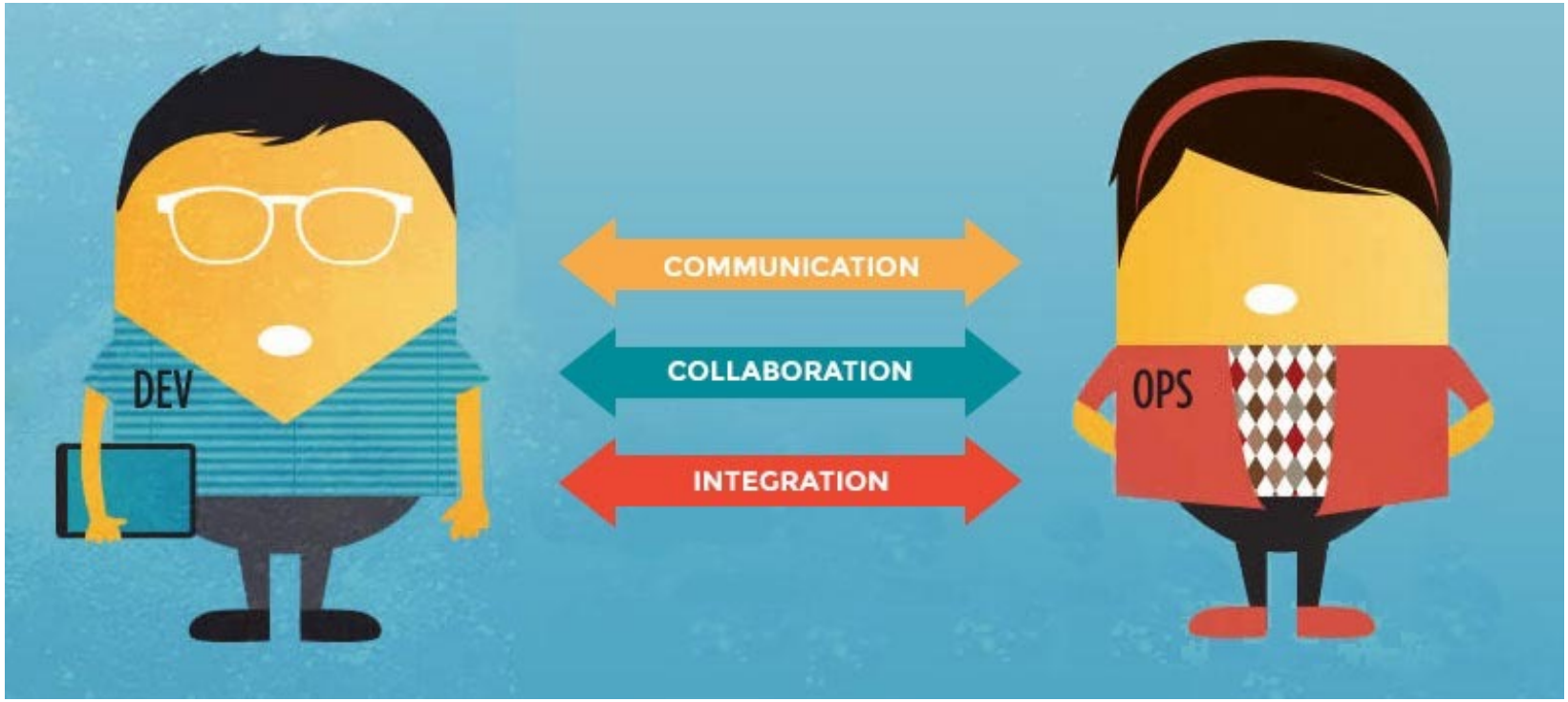


DEVOPS/CONTINUOUS INTEGRATION/ CONTINUOUS DEPLOYMENT FOR BEGINNERS

Dave Wade-Stein

BRIEF HISTORY OF SOFTWARE DEVELOPMENT

JOSEPH MARIE JACQUARD: THE AUTOMATOR

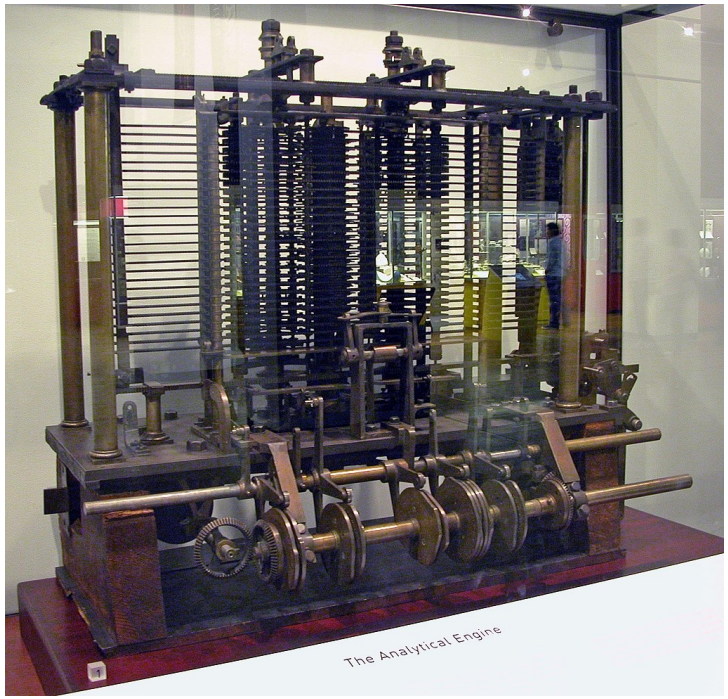
- "Jacquard Machine" (1804) fitted to a loom in order to simplify manufacturing textiles
- punched cards(!) were used to control the pattern of the weave made by the loom
- precursor to programming
- ...and its proliferation was decried by the...

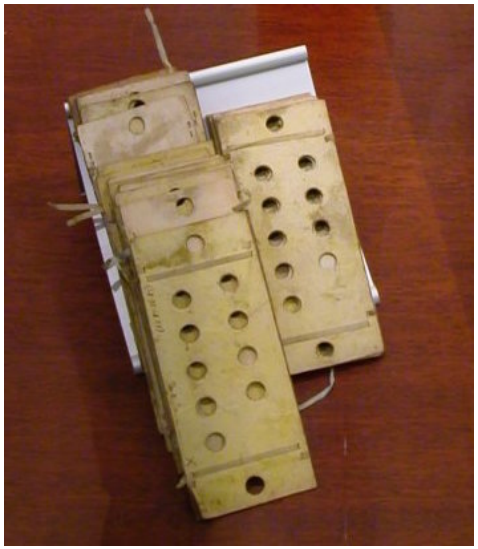
LUDDITES (1811–1816)

- they protested the use of machinery in a "fraudulent and deceitful manner" to circumvent standard labour practices
- they were afraid that the time they spent learning their craft would go to waste as machines would replace their role in the industry

CHARLES BABBAGE: FATHER OF THE COMPUTER

- 1822: described the "difference engine" to aid in navigational calculations (never finished but proven correct in 1991)
- 1837: Analytical Engine, a general purpose computer (also never finished)
- also programmed with punched cards





ADA LOVELACE: THE FIRST PROGRAMER

- 1842-1843: she translated an article about the Analytical Engine, and supplemented it with an elaborate set of notes
- the notes contain an algorithm for computing Bernoulli numbers, considered to be the first computer program

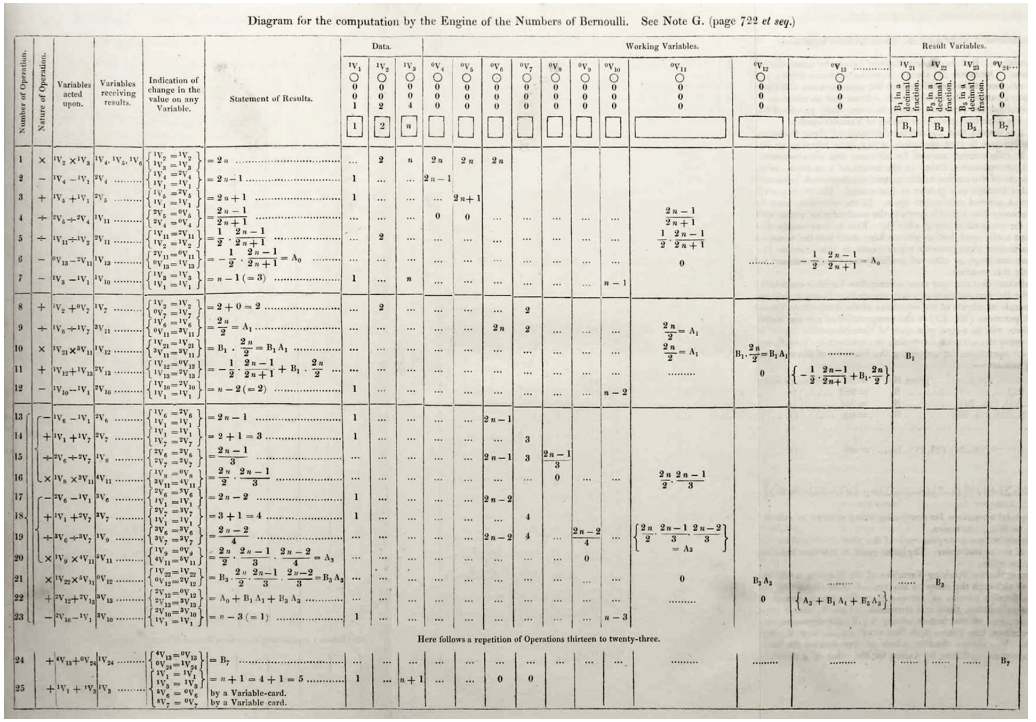
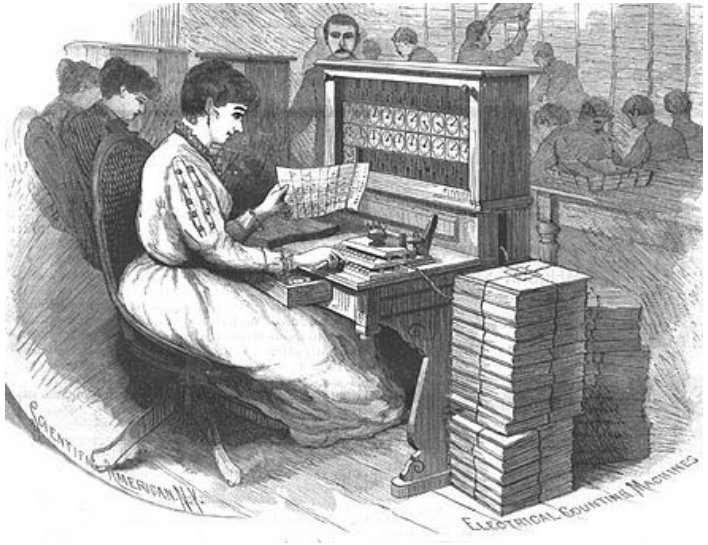


Diagram for the computation by the Engine of the Numbers of Bernoulli. See Note G. (page 722 *et seq.*)

Number of Operation.	Nature of Operation.	Variables acted upon.	Variables receiving results.	Indication of change in the value on any Variable.	Statement of Results.	Data. $^{1}V_1$	Data. $^{1}V_2$	Data. $^{1}V_3$	Working Variables. $^{0}V_4$	$^{0}V_5$	$^{0}V_6$	$^{0}V_7$	$^{0}V_8$	$^{0}V_9$	$^{0}V_{10}$	$^{0}V_{11}$	$^{0}V_{12}$	$^{0}V_{13}$ ...	Result Variables. $^{1}V_{21}$	$^{1}V_{22}$	$^{1}V_{23}$	$^{0}V_{24}$...
						1	2	n											B_1	B_2	B_3	B_7
1	×	$^{1}V_2 \times {}^{1}V_3$	$^{1}V_4, {}^{1}V_5, {}^{1}V_6$	$\begin{cases} {}^{1}V_2 = {}^{1}V_2 \\ {}^{1}V_3 = {}^{1}V_3 \end{cases}$	$= 2n$	...	2	n	$2n$	$2n$	$2n$											
2	−	$^{1}V_4 - {}^{1}V_1$	$^{2}V_4$	$\begin{cases} {}^{1}V_4 = {}^{2}V_4 \\ {}^{1}V_1 = {}^{1}V_1 \end{cases}$	$= 2n-1$	1	...	...	$2n-1$													
3	+	$^{1}V_5 + {}^{1}V_1$	$^{2}V_5$	$\begin{cases} {}^{1}V_5 = {}^{2}V_5 \\ {}^{1}V_1 = {}^{1}V_1 \end{cases}$	$= 2n+1$	1	...	...	...	$2n+1$												
4	÷	$^{2}V_5 \div {}^{2}V_4$	$^{0}V_{11}$	$\begin{cases} {}^{2}V_5 = {}^{0}V_5 \\ {}^{2}V_4 = {}^{0}V_4 \end{cases}$	$= \dfrac{2n-1}{2n+1}$	...	...	...	0	0	...	...	...	...	...	$\dfrac{2n-1}{2n+1}$						
5	÷	$^{2}V_{11} \div {}^{1}V_2$	$^{2}V_{11}$	$\begin{cases} {}^{2}V_{11} = {}^{2}V_{11} \\ {}^{1}V_2 = {}^{1}V_2 \end{cases}$	$= \dfrac{1}{2} \cdot \dfrac{2n-1}{2n+1}$	...	2	...	...	...	...	...	...	...	...	$\dfrac{1}{2} \cdot \dfrac{2n-1}{2n+1}$						
6	−	$^{2}V_{13} - {}^{2}V_{11}$	$^{1}V_{12}$	$\begin{cases} {}^{2}V_{13} = {}^{0}V_{13} \\ {}^{0}V_{13} = {}^{1}V_{13} \end{cases}$	$= -\dfrac{1}{2} \cdot \dfrac{2n-1}{2n+1} = A_0$	...	...	...	...	...	...	...	...	...	...	0	.....	$-\dfrac{1}{2} \cdot \dfrac{2n-1}{2n+1} = A_0$				
7	−	$^{1}V_3 - {}^{1}V_1$	$^{1}V_{10}$	$\begin{cases} {}^{1}V_3 = {}^{1}V_3 \\ {}^{1}V_1 = {}^{1}V_1 \end{cases}$	$= n-1\ (=3)$	1	...	n	...	...	...	...	...	...	$n-1$							
8	+	$^{1}V_2 + {}^{0}V_7$	$^{1}V_7$	$\begin{cases} {}^{1}V_2 = {}^{1}V_2 \\ {}^{0}V_7 = {}^{1}V_7 \end{cases}$	$= 2 + 0 = 2$	...	2	...	...	...	...	2										
9	÷	$^{1}V_6 \div {}^{1}V_2$	$^{2}V_{11}$	$\begin{cases} {}^{1}V_6 = {}^{1}V_6 \\ {}^{0}V_{11} = {}^{2}V_{11} \end{cases}$	$= \dfrac{2n}{2} = A_1$	...	...	...	...	...	$2n$	2	...	...	...	$\dfrac{2n}{2} = A_1$						
10	×	$^{1}V_{21} \times {}^{2}V_{11}$	$^{1}V_{12}$	$\begin{cases} {}^{1}V_{21} = {}^{1}V_{21} \\ {}^{2}V_{11} = {}^{2}V_{11} \end{cases}$	$= B_1 \cdot \dfrac{2n}{2} = B_1 A_1$	...	...	...	...	...	...	...	...	...	...	$\dfrac{2n}{2} = A_1$	.....					
11	+	$^{1}V_{12} + {}^{1}V_{12}$	$^{2}V_{13}$	$\begin{cases} {}^{1}V_{12} = {}^{0}V_{12} \\ {}^{1}V_{13} = {}^{2}V_{13} \end{cases}$	$= -\dfrac{1}{2} \cdot \dfrac{2n-1}{2n+1} + B_1 \cdot \dfrac{2n}{2}$	...	...	...	...	...	...	...	...	...	...	.....	$B_1 \cdot \dfrac{2n}{2} = B_1 A_1$	.....	B_1			
12	−	$^{1}V_{10} - {}^{1}V_1$	$^{2}V_{10}$	$\begin{cases} {}^{1}V_{10} = {}^{2}V_{10} \\ {}^{1}V_1 = {}^{1}V_1 \end{cases}$	$= n-2\ (=2)$	1	...	...	...	...	...	...	...	...	$n-2$		0	$\left\{-\dfrac{1}{2} \cdot \dfrac{2n-1}{2n+1} + B_1 \cdot \dfrac{2n}{2}\right\}$				
13	−	$^{1}V_6 - {}^{1}V_1$	$^{2}V_6$	$\begin{cases} {}^{1}V_6 = {}^{2}V_6 \\ {}^{1}V_1 = {}^{1}V_1 \end{cases}$	$= 2n-1$	1	...	...	...	...	...	$2n-1$										
14	+	$^{1}V_1 + {}^{1}V_2$	$^{2}V_7$	$\begin{cases} {}^{1}V_1 = {}^{1}V_1 \\ {}^{1}V_2 = {}^{2}V_7 \end{cases}$	$= 2 + 1 = 3$	1	...	...	...	...	...	...	3									
15	÷	$^{2}V_6 \div {}^{2}V_7$	$^{1}V_8$	$\begin{cases} {}^{2}V_6 = {}^{2}V_6 \\ {}^{2}V_7 = {}^{2}V_7 \end{cases}$	$= \dfrac{2n-1}{3}$	...	...	...	...	...	$2n-1$	3	$\dfrac{2n-1}{3}$									
16	×	$^{1}V_8 \times {}^{0}V_{11}$	$^{0}V_8$	$\begin{cases} {}^{1}V_8 = {}^{0}V_8 \\ {}^{2}V_{11} = {}^{2}V_{11} \end{cases}$	$= \dfrac{2n}{2} \cdot \dfrac{2n-1}{3}$	...	...	...	...	...	...	...	0	...	...	$\dfrac{2n}{2} \cdot \dfrac{2n-1}{3}$						
17	−	$^{2}V_6 - {}^{1}V_1$	$^{2}V_6$	$\begin{cases} {}^{2}V_6 = {}^{2}V_6 \\ {}^{1}V_1 = {}^{1}V_1 \end{cases}$	$= 2n-2$	1	...	...	...	...	...	$2n-2$										
18	+	$^{1}V_1 + {}^{2}V_7$	$^{2}V_7$	$\begin{cases} {}^{1}V_1 = {}^{1}V_1 \\ {}^{2}V_7 = {}^{2}V_7 \end{cases}$	$= 3 + 1 = 4$	1	...	...	...	...	...	...	4									
19	÷	$^{2}V_6 \div {}^{2}V_7$	$^{1}V_9$	$\begin{cases} {}^{2}V_6 = {}^{2}V_6 \\ {}^{2}V_7 = {}^{2}V_7 \end{cases}$	$= \dfrac{2n-2}{4}$	...	...	...	...	...	...	$2n-2$	4	...	$\dfrac{2n-2}{4}$	...	$\left\{\dfrac{2n}{2} \cdot \dfrac{2n-1}{3} \cdot \dfrac{2n-2}{3}\right\} = A_2$					
20	×	$^{1}V_9 \times {}^{2}V_{11}$	$^{2}V_{11}$	$\begin{cases} {}^{1}V_9 = {}^{0}V_9 \\ {}^{4}V_{11} = {}^{0}V_{11} \end{cases}$	$= \dfrac{2n}{2} \cdot \dfrac{2n-1}{3} \cdot \dfrac{2n-2}{4} = A_3$	...	...	...	...	...	...	...	...	...	0							
21	×	$^{1}V_{22} \times {}^{2}V_{11}$	$^{0}V_{12}$	$\begin{cases} {}^{1}V_{22} = {}^{1}V_{22} \\ {}^{0}V_{12} = {}^{2}V_{12} \end{cases}$	$= B_3 \cdot \dfrac{2n}{2} \cdot \dfrac{2n-1}{3} \cdot \dfrac{2n-2}{4} = B_3 A_3$	...	...	...	...	...	...	...	...	...	...	0	$B_3 A_3$	.....	.....	B_3		
22	+	$^{2}V_{12} + {}^{2}V_{13}$	$^{2}V_{13}$	$\begin{cases} {}^{2}V_{12} = {}^{0}V_{12} \\ {}^{2}V_{13} = {}^{2}V_{13} \end{cases}$	$= A_0 + B_1 A_1 + B_2 A_2$	...	...	...	...	...	...	...	...	...	...	.....	0	$\{A_2 + B_1 A_1 + B_2 A_2\}$				
23	−	$^{2}V_{10} - {}^{1}V_1$	$^{2}V_{10}$	$\begin{cases} {}^{2}V_{10} = {}^{2}V_{10} \\ {}^{1}V_1 = {}^{1}V_1 \end{cases}$	$= n-3\ (=1)$	1	...	...	...	...	...	...	...	...	$n-3$							
					Here follows a repetition of Operations thirteen to twenty-three.																	
24	+	$^{2}V_{13} + {}^{0}V_{24}$	$^{1}V_{24}$	$\begin{cases} {}^{2}V_{13} = {}^{0}V_{13} \\ {}^{0}V_{24} = {}^{1}V_{24} \end{cases}$	$= B_7$	...	...	...	...	...	...	...	...	...	...	.....	.....	.....	.....	.....	.....	B_7
25	+	$^{1}V_1 + {}^{1}V_3$	$^{1}V_3$	$\begin{cases} {}^{1}V_1 = {}^{1}V_1 \\ {}^{1}V_3 = {}^{1}V_3 \\ {}^{0}V_6 = {}^{0}V_6 \\ {}^{0}V_7 = {}^{0}V_7 \end{cases}$ by a Variable-card. by a Variable card.	$= n+1 = 4+1 = 5$	1	...	$n+1$	...	...	0	0											

HERMAN HOLLERITH: FATHER OF DATA PROCESSING

- invented data storage on punched cards
- used for the 1890 Census, which was processed in 6 years (1880 Census took 8 years to process!)
- his company eventually combined with four others to form IBM



THE BIRTH OF PROGRAMMING

- late 1950s: large computers became available to universities and research institutions
  - primarily in engineering and sciences
  - no longer available only to insiders
- with this movement into the public domain, a new profession was born–programmer
- but no interactivity at this point
  - programmers brought their programs (stacks of punched cards) to the counter, dropped them off, and came back for the results hours (or days!) later

THE EARLY DAYS OF PROGRAMMING

- sophisticated task, full of tips and tricks
- programming languages began to be developed
  - FORTAN (1957)
  - ALGOL (1958)
  - COBOL (1962)
- as compute capacity grew, so did demands on programmer
  - this programming stuff was determined to be difficult
- Computer Science was born
- waterfall-like software development model first described by Herbert Bennington in 1956
- interactivity, in the form of the first time-sharing system appeared in 1963

TIME-SHARING SYSTEMS

- many users can more efficiently make use of a computer than one person could
- dramatically lowered cost
- but the transition from batch processing (punched cards) to time-sharing was more difficult than anyone anticipated
- in 1968 a conference dedicated to multiprocessing and concurrent programming was sponsored by NATO, out of which came the terms
  - *software engineering*
  - *software crisis*

PROGRAMMING AS A DISCIPLINE

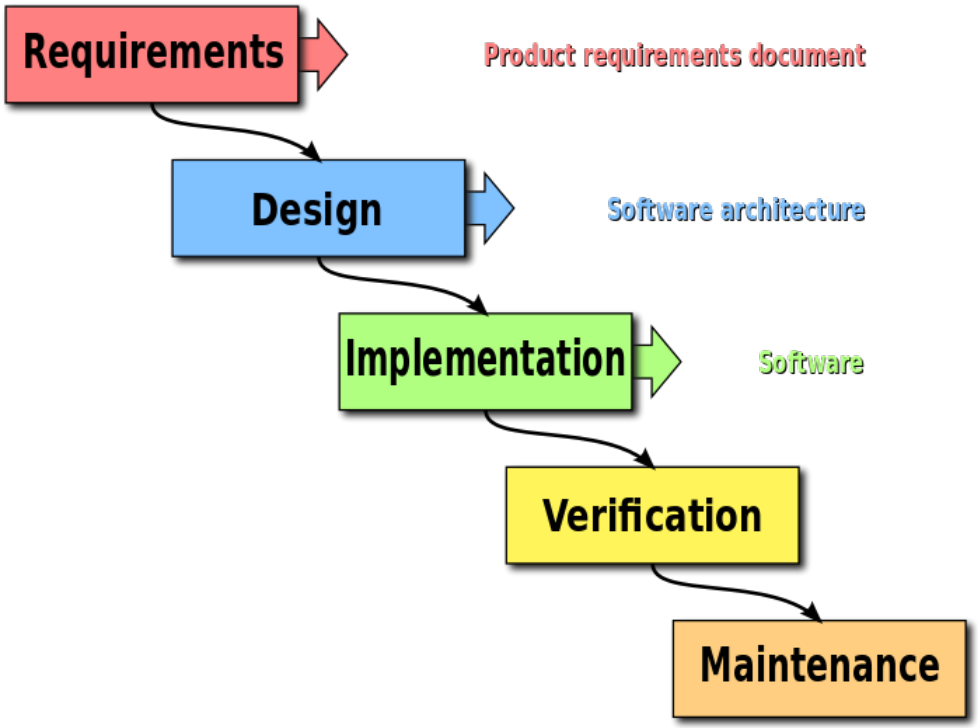
- several academics such as Edsger Dijkstra and C.A.R. Hoare wrote seminal papers
- programming sought to become a structured and disciplined methodology
- industry was unable to change rapidly
- Dept. of Defense realized problems were growing and a solution was needed
  - ...which led to the development of the Ada programming language (1980)

C AND UNIX: A GIANT LEAP BACKWARD (1969–1973)

- Unix was a welcome change from the operating systems that previously existed on mainframe computers, but...
- ... along with it came C, the language in which it was written
- from a software engineering standpoint, C represented a giant leap backward
  - too much freedom
  - rules could be broken
- at the same time, software engineering stagnated
  - pseudo-tools such as software metrics proved unhelpful
    - ...software engineers no longer judged by the number of lines of code they wrote

WATERFALL MODEL

- Winston Royce (1970)
  - presented it as a flawed, non-working model ("I believe in this concept, but the implementation described above is risky and invites failure")
  - involve the customer!
- Bell & Thayer (1976)
  - earliest use of the term
- DOD-STD-2167A (1985)



THE 1980s

- by the mid-1980s, languages such as Ada and C++ appeared and attempted to remedy the problems with C
- but they became large and difficult to describe
  - compilers became bulky and complex
  - according to Dijkstra, "They belonged to the problem set rather than the solution set"
- object-oriented programming gained ground after first appearing in the 1970s
  - Smalltalk (1980)
  - C++ (1985)

THE 1990s

- Open Source
  - Pros
    - the welcome alternative to industrial control and huge profits (as well as helpless dependence)
  - Cons
    - what of code quality?
    - counter to the notion of high-quality engineering and professionalism

HARDWARE EFFECT ON SOFTWARE QUALITY

- increased hardware capabilities were beneficial for numerous applications (e.g., banking, transportation, engineering, science, etc.)
- progress in software engineering was eradicated by the higher complexity of the tasks
- Reiser's Law: "Software is getting slower faster than hardware is getting faster"
- Demands rise, work often done under pressure
- Waste of cheap resources (CPU, RAM) results in inefficient code and a glut of data

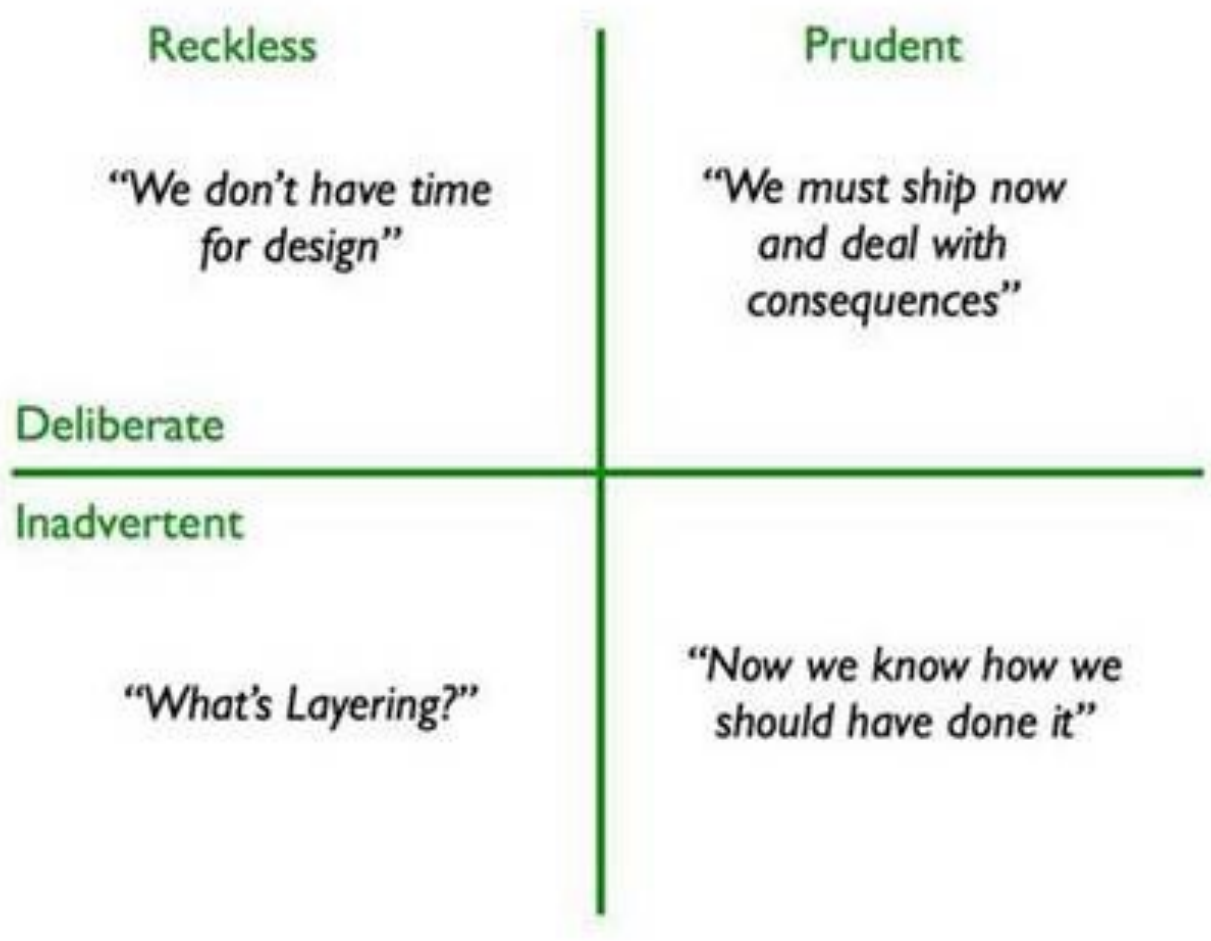
HARDWARE EFFECT ON SOFTWARE QUALITY (CONT'D)

- inefficient code can be hidden by fast CPUs
- poor data design can be hidden by more (cheap) storage
- side effect: quality went out the window
- good deign is time consuming and costly
- ...but poor design is more expensive in the long run

TECHNICAL DEBT

- "...the difference between what was promised and what was delivered"
- any time we cut corners and take the easy solution as opposed to the correct solution
- "Shipping first-time code is like going into debt. A little debt speeds development so long as it is paid back promptly with a rewrite...The danger occurs when the debt is not repaid. Every minute spent on not-quite-right code counts as interest on that debt." –Ward Cunningham, 1992

CLASSIFYING TECHNICAL DEBT (MARTIN FOWLER)



CLASSIFYING TECHNICAL DEBT (CONT'D)

- prudent and deliberate (compelling ROI)
  - responding to a threat or exploiting an opportunity (e.g., entry into new market, regulatory/legal requirement, Xmas)
- reckless and deliberate (poor management!)
- reckless and inadvertent (incompetent programmers)
- prudent and inadvertent (natural occurrence)
  - improve upon what has been done after gaining experience and relevant knowledge

SOLUTIONS?

- increase quality
- reduce complexity
- time to work
- Agile
- DevOps

AGILE AND DEVOPS ARE CHANGING THE (SOFTWARE) WORLD

AGILE ADVANTAGES

- Agile developers focus on sustainable development
  - good estimation
  - effective branching strategies
  - automated testing
  - continuous deployment
- Agile helps teams maintain a focus on rapid delivery of business value in the face of a constantly evolving functional and technical landscape
  - Agile teams rarely face the "death march" common in industry
- "the ability to respond to change without going off the rails"
- identifying and managing technical debt

DEVOPS ADVANTAGES

- Collaboration and communication
- Automation
- Shared responsibility (shared successes)
- Ending the culture of blame
- Improved customer experience and satisfaction–and that's what it's all about, right?

AGILE

AGILE MANIFESTO (2001)

- "We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:
  - Individuals and interactions over processes and tools
  - Working software over comprehensive documentation
  - Customer collaboration over contract negotiation
  - Responding to change over following a plan
- That is, while there is value in the items on the right, we value the items on the left more."

AGILE MANIFESTO (CONT'D)

1. Our highest priority is to **satisfy the customer** through **early and continuous delivery of valuable software**.
2. **Welcome changing requirements**, even late in development. Agile processes harness change for the customer's competitive advantage.
3. **Deliver working software frequently**, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. **Business people and developers must work together daily throughout the project.**
5. **Build projects around motivated individuals**. Give them the environment and support they need, and trust them to get the job done.
6. The **most efficient and effective method of conveying information** to and within a development team is **face-to-face conversation**.
7. **Working software is the primary measure of progress**.
8. **Agile processes promote sustainable development.** The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. **Continuous attention to technical excellence and good design enhances agility.**
10. **Simplicity**--the art of maximizing the amount of work not done--**is essential.**
11. **The best** architectures, requirements, and designs **emerge from self-organizing teams.**
12. **At regular intervals, the team reflects on how to become more effective**, then tunes and adjusts its behavior accordingly.

WHAT IS AGILE?

- From **www.agilemethodology.org**
  "Not a methodology! The Agile movement seeks alternatives to traditional project management. Agile approaches help teams respond to unpredictability through incremental, iterative work cadences and empirical feedback. Agilists propose alternatives to waterfall, or traditional sequential development."
- From **dictionary.com**
  "relating to or denoting a method of project management, used especially for software development, that is characterized by the division of tasks into short phases of work and frequent reassessment and adaptation of plans"
- it's a cultural and technical philosophy

AGILE SOFTWARE DEVELOPMENT METHODS/PRACTICES

- many Agile methods, but for this overview we'll focus on managing the flow of work via Scrum and Kanban
- similarly, there are many Agile practices, but we'll focus on some of the more common ones which are used in Scrum

SCRUM

- framework for managing software development
- work is broken into sprints, which are 1-4 week cycles
- key principles
  - customers will change their minds
  - there will be unpredictable challenges

SCRUM ROLES

- product owner ("voice of the customer")
  - focuses on business side of product development, working with stakeholders
  - NO development or technical problem solving!
- development team (sometimes "delivery team")
  - cross-functional
  - typically 3-9 people, not all developers
- Scrum master
  - ensure framework is followed
  - removing impediments
  - coaching

SCRUM PRACTICES: PLANNING/ESTIMATION

- usually done with story points and NOT HOURS
  - dates don't account for non-project related work that always creeps in
  - dates have emotional attachment
  - the velocity of each time will be different
- often performed via "planning poker" ("scrum poker")
- each person estimates with a face-down card, then all discuss once cards are revealed

SCRUM PRACTICES: DAILY STANDUP

- focused meeting in which all team members give information
- no more than 15 minutes!
- everyone stands up in order to ensure the meeting is short
- goal is communication
  - NOT status update for management
  - NOT problem solving

SCRUM PRACTICES: SPRINTS

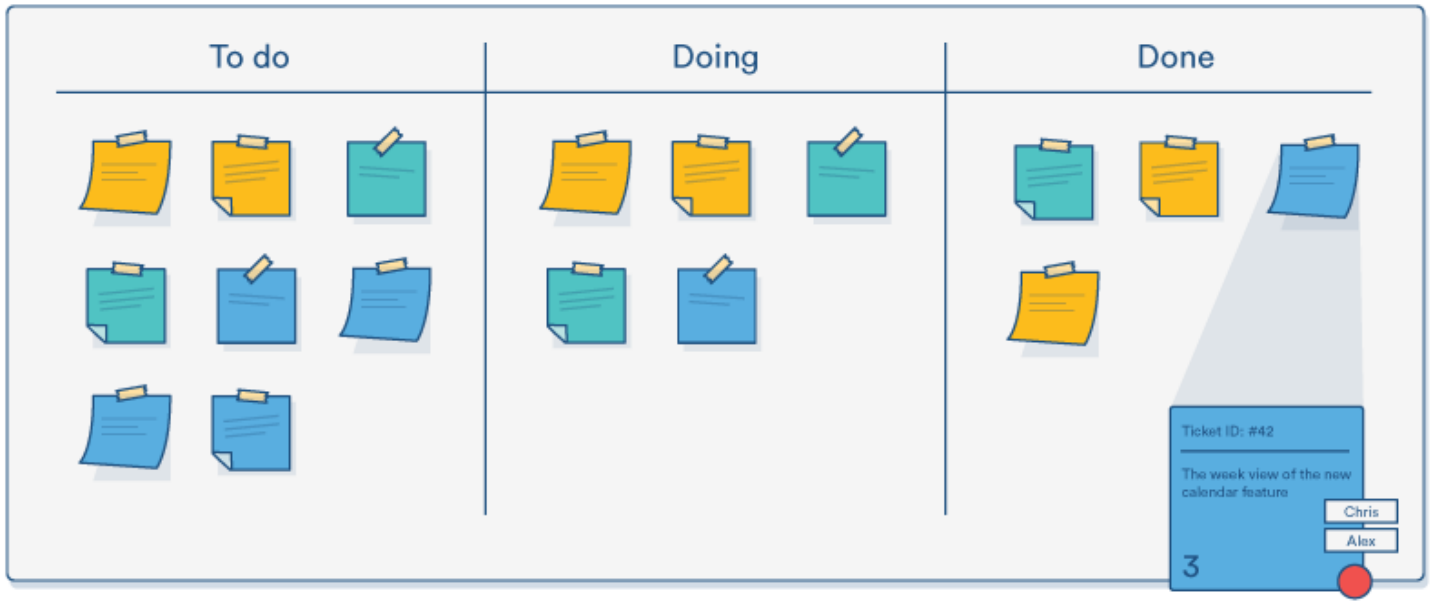
- 1-4 week interval of work
- basic unit of development for Scrum
- begins with a sprint planning meeting (~2 hours per week of sprint duration)
  - define backlog
  - identify work to be done
  - estimate forecast for sprint goal
- ends with a review (~1 hour per week of sprint duration)
  - review work that was (and was not) completed
  - demo completed work to stakeholders
  - team + stakeholders collaborate on what to work on next

SCRUM PRACTICES: SPRINT RETROSPECTIVE

- reflect on past sprint
- usually 1.5 hours for 2-week sprint
- facilitated by Scrum Master
- identify and agree upon actions for continuous improvement
- ask two main questions
  - what went well?
  - what could be improved in next sprint

KANBAN

- way to visualize the flow of work
- simplest Kanban board (they vary considerably)



- team members pull work as capacity permits
- focus is on customer and work which meets their needs, as opposed to activities of individuals

KANBAN (CONT'D)

- Kanban practices
  - visualization
  - limit work in progress
  - flow management
  - making policies explicit
  - feedback loops
  - collaborative evolution

DIFFICULTIES IN ADOPTING AGILE

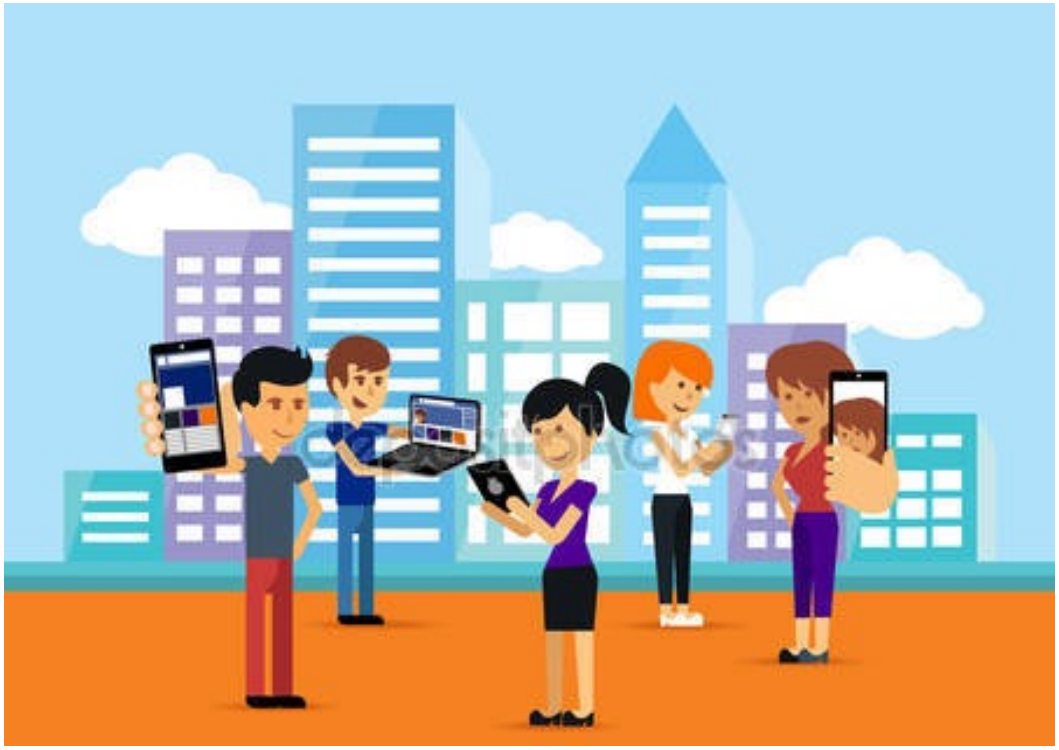
- Getting project leaders to buy into the notion that quality is more important than scope or schedule!

DEVOPS

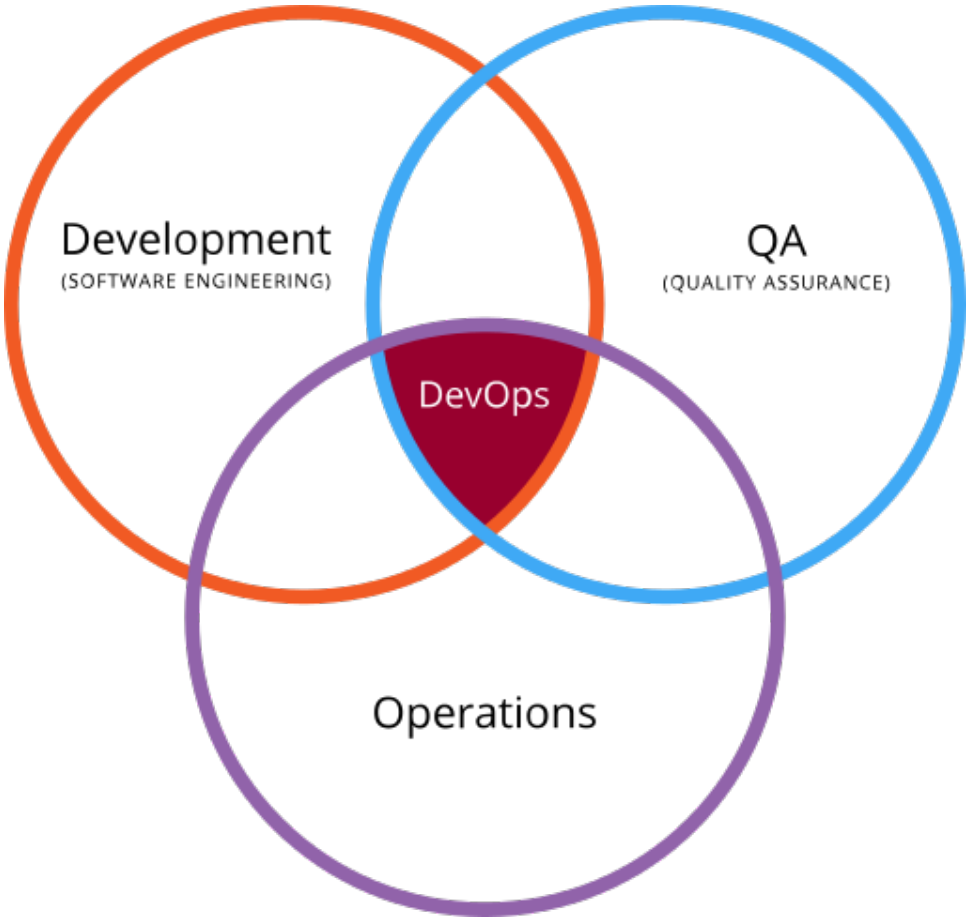
THE DARK AGES (PRE-2009)

- Before DevOps, IT departments did not really interact with developers and quality assurance (QA) engineers
- ...each group saw the other as an adversary, rather than an ally
- ...these teams were siloed off from one another
- interaction, if it occurred at all, was often "out of band", i.e., devs would walk over to their favorite IT person and ask for help

WHAT CHANGED?



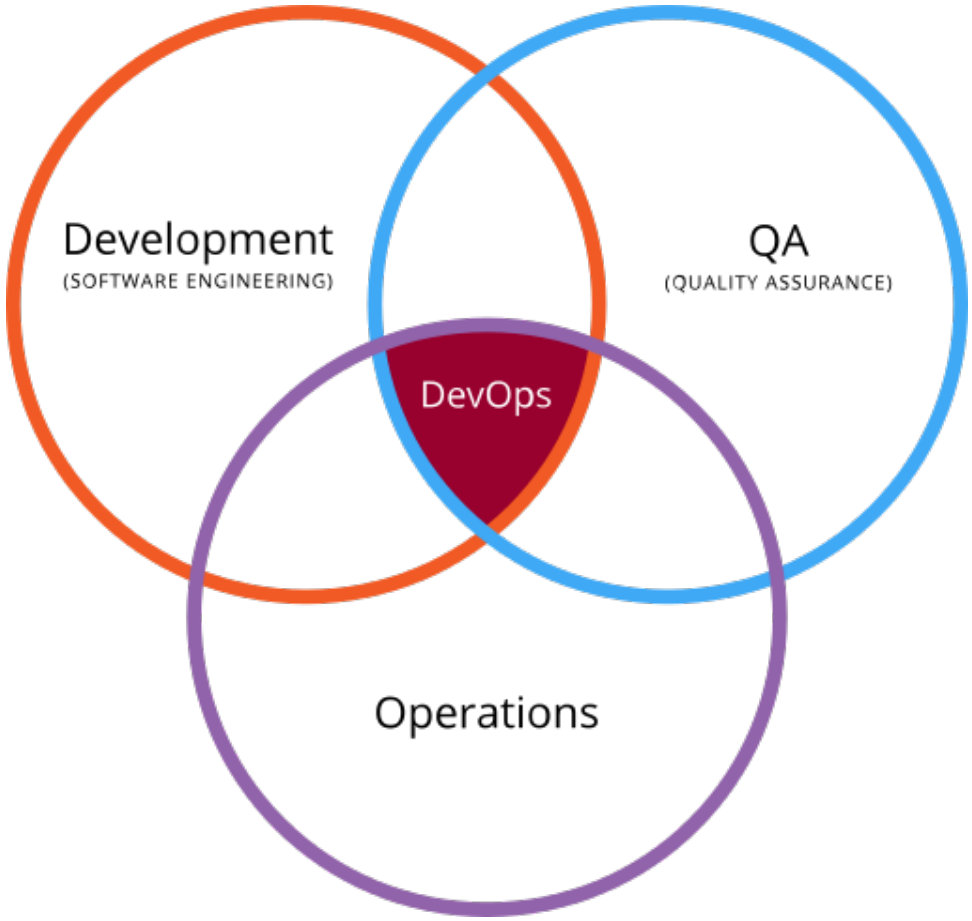
- We exist in an "always on" world, and mobile devices are being used more and more
- ...enterprise IT departments are now having to deal with ever-increasing numbers of customers and transactions without a proportional increase in IT headcount



WHAT IS DEVOPS?

- DevOps = development + operations
- Wikipedia–it's a culture, movement, or practice emphasizing collaboration and communication of software developers, QA, and other IT (operations) professionals, while automating the process of software delivery and infrastructure changes
- ...

WHAT IS DEVOPS? (CONT'D)



- not a job title (although you'll often see "DevOps Engineer"), but rather, a methodology
- mindset based on automating as much as possible for maximum efficiency, to get the most amount of work completed with the least amount of input
- the convergence of Developer tools and methodologies with IT operations
- an umbrella concept that refers to anything which eases the interaction between development and operations

DEVOPS IS A CULTURE SHIFT

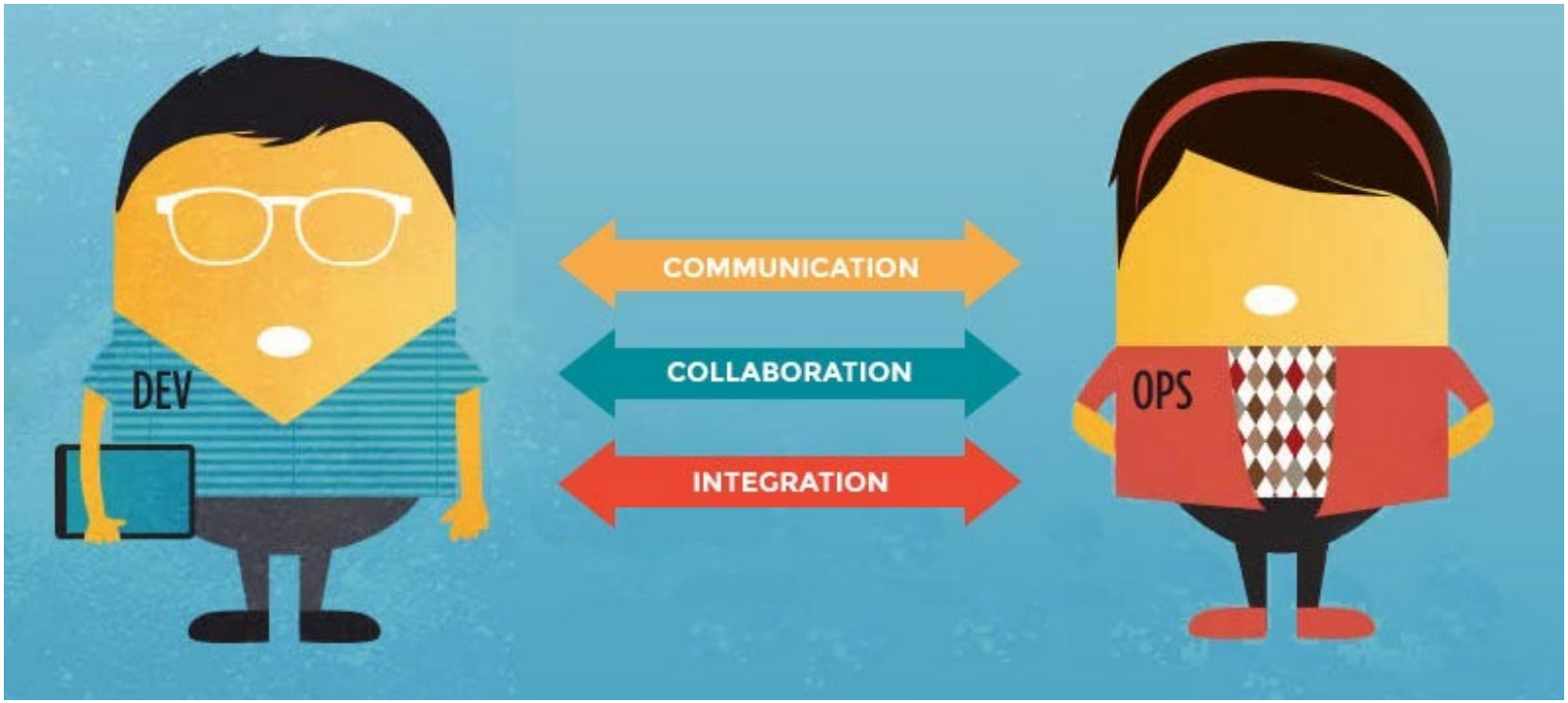
- DevOps is a massive organizational culture shift!
- ...difficult to achieve because of conflicting departmental goals
  - operations traditionally seek STABILITY
  - developers traditionally seek CHANGE
  - testers traditionally seek to REDUCE RISK

HOW DO WE ACCOMPLISH THIS CULTURE SHIFT?

- strong interdepartmental communication
- team-building
  - break down silos, borders, etc.
- co-location
  - not uncommon for Dev and Ops to be in different buildings, different floors of the same building, etc.

...CULTURE SHIFT (CONT'D)

- At its essence, DevOps is about
  - COMMUNICATION
  - COLLABORATION
- ...which didn't happen in the old world



...CULTURE SHIFT (CONT'D)

- DevOps grew out of new operations tools (e.g., virtualization, cloud computing and automated configuration management) and established agile engineering practices
  - But, tools are not enough!
- If we don’t change the organizational culture, DevOps will not be effective

WHAT ISN'T DEVOPS?

- it's not a product–you can't buy it
- it's not a title or a department
- it's not compliance, there is no DevOps certification
- it's not just a mixture of Dev and Ops

WHERE DID DEVOPS COME FROM?

- both development and operations had inefficiencies
- operations couldn't leverage the more efficient developer tools that existed
- developers couldn't control infrastructure on their own, slowing their progress
- What was needed?
  - operations needed to function more efficiently and in a more repeatable and collaborative fashion
  - developers needed to be more independent–they needed to manage their own infrastructure

DEVOPS PHILOSOPHY

- revolution
- evolution
- automation
- collaboration
- unifying
- empowerment
- generalization
- break down silos
- speed + quality
- agility
- people, process, and tools
- eradicate the culture of blame

DEVOPS PHILOSOPHY (CONT'D)

- DevOps is at its essence a human problem
  - more importantly, DevOps is a management problem
- must be all-in...behavior patterns cannot be changed without buy-in from leadership
- one team cannot "become" DevOps while the rest of the company stays the same
- “You can’t directly change culture. But you can change behavior, and behavior becomes culture” –Lloyd Taylor

DEVOPS PHILOSOPHY (CONT'D)

- an increase in collaboration
  - historically, development and operations were siloed, i.e., separate “structures” which did not interact with one another
- shared responsibility
  - developers share the pain of operations and can identify ways to simplify deployment and maintenance (e.g., deployment automation, improved logging)
  - operations shares responsibility for business goals and work more closely with developers to better understand their needs

THE CALMS MODEL

- we can sum up the DevOps philosophy via the CALMS model...

C = Culture

A = Automation

L = Lean (reduce time spent on non-value activities)

M = Metrics (measure everything/show improvement)

S = Sharing

THE WESTRUM MODEL OF ORGANIZATIONAL CULTURE

- **Pathological (power-oriented)**
  - low cooperation across groups
  - a culture of blame
  - information often withheld for personal gain
- **Bureaucratic (rule-oriented)**
  - preoccupied with rules and positions,
  - responsibilities compartmentalized by department
  - little concern for the overall mission of the organization.
- **Generative (performance-oriented)**
  - good information flow
  - high cooperation and trust
  - bridging between teams
  - conscious inquiry

...THE DEVOPS CULTURE MATRIX

Power Orientated	Rule/Bureaucratic Orientated	Performance/(DevOps) Orientated
Low Cooperation (silos)	Modest Cooperation (trade)	High Cooperation (same team)
Communicators Punished	Communicators Ignored	Communicators Rewarded
Avoid Responsibilities	Narrow Responsibilities	Shared Responsibilities & Risks
Failure = Scapegoats	Failure = Apply Justice	Failure = Discovery & Improvement
Innovation is Crushed	Innovation Ignored	Innovation is Continual
Hide	Only what is Required	Engaged
Hands Off	Own a Piece	Own it
Training for Some	Train per Skill Set/JD	Continual Training

"LEAN PRODUCT MANAGEMENT"

- these factors were statistically significant in predicting higher IT performance and lower deployment pain
  - extent to which teams decompose products and features into small batches that can be complete in under a week and released frequently ("limit work in progress")
  - whether teams have a good understanding of the flow of work from the business to the customers, and whether they have visibility into this flow ("make work visible")
  - whether organizations actively and regularly seek customer feedback, and then incorporate it into product design

(source: 2016 State of DevOps)

DEVOPS GOALS

- improved deployment frequency, i.e., more releases!
- faster time to market
- lower failure rates
- quicker fixes
- faster recovery time if new release crashes
- processes become increasingly programmable and dynamic
- maximize predictability, efficiency, security, and maintainability of operational processes

DEVOPS GOALS (CONT'D)

- ease release management
  - standardizing development environments with developer control
- find bugs early!
  - frequent build/test cycle means finding bugs right after code commits rather than weeks or months later

DEVOPS ADVANTAGES

- improved customer experience and satisfaction
- break down the silos / self-organization
- collaboration
- agility
- ability to quickly respond to changing market/customer demands
- continuous feedback
- rapid innovation
- increased software quality
- reduced risk of change

DEVOPS ADVANTAGES (CONT'D)

- team empowerment
- streamlining processes
- productivity
- continuous improvement
- saving time (...and money)
- improving ROI of data
- competitive advantage (over your old school competitors)
- survival–if your competitors have embraced DevOps and you haven't, you will be left behind

DEVOPS ERASES TECHNICAL DEBT

- What is Technical Debt?
  - A concept in programming that reflects the extra development work that arises when code that is easy to implement in the short run is used instead of applying the best overall solution
- every time we fail to automate it creates technical debt
- "Technical debt is what we feel the next time we want to make a change" –Ward Cunningham ("The best way to get the right answer on the Internet is not to ask a question, it's to post the wrong answer.")

DEVOPS "ONE-PAGER"

- collaborate/remove silos
- automate as much as possible
- continuous integration
- measure everything/strive to do better
- kaizen = continuous improvement (on every front...people, process, and tools)
- value feedback
- test-driven development
- focus on value activities/avoid non-value activities
- autonomy
- share responsibility

SETTING UP A DEVOPS ENVIRONMENT

SETTING UP A DEVOPS ENVIRONMENT

- remove silos between development and operations
- co-locate development and operations when possible
- focus on daily collaboration
- avoid handovers and sign-offs, which lead to a culture of blame and are antithetical to collaboration
- avoid documentation of processes as a substitute for actual collaboration on those processes
- both developers and operations must be responsible for successes and failures

SETTING UP A DEVOPS ENVIRONMENT (CONT'D)

- avoid assigning a “DevOps” role or “DevOps Team– remember, DevOps is a culture, not a job title
- trust your teams–let them make decisions without convoluted processes and oversight
- releasing code must become the status quo
- team must value building quality into the development process (e.g., continuous delivery and test-driven development, both of which we will discuss)

INCREASE AUTONOMY

KAIZEN

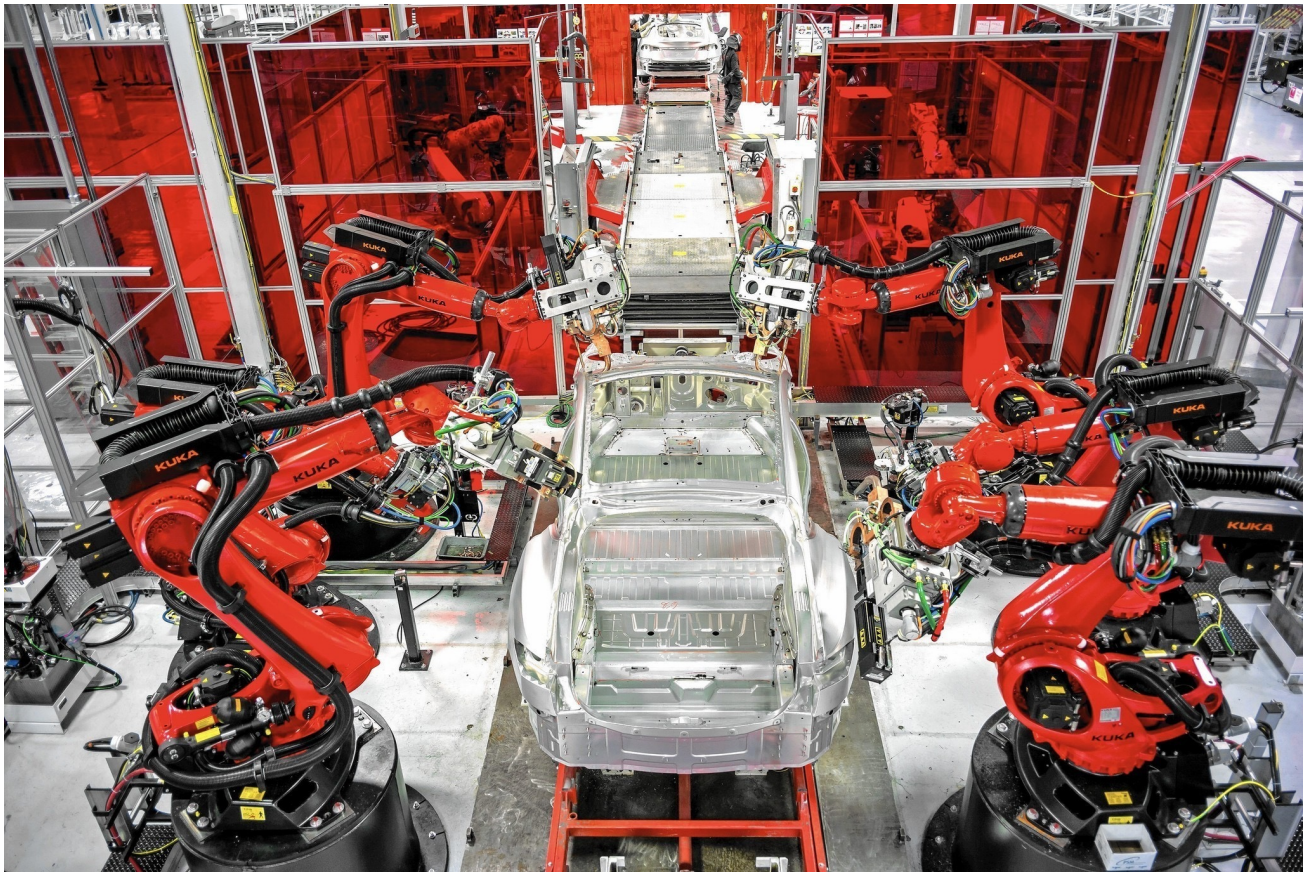
= JAPANESE BUSINESS PHILOSOPHY OF CONTINUOUS IMPROVEMENT OF WORKING PRACTICES, PERSONAL EFFICIENCY, ETC.

VALUE FEEDBACK

"They don't really encourage feedback around here, do they?"

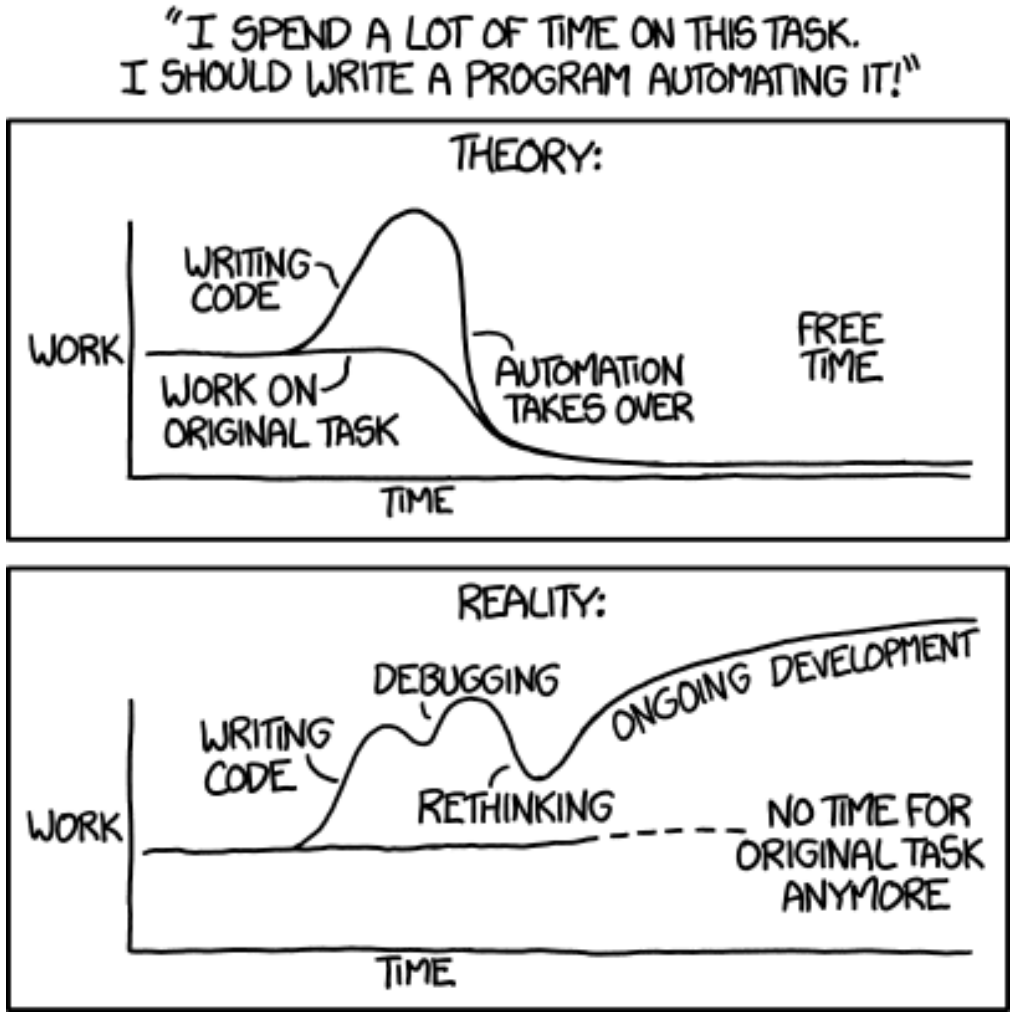
REDUCE HUMAN ERROR

EMBRACE AUTOMATION



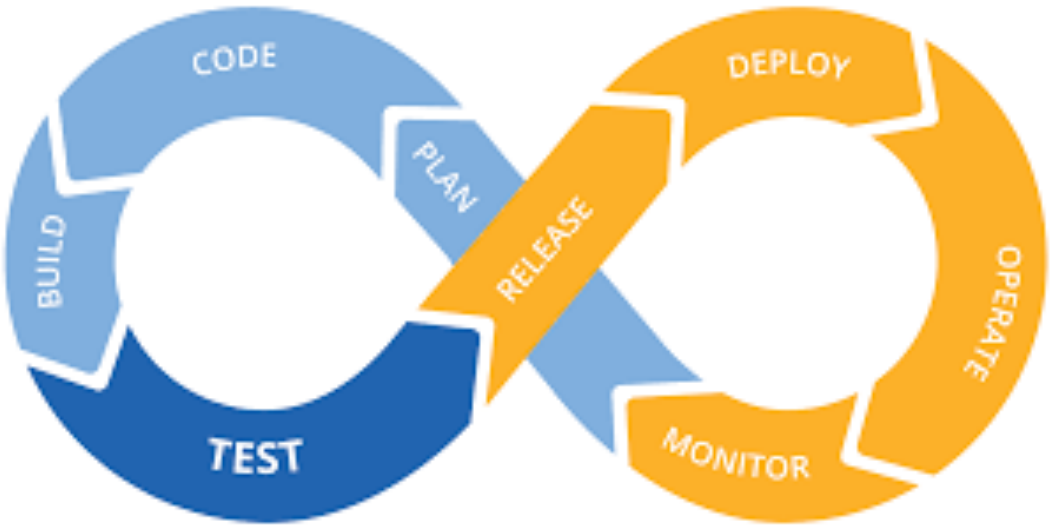
ALLOW PEOPLE TO FOCUS ON OTHER TASKS

AUTOMATE YOUR TASKS/TESTS
AND THEY WILL SERVE AS DOCUMENTATION



DEVOPS LIFECYCLE

THE DEVOPS LIFECYCLE



- rapid release cycle with a strong feedback loop
- continuous release/ continuous improvement
- test-driven development ensures software is robust
- code is built regularly and tested to ensure quality
- code is released and continuously monitored to identify bugs

DEVOPS PRACTICES: OVERVIEW

INFRASTRUCTURE AS CODE

CHEF

ANSIBLE

- managing and provisioning machines/infrastructure via software tools which read configuration files describing how you want the machines to be set up
- benefits
  - reduction in cost by freeing up IT staff
  - speed (e.g., can configure/provision 1000s of machines in parallel)
  - reduces human error

PROBLEM SOLVED: LACK OF SPEED

- in the old days (mid-2000s), IT purchased new servers for the apps the company needed to run
- setting up those servers was done by hand and consumed hours or even days
- even when scripts were used, they were not robust–they were brittle, did not scale, and contained a lot of org-specific information, increasing the on-boarding time for new staff

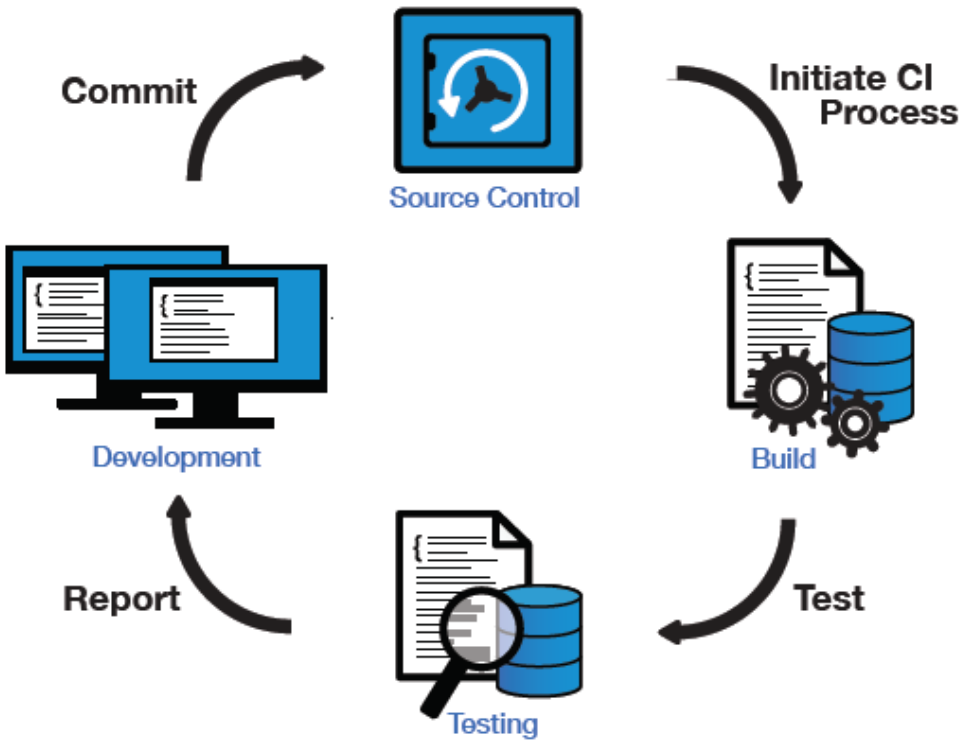
IT AUTOMATION

- often refers to automatic provisioning of new machines via a cloud provider
- but really can refer to automation of any IT process
- automation is a key DevOps principle, not just for Dev, but also for Ops

PROBLEM SOLVED: LACK OF SPEED

- new machines not only required time for configuration, but turnaround time to get them in the first place was long
- IT processes in general were typically not automated, resulting in slow turnaround and long handoffs

CONTINUOUS INTEGRATION

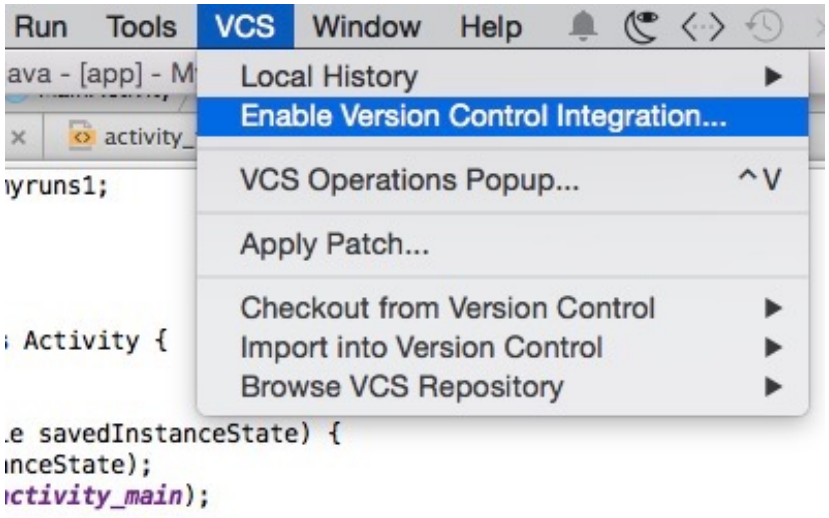


- any time code is committed, a build is triggered, the code is integrated (i.e., combined with other parts of the system), and tested
  - errors are found immediately
  - work is made visible
  - ensures communication among the team

PROBLEM SOLVED: INTEGRATION HELL

- poor code quality
- nail-biting integrations, everybody working weekends, etc.
- quicker/cheaper to debug since the longer you wait to fix a bug, the costlier it is to fix it
- human error in deployments
- late feedback from QA

VERSION CONTROL INTEGRATION



- integrates version control (i.e., recording all changes to your code so you can access a specific version later) into your IDE (interactive development environment)
- makes it easier to work since dev tools are integrated
- solves the problem in which developers don't commit as often as the should (sort of)
- more feedback = good

APPLICATION/INFRASTRUCTURE VERSION MANAGEMENT

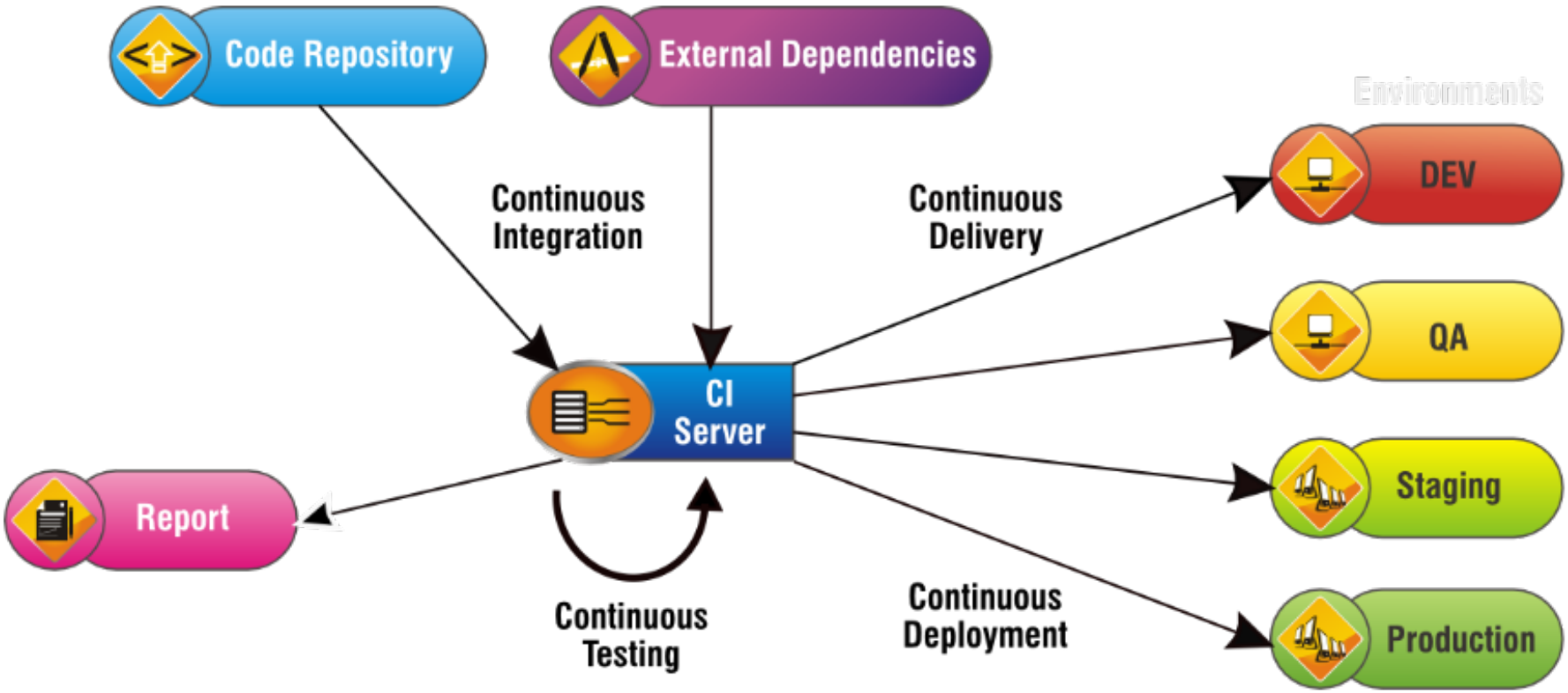
- we are DevOps, so everything (including the "obvious" stuff) should be kept in version control, not just the source code of our application
- deployment scripts
- provisioning/configuration scripts for setting up your infrastructure via configuration management tools such as Puppet and Chef

PROBLEM SOLVED: CHAOS

- ad hoc testing environments
- seat-of-the-pants deployment
- what's obvious now may not be obvious later
  - process could (and likely will) become more complicated
  - new employees
  - etc.

MONITORING/LOGGING

- measuring everything is a key principle of DevOps
  - (otherwise, how do we define improvement?)
- increased awareness is a good thing–what should we monitor?
  - development milestones (burndown rate, ratio of promised-to-delivered features, etc.)
  - vulnerabilities (NVD as well as those in our own code)
  - deployments
  - server health



THE DEVOPS TOOL STACK

PERIODIC TABLE OF DEVOPS TOOLS (V2)

Os	Open Source
Fr	Free
Fm	Freemium
Pd	Paid
En	Enterprise

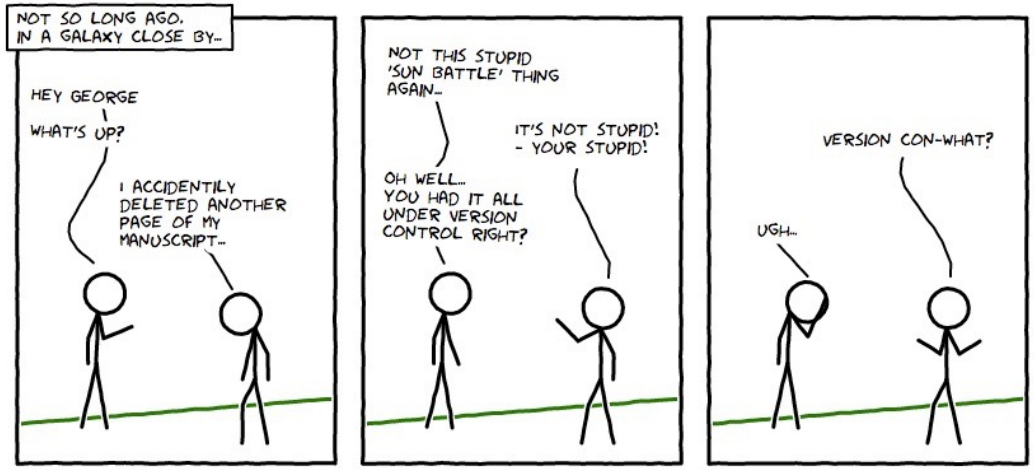
SCM	Database Mgmt	Build
CI	Repo Mgmt	Testing
Deployment	Config / Provisioning	Containerization
Cloud / IaaS / PaaS	Release Mgmt	Collaboration
BI / Monitoring	Logging	Security

1 Fm Gh Github																		2 Fm Aws AmazonWeb Services
3 Os Gt Git	4 En Dm DBmaestro													5 En Ch Chef	6 En Pu Puppet	7 Os An Ansible	8 Os Sl Salt	9 Os Dk Docker	10 Pd Az Azure
11 Fm Bb Bitbucket	12 Os Lb Liquibase													13 Os Ot Otto	14 En Bl BladeLogic	15 Os Va Vagrant	16 Fr Tf Terraform	17 Os Rk rkt	18 En Gc Google Cloud Platform
19 Os Gl GitLab	20 En Rg Redgate	21 Os Mv Maven	22 Os Gr Gradle	23 Os At ANT	24 Os Fn FitNesse	25 Fr Se Selenium	26 Os Ga Gatling	27 Fr Dh Docker Hub	28 Os Jn Jenkins	29 Pd Ba Bamboo	30 Os Tr Travis CI	31 Pd Gd Deployment Manager	32 Os Sf SmartFrog	33 Os Cn Consul	34 Os Bc Bcfg2	35 Os Mo Mesos	36 En Rs Rackspace
37 Os Sv Subversion	38 En Dt Datical	39 Os Gt Grunt	40 Os Gp Gulp	41 Os Br Broccoli	42 Fr Cu Cucumber	43 Os Cj Cucumber.js	44 Fr Qu Qunit	45 Os Npm npm	46 Fm Cs Codeship	47 Pd Vs Visual Studio	48 Fm Cr CircleCI	49 Fr Cp Capistrano	50 Fr Ju JuJu	51 Os Rd Rundeck	52 Os Cf CFEngine	53 Fr Ds Swarm	54 Os Op OpenStack
55 Os Hg Mercurial	56 En Dp Delphix	57 Fr Sb sbt	58 Os Mk Make	59 Os Ck CMake	60 Fr Jt JUnit	61 Fr Jm JMeter	62 Fr Tn TestING	63 Os Ay Artifactory	64 Fm Tc TeamCity	65 Fm Sh Shippable	66 Os Cc CruiseControl	67 En Ry RapidDeploy	68 Fm Cy CodeDeploy	69 En Oc Octopus Deploy	70 En No CA Nolio	71 Os Kb Kubernetes	72 Fm Hr Heroku
73 En Cw ISPW	74 En Id Idera	75 Os Msb MSBuild	76 Os Rk Rake	77 Fr Pk Packer	78 Os Mc Mocha	79 Fr Km Karma	80 Os Jm Jasmine	81 Os Nx Nexus	82 Os Co Continuum	83 Fm Ca Continua CI	84 Pd So Solano CI	85 En Xld XL Deploy	86 En Eb ElasticBox	87 Fm Dp Deploybot	88 En Ud UrbanCode Deploy	89 Os Nm Nomad	90 En Os OpenShift

XebiaLabs

Follow @xebialabs

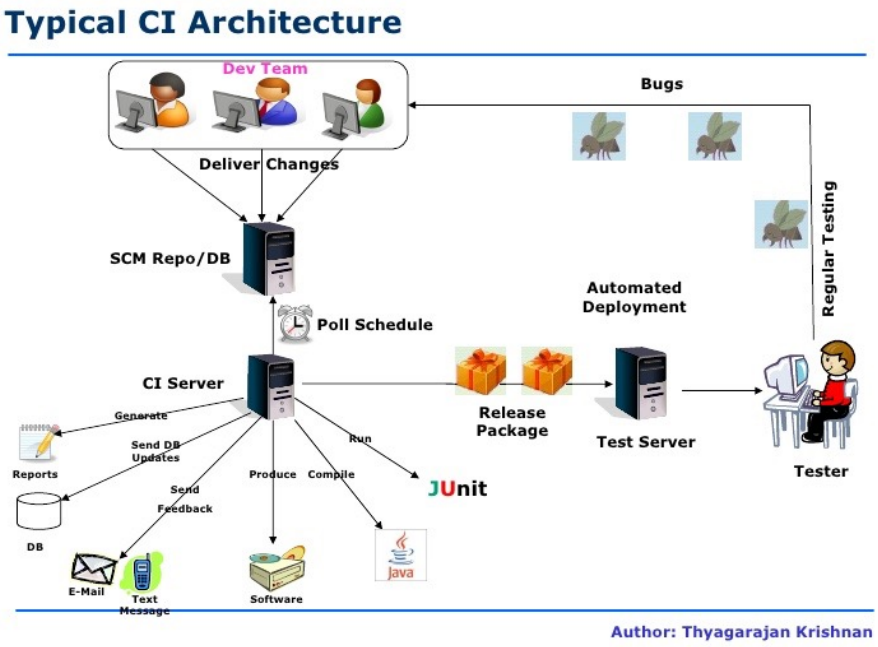
91 En Xlr XL Release	92 En Ur UrbanCode Release	93 En Bm BMC Release Process	94 En Hp HP Codar	95 En Au Automic	96 En Pl Plutora Release	97 En Sr Serena Release	98 Pd Tfs Team Foundation	99 Fm Tr Trello	100 Pd Jr Jira	101 Fm Rf HipChat	102 Fm Sl Slack	103 Fm Fd Flowdock	104 Pd Pv Pivotal Tracker	105 En Sn ServiceNow
106 Os Ki Kibana	107 Fm Nr New Relic	108 En Dt Dynatrace	109 Os Ni Nagios	110 Os Zb Zabbix	111 En Dd Datadog	112 Os El Elasticsearch	113 Fm Ad AppDynamics	114 En Sp Splunk	115 Fm Le Logentries	116 Fm Sl Sumo Logic	117 Os Ls Logstash	118 Os Sn Snort	119 Os Tr Tripwire	120 En Ff Fortify



VERSION CONTROL

- simply put version control is a system which keeps track of all the changes to your code so you can access a specific version later
- need to be able to keep track of releases, test builds, etc., as well as roll back issues
- e.g., Git, Subversion (svn), Mercurial, Perforce
- "Software eats the world" – Marc Andreessen

CI SERVER



- pulls code from version control, orchestrates builds, tests, etc., logs results
- typically set up to poll version control system on a regular basis to launch builds as soon as changes are committed
- solves the problems of trying to manage building and testing your code by hand
- e.g., Jenkins, Bamboo, TravisCI, TeamCity

OTHER TOOLS

- **build** tools–project or platform specific, these tools build our code (e.g, Maven, Gradle, gcc, VisualStudio)
- **testing** tools abound–"untested code is broken code"
- **logging** tools such as LogStash and Splunk allow us to monitor our infrastructure
- **cloud** platforms (PaaS) enable us to extend our infrastructure into the cloud or operate solely in the cloud with no local hardware

HOW TO AVOID FAILURE?

- first, you must define success
  - "The only company you need to be better than is the company you were yesterday" (with apologies to Bruce Lee)
  - remember that success is incremental
  - remember that monitoring/measuring is a key principle of DevOps–if you don't measure, how will you know if you're improving?

HOW TO AVOID FAILURE? (CONT'D)

- get management buy-in–DevOps is such an enormous culture shift that it will absolutely fail without management on board
- "fail fast, fail often"
  - turn failures into learning opportunities
  - we all fail along the path to success
  - keep moving forward, keep measuring, keep improving
- there is no easy way to succeed in anything

HOW IS LIFE DIFFERENT FOR SARA, A DEVELOPER?

- Sarah will be writing code, as before, AND...
- building tools
- designing infrastructure
- monitoring
- virtualizing
- automating
- optimizing/tuning
- managing configurations using Chef or other config management tools
- continuously integrating
- continuously deploying
- releasing!

HOW IS LIFE DIFFERENT FOR JOSH, A QA ENGINEER?

- Josh's job is no longer about finding bugs, as it was before DevOps
- Now, his directive is to release code when it's working, and roll back when it's not, i.e., QA is responsible for ensuring defective code does not make it onto public site
- Josh and his team own continuous improvement and quality tracking across the entire development cycle
- QA is primarily responsible for identifying problems not just in the product but also in the process
- Josh should recommend changes wherever possible, and look for any opportunity to improve repeatability and predictability

CONTINUOUS...

CONTINUOUS DELIVERY VS. DEPLOYMENT

- **Continuous Delivery** does NOT mean every change is deployed, only that it can be (i.e., proven to be deployable)
  - stakeholders decide when the release happens (not the engineers)
- **Continuous Deployment** means that every build that passes testing is put into production
  - should be the goal of most companies, unless you're constrained by regulatory or other requirements

CONTINUOUS INTEGRATION/DELIVERY/DEPLOYMENT