Dev(Sec)Ops

Thoughts from the Trenches.

Chris Madden
Distinguished Engineer - Software Security and System Architect
CISSP, ISSAP, CISA

The DevOps Factory

DevOps is a modern software development approach that strives to bring development and operations teams together along with other stakeholders to improve efficiency and outcomes by focusing on shared business goals. DevOps follows and expands on key principles of the Agile software development and Lean engineering movements and represents a fundamental shift in how large, distributed enterprise organizations develop and deliver software.

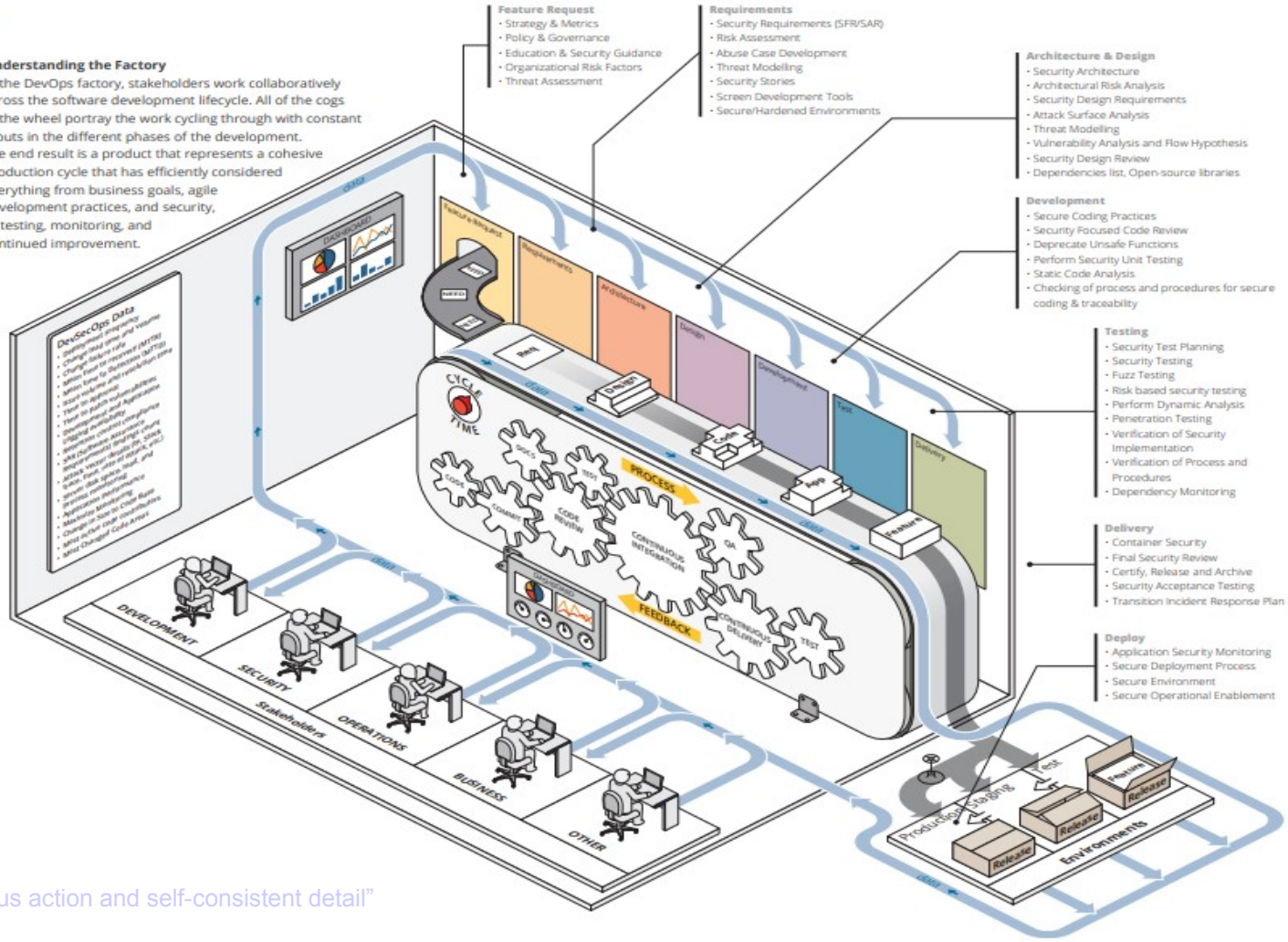
By cultivating cross-functional collective engagement in software development projects throughout the software development lifecycle (SDLC), DevOps affects the people, processes, and technology of an organization. DevOps also requires adopting and implementing cutting-edge practices based on the primary tenants of collaborative culture, automation, data-driven processes, infrastructure as code, and ubiquitous, real-time system monitoring.

The features and benefits of DevOps include

- **Consistently developing** software systems with higher quality and accuracy of project budgeting and estimation
- **Increased visibility** and stakeholder input into features for the next release as it is being developed
- **Engaging stakeholders** early and consistently throughout the SDLC, leading to fewer defects and incorrect requirements
- **Building trust** between software development and IT, enabling organic process improvement and risk mitigation
- **Maximizing business value** by enabling technical staff to adapt to changing requirements or environmental factors

Understanding the Factory

In the DevOps factory, stakeholders work collaboratively across the software development lifecycle. All of the cogs in the wheel portray the work cycling through with constant inputs in the different phases of the development. The end result is a product that represents a cohesive production cycle that has efficiently considered everything from business goals, agile development practices, and security, to testing, monitoring, and continued improvement.



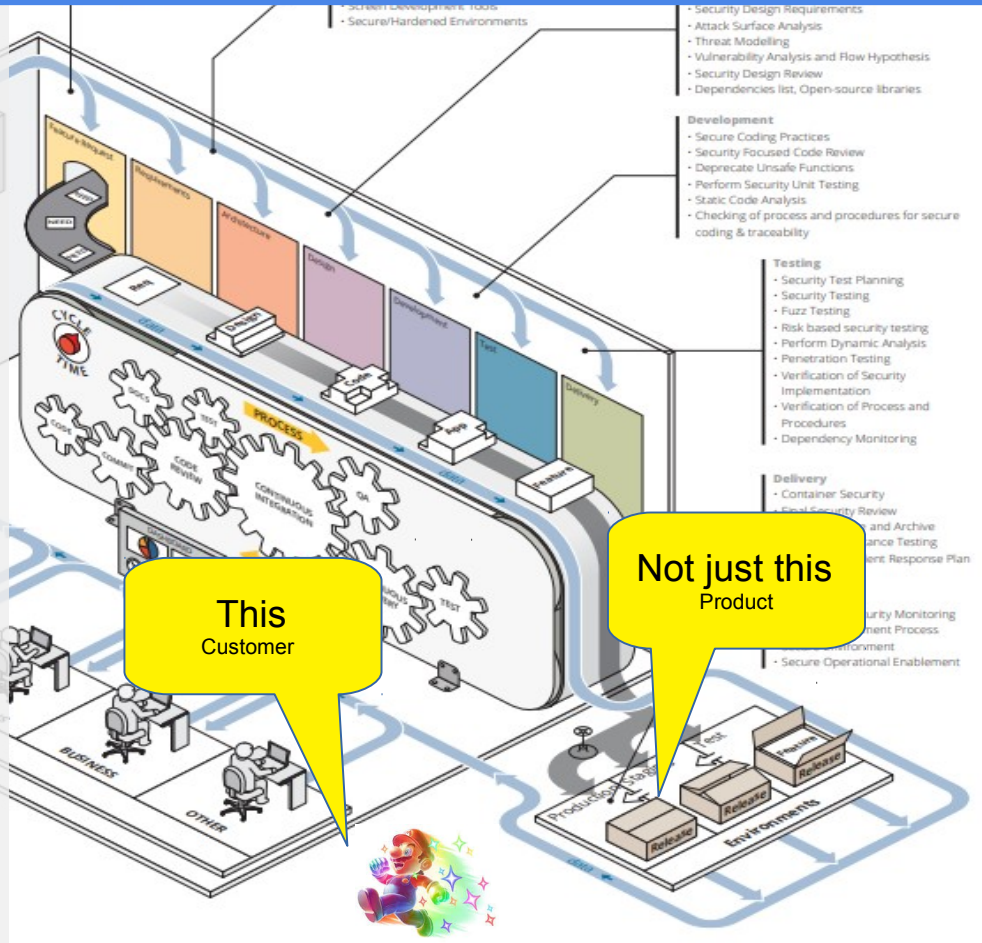
Continuity: “the maintenance of continuous action and self-consistent detail”

Purpose: Deliver Value to the Customer

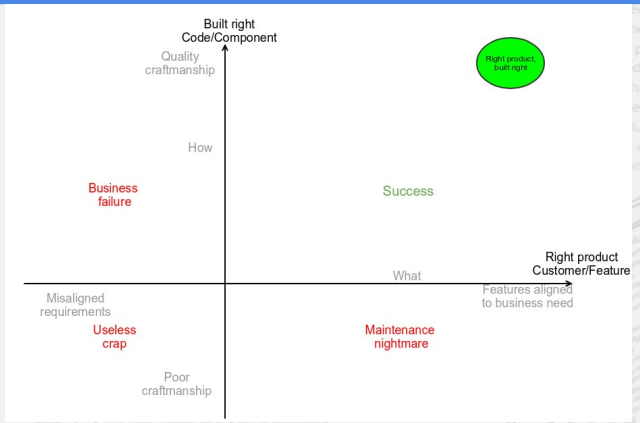
Deliver **Value** to the customer through a timely efficient solution

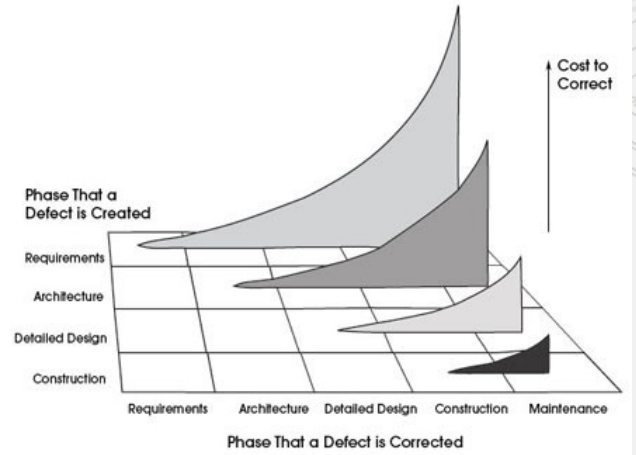
1. **Value**: deliver customer value - not hardware - not software - and not the things that have no value
2. **Timely**: on time but just in time e.g. not efficient to deliver stuff that is not needed or used for many months
3. **Efficient**: minimal functionality and effort required
4. **Solution**: not just the thing that is shipped - but all the things the customer needs to use the features to create value for them

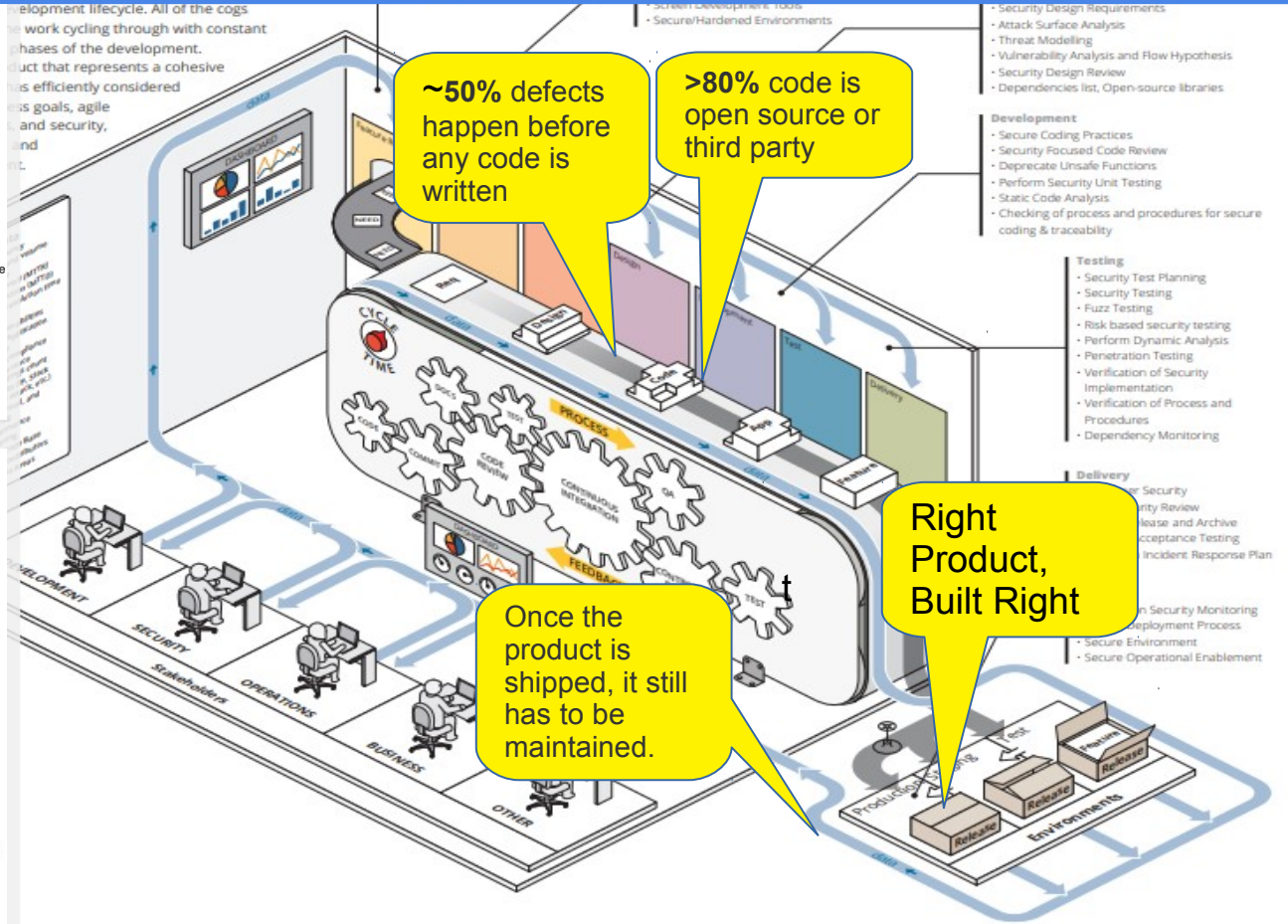
The customer cares about what they are trying to do - not your product.



Right Product, Built Right







People: The Most Important Component

1. **Management exists to enable / facilitate the teams to do their job - not to dictate / control how it is done.**
   a. **Happy People are Productive People are Happy People**
   – Any initiative will fail without support from the top E.g. if executive management don't embrace Agile/dev(sec)ops then it won't work.
2. **Agile/dev(sec)ops works because it Drives people (Mastery, Autonomy, Purpose)**
3. **The techie problems are easy. The people problems are hard.**
4. Continuous Communication is key ("most critical part of software development")
   a. Living documentation (social media way of working) is key - avoid derivate snapshots (e.g. PowerPoint, Excel)
   b. A developer, program manager, or executive should be able to see the same data in real time

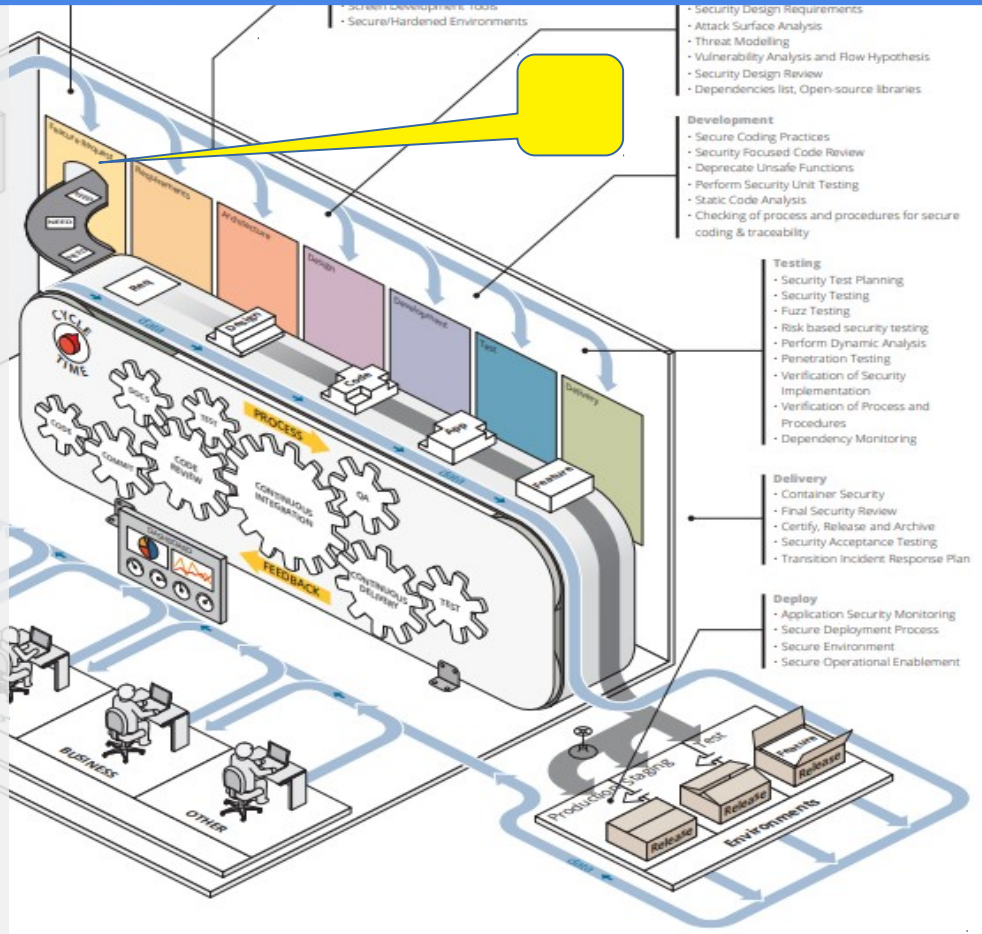
People's goals/rewards need to be aligned towards delivering customer value e.g. if the SQA group are rewarded for finding more defects, then it is not in their interest to help prevent, detect these defects at development time. Or if the IT department is rewarded for low risk or low vulnerabilities, then the systems may be secure but deliver low value to, or not be useable by, the customer.

Despite advances in Artificial Intelligence, and Machine Learning, systems, today, are developed and used by **people**.

Feature Request: The Most Important Step

The Feature Request Intake needs to be managed (The further upstream work is managed, the bigger the impact)

1. This is done via a well maintained Product Backlog. There should be a “healthy tension” between those asking for the work and those doing the work.
2. Feature Request can come from:
    1. Customer (e.g 70%)
    2. Engineering (e.g. 30%)
        1. Enablers - Architectural Runway
        2. Technical Debt
3. “Average company, more than a third of what they are building is junk stories”
   Jeff Sutherland, CEO of Scrum Inc.



Feature Request - Mind The Gap!

1. **Tell me what you want**
   a. **Not what you think you can have**
   b. **Don't try to sound smart**
   c. **Don't tell me how to do it**
2. **Multiple independent forms of Requirements, Specification, and Test - and Prototypes**

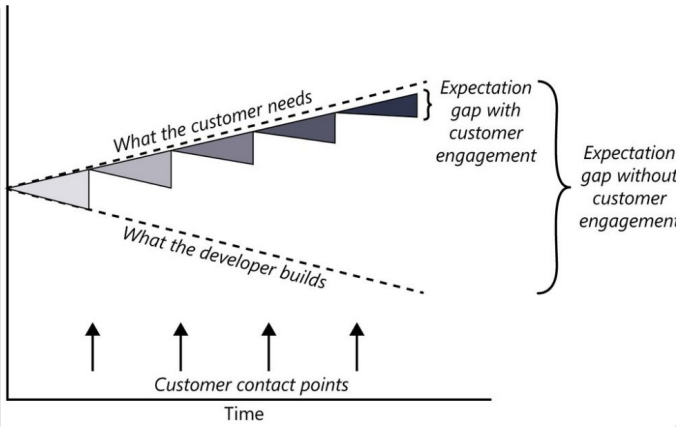
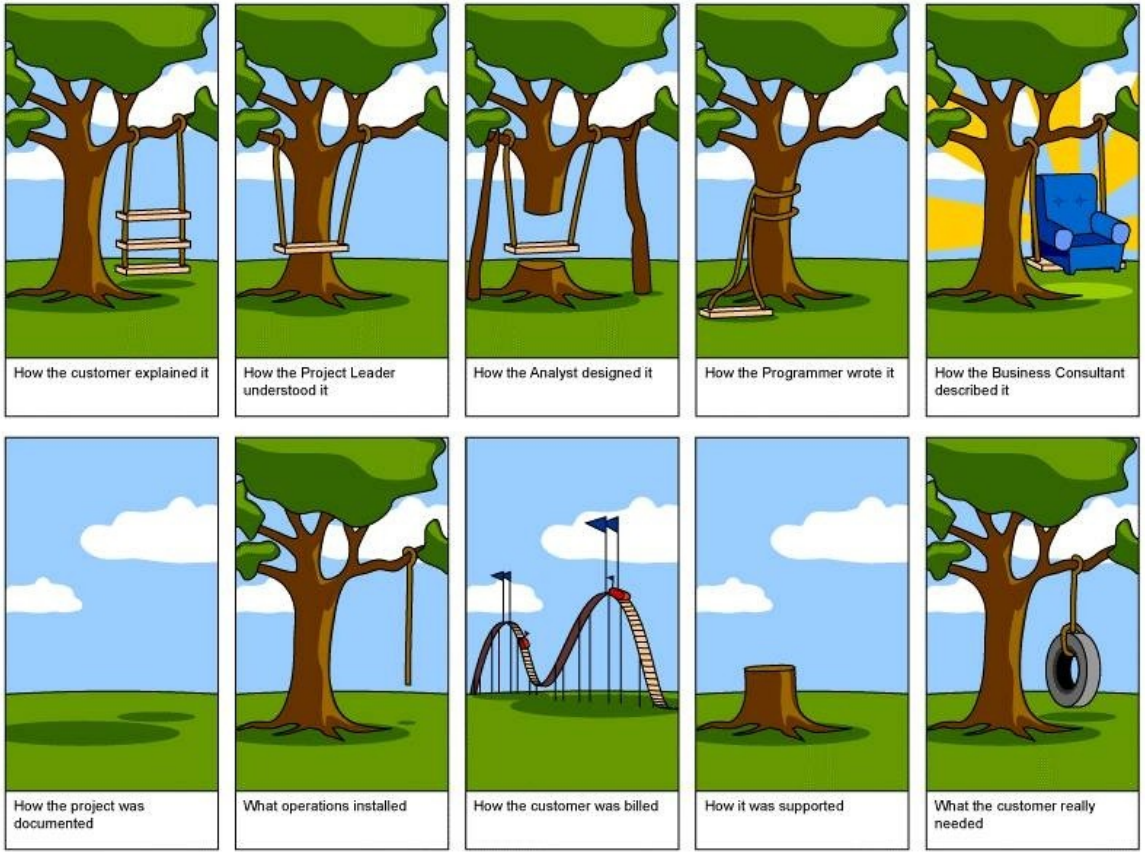


Figure 2-1. Frequent customer engagement reduces the expectation gap.



No Continuity or Consistency or Feedback

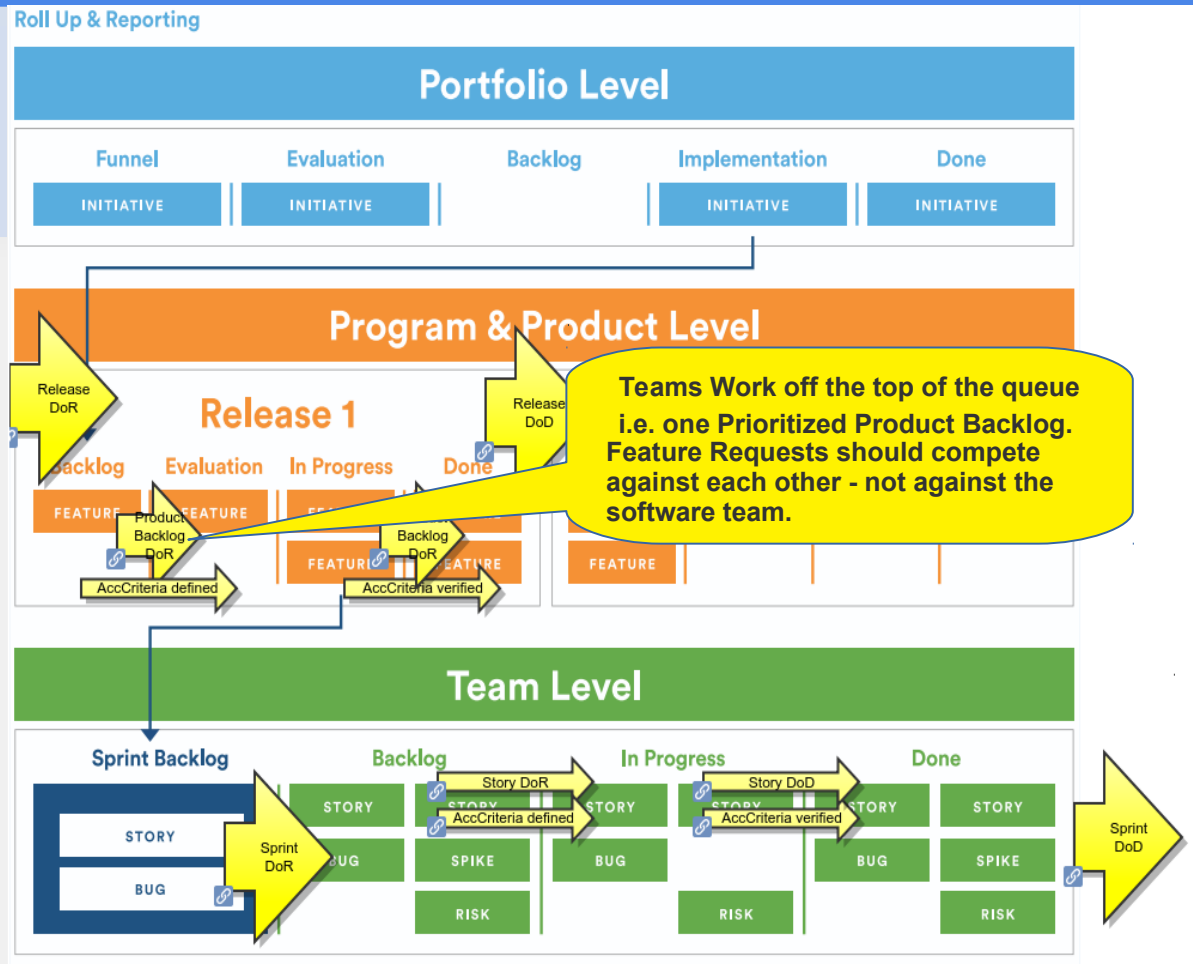
Feature Request: DoR, DoD, AC

For an enterprise, there are 3 levels of requirement definition and refinement:

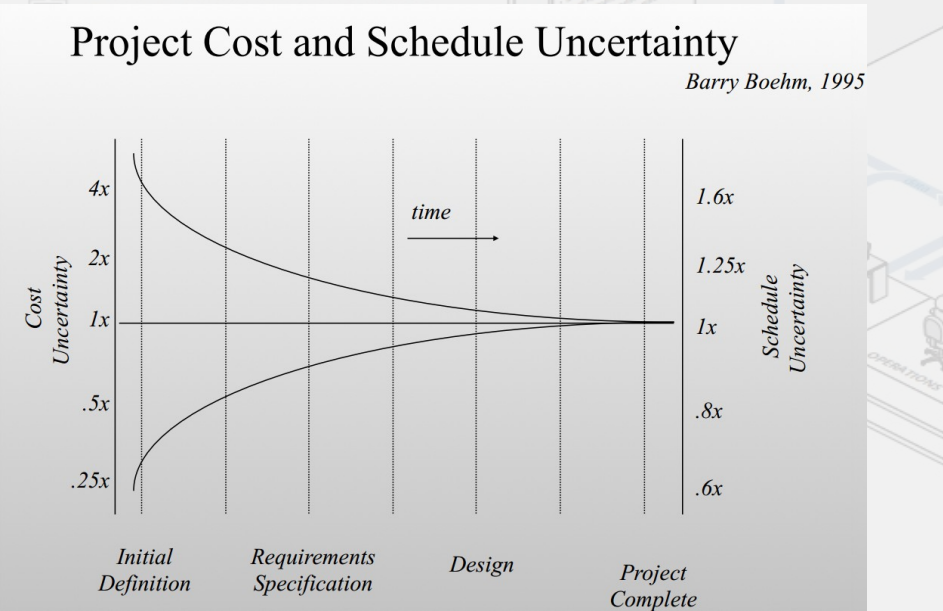
1. Portfolio
   a. e.g. a new product for a given value stream / business area
2. Product
   a. e.g. a set of features for that product
3. Team / Sprint
   a. e.g. a set of enhancements for each feature

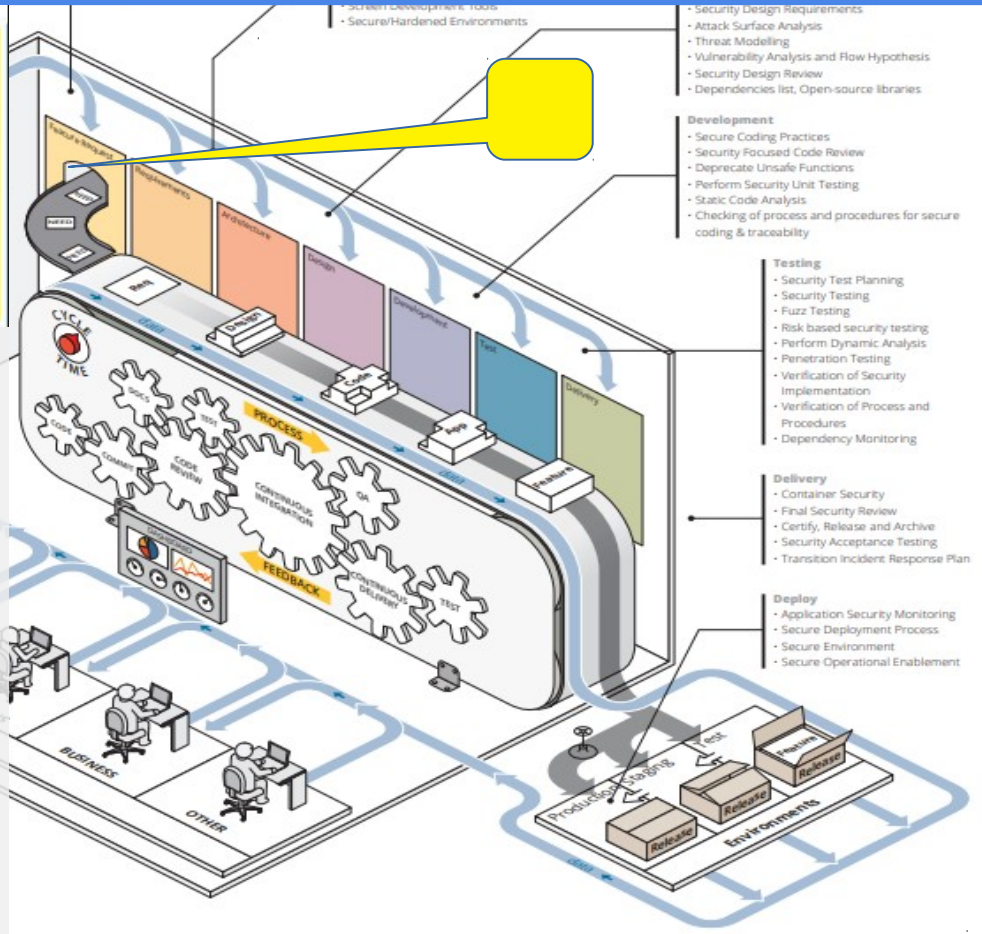
It is critical that at each level the following are defined before the item progresses:

1. Definition of Ready
2. Definition of Done
3. Acceptance Criteria

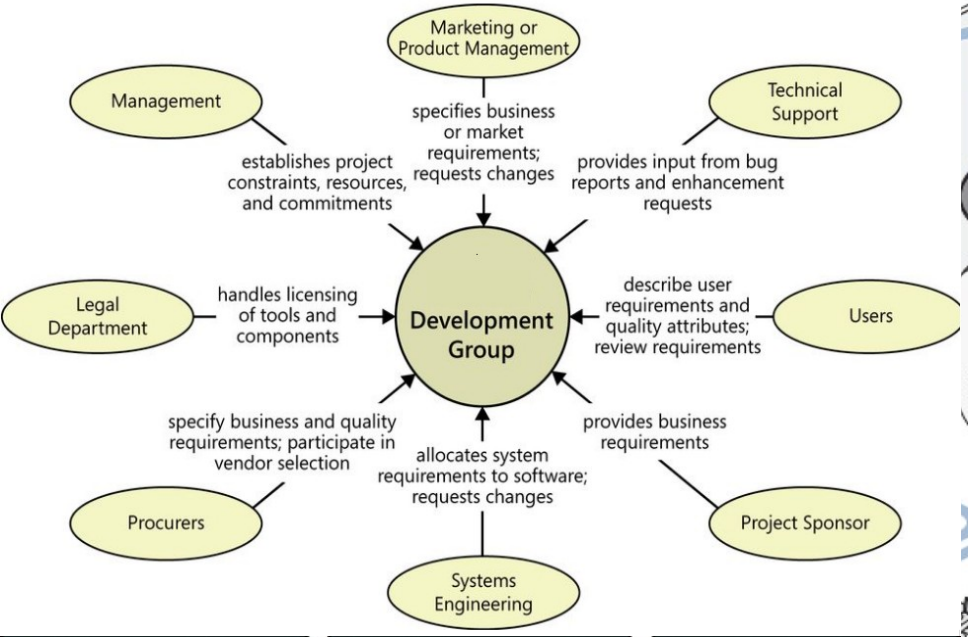


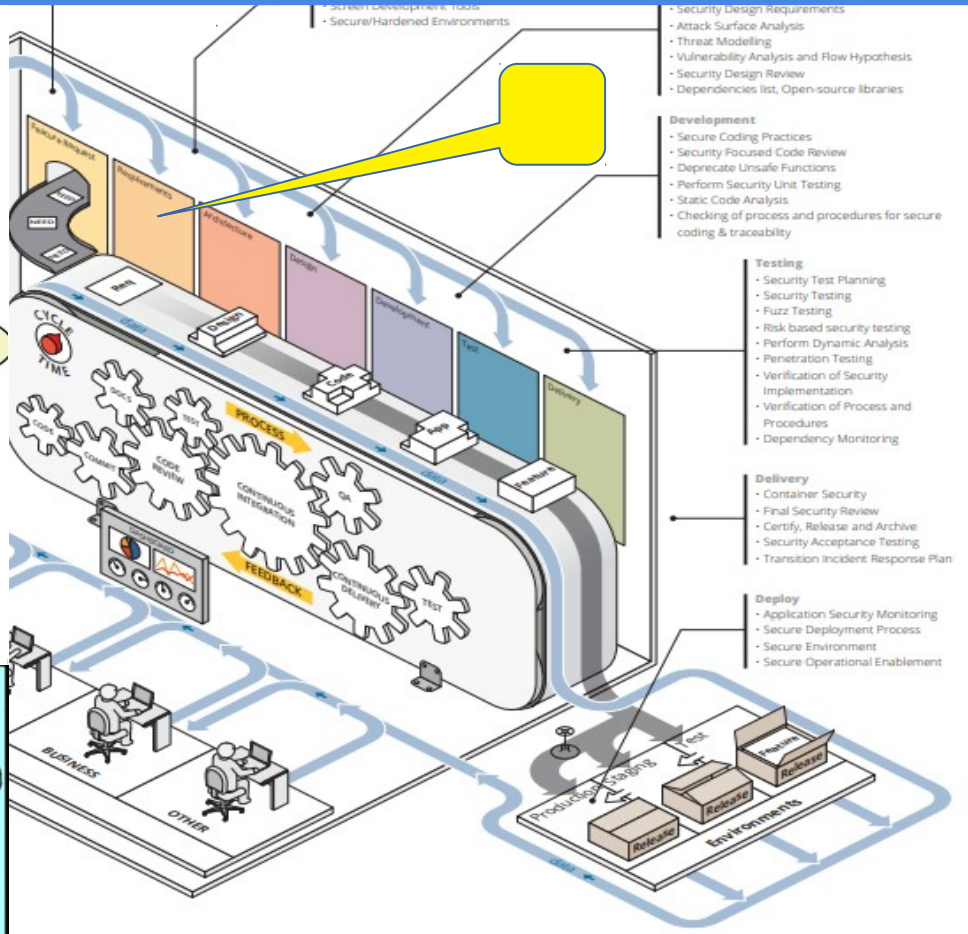
Feature Request: When can we have it?





Requirements





Requirements: Requirements, Documentation, Tests as Code

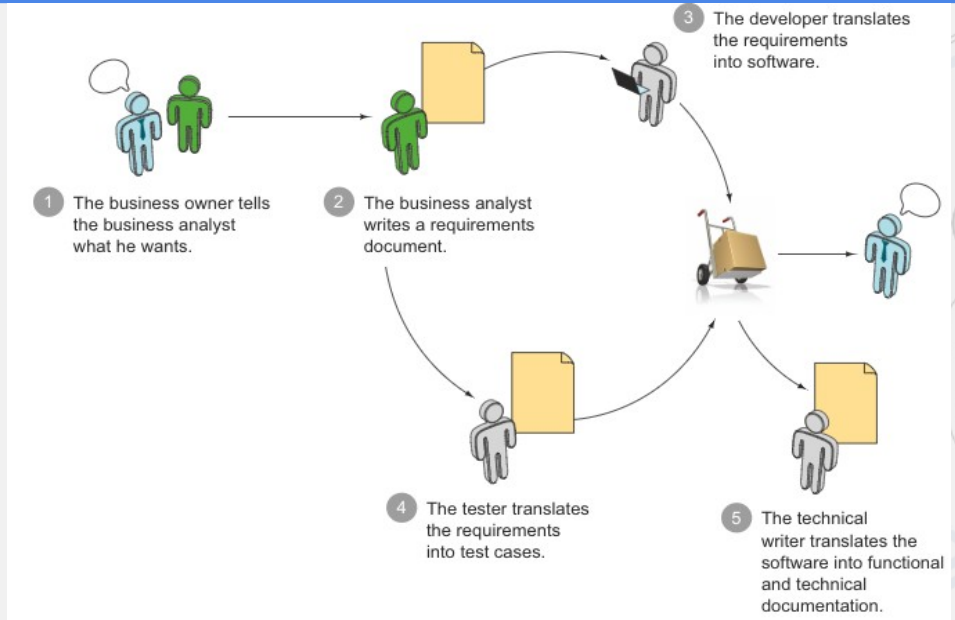


Figure 1.1 The traditional development process provides many opportunities for misunderstandings and miscommunication.

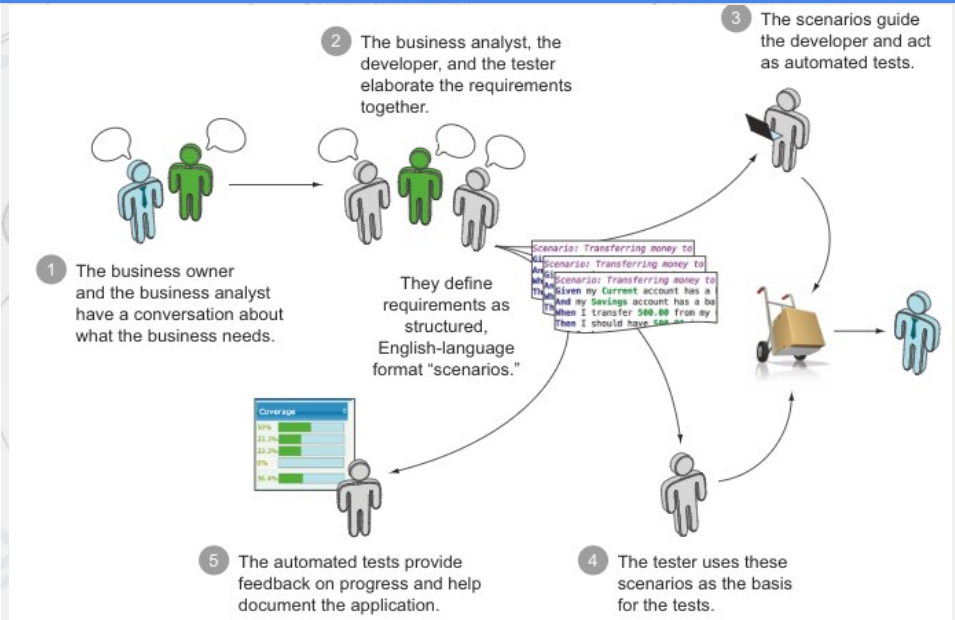


Figure 1.2 BDD uses conversations around examples, expressed in a form that can be easily automated, to reduce lost information and misunderstandings.

Behavior Driven Development (BDD) Requirements, Documentation, Tests as Code gives Continuity i.e. executable requirements specification.

Continuity + Consistency + Feedback

When Kent Beck (who originated the notion of a "story") developed his ideas on software development, he called out communication as a key value of effective teams. Stories are the building blocks of communication between developers and those who use their work. Story maps organize and structure these building blocks, and thus enhance this communication process—which is the most critical part of software development itself.

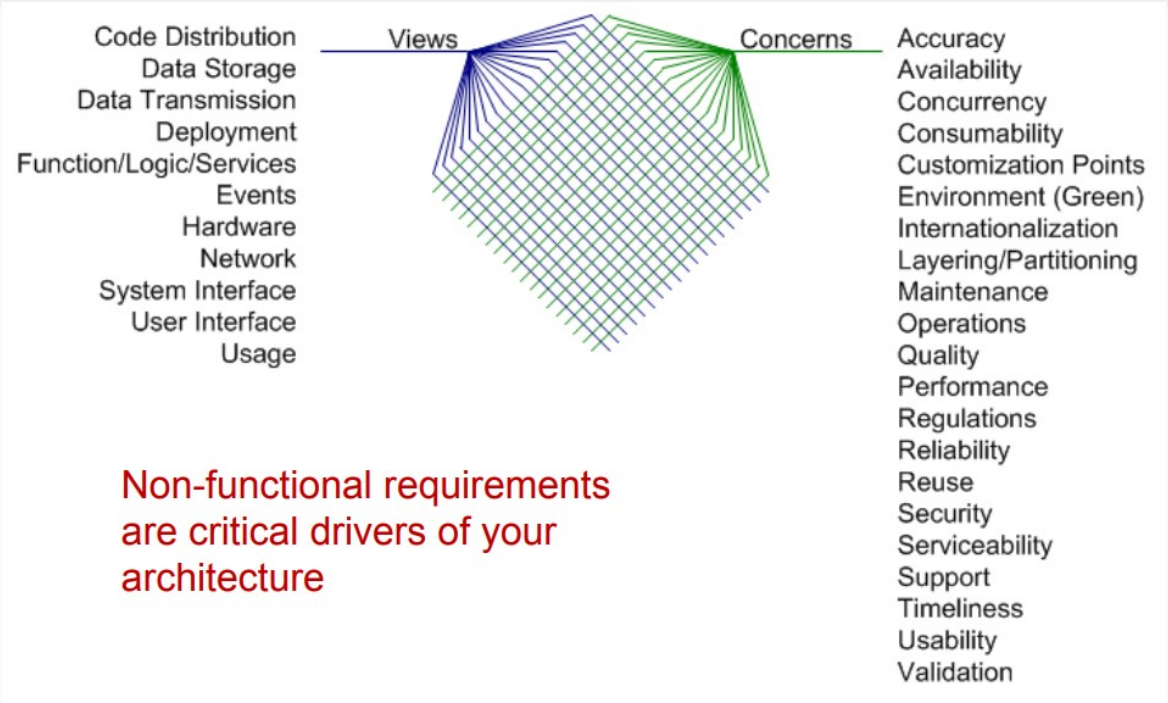
From the book: User Story Mapping

Architecture: Continuous Architecture Principles

1. **Architect products, not just solutions for projects:** Architecting products is more efficient than just designing point solutions to projects and focuses the team on its customers.
2. **Focus on Quality Attributes, not on functional requirements. Quality attribute requirements drive the architecture.**
3. **Delay design decisions until they are absolutely necessary.** Design architectures based on facts, not on guesses There is no point in designing and implementing capabilities that may never he used; it is a waste of time and resources.
4. **Architect for change - leverage "the power of small."** Big, monolithic, tightly coupled components are hard to change. instead, leverage small, loosely coupled services
5. **Architect for build, test, and deploy.** Most architecture methodologies exclusively focus on software building activities, but we believe that architects should be concerned about testing and deployment activities in order to support Continuous Delivery.
6. **Model the organization after the design** The way teams are organized drives the architecture and design of the systems they are working on

Architecture > Continuous Architecture principles > Focus on Quality Attributes

Focus on Quality Attributes, not on functional requirements. Quality attribute requirements drive the architecture.

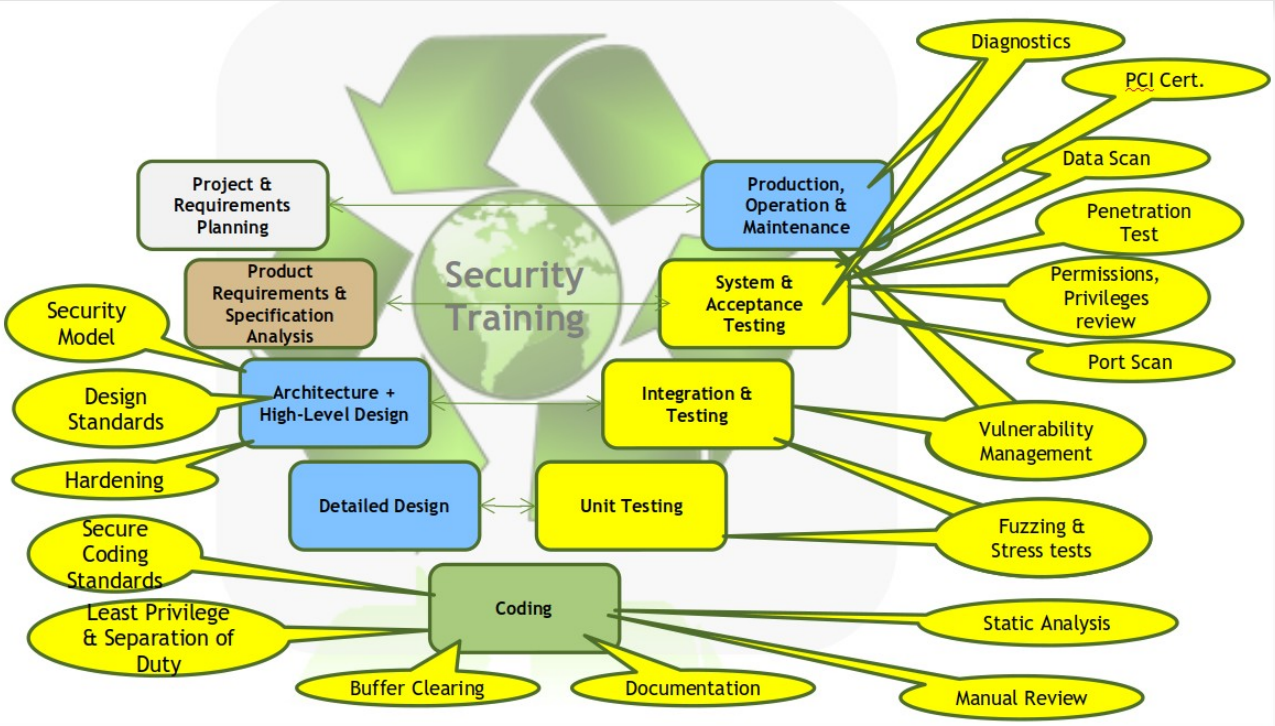


Development: Software Assurance: Lifecycle

Software assurance is defined as "the level of confidence that software is free from vulnerabilities, either intentionally designed into the software or accidentally inserted at any time during its lifecycle, and that the software functions in the intended manner."

Software Assurance = Quality + Security

Security Training (e.g. secure coding, applied cryptography) for engineers is the foundation for Software Assurance

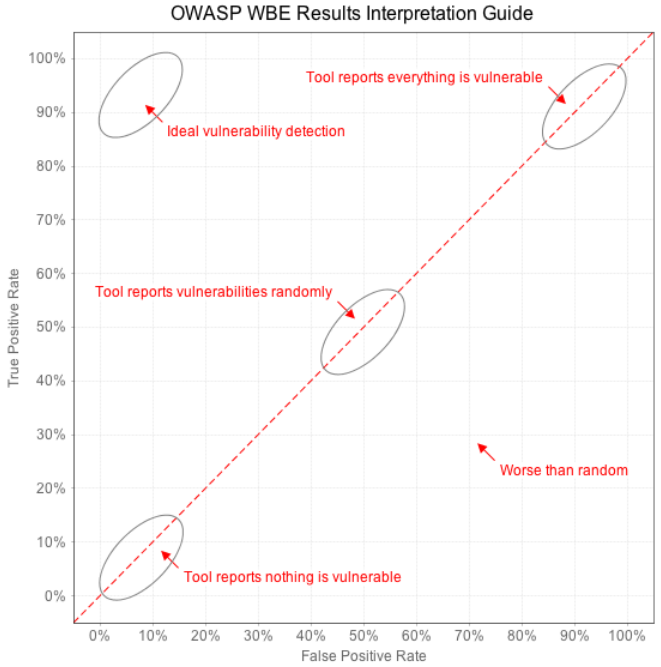


Development: Software Assurance: Right Tool for the Job

Signal-to-Noise

	Negative	Positive
True	1. issue was not reported - and was not an issue 2. the tool correctly did not report something as an issue	1. issue was reported - and was an issue 2. we want the tool to report as many of these as possible 3. this is the **Signal** we are looking for
False	1. issue was not reported - and was an issue 2. the tool failed to detect this issue 3. this is loss of signal.	1. issue was reported - and was not an issue 2. we want the tool to report as few of these as possible - as these causes **Noise** in the results and hides the signal

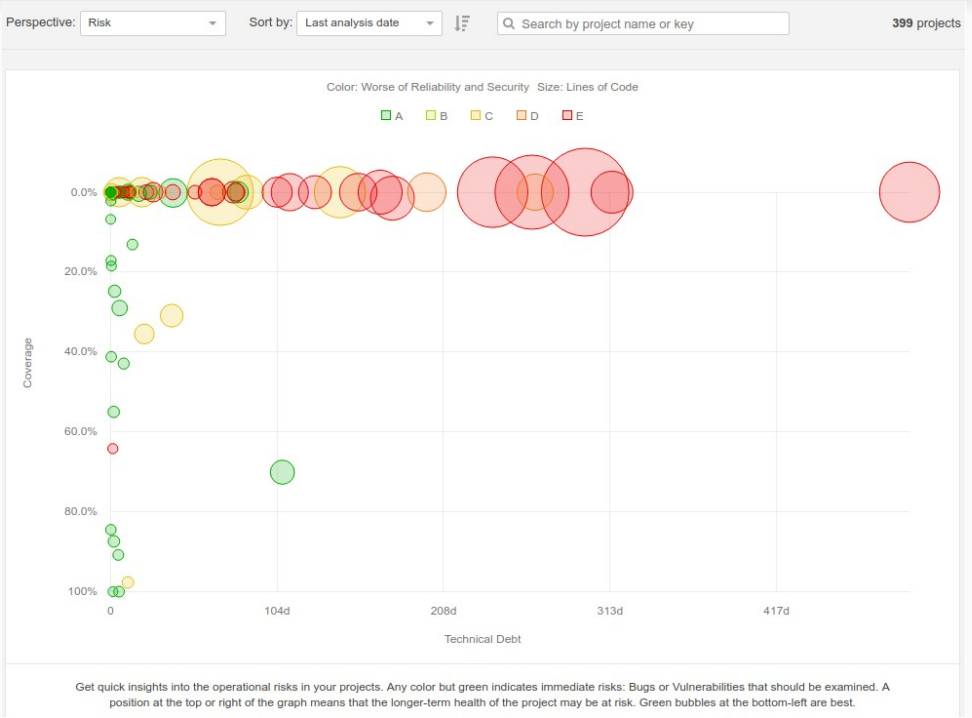
Using multiple independent/orthogonal tools can be more optimal than using one tool (see notes for worked example)



Development: Software Assurance: SonarQube

SonarQube is a continuous inspection dashboard tool that measures source code quality across 7 axes

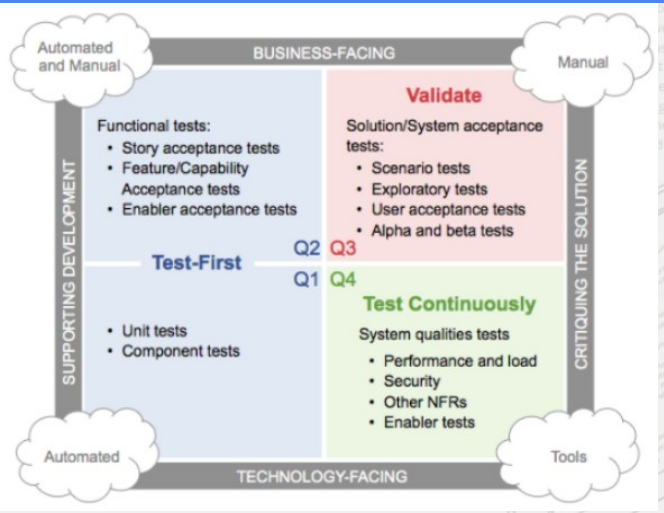
1. Coding standards—respect coding standards and follow best practices
2. Potential bugs—eliminate code violations to prevent vulnerabilities
3. Documentation and comments—provide documentation especially for the Public API, the source code
4. Duplicated code—isolates and refines duplications,
5. Don't Repeat Yourself Complexity—equalizes disproportionate distributed complexity among components; eliminates complexity if possible
6. Test coverage—writes unit tests, especially for complex parts of the software
7. Design and architecture—minimize dependencies

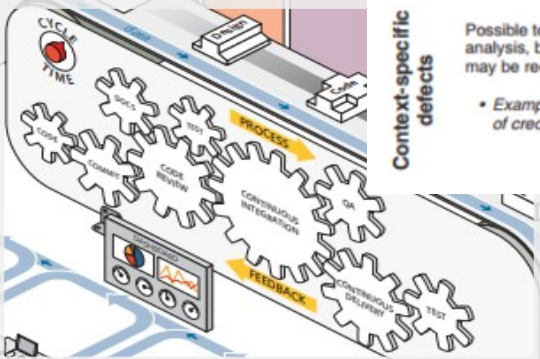


Development: Software Assurance: Secure Coding

1. We developed a Secure Coding training as follows:
    a. Using SEI CERT C Coding Standard (C++, Java, Perl, Android also supported) as the foundation.
    b. Feedback the issues found by the tools (Cppcheck, Coverity, SonarQube, Public Vulnerability (CVE), Manual Code Reviews)
        i. into our Secure Coding training (CWEs)
            1. For every CWE, we listed 2 known bad examples from our code - along with a known good example and guidelines.
        ii. into our tools/checkers
            1. E.g. check and warning at compile time for deprecated/unsafe functions e.g. strcpy()
2. Note:
    a. CWE stands for Common Weakness Enumeration, and has to do with the vulnerability—not the instance within a product or system.
    b. CVE stands for Common Vulnerabilities and Exposures, and has to do with the specific instance within a product or system—not the underlying flaw

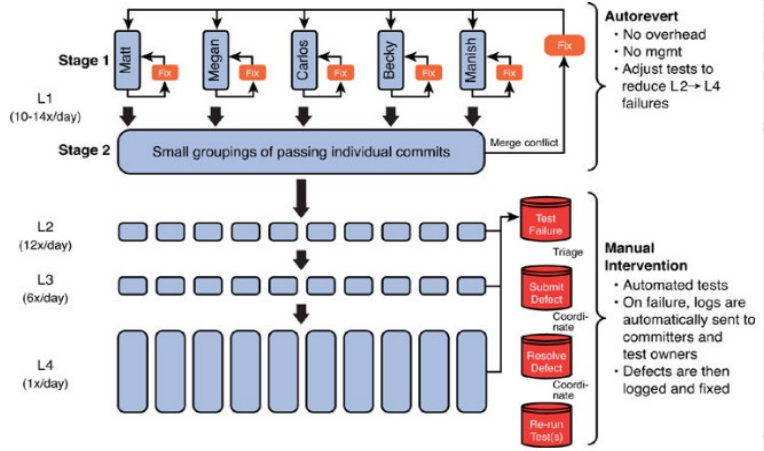
Test: Often and Early: Continuous And Automatic



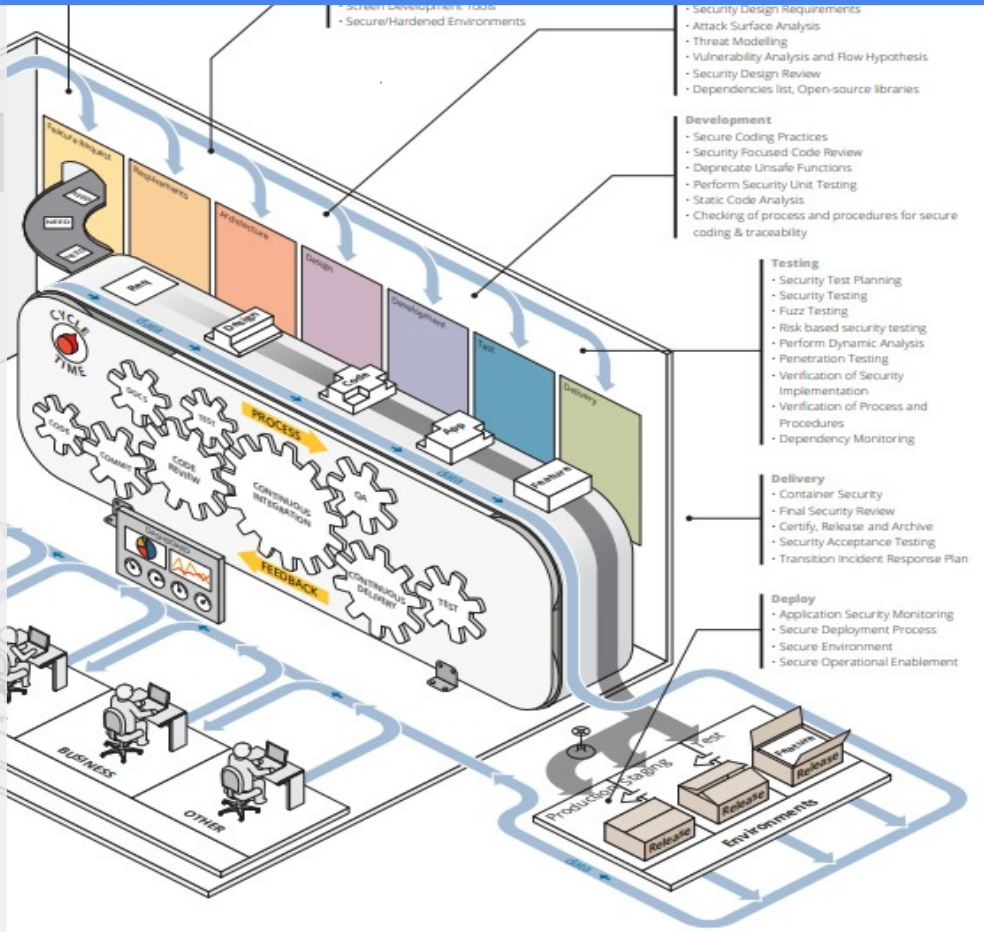


	Visible in the code	Visible only in the design
Generic defects	Static analysis sweet spot. Built-in rules make it easy for tools to find these without programmer guidance. • *Example: buffer overflow.*	Most likely to be found through architectural analysis. • *Example: the program executes code downloaded as an email attachment.*
Context-specific defects	Possible to find with static analysis, but customization may be required. • *Example: mishandling of credit card information.*	Requires both understanding of general security principles along with domain-specific expertise. • *Example: cryptographic keys kept in use for an unsafe duration.*

Continuous Integration Continuous Test Pipeline

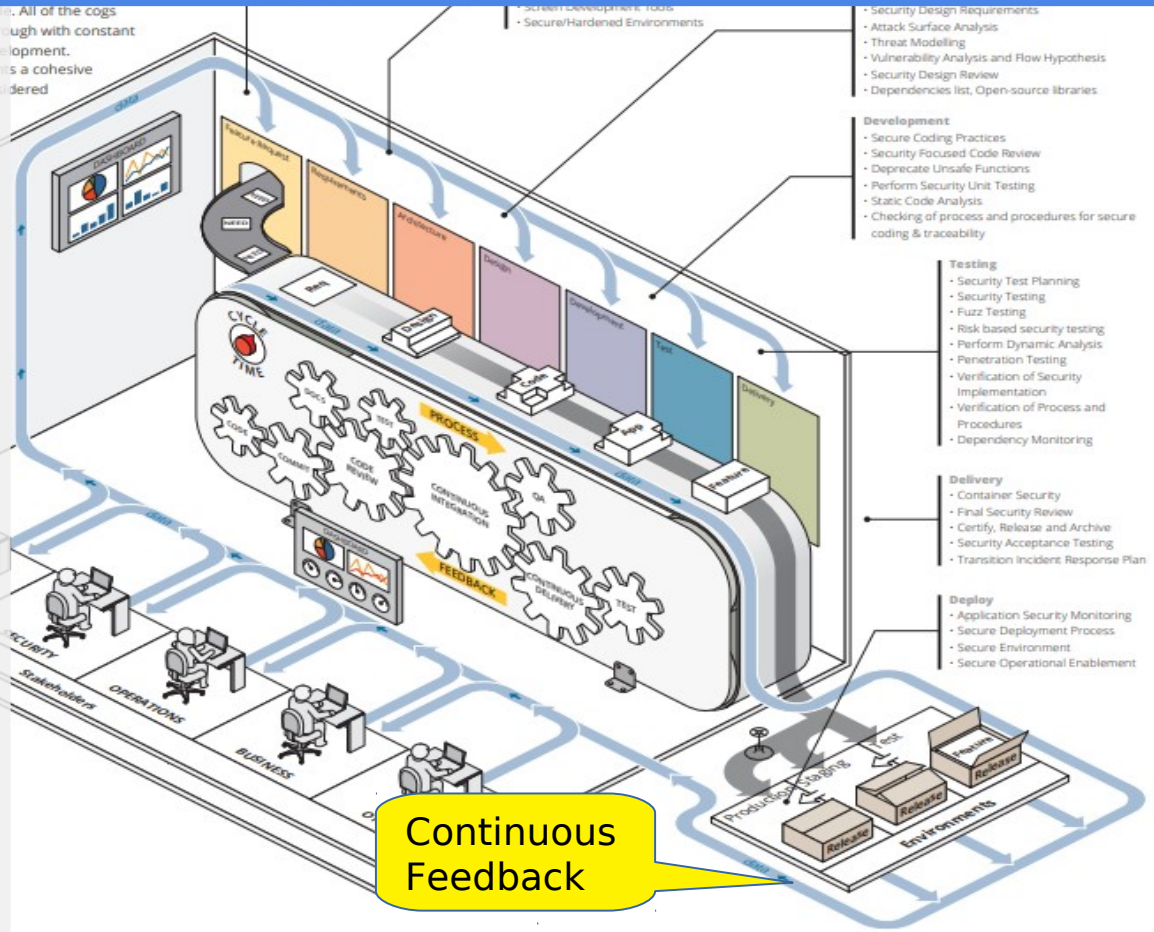


Integration	L0	• Pre-commit testing. Done by the developer on his/her own development machine before committing to the main trunk (at the bare minimum, includes running "qbar" locally).
Integration	L1	• Commit testing. Happens automatically at the time a developer commits new code to the main trunk of the SCM (broken up into Stage 1 and Stage 2 for easier autorevert). This is now our "Integrate on Queuing" (IQ) system.
Stability	L2	• Quick-turn broad-based testing by pillar (runs every 2 hours, across multiple test machines in parallel). Intended as a quick feedback loop to find broad-based failures from new commits in as narrow of a commit window as feasible.
Stability	L3	• Same purpose as L2, but on real hardware you can know that the full build, firmware download, boot, and general capability are working end-to-end. Runs every 4 hours.
Regression	L4	• Full regression test suite of all automated tests. Kicks off at midnight daily and provides complete view of the quality of the system. If passing rate drops substantially, should have seen a dip in L2 testing (otherwise, update the L2 list).



Delivery: Continuous Feedback

1. Current: Customer reports an issue, get logs, diagnose, release a fix
2. Short Term Future: Issue is found before customer sees or reports it
3. Long Term Future: Predictive Analytics: Issue is predicted before it happens and fixed before customer sees it



The DevOps Factory

DevOps is a modern software development approach that strives to bring development and operations teams together along with other stakeholders to improve efficiency and outcomes by focusing on shared business goals. DevOps follows and expands on key principles of the Agile software development and Lean engineering movements and represents a fundamental shift in how large, distributed enterprise organizations develop and deliver software.

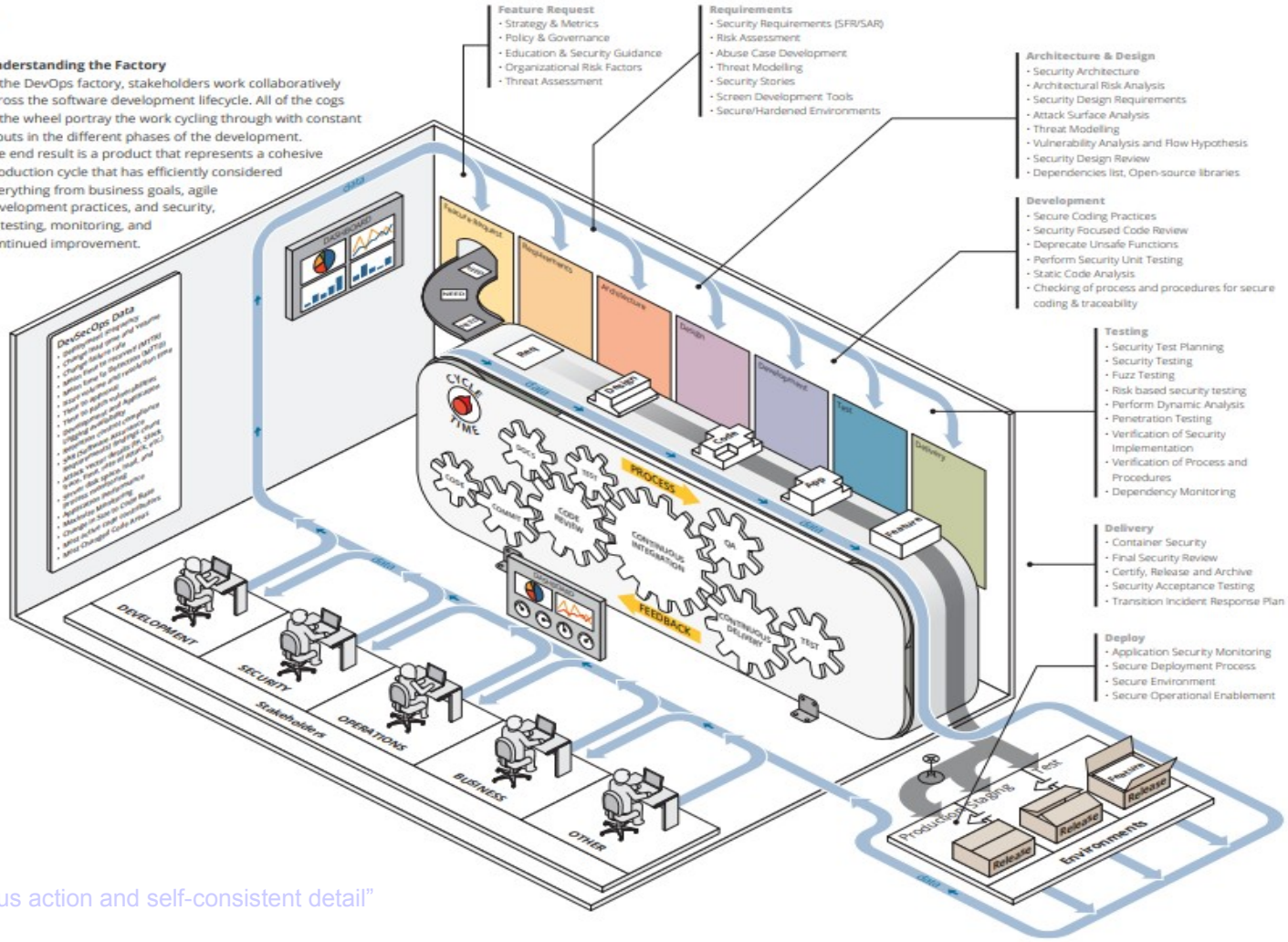
By cultivating cross-functional collective engagement in software development projects throughout the software development lifecycle (SDLC), DevOps affects the people, processes, and technology of an organization. DevOps also requires adopting and implementing cutting-edge practices based on the primary tenants of collaborative culture, automation, data-driven processes, infrastructure as code, and ubiquitous, real-time system monitoring.

The features and benefits of DevOps include

- **Consistently developing** software systems with higher quality and accuracy of project budgeting and estimation
- **Increased visibility** and stakeholder input into features for the next release as it is being developed
- **Engaging stakeholders** early and consistently throughout the SDLC, leading to fewer defects and incorrect requirements
- **Building trust** between software development and IT, enabling organic process improvement and risk mitigation
- **Maximizing business value** by enabling technical staff to adapt to changing requirements or environmental factors

Understanding the Factory

In the DevOps factory, stakeholders work collaboratively across the software development lifecycle. All of the cogs in the wheel portray the work cycling through with constant inputs in the different phases of the development. The end result is a product that represents a cohesive production cycle that has efficiently considered everything from business goals, agile development practices, and security, to testing, monitoring, and continued improvement.



Continuity: “the maintenance of continuous action and self-consistent detail”

The Future

The Last 10 Years

1. It’s about the Customer - not the Product
2. Continuous Information across Lifecycle
    a. Sharing information (Confluence)
    b. Executive Dashboards driven from Jira (not PowerPoint where dreams come true)
3. Going from “Thinking we’re Agile” to “Being Agile”
    a. “We’re doing standups, sprints, and using Jira. We’re Agile. Right?”
    b. Agile at SW team level (Sprint backlog)
    c. Agile at Enterprise level (Sprint backlog, Product Backlog, Portfolio Backlog)

The Next 10 Years

1. More Continuity between Requirements - Documentation - Test - SW
    a. Less heavy lifting required by people to build software
    b. The breadth and depth and rate of change of new technologies means that developers can’t keep pace
    c. Value moves further up the stack - the lower layers become commodities
2. From Continuous Delivery to Predictive Delivery
    a. More data measured and analysed by machines allowing prediction at each stage in the lifecycle.
    b. From Feed-Back to Feed-Forward
    c. From Lagging indicators to Leading indicators
    d. Predictive Analytics
3. From People to Machines as Consumers/Customers/Creators
4. From Agile to Agile at Scale

Recommended Reading: Books

1. Hands-On Security in DevOps: Ensure continuous security, deployment, and delivery with DevSecOps 30 Jul 2018
2. Agile Application Security Paperback – Enabling Security in a Continuous Delivery Pipeline - 26 Sep 2017
3. Accelerate: The Science of Lean Software and Devops: Building and Scaling High Performing Technology Organizations Paperback – 30 Apr 2018
4. A Practical Approach to Large-Scale Agile Development: How HP Transformed LaserJet FutureSmart Firmware (Agile Software Development)

Recommended Reading: Deliver Value at Scale: SAFe

1. Agile was developed with small teams in mind.
2. There are several approaches to scaling agile per AgileScalingKnowledgebase. Of these ScaledAgileFramework (SAFe) is the most mature and supported.
3. Problems of Scale: Interdependencies between teams, components, deliverables. Capacity planning.