BUILDING A SECURE SOFTWARE DEVELOPMENT LIFECYCLE

By Avi Douglen
CEO, Bounce Security

Bounce SECURITY

“All software developers
are security engineers
(whether they know it, admit it, or like it)”

- Jim Manico

Bounce SECURITY

Agenda

- Background and History
- Primary Activities
- Positive Practices

I am… **Avi Douglen**

- Email: AviD@BounceSecurity.com
- Twitter: @sec_tigger
- He / Him

- The important stuff:
  - *Whisky: smokey*
  - *Beer: stout*
  - *Coffee: strong*

- Product Security Consulting Bounce SECURITY
- OWASP Israel Leader
- Global Board of Directors
- Threat Model Project Leader
- Moderator Security.StackExchange
- Startup Advisor @ OurCrowd Labs/02
- Co-Author of TM Manifesto

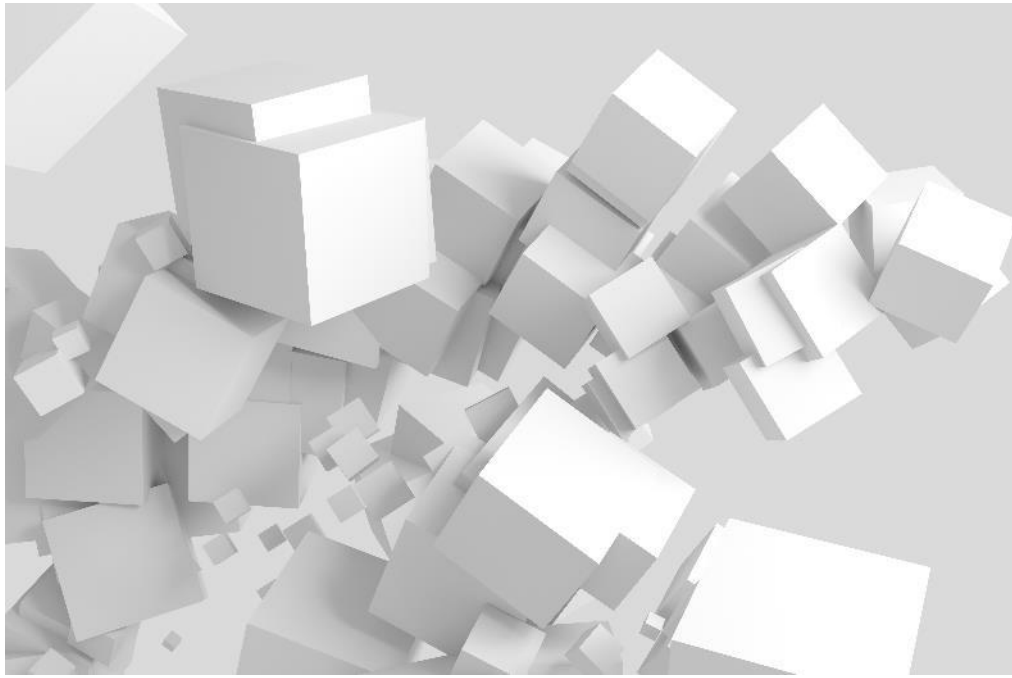
Bounce SECURITY

History of SSDLC

Bounce SECURITY

SDLC

- Software
- Development
- Life
- Cycle



SSDLC

- Secure
- Software
- Development
- Life
- Cycle

SDLC

- Started as Systems Development Lifecycle
- Dates back to 1960s
- Most common structures:
  - *Waterfall: One long linear process with distinct steps*
  - *Agile: An iterative process with shorter overlapping increments*

SSDLC Concept

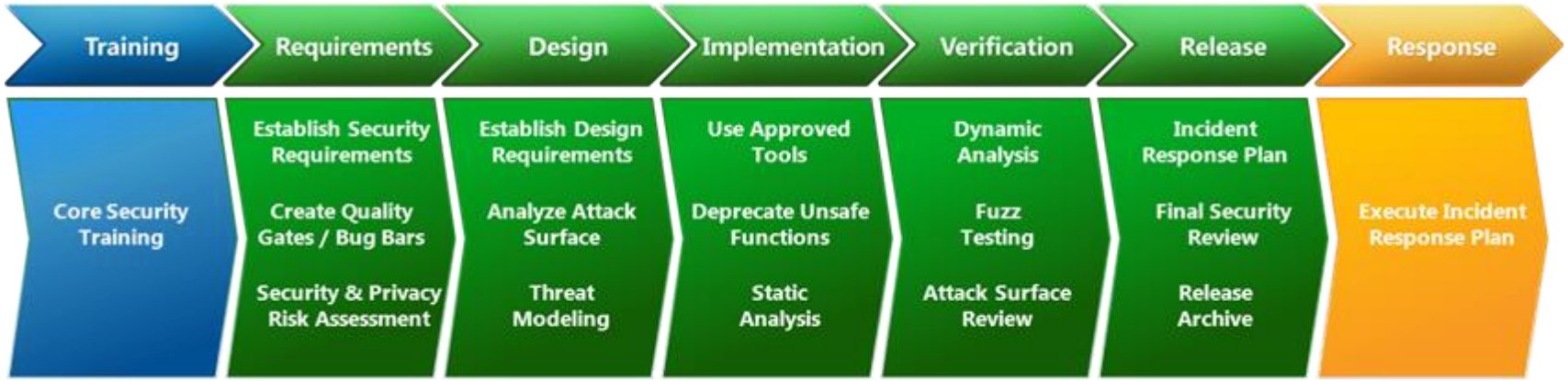
- Problem statement:
  - *How do we make security part of the SDLC?*

- Solution:
  - *Start some security related "activities"*
  - *Map to SDLC "stages"*
  - *Make activities convenient/natural for developers*

Bounce SECURITY

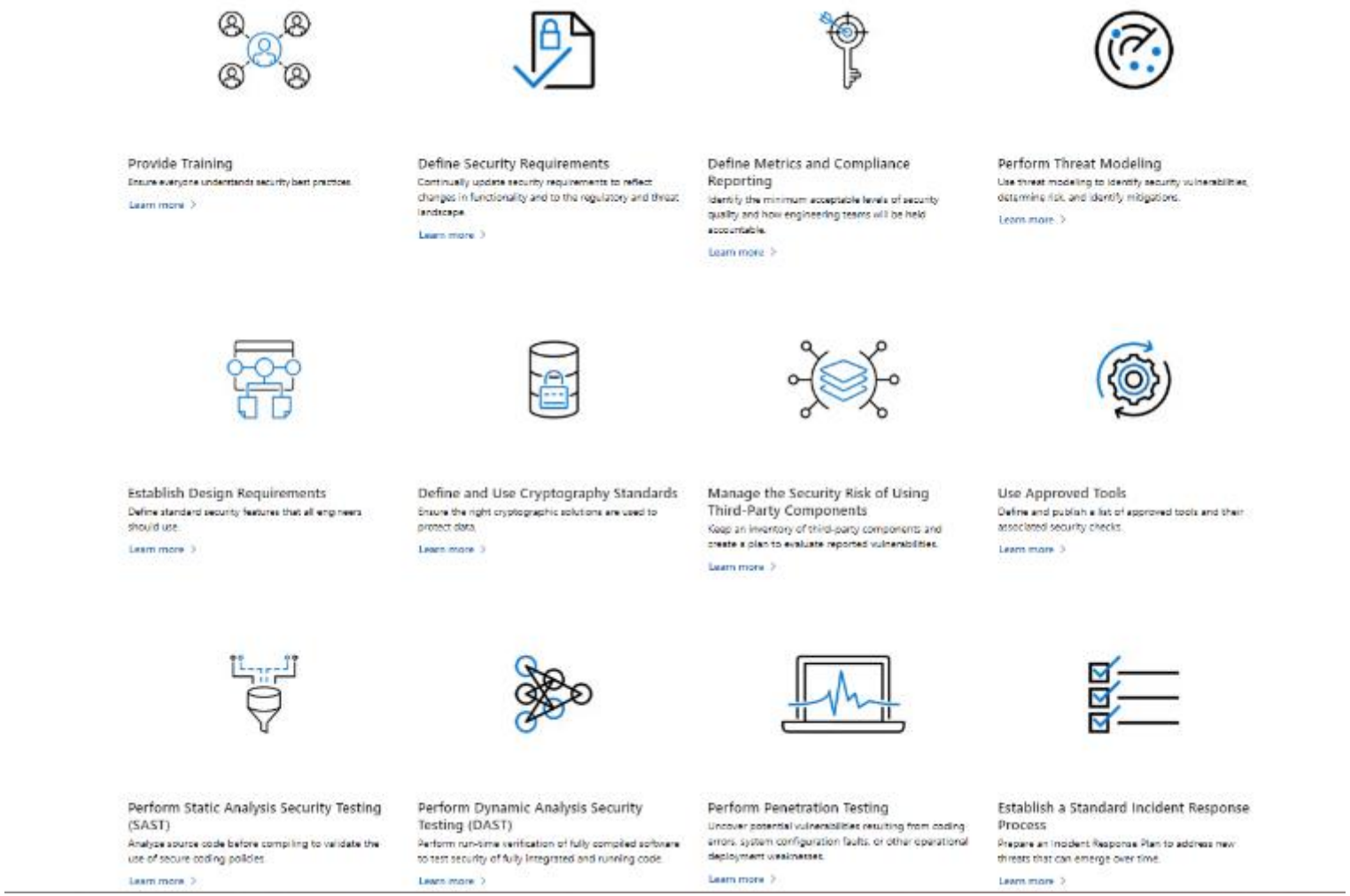
Common models - SDL

- Originated by Microsoft in 2004
- Various modifications since then including:
  - *Version customized to Agile*
  - *Simplified version*



https://learn.microsoft.com/en-us/windows/security/threat-protection/msft-security-dev-lifecycle

SDL Current State



Provide Training
Ensure everyone understands security best practices.
Learn more >

Define Security Requirements
Continually update security requirements to reflect changes in functionality and to the regulatory and threat landscape.
Learn more >

Define Metrics and Compliance Reporting
Identify the minimum acceptable levels of security quality and how engineering teams will be held accountable.
Learn more >

Perform Threat Modeling
Use threat modeling to identify security vulnerabilities, determine risk, and identify mitigations.
Learn more >

Establish Design Requirements
Define standard security features that all engineers should use.
Learn more >

Define and Use Cryptography Standards
Ensure the right cryptographic solutions are used to protect data.
Learn more >

Manage the Security Risk of Using Third-Party Components
Keep an inventory of third-party components and create a plan to evaluate reported vulnerabilities.
Learn more >

Use Approved Tools
Define and publish a list of approved tools and their associated security checks.
Learn more >

Perform Static Analysis Security Testing (SAST)
Analyze source code before compiling to validate the use of secure coding policies.
Learn more >

Perform Dynamic Analysis Security Testing (DAST)
Perform run-time verification of fully compiled software to test security of fully integrated and running code.
Learn more >

Perform Penetration Testing
Uncover potential vulnerabilities resulting from coding errors, system configuration faults, or other operational deployment weaknesses.
Learn more >

Establish a Standard Incident Response Process
Prepare an Incident Response Plan to address new threats that can emerge over time.
Learn more >

SDL – Key points

- Now appears as a series of security practices
- Focused on the what, not the how
- Some further links but not much in-depth guidance
- In summary:
  - *Good for ideas but not implementation*

OWASP Software Assurance Maturity Model (SAMM)

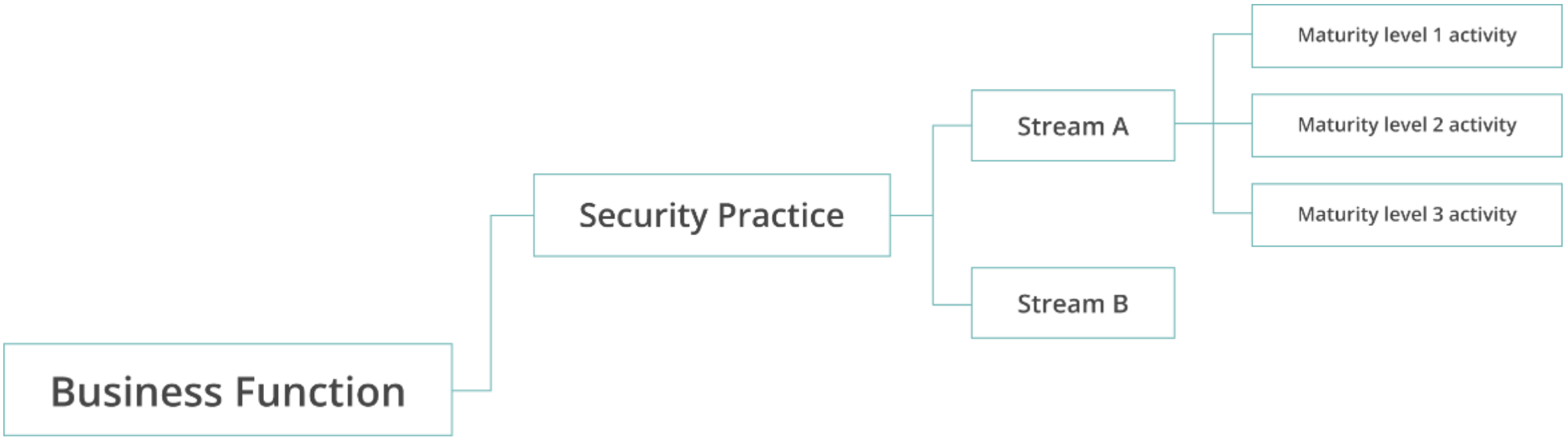
- Benchmark for Secure Software Development Lifecycle
- Framework for activities, to set a baseline or measure maturity
- Active community and lots of videos with information about it

Bounce SECURITY

OWASP SAMM – Framework Structure

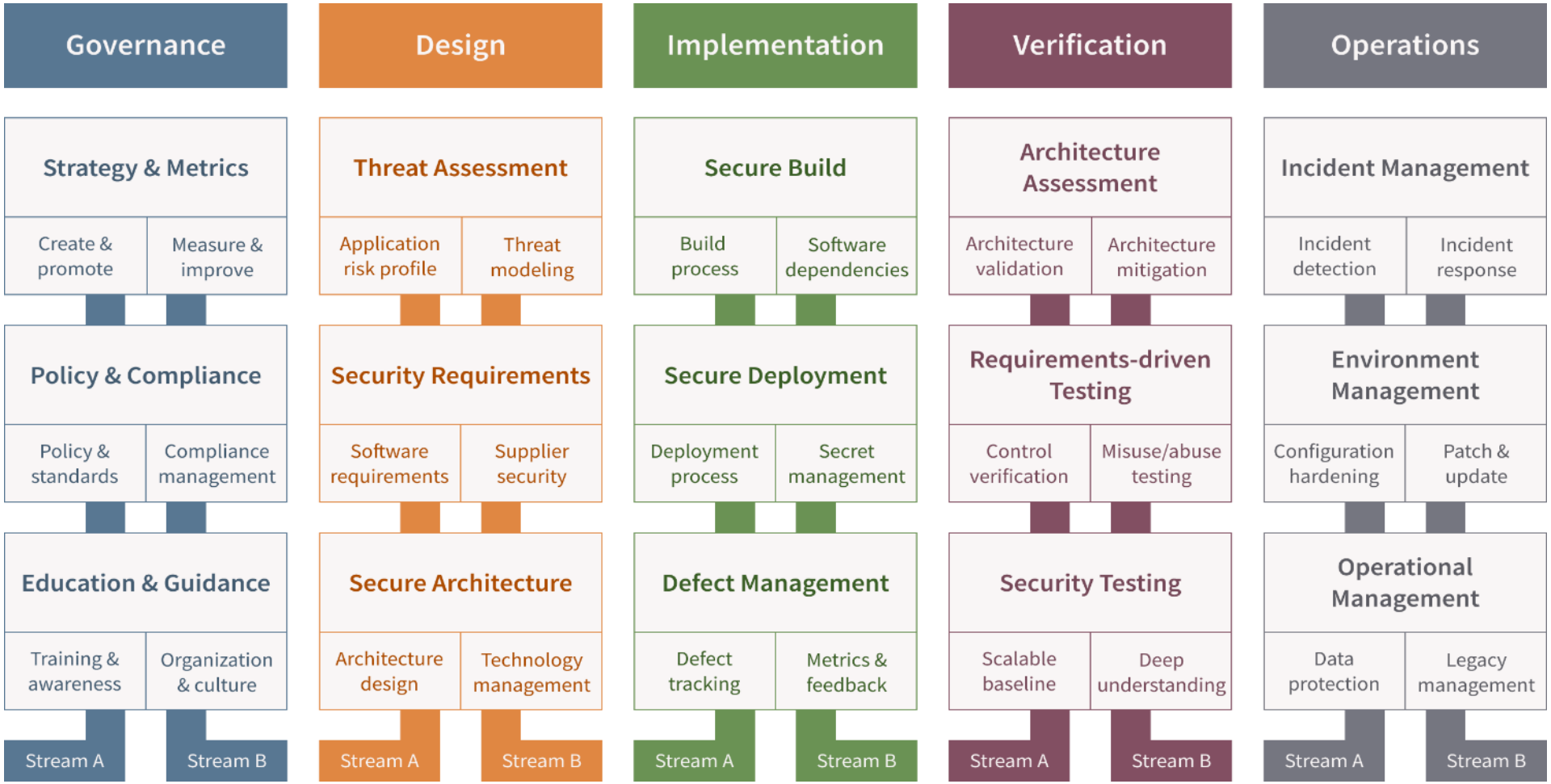
- Business Function:
  - *High-level activities related to development*
- Security Practice:
  - *Activity which provides security benefit*
- Stream:
  - *Specific approach for that activity*
- Maturity:
  - *Target implementation for growing levels of sophistication*

OWASP SAMM – Framework Structure



https://owaspsamm.org/about/#owasp-samm-structure

OWASP SAMM - Practices & Streams



OWASP SAMM – Key points

- Even better source of ideas than SDL
- Great for assessing current state and maturity
- Too detailed to use as an end goal / target state
- In summary:
  - *A great resource but not ideally suited for implementation*

Primary SSDLC Activities

Bounce SECURITY

Application Inventory

- Provides clear breakdown of what is where
  - *And who owns what*
- Hard to really do anything without this
- Helps allocate ownership and responsibilities

Bounce SECURITY

Business Impact Analysis (BIA)

- Define how a product (or feature) affects the organization
- Helps understand the relative value (or damage)
- How critical and sensitive is this
- Can be used to align security efforts to business goals

Feature Risk Weighting

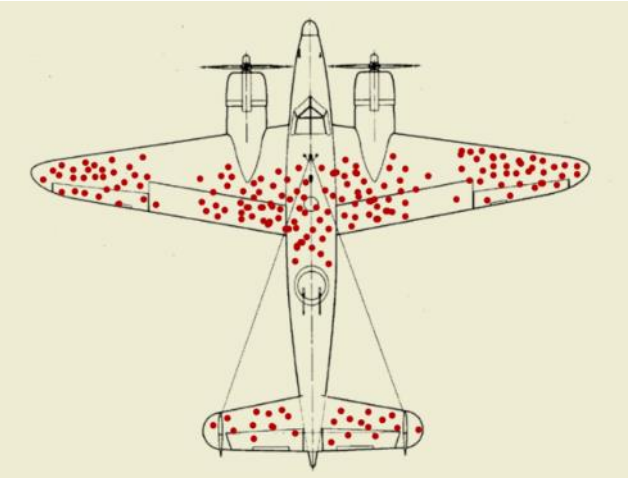
- Provides security weighting for a feature
- Helps guide how much security attention is required
- Important step to balance resources
- Derived from BIA

Security Requirements

- Build security into requirements along with other functionality
- Important to get business perspective
- By defining requirements, QA should be able to verify them

Threat Modeling

- Structured security-based analysis
- Framework to understand security issues
- Prioritize security efforts by risk
- Custom solutions instead of generic “best practices”



Security Design Review

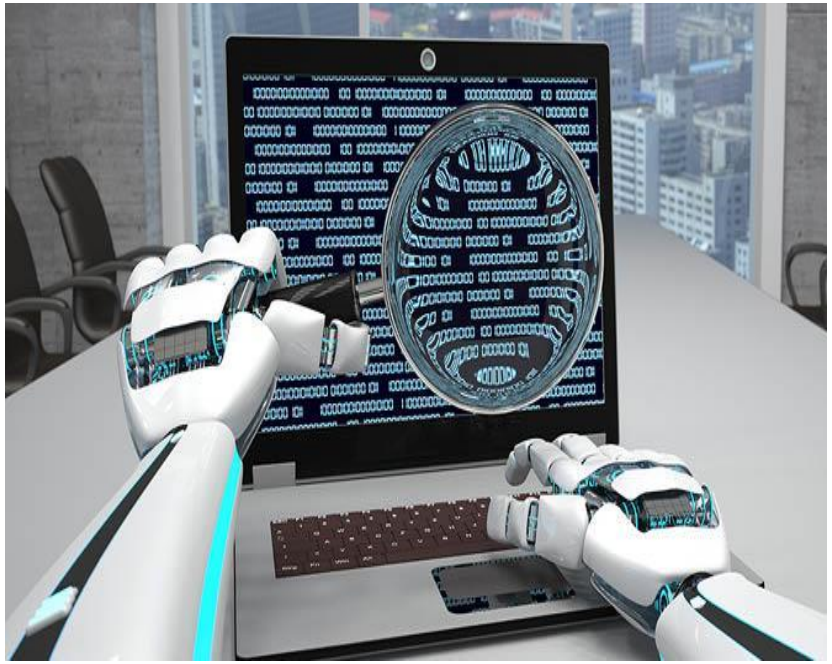
- Ideally built into the regular design review
- Looking for security issues or considerations
- The earlier the discovery, the easier to address

Secure Coding Guidance

- Provide developers with practical guidance
  - *For their language / platform / framework*
- May also use organization specific libraries
- The best solutions are tailored to the developer/organization

Automated Code Scanning

- Static Analysis (SAST) scans your code for common vulns
- Variety of tools / quality / coverage
- Make sure to fine-tune the SAST rules for your codebase!
- Should be run on every commit / push / merge / version



Third Party Library Risk Management

- ~80% of your app's code – isn't even yours
- Your dependencies have dependencies
  - *And those dependencies have their own dependencies*
- You can't secure what you don't know!
- Always have an SBoM (Software Bill of Material)
- SCA tracks 3rd party versions and known vulnerabilities

Security QA

- Ideally flows from security requirements / threat model
- Can also define standard security tests with QA team
- Good chance to create regression tests for past security vulnerabilities

Dynamic Application Vulnerability Scanning

- Dynamic Application Security Testing (DAST) scans a running application
- Most often refers to web apps / APIs / cloud apps
- Simulates a low level attacker
- Finds generic vulnerabilities

Bounce SECURITY

Primary SSDLC activities

Application Penetration Testing

- Someone external security tests application dynamically
- Fresh pair of eyes from a security expert
- Ideally done in an open as way as possible
- Best if done with specific goals
  - *e.g. based on a threat model*

Bounce SECURITY

Final Security Review

- Checking that security activities happened before development ends
- Ideally not hard gate but based on ongoing metrics
- Maybe part of policy enforcement

Bounce SECURITY

Primary SSDLC activities

Application Security Training

- Job focused training on security concepts
- Prepared for anyone involved in application development
- Ideally highly interactive and hands on
- Good opportunity to identify security champions

Positive Practices

Practice 1:
Process, Not a Project

Bounce SECURITY

Security is a Journey

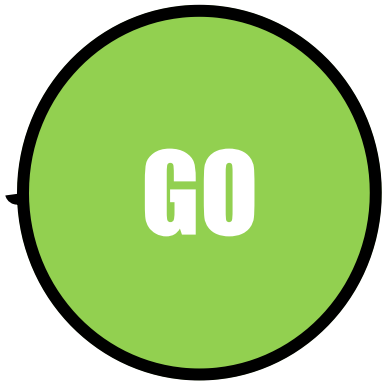
Practicalities

- Trying to implement every activity may not be valuable
  - *Need to prioritize*
  - *What brings clear value / solves a problem*
- Real implementation and "bedding-in" will take time
- You want activities to feel natural and "slot in" with regular development activities

Common Anti-Patterns

- Trying to implement a "full SSDLC"
- Taking a "big bang" approach
- Using a project plan or clearly defining a start and finish

Suggested Actions



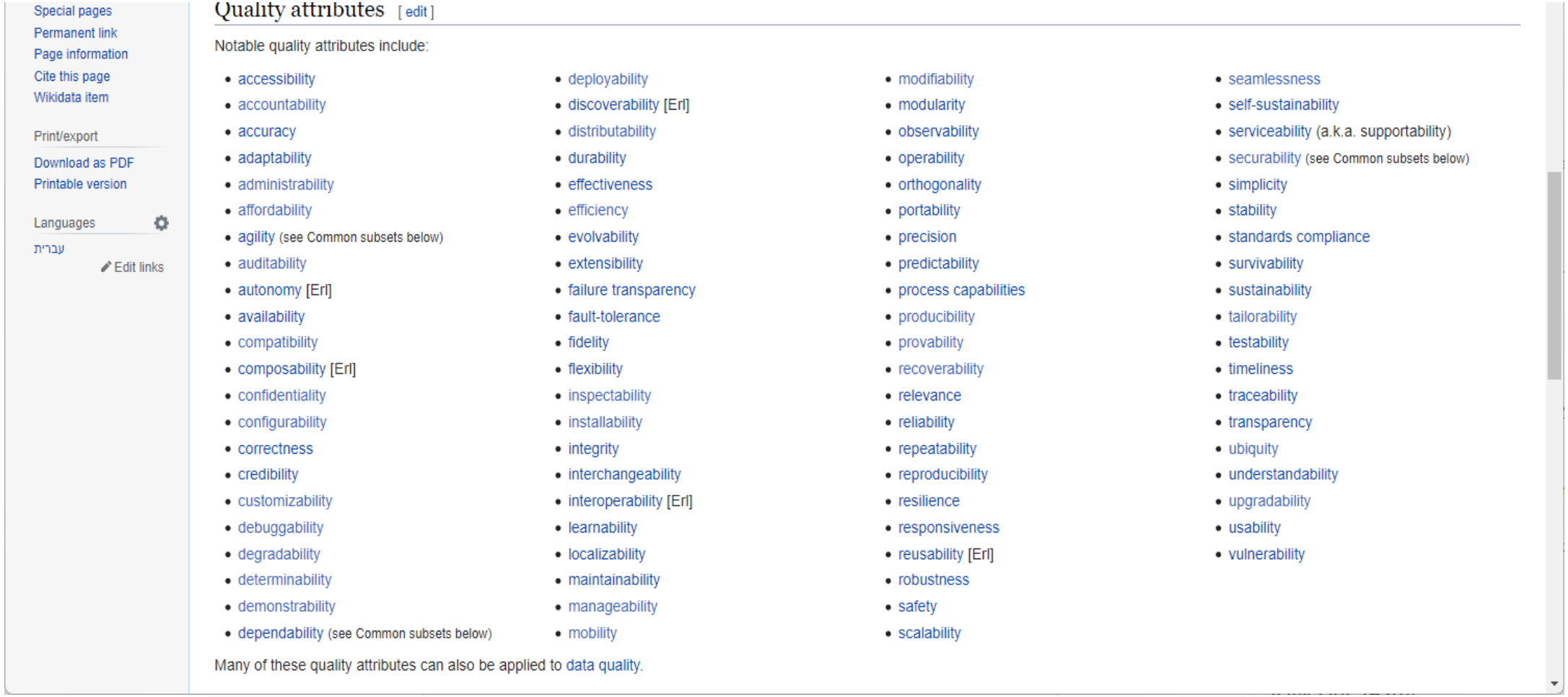
- Plan an incremental approach and manage expectations
- Prioritize activities based on mix of:
  - Easy to implement
  - High value (though complex)
- Define key milestones for the process

Practice 2:
Engineering Ownership

Bounce SECURITY

Security is not special…

Quality Attributes of Software



Special pages
Permanent link
Page information
Cite this page
Wikidata item

Print/export

Download as PDF
Printable version

Languages

עברית

Edit links

Quality attributes [edit]

Notable quality attributes include:

- accessibility
- accountability
- accuracy
- adaptability
- administrability
- affordability
- agility (see Common subsets below)
- auditability
- autonomy [Erl]
- availability
- compatibility
- composability [Erl]
- confidentiality
- configurability
- correctness
- credibility
- customizability
- debuggability
- degradability
- determinability
- demonstrability
- dependability (see Common subsets below)
- deployability
- discoverability [Erl]
- distributability
- durability
- effectiveness
- efficiency
- evolvability
- extensibility
- failure transparency
- fault-tolerance
- fidelity
- flexibility
- inspectability
- installability
- integrity
- interchangeability
- interoperability [Erl]
- learnability
- localizability
- maintainability
- manageability
- mobility
- modifiability
- modularity
- observability
- operability
- orthogonality
- portability
- precision
- predictability
- process capabilities
- producibility
- provability
- recoverability
- relevance
- reliability
- repeatability
- reproducibility
- resilience
- responsiveness
- reusability [Erl]
- robustness
- safety
- scalability
- seamlessness
- self-sustainability
- serviceability (a.k.a. supportability)
- securability (see Common subsets below)
- simplicity
- stability
- standards compliance
- survivability
- sustainability
- tailorability
- testability
- timeliness
- traceability
- transparency
- ubiquity
- understandability
- upgradability
- usability
- vulnerability

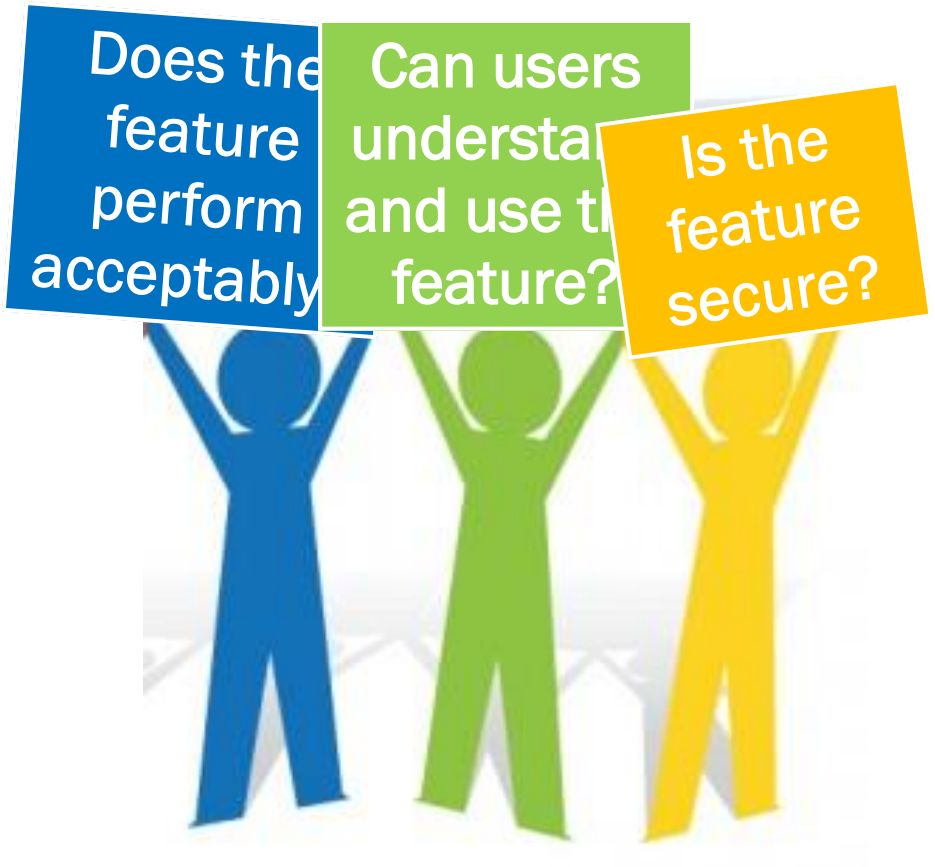
Many of these quality attributes can also be applied to data quality.

Quality Attributes – ISO/IEC 25010:2011

- Functional suitability
  - Functional completeness
  - Functional correctness
  - Functional appropriateness
- Performance efficiency
  - Time behaviour
  - Resource utilization
  - Capacity
- Compatibility
  - Co-existence
  - Interoperability
- Usability
  - Appropriateness recognizability
  - Learnability
  - Operability
  - User error protection
  - User interface aesthetics
  - Accessibility
- **Security**
  - **Confidentiality**
  - **Integrity**
  - **Non-repudiation**
  - **Accountability**
  - **Authenticity**
- Maintainability
  - Modularity
  - Reusability
  - Analysability
  - Modifiability
  - Testability
- Portability
  - Adaptability
  - Installability
  - Replaceability
- Reliability
  - Maturity
  - Availability
  - Fault tolerance
  - Recoverability

https://www.iso.org/obp/ui/#iso:std:iso-iec:25010:ed-1:v1:en

Just Another Attribute



Bounce SECURITY

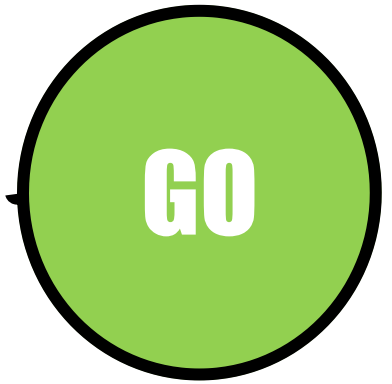
Practicalities

- Security is NOT everyone's job
- Security needs to "shift up" to get engineering buy-in
- Development/engineering needs to take the lead on security

Common Anti-Patterns

- Security get a blessing from engineering rather than ownership
- Security trying to add to developer workload "from the side"
- AppSec expected to own everything

Suggested Actions



- All new activities have clear ownership
  - *Accountable/Responsible should be engineering*
  - *AppSec experts should provide consultation only*
- Overall ownership of software security:
  - *Product Management*
  - *Engineering*
- Clarify for all new activities how to ensure it will happen

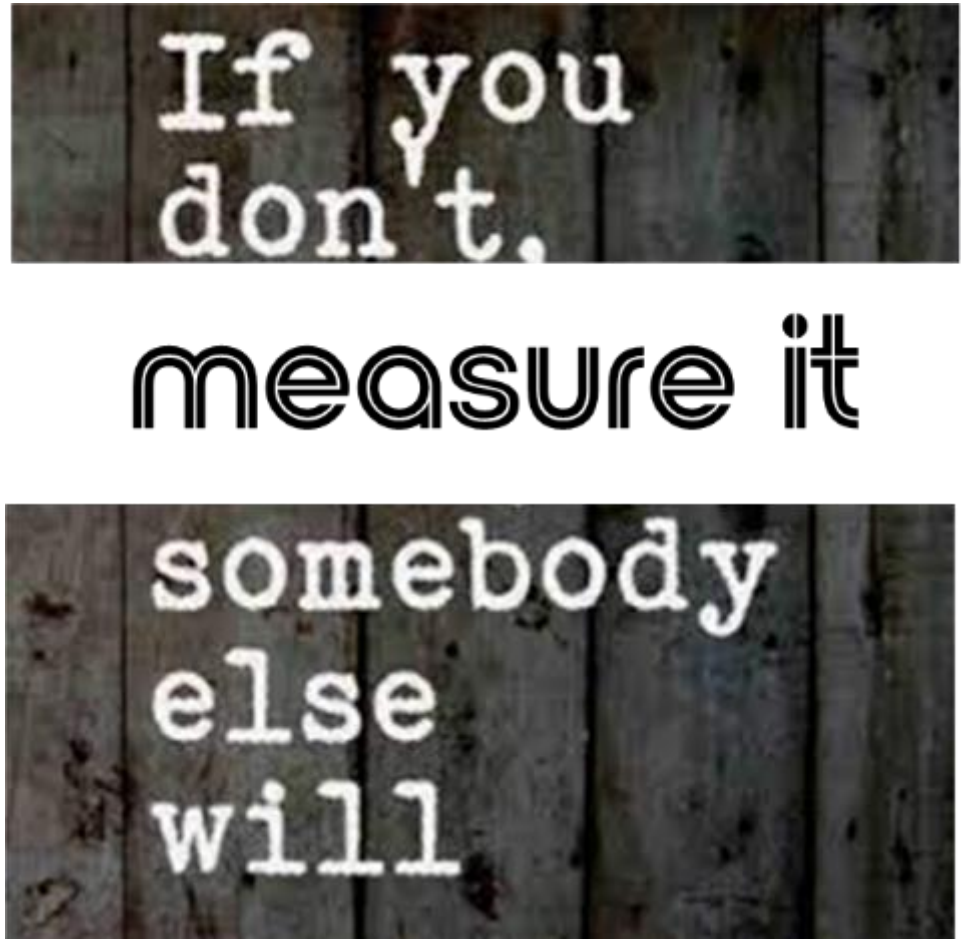
Practice 3:
Useful Measurements

Bounce SECURITY

Measuring Performance

- Need to know how we are doing
- Need to be able to demonstrate that upwards
- Because....

Measuring Performance



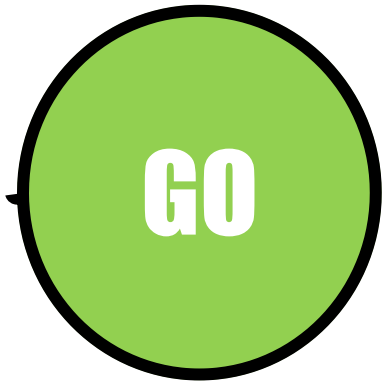
Measurement Types

- How do you know:
  - Is an activity taking place?
  - What are the results of the activity?
- Comparable metrics between teams/groups
- Between the team and itself over time

Common Anti-Patterns

- Not tracking whether activities are being performed
- Providing inaccurate / unadjusted numbers
- Manual collection of measurements
- Focusing on wrong technical metrics
- Short-term view of results with unreasonable expectations

Suggested Actions



- Every activity defined with metrics
  - *Is the activity being performed*
  - *Is the activity successful / valuable*
  - *Practical ways of tracking output*
- Consider what qualitative measures are also needed
- Automate metric collection wherever possible

Practice 4:
Useful Tools Where Appropriate

Bounce SECURITY

Variety of Tool Types

- SAST, SCA, DAST
- IAST, RASP, OAST, MAST
- Secrets scanning, container scanning, IaC scanning
- ASPM, CSPM, ASOC
- Etc…

Practicalities

- These tools detect a variety of vulnerabilities
  - *or some other risks*
- Some form of automation is needed to "force multiply“
- Lots of low hanging fruit
- Breadth vs depth
- Regulation may require certain processes/tools

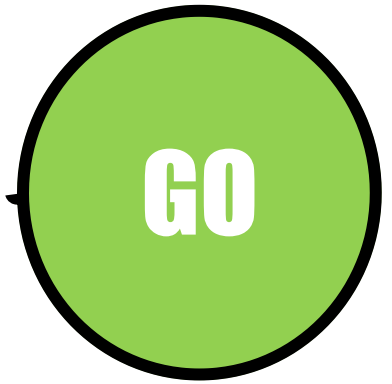
Key Challenges

- Easy to become overwhelmed
- Not every tool is useful in every case
- Most tools are mostly generic
- Tools have their own time cost
- The myth of automation

Common Anti-Patterns

- Tools become the SSDLC / AppSec programme
- Tool frustration causes negative perspective on security
- More time spent on false positives than on true positives
- Assumption that tool automation eliminates the need for manual effort

Suggested Actions



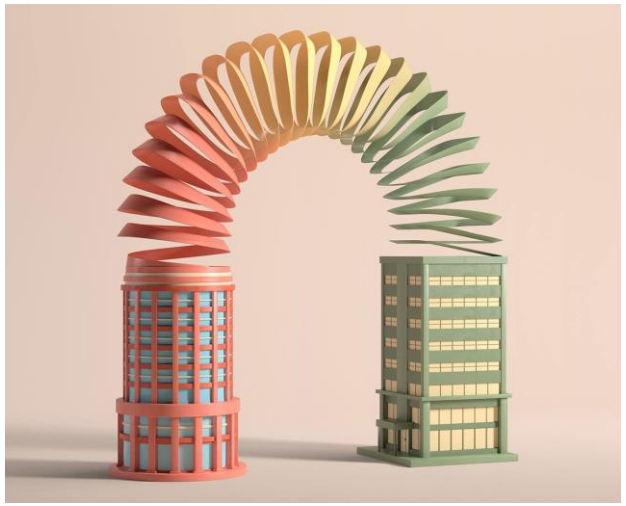
- Focus on process, supported by tools
- Allocate the manual work across the organization
- Take a gradual approach
  - *implementation, integration, configuration*
- Focus on signal over noise
- Make sure tools become a force multiplier
  - *And not a broken fence to work around*

Practice 5:
Standardize on Goals,
Be Flexible on Implementation

Bounce SECURITY

Practicalities

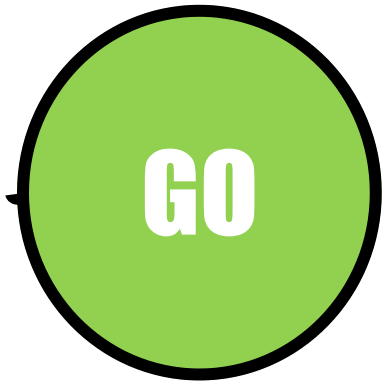
- Large organizations will have many different teams
- Each team will have different
  - *Tech*
  - *Workflow*
  - *Platforms*
  - *Culture*
- Security activities need to fit the team, but still standardized



Common Anti-Patterns

- One size fits all SSDLC
- Defined as top-down policy
- Varied and different teams forced to work the same
- Ignoring different team structure, tech stack, platforms, technical workflows, business constraints…

Suggested Actions



- Define requirements, not implementation
- Allow different teams to adapt to their existing workflow and platforms
- Abstract the measurements to support commonalities

Summary

SSDLC – Summary

- SSDLC can help us spread security across the development process
- Incremental approach of gradual improvement
- Ownership and metrics are key to success

Bounce SECURITY

THANKS FOR YOUR ATTENTION!

Avi Douglen
Bounce Security
@sec_tigger

Bounce SECURITY