DevOps – What is it?

- DevOps (a clipped compound of "**development**" and "**operations**")
- A practice that emphasizes the **collaboration** and **communication** of both software developers and other information-technology (IT) professionals while **automating** the process of software **delivery** and infrastructure changes.
- It aims at establishing a culture and environment where building, testing, and releasing software, can happen rapidly, frequently, and more reliably.

More on the Traditional Dichotomy

- Development
  - Development focus
  - Requirements, Design, Coding and testing focus
  - Traditionally, little direct customer engagement
  - More governed by week to week, or month to month concerns
  - May have a maintenance team
  - Relatively little sys admin type activities
  - Programmers, Testers, Analysts
- Operations
  - Deployment focus
  - Often the point of contact for support and maintenance
  - Helpdesks
  - Service Level Agreements (SLAs)
  - More Sys Admin type activities than Development
  - Customer interfacing
  - More governed by day to day concerns?
  - Some scripting and configuration for complex systems
  - Operations Engineers / Support Engineers / Customer Services Rep.

Is DevOps another agile method?

- Organizations that have adopted agile software development are seeing increasing quantities of releases.
- DevOps was essentially born from this increasing popularity of agile development.
- Agile and DevOps are **similar**, but **differ** in a few important respects.
  - Agile represents a change in thinking of standard development teams.
  - DevOps implements **organizational** cultural change.
- One goal of DevOps is to establish an environment where releasing more reliable applications faster and more frequently can occur.

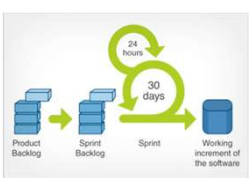
Is DevOps just Continuous SE?

- Continuous Software Engineering and DevOps are similar in their meanings, but they are two different concepts.
- **DevOps** has a broader scope, and centers around the cultural change
  - Specifically the collaboration of the various teams involved in software delivery (developers and operations), as well as automating the processes in software delivery.
- **Continuous SE** is an approach to automate the development and delivery aspects, and focuses on bringing together different processes and executing them more quickly and more frequently.
- They have common end goals and are often used in conjunction to achieve them… and indeed, some consider them to be largely the same thing.

Development vs. Operations

Development
Responsible for implementation of requirements into a runnable software system
Focus on features, confronted with changing requirements

Operations
Responsible for running the systems in production
Focus on stability, confronted with failures, attacks, maintaining infrastructure, etc.



Business, Dev, and Ops

- In many markets, customers want new features often
- New features means new releases
- Dev can implement new features fast – new "runnable" product delivered from every Sprint (~2-6 weeks)
- Ops takes "runnable" product, deploys onto servers
  - Needs:
    - Operating systems, databases and other infrastructure
    - Web servers
    - Load balancers
    - Logging and monitoring
    - Configuration scripts to set up all of the above
  - Often manual, tedious, and error-prone process

Development vs. Operations

- It is not always clear who is responsible for problems
- Examples:
  - After deployment, servers experience high load (! slow)
  - Code isn't building
  - Can't log into the application
- Typical Responses:
  - Devs: "It works on my machine"
  - Ops: "It's not the server, it's your code"
- Conflicting responsibilities
  - Dev process is agile, focus on new features and change
  - Ops process is more static, with an emphasis on stability
- System downtime can cost millions
  - Imagine Vodafone down for 60 seconds…

DevOps "movement" to the rescue

- DevOps is a movement emerging from practice
  - Term 'DevOps' coined in 2009 – first "DevOpsDays" event
- Recognition that "Dev" and "Ops" must collaborate
  - Break down "silos"
- Aim of DevOps is to facilitate Continuous Deployment
  - **Continuous integration** -> test as soon as feature implemented
  - **Continuous delivery** -> ability to deliver new versions quickly
  - **Continuous deployment** -> ability to deploy new version as quickly as possible
- DevOps is not a new team or department
  - Instead it is about collaboration between dev teams and ops teams

Key themes in DevOps

- **Culture**
  - Collaborate across organizational boundaries
  - No 'finger-pointing' (blaming)
  - Improve Communication through involving "Ops" in stand-ups and retrospectives
- **Automation**
  - Automate everything (as far as possible)
    - Builds, deployments, testing, monitoring, system roll-outs, configurations
  - Treat **Infrastructure as code**
    - Store configurations in version control
    - Toolsets such as Chef, Puppet can be used to "program" the infrastructure

Key themes in DevOps

- **Metrics and Monitoring**
  - Logging
  - Measure everything
    - Deployment frequency
    - Mean Time To Failure (MTTF), Mean Time To Recovery (MTTR)
    - Deployment lead time (how long for a feature to be deployed?)
  - Feedback and ability to diagnose problems quickly
- **Sharing**
  - Experiences
  - Train each other
  - Responsibilities and Celebrations (e.g. successful releases)

Principles of Infrastructure as Code

- **Reproducibility** – ability to reproduce an environment
- **Consistency** – if servers have different configurations (e.g. different storage space) then cannot expect same results.
- **Repeatability** – treat every task as if you would have to repeat it -> script it (and test it)
- **Disposability** – "Treat your servers like cattle, not pets" don't make any manual changes, but everything through automated systems. Assume a server may disappear at any time. Also facilitates easy scaling up/down.
- **Service continuity** – ensure that critical services are always available. Decouple these from the rest.
- **Self-testing systems** – automated tests part of development process. Tests provide early feedback.
- **Self-documenting systems** – difficult to update documentation every time scripts are updated. Infrastructure as Code aims to make configurations "self-documenting".
- **Small changes** – easier to fix and debug small changes if things go wrong. Small increments is also motivating – regular stimulation and feedback for developers.
- **Version all the things** – allows traceability (history of changes), roll-back (if things go wrong), transparency (peer review)

Who applies DevOps?

- Lots of companies…
- Ireland: Openet
  - privately owned Irish Software company ~800 employees worldwide
  - builds telecoms network grade software for mobile operators worldwide
  - customers are Tier 1 operators such as AT&T, Verizon, Sprint, BT, Orange, T-Mobile and Eir
  - Extensive use of tooling: build, automated test, virtualization, source code control, continuous integration, automated deployment.

Interesting DevOps Link

- https://www.youtube.com/watch?v=_I94-tJlovg&feature=youtu.be

Continuous Software Engineering

- Core to the concept of CSE is Continuous Integration: a set of software engineering practices that **speed up the delivery** of software **by decreasing integration times**.
- Emerged in the Extreme Programming (XP) community.
- Perhaps there is no single agreed definition for the full scope of CSE?

Continuous Software Engineering

- Described as "a software development practice where members of a team **integrate** their work **frequently**, usually **each person** integrates **at least daily** - leading to multiple integrations per day, where **each integration is verified** by an **automated build** (**including test**) to detect integration errors as quickly as possible".

Continuous Software Engineering

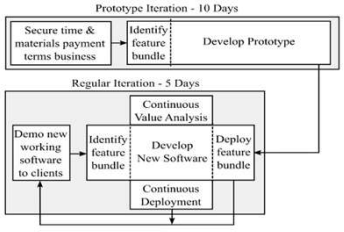
- Defined as "as the application of **automation** via **tools** to **increase** the **deployment frequency** of new releases of **commercial-grade software**".
- CSE permits software feature delivery at rates which **just a few short years ago** may have been considered **unachievable**.
- Some take the view that one can consider CSE to be the **next major wave in software engineering** to follow the agile software development and lean software development movements.

Continuous Software Engineering

- CSE **only possible** recently due to a convergence of **tools** in a **maturing tooling** landscape
  - configuration management tools, such as *Git*
  - build tools such as *Jenkins*
  - deployment tools, such as *Docker*.
- To fully achieve its goals, CSE may also require **architectural rethinking**, e.g. microservices architectures, which seek to deliver small, self-contained and rigidly enforced atoms of functionality (i.e. can deliver small atoms of functionality – but they are not capable of breaking other pre-existing atoms).

Continuous Software Engineering

- One possible CSE process architecture overview:



Continuous Software Engineering

- This particular process architecture requires five key **technology enablers**:
  - (1) **JavaScript and Node.js** which enable extremely rapid code development by utilizing the same programming language across the entirety of the system;
  - (2) Alongside a **distributed microservices architecture**, under which the system is broken down into a set of discrete co-operating processes, typically each service is of the order of several hundred lines of code only;
  - (3) The adoption of **cloud technologies**, whereby all infrastructure may be treated as code, supports the rapid creation of testing and production environments;

Continuous Software Engineering

- This particular process architecture requires five key technology enablers:
  - (4) The architectural approach is coupled with a **continuous deployment model**, layered over the Docker **container engine**, whereby individual services (or several services at a time) may be deployed without perturbing the system as a whole;
  - (5) High quality achieved through steps such as **code commit hooks** via *GitHub* and the *Travis* **Continuous Integration** tool set to construct continuous delivery pipelines.

About Hooks

- "Hooks" provide a mechanism to automatically execute some custom scripts following certain actions, for example committing code and merging code.
- Generally speaking, there are two types: client-side and server-side hooks.
- Client-side hooks are triggered by events including committing and merging, while server-side hooks execute on network operations, for example when receiving pushed commits.
- This allows us to automate actions when events are triggered. More on Git commit hooks here:
  - https://git-scm.com/book/en/v2/Customizing-Git-Git-Hooks

Continuous Software Engineering

- Advantages of CSE:
  - reduced integration risks (errors are found early)
  - motivation (you see a working system)
  - Continuous integration is also an alternative to 'big bang' integration, where all modules are combined in one go, and which usually results in large numbers of errors, hard to isolate and correct

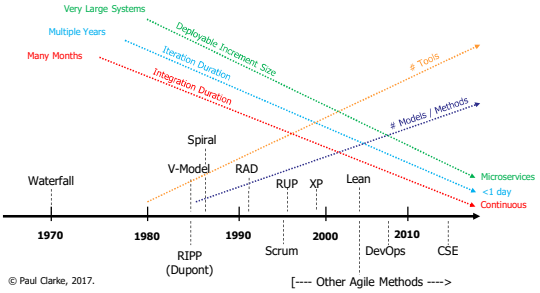
Continuous Software Engineering

- Disadvantages of CSE:
  - Potential degeneration of architecture due to lack of focus on overall design and time spent on too frequent releases of too poor quality
  - implementing continuous integration in large software development organizations is challenging because of organizational, social and technical reasons
  - large software development organizations may be unable to rapidly prioritize the test cases which can be executed quickly and trigger the most failures as early as possible

Continuous Software Engineering

- Software Engineering beyond "development" and "operations"
- Continuous software engineering refers to the **organizational capability** to develop, release and learn from software in very short rapid cycles, typically hours, days or very small numbers of weeks
- Continuous Software Engineering: **all activities** related to continuous software development and usage: dev, ops, use, budgeting, planning.

Historical Review – General Patterns



Factors contributing to iteration duration decrease

- Several Reasons, including
  - **Architectures**
  - **Tooling**
  - **Methods**
  - **Technology**

Process Predictions?

- Consolidation in the tooling space?
  - There remains plenty of room for further growth, but long-term consolidation is foreseeable. Who wants a 15 piece tool chain if 1 tool will do?
- Consolidation in the model / method space?
  - Who wants 50 different (if similar) models / methods if just a small number will suffice?
  - Perhaps a theory for software engineering will emerge.
  - Perhaps this theory will catalyse method consolidation.
- Further reductions in deployable increment size?
  - This is likely but can it go much lower than a single, small microservice? Would there be any real value in it doing so?

Process Predictions?

- Further reductions in iteration durations?
  - This is likely but how far can it go? It is already down to ~1 hour… could it ever go to 1 minute? Unlikely, as it will take longer than that to write most pieces of useful code!

Questions