RISK AND RESILIENCY BOOTCAMP

TEKsystems Global Services

WORKFORCE DEVELOPMENT

THE ENGINEERING PROCESS

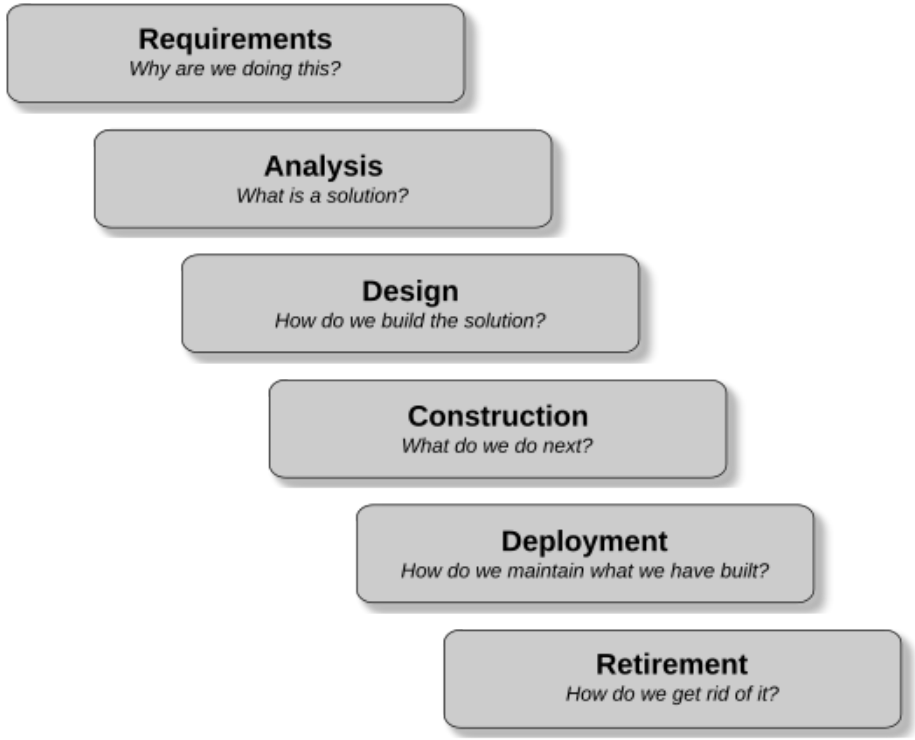
In this module, we will

- Explore the generic engineering process
- Identify the risk and resiliency issues involved in the process
- Identify mitigation and recovery strategies

Image credit: https://skefto.com/blog/resilience-strategy-and-risk/

THE ENGINEERING PROCESS

The engineering process is a description of the logical sequence of natural questions or problems as they occur in any kind of construction process
The process is universal and can be observed any time we are successfully building or delivering something in an engineering or business context
This includes software, operations environments, data and IT architecture



REQUIREMENTS

The basic question that has to be answered during requirements is "*Why are we building this?*"

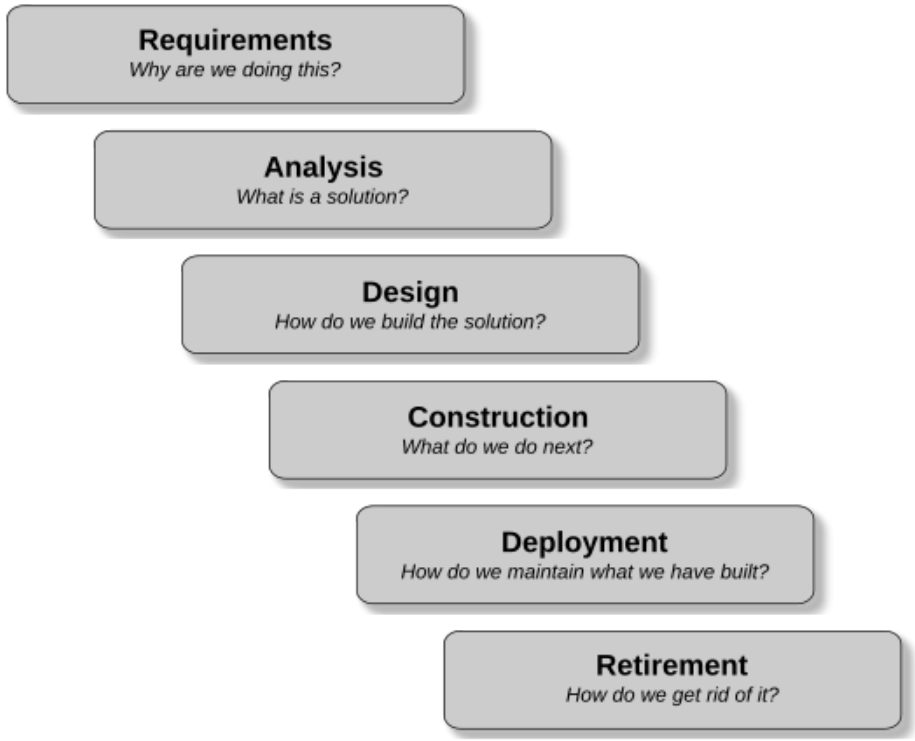
- This is often called the "value proposition"
- Identification of the problem to be rectified

This is generally followed by two other kinds of questions

"What do the users need the solution we are building to do for them?"

- At this stage, we need to know the functional requirements (what the stakeholders expect what we are building to do for them)
- And the non-functional requirements (what the performance criteria are that the solution must meet)

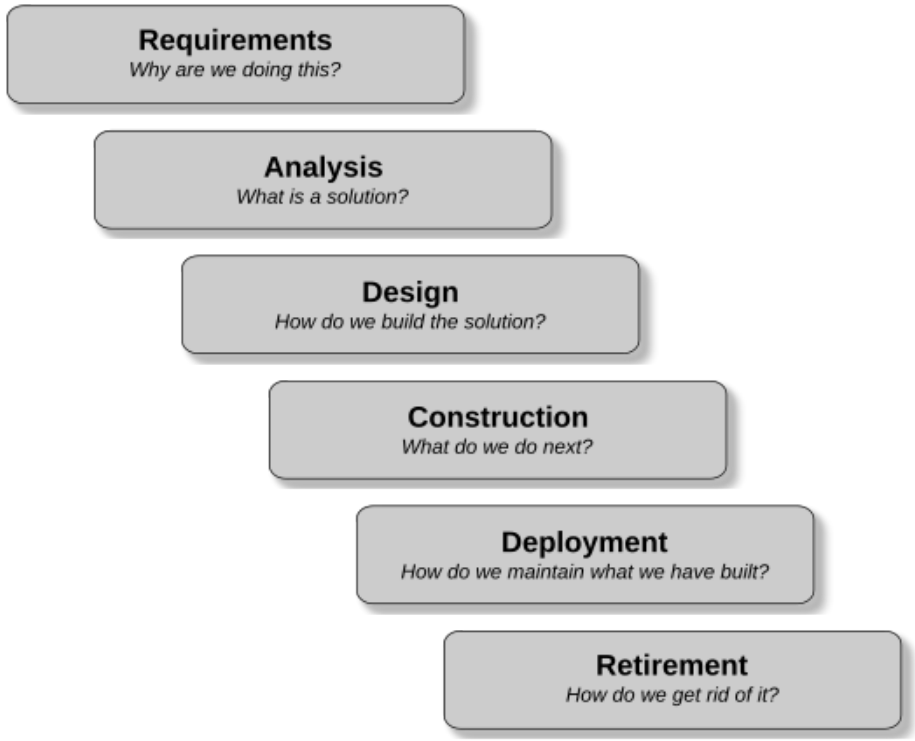
"What are the constraints – budget, time, architecture, etc. – that have to be considered when building the solution?"



REQUIREMENTS

It is not always possible to identify all of the requirements at the beginning of the project

- We may need to use some adaptive methodology like an Agile approach
- Iterations help validate assumptions about the underlying problem
- And help identify erroneous assumptions about what the requirements are



REQUIREMENTS

"Go to the source of the problem"

- Find out what the real requirements are directly from those who have them
- Requirements are often supplied second hand
  - "This is what the people I supervise need."
  - Actually "This is what I *think* my people need"
- Identify stakeholders
  - Groups that have requirements
  - And the capability to force you to accommodate their requirements

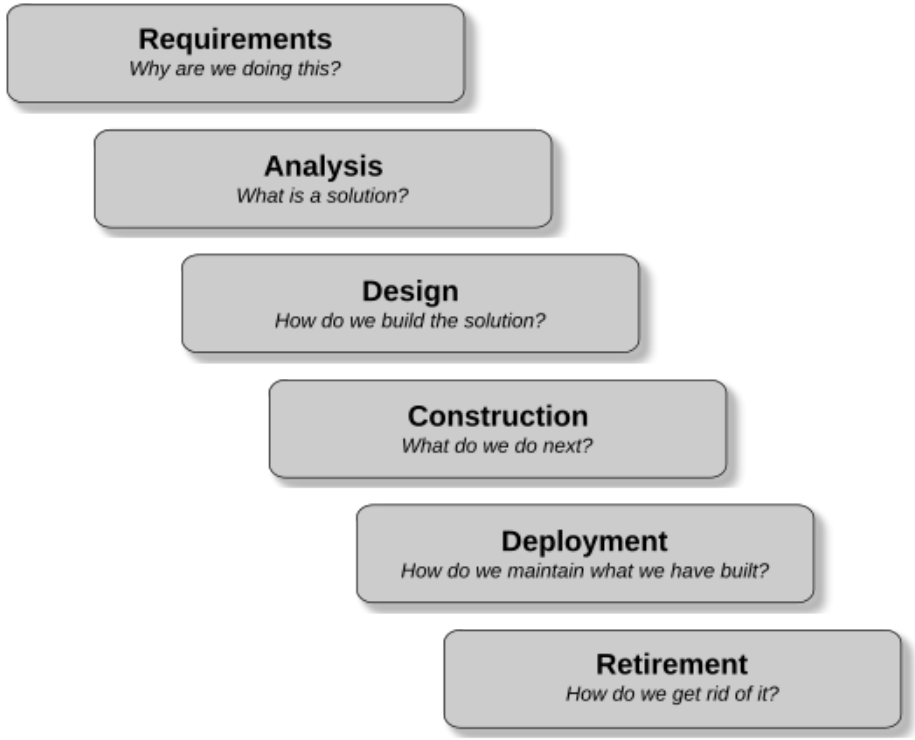


Image credit: https://skefto.com/blog/resilience-strategy-and-risk/

REQUIREMENTS

Primary stakeholders

- Impacted directly by the project
- Customers, staff, process owners, data owners, budget controllers
- Have project specific requirements we can ask them about

Secondary stakeholders

- Usually regulate a domain of activity
- Data governance, accessibility standards
- Do not have project specific requirements
- Their requirements are usually published
- Non-responsive to inquires (“Go look it up in our publication on the topic”)

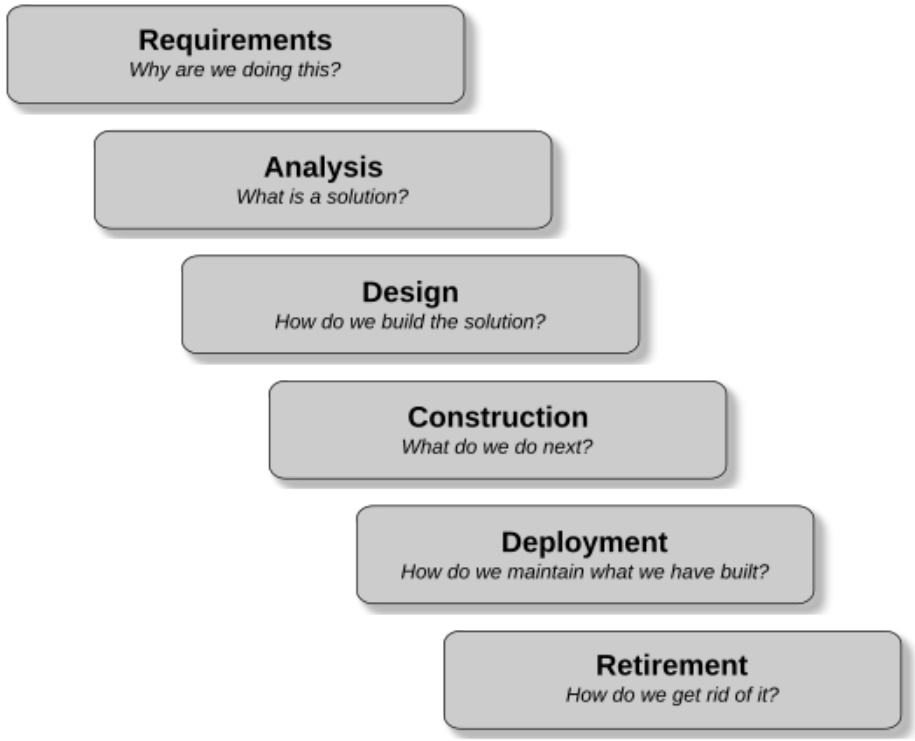


Image credit: https://skefto.com/blog/resilience-strategy-and-risk/

REQUIREMENTS

Secondary stakeholders

- Often are the source of constraints
- IT infrastructure standards
- Legal and regulatory compliance
- SLAs for that type of project
- Business case requirements
- Security requirements
- Audit and accountability requirements

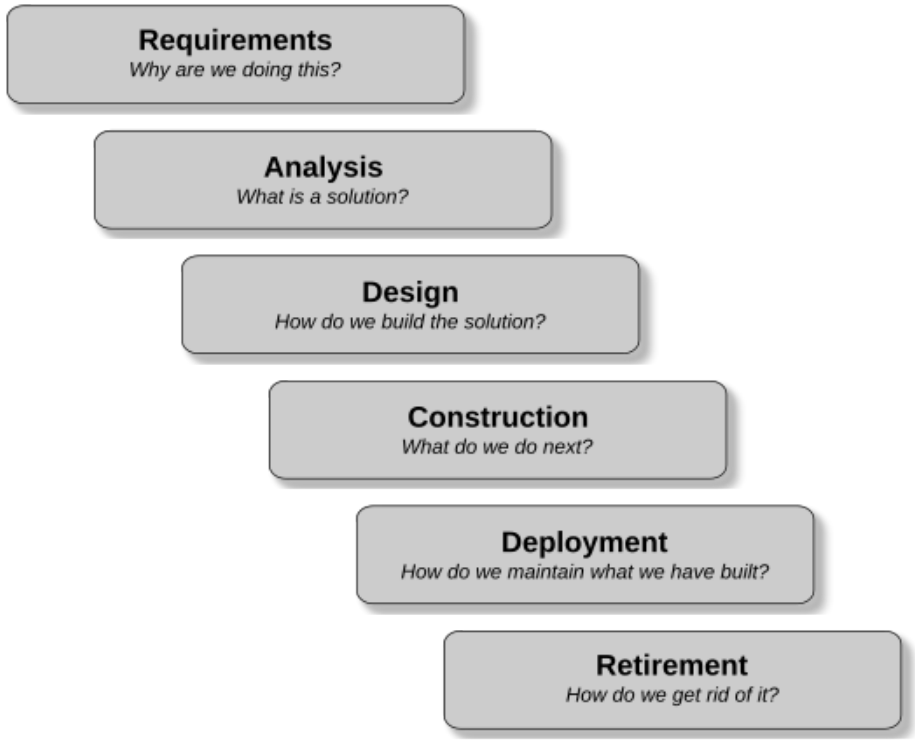


Image credit: https://skefto.com/blog/resilience-strategy-and-risk/

ANALYSIS

In the analysis phase, we find a solution to the problem
We formulate the solution in a specification that describes what is to be built

- The specification serves as a design target
- Whatever design we propose must create a product that meets the description of the solution in the specification

We may come up with multiple ways to solve the problem which differ in terms of cost, quality and other factors

- But eventually, we need to settle on a single solution

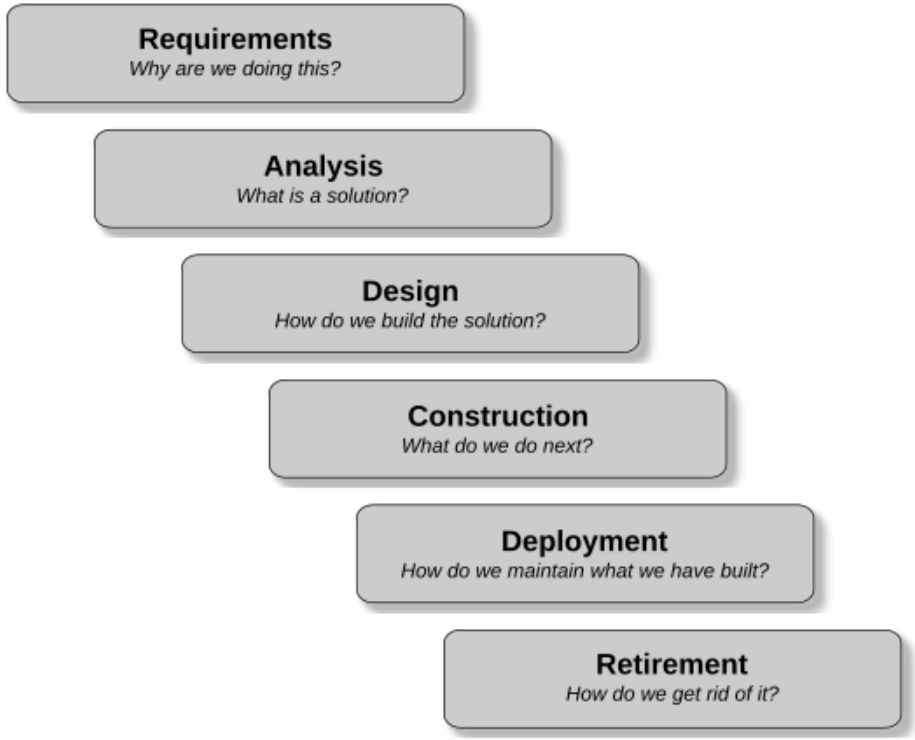


Image credit: https://skefto.com/blog/resilience-strategy-and-risk/

ANALYSIS

Analysis will propose an architecture

- Often referred to as a high level design
- Identifies major components
- For example, the data architecture

It should decompose the system into subsystems

- Called a functional decomposition

The specification

- Defines what the system should look like and how it behaves
- Often with a full set of acceptance tests

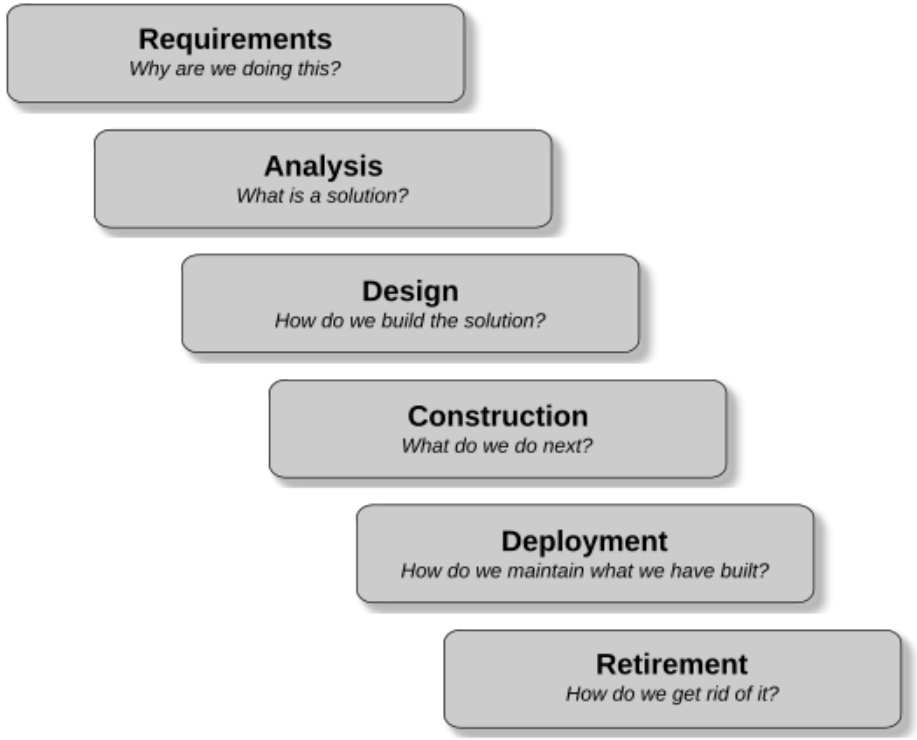


Image credit: https://skefto.com/blog/resilience-strategy-and-risk/

ANALYSIS

Comparative analysis

- What other similar projects are doing to solve similar problems
- The types of outcomes that result
- Learn from their mistakes and successes

Historical analysis

- What have we tried to date?
- What have we learned?
- Usually about domain issues

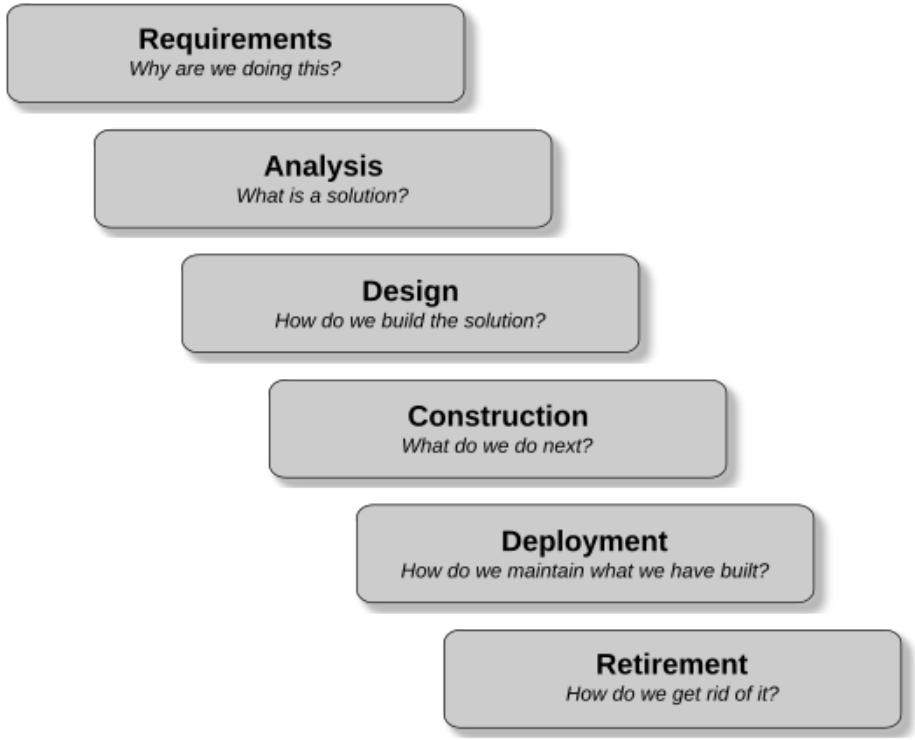


Image credit: https://skefto.com/blog/resilience-strategy-and-risk/

ANALYSIS

Alternatively

- We may need to experiment

Small pilots

- Experiment with different architectures
- Experiment with different interfaces
- See if our assumptions were valid

Often requires a detailed knowledge of the domain

- Design Thinking
- Design Driven Development

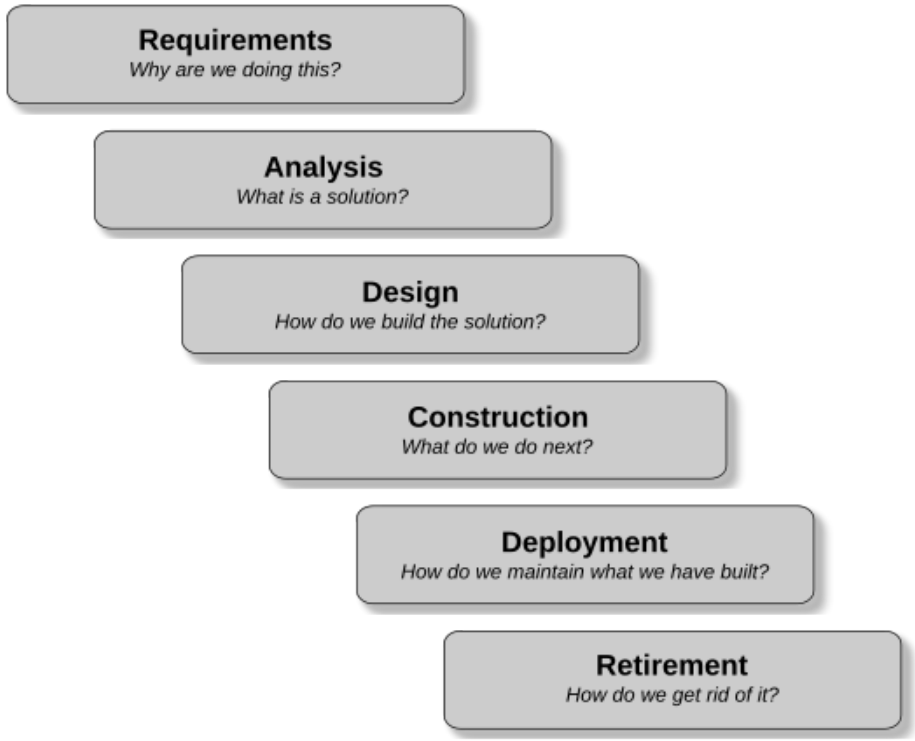


Image credit: https://skefto.com/blog/resilience-strategy-and-risk/

COMPLEXITY AGAIN

Three kinds of complexity we have to deal with

- Domain complexity: trying to automate in a domain that is inherently complex
- Design complexity: Designs that are not elegant (simple and effective)
- Environmental complexity: Resulting from a disorganized and poorly functioning business

We can eliminate a lot of design complexity with good design and engineering practices

- We cannot make domain complexity go away, we can only manage it

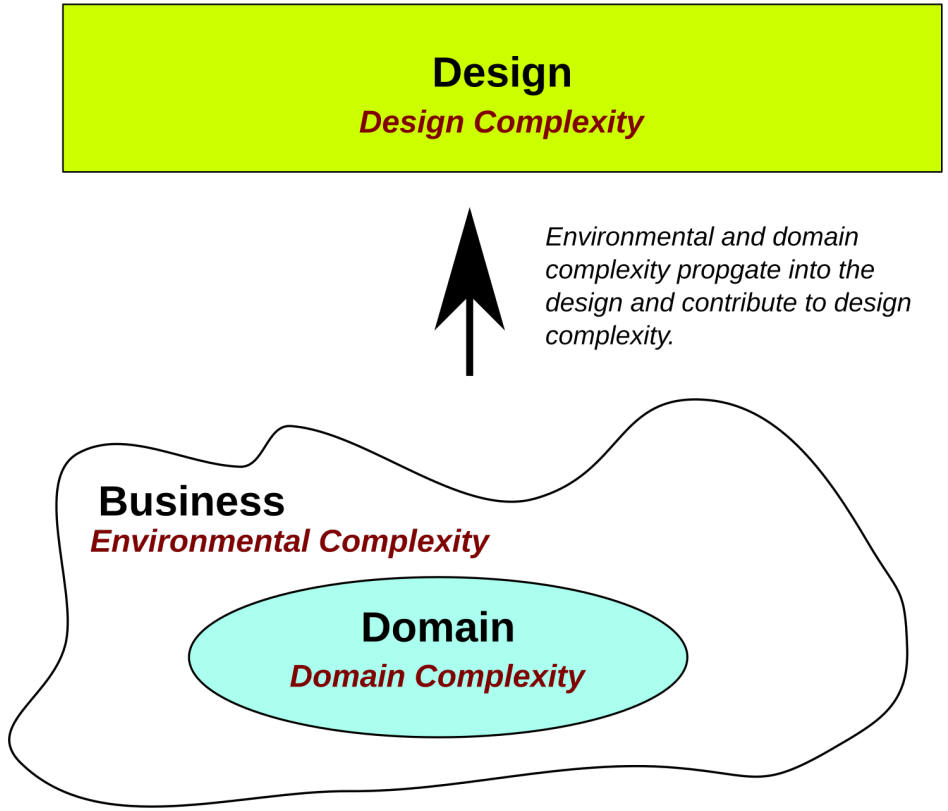


Image credit: https://skefto.com/blog/resilience-strategy-and-risk/

PROPAGATION OF COMPLEXITY

Not properly managed, environmental and domain complexity directly propagate into design

- We should not confuse environmental complexity with domain complexity
- We cannot deal with it, so we have to ignore it in understanding the domain

The most important single aspect of software development is to be clear about what you are trying to build.
Edsger Dijkstra

First, solve the problem. Then, write the code
Donald Knuth

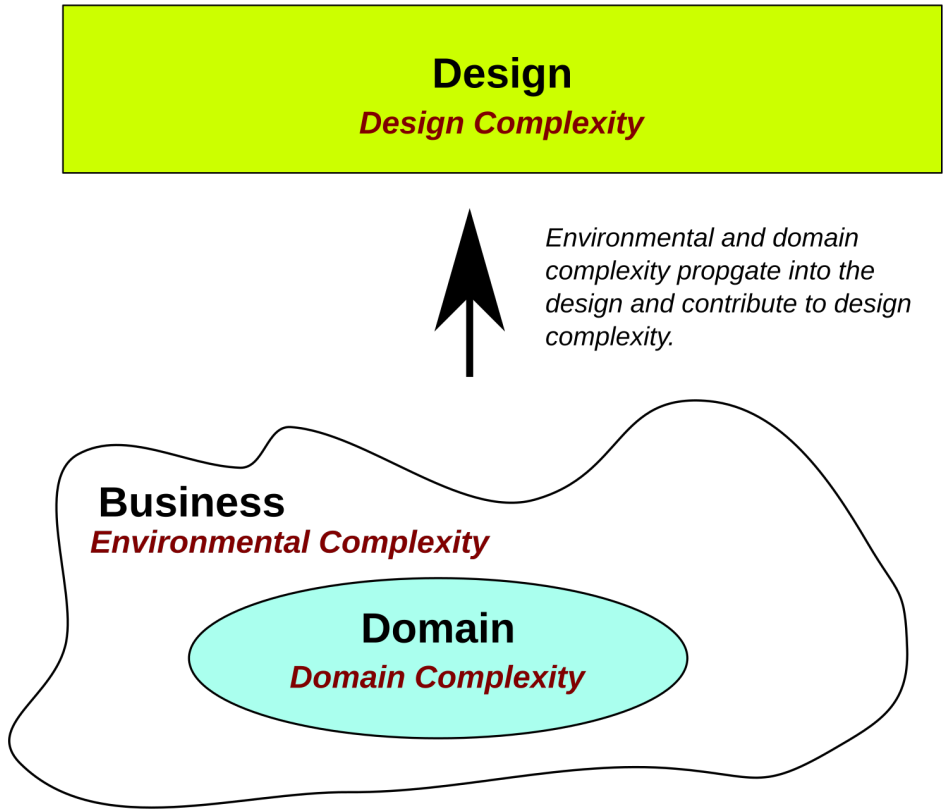


Image credit: https://skefto.com/blog/resilience-strategy-and-risk/

RESOLVING COMPLEXITY

When dealing with domain complexity

- We extract the essential elements of the domain that are relevant to the problem to be solved
- We abstract out what is irrelevant to solving the problem
- The resulting domain model is a simplification of the problem domain that retains all the critical information

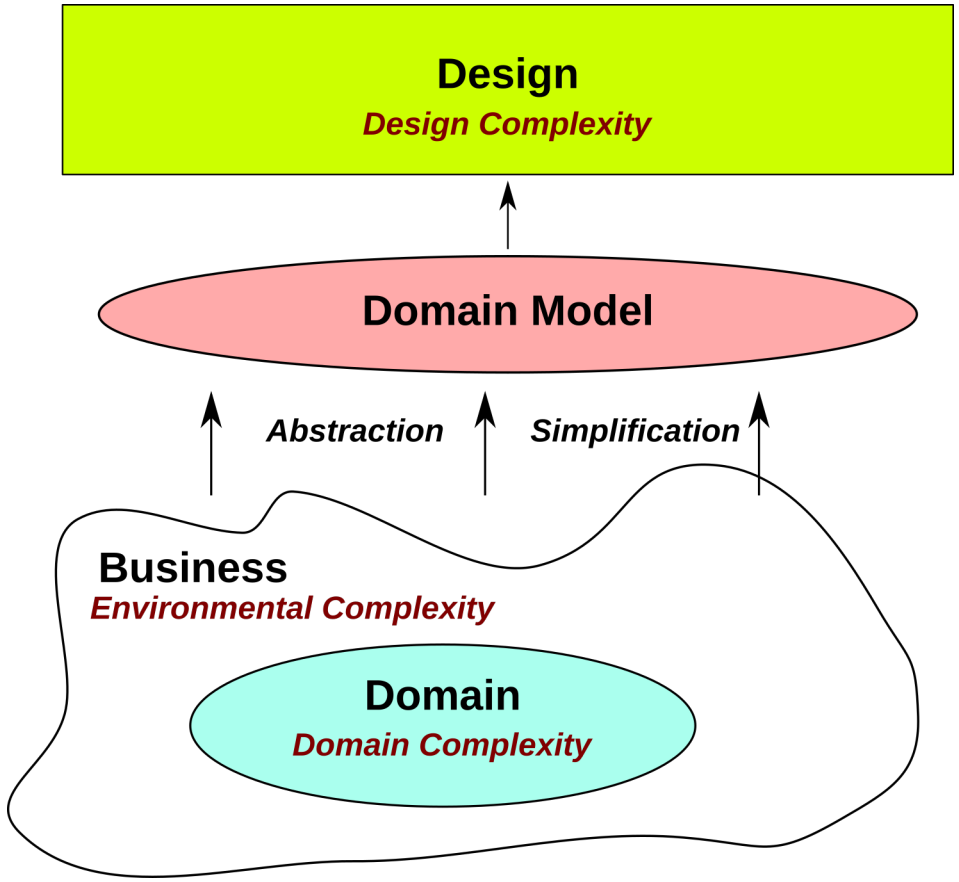
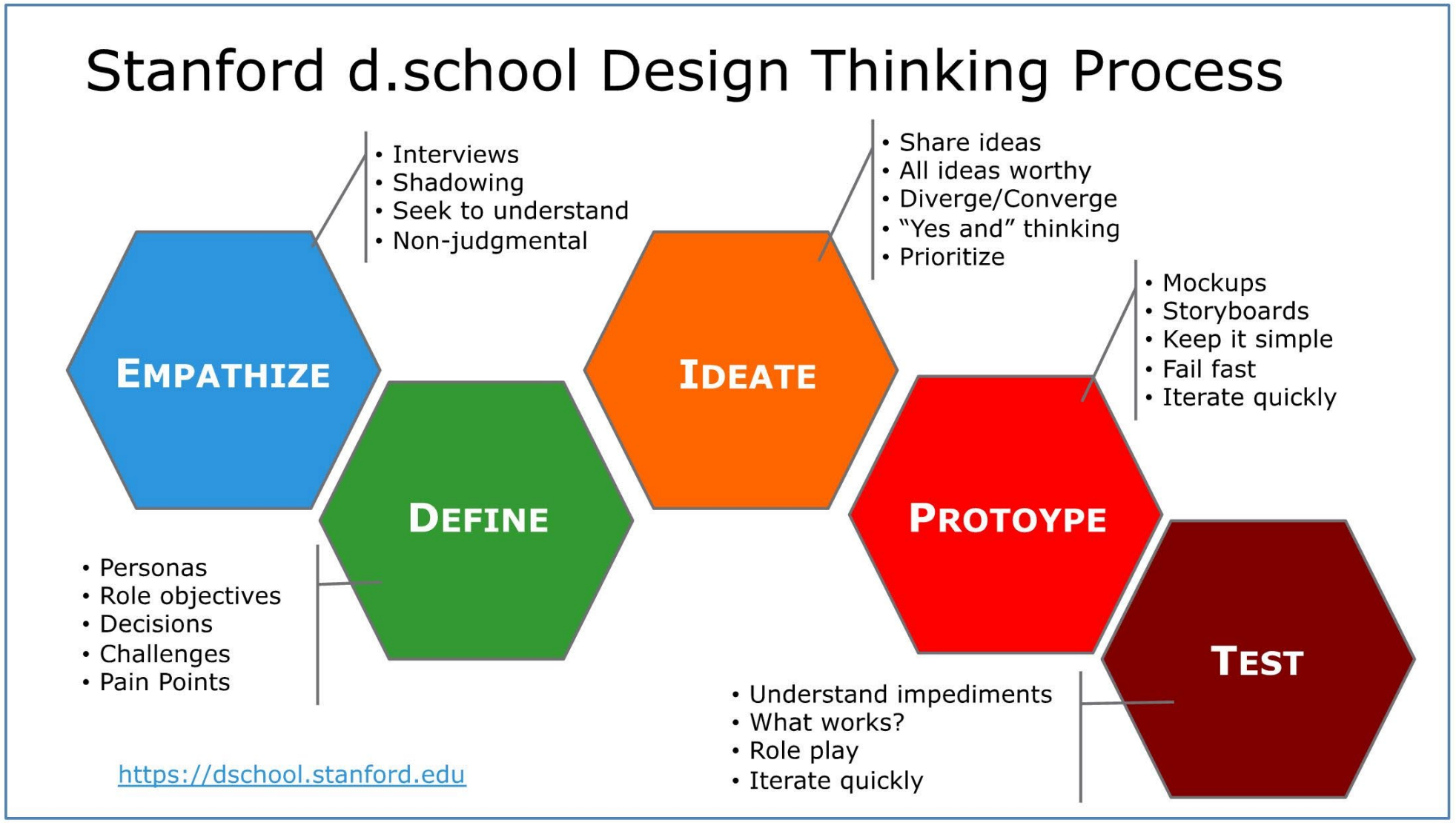
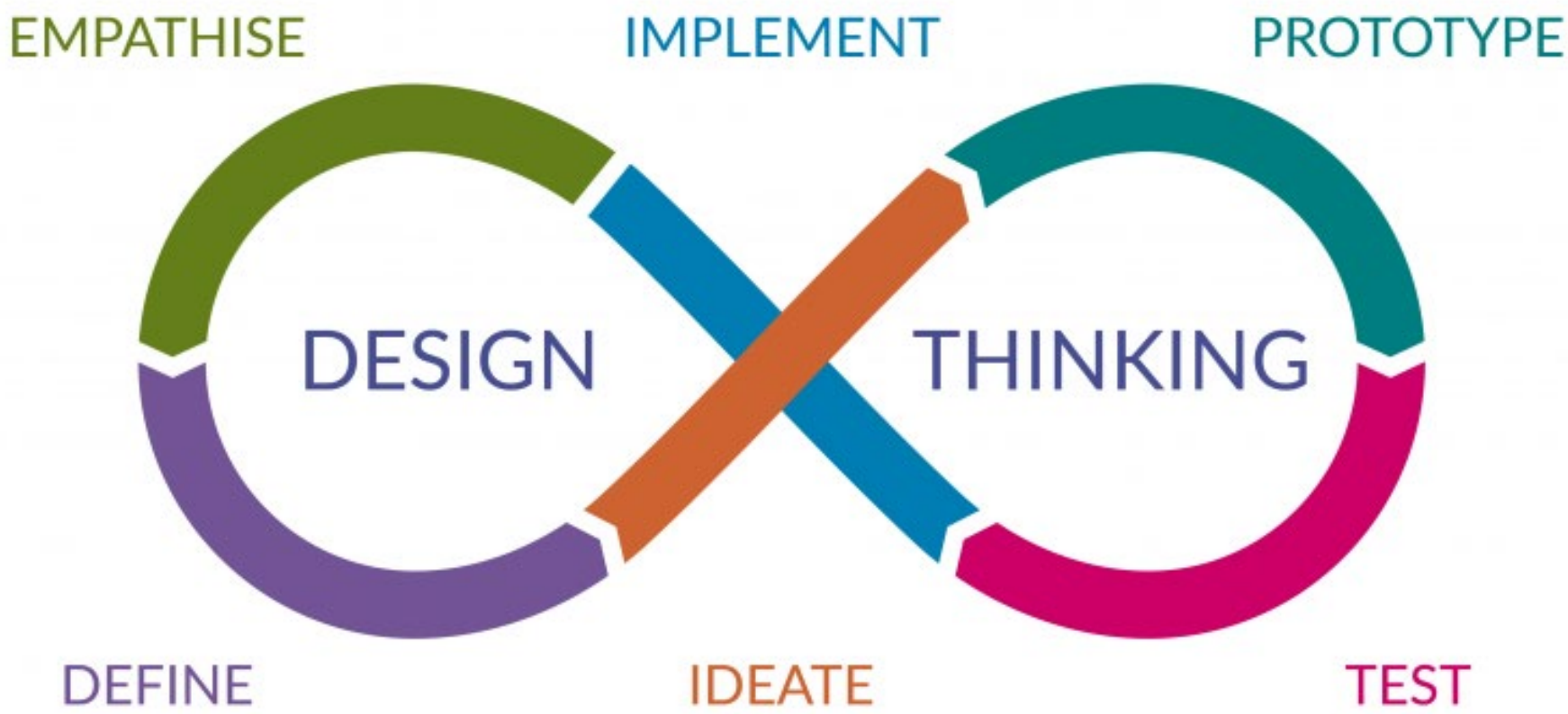


Image credit: https://skefto.com/blog/resilience-strategy-and-risk/

DESIGN THINKING



DESIGN THINKING



KNOWLEDGE CRUNCHING

Key idea

- Successive deep dives into the problem domain
- Understand concepts and relationships in the domain
- Work from the perspective of the domain experts
- Collaboration with SMEs
- Refine the domain model with insights
- Do it again

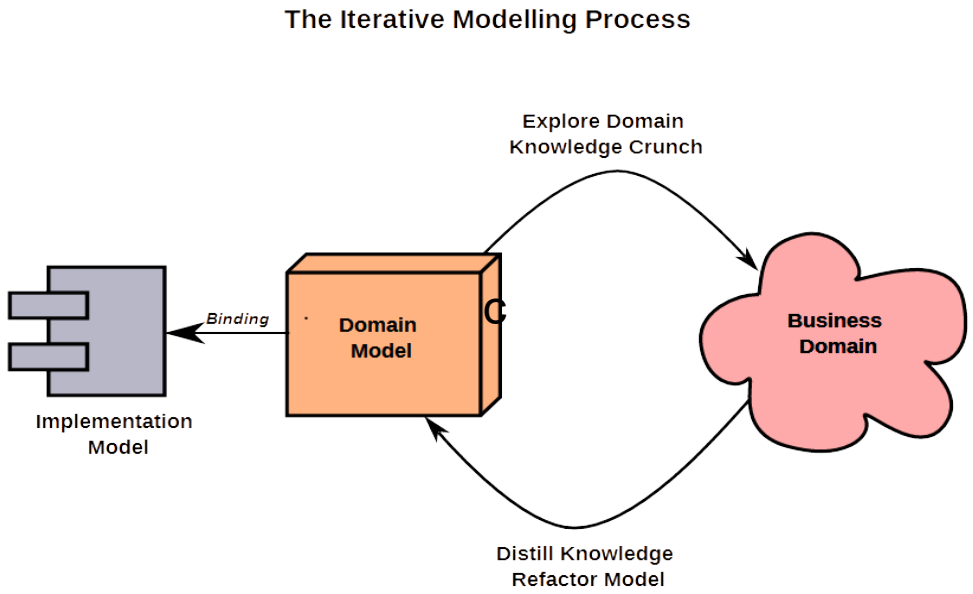


Image credit: https://skefto.com/blog/resilience-strategy-and-risk/

RESOLVING COMPLEXITY

Breakthrough iterations

- Happen when we get sudden new insight into the domain
- Often results in a simpler and more powerful model
- Closely related to design thinking

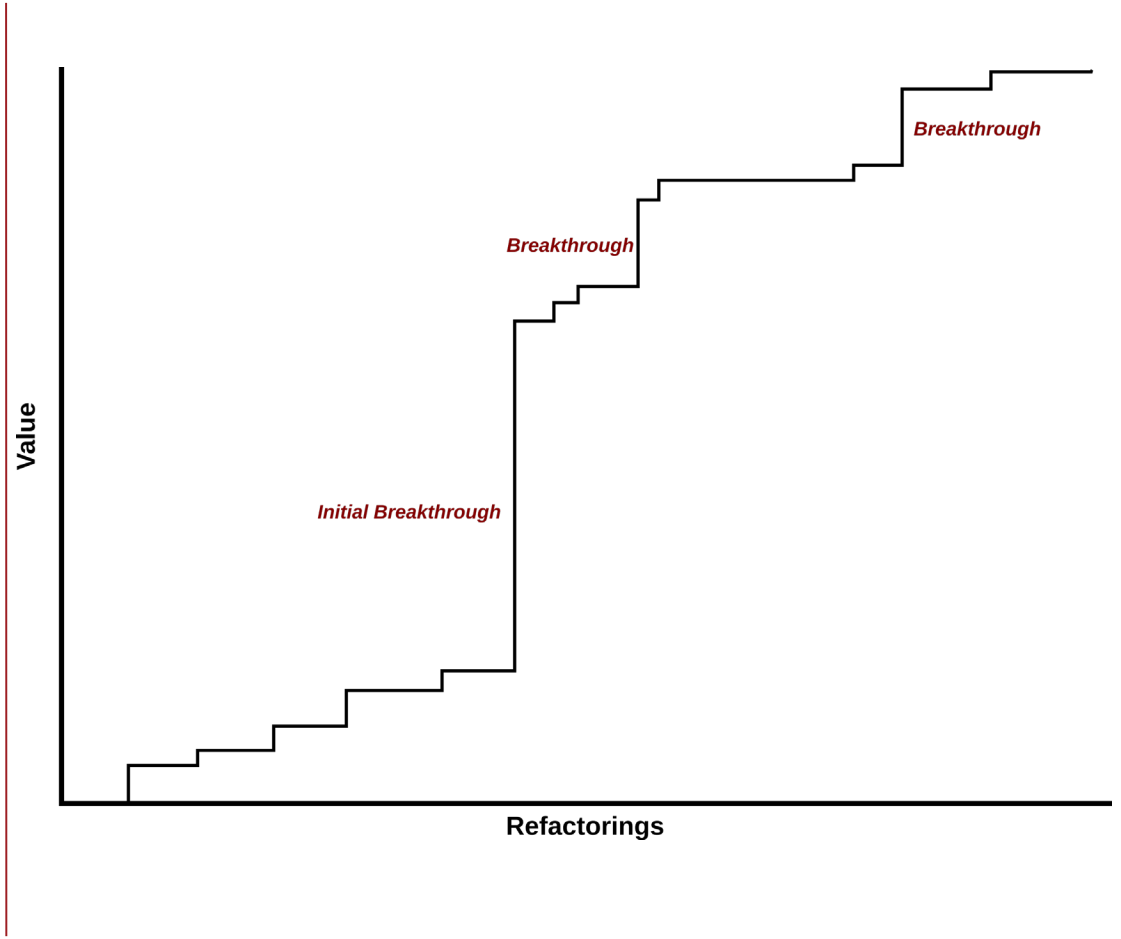


Image credit: https://skefto.com/blog/resilience-strategy-and-risk/

THE SPEC

- The specification is often called the SRS (software requirements specifications) document
  - An SRS is the basis of a complete and unambiguous communication of information between all the different parties involved in systems development
  - The basis for this section is the IEEE standard *"Best practices for software requirements specifications"*
- The intent, as noted by the IEEE, is to
  - Establish the basis for agreement between the customers and the suppliers on what the software product is to do
    - The complete description of the functions to be performed by the software specified in the SRS will assist the potential users to determine if the software specified meets their needs or how the software must be modified to meet their needs

THE SPEC

- Reduce the development effort
  - The preparation of the SRS forces the various concerned groups in the customer's organization to consider rigorously all of the requirements before design begins and reduces later redesign, re-coding, and retesting
  - Careful review of the requirements in the SRS can reveal omissions, misunderstandings, and inconsistencies early in the development cycle when these problems are easier to correct
- Provide a basis for estimating costs and schedules
  - The description of the product to be developed as given in the SRS is a realistic basis for estimating project costs and can be used to obtain approval for bids or price estimates
- Provide a baseline for validation and verification
  - Organizations can develop their validation and verification plans much more productively from a good SRS
  - As a part of the development contract, the SRS provides a baseline against which compliance can be measured
- Facilitate transfer
  - The SRS makes it easier to transfer the software product to new users or new machines
  - Customers thus find it easier to transfer the software to other parts of their organization, and suppliers find it easier to transfer it to new customers

THE SPEC

- Serve as a basis for enhancement.
  - Because the SRS discusses the product but not the project that developed it, the SRS serves as a basis for later enhancement of the finished product
  - The SRS may need to be altered, but it does provide a foundation for continued production evaluation
- Properties of a good spec
  - These are listed in table on the next slide
  - These are considered the essential features of a spec that reduce the risk of errors creeping into the development process

THE SPEC

Complete	*Specification description covers all possible alternatives that can occur, both valid and expected as well as invalid and unexpected.*
Consistent	*No two spec items require to system to behave differently when the state of the system and input conditions are the same.*
Correct	*The result of each spec item meets the acceptance criteria.*
Testable	*All inputs, states, decisions and outputs are quantified and measurable.*
Verifiable	*There is a finite cost effective process for executing each test case alternative.*
Unambiguous	*There is only possible way to interpret or understand each spec item.*
Valid	*Everyone can read, understand, and analyze the spec well enough to formally approve the described items.*
Modifiable	*The spec items are organized in a way so that they are easy to use, modify and update.*
Ranked	*The spec items are in a priority order that everyone on both the business and technical sides agree on.*
Traceable	*Every spec item can be traced back to an example and acceptance criteria that motivated it and then back to the original requirement.*

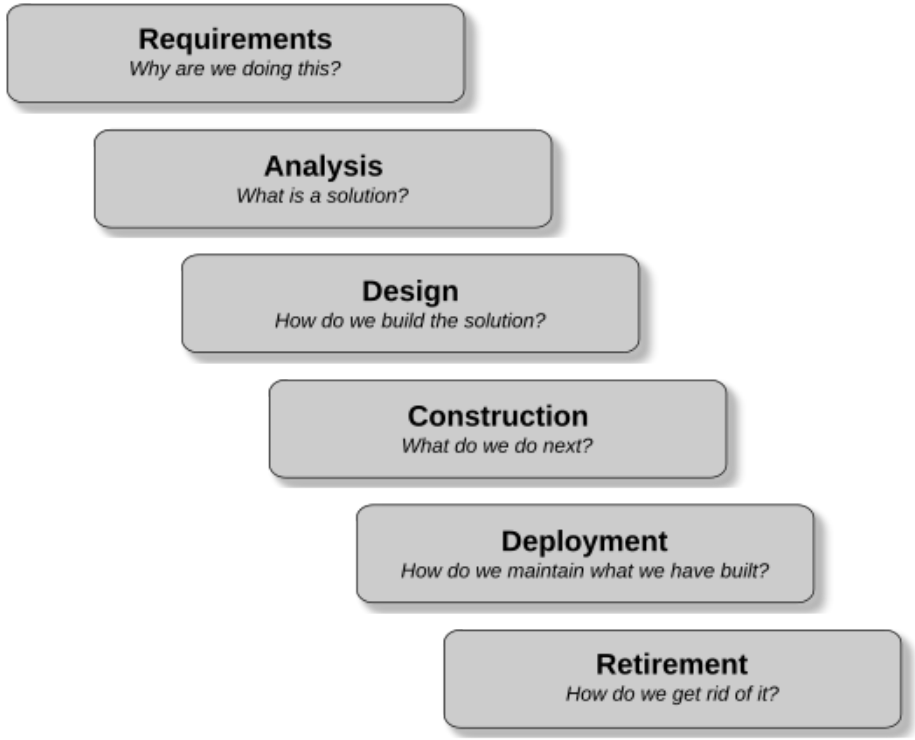
DESIGN

A specification describes a solution but does not tell us how that solution should be built

- A design describes how the solution is to be built given a specific set of resources, capabilities and constraints
- For a single specification, we may have a number of different designs depending on what we have available to work with

Early prototypes during Analysis

- Will have a putative design
- Lightweight; more of a plan for experimentation



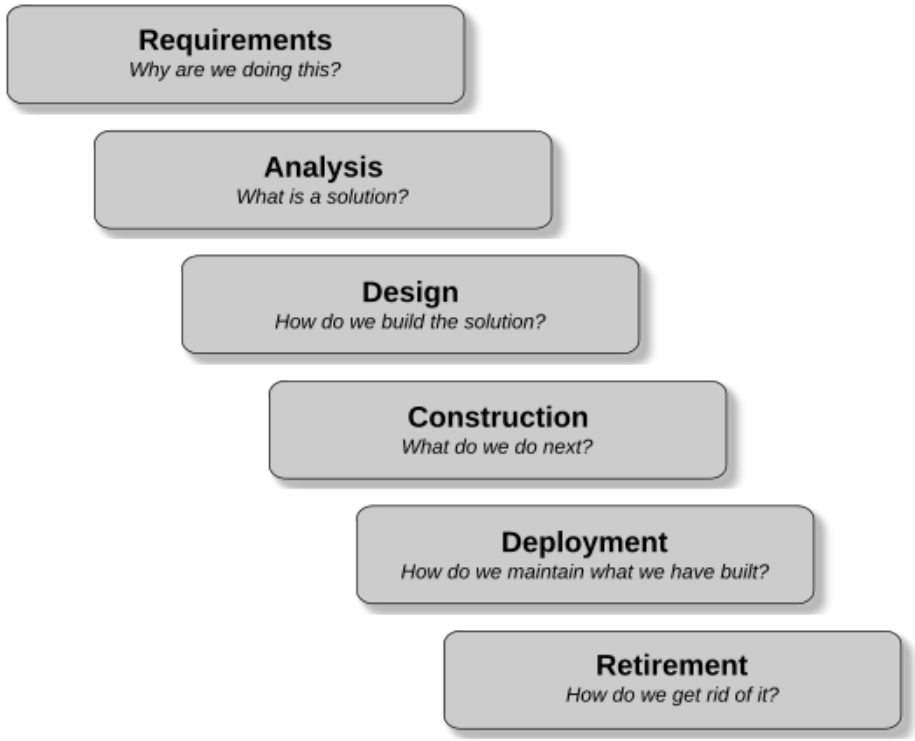
DESIGN

A Design:

- Can be thought of as the project plan
- Defines an application architecture
- Defines the components and how they interact
- Defines the QA requirements and baselines

DevOps approach

- We also define the automation CICD pipelines that will be used
- We design the dev and ops infrastructure as part of the design
- Includes the monitoring, continuous evaluation and feedback
- Also any supporting data engineering



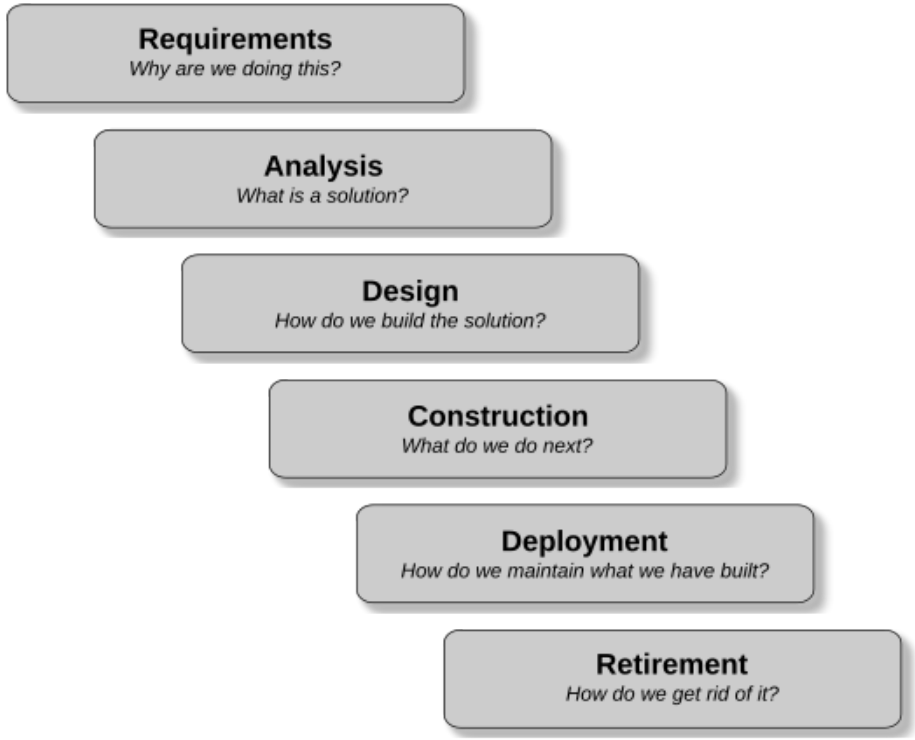
CONSTRUCTION

The product described in the specification is built according to a given design and construction or project plan

- Having a design alone is not enough for something to be built
- Especially as the size and complexity of what you are building increases
- We may also need to build the development infrastructure

You also have to use the appropriate construction techniques and plan out the development activities

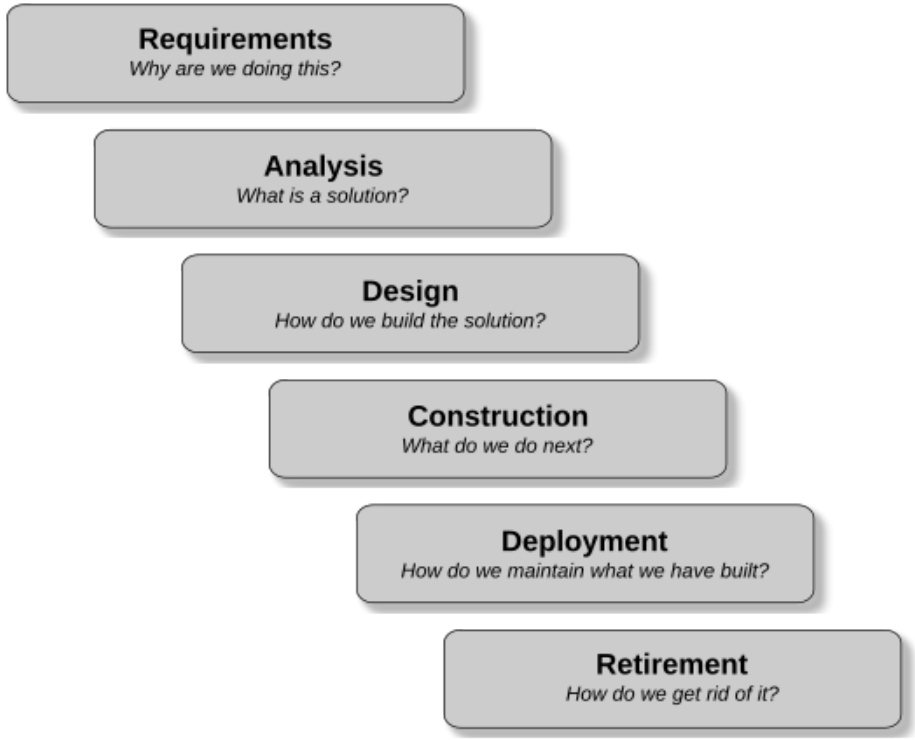
- This is where the actual programming takes place



DEPLOYMENT

The completed solution is deployed when it goes into use
The questions that we have to ask in the deployment phase deal with operational issues such as:

- “How are changes or repairs to be made?”
- “How are users and maintenance people to be trained?”
- “How do we monitor the product for possible malfunctions?”
- “How do we monitor the level of performance in terms of expected SLAs?”



DEPLOYMENT

If we are using a DevOps process

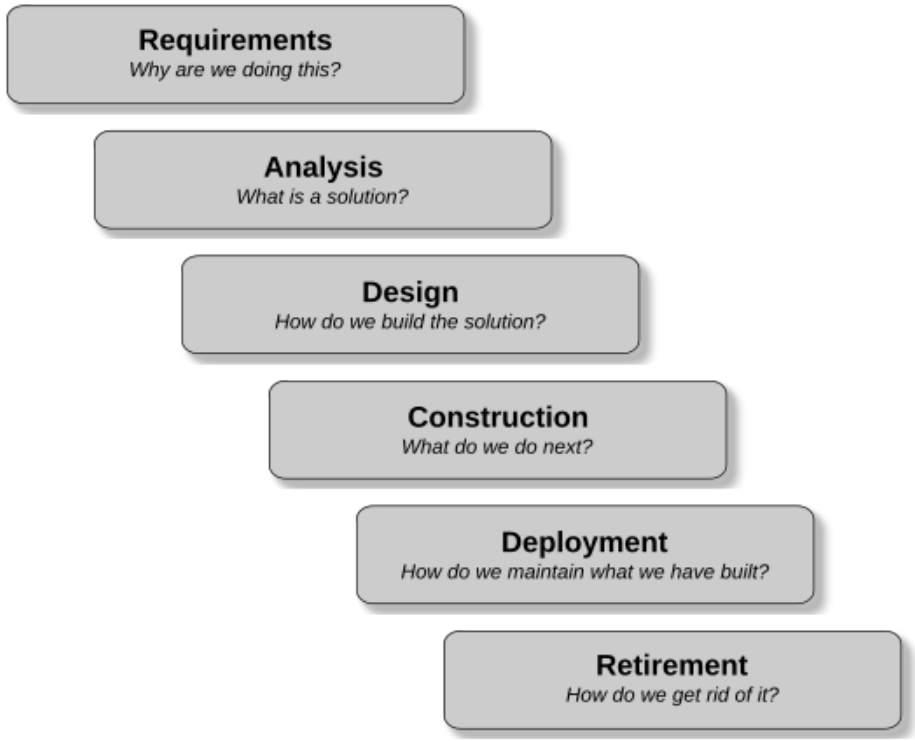
- We will use some sort of staged deployment
- In both production and prototypes

Several basic types

- Canary deployment
- Blue/Green deployment
- A/B deployment
- Rolling deployments

Preferably automated

- Each allows a roll-back path
- In case the deployment goes bad, we can immediately roll back to the previous deployment



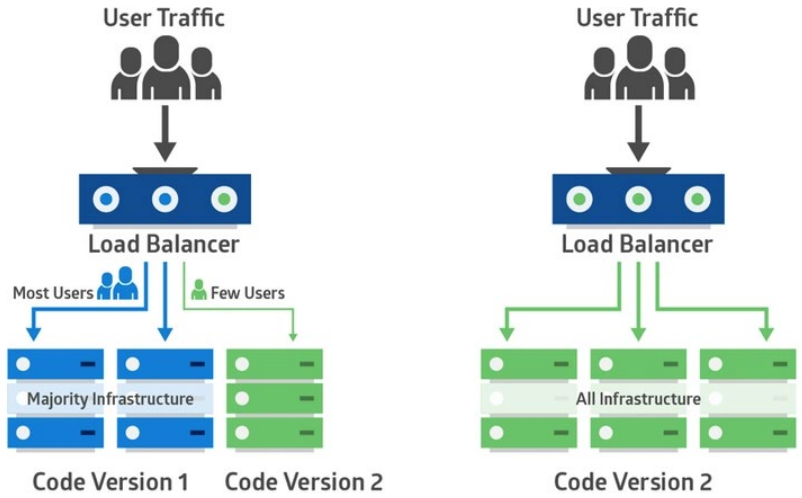
CANARY DEPLOYMENT

Intended to evaluate risk

- Application is deployed in a small part of the production infrastructure
- Only a few users are routed to it
- This minimizes any impact of problems from the new deployment

Performance of the canary is monitored

- Evaluate performance against SLAs and KPIs
- Evaluate performance against the non-canary deployment
- If standards are met, more users are routed to the canary
- The non-canary is left intact in case we have to roll-back during deployment



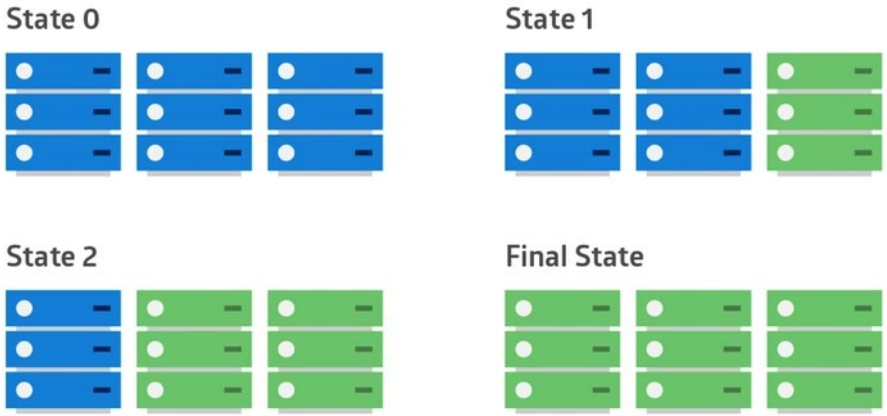
ROLLING DEPLOYMENT

Often used to control issues with scale

- Different rollout targets are identified
- Operational units or regional units
- As each unit is updated, the overall performance is monitored

Scale issues

- The application may be working locally
- But when scaled up, starts to underperform
- A common cause is overloading the infrastructure that supports the application



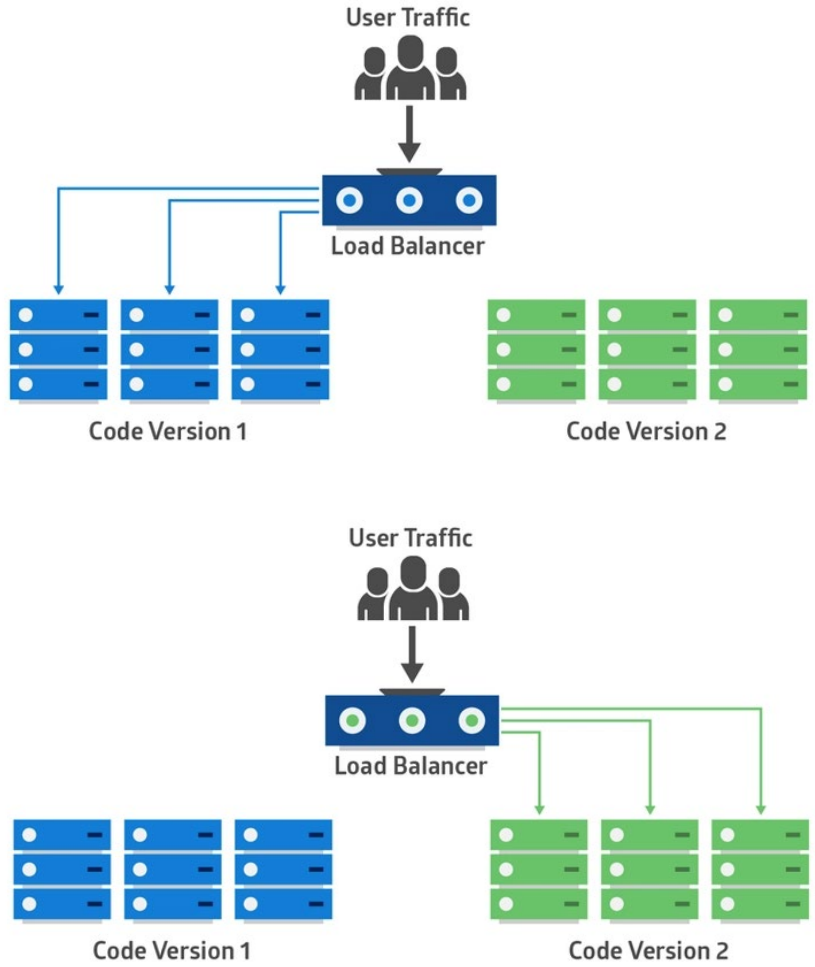
BLUE/GREEN DEPLOYMENT

Similar to a rolling deployment coupled with a canary deployment

- Not as risk sensitive as a canary deployment

How it works

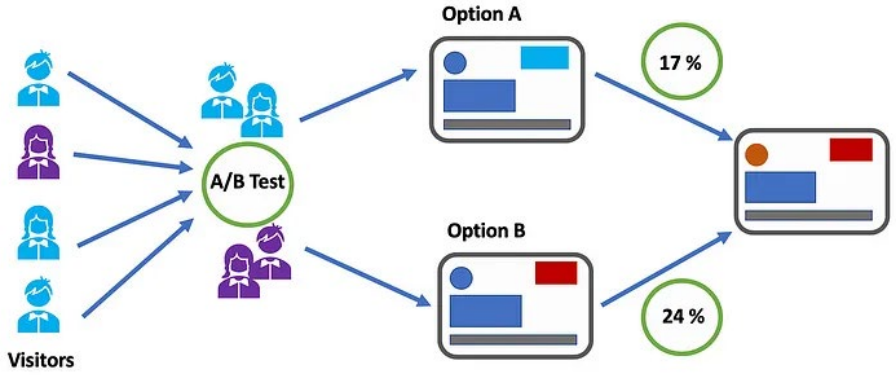
- The new version of the application is deployed in the green environment and tested for functionality and performance
- Once the testing results are successful, application traffic is routed from blue to green.
- Green then becomes the new production
- If testing fails, we can switch back to blue
- Eventually blue version is archived
- Usually used for upgrades to an application



A/B DEPLOYMENT

Used to test two different alternatives

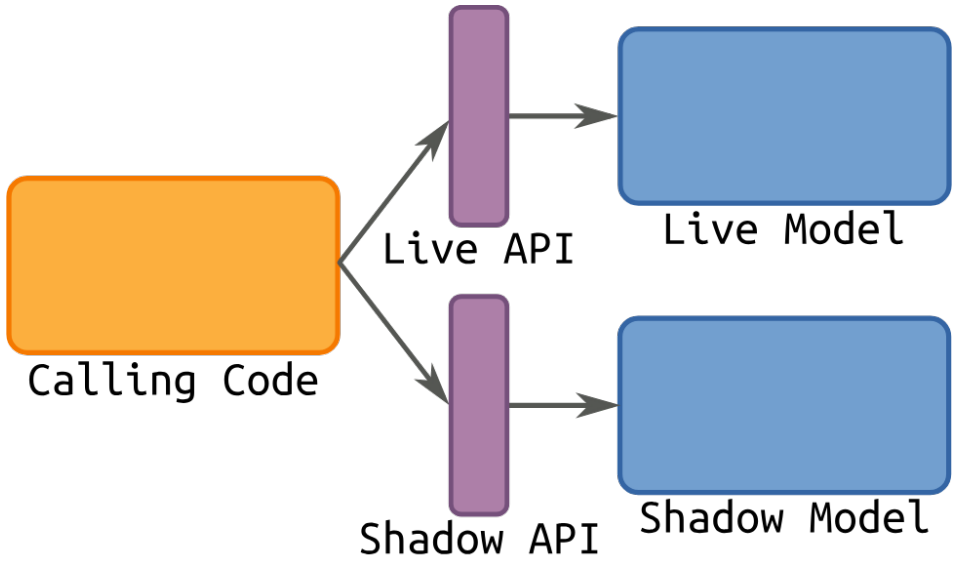
- Some users are routed to version A, some to version B
- Then the performance of each version can be compared
- For example, a decision support tool using two different user interfaces



SHADOW DEPLOYMENT

The new application is deployed in parallel with the existing system

- All requests are still handled by the existing system
- They are also routed to the new app which is shadowing the existing system
- Outputs of the shadow deployment are compared with the existing system
- Ensures that the shadow system works to spec before it replaces the existing system



DEPLOYMENT

Prior to deployment, KPIs and SLAs need to be established

Some of these are performance based

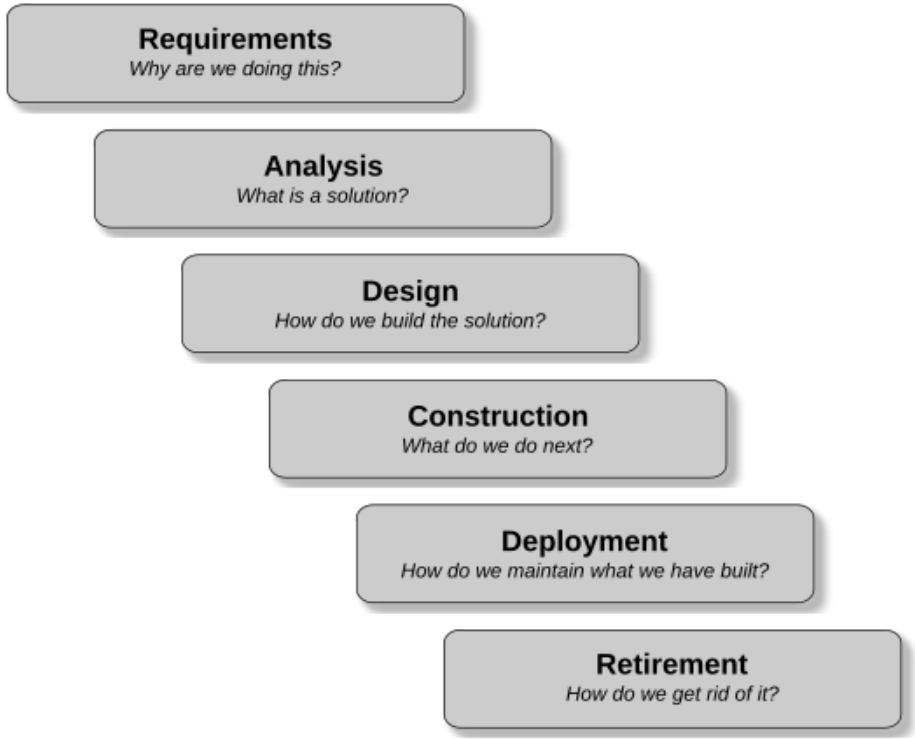
- Often collected and evaluated with automated tools

Some of these are acceptance based

- Requires gathering feedback from users of the application

Some of these are correctness or value based

- Are we getting the right results?
- May require a manual review



RETIREMENT

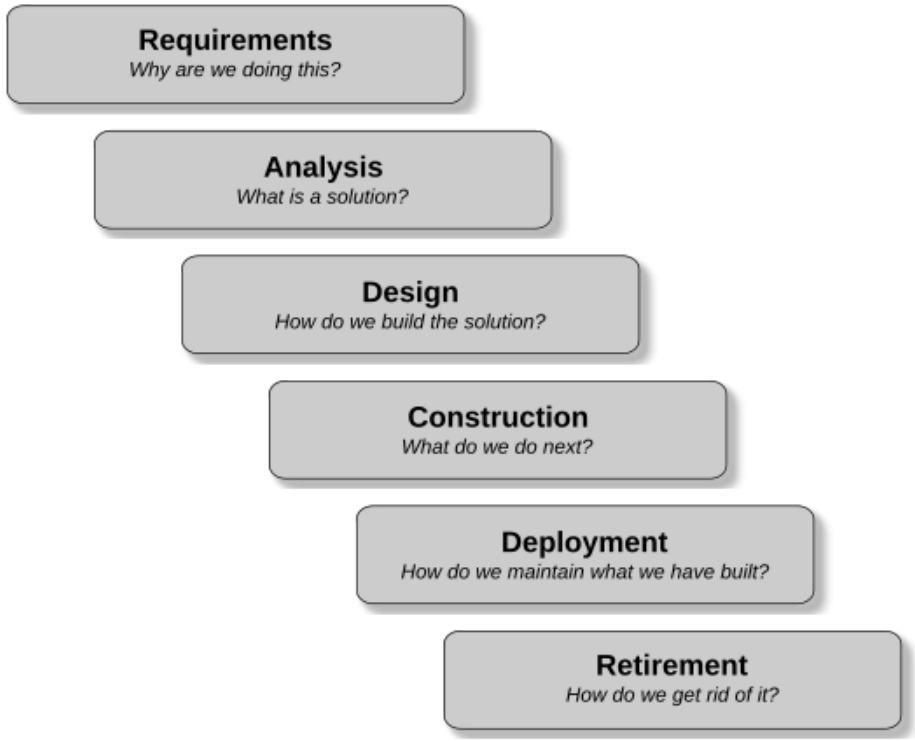
All products and systems have to be taken out of service at some point

- Whether it's a nuclear reactor, a make of car or a software application

How we move the product out of the production and replace it, if necessary, with its successor

Issues we have to consider are:

- Business functional continuity
- Data continuity
- Transfer of functions to new services
- We often need to plan for retirement from the earliest stages of the project



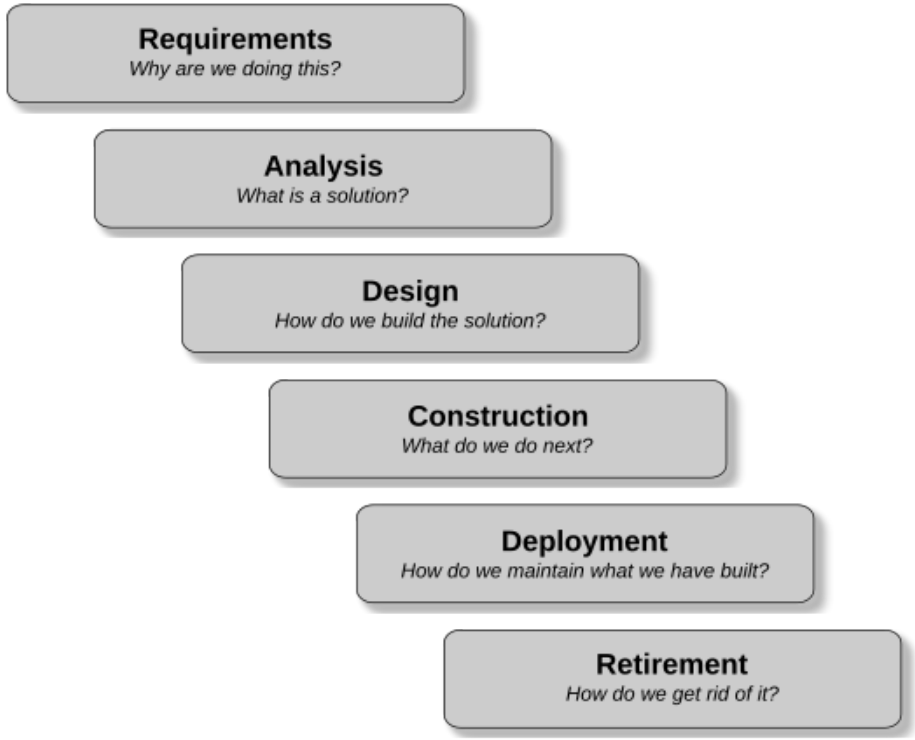
RETIREMENT

Retiring the application is a governance decision

- Do we replace it or upgrade it?

As part of our initial design

- We should have included a retirement facility
- An “un-rollout” strategy for a staged phase out
- Including a potential migration path for data and other artifacts to another system
- Preservation of whatever is needed for records compliance or legal compliance



Q&A AND OPEN DISCUSSION