

# Introduction to DevOps

# Disclaimer

- The information and opinions expressed in this presentation and on the following slides are solely those of the author.
- The following slides are based on Len Bass's used in the course “DevOps: Modern Deployment” at CMU (2018), as reviewed by A. Capozucca

# Software development: workflow

Let  $S_1$  be a system in production.

Then an event  $e$  arrives. This event  $e$  could be (just to mention some):

- an idea for a new feature on  $S_1$ ,
- a merger between two organizations
- a problem with  $S_1$ ,
- a competitive pressure
- a desire for more efficiency

In any case, the event  $e$  causes a requirement for creating a new system  $S_2$  or modifying  $S_1$ .

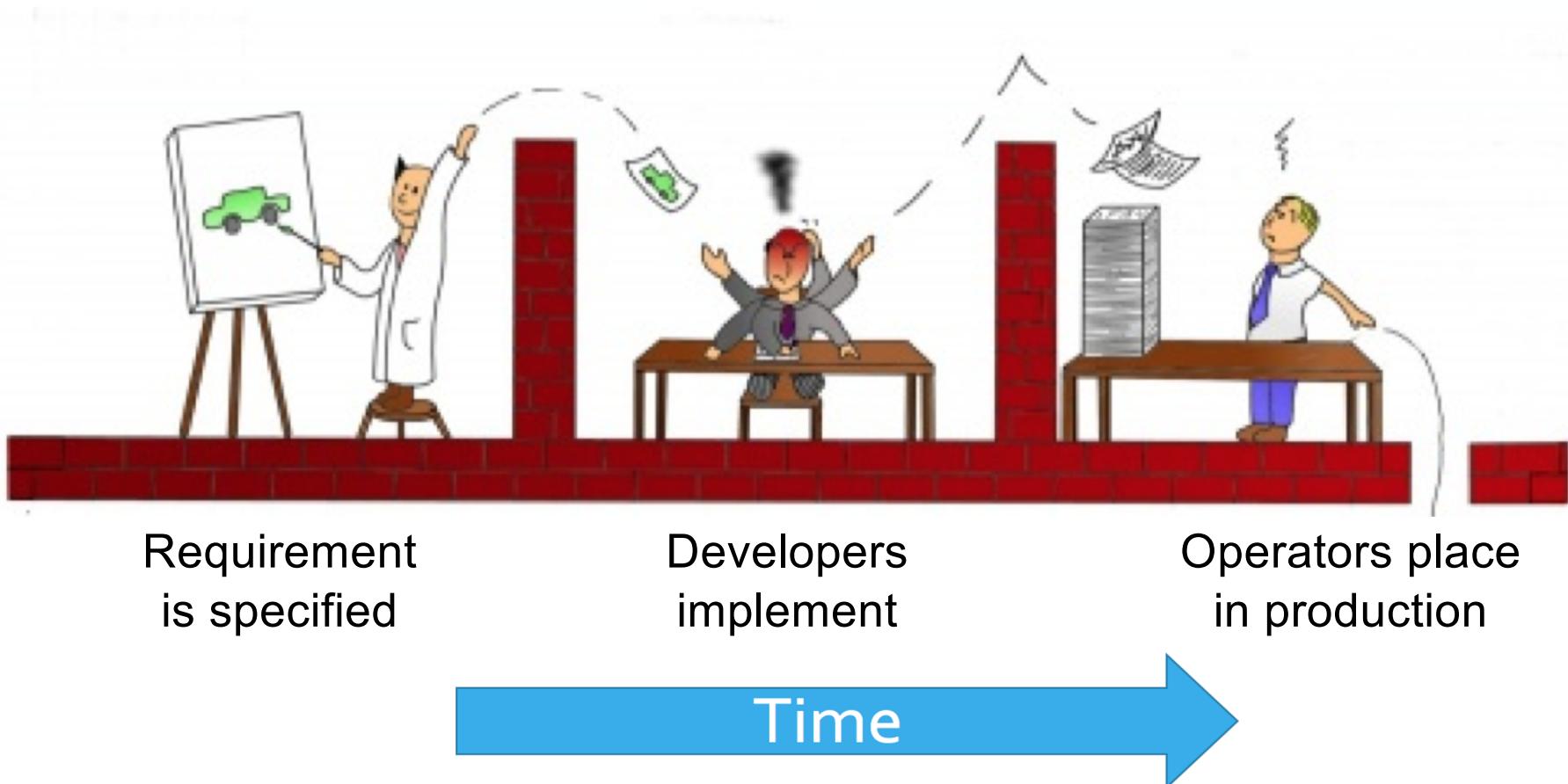
Remark: modifying  $S_1$  is, by far, the most common response.

# Requirements

- The triggering of event **e** causes requirements to be generated
  - Requirements Engineering
    - Elicitation: capture the stakeholders needs
    - Analysis: refine stakeholders needs into “formal” product requirement specification
- “Formal” product requirement specification
  - Documents, Models (Waterfall/RUP)
  - User stories (Agile)
  - Orally (everywhere)
- Requirement specifications are divided and assigned to development teams

Remark: from now on, “requirement specification” = “requirement”

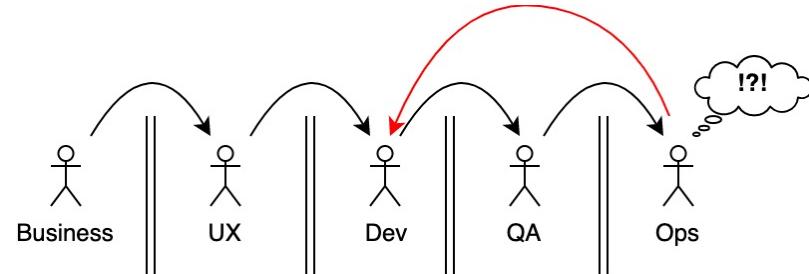
# Over the wall development



# Developers organisation and work

- There are several (and different) development teams.
- Each dev team gets its requirements.
- Developers on each dev team work on requirements.
- At some point the dev team feels they have satisfied the requirements they have been assigned.
- Their code is complete -> their job is done.

# The wall of confusion



- refers to the phenomena that occur when one group in a value stream approaches their job as complete when they've passed it onto the next group.
- The canonical example is when developers throw their code “over the wall” to their ops colleagues and don’t pay attention to what must be done to actually get the software they’ve written into a deployable state
- This often causes the ops team to struggle with getting the software actually deployed, and can lead to significant delays in releases and raised the risk of defects and other problems
- This phenomenon is not limited to development and IT operations

<https://levelup.gitconnected.com/the-wall-of-confusion-623057a4dd26>

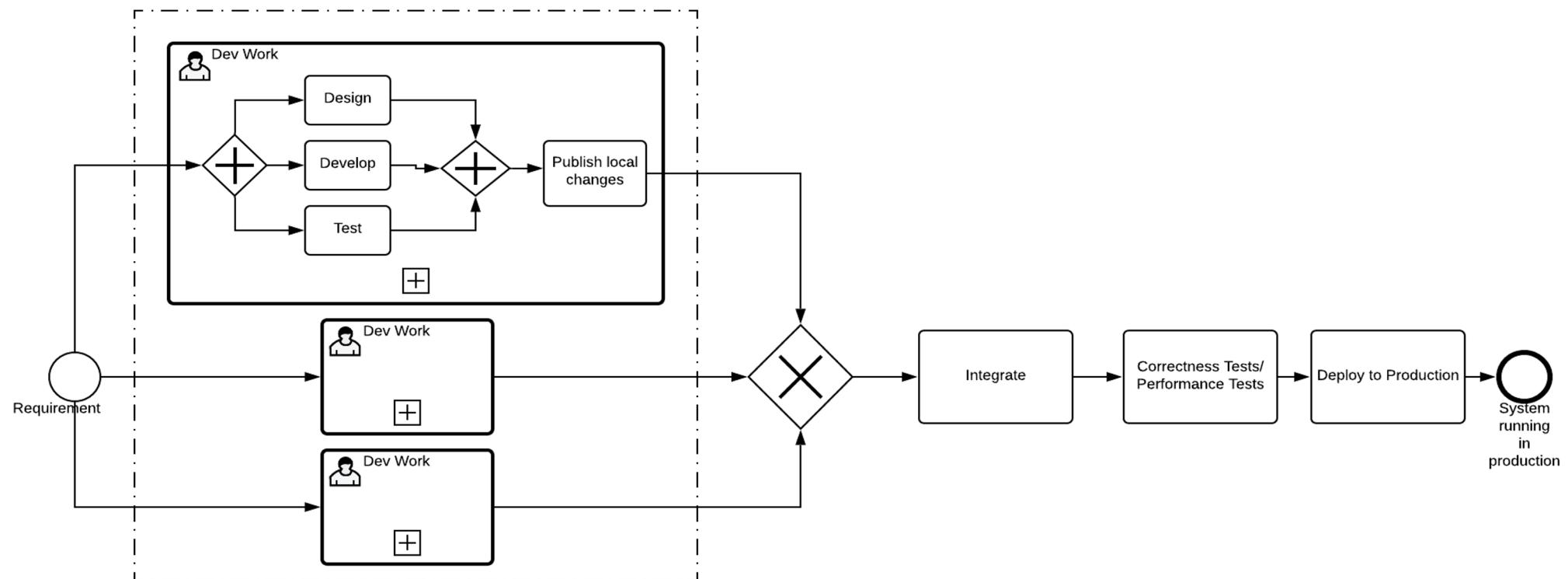
# What is wrong?

- Code Complete  Code in Production
- Between the completion of the code and the placing of the code into production there is a step called: Deployment
- Deploying completed code can be **very time consuming** because of concern about errors that could occur.

# Multi-team development workflow

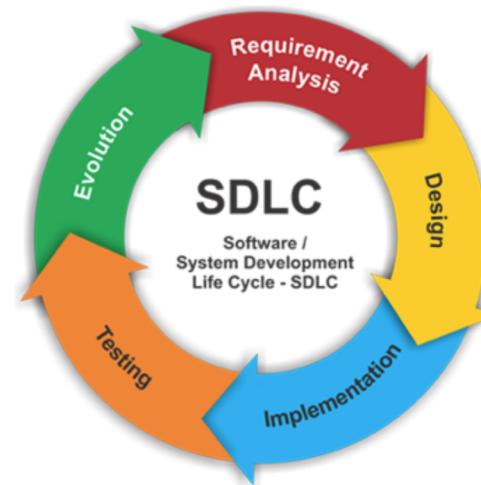
- You **design, develop** and **test** your code in isolation.
- Your code is **integrated** with code developed by other team mates or teams to see if *an executable* can be constructed.
- The built system is **tested** for correctness.
- The built system is **tested** for performance and other qualities.
- The built system is placed into **production**.

# Multi-team development workflow (BPMN 2.0)



# Software Development Life Cycle (SDLC) model

- SDLC: it is a series of **phases** that provide a common understanding of the software building process.
- SDLC Examples (\*)
  - Waterfall
  - V-Shape
  - Prototyping
  - Spiral
  - Iterative and Incremental (eg. RUP)
  - Agile



*What is the SDLC that corresponds to a multi-team development workflow?*

(\*) Mohamed Sami, (March 15, 2012). "Software Development Life Cycle Models and Methodologies," in Mohamed Sami - Personal blog. Retrieved August 12, 2019, from <https://melsatar.blog/2012/03/15/software-development-life-cycle-models-and-methodologies/>.

# Iterative and Incremental - multiteam

## Pros

- Produces business value **early** in the development lifecycle.
- Better use of **scarce** resources through proper increment definition.
- Can accommodate some **change requests** between increments.
- More focused on **customer value** than the linear approaches.
- We can detect **project issues** and changes earlier.

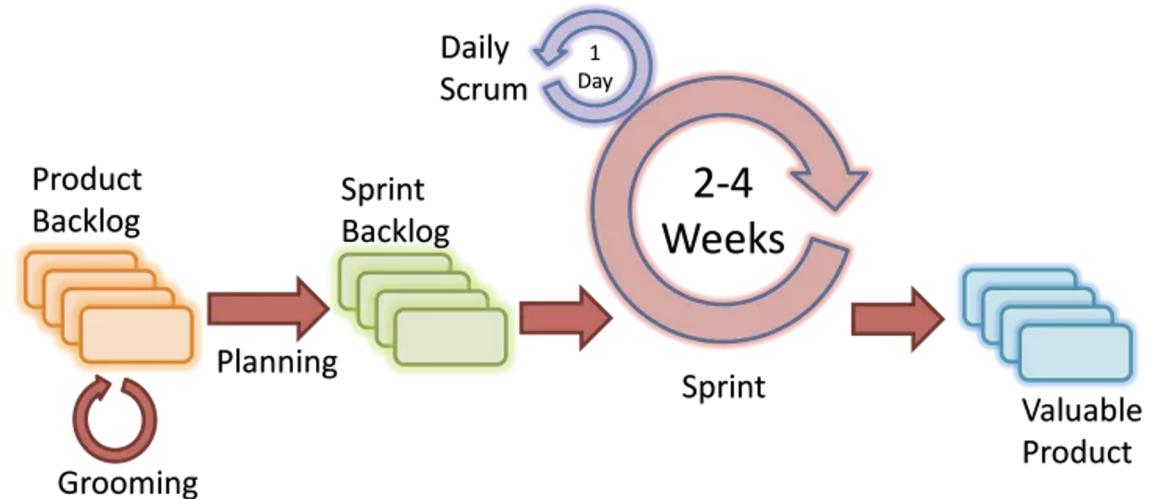
## Cons

- Requires **heavy documentation**.
- Follows a defined **set of processes**.
- Defines increments based on function and feature dependencies.
- Requires more **customer involvement** than the linear approaches.
- **Partitioning** the functions and features might be problematic.
- **Integration** between the iterations can be an issue if it is not considered during the development and project planning.

# Agile

## Pros

- Decrease the **time required** to make use of some system features.
- Face to face communication and continuous inputs from **customer representative** leaves no space for guesswork.
- The end result is the **high-quality** software in the **least possible time** duration and satisfied customer.

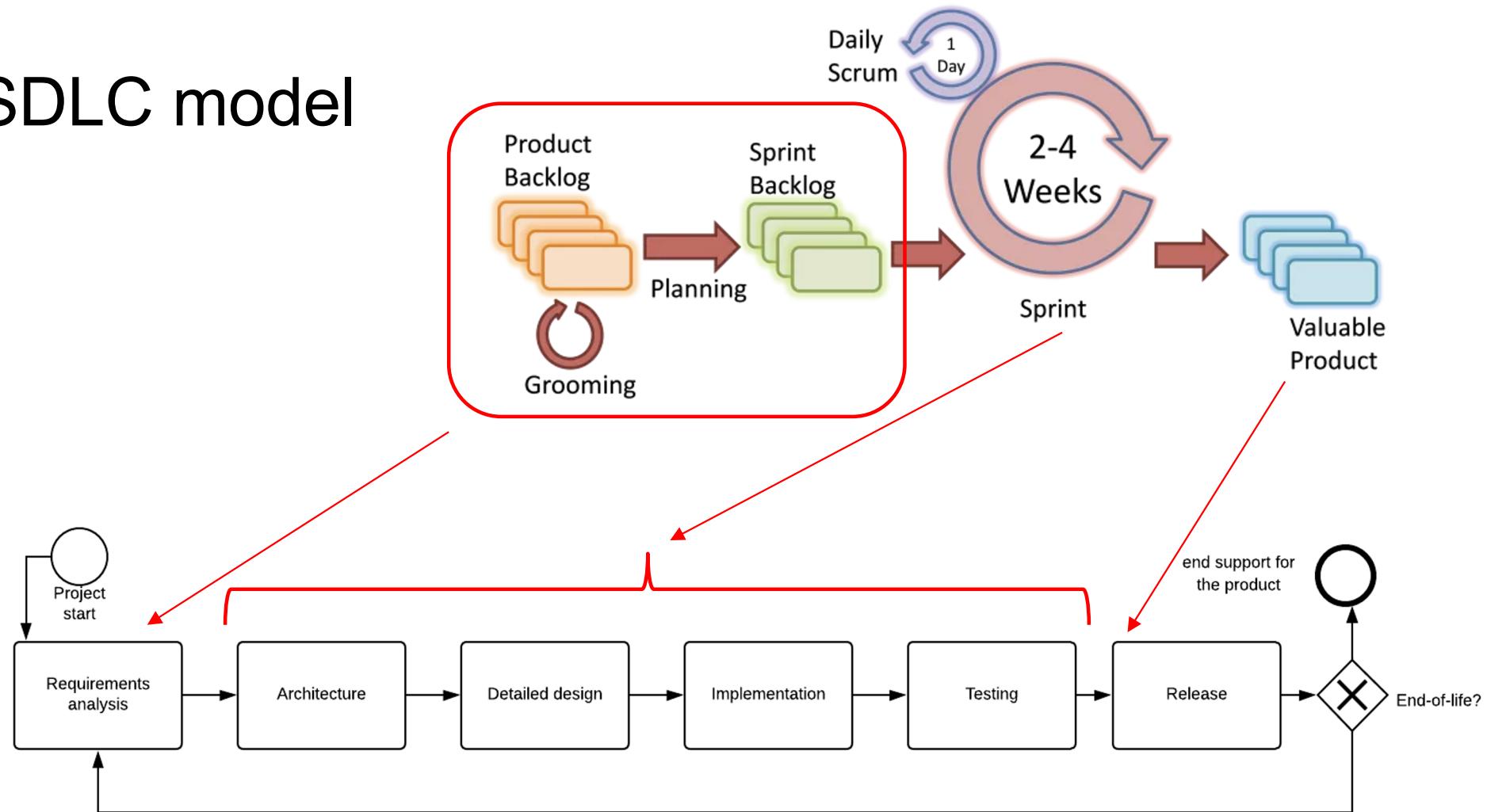


## Cons

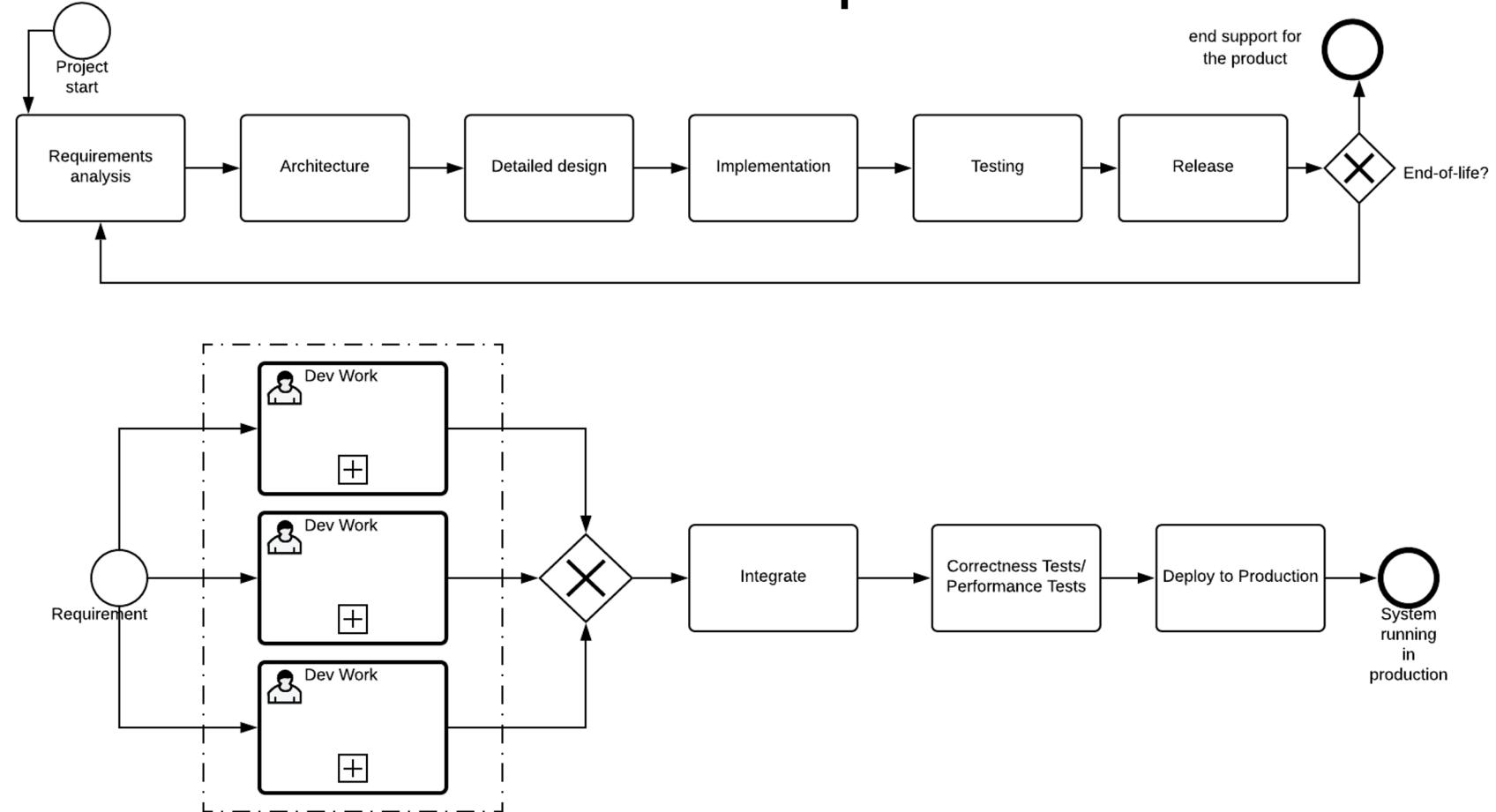
- **Scalability. (\*)**
- The ability and collaboration of the **customer** to express user needs.
- **Documentation** is done at later stages.
- Reduce the usability of **components**.
- Needs **special skills** for the team.

(\*) Read [Agile at Scale](#)

# SDLC model

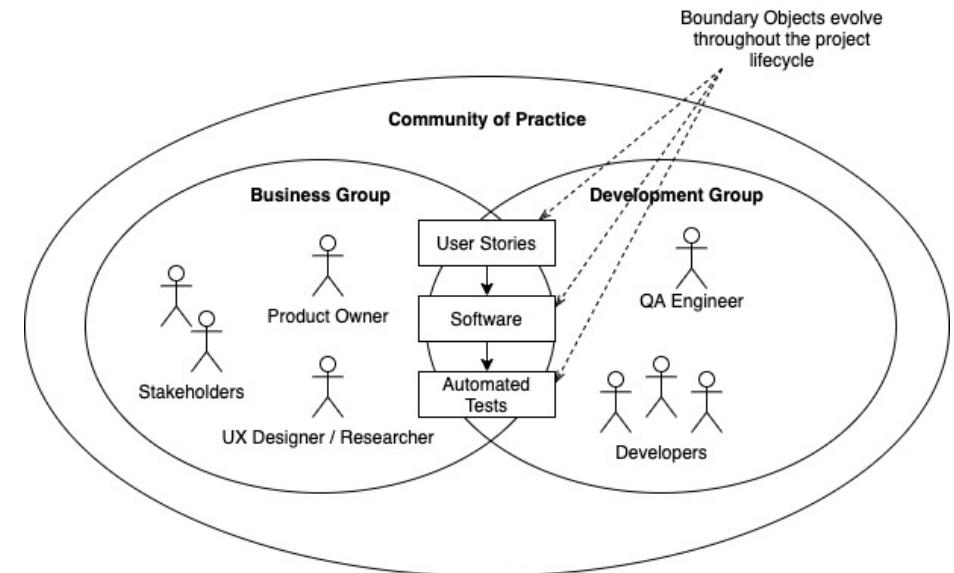


# SDLC vs. Multi-team development



# Boundary objects

- In earlier stages of the project, a rapid prototype or wireframes might be the boundary object. In later stages, it might be automated tests.
- [Agile User Stories](#) are a popular example of boundary objects. [Kanban cards](#) are another take on this sort of boundary objects. Both types are formatted to be small enough to follow the work through the system, but can also contain deeper layers of information in the form of [Gherkins](#), diagrams, mockups, discussions, decisions, and any other relevant information needed to get the work done.



What can go wrong?



# What can go wrong? – Integration

- Not all portions of the system are available
  - Portions developed by other teams
  - Portions developed by 3rd party
  - Names and signatures of methods from other software are inconsistent
- Sequencing errors
  - Other teams do not follow the contract with your code in terms of sequence of method calls
- Version incompatibility
  - Your team assumed version A of 3rd party software but the build downloads version B

# What can go wrong? – Integration (2)

- Data problems
  - Database data is not refreshed for each test
  - Data does not flow correctly to your code
- Configuration problems
  - Configuration parameter settings for code developed by different teams is incompatible
  - Configuration parameters are not specified
  - External services are not reachable for security or configuration reasons
- ...

# What can go wrong? – Performance / Acceptance Test

- Configuration problems
  - External services not available because of lack of permissions
  - Inconsistent configuration settings
  - Leaking into production environment
- Data problems
  - Database is not representative of production database
  - Stale database after tests
- ...

# What can go wrong? – Production

- Configuration problems
  - Requires authentication and authorization
  - Inconsistent configurations
- Performance problems
  - Under actual load, system may not have adequate performance
- Logical problems
  - May require new version to be rolled back
  - Database may have been corrupted
- ...

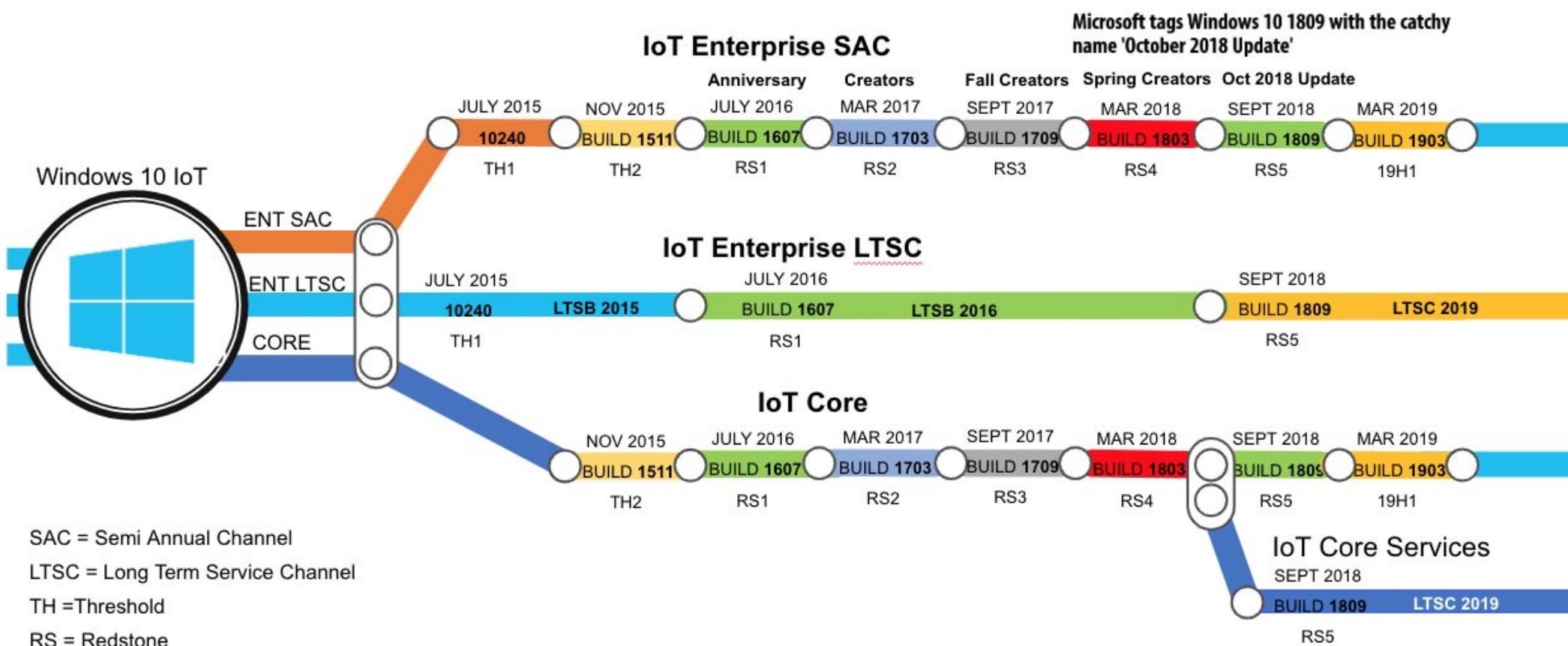
# Time is passing

- Every error must either be **corrected or prevented**.
- Preventing errors can be done through some combination of:
  - Software Development Process
  - System Architecture
  - Tooling
  - Coordination among teams.
- Coordination **takes time**.
- Correcting errors **takes time**.

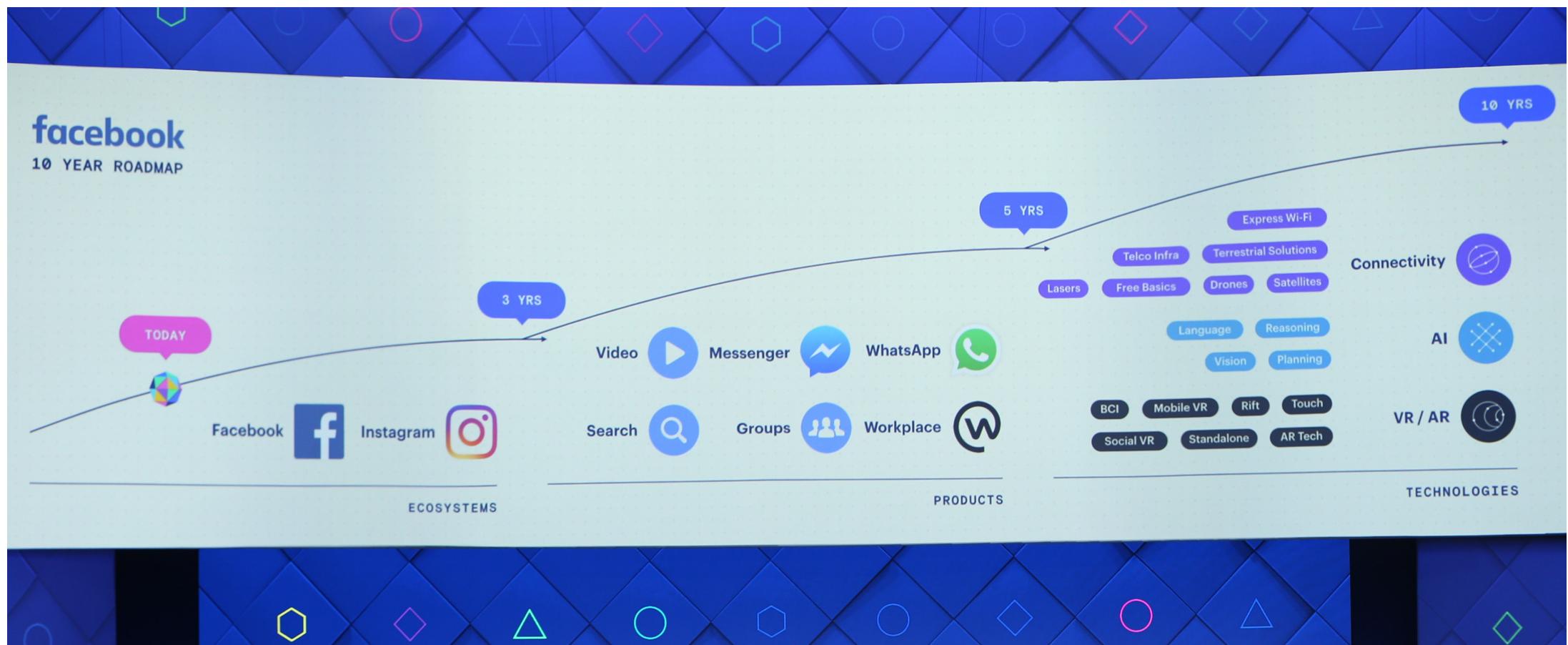
# How much time?

- Historically, for complex software products (eg. Office or Windows) releases are scheduled for once a quarter, or once a year to give time to coordinate and adequately test.
- This means there may be **months delay** before a new concept or feature is added to a system.
- **This delay has become more and more unacceptable.**
  - Internet companies deploy **multiple times a day**
    - Amazon had a new release to production every **11.6 seconds** in May of 2011.
      - Velocity 2011: Jon Jenkins, "[Velocity Culture](#)". Jump to min 10:00 to go straight to the point.

# Windows 10 IoT Road Map 2019

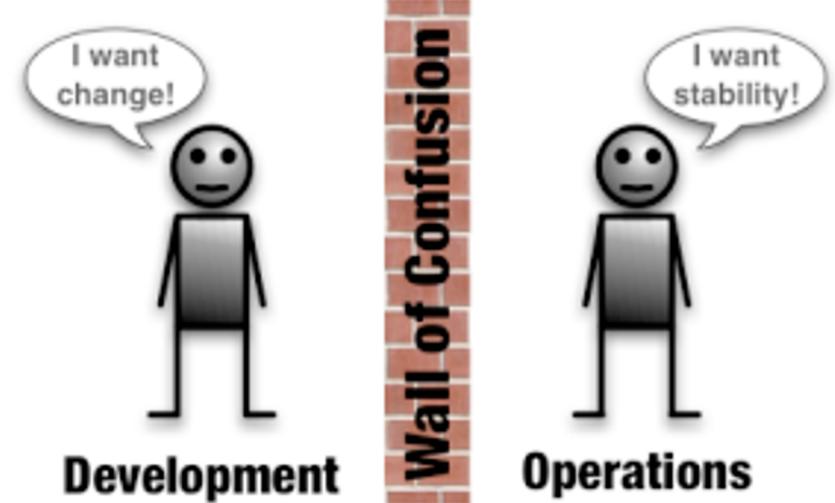


# Facebook roadmap (Zuckerberg 2016)



# Problem

- Velocity of new releases (Dev)
  - The businesses want to get valuable new software to keep increasing revenue
    - Pressure over developers.
- Ensuring quality of service (Ops)
  - If there is no requirement for quality, then releasing that the system being released is usable.
    - Ops see any change as a risk.



## What can we do?

see "[Balance Velocity and Risk by Having DevOps Teams Earn Leaner Processes](#)" (2018)

# What is DevOps?

- No standard definition of DevOps until this year.
  - “DevOps is a cultural movement” [2]
  - “DevOps is interaction between development and operations personnel” [4]
  - “DevOps is ...” -> see [3] for a long list of definitions
- A standard definition of DevOps has been issued recently:
  - [IEEE Standard Initiative Working group](#) created in 2016. [6]
  - The final standard IEEE Std 2675-2021 [9]

# IEEE Standard 2675-2021 DevOps



Contents	
1. Overview....	9
1.1 Scope....	9
1.2 Purpose....	10
1.3 Word usage .	10
2. Normative references...	11
3. Definitions, acronyms, and abbreviations.....	11
3.1 Definitions .....	11
3.2 Acronyms and abbreviations.....	14
4. Conformance ...	16
4.1 Compliance criteria....	16
4.2 Full conformance to outcomes.....	17
4.3 Full conformance to tasks....	17
4.4 Tailored conformance ..	17
5. DevOps concepts....	17
5.1 Value of DevOps...	17
5.2 DevOps principles...	18
5.3 DevOps and organizational culture.....	20
5.4 DevOps and life cycle processes .....	23
6. Relation of software life cycle processes to DevOps.....	23
6.1 Agreement processes....	23
6.2 Organizational Project-Enabling processes...	27
6.3 Technical Management processes .....	38
6.4 Technical processes... 61	
Annex A (informative) Bibliography .....	88

# DevOps definition from the IEEE standard 2675

- *Set of principles and practices which enable better communication and collaboration between relevant stakeholders for the purpose of specifying, developing, and operating software and systems products and services, and continuous improvements in all aspects of the life cycle [9]*
- Principles and the related cultural background (agile, lean) are included in the definition (DevOps is a social construct)



# DevOps definition for us

- “*DevOps is a set of **practices** intended to **reduce the time** between committing a change to a system and the change being placed into normal production, **while ensuring high quality.***” [1]
- Goal oriented definition
- DevOps practices involve developers and operators’ processes, architectures, and tools.
- These processes, architectures, tools are aimed at:
  - Preventing errors
  - Speeding up the detection and correction of errors

# Four times implied by our definition

*“DevOps is a set of practices intended to reduce the time between **committing** a change to a system and the change being **placed into normal production**, while ensuring high quality.”*

1. **(P1) Pre-commit:** The time during which a developer has sole, private access to the code.
2. **(P2) Commit-Deployment:** The time during which the code is being built, tested against other portions of the system and being prepared for deployment.
3. **(P3) Probation:** The time between the code being placed in production (or production-like) and the developers/operators having confidence in its quality.
4. **(P4) Normal production:** The time after the developers/operators have confidence that the quality is equal to other portions of the production system.

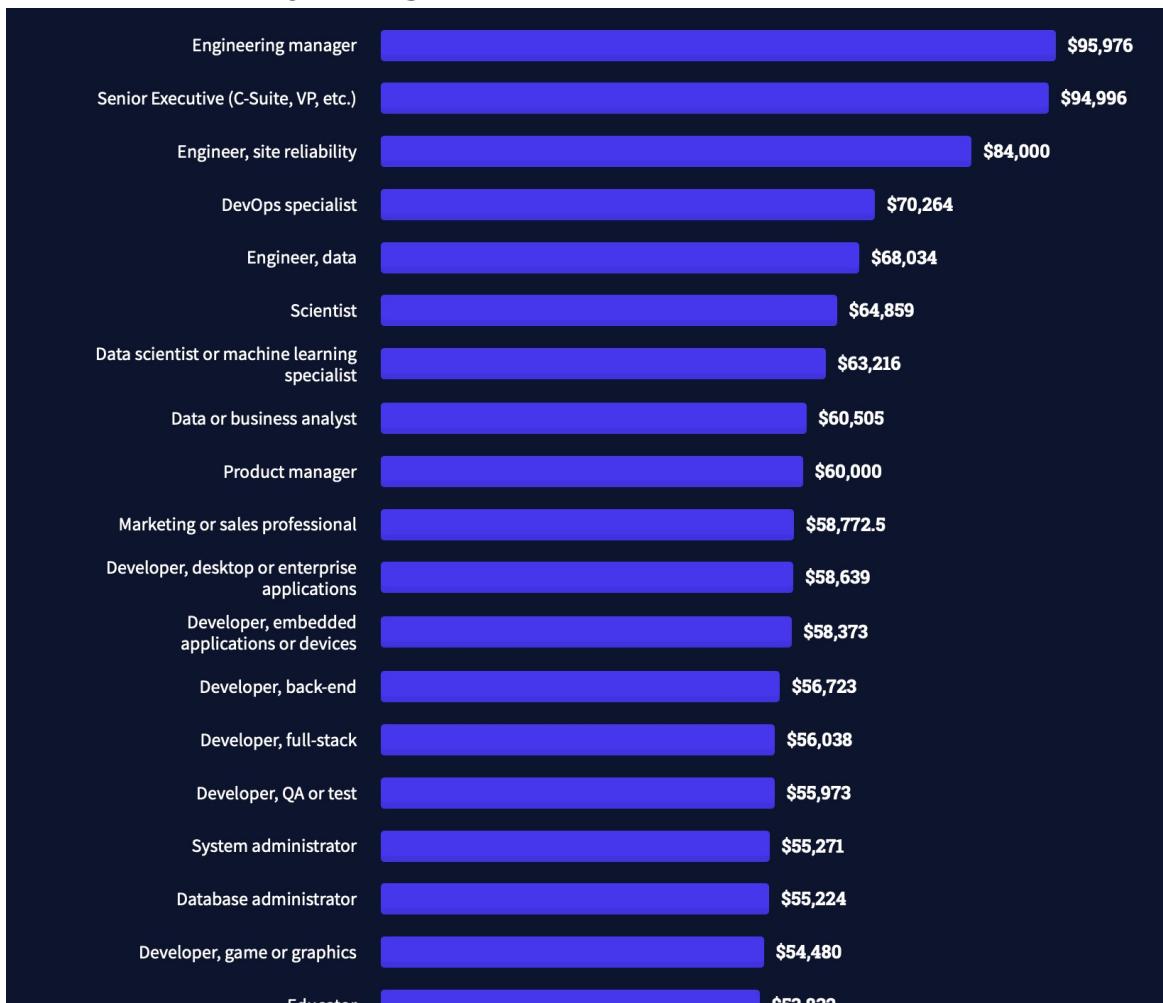
# Team structures & Roles

- Size
  - “Two pizza rule”
- Structure
  - DevOps department
  - DevOps team
  - NoOps (i.e. individual)
- Roles
  - Developer
  - Service owner (i.e. service interface)
  - Reliability engineer (i.e. firefighter)
  - Gatekeeper (aka release engineer/coordinator)
  - DevOps Engineer (dev/support the DevOps environment)



Reading: [1] (pp 13-14), [4], and [5] (pp 9-12)

# Today's job offers (or why it's a good to know about DevOps)



Salary by Developer Type,  
in 2021,  
in the world,  
in USD

Source: Stack Overflow Survey

# More implications coming from the definition

- “*DevOps is a set of **practices** intended to reduce the time between committing a change to a system and the change being placed into normal production, while ensuring high quality.*” [1]
- What is a practice?
  - Def (\*): **action** rather than thought or ideas.
    - In software development [7, page 89], a practice is:
      - “*an activity or mode of working*”
      - It MUST be
        - performed regularly (activity)
        - enforced systematically (mode of working)
          - otherwise, it’s only a nice technique
    - A practice is implied by a **principle**.

(\*) [Cambridge Dictionary](#)

# Agile best practices: examples



These determine KPIs, reports and other metrics per user, in order to monitor that the process is working as agreed

Weekly delivery of developments for the end user to operate and request for changes. It is IDEAL that the end user is part of the process of development and testing

Mapping processes to determine the starting point and the 'Actual State'

Optimize a process with a right combination of technologies



# More implications coming from the definition (cnt'd)

- What is a principle?
  - Def (\*): a basic idea or rule that explains or controls how something happens or works.
    - In software development [7, page 49], a principle is:
      - “*A methodological rule that expresses a general view of how software should be developed*”.
      - It MUST be
        - abstract: it is a generic rule.
        - falsifiable: it may make sense to disagree with the principle.
  - Examples:
    - Principle: “Spend less than you make”
    - Practice:
      - “Put 10% of your earnings every month in your savings account”
      - “Any time you save a little cash, transfer it into your savings account”.

# DevOps principles

- P1. Have established close collaboration between software Development and Operations teams.
- P2. Treat Operations as first class citizens for the requirements analysis activity.
- **P3. Automate the “*DevOps environment*” as much as possible.**
- **P4. Include a deployment pipeline in your “*DevOps environment*”.**
- **P5. Make and publish measurements.**
- **P6. Keep everything version controlled.**
- **P7. Do not deliver low quality artifacts.**
- P8. Ensure and enhance team spirit.
- P9. Maximize compatibility between Development and Operations tools.
- P10. Ensure incremental improvement.

Reading: [3] chapter 2

*DevOps environment: represents the collections of services and resources that allow not only the particular system to work properly, but also those required by the organisation to succeed in making such a system and, eventually, achieve its release.*

Reading: [8] pages 277-279.

# (Focused) DevOps principles

- **P3. Automate the “*DevOps environment*” as much as possible:**
  - Automate as much as possible the setup of the *parts* that compose the DevOps environment, and the execution of the *processes it supports*. This leads to garner efficiencies with process speed, consistency and auditability, and above all, it allows to make the (re)creation process less error-prone by reducing manual interventions.
- **P4. Include a deployment pipeline in your “*DevOps environment*”:**
  - A deployment pipeline is a model that describes the process of getting software (whether code, configuration files, database schemes or data) from a main repository into the hands of its expected users. A DevOps environment implements a deployment pipeline.

# (Focused) DevOps principles (cnt'd)

- **P5. Make and publish measurements.**
  - Define and track meaningful metrics. Their semantics should be transparent. The collected data should be accessible to all, capable of being visualised, and drive the feedback loop.
- **P6. Keep everything version controlled.**
  - Everything you need to build, test, deploy, and release the system under development should be kept in some form of versioned storage. The relevant version of every asset should be identifiable for any given build.
- **P7. Do not deliver low quality artifacts.**
  - Write automated tests at multiple levels (unit, component, acceptance) and run them every time a change is made to the system under development, its configuration files, or the environment and software stack where it runs. Everybody on the team (developer, operators, devops engineers) is responsible for the quality of the application at all times.

# DevOps practices

- Pr1. Continuous planning.
- Pr2. Continuous feedback loop.
- Pr3. Performance metrics measurement.
- Pr4. Automated dashboards and monitoring tools.
- Pr5. Continuous monitoring.
- **Pr6. Continuous integration.**
- **Pr7. Continuous deployment.**
- **Pr8. Continuous delivery.**
- **Pr9. Configuration management for code and infrastructure.**
- Pr10. Change management.
- **Pr11. Automated testing.**
- **Pr12. Continuous testing.**
- Pr13. Process standardization.
- **Pr14. Infrastructure as code.**
- Pr15. Stakeholder participation in defining requirements.
- Pr16. Continuous improvement.
- Pr17. Code ownership.
- Pr18. Constant, seamless communication.
- Pr19. Cross-functional teams.
- Pr20. Blameless postmortems.
- Pr21. Shared responsibilities, goals, values as well as mutual respect and trust.
- Pr22. Automated Deployment.
- **Pr23. Automated Code Review.**

Reading: [3] chapter 2.

# (Focused) Practices

Practice	Description
Pr6. Continuous integration	CI is the <b>process</b> of automating the build and testing of code every time a team member commits changes to version control.
Pr7. Continuous deployment	CD is the <b>process</b> to automate the steps aimed at deploying the system generated by the CI process into production, without any human interaction.
Pr8. Continuous delivery	CDe is the same <b>process</b> than CD, but (only) the deployment into production requires human intervention.
Pr9. Configuration management for code and infrastructure	Rely on configuration parameters to <b>adapt</b> how the SUD behaves and moves through various environments on their way to production, as well as easing the <b>setup</b> of the environments.
Pr11. Automated testing	<b>Write</b> test cases such that they can be executed using a tool that does not require any manual intervention to reach the end of the test case execution.
Pr12. Continuous testing	It is the <b>process</b> of executing automated test cases as part of the CI, CD or CDe pipeline.
Pr14. Infrastructure as code	<b>Manage</b> the DevOps environment parts (networks, virtual machines, load balancers, proxies, storage) using descriptive models that can be read, write, maintain and <i>use</i> in the same way as source code.
Pr23. Automated code review	It is the <b>process</b> to verify the source code adheres to a predefined set of rules or best practices using an automated tool. The Automated Code Review is triggered during the Continuous Integration as soon as the developers check-in the code.

# Principles and practices mapping

Practice	Principle				
	P3	P4	P5	P6	P7
Pr6. Continuous integration	X	X		X	
Pr7. Continuous deployment	X	X			
Pr8. Continuous delivery	X	X		X	
Pr9. Configuration management for code and infrastructure	X			X	
Pr11. Automated testing	X				X
Pr12. Continuous testing.	X	X			X
Pr14. Infrastructure as code	X			X	
Pr23. Automated code review	X				X

# DevOps anywhere?

- Domain
  - Trade-off between “time-to-market” vs. “risks that something goes wrong”
  - High level of safety and regulations
    - Financial
    - Automotive
      - Anyway, something may go wrong: see [Subaru](#)
      - Others may try: see [Tesla’s over-the-air update](#)
    - Aerospace
- Product
  - Monolithic architecture
  - Technical debt
- Resources & Costs
  - DevOps Engineer > Devs > Ops (see again [Stack Overflow Survey](#) on salaries)

# References

1. Bass, L., Weber, I.M., Zhu, L.: DevOps : a software architect's perspective. Addison-Wesley Professional New York (2015)
2. Effective DevOps: Building a Culture of Collaboration, Affinity, and Tooling at Scale, O'Reilly Media; 1 edition (June 24, 2016)
3. Konchenko, S.: [Quality assessment of DevOps practices and tools](#), Ms Thesis, University of Luxembourg, 2018.
4. Erich F., Amrit C, Daneva M. A qualitative study of DevOps usage in practice. J Softw Evol Proc. 2017;29: e1885.  
<https://doi.org/10.1002/smri.1885>
5. Hütermann M.: DevOps for Developers, Apress; 1 edition (September 12, 2012).
6. IEEE. 2675 - DevOps - standard for building reliable and secure systems including application build, package and deployment. <https://standards.ieee.org/develop/project/2675.html>
7. Meyer, B: Agile!: The Good, the Hype and the Ugly, Springer International Publishing (2014).
8. Humble J., Farley D.: [Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation](#) (Addison-Wesley Signature Series (Fowler)) Addison-Wesley Professional; 1 edition (August 6, 2010)
9. IEEE Std 2675-2021, DevOps: Building Reliable and Secure Systems Including Application Build, Package, and Deployment, 2021

# Questions?