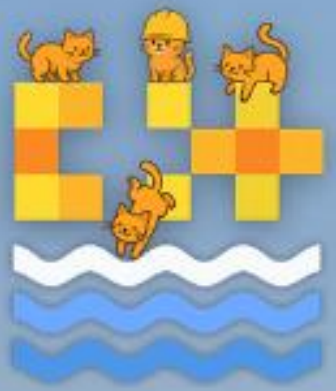


Software Engineering Completeness Pyramid

Knowing When You Are Done and Why It Matters

Peter Muldoon

2025

Software Engineering Completeness

Knowing when you are Done and why it matters

Engineering

Bloomberg

C++OnSea
June 23rd, 2025

Peter Muldoon
Senior Engineering Lead, Ticker Plant

TechAtBloomberg.com

Who Am I

- Starting using C++ professionally in 1991
- Professional Career
  - Systems Analyst & Architect
  - 21 years as a consultant
  - Bloomberg Ticker Plant Engineering Lead
- Conference talks focused on practical Software Engineering
  - Based in the real world
  - Take something away and be able to use it

Why?

What does "*Done*" mean?

Dictionary Definition:

Done: to work on (something), to bring it to *completion* or to a *required state*.

What does "*Done*" mean for Engineering?

The Scrum Guide says the definition of "*Done"* is a **formal description of the state of the Increment when it meets the quality measures required** for the product.

The definition of "*Done"* is a formal description of your quality standards.

Why bother?

The benefits of establishing a definition of **“Done”** include **creating a shared understanding and unified language for software delivery**, ensuring that new employees have **access to tribal knowledge and process expectations**

A proper definition of “**Done**” across an organization acknowledges the shared responsibility **and** helps a software organization maintain alignment on projects/deliverables

Example

The goal of your project is to develop 10 new product features and deploy them to users. But you haven't deployed any of them yet.

Your completion percentage is 0/10 = 0%.

(This is where a manager/developer will say: "But we've coded and tested five of them; 50% of them are done!")

Better Question would be:
When do we expect < something specific > to be deployed and active everywhere in Production aka "***done***"**?**

Basic Terminology

What is the *Business Value* of Software Engineering?

Delivering desired product outcomes in incremental steps

Why incremental steps ?

- Shorter time horizons
- Lower risk
- Better feedback from customers

< Cadence and delivery of these steps is important to know>

Basic Terminology

What is the *Business Value* of Software Engineering?

Software Value is actualized when it's

- Available
- Usable
- Reliable

Software Value (future looking)

- Configurable
- Flexible
- Fix issues quickly
- Evolve quickly

Production Changes

What types of improvement are delivered to Production?

- New Features
- Bug fixes
- Feature Flag changes
- Configuration
- Technical Debt reduction
  - ❑ Refactoring
  - ❑ Deprecation
- Environment / Infrastructure
  - ❑ New compiler version
  - ❑ Machine architecture

Production Changes

What other types of change are delivered to Production?

- Broken functionality
- Missing functionality
- Performance issues
- Security issues
- System unreliability

Production Changes

What’s in a commitment to deliver change?

Know what are you delivering

What are Acceptance criteria (AC)?

Acceptance criteria are the conditions a software product must meet to be accepted by the user

Good acceptance criteria should possess:

- **Clarity:** Straightforward and easy to understand for all team members and the *customer*
- **Conciseness:** The criteria should communicate the necessary information without unnecessary or vague detail to determine success
- **Testability:** Each criterion must be verifiable and clearly determined whether it has been met
- **Observable:** The focus should be on delivering results visible to the customer

Note: AC describe *what* the change will do *not* the *how*

Know what are you delivering

Writing Acceptance criteria (AC)

- Written by you/product owner
- Shared with the team
- Reviewed and validated with the ***customer/proxy***

Can use the Given/When/Then Gherkin style

Scenario: Reference machines in bad health swapped out of QA daily comparison testing
Given: Scheduling daily suite of comparison testing between a test and reference machine
When: A reference machine has either a BUILDING tag, Non-standard software or a bad environment
Then: Replace reference machine used with a machine from the backup list
And: Repeat this check

Requirements and Scope?

What is Project scope?

Project Scope refers to the complete list of features or deliverables of a project
It can also specify items that are out of scope

These deliverables are created using the requirements of the project.

Scope Creep: Expanding the list of features or deliverables from those originally agreed

Requirements: Specify what the change should do in Unambiguous and bounded terms

Timescale Specificity?

When will the change/bugfix be "*done*"?

- Whenever, in the hands of the gods …
- Soon, a wee while …..
- Couple of weeks, a month or so ….
- Actual date: 21 Sep 2024

Most engineers have misplaced optimism on timelines
Assumes a best case scenario and (usually) are using a poor definition of "*done*"

Question often forgotten: When will <Change> start!

*Timescale estimation clarity is affected by domain type

Good Time Estimates?

Good estimates create trust and political capital with your customers and other teams/management

How to give better estimates:

- Break changes down into smaller pieces
- Have specific requirements / scope
- Embrace empirical reality
- Lean towards contingency fund for delays (~10%)
- As time progresses, uncertainty lessens
  - ❑ Communicate this better information to stakeholders

Are we Done?

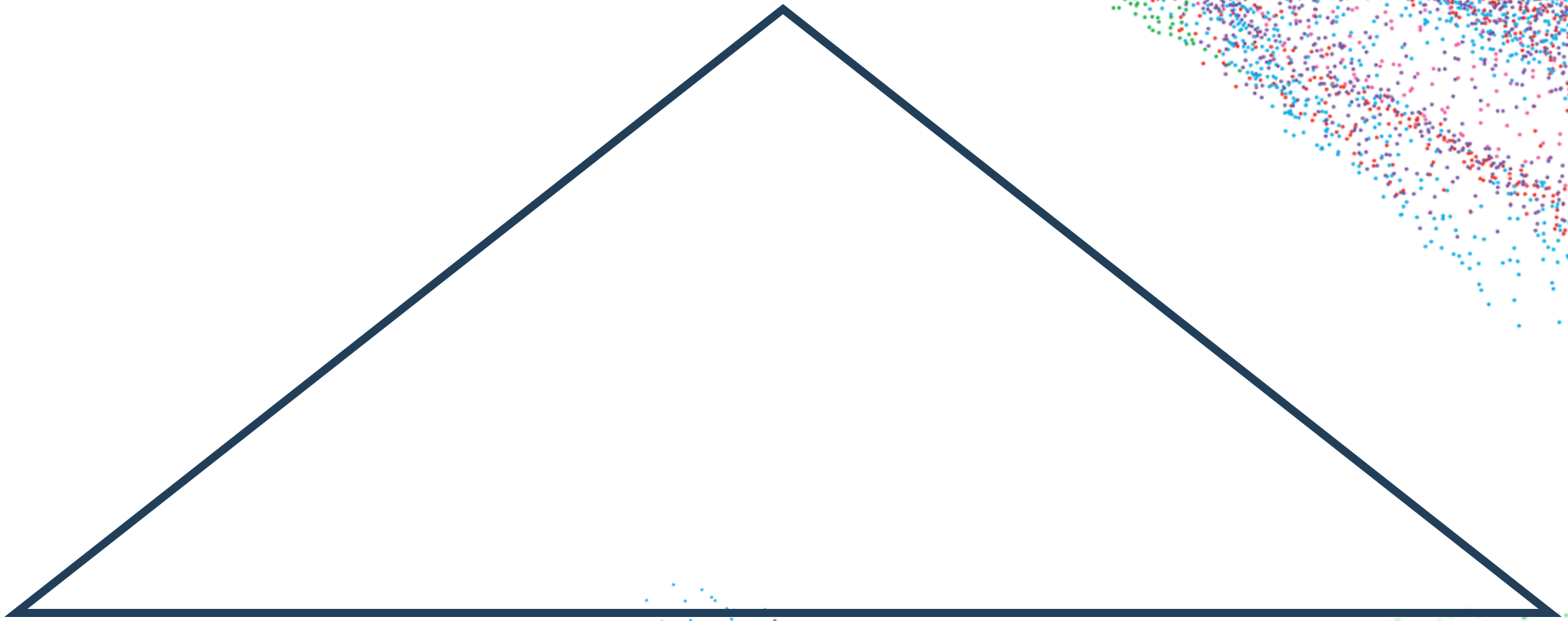
Forewarning:

School of thought : Never commit to anything concrete or easily measurable aka *be slippery*

Being vague, lack of preciseness, more about providing cover for potential failure rather than achieving positive outcomes

Let's begin ….

Software Engineering Pyramid



Development Done?

Have the changes been verified and applied?

- Met the change Acceptance Criteria?
  - ❑ Fully vs partially
- Passed all testing driven by the validation system?
  - ❑ Unit tests/integration all passed
  - ❑ Added tests for a new change
- Passed **code review** and been committed into the repository?
  - ❑ Executed in a structured sane manner
- Merged into a package for Production release?
  - ❑ Ready for deployment

Is the change (in behavior) code complete?

Deployment Done?

Are the changes deployed everywhere?

- What is the pace of deployment through Production stages
- When is the code deployed ***Everywhere***
- Any staggered dependencies needing tracking
- Any code freezes imminent

Feature Flags

What is a Feature Flag?

Feature flags are a software development tool that allow you, at runtime, to enable or disable a change without modifying the source code or requiring a rollback/redeploy.

Safety based if-statements are placed in the code base that act as circuit breakers for "*untested*"[1] code

Disables a single change which obviates the need for system rollbacks which would affect multiple unrelated changes

1: Untested in production

Feature Flag Enablement Done?

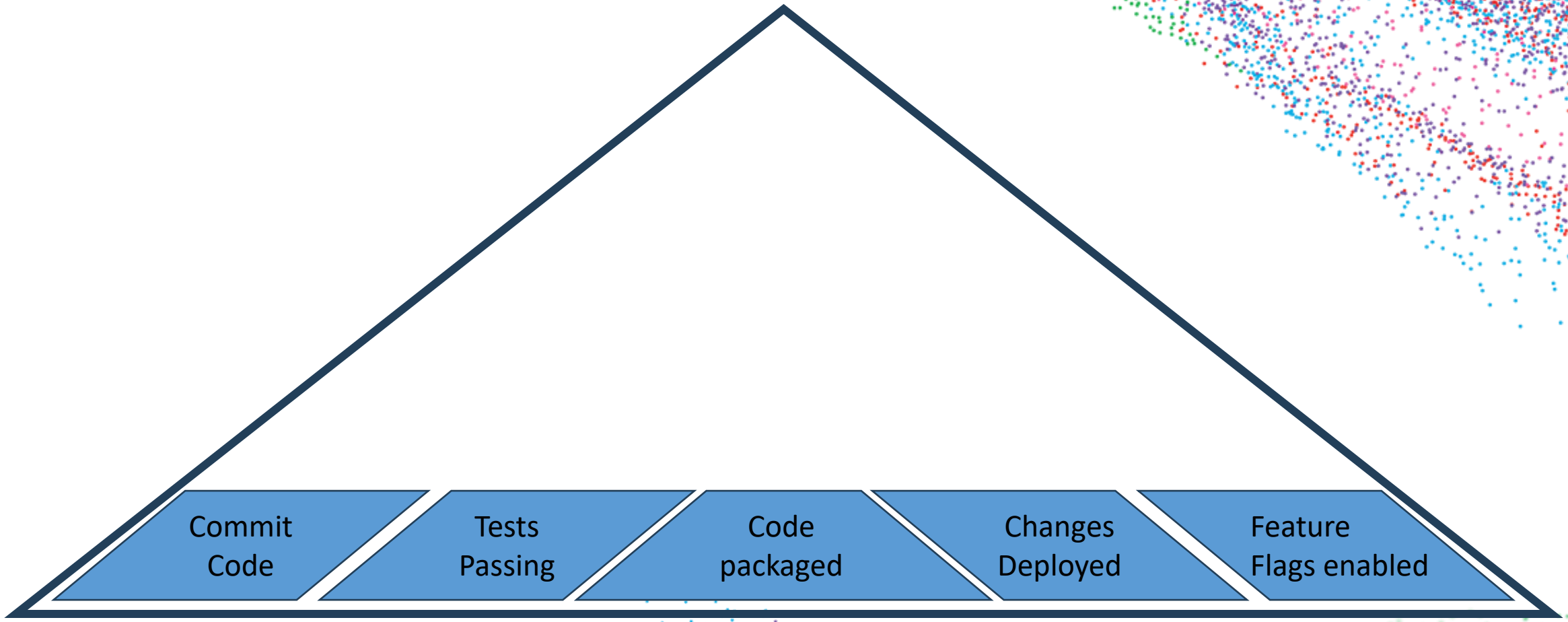
Deployed code that's not being utilized is not useful (yet)

Are Feature flags enabled everywhere in Production?

- What is the pace of enablement?
- When is enablement complete?
- *When is the feature flag being removed?

*Dealt with in later section

Software Engineering Pyramid



Survival Achieved

Congratulations, you're a competent hacker

Rationale!

Why is **survival** not enough?

Any system where engineering is invested completely in feature change/bug fixing, that system will devolve, over time, into a complex brittle codebase.

Efforts are needed to stabilize/reverse the entropy in the code base.

Code Health Basics

What is software decommissioning?

Decommissioning is the ***strategic*** process of retiring outdated software and related infrastructure, to streamline and enhance overall maintainability/efficiency.

Two categories

- Code that is structurally never abled to be accessed
  - ❑ Actual functionality that is never called
  - ❑ Find with static analysis tools – Coverity, cppcheck, IDE
- Code that is never accessed with current user input
  - ❑ Monitor usage in production over a “length of time”
  - ❑ Feature flag removal

Decommissioning done?

Any decommissioning needed?

- If replacing something, have we planned for the removal/decommissioning of the older functionality
- Feature flag removal?
  - ❑ Eliminate dead branching of code

Code Health Basics

What is software refactoring?

Refactoring is the ***disciplined*** process of changing a system's software in such a way that it does not alter the function of the code yet improves its internal structure and/or efficiency.

Why is refactoring needed?

Tactical => Strategic change/implementation

- ❑ Time challenges
- ❑ Refactoring time not budgeted
- ❑ Inadequate time to research
- ❑ ***Poor code reviews***

Note: No refactoring, over time, will lead inevitably to a system rewrite

Refactoring done?

Any Refactoring needed due to :

- Recurring / Duplicated Patterns in code
- Low readability/maintainability code
  - ❑ Code smells
  - ❑ Overly complicated / redundant logic
  - ❑ Not using standard components
- Paradigm changes
- Technical depreciation
  - ❑ Code ages
  - ❑ Technological advancements
  - ❑ More efficient alternatives

Code Health Basics

What is Technical Debt?

Technical Debt: Cost of working/fixing code in the future due to limited quick solutions (shortcuts) implemented now

Fomented by

- Deadlines
- Firefighting
- Cleverness / premature optimization
- Lack of skills / seasoning / poor culture
- Lack of standards / **poor code reviews**
- Organic growth
- Poor documentation

* Over a span of time

Tech Debt Basics

“There is nothing so permanent as a temporary decision”
- Knapton

Categories of Technical Debt:

- Intentional
  - ❑ Taken on consciously for strategic reasons
  - ❑ Items placed on backlog to mitigate
- Unintentional
  - ❑ The non-strategic result of doing a poor/sloppy job
  - ❑ No plan to mitigate

“A project isn’t done until you go back and adjust whatever it was you took on as technical debt; *and everybody agrees this is how we define ‘done,’*”
- Knapton

Tech Debt Basics

Tracking Tech Debt:

- Make a list with key attributes
  - ❑ Effort Size
  - ❑ Severity
- Be intentional
  - ❑ Add to list when suboptimal solutions used
- Keep visible
  - ❑ Prioritize on roadmaps
  - ❑ Tech Debt sprints
- Advocate/Champion for improving the codebase
  - ❑ Highlight business risks
  - ❑ Time to market
  - ❑ Maintenance load

maintenance load: How much time and effort are developers spending on tasks that *are not* adding features or removing features?

Tech Debt Done

Tech Debt prevention :

- Proper timeline planning
- Design reviews
- **Code review** guidelines
- Coding standards
- Tracking
  - ❑ Champion its demise
- Training

Testing Done?

All Testing Pillars accounted for?

- Unit
- Integration
- System end-to-end (QA)

Maintain quality through testing

Testing Done?

When we find problems in functionality, are we leveraging the experience?

- Create tests based on production problems (for next time)
- Adequate test coverage for scenarios - BDD?
  - ❑ Identified corner cases
  - ❑ Test unhappy paths

* Automation of testing – with alarms - is key

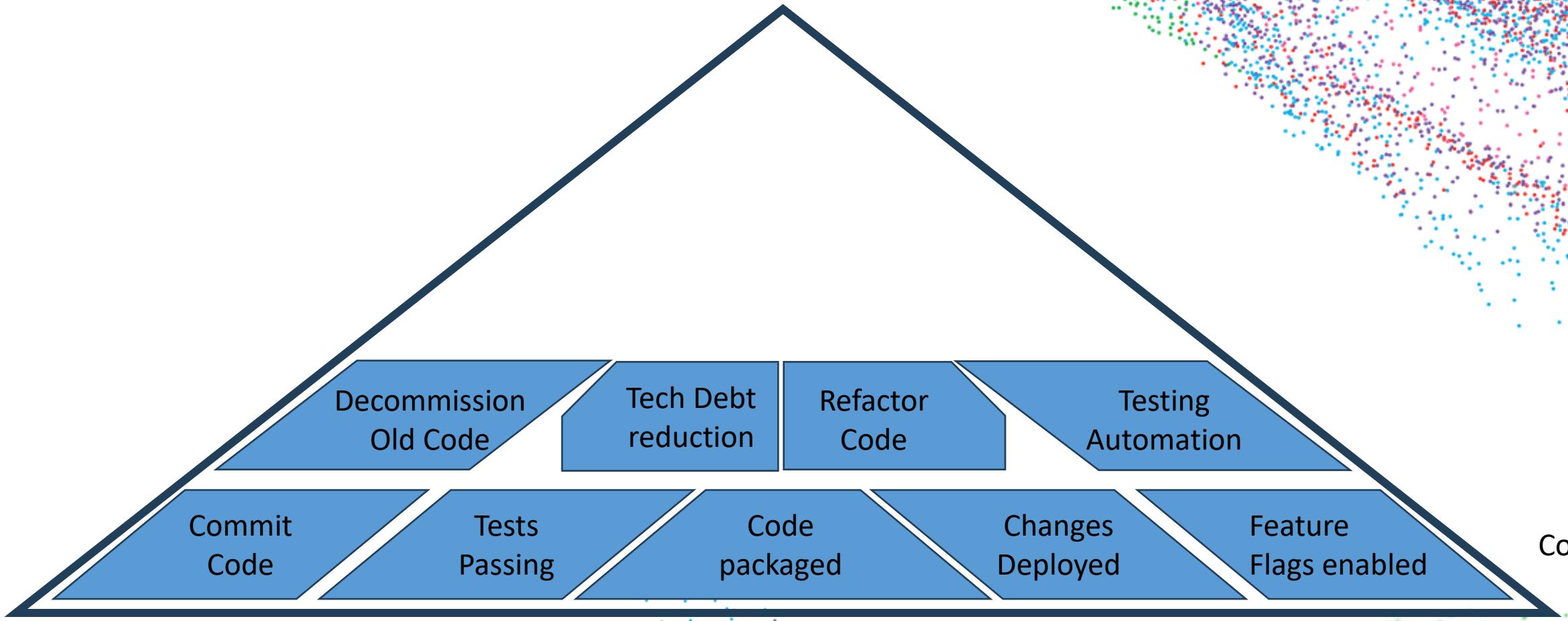
Rationale!

Why is **survival** not enough?

Any system where engineering is invested completely in feature change/bug fixing will devolve, over time, into a complex brittle codebase.

Efforts are needed to stabilize/reverse the entropy in the code base.

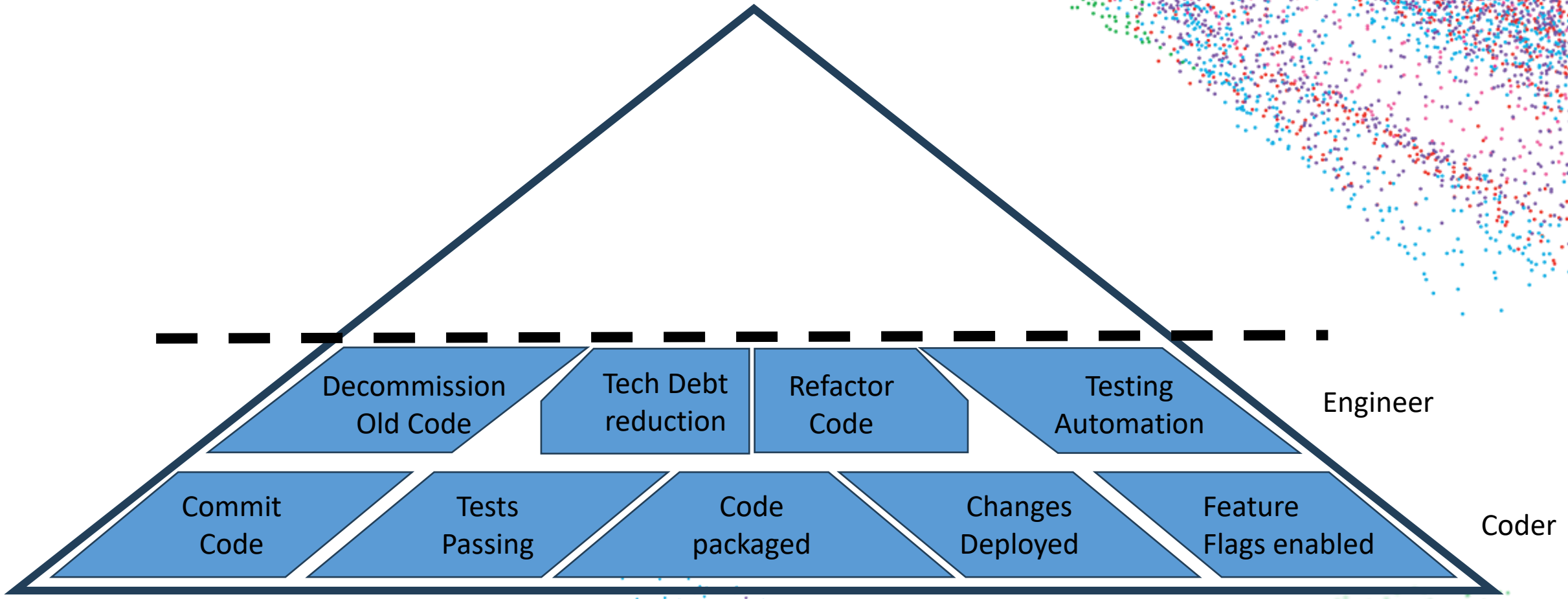
Software Engineering Pyramid



Sustainability Achieved

Congratulations, you're a competent engineer

Software Engineering Pyramid



Change of Emphasis

Probably not!

System Reliability & Resiliency

What is system reliability ?

System performs its intended function correctly with no failures or downtime

What is system resilency ?

The ability of the system to degrade gracefully in times of stress as opposed to cease to function altogether

System Reliability & Resilency

Why do we care ?

Impacts on:

- User trust
- Business continuity
- Brand reputation

System Reliability & Resiliency

How healthy is your system?

Are changes impacting your systems' (aggregate) health?

Can you catch problems before your clients notice?

System Reliability & Resiliency

Do you have System health monitoring?

Operational Metrics

- Latency – Time taken to service a request (response time)
- Traffic/Throughput - How much stress is the system taking, at a given time, from users or transactions processing through the service
  - ❑ e.g. How is latency affected by throughput (requests per minute)
- Saturation – overall capacity/utilization of the service (%CPU/RAM/disk net free, queue depths)
- Errors – Rate of failing requests to total requests – assuming requests are well-formed

Any other metrics tailored for your system

System Reliability & Resiliency

Using System health monitoring:

- Automatic alarms
  - ❑ Avoid “eyes on glass”
  - ❑ Imminent outages
- Monitoring Trends over spans of time
  - ❑ Capacity planning

Production Support

Any new technology/features (to you) being introduced?

- Supported by current operations
  - ❑ Or supported by developers
- Effective distributed triage
  - ❑ Widely known / not a singular person
  - ❑ Effective logging & observability
  - ❑ Support ongoing runbooks

Improvement Planning

Improvement planning : Identifying areas of risk that need fixing or optimization

Broad scale functionality changes require:

- Multiple stages / iterations needed
  - ❑ Complicated problems
  - ❑ Mitigate risk to current production
- What are predicted timelines/effort for this?
  - ❑ Mid level time horizons so less certainty
  - ❑ Non-trivial releases will need fixes/features added
- Many external dependencies
  - ❑ Co-ordination and communication become critical and time consuming

Architectural Design

Strategic local re-engineering

- Broad complex changes in system architecture
- Vision for managing future performance
- Where to target the system for most benefit
- Leveraging existing technologies or introducing new ones

Strategic Vision

***Strategic* vision**: Decisions that are long-term and influence future direction and are beyond just the technical details.

Dealing with:

- Business shifts
- Technology shifts
- Future-proofing

Usually weighing trade-offs for a number of solution

Strategic Vision

Strategic Planning and decisions - recognition

- Many people/stakeholders are involved
  - ❑ More people, more strategic
- Time spans for decisions are long
  - ❑ Longer time horizons, more strategic
- Longer-term outcomes
  - ❑ Not easy to change

Rationale!

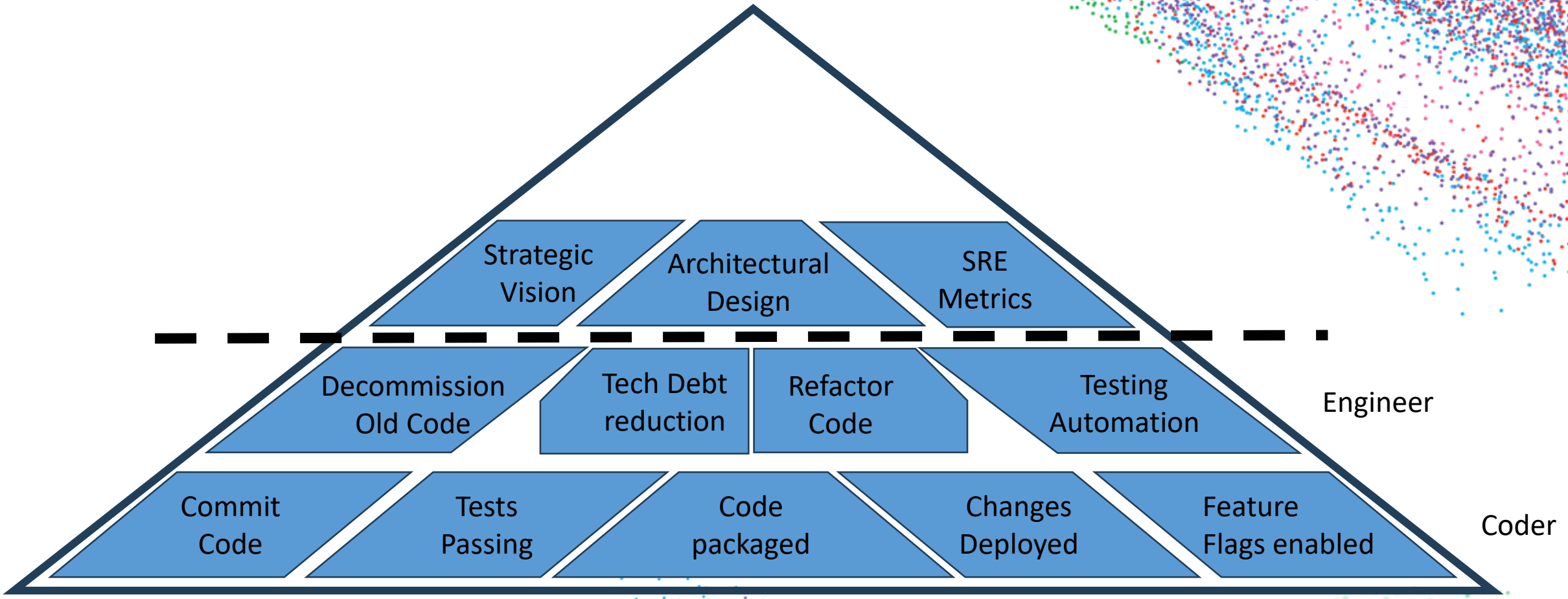
Why is **sustainability** not enough?

Sustainability means to maintain the current state of the code. The Code (development) itself does not live in a vacuum but in an eco-system.

Efforts are needed for future planning in the areas of **manual toil reduction, capacity planning and general observability within the system**.

Further efforts are required to construct a roadmap of where do we want to go in the long term

Software Engineering Pyramid



System Reliability Achieved

Congratulations, you're a competent Systems Engineer

Rationale!

Why is overall **system reliability & resiliency** not enough?

Engineering must align with the business' needs.

Efforts are needed to communicate clearly and bidirectionally

- Engineering understands and aligns with the Business needs
- Business understands Engineering constraints & capabilities

Roadmaps

What is a Roadmap meeting?

A formal meeting where a group assembles to discuss the current progress and future direction of a product or project

This is to ensure everyone is striving collaboratively towards shared objectives

Business/Stakeholder involvement

Roadmap meeting composition (Roles):

- Product owner
  - ❑ Create your roadmap and present all your strategic goals
- Business analyst/proxy
  - ❑ Explain user engagement and business goals
- Development representatives from each major area involved
  - ❑ Explain roadblocks, effort and timelines
  - ❑ Flag risky shortcuts
- Executive stakeholder
  - ❑ When you need approval in your decision-making process
- Product manager / Delivery specialist
  - ❑ Long-range timelines

Business/Stakeholder involvement

Roadmap meeting addresses:

- Reviewing the Current Landscape
  - ❑ Presenting progress since last meeting
  - ❑ Listing any roadblocks and current status of each
- Setting Strategic Direction
  - ❑ Where we are going?
  - ❑ How we are getting there?
  - ❑ Gaining consensus/buy-in
- Prioritization and Resource Allocation
  - ❑ In what order are items getting tackled?
  - ❑ By Whom?
- Action Planning and Collaboration
  - ❑ Actionable items with clear success criteria and dates attached
  - ❑ Review long-term timelines

Future planning Done?

Are you prepared for:

- Software stability?
- Security vulnerabilities?
- System scalability?

Are you ready for:

- New technical opportunities?
- New business opportunities?
- New regulation/restrictions?
- Staying competitive?

Rationale!

Why is overall **system reliability & resiliency** not enough?

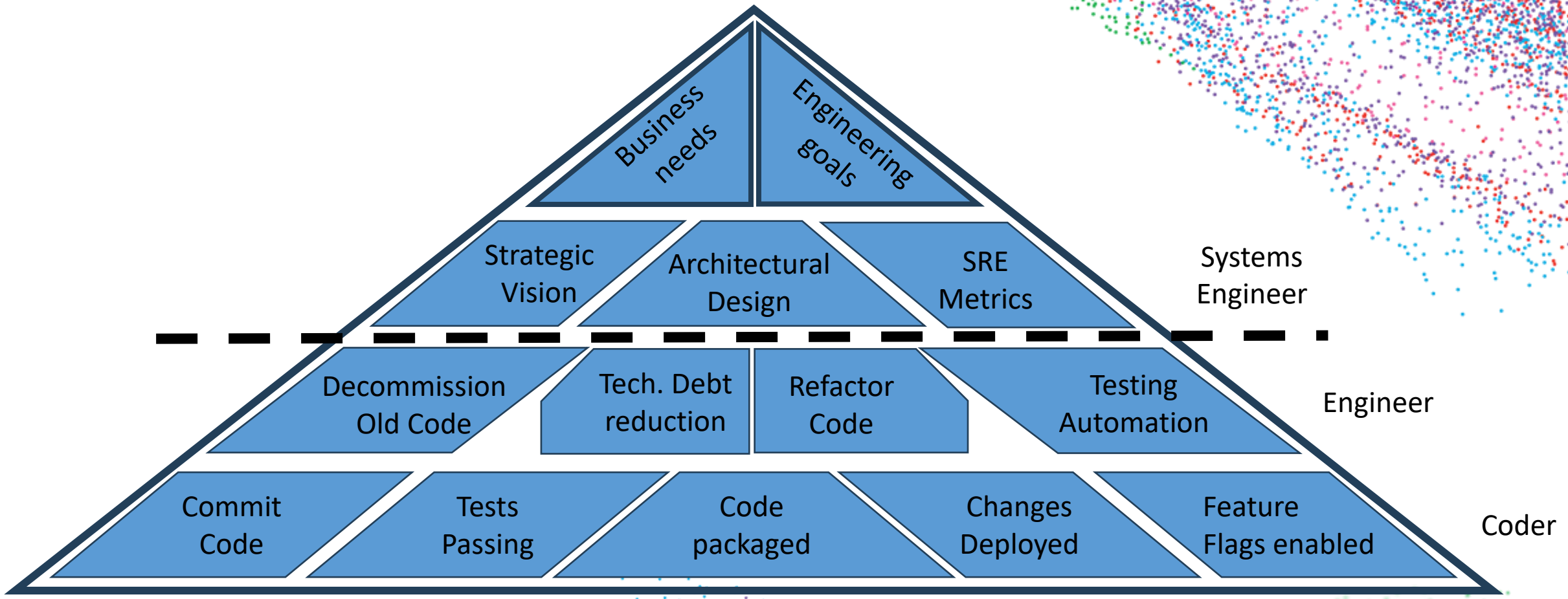
Engineering must align with the business' needs.

Efforts are needed to communicate clearly and bidirectionally

- Engineering understands and aligns with the Business needs
- Business understands Engineering constraints & capabilities

Because Engineering and Business need to collaborate, compromise, and effectively communicate

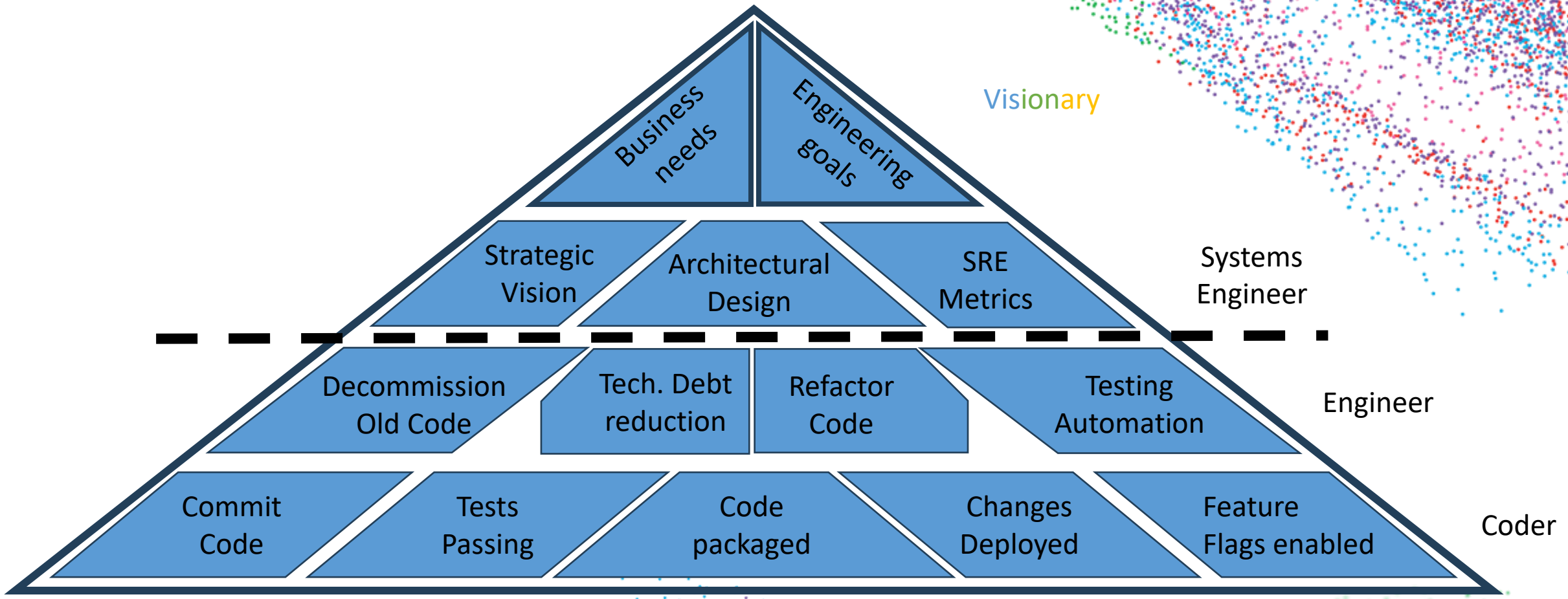
Software Engineering Pyramid



Engineering & Business Alignment Achieved

Congratulations, you're a Visionary

Software Engineering Pyramid



Recap

Software Engineering: What level are you operating at?

- Future direction and market competitiveness
- System aggregate health and stability
- Code base health and sustainability
- Feature Implementation / Bug fix

Recap

Software Engineering: When are you "*Done*"?

What type of "*Doneness*"?

- Be Specific on
  - ❑ Scope
    - ❑ Milestones and deliverables
    - ❑ Acceptance Criteria
    - ❑ Any follow-on work needed?
  - ❑ Time
    - ❑ Deliverable dates
    - ❑ Milestone dates
- Intentional Tech Debt
  - ❑ Remediation required
- Communicate clearly to stakeholders
  - ❑ Regular progress meetings/roadmaps

Recap

Software Engineering: When are you "*Done*"?

- Feature/Bug-fix/Change is running smoothly and engaged **everywhere** in Production
- Well supported by the team
- Tech Debt is decreasing and not increasing
- Structured maintainable Code
  - ❑ Happy developers
  - ❑ Retained engineers
- Users/Clients are happy
  - ❑ Then, so is the business
- Poised to meet future challenges

And Finally:

This presentation is well and truly

Done

Thank You

Other Engineering Talks:

Retiring The Singleton Pattern: Concrete Suggestions on What to Use Instead
Redesigning Legacy Systems: Keys to success
Managing External APIs in Enterprise Systems
Exceptions in C++: Better Design Through Analysis of Real World Usage
Dependency Injection in C++: A Practical Guide
Mastering the Code Review Process : Boosting Code Quality in your Organisation

Questions?

Contact: pmuldoon1@Bloomberg.net