# CSCI 3509
# Software Development Life Cycle

2. Software Development Life Cycle


Emil Abbasov

# Outline

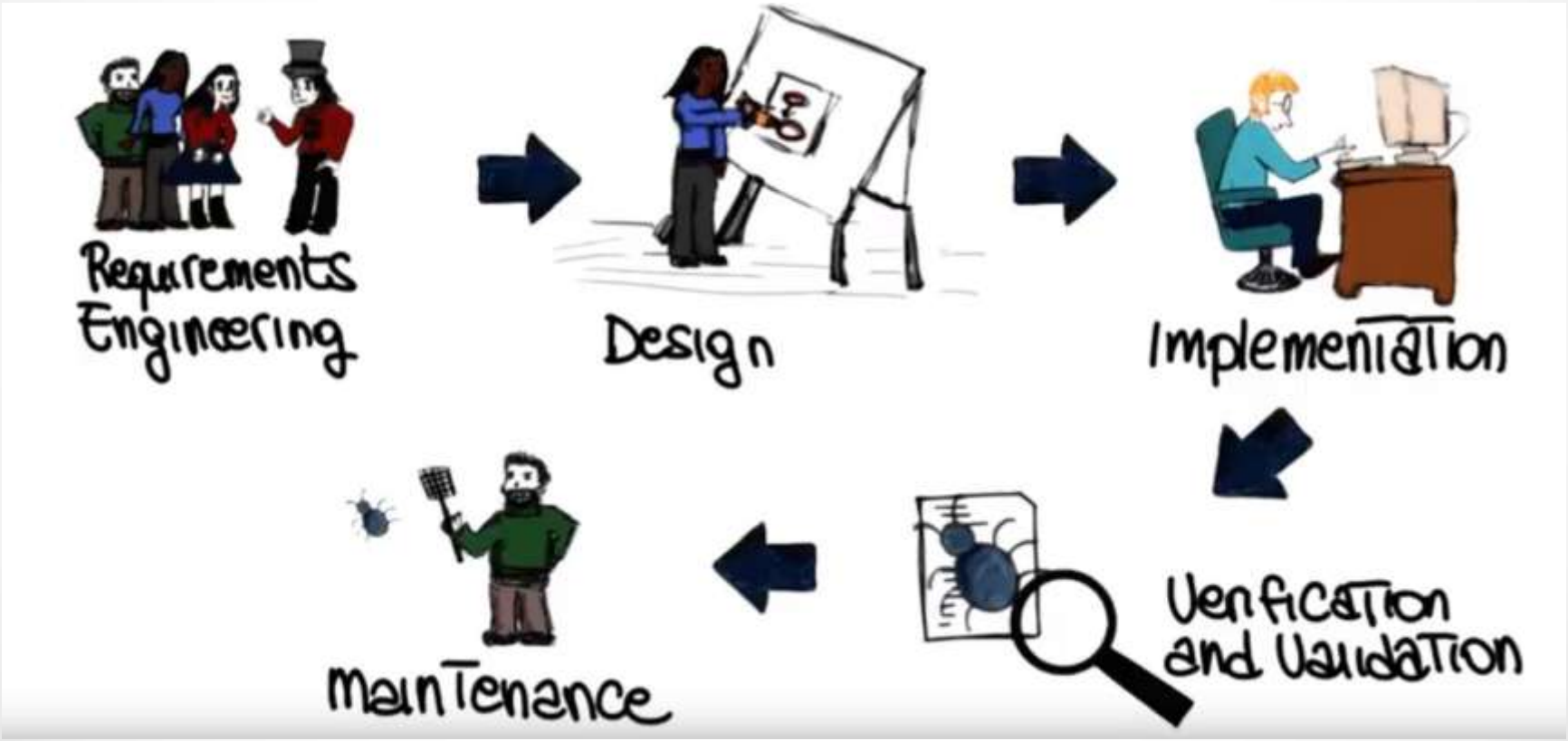
- Introduce several traditional SDLC models
  - Their main advantages and disadvantages

- Classic mistakes in Software Engineering

# What is SDLC?

- Answered by Prof. Barry Bohem (one of the fathers of SE)
- SDLC is a sequence of decisions that determine the history of your software
- It answers to the question "What should I do next?" and "How long should I do it for?"
- Since there are many ways in which you can make those decisions, it is important to understand which models are good for which situations
- Thus, the bottom line is that choosing the right lifecycle model is fundamental importance!

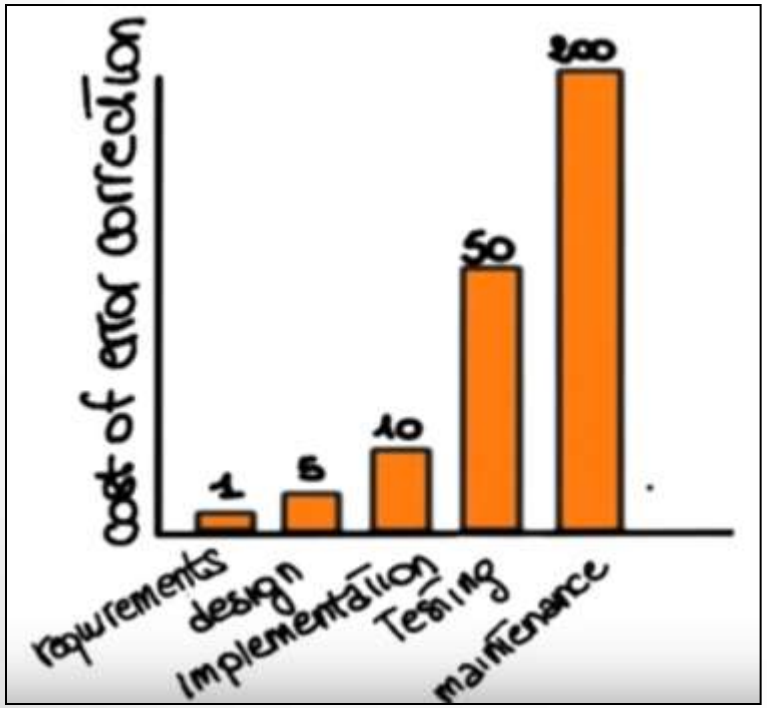
# Traditional Software Phases (remember?)

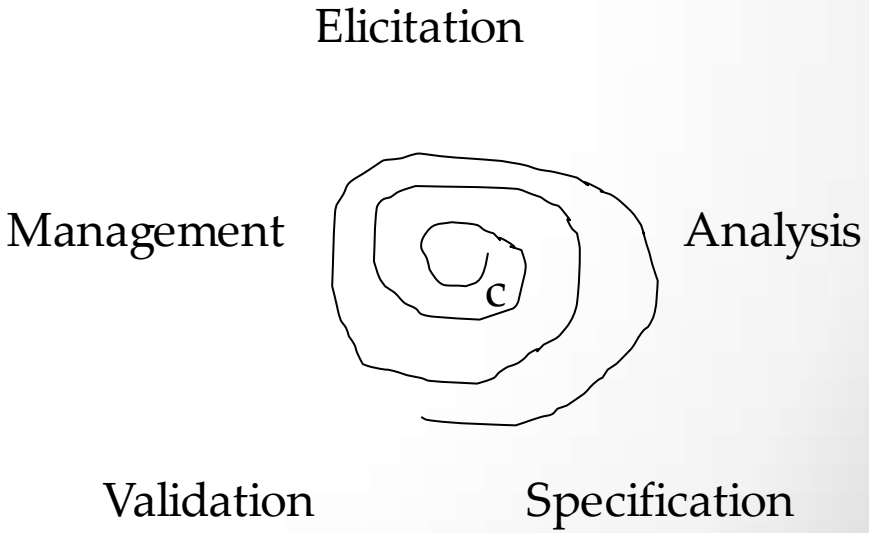


# Requirements Engineering


- Requirements Engineering is the process of establishing the needs of stakeholders (or idea authors) that are to be solved by software.

Cost of late correction



# Requirements Engineering


- Elicitation– Collection of requirements from stakeholders and other sources and can be done in various ways. We'll discuss some of them.
- Analysis – Involves study and deeper understanding of the collected requirements .
- Specification – Collective requirements are suitably represented, organized and save so that they can be shared. Also in his case, there are many ways to do this and we will see some of this ways when we talk about the requirements engineering in the dedicated lesson.
- Validation – Once the requirements have been specified, they can be validated to make sure that they're complete, consistent, no redundant and so on. So that, they've satisfied a set of importance properties, for requirements.
- Management – Accounts for changes to requirements during the lifetime of the project.

# Design


Design Activities

Architectural Design → Abstract Specification → Interface Design → Component Design → Data Structure → Algorithm Design

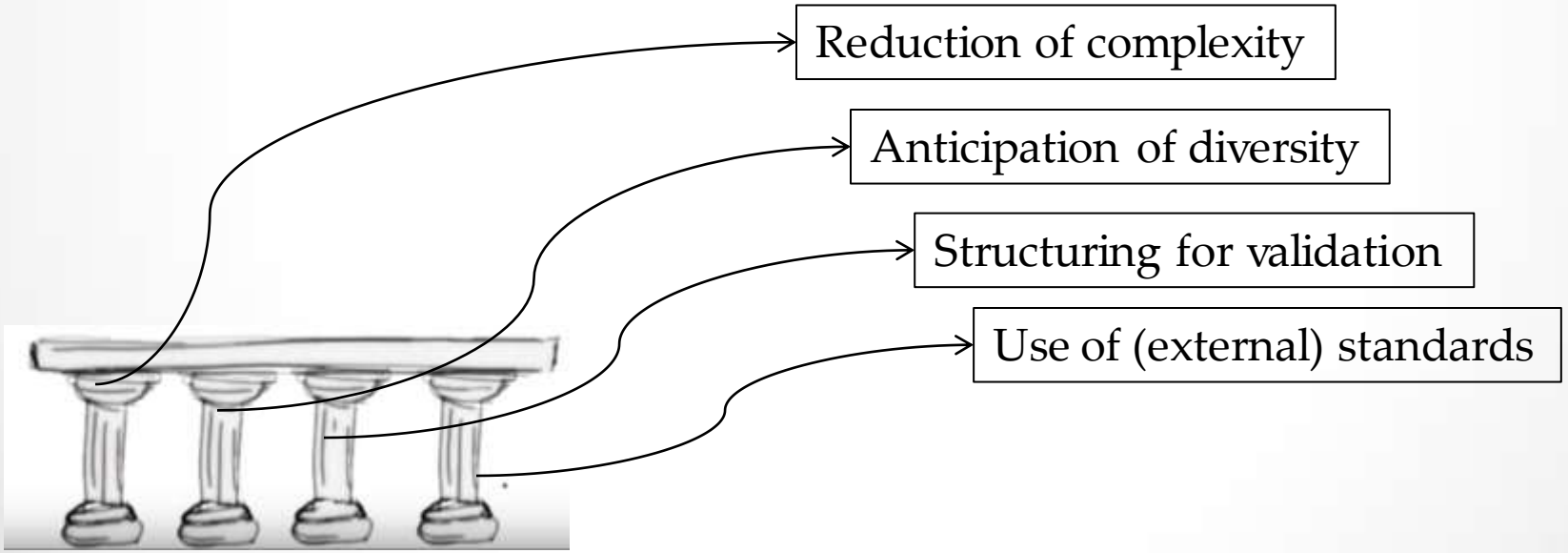
Note. This is just a possible list of activities. But you can also characterize design activities in many different ways. But the core idea is the same:
"Go from high level (Architectural Design) to low level view (Algorithm Design)".

System Structure | Software Specification | Interface Specification | Component Specification | Data Structure Specification | Algorithm Specification

Design Products

# Implementation


There are 4 fundamental principles (or pillars) that can affect the way the software is constructed:



# Implementation

- Reduction of Complexity → This aims to build software that is easier to understand and use.

- Anticipation of diversity → It takes into account that software construction might change in various way over time. That is that software evolves. In many cases it evolves in unexpected ways. And therefore, we have to be able to anticipate some of these changes.

# Implementation

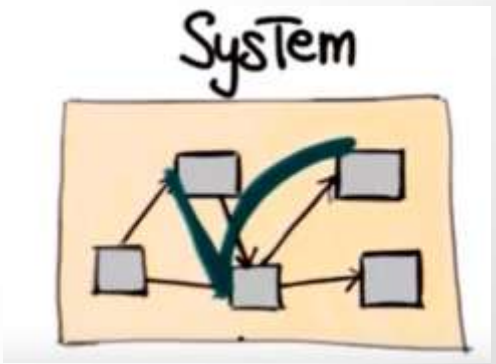
- Structuring for validation → Also called design for testability. This means we want to build software so that it is easily testable during the subsequent validation and verification activities.

- Use of standards → It is important that the software conforms to a set of internal or external standards. And some examples of this might be, for example, for internal standards, coding standards within an organization, or naming standards within an organization. As for external standards, if for example you are developing some medical software, there are some regulations and some standards that you have to adhere to in order for your software to be valid in that domain.

# Verification & Validation

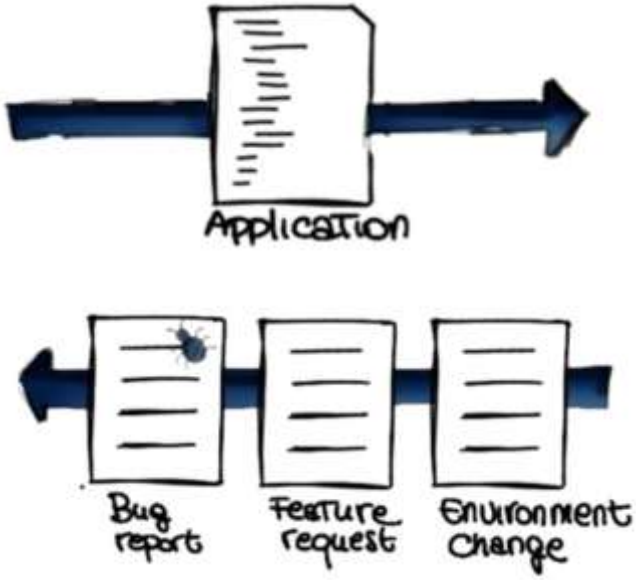
- Once we have built our system, verification and validation is that phase of software development that aims to check that the software system meets its specification and fulfills its intended purpose.
- Validation: Did we build the *right system*?
- Verification: Did we build the *system right*?





# Maintenance

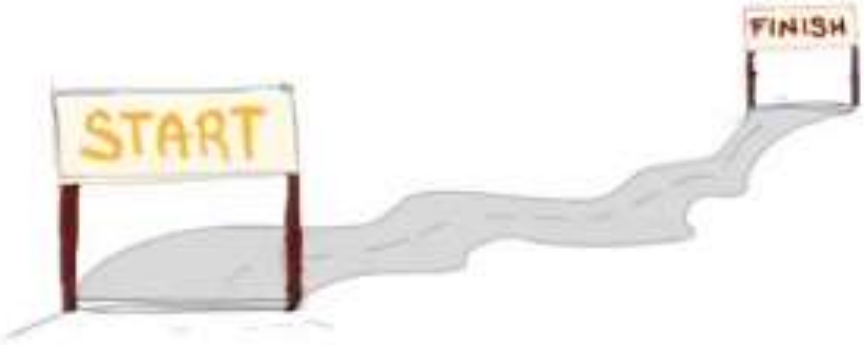


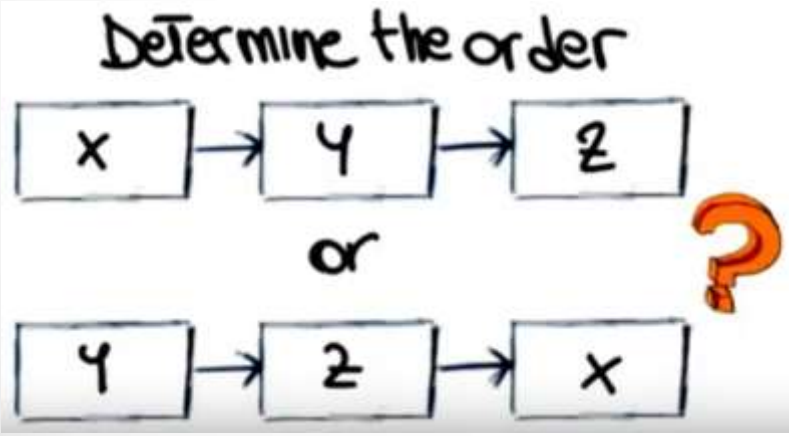


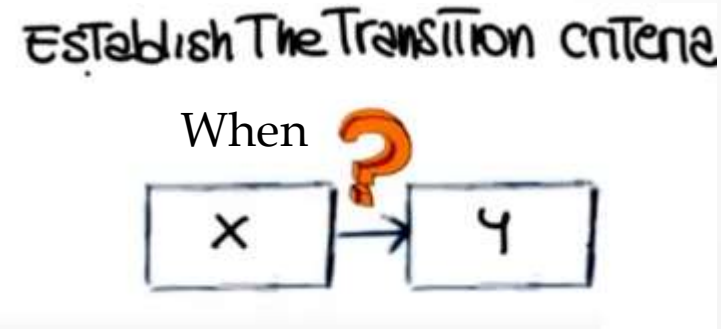

- Corrective Maintenance
- Perfective Maintenance
- Adaptive Maintenance

Maintenance is Expensive! Why?
REGRESSION TESTING!

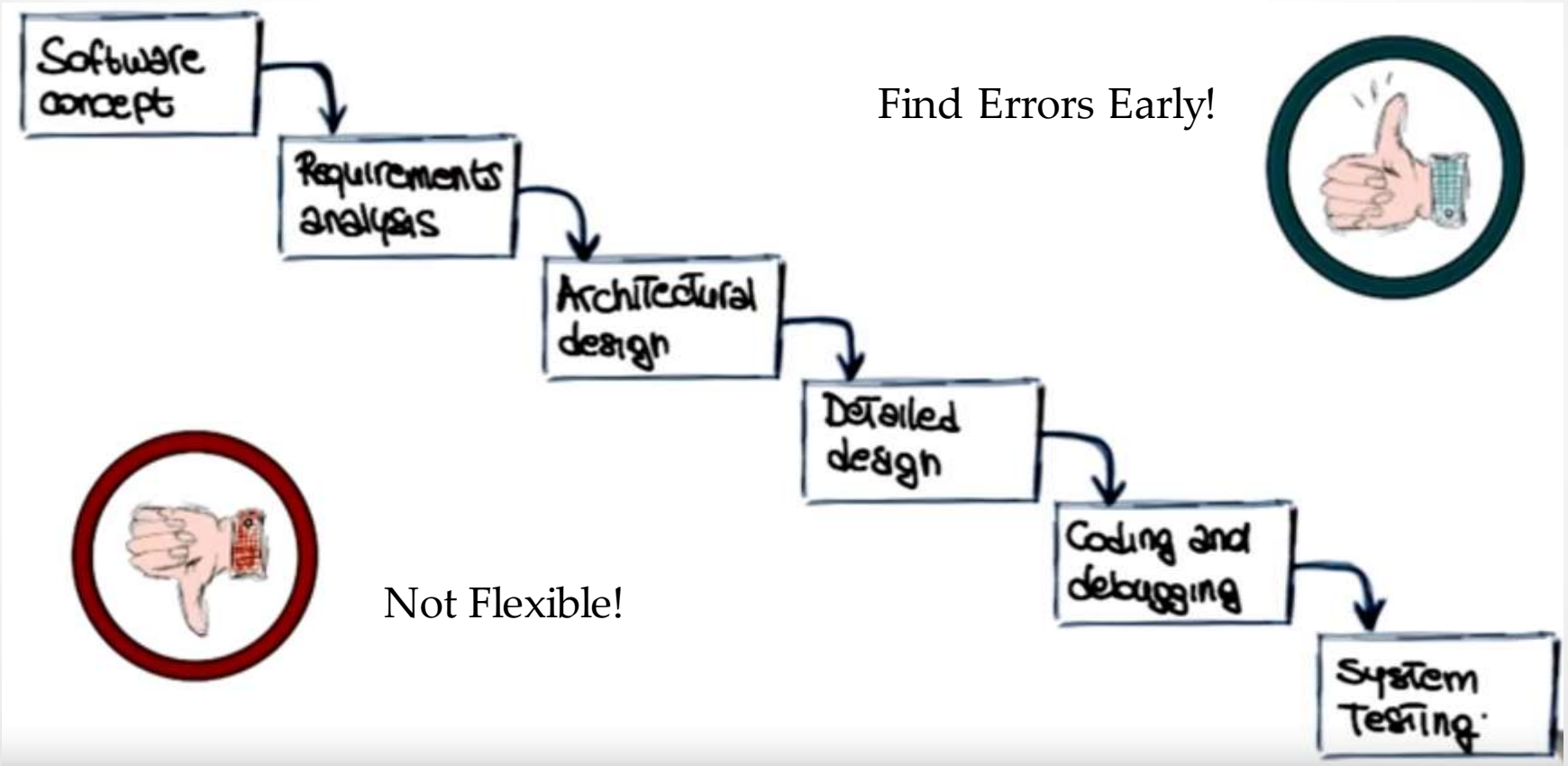
# Software Process Model



# Waterfall


Software concept
Requirements analysis
Architectural design
Detailed design
Coding and debugging
System Testing
Find Errors Early!
Not Flexible!

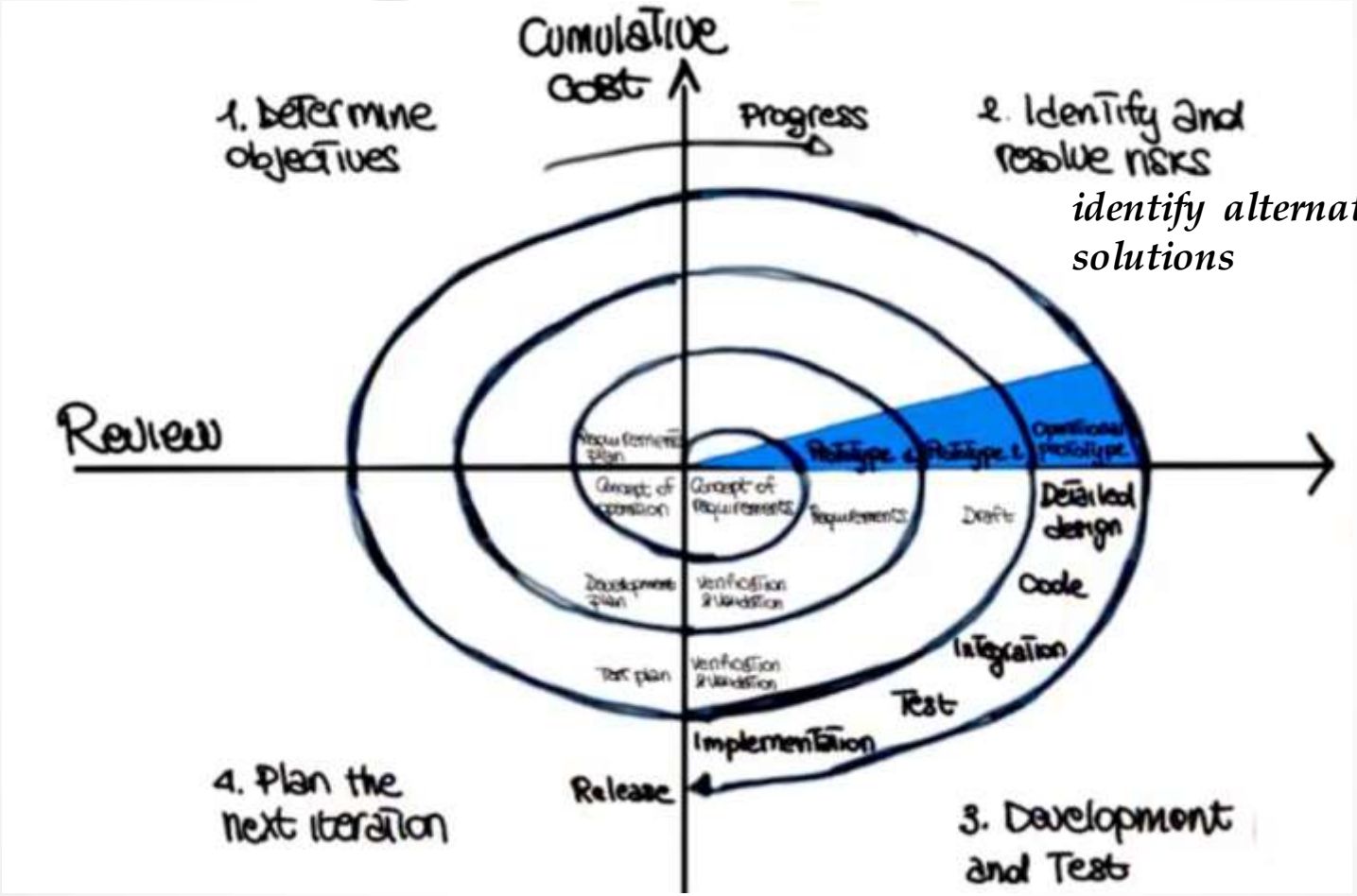
# Waterfall

- The pure waterfall model performs well for software products in which there is a ***stable product definition***, ***the domain is well known*** and ***the technologies involved are well understood***. In these kind of domains, the waterfall model helps you to ***find errors in the early***, local stages of the projects.

- Main Advantage: Find Errors Early!
- Main Disadvantage: Not Flexible!

- Normally, it is difficult to fully specify requirements at the beginning of a project. And this lack of flexibility is far from ideal when dealing with project in which requirements change, the developers are not domain experts or the technology used are new and evolving, that is it is less than ideal for most real world projects.

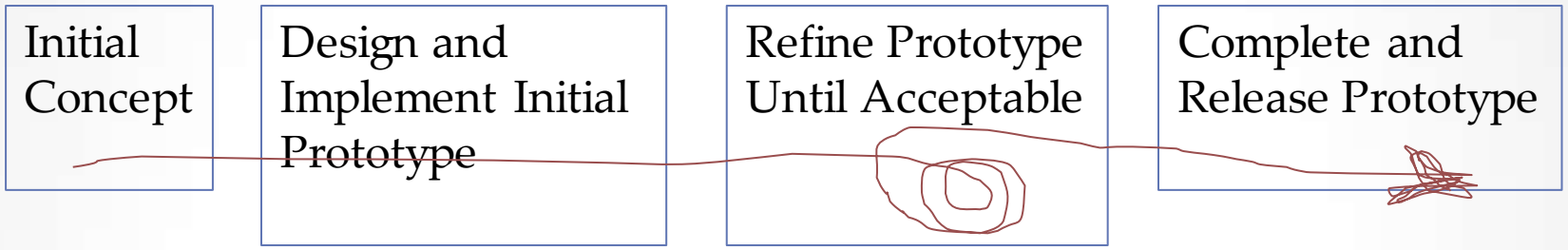
# Spiral Process



# Spiral Process

- So basically, the spiral process prescribes a way of developing software by going through the four phases in an iterative way, in which we learn more and more of the software, we identify more and more, and account for more and more risks and we go more and more towards our final solution, our final release.
- Main Advantages:
  - ✓ Risk Reduction → The extensive risk analysis does reduce the chances of the project to fail.
  - ✓ Functionality can be Added → New functionality can be added at a later phase because of the iterative nature of the process.
  - ✓ Software Produced Early → At any iteration, we have something to show for our development. We don't to wait until the end before producing something.
- Main Disadvantages:
  - ✓ Specific Expertise → The risk analysis requires a highly specific expertise.
  - ✓ Highly Dependent on Risk Analysis → Risk analysis has to be done right.
  - ✓ Complex → The Spiral Model is way more complex than other models, like for example, the Waterfall model. And therefore it can be ***costly*** to implement.

# Evolutionary Prototyping Process (EVP)




- Evolutionary Prototyping Process is an ideal process (model) when not all requirements are well understood.

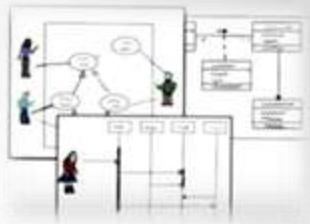
# Evolutionary Prototyping Process (EVP)

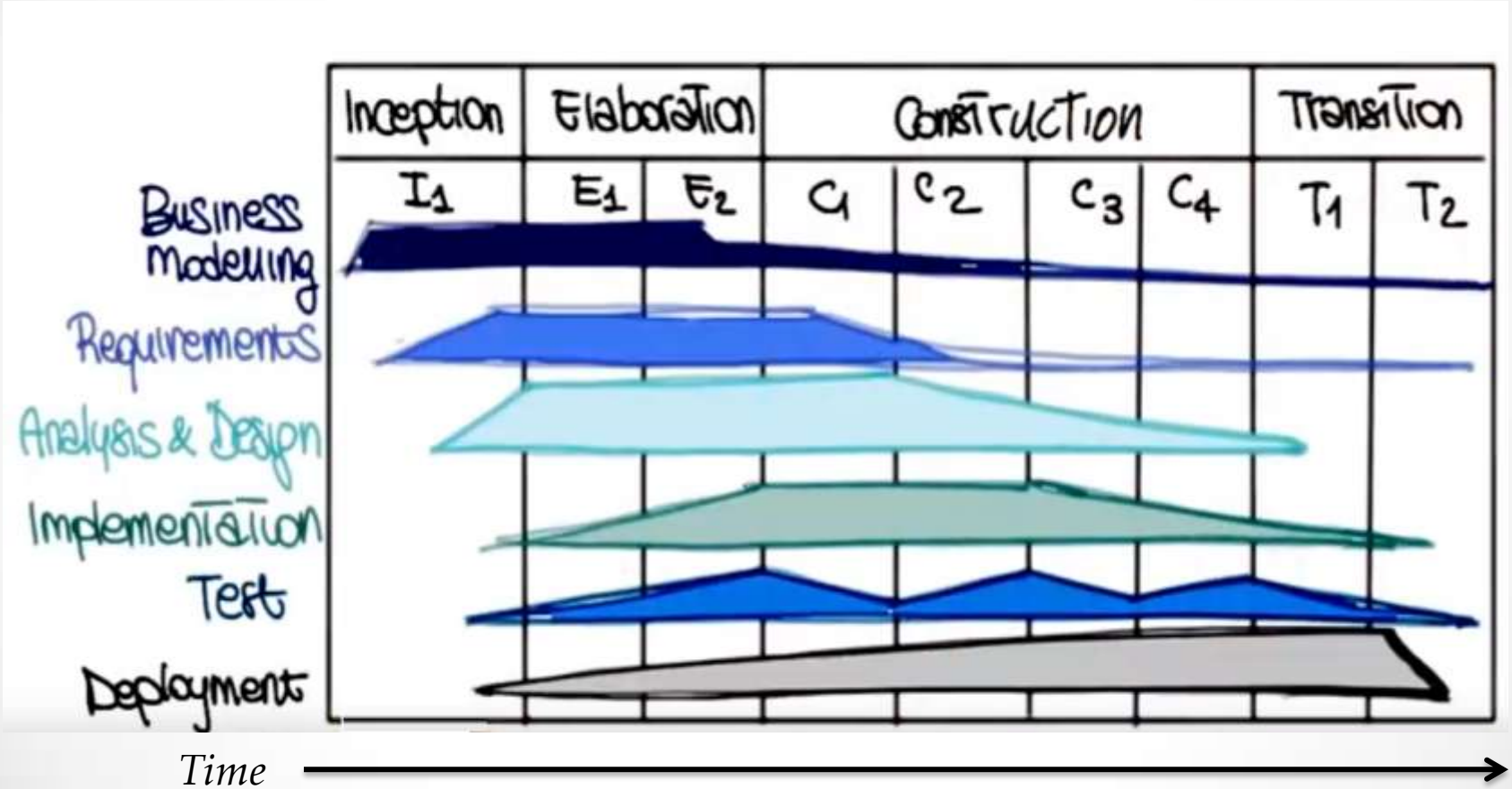
- Developers start by developing the ***parts of the system that they understand***, instead of working on developing a whole system, including parts that might not be very clear at that stage.
- The ***partial system*** is then shown to the customer and the ***customer feedback*** is used to drive the next iteration, in which either changes are made to the current features or new features are added.
- So, either the current prototype is improved or the prototype is extended.
- Finally, when the customer agrees that the prototype is good enough, the developers will complete all the remaining work on the system and release the prototype as the final product.

# Evolutionary Prototyping Process (EVP)

- Main Advantage:
  - Immediate Feedback! → Developers get feedback immediately as soon as they produce a prototype and they show it to the customer and therefore, the risk of implementing the wrong system is minimized.
- Main Disadvantage:
  - Difficult to plan! → When using evolutionary prototype it is difficult to plan in advance how long the development is going to take, because we don't know how many iterations will be needed.
  - Always Depend on the Customer! → It can easily become an excuse to do kind of do cut and fix kind of approaches in which we hack something together, fix the main issues when the customer gives us feedback, and then continue this way, until the final product is something that is kind of working, but it's not really a product of high quality.
- There are many different kinds of prototyping, so Evolutionary Prototyping is just one of them. For example, **Throwaway Prototyping** is another kind of prototyping in which the prototype is just used to gather requirements, but is thrown away at the end of the requirements gathering, instead of being evolved as it happens here.

# Rational Unified Process (RUP)

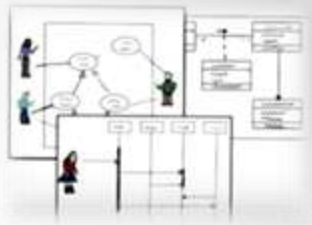




# Rational Unified Process (RUP)

- RUP works in an iterative way, which means it that it performs different iterations. And at each iteration, it performs four phases.
- In each one of the 4 phases we perform standard software engineering activities, the ones that we just discussed. And we do them to different extent, based on the phase in which we are.

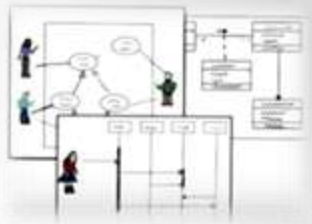
# Rational Unified Process (RUP)



1. In the Inception phase the work is mostly to sculpt the system. So basically figuring out what is the scope of the work, what is the scope of the project, what is the domain. So that we can be able to perform initial cost and budget estimates.

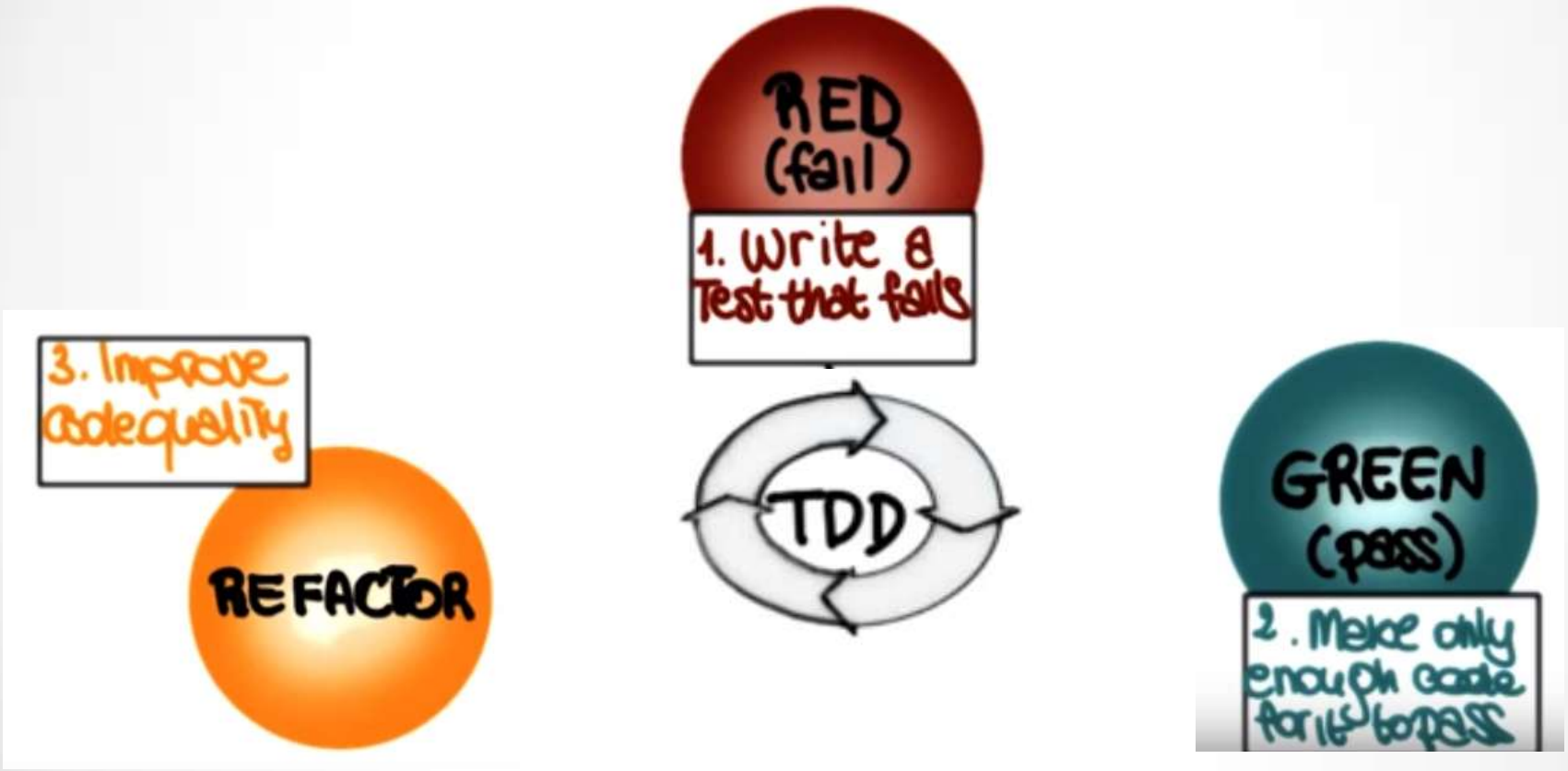
2. The Elaboration phase is the phase in which we focus on the domain analysis and define the basic architecture for the system. So this is a phase in which analysis and design are particularly paramount.

# Rational Unified Process (RUP)



3. In the Construction phase the bulk of the development and most of the implementation actually occurs.
4. In the transition phase the system goes from development into production, so that it becomes available to users. This is the phase in which the other activities in software development become less relevant and deployment becomes the main one.

# Agile Process(es)

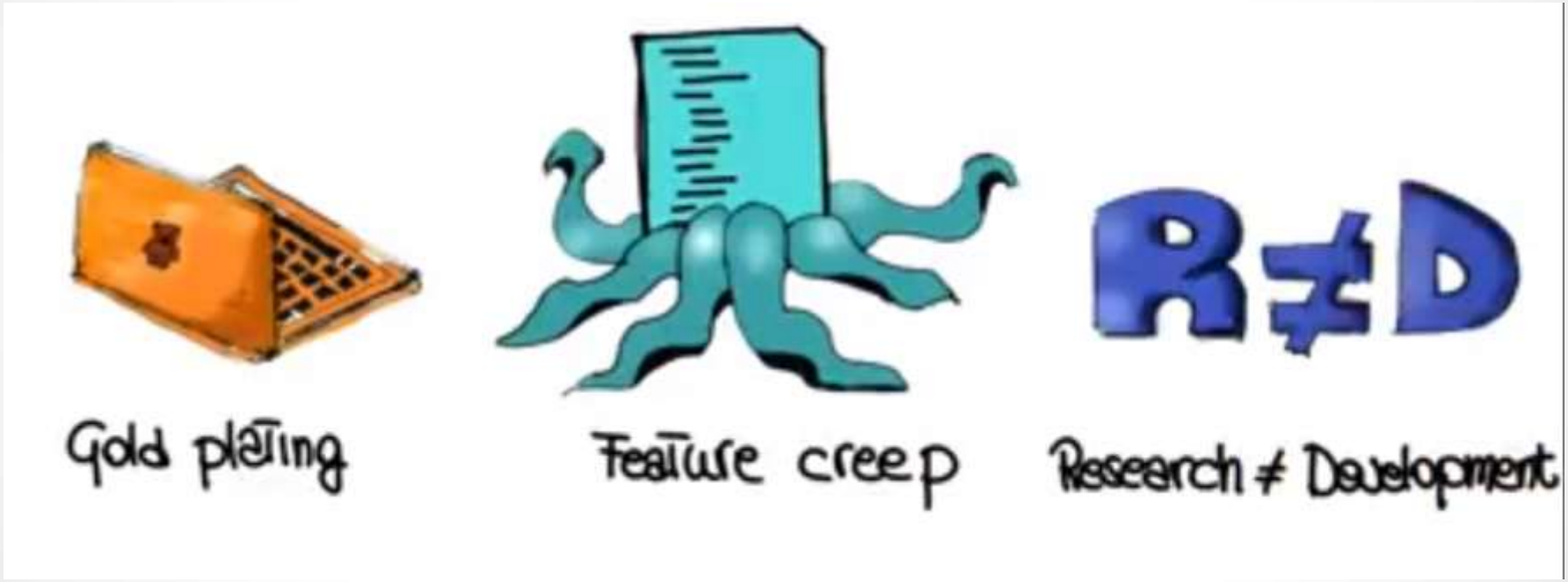
# Agile Process(es)


- This is a group of software development methods based on highly iterative and incremental development (on the previous slide TDD – Test Driven Development method is shown).

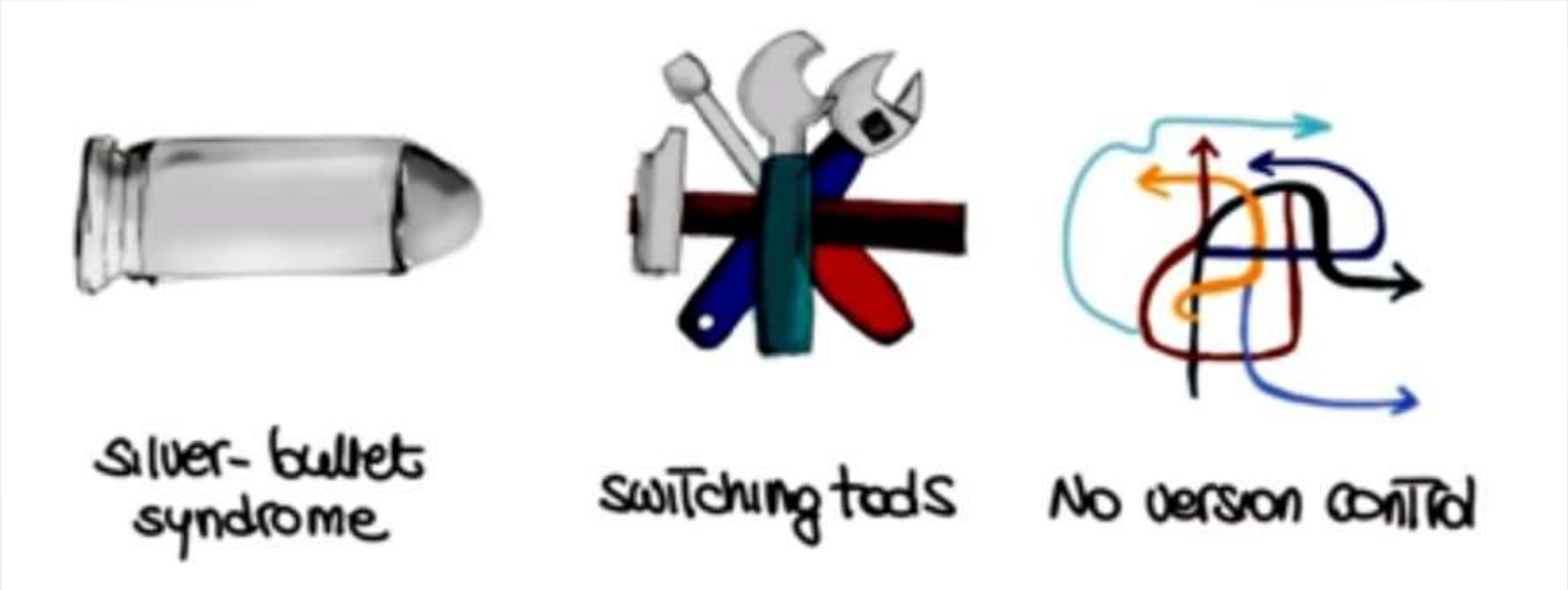
# Classis Mistakes: People

# Classis Mistakes: Process

# Classis Mistakes: Product



# Classic Mistakes: Technology



The End.