SEMAT

Essence
Software Engineering Essentialized

Essence
Software Engineering Essentialized
Part 4 – Large Scale Complex Development

Giuseppe Calavaro, Ph.D.

www.semat.org

Agenda of this Teaching Module

- Large-scale complex development
- The three dimensions of Scaling
  - Zooming In
  - Scaling Up
  - Reaching Out
- TravelEssence Large Scale development using Essence
- Value of Essence to Large Scale Development

Example of using Essence on our project "TravelEssence" are in green boxes

Module Objective

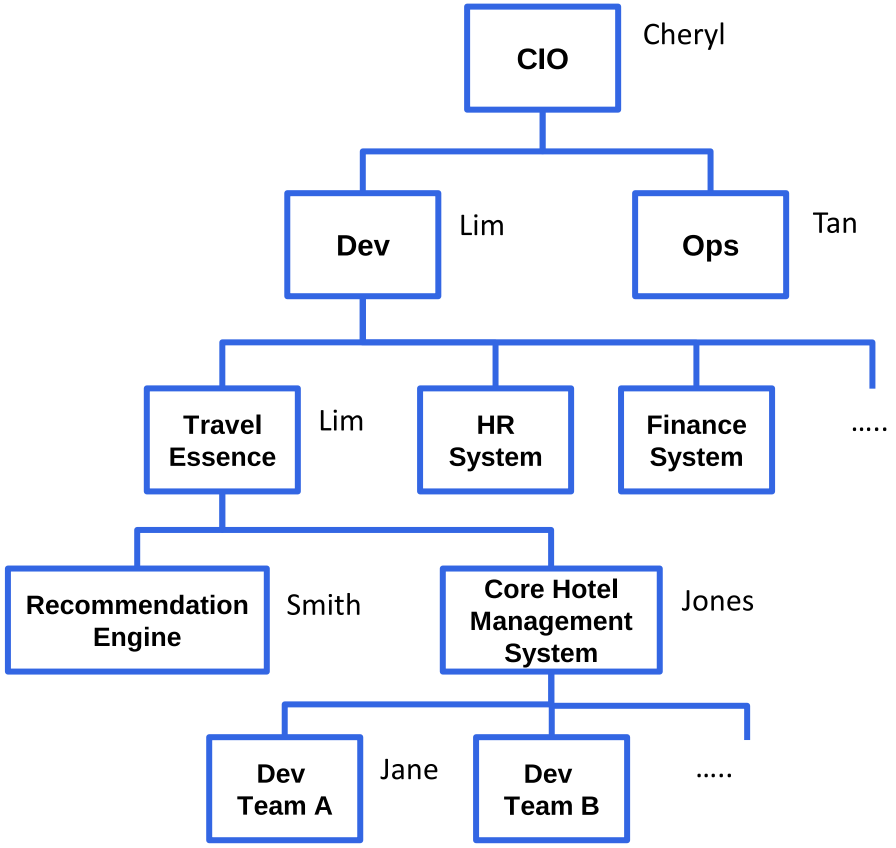
- The objective here is to provide an understanding for what it means to work with large-scale development and how that is different from working with small-scale development
  - Here, it is **not the objective** to equip you with the competency needed to actually participate in large-scale development,
  - but to let you have a glimpse of what it is like
- We also want to highlight that Essence is scalable to large-scale development
- Large-scale complex development encompasses many challenges both collaboration and architectural
  - We have chosen to focus on the collaborative aspects
  - We leave the more advanced architectural aspects out of the scope for this book.

Large-scale complex development

- Large Scale Projects have a lot of people involved, in many teams, working in parallel and collaborating with each other.
  - Usually there are multiple groups of stakeholders and multiple software systems being built or maintained
  - Yet, Large-scale projects is not just about having more people
- The software system itself usually has more complex and more stringent requirements
  - For example performance, reliability, security, compliance
- A **program** is a larger endeavor, which usually comprises several smaller, still potentially large related endeavors.
  - Different teams may work on different endeavors within the program.

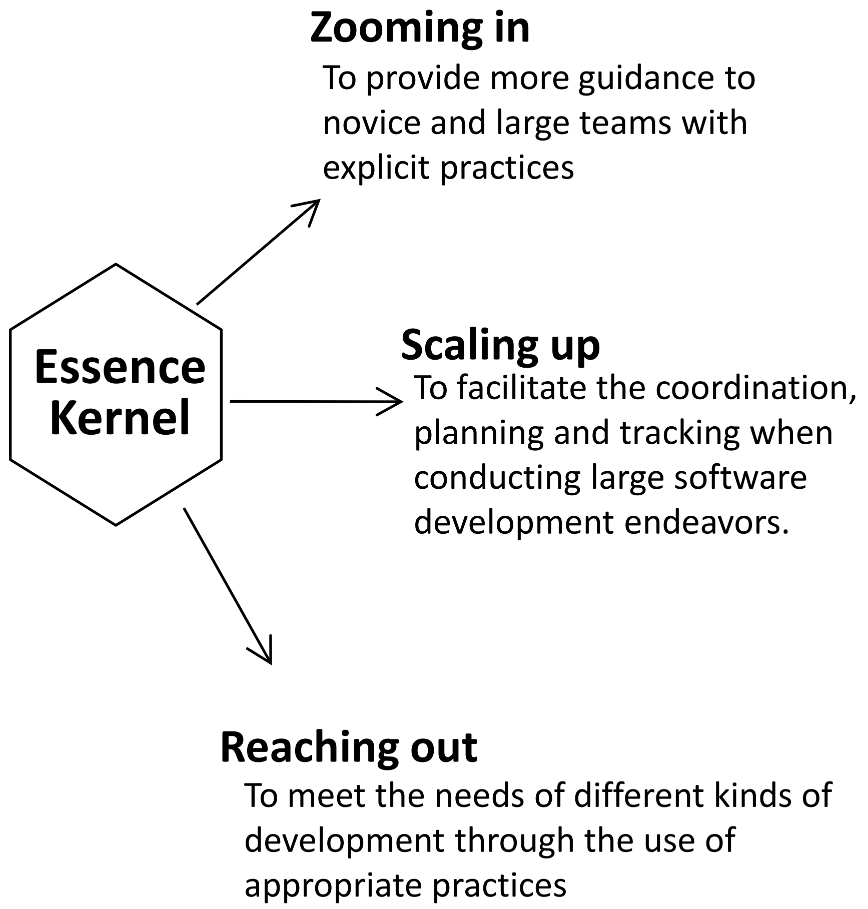
TravelEssence journey continued

- We will continue the journey with TravelEssence and understand how Smith's development team operates as part of a much larger organization
  - You should now be familiar with Smith's development team.
- Let's now see how that team fits within their larger development context.
  - Here you can see some example names for individuals that will play in our story



Scaling from Small to Large Development

- Scaling from Small to Large development poses several challenges:
    1. Members will often have diverse background and experiences. The need for guidance can be very different among people
    2. Coordination of the work between members becomes a challenge. A major challenge is to ensure that members discuss, share and agree on their plans, progress and the results of their work
    3. Different kinds of software endeavors have different risks and constraints, and therefore face different challenges. For example, some have quality priority and other speed priority
- The above are three distinct challenges to scaling
- Accordingly, when considering what it means to scale, we need to address three corresponding dimensions of scaling


Essence Kernel
Zooming in
To provide more guidance to novice and large teams with explicit practices
Scaling up
To facilitate the coordination, planning and tracking when conducting large software development endeavors.
Reaching out
To meet the needs of different kinds of development through the use of appropriate practices

The three dimensions of Scaling

Zooming In

- To provide more **guidance to novice and large teams** with different background **with explicit practices**
  - Beyond what the kernel provides
  - Essence allows *zoom-in* through *essentialization*
- It is about making practices explicit and available through a practice architecture
  - By making the alphas, alpha states, work products, activities and patterns explicit, Smith's team received clear explicit guidance on how to proceed and continually improve.

Scaling Up

- Expanding the use of the kernel
  - from a team
    - Small number of members and low complexity
  - to endeavors,
    - Large numbers of people and systems with high complexity
- Examples:
  - Building Product lines (e.g. a smartphone series with various screen sizes and configurations)
  - large-scale systems and software engineering (e.g. building an aircraft)
- **The Essence kernel can be extended with practices for such complex development**

Reaching Out

- Extending the kernel with appropriate practices to meet the specific needs of **different kinds of development**
- For example outsourcing some development to another company
  - Outsourcing refers to contracting work to an external organization
- Selecting **appropriate practices to address the risks and challenges**
  - A simple approach is to empower the team to build their own method.
- Training and coaching teams about those practices and transforming them into a learning organization

Practices come from many sources

- Practices are just good ways to deal with specific problems in certain contexts
  - Some of these practices are given names, while others have not.
- The popular and more successful practices spread across the community
  - These practices were presented, debated, tried and tested before ending up in papers, books, and adopted by fellow practitioners as practices within their methods.
- Examples of popular Team oriented practices:
  - Scrum,
  - Use-cases,
  - extreme programming (XP)
- Examples of Large-scale development frameworks (of Practices):
  - Scaled Agile Framework (SAFe),
  - Disciplined Agile Delivery (DAD),
  - Large Scale Scrum (LeSS)
  - Scaled Professional Scrum

SEMAT

Monolithic Methods and Fragmented Practices

- For monolithic we mean that a method is non-modular
  - It is difficult to add, modify, replace (substitute or exchange) a practice from outside
  - Example of monolithic methods:
    - Rational Unified Process (RUP),
    - Scalable Agile Framework (SAFe),
    - Disciplined Agile Delivery (DAD),
    - Large Scale Scrum (LeSS)
- Monolithic methods are an amalgam of practices where the practices are not clearly distinguishable from one another
  - Practitioners rarely, if ever, apply any of these methods in their entirety, but only parts of them
- A method is fragmented when it is difficult to see or understand how its constituent pieces fit into the whole
  - These Methods focus to address a certain challenge, with no regards as how they relate to other practices
    - Ex: product design, team collaboration, and so on,
- Many times, different terms are used to denote the same thing
  - For example, iterations and sprints are largely similar

Customizing Practices

- Organizations would typically evolve their own practices as they find better ways of working
  - These home-grown practices initially appeared only in conversations and were not made explicit through documentation
- Essence provides a solid architectural structure to coach colleagues and sustain improvements

Essentialization

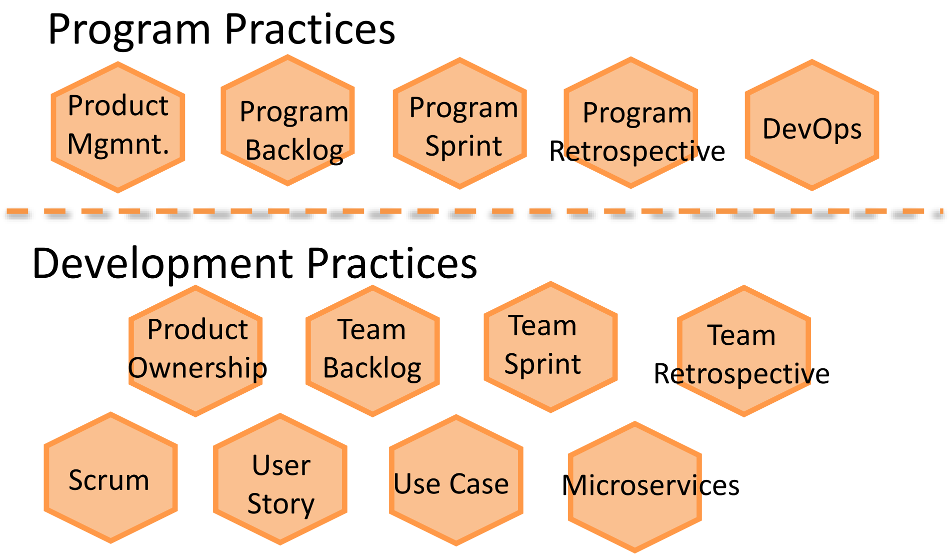
- From a methodology point of view, it is about identifying and scoping a practice from a practitioner’s perspective, describing it using the Essence language such that it is compatible with other practices.
- In the larger picture, it is also about introducing such a practice to teams, teaching and training them such that they can learn, apply and adapt the practice to their context
- The essentialization of a practice is the activity of making a practice explicit by describing it using the Essence kernel and language as a platform.

Composition

- Often teams need many practices that are designed to be composed together in what we call a practice architecture. For example
  - use cases
  - microservices
- A simple way to think about practice composition is to think about a merge operation
  - For example, a practice may require that an alpha may have some checklist items and another practice may require the same alpha to have other checklist items. The composition or merge operation will merge these two checklist items together into that alpha

Establishing a Reusable Practice Architecture

- Having extracted practices, it is important to have a way to organize and present the practices so that they can be reused by teams selecting their own method
- One way to construct a practice architecture is in a layered form
  - With more generic practices at the bottom and
  - more specialized practices at the top



Development Practices

Here we find some new general practices such as:

- **Product Ownership** – This is a practice for deciding and prioritizing the kind of requirements that a team would deliver.
- **Team Backlog** – This practice provides guidance on how to manage and track items in a team's backlog.
- **Team Sprint** – This practice provides guidance for how teams can work iteratively, from sprint planning to sprint delivery
- **Team Retrospective** – This practice provides guidance for how teams can continually reflect on how they are working and making necessary improvements

Program Practices

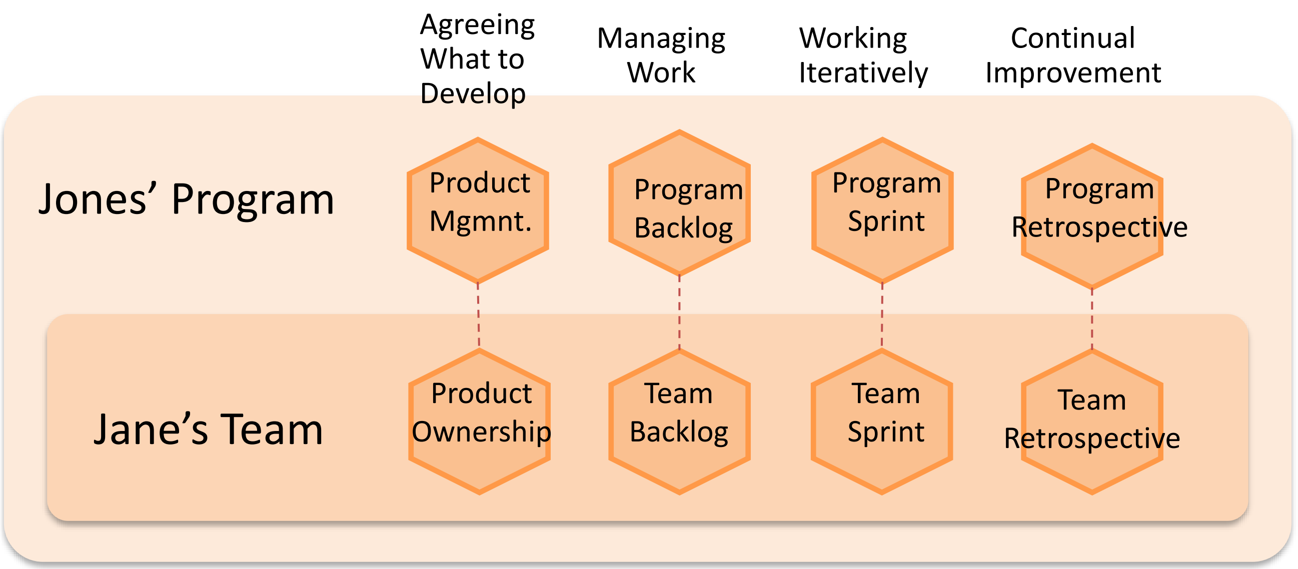
- You can think of a "*program*" as a large "*project*" involving smaller "*projects*"
  - Organizing the work within a program can be complicated
  - You need to balance priorities from different stakeholders, and manage the dependencies across the teams within the program
  - Moreover, the work may cut across different departments.
- **Program Backlog** – Provides guidance on the prioritization and acceptance of the program backlog items
- **Program Sprint** – Programs applied an iterative cycle to deliver Program Backlog Items (PBI), allocated to development teams
- **Program Retrospective** – to analyze the way of working and results to seek improvements.
- **DevOps** – An automated deployment pipeline is a set of automated scripts and mechanisms that performs checks, such as compilation checks, and testing to ensure that the deployed software works not just on the developer's machine, but everywhere the deployed software needs to run, even in the production environment

Scaling Up to Large and Complex Development

- In large-scale development, the team members are often distributed across geographical locations
  - These development teams will likely have multiple requirement sources coming from different customers at any point in time
  - They may also be serving different opportunities from different stakeholder groups
- There are many approaches to large-scale development
  - The Capability Maturity Model Integration (CMMI) helps teams improve processes
  - the Rational Unified Process (RUP) is an iterative software development method
- It is not easy to replicate what happens in the real world of large-scale development in a classroom setting
  - It is also not easy for new students who work mostly alone to visualize how large teams operate

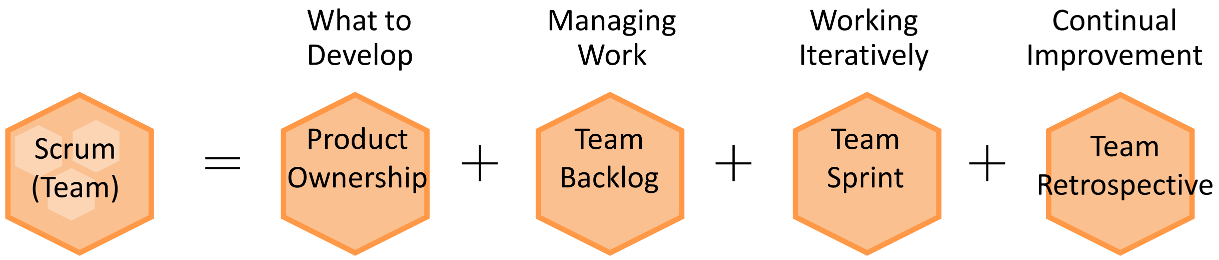
Organizing practices into layers

- The team layer comprises practices useful for teams to work effectively together to deliver great quality software.
- The program layer comprises practices to help coordinate the work across teams and across departments or IT systems



Team Level Practices

- We presented Scrum Lite as a single practice
- Here we show Scrum as a composition of practices at Team level
  - product ownership (what to develop),
  - backlog management (managing the work),
  - team sprint (working iteratively), and
  - retrospectives (continual development)



SEMAT

Program Level Practices

- The concepts at the program layer are very similar to that at the team level, except that they operate at a larger granularity, with a longer planning horizon.
  - The practice architecture was organized to show how each practice contributes to helping the organization work effectively at each level
  - Organizations have different problems at different organizational levels, yet they can reuse the same principles to solve them
- The lines linking practices in show a simple relationship to indicate that these practices should be applied together to achieve synergy
  - For example, take the example of Team Sprint & Program Sprint
    - They both shared the same notion of sprints
    - i.e. to work within periodic cycles

Kick Starting Large Scale Development

- Running large-scale development is not just about selecting and applying practices as we have seen above
- The steps for large-scale development are
    1. Understanding the context through the lens of Essence
    2. Agree on development scope and checkpoints (where it begins and where it ends)
    3. Agree on practices to apply
    4. Agree on the important things to watch

Understanding the context

- We walk through each kernel alpha and assessing its state, along with providing rationale for each assessment
- In the interest of simplicity, we just look at the alpha states achieved for Jones' Program team

Alpha	Program State Achieved	Rationale for achieving the state
Stakeholders	Recognized	
Opportunity	Value Established	
Requirements	Conceived	
Software System	Architecture Selected	
Work	Started	
Way of Working	Foundation Established	
Team	Formed	

EMAT

Agree on development scope

- The scope of development context can be defined easily by agreeing
  - what should be achieved before the beginning of each cycle
  - what the team or program should achieve by the end of each cycle
    - These can be defined easily through alpha states

Alpha	Before Sprint Starts	Before Sprint Ends
Stakeholders	Involved	Satisfied for Deployment
Opportunity	Value Established	Addressed
Requirements	Coherent	Fulfilled
Software System	Architecture Selected	Ready
Work	Prepared	Concluded
Way of Working	Principles Established	Working Well
Team	Formed	Collaborating

Agree on practices to apply

It is key to agree on which practice to apply at **both levels**: the program level and the team level

- These practices are different even if aiming to the same purpose

Purpose	**Program Level Practice**	**Team Level Practice**	**Guidance to Kernel Alpha Progression**
Agreeing what to develop	Product Management	Product Ownership	Opportunity Requirements
Managing the work	Program Backlog	Team Backlog	Requirements, Work
Running Iteratively	Program Sprint	Team Sprint	Work
Continual Improvement	Program Retrospective	Team Retrospective	Way of Working

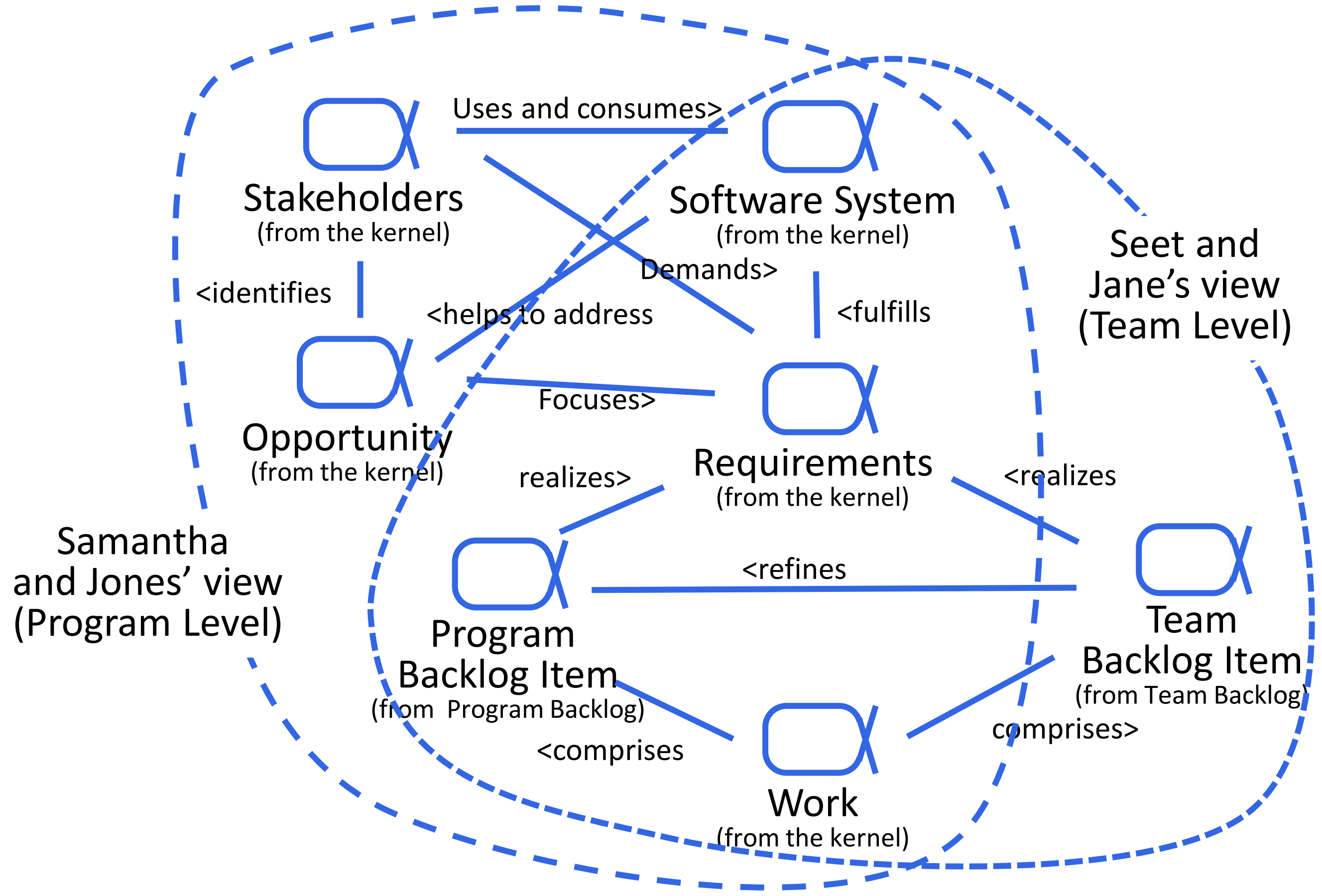
Agree on the important things to watch

- On each endeavor, regardless of size, there are alphas from the kernel and alphas from the practice level that teams agree need to be watched to assess the progress of the endeavor accurately

Layer	Program	Team
Alphas from the Kernel	Opportunity, Stakeholders, Requirements, Software System, Work	Requirements, software system, Work
Alpha from the practices	Program Backlog Item	Team Backlog Item, Program Backlog Item

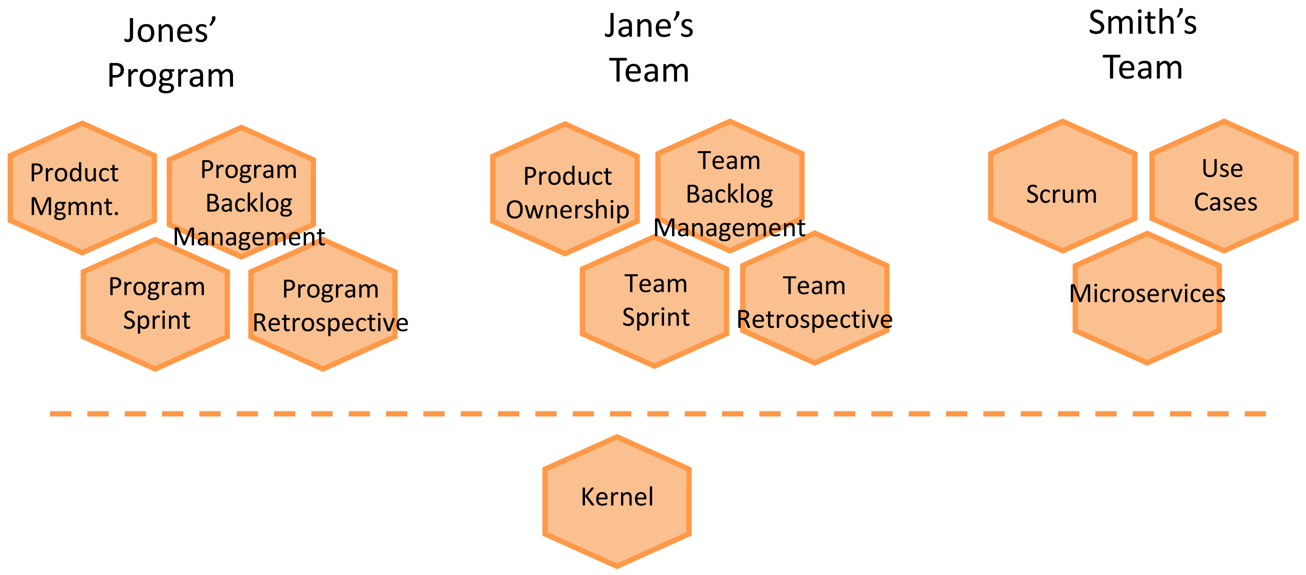
Program vs Team Level: things to watch

- Jones' has a Program Level perspective to the important things to watch that is different but overlapping partially to Jane's Team level



Value of Essence to Large Scale Development

- What we have described before is the way of working of Jones' program and Jane's team
  - You would notice that there were significant differences.
  - Even between different teams there could be differences in the practices adopted
  - Regardless of whether they were working on the team level or the program level, or what specific practices they applied, their practices were all defined on top of the kernel


Jones' Program
Product Mgmnt.
Program Backlog Management
Program Sprint
Program Retrospective
Jane's Team
Product Ownership
Team Backlog Management
Team Sprint
Team Retrospective
Smith's Team
Scrum
Use Cases
Microservices
Kernel