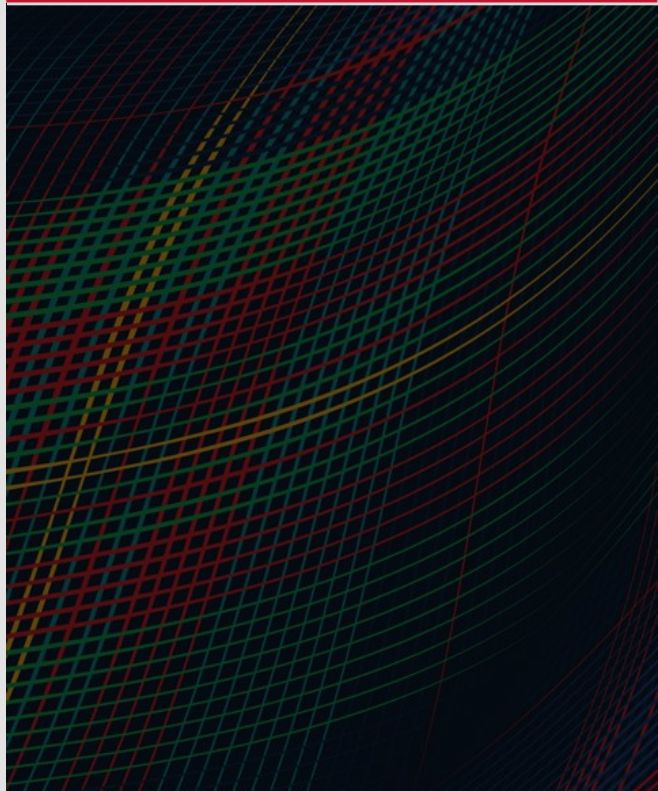
Carnegie Mellon University
Software Engineering Institute

Agile & Requirements in Government Settings



Module 4:

March 2024

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM24-0204

Topics the SEI will address in this course include:

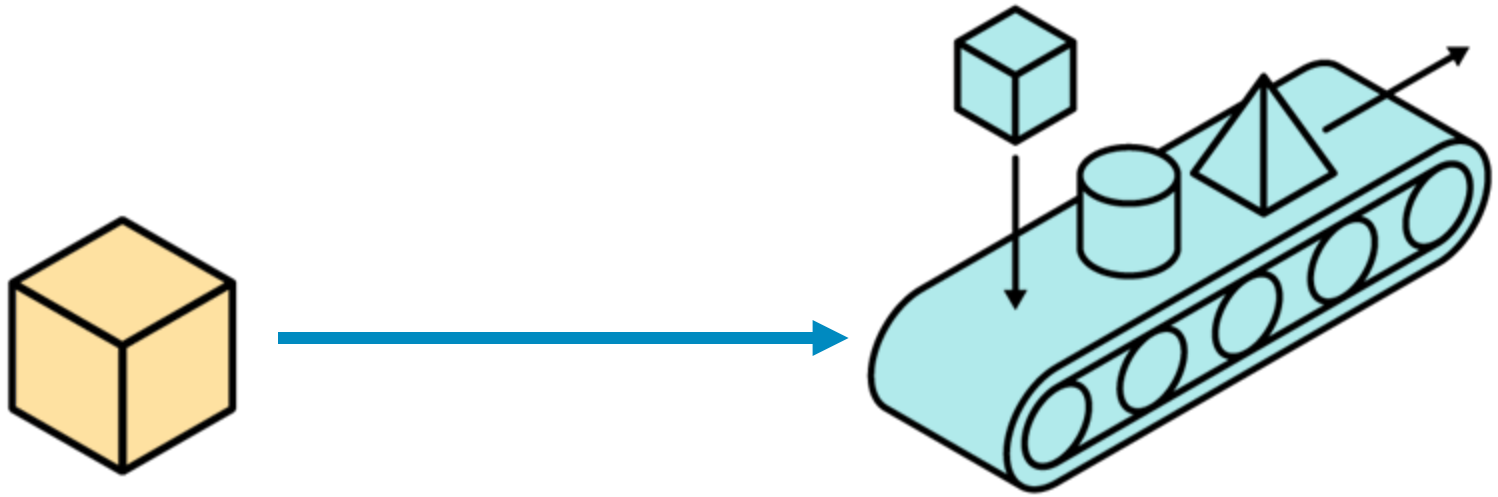
- Transitioning to Agile Requirements
- Team Level Agile Requirements Model
- Scaling the Agile Requirements Model
- Scaled Agile (SAFe) Requirements Model
- Backlog Priority and Maintenance
- Acceptance Criteria or Definition of Done
- Writing, Splitting and Accepting User Stories

e optional

Transitioning to Agile Requirements

Carnegie Mellon University
Software Engineering Institute

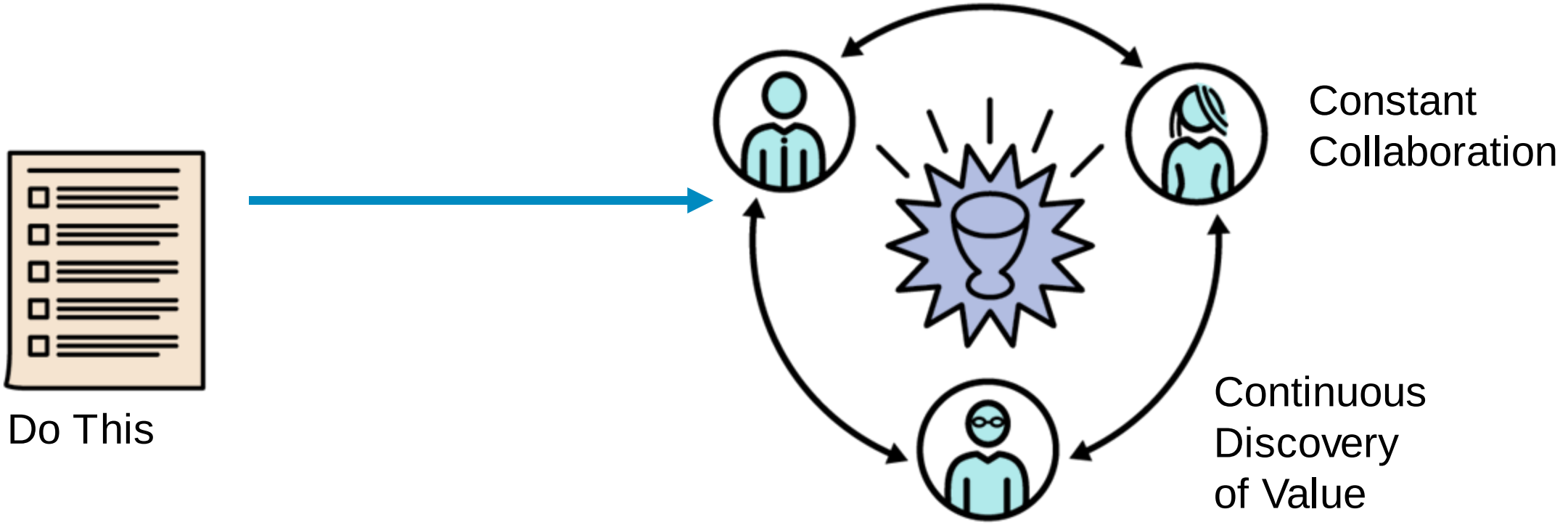
Agile Represents a Major Requirements Transition – Different Acquisition Objectives



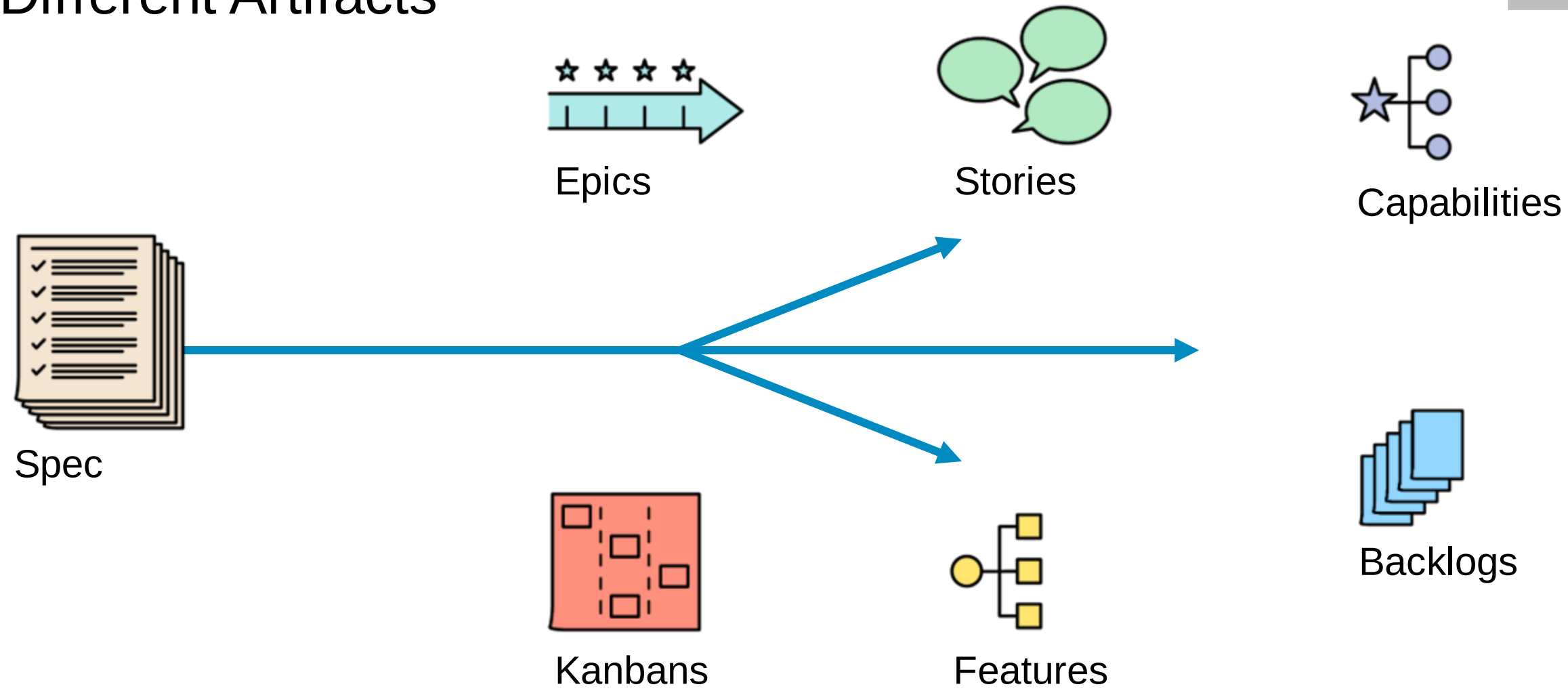
Buying a Box

Buying an ongoing delivery stream

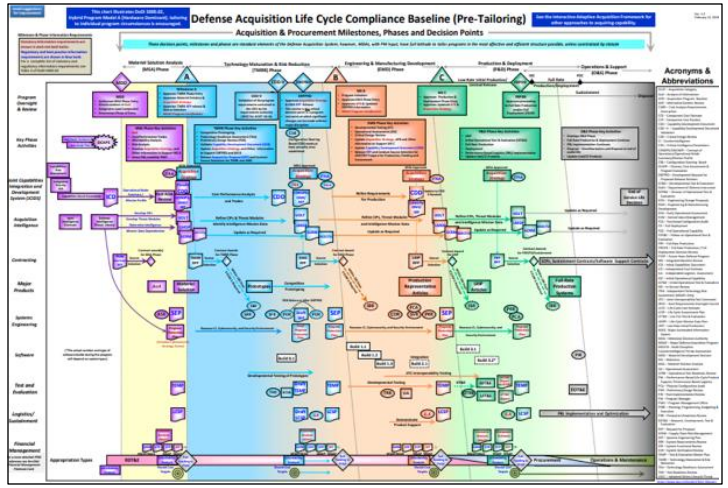
Agile Represents a Major Requirements Transition – Different Vendor Interactions



Agile Represents a Major Requirements Transition – Different Artifacts



Agile Represents a Major Requirements Transition – Different Practices for Requirements Management



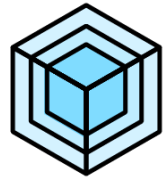
Date-based Milestones

Increment-Based Demos

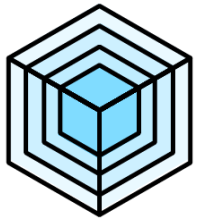
Definition of Done

Team Increment

System Increment

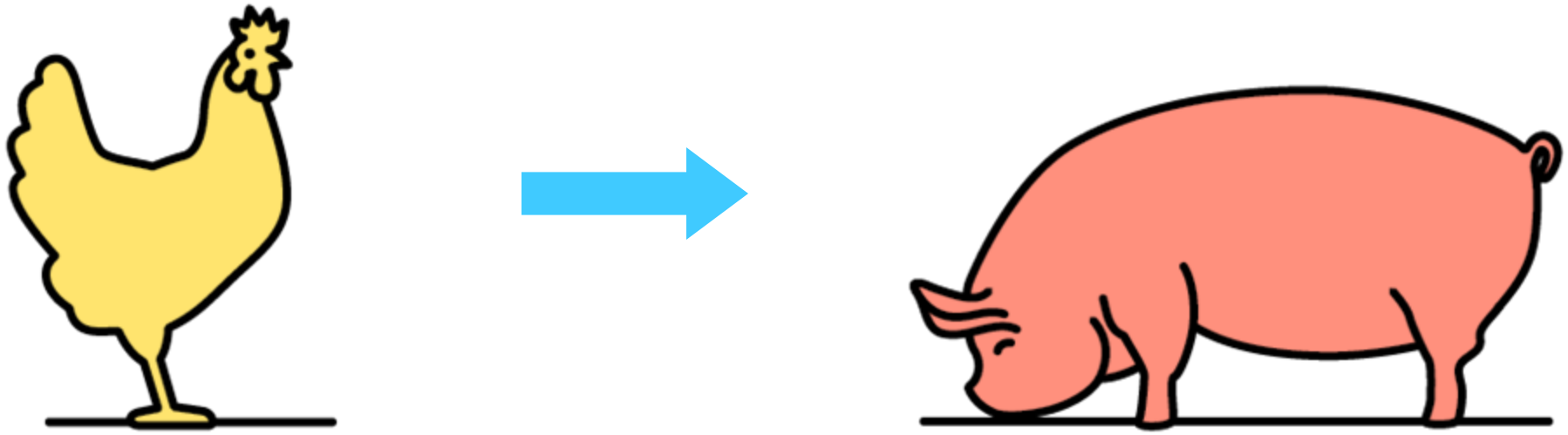


Solution Increment



Release

Agile Represents a Major Requirements Transition – Different Overall Risk Profile

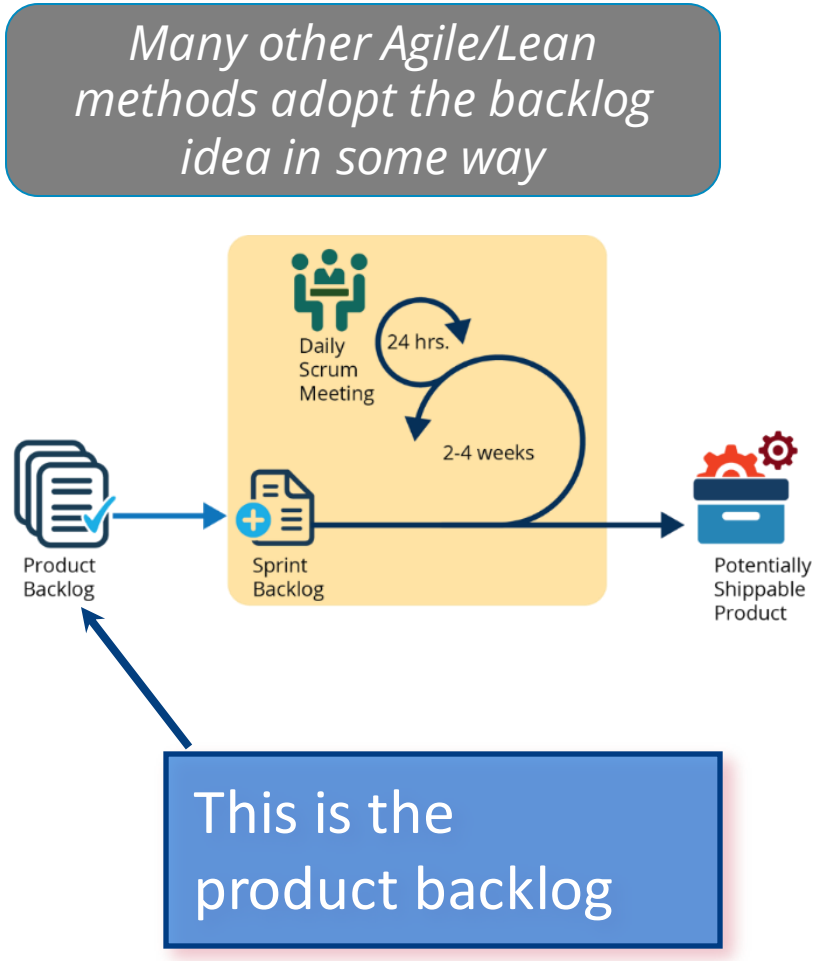


Carnegie Mellon University
Software Engineering Institute

Team Level Agile Requirements Model

(KTR-focused unless Govt is actually doing development)

Product Backlog: Scrum's Way of Organizing and Prioritizing Requirements



- A list of all desired work on the project
- Ideally expressed such that each item has value to the users or customers of the product
- Prioritized by the *product owner* (represents the business/acquirer)
- Reprioritized at the start of each release and each sprint

Agile Motto for the Backlog Process

Scaling the Agile Requirements Model

Carnegie Mellon University
Software Engineering Institute

Characteristics of Requirements Models for Large-Scale Agile Systems

Carnegie Mellon University Software Engineering Institute

Layered - Requirements are divided into layers representing different levels of abstraction / scope

Cascading – Each layer of the Requirements Model feeds the layer below

Categorized – Two general types of requirements expressions – Business/Mission (user-focused) and Enablers (system/infrastructure-focused)

Prioritized—part of what makes scaling feasible is that we're not dealing with 100% of the details of all the requirements at the same time

Adapted from https://www.scaledagileframework.com/epic

Example of a Large Scale Agile Requirements Model

Area of Concern	Requirements Layer (SAFe-specific)	Development Timeframe / Work item Size (SAFe-specific)
Mission Value & Rationale (ConOps)	Portfolio Epics	"Epics typically do not need a traditional scope completion end-state. Instead work continues until the achievement of the optimum economic benefit"
Mission Deliverables	Capabilities	One Program Increment (approximately 10 weeks Dev & Test) across Multiple ARTs
Program Increment and Release Contents	Features	One Program Increment using a single ART
Agile Team Implementation	Stories	One sprint (approximately 2 to 4 weeks) using a single Agile team

https://www.scaledagileframework.com/epic

A Useful Construct for Non-functional Requirement: Enablers (1)

Not all development work will directly touch the user.
Some development activities instead provide “indirect” value to the product or the program.

- Evolving the Product Architecture
- Improving the Program Infrastructure
- Ensuring Compliance
- Exploring Technical Feasibility
- etc.

This type of development work is referred to in SAFe as “enablers”

https://www.scaledagileframework.com/Enablers/
https://www.scaledagileframework.com/story/

Enablers (2)

Carnegie Mellon University Software Engineering Institute

Enablers are stored in the backlog because …

If it ISN'T in the backlog, it WON'T get done

Enablers can be expressed at different levels of abstraction e.g.

- Epics
- Capabilities
- Features
- Stories

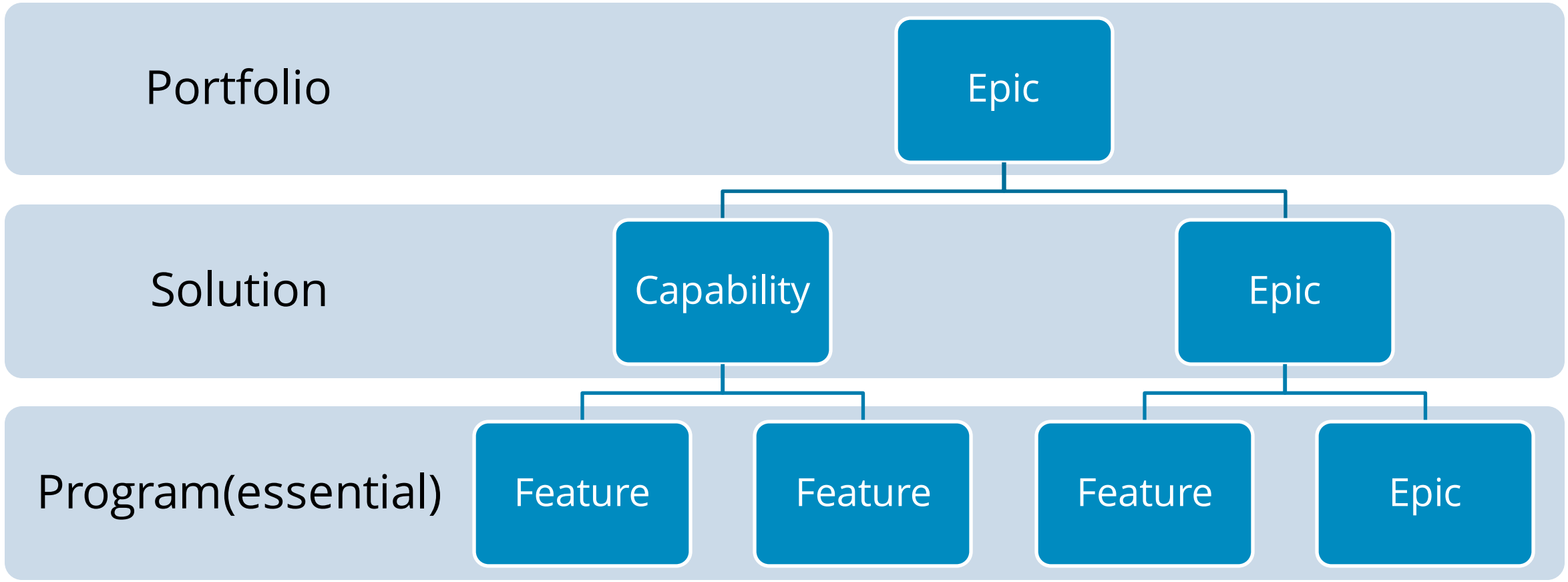
Usually expressed in technical language rather than using the user-centric story template

https://www.scaledagileframework.com/Enablers/
https://www.scaledagileframework.com/story/

Scaled Agile (SAFe) Requirements Model

Carnegie Mellon University
Software Engineering Institute

Agile Requirements Hierarchy

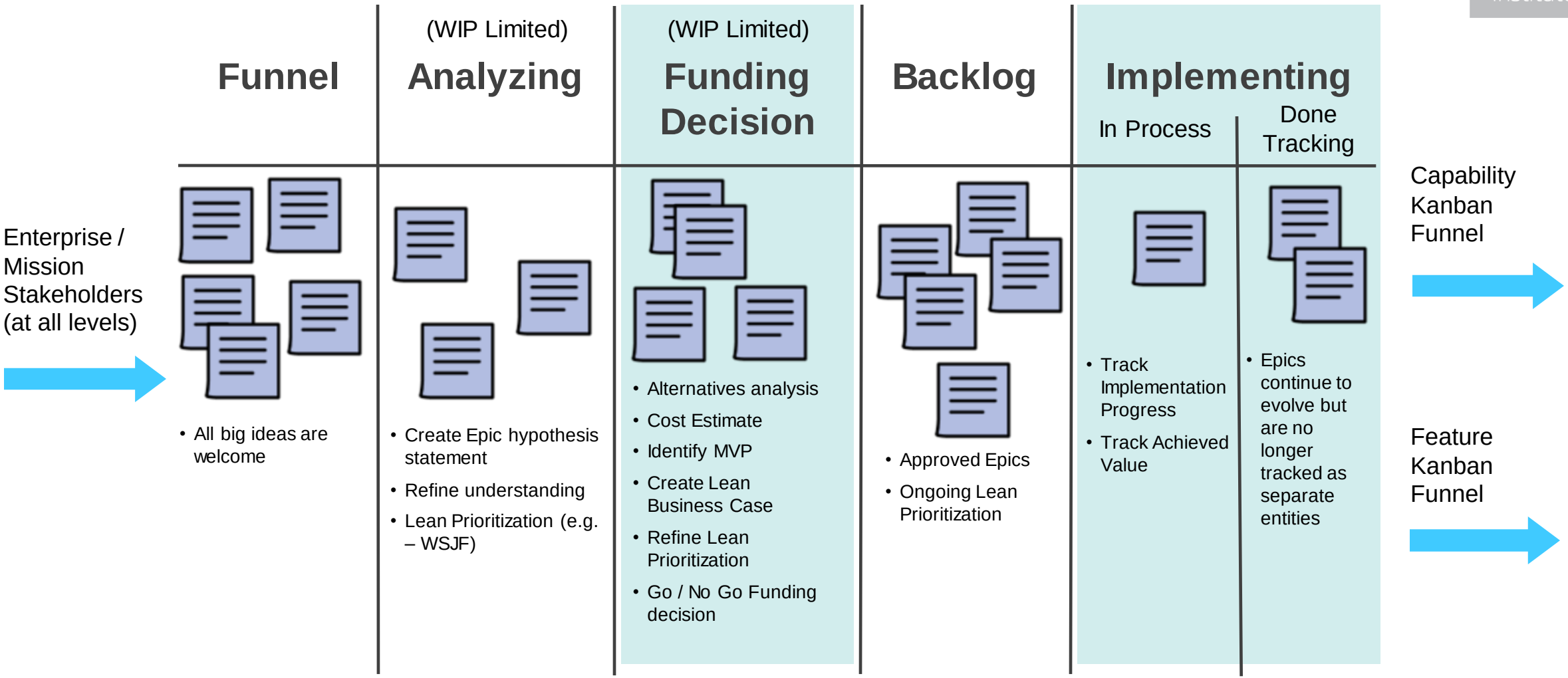


Epics

- Portfolio Level Epic (Business/Mission or Enabler)
  - Represents a significant, strategic enterprise-level initiative / mission. Implementation typically impacts the work of multiple ARTs or Solution Trains.
  - Typically is sponsored and shepherded by an Epic Owner
  - Has sufficient organizational impact to warrant the creation of a Lean business case. Requires Portfolio Level approval.
- Large Solution Level Epic (Business/Mission or Enabler)
  - Represents a significant effort, typically requiring implementation across multiple ARTs and PIs.
  - May warrant creation of a Lean Business Case and approval beyond Solution Management.
- Program Level Epic (Business/Mission or Enabler)
  - Represents a significant effort, often requiring implementation across multiple PIs.
  - May warrant creation of a Lean Business Case and approval beyond Program Management.

https://www.scaledagileframework.com/portfolio-kanban
https://www.scaledagileframework.com/epic

Epic Kanban (all levels)



https://www.scaledagileframework.com/portfolio-kanban
https://www.scaledagileframework.com/epic

Backlog Items Details (Portfolio Epics)

	Portfolio-Level Epic
What is it?	Represents a significant, strategic enterprise-level initiative / mission. Implementation typically impacts the work of multiple ARTs or Solution Trains An Epic may be user-facing or may be a technical enabler of user-facing backlog items.
Sponsorship	Typically is sponsored and shepherded by an Epic Owner
Definition	Described using an Epic Hypothesis Statement
Funding Approval	Has sufficient organizational impact to warrant the creation of a Lean business case. Requires Portfolio Level approval.

** Epics may be User-Facing or may be Enablers.

Large Solution Level Capabilities and "Enabler Capabilities"

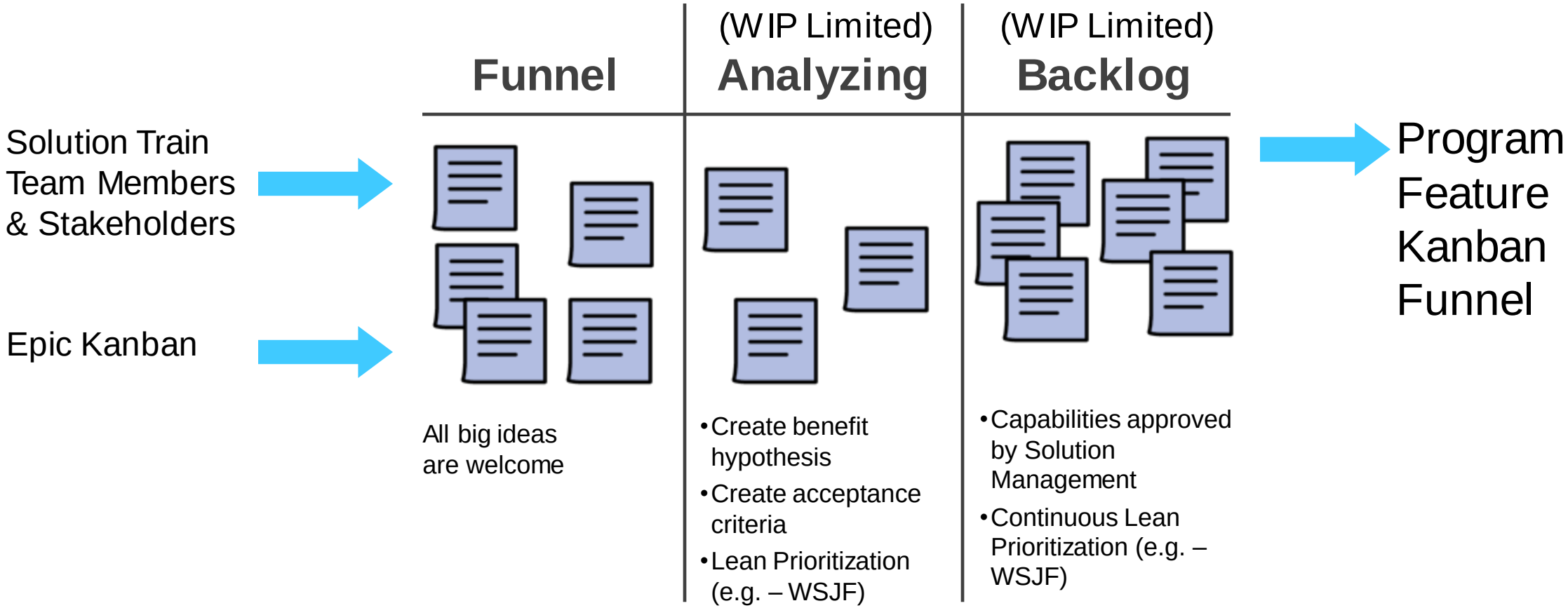
Carnegie Mellon University Software Engineering Institute

- Capability
  - Is a higher level solution behavior
  - Is defined and prioritized by Solution Management
- Enabler Capability
  - Is a development activity that provides technical support for (e.g. - enables) user-facing capabilities or features
  - Is defined and prioritized by the Solution Architect
  - Is typically one of the following 4 types
    - Exploration Enabler; Architectural Enabler; Infrastructure Enabler; Compliance Enabler
- Both Capability and "Enabler Capability"
  - Are sized to be implemented within a single Program Increment, usually by multiple ARTs
  - Are described by a phrase and a benefit hypothesis
  - Have associated acceptance criteria to ensure correct implementation and delivery of value
  - Must be decomposed into features and fed into the Program Feature Kanban to be implemented.

https://www.scaledagileframework.com/features-and-capabilities
https://www.scaledagileframework.com/enablers/

https://www.scaledagileframework.com/program-and-solution-kanbans

Large Solution Capability Kanban



https://www.scaledagileframework.com/program-and-solution-kanbans

https://www.scaledagileframework.com/portfolio-kanbans

Backlog Items Details (Capabilities)

	Capability	Enabler Capability
What is it?	A higher level solution behavior	A development activity that provides technical support for (e.g. - enables) user-facing backlog items. Enablers are typically classified as one of the following 4 types: Exploration; Architectural; Infrastructure; or Compliance
SAFe Level/Backlog	Large Solution Level / Solution Backlog	
Defined & prioritized by	Solution Management	Solution Architect
Sizing	Are sized to be implemented within a single Program Increment, usually by multiple ARTs	
Definition	Are initially described by a phrase and a benefit hypothesis Associated acceptance criteria will be added to elaborate the definition and ensure correct implementation and delivery of value	
Path to Implementation	Must be decomposed into features and fed into the Program Feature Kanban to be implemented	

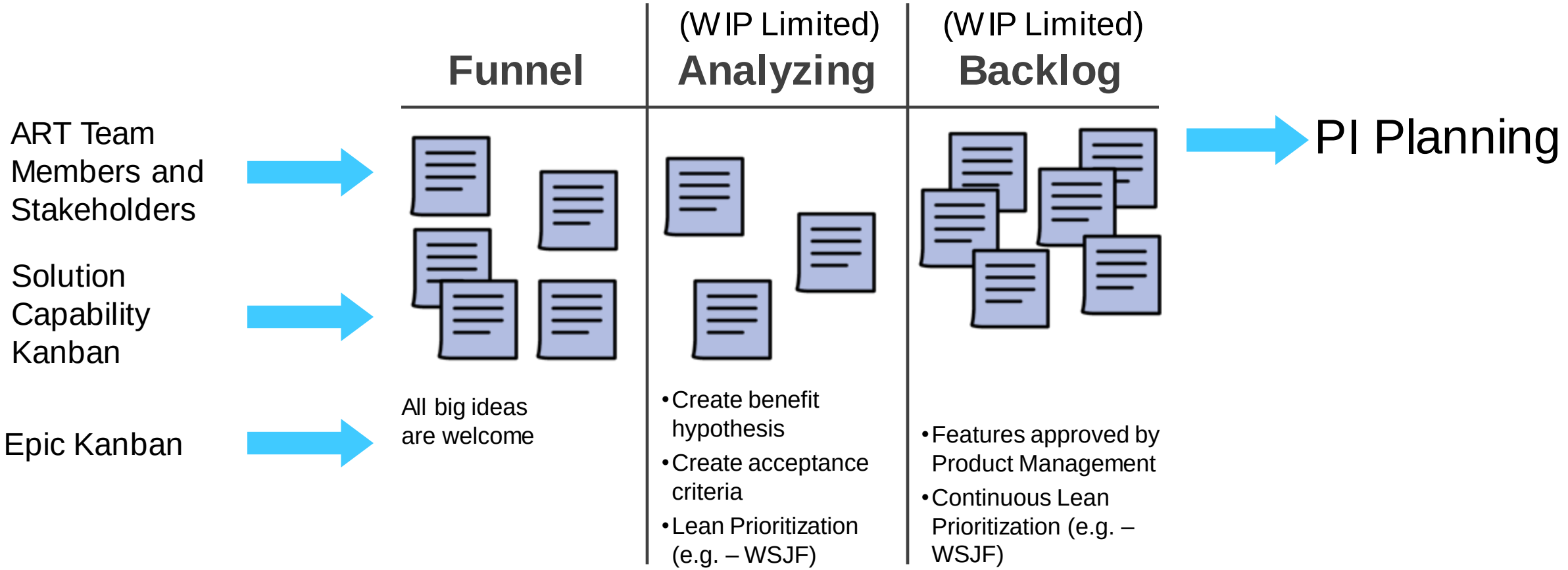
Program Level Features and "Enabler Features"

Carnegie Mellon University Software Engineering Institute

- Feature
  - Is a service that fulfills a stakeholder need
  - Is defined and prioritized by Product Management
- Enabler Feature
  - Is a development activity that provides technical support for (i.e. - enables) user-facing features
  - Is defined and prioritized by the System Architect
  - Is typically one of the following 4 types
    - Exploration Enabler; Architectural Enabler; Infrastructure Enabler; Compliance Enabler
- Both Feature and "Enabler Feature"
  - Are sized to be implemented by a single ART within a single Program Increment
  - Are described by a phrase and a benefit hypothesis
  - Have associated acceptance criteria to ensure correct implementation and delivery of value
  - During PI Planning, are decomposed into stories which are implemented by agile teams within the ART

https://www.scaledagileframework.com/features-and-capabilities https://www.scaledagileframework.com/program-and-solution-kanbans
https://www.scaledagileframework.com/enablers/

Program Feature Kanban



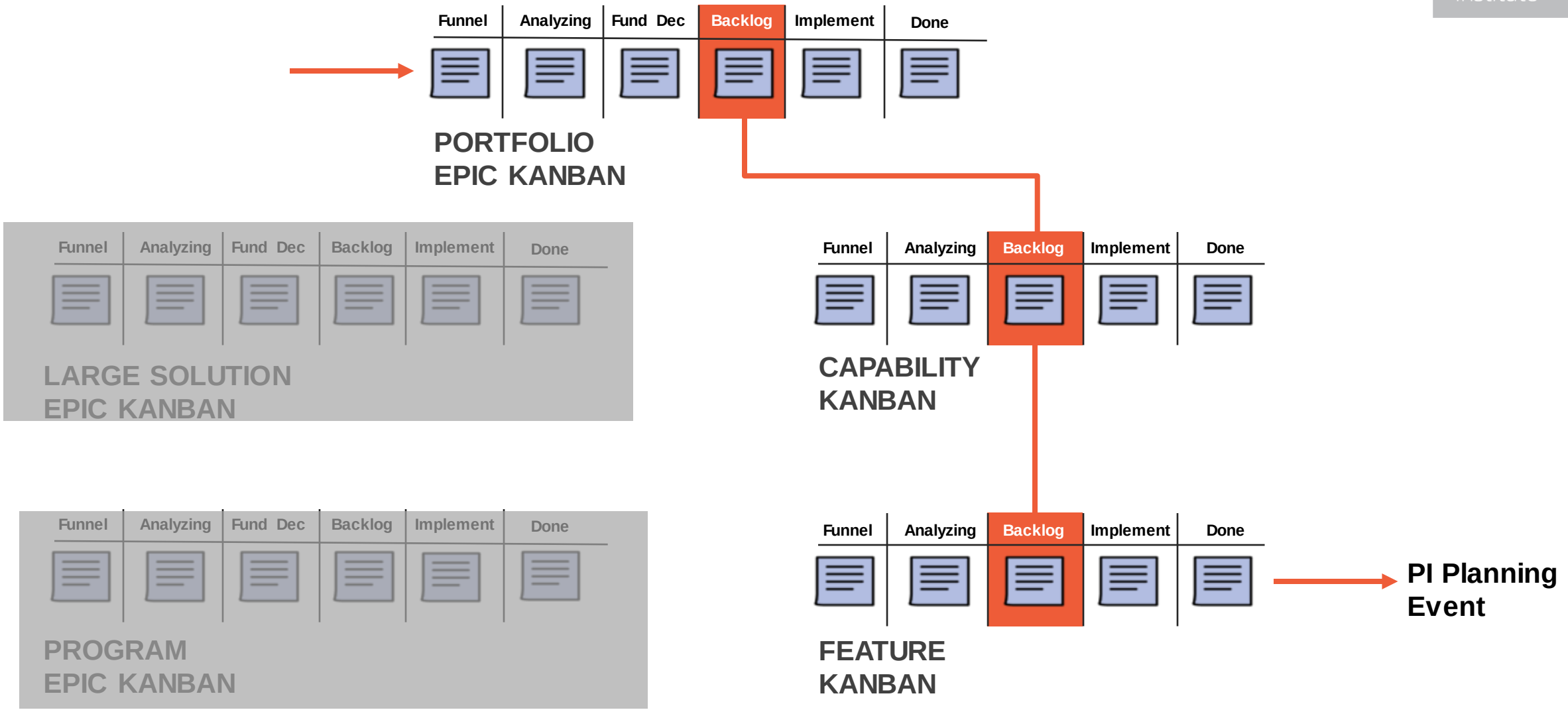
https://www.scaledagileframework.com/program-and-solution-kanbans

https://www.scaledagileframework.com/portfolio-kanbans

Backlog Items Details (Features)

	Feature	Enabler Feature
What is it?	A service that fulfills a stakeholder need	A development activity that provides technical support for (e.g. - enables) user-facing backlog items. Enablers are typically classified as one of the following 4 types: Exploration; Architectural; Infrastructure; or Compliance
SAFe Level/Backlog	Program Level / Program Backlog	
Defined & prioritized by	Product Management	Systems Architect
Sizing	Are sized to be implemented by a single ART within a single Program Increment	
Definition	Are initially described by a phrase and a benefit hypothesis Associated acceptance criteria will be added to elaborate the definition and ensure correct implementation and delivery of value	
Path to Implementation	During PI Planning, are decomposed into stories which are implemented by Agile teams within the ART	

Kanban Flow – Full SAFe – Cascading Flow



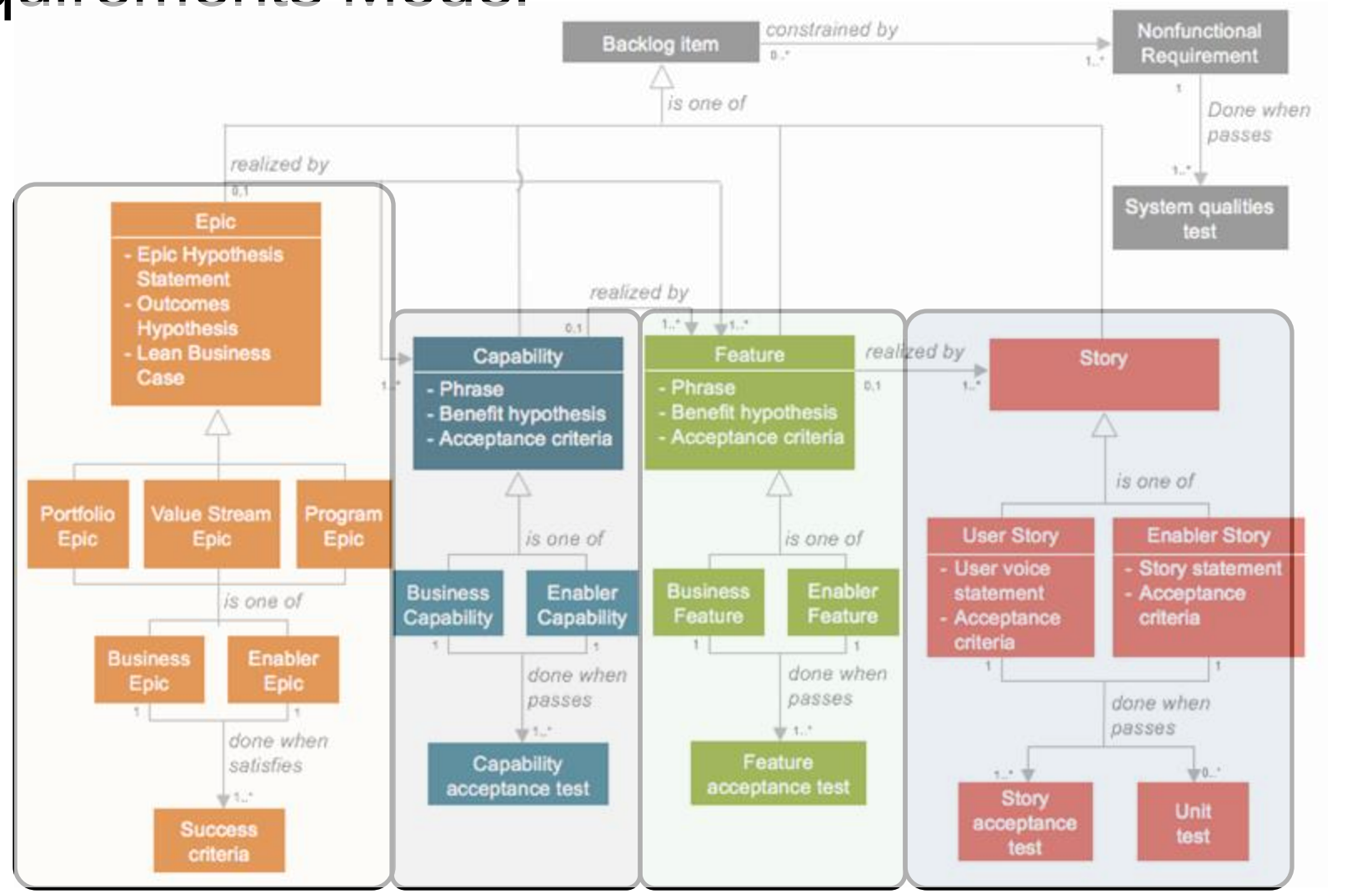
Cascading – A Note of Caution

Agile Cascading / Linked Kanbans do not imply 100% 2 way traceability between Elements of the Requirements Model Layers

For Example:

- At any given point in time, there may be aspects of a Feature that are not expressed as stories within a Team backlog
- There may be stories in a Team Backlog that do not directly trace to a Feature in the Program Kanban

SAFe Requirements Model



Epic Owners | Solution Managers | Product Managers | Product Owners

Frequent Question About Requirements and Sprints

Probably the most frequently asked question about overseeing Agile that we're asked is some variation of:

- *How do we know if deferral of requirements from one iteration/sprint to another is "OK" vs. a sign of a problem?*

Some of the things to look for to answer that question:

- How early is it in the development?
  - Most teams take at least three, as many as six, iterations to get their estimation heuristics consistent enough to achieve their estimates
- Especially early, is there an identifiable "new" dependency that has been discovered that makes deferral of stories appropriate?
- Does the developer recognize they are incurring "technical debt" by deferring stories, and have a strategy for addressing?
- Are the deferrals a result of a larger amount of rework due to defects in previously delivered code?
  - Often occurs when not enough automated testing is used for build integration

Backlog Priority and Maintenance

Carnegie Mellon University
Software Engineering Institute

Backlog Prioritization Methods

Weighted Shortest Job First (WSJF) – SAFe

Prioritization by Committee

Alphabetical

Contractor Choice

Gut Instinct

Mission Based Prioritization

WSJF

$$\text{WSJF} = \frac{\text{Cost of Delay}}{\text{Job Duration (job size)}}$$

Feature	User- business value		Time criticality		RR | OE value		CoD		Job size		WSJF
		+		+		=		÷		=	
		+		+		=		÷		=	
		+		+		=		÷		=	

- Scale for each parameter: 1, 2, 3, 5, 8, 13, 20
- Note: Do one column at a time, start by picking the smallest item and giving it a "1."
- There must be at least one "1" in each column!
- The highest priority is the highest WSJF.

Other

- Prioritization by Committee
- Alphabetical
- Contractor Choice
- Gut Instinct

Mission Based Prioritization

Carnegie Mellon University Software Engineering Institute

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	S	T	U
1				Provides strategic or tactical advantage	Foundational (other features depend on this feature)	Replaces deficient functionality	Supplies missing functionality in existing system	High sponsorship / visibility	Fulfills direct user request	Required to field	Hardens security	Time Critical / Urgent	Previously de-prioritized	Difficulty to implement	Uncertainty	# of iterations to implement	Penalty		Add large feature penalty? (Y/N)	Y
2	Feature	Priority	Weight	1	1	1	1	1	1	1	1	1	1	1 easy - 5 hard	1 solution well known - 5 high uncertain				Number of iterations in a PI	4
3		1																	When to apply penalty % of iterations (default > 45%)	45%
4		1																		
5		1																		
6		1																		
7		1																		
8		1																		
9		1																		

Generate Bucket & Prioritized List

Acceptance Criteria or Definition of Done?

Carnegie Mellon University
Software Engineering Institute

Acceptance Criteria

Difficult to write good acceptance criteria without getting into designing solutions.

- Requires practice
- Take perspective of the system user
- Let the developer do their best work

Bad Acceptance Criteria

Constrain design unnecessarily

(e.g., Design is stated in acceptance criteria)

May compromise architecture

(e.g., the criteria calls for use of .NET when the system architecture is J2EE-based)

Make maintenance more difficult

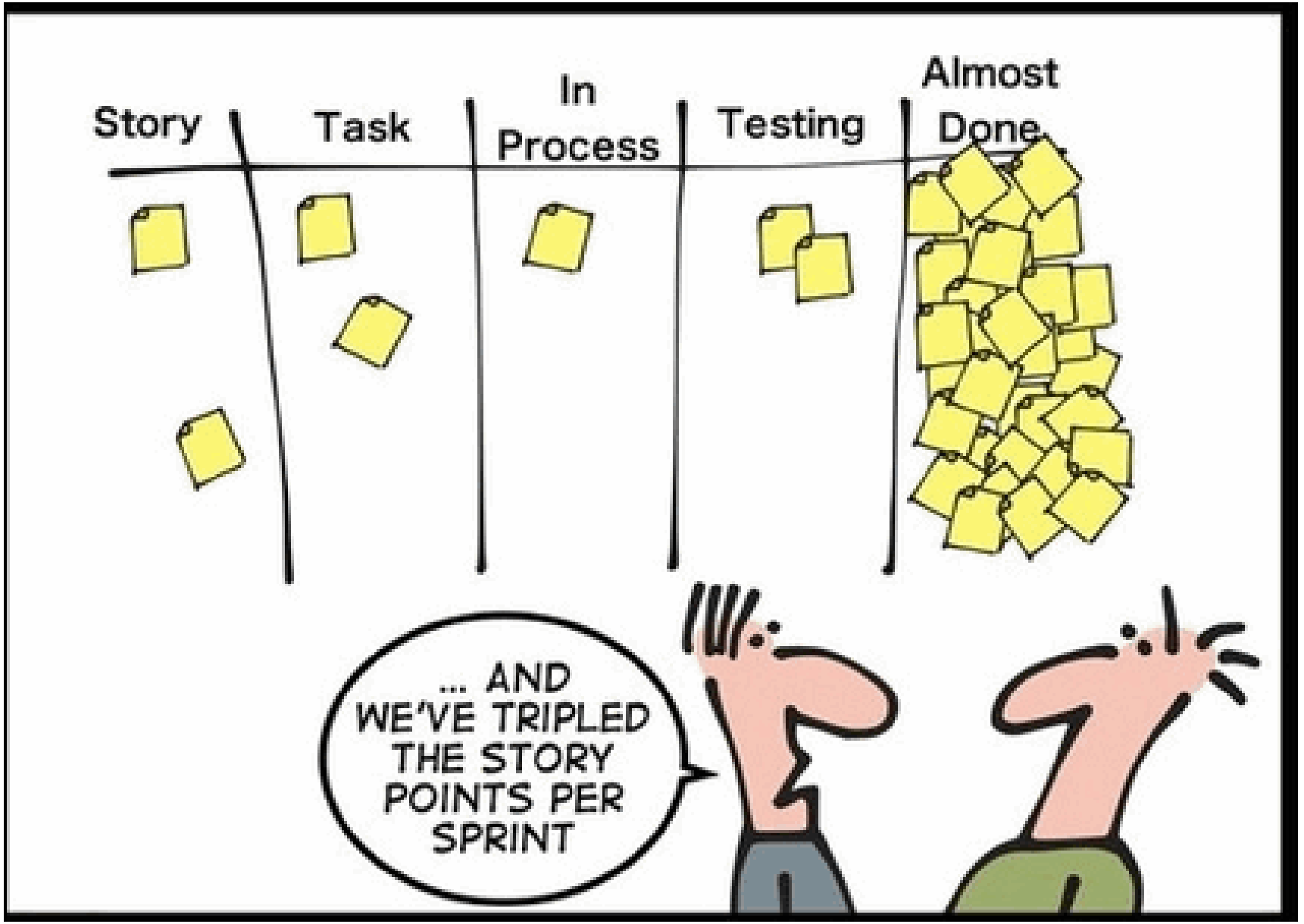
(e.g., References another document)

Limit searching for good solutions

(e.g., Narrows down to one design solution)

SAFe - Scaling the Definition of Done

Team Increment	System Increment	Solution Increment	Release
• Stories satisfy acceptance criteria • Acceptance tests passed (automated where practical) • Unit and component tests coded, passed, and included in the BVT • Cumulative unit tests passed • Assets are under version control • Engineering standards followed • NFRs met • No must-fix defects • Stories accepted by Product Owner	• Stories completed by all teams in the ART and integrated • Completed features meet acceptance criteria • NFRs met • No must-fix defects • Verification and validation of key scenarios • Included in build definition and deployment process • Increment demonstrated, feedback achieved • Accepted by Product Management	• Capabilities completed by all trains and meet acceptance criteria • Deployed/installed in the staging environment • NFRs met • System end-to-end integration, verification, and validation done • No must-fix defects • Included in build definition and deployment/transition process • Documentation updated • Solution demonstrated, feedback achieved • Accepted by Solution Management	• All capabilities done and meet acceptance criteria • End-to-end integration and solution V&V done • Regression testing done • NFRs met • No must-fix defects • Release documentation complete • All standards met • Approved by Solution and Release Management



Carnegie Mellon University
Software Engineering Institute

A Common Requirements Expression at Team Level: User Stories

Note: this section is here for your reference, so we won't go deep on this content in this session.

User Stories - Including the “Who” and “Why” of a Requirement

Purpose

To express concepts in a way operational user would find useful

Template

As a “role,”

I want to “function”

so that I can “operational goal”

Attributes of Good User Stories-1

Carnegie Mellon University Software Engineering Institute

Stories are a short, simple, and clear description of customer valued functionality
Each part should be completed; role and business value provide context (who and why) for the functionality
Stories should satisfy the 3 Cs

- Card – the story should be simple enough to fit on one side of an index card
- Conversation – the story should be the basis of a conversation within the team and with other stakeholders (E.g., Product Managers and Users)
- Confirmation – Each story should have clear acceptance criteria, so that developers can know when they are done

https://ronjeffries.com/xprog/articles/expcardconversationconfirmation/
https://ronjeffries.com/articles/019-01ff/3cs-revisited/

Attributes of Good User Stories-2

Carnegie Mellon University Software Engineering Institute

Stories should satisfy the INVEST criteria

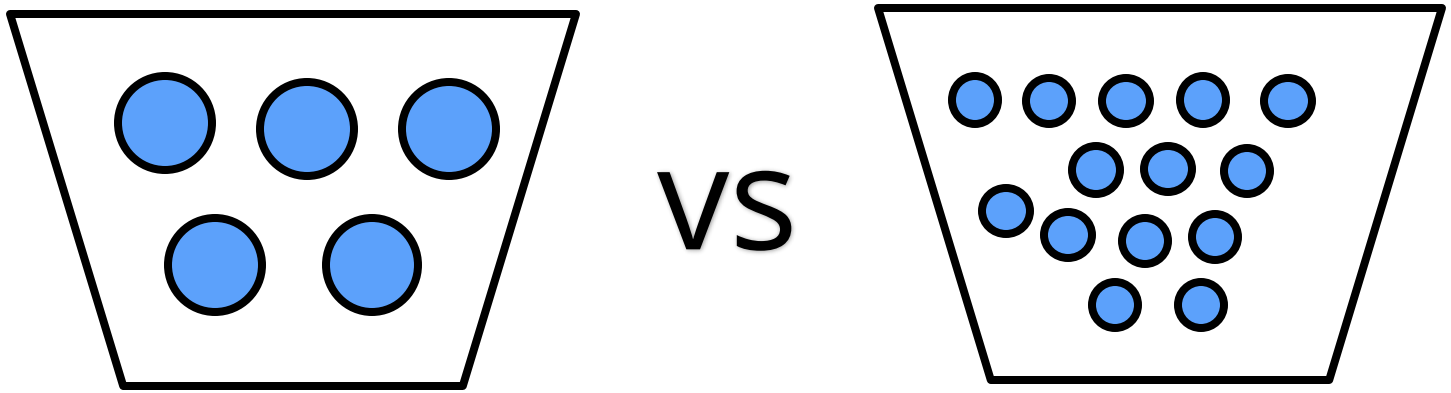
- Independent - among other stories
- Negotiable – a flexible statement of intent, not a contract
- Valuable - providing a valuable vertical slice to the customer
- Estimable - small and negotiable
- Small – Fits in an iteration
- Testable - understood enough to know how to test it

https://xp123.com/articles/invest-in-good-stories-and-smart-tasks/

Key Points in Writing Stories

- Keep stories short and expressed in business language
- Seek a level of granularity that can be completed in a few days
- Keep stories mutually independent
- Do not include implementation details
- Do not stop talking

Story Splitting—Needed When Stories are Too Big to Complete in a Single Iteration



Story splitting is a critical enabler of Agile development
Story splitting requires combining insight into business value with engineering judgment

Goals for Splitting Stories

- Limit story size.
  - Ensure stories can be completed within the sprint.
- Accelerate time to value.
  - Reveal the 80 / 20 split.
- Accelerate learning and risk reduction.
  - Create vertical slices representing end to end functionality to maximize points of integration.
  - Conduct focused experiments.

http://www.scaledagileframework.com/story/

http://www.agileforall.com/patterns-for-splitting-user-stories

Story Splitting Patterns

- Workflow steps
- Business rule variations
- Major effort
- Simple/complex
- Variations in data
- Data entry methods
- Defer Quality attributes (but tread carefully!)
- Operations (ex., Create, Read, Update, Delete [CRUD])
- Use-case scenarios (happy path, alternate flow, exceptions processing)
- Break-out spike

http://www.scaledagileframework.com/story/

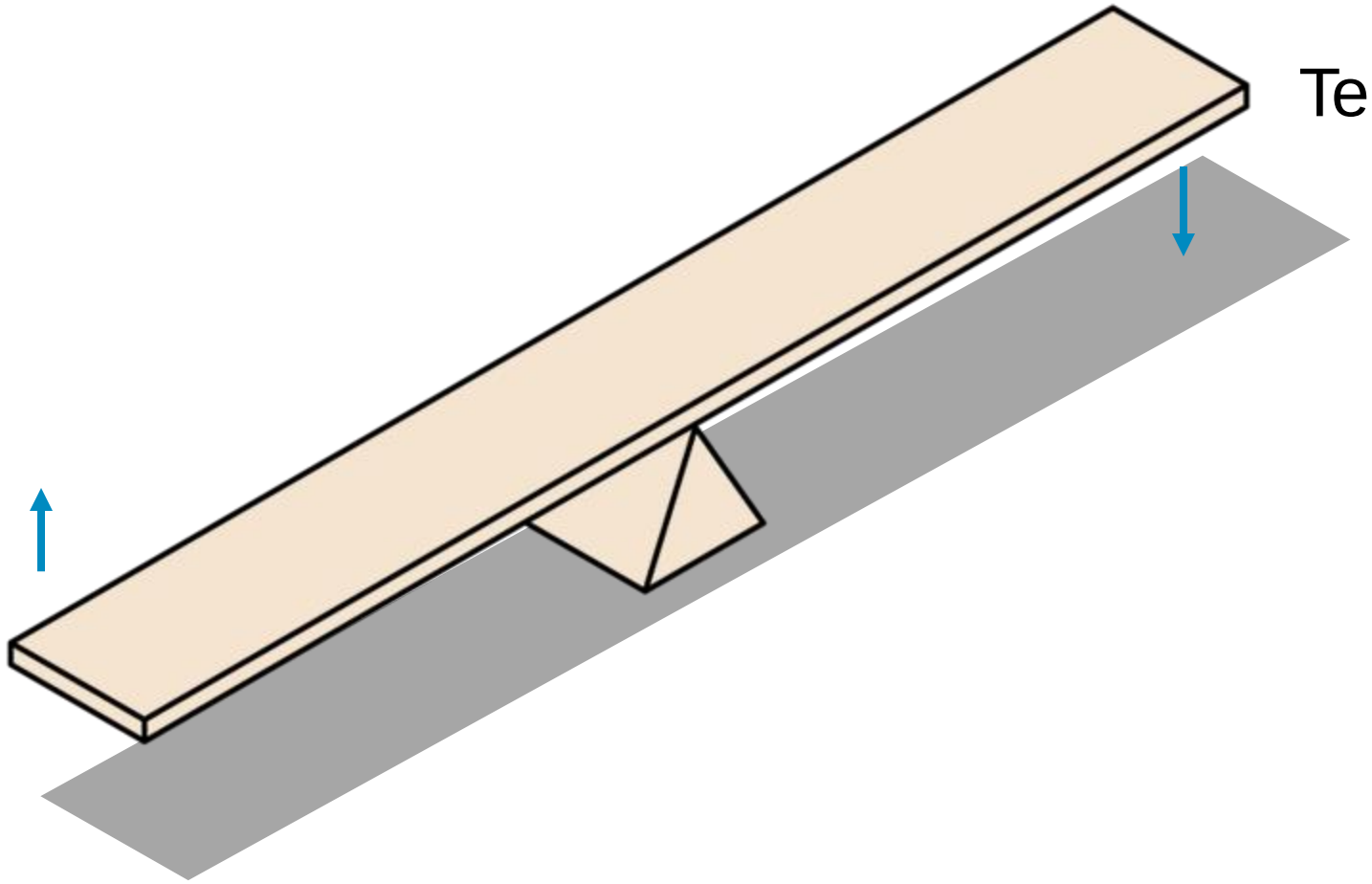
http://www.agileforall.com/patterns-for-splitting-user-stories

Story Splitting Tradeoffs

Carnegie Mellon University Software Engineering Institute

Technical Debt

Learning &
"Time to Value"



Final Thoughts
(not a summary exactly…)

Carnegie Mellon University
Software Engineering Institute

This is truly the tip of the iceberg!!!

Carnegie Mellon University Software Engineering Institute

It's tempting to focus on the team-level stories construct for derived requirements

- Appropriate in small programs, and/or where government actually has development teams in play

Most of the government interaction with development contractors is at a higher level of abstraction

- Features (often where Use Cases show up, if you're using them; often SRS/HRS level)
- Capabilities (large grained description of capabilities –often a combination of Govt ORD/TRD types of documents and KTR System/Subsystem Specifications)
- Epics – the strategic, mission-focused needs statements that justify why the system is being built (often represented in the CDD and Con Ops)

Final Thoughts-2

Extent to which the Program adopts a Lean/Agile approach to their requirements, not just the KTR requirements, determines what skills/changes to local government work practices need to be considered

- "Agile at the Bottom of the V" – only the requirements that will be implemented for software and only at software team level are translated into Agile
  - Not much difference in SPO work, but not much benefit
- SAFe-ish Requirements Approach
  - Joint KTR/Govt mgmt of Feature backlog
    - Changes from spec-based single definition to incremental continuous definition and refinement
    - More small-batch engagements between govt and KTR product managers
  - Govt ownership of requirements backlogs at Capability and/or Epic layers
    - Changes the way work and projects are conceived of
    - Skill, procedure, structure, strategy changes all come into play

Final Thoughts-3

Carnegie Mellon University Software Engineering Institute

Lots of topics related to requirements that we didn't deal with here:

- Agile requirements and how they support shift-left testing (we'll address in Agile & Testing learning module)
- Agile requirements and program (not story) level estimation (not addressed directly in this series)
- Agile requirements and progress measurement (we'll address in Agile & Measurement learning module)
- Contracting for Agile requirements (we'll address in Agile & Contracting)

Contact Information

Keith Korzec
Senior Member of Technical Staff
SSD/CDC
Email: kkorzec@sei.cmu.edu

U.S. Mail
Software Engineering Institute
Customer Relations
4500 Fifth Avenue
Pittsburgh, PA 15213-2612
USA

Customer Relations
Email: info@sei.cmu.edu
SEI Phone: +1 412-268-5800
SEI Fax: +1 412-268-6257