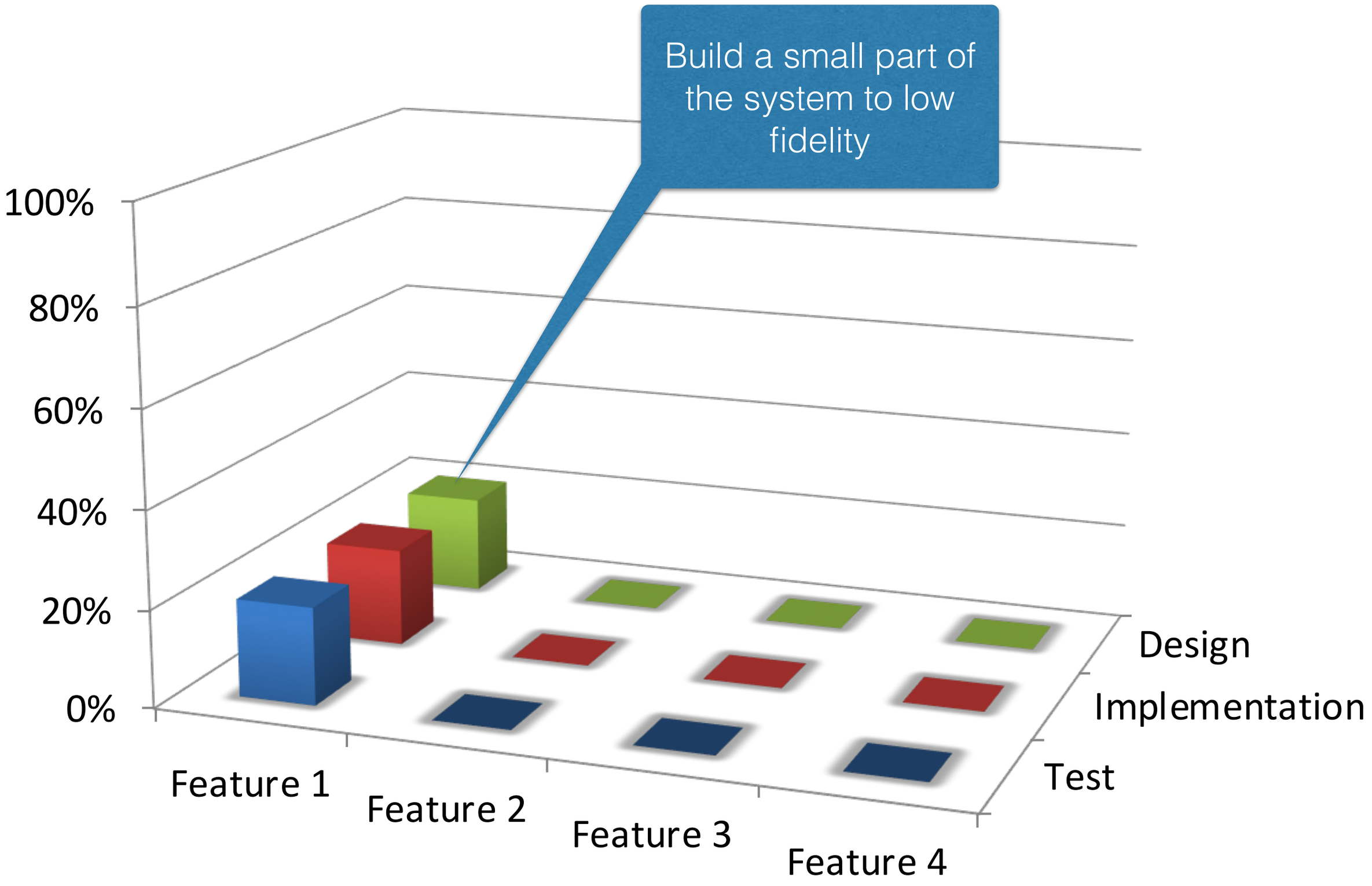
Incremental delivery



Incremental delivery

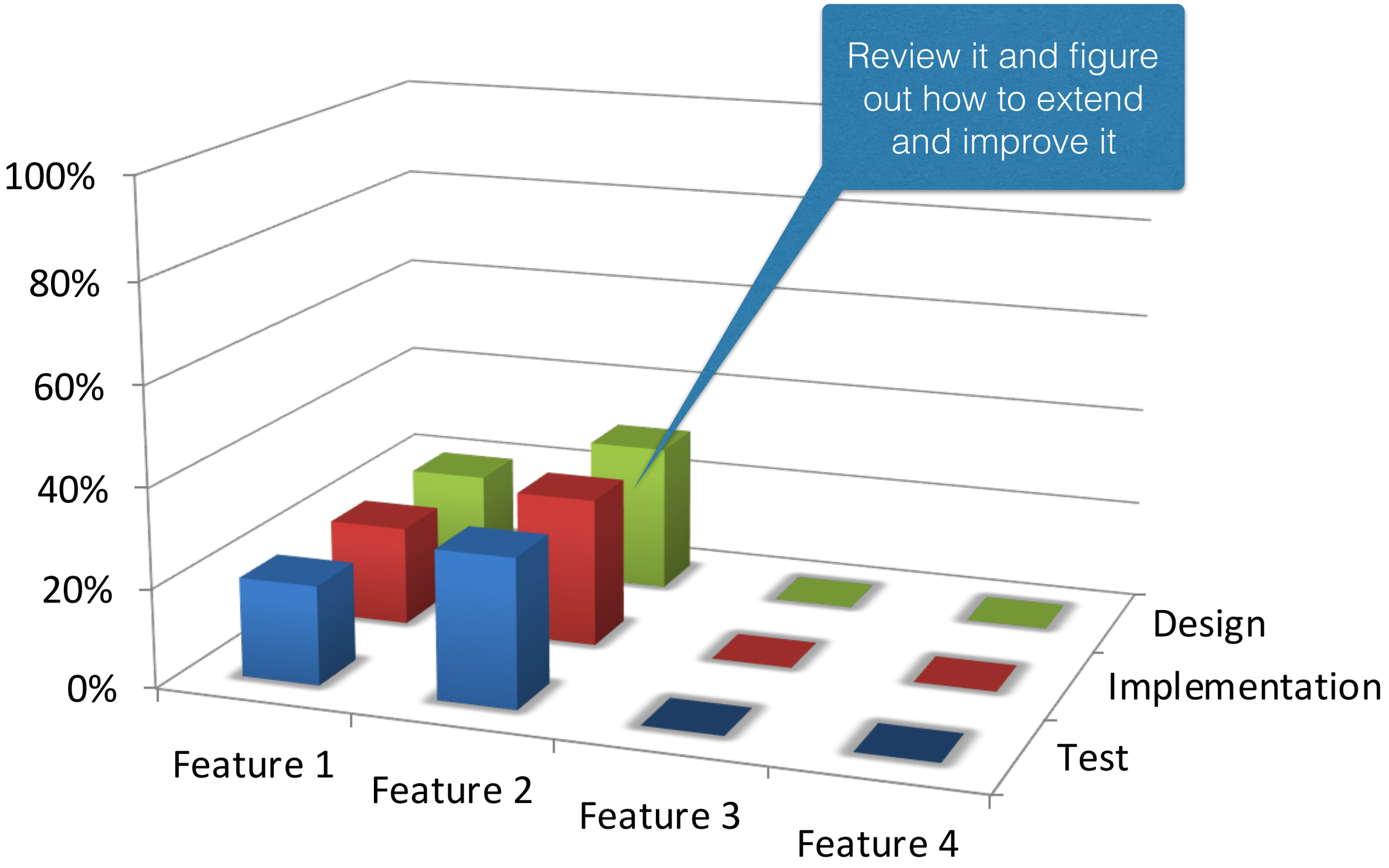
- Customer value can be delivered with each increment so system functionality is early available for customer's feedback
- Early increments act as prototype to help elicit requirements for later increments
- Reduced risk of project failure
- The highest priority system services tend to receive the most testing

Iterative–incremental - iteration 1



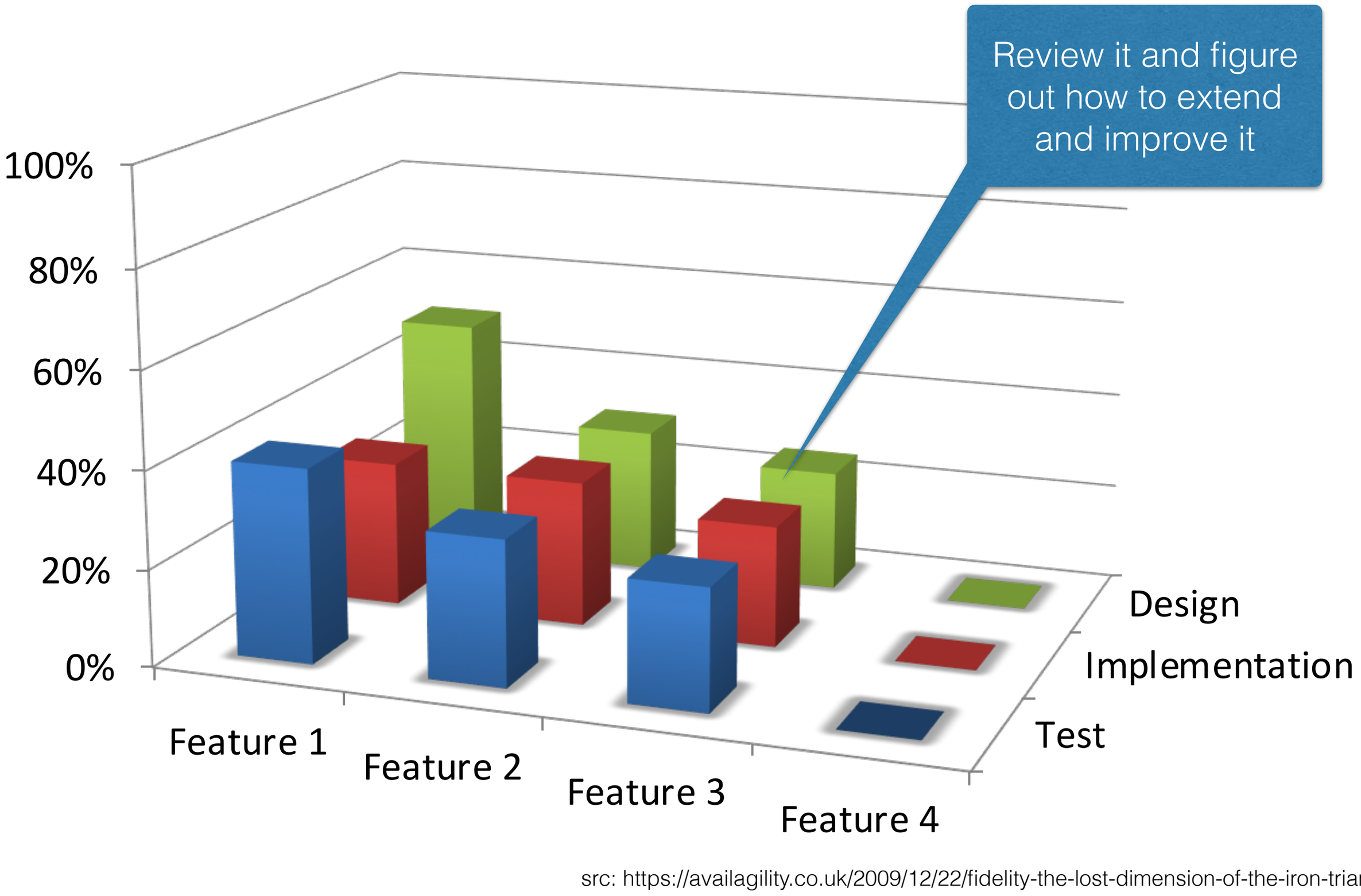
src: https://availagility.co.uk/2009/12/22/fidelity-the-lost-dimension-of-the-iron-triangle/

Iterative–incremental - iteration 2



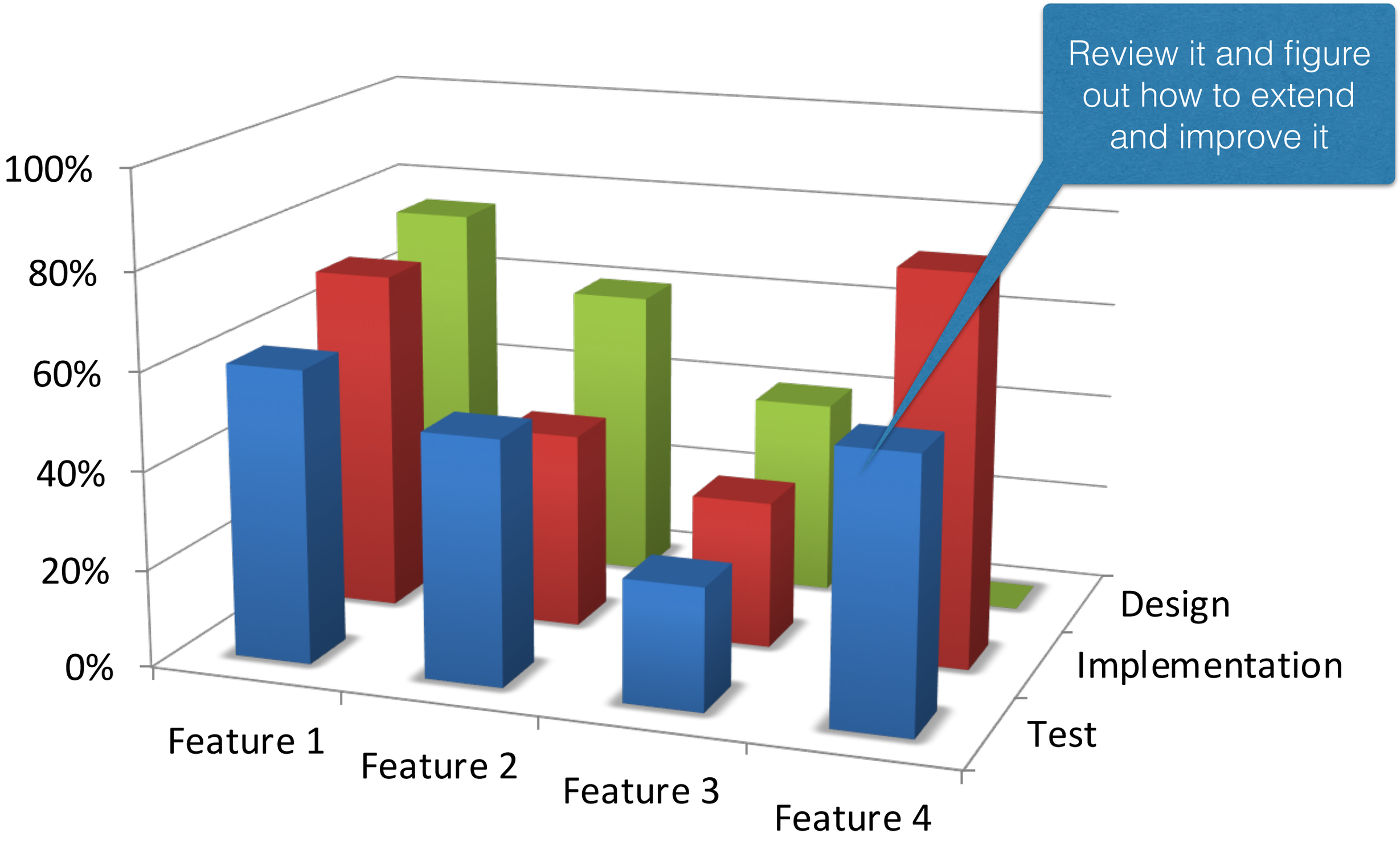
src: https://availagility.co.uk/2009/12/22/fidelity-the-lost-dimension-of-the-iron-triangle/

Iterative–incremental - iteration 3



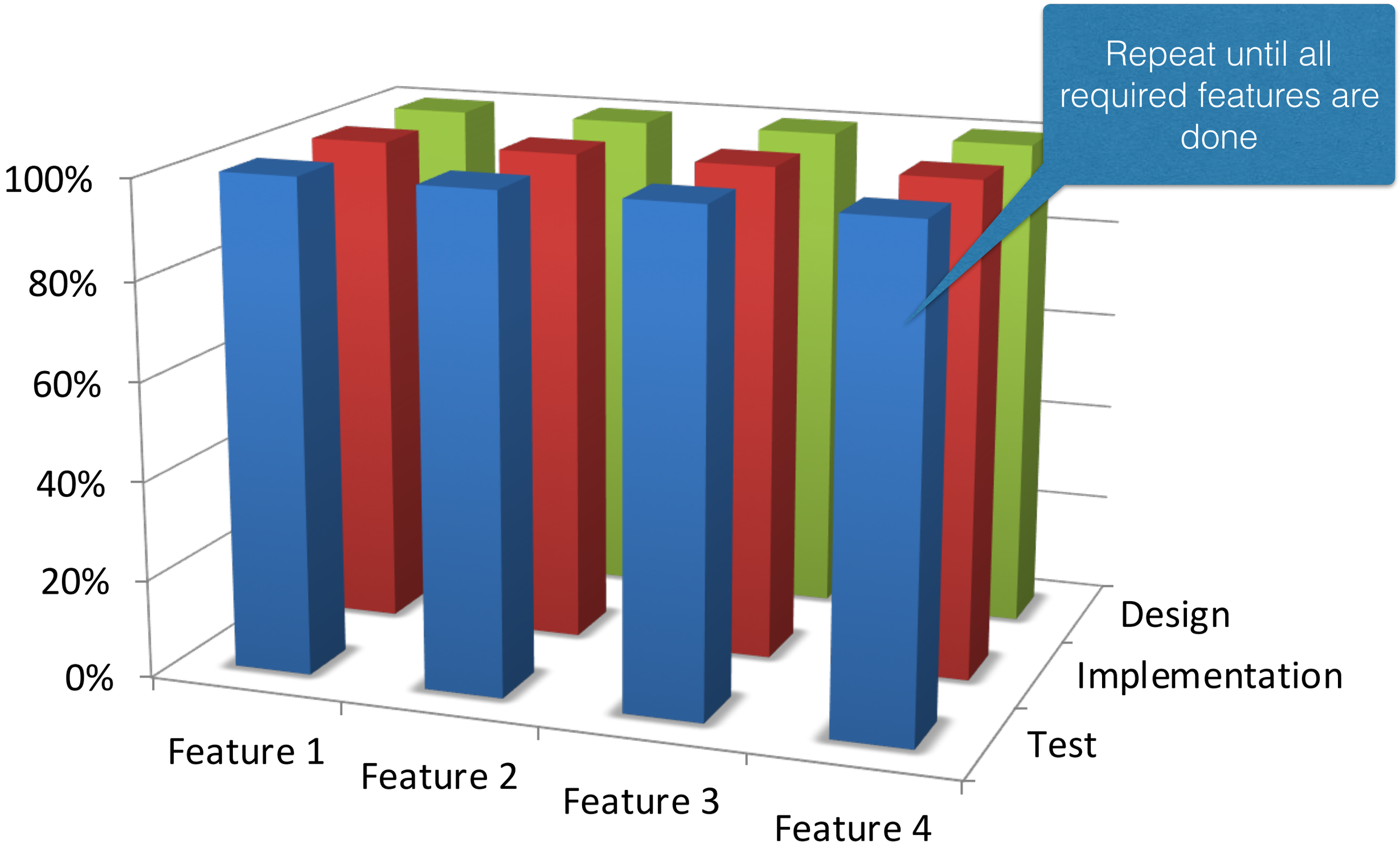
src: https://availagility.co.uk/2009/12/22/fidelity-the-lost-dimension-of-the-iron-triangle/

Iterative–incremental - iteration *k*



src: https://availagility.co.uk/2009/12/22/fidelity-the-lost-dimension-of-the-iron-triangle/

Iterative–incremental - iteration *n*



src: https://availagility.co.uk/2009/12/22/fidelity-the-lost-dimension-of-the-iron-triangle/

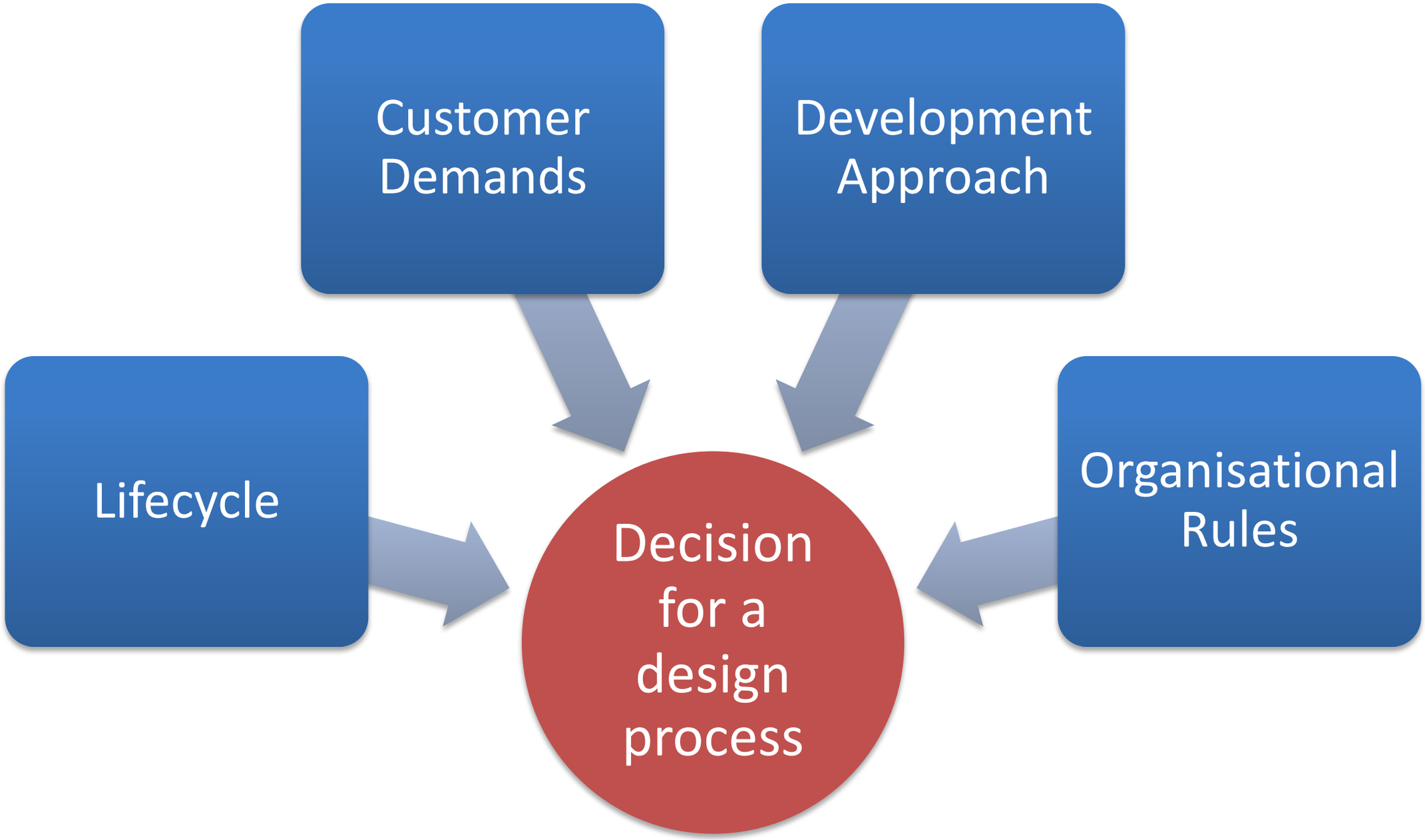
Lifecycles – Summary

- Different process lifecycles deliver different value at different points in time
- ”Best” lifecycle depends on the concrete project (see ”Selecting a Process”): there’s no ”one size fits all”

- Waterfall and V-Model driven by up-front requirement and design
- Spiral model is risk-driven
- Iterative, incremental, and iterative-incremental lifecycles deliver value in regular intervals

Lifecycle model and processes

Choosing a Software Process



Criteria for Process Selection

- Availability of the customer or a representative
- Procurement process of the customer
- External regulations
- Longevity of the system
- Scale of the system
- Criticality of the system
- Scale of the development effort
- Expected duration of the development effort
- Distribution of the development team
- Expected maintenance approach
- ...

Criteria for Process Selection: Teamwork

- Agile processes expect
  - team members volunteer to take up tasks that need to be done
  - team members to take responsibility for task completion
  - team members to take responsibility for communicating issues and problems
  - the team as a whole establishes norms and processes to help them achieve their goals
  - managers to create a work environment characterised by trust and continuous improvement
  - management to understand that the team is self-organised and responsible
  - that the development team is physically co-located

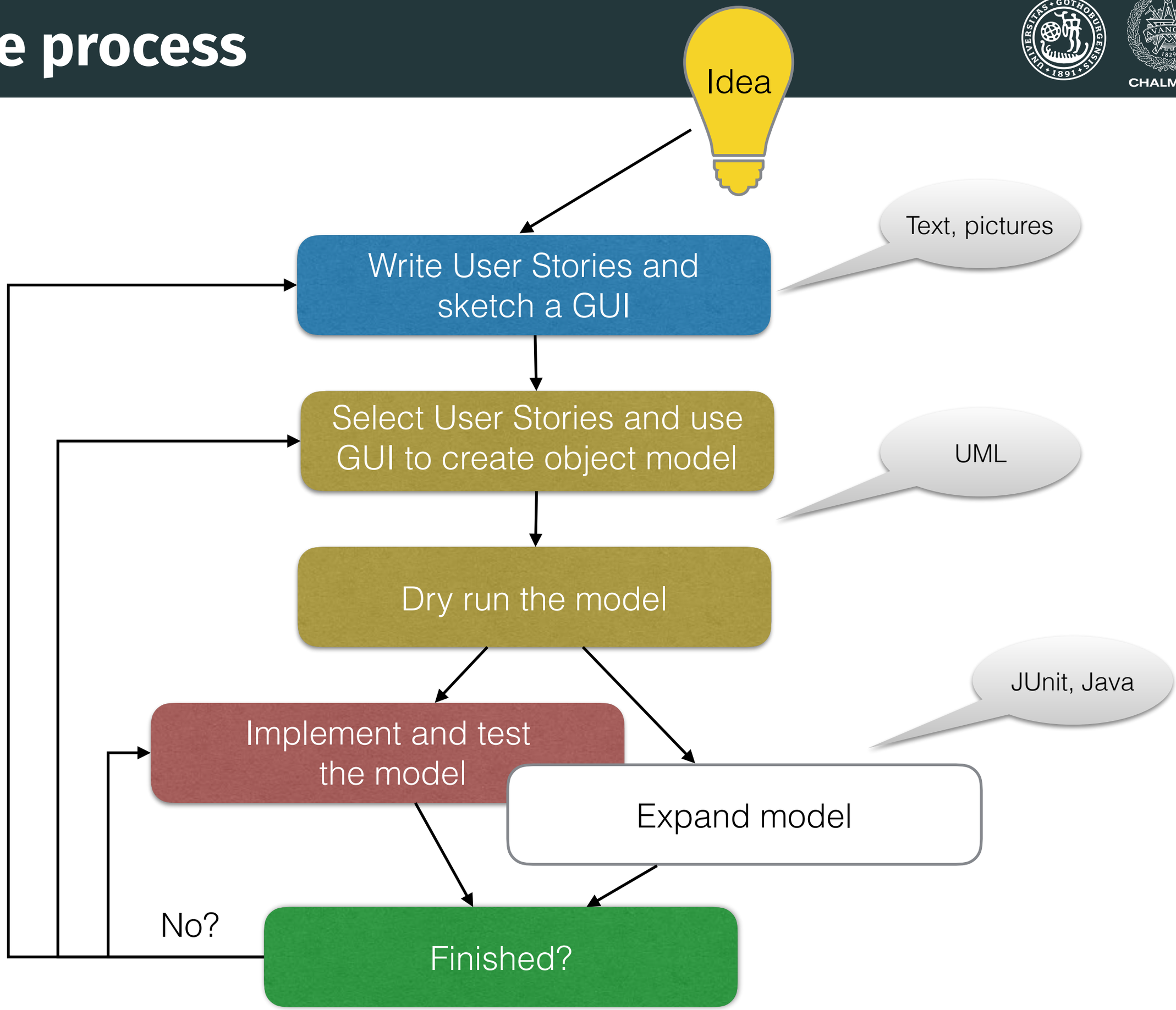
Selection Matrix

Factors	Waterfall	V-Shaped	Spiral	Iterative and Incremental	Agile Methodologies
Unclear User Requirement	Poor	Poor	Excellent	Good	Excellent
Unfamiliar Technology	Poor	Poor	Excellent	Good	Poor
Complex System	Good	Good	Excellent	Good	Poor
Reliable system	Good	Good	Excellent	Good	Good
Short Time Schedule	Poor	Poor	Excellent	Excellent	Excellent
Strong Project Management	Excellent	Excellent	Excellent	Excellent	Excellent
Cost limitation	Poor	Poor	Poor	Excellent	Excellent
Visibility of Stakeholders	Good	Good	Excellent	Good	Excellent
Skills limitation	Good	Good	Poor	Good	Poor
Documentation	Excellent	Excellent	Good	Excellent	Poor
Component reusability	Excellent	Excellent	Poor	Excellent	Poor

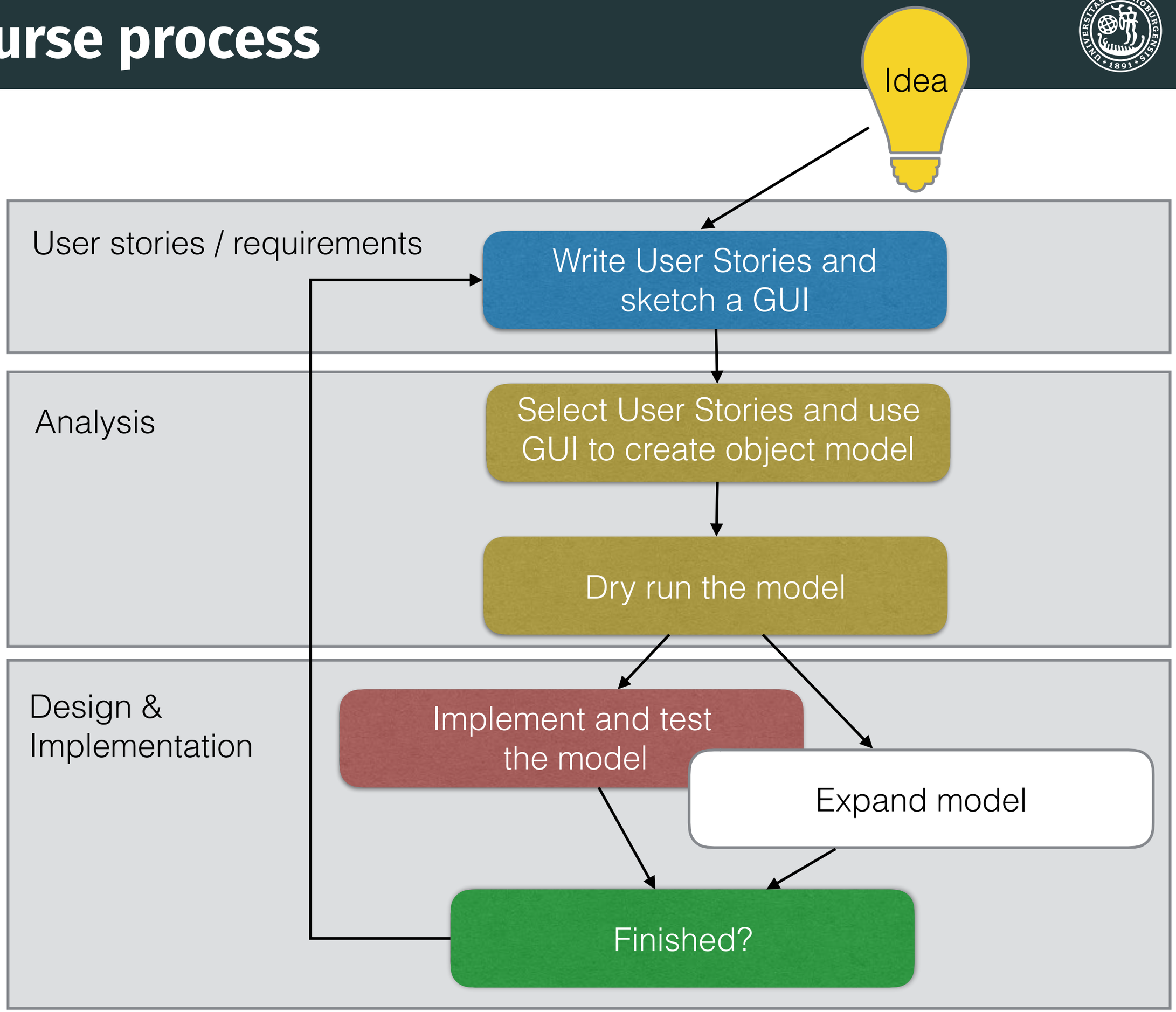
src: https://melsatar.wordpress.com/2012/03/21/choosing-the-right-software-development-life-cycle-model/

Course process

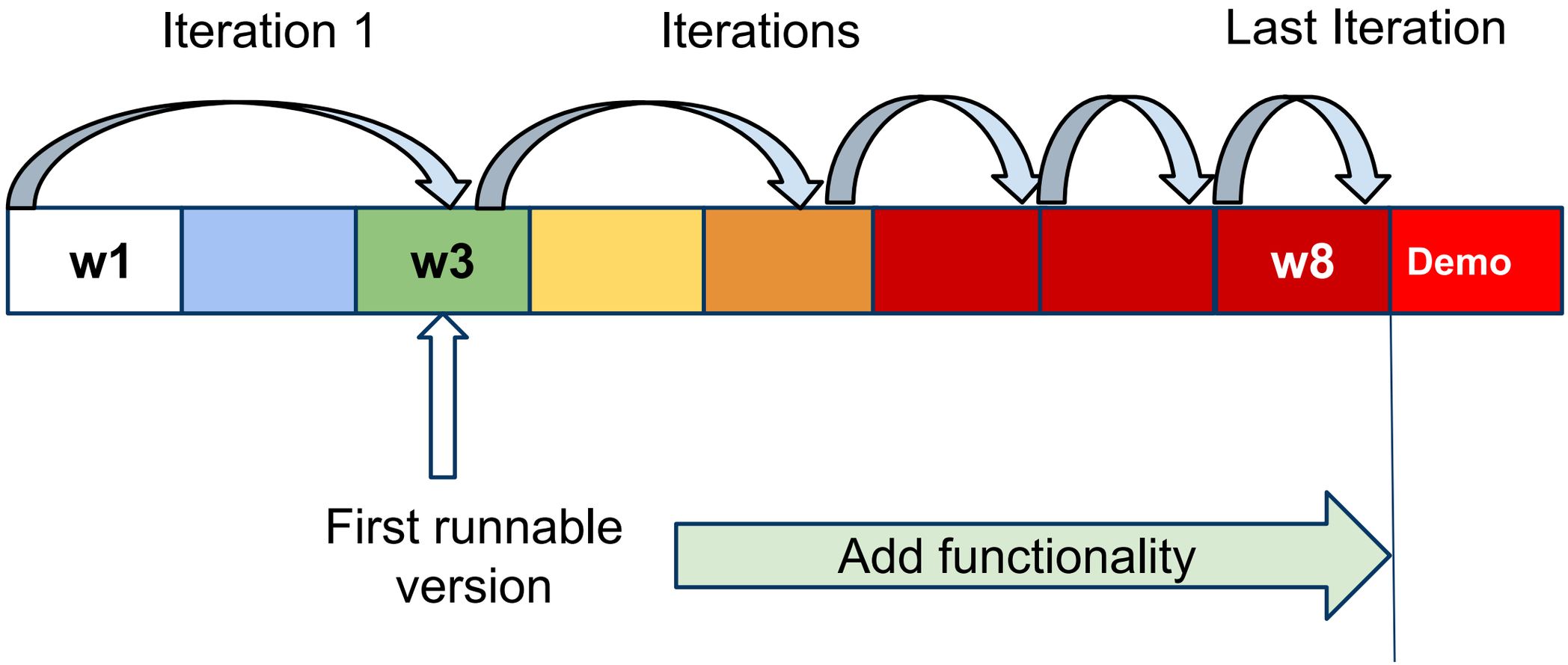
Course process



Course process



Iteration planning



Communication

- Effective communication is a fundamental requirement for software development
- Find a room with a whiteboard and gather
  - Don't spread the group!
- Use issue trackers, can't remember everything...
  - Most IDE have “TODO” lists (use // TODO in NetBeans, IntelliJ, ...)
  - Web based issue trackers
- Wordlist for definitions
  - Have experienced group members using same notion for different concepts! Confusion ... time lost...
  - If any ambiguity write down in wordlist (and/or as class comment)

Further reading

[Sommerville, 2011] Ian Sommerville. Software Engineering – 9th Edition. Pearson, 2011

[Royce, 1970] Winston Royce. Managing the Development of Large Software Systems. Proceedings of IEEE WESCON 26, pages 1–9, August 1970

[Forsberg & Mooz, 1991] Kevin Forsberg and Harold Mooz. The Relationship of System Engineering to the Project Cycle. In Proceedings of the First Annual Symposium of National Council on System Engineering, pages 57–65, October 1991

[Boehm, 1988] Barry Boehm. A Spiral Model of Software Development and Enhancement. ACM SIGSOFT Software Engineering Notes, ACM, 11(4):14-24, August 1986

[Boehm, 2000] Barry Boehm. Spiral Development: Experience, Principles,and Refinements. Special Report CMU/SEI-2000-SR-008, July 2000

[Henderson-Sellers & Ralyté, 2010] Brian Henderson-Sellers and Jolita Ralyté. Situational method engineering: State-of-the-art review, Journal of Universal Computer Science, 16 (2010), pp. 424–478.

[Gren et al., 2016] Lucas Gren, Richard Torkar, Robert Feldt. Group Development and Group Maturity when Building Agile Teams, submitted to JSS, 2016

Jeff Patton on iterative vs. incremental and what it means to produce something ”shippable”: http://jpattonassociates.com/dont_know_what_i_want/