Principles, Practices Lean and Agile Software Management In The Large

Experiences of a Playing Coach

Dave Thomas

Object Mentor Inc. and Bedarra Research Labs
Carleton University Canada, Queensland University of Technology Australia

www.davethomas.net
davethomas@objectmentor.com

Outline

- Lean, Agile and Agility
- Large-Scale Lean and Agile Development At-A-Glance
    - Common Work Practices
    - Envisioning
    - Definition
    - Development
    - Release Engineering
- Social Engineering - Peopleware
- Discussion

Large?

Large Scale Product Software – Why So Wicked?

Exists in the muddle between Lean and Agile

- End Customers not available to explain requirements and manage scope
- Release dates and content must be fixed 12 – 18 months ahead meet the needs of the market
- Product Lines/Platforms are used to deliver multiple products
- Developers are geographically distributed
- Software Teams have dependencies dictated by architecture/features and accidentally by the code base
- Software relies on many 3rd party products
- Builds and Tests often require special environments and manual labor
- Performance, Usability … requirements
- CMM Level 3+, Six Sigma and SOX are all must have requirements

Classic Faults In The Large

- Management versus Leadership
  - Top-down dictation of schedules
  - PMO and excess project managers
  - Death by meetings (i.e. Productivity = 1/number of meetings)
  - Managers managing up!
  - Quality is a police force rather than a practice
  - Requirements are documented rather than communicated

- Draconian water fall process complete with templates and signoffs
  - Faked by almost everyone
  - Requirements database (e.g. 5000 item wish list with no business value or examples)
  - Death by documentation – (e.g. analysis, architecture, design documents most of which can't be round tripped because it changes when the code is written)
  - CMM certification measures you have a process, not that you actually follow it!

Classic Faults In The Large

- Legacy Architecture, Technology and Organization
  - Architecture built to last, impossible to change
  - Technology once best of breed no legacy and limiting productivity
  - Architecture and Technology constrain customer ability to upgrade
  - People aligned with architecture and technology capable but locked in the legacy

- “*Over the wall* testing” with associated slips in schedule and quality
  - System test or SQA with no clear ownership of defects in requirements or code
  - SQA police with metric whips for code, but none for requirements, infrastructure or schedule sanity
  - Increased emphasis on process as a patch
  - Stabilization Releases - The name says it all! We ship it with bugs so we can make the date we promised and try to fix it later

Generally Accepted Software Physics – Often Ignored

- It has been written at least twice before, ideally by the same people
- It needs to be written at least twice after it is released to really be good
- One major version per 12 – 18 months with 2 - 3 dot releases
- All developers on the team should work on the same code stream
- Fix the date, reduce the scope (forget more resources, more time)
- Make sure every one knows the Vision – Tell a simple story and stick to it
- Ship Your Organization – Staff The Architecture
- Build To Last – ADD (API First Design)
- Just `DoIt` - Fail Fast, Fail Often; Don't debate it when you can test it
- KLOCS kill, less code is always more
- Negotiation estimates to match reality
- Don't do new design in a product release time box
- Don't develop components and expect to use them in the same time box
- Be cautions of framework development
- Kill defective components before they degrade so much they kill you
- Use common practices and tools (the best wrong thing)
- Automate everything
- Listen to outside critics – Don't believe your own happy talk

Challenges Facing Our Large Clients

- Business is less requirements driven and more reaction driven as they lose leadership and begin to follow their customers. Business decisions constantly changing, R&D decisions are slow.
- Legacy code base and management thinking blocks innovation and rapid development. Robust and Reliable becomes an excuse for slow and not good enough.
- Integration as important as development but has not been a major R&D investment. Component quality is good but system quality and delivery are much less predictable. Component design skills different from service design skills.
- Use of low level languages, tools and few components means that *age and energy* become the only competitive differentiator which favors new players in emerging countries.
- Marketing has no sense of risk, engineering is risk averse and little sense of urgency and is constrained by legacy. IT does it right, but competitors steal market share faster and don't seem to do it wrong.

Solutions for Improved Agility

- Aggressive partnering to leverage components and skills.
- Offshore development to leverage cost, age and energy.
- Streamline Software Development with Lean Software Management and Agile Practices.
- Use of more productive software technologies, practices and tool chains.

You Need to Change The Game To Achieve Increased Agility!

Lean and Agile and Agility

- Both Lean and Agile and *Death By Quality* (TQM, Six Sigma, CMM) come from Toyota ***Just In Time*** (now called **Lean**)
- **Lean** appeals to the business which often don't understand software and often feel hostage to it. Lean is a top down process to identify **improvements** to the software value chain. It requires constant innovation and improvement in all parts of the business.
- **Agile** appeals to developers many of whom don't understand business sometimes feel hostage to it. Agile is a bottom team centered process for improving software development **predictability** and **quality**.
- **Agile Development** (Scrum, XP, TDD …) is the best set of practice for small teams to develop software especially when scope is managed by a knowledgeable business customer/product owner.
- **Agility** is the capability of the business to respond to the rapidly changing needs of their customers, partners and their competitor. It naturally appeals to business leaders but they confuse it with Agile Development which promises improved predictability and quality but not productivity or flexibility).

Solutions for Improved Agility

- Aggressive partnering to leverage components and skills.
- Offshore development to leverage cost, age and energy.
- Streamline Software Development with Lean Software Management and Agile Practices.
- Use of more productive software technologies, practices and tool chains.

- You Need to Change The Game To Achieve Increased Agility!
- The challenge is not to minimize expenses rather it is to maximize business value!
- Experience Matters!

Lean Software in a Nutshell

- **Respect the individual** (“work with”)
- **Empower Team** through leadership and coaching
- Proactively **identify and eliminate software waste**
- Use **common work practices**, rhythm and tools
- **Maintain Visibility** - Make all activities, artifacts and risks visible
- **Decide as late** as possible
- **Deliver as fast** as possible

Lean Software in a Nutshell

- **Make Requirements Tangible** – So the developer can test and the customer can see
- **Design Quality In** – Make automated testing part of requirements, design and development. **Done** = **Unit** and **Acceptance** Tested Components and Features
- **Each team** owns its **predictability**, **progress** and **quality**
- **Build To Last** - Interfaces are *essential* for independent development and acceptance testing. Components support Platform based products
- **Ship Your Organization** – Align the Teams with the Product Architecture
- **Defer Commitments** to as late as possible
- **Align compensation** with predictable delivery and quality
- **Continuously improve** your people and practices hence your products and business => Encourage and Reward Learning and Innovation

Lean Software

- Lean principles derived from Just-In-Time Manufacturing
- Incorporates values and practices articulated in Peopleware, Spiral and Rapid Application Development

Seven Lean Principles

Eliminate waste
Amplify learning
Decide as late as possible
Deliver as fast as possible
Empower the team
Build integrity in
See the whole

Core Shared Values

Client-focused
Client-driven
Incremental results
Continuous questioning and introspection
Change is progress to a better solution

Lean Software in a Nutshell – Maximize The Value Delivered

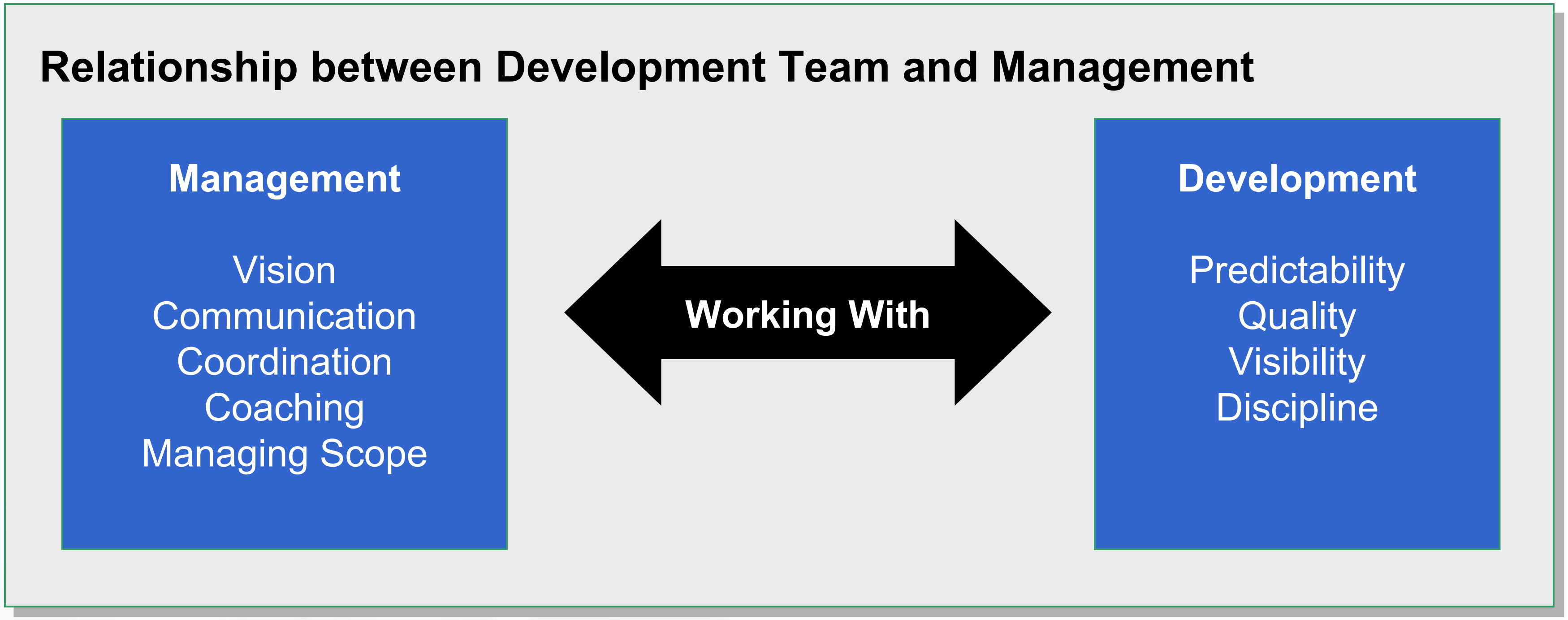
- **Respect the individual** (“work with”)
- **Empower Team** through leadership and coaching
- Proactively **identify ways to maximize value and eliminate waste**
- Use **common work practices**, rhythm and tools
- **Maintain Visibility** - Make all activities, artifacts and risks visible
- **Make Requirements Tangible** – So the developer can test customer can see
- **Decide as late** as possible, **Defer Commitments** to as late as possible
- **Deliver as fast** as possible, Cycle Time Matters!
- **Design Quality In** – Automated testing of requirements, design and development. **Done** = **Unit** and **Acceptance** Tested Components and Features
- **Each team** owns its **predictability**, **progress** and **quality**
  - **Build To Last** - Interfaces are *essential* for independent development and acceptance testing. Components support Platform based products
- **Ship Your Organization** – Align the Teams with the Product Architecture
- **Align compensation** with predictable delivery and quality
- **Continuously improve** your people and practices hence your products and business
  - Encourage and Reward Learning and Innovation

Common Examples of Software Waste

- Lack of common vocabulary, rhythm and tools
- Too many meetings – 1/meetings is a productivity metric
- Lack of Transparency
- Defects in requirements, architecture, design, programs or tests
- Lack of understanding/training – requirements, design, code, test
- Unmanaged supplier, development or requirement risks
- Unnecessary Fire Drills – feature request disguised as a critical defect
- Excessive component repair vs. timely replacement
- Manual Testing
- Unnecessary process artifacts
- Big Analysis, Architecture, Design, excessive dependencies, coupling
- Gold Plating – Requirements, Code or Tests

Agile Commitments

- Successful development requires trust and transparency between customer/management and supplier/development
- Need to foster a “work with” instead of “works for” relationship



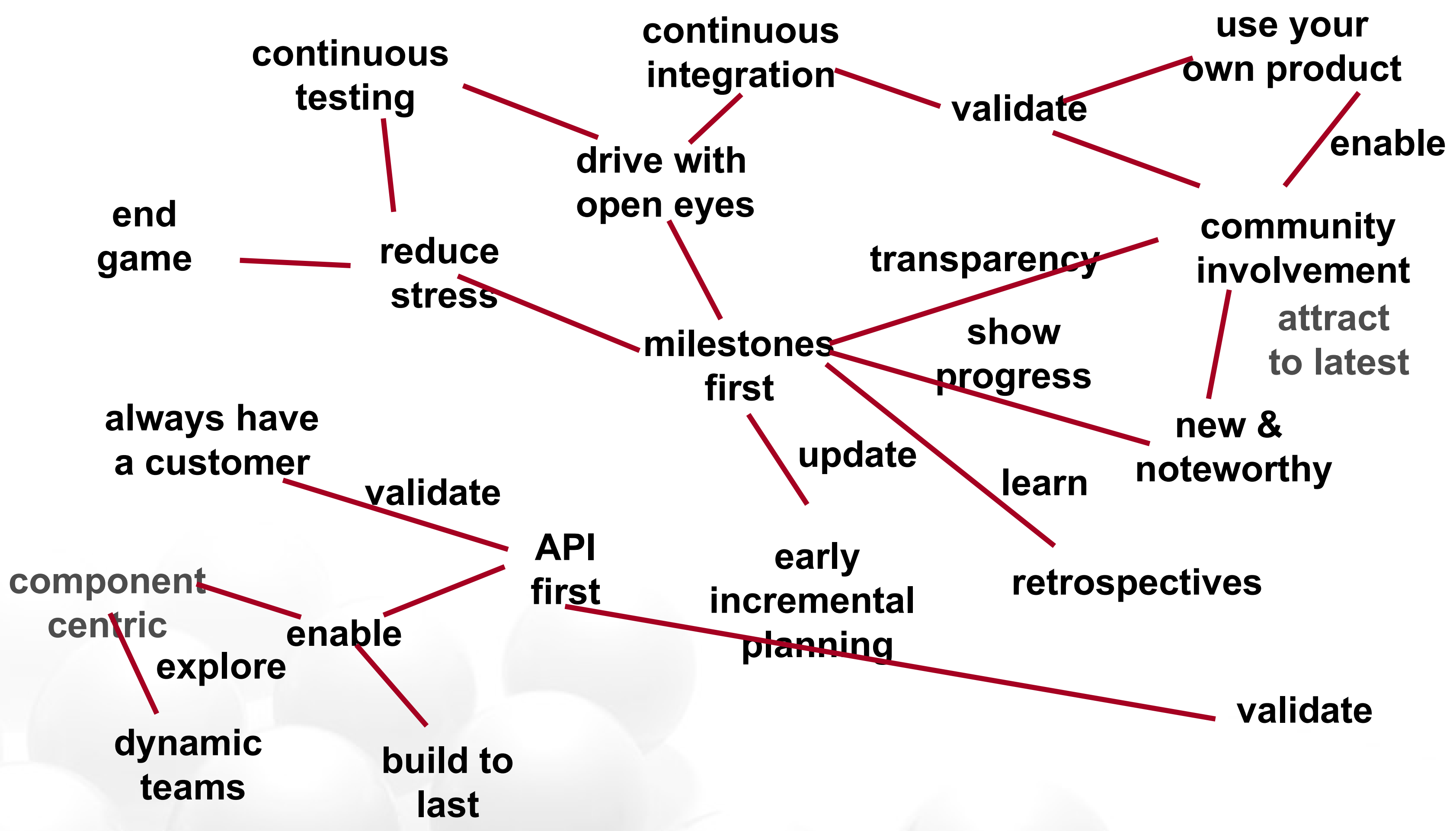
Software is Product Development!

Why should software be different than mechanical or electrical product design?

- Customer Focused
- Requirements Driven
- Built from existing parts – Components and Platforms
- Simulation/Emulation
- Prototype construction
- Prototype evaluation and test
- Development of production product

A Large Scale Lean and Agile Success Story

The IBM initiated Eclipse.Org involves over 1000 developers from multiple companies globally developing software using Lean and Agile practices.



Good Software Comes in 3s

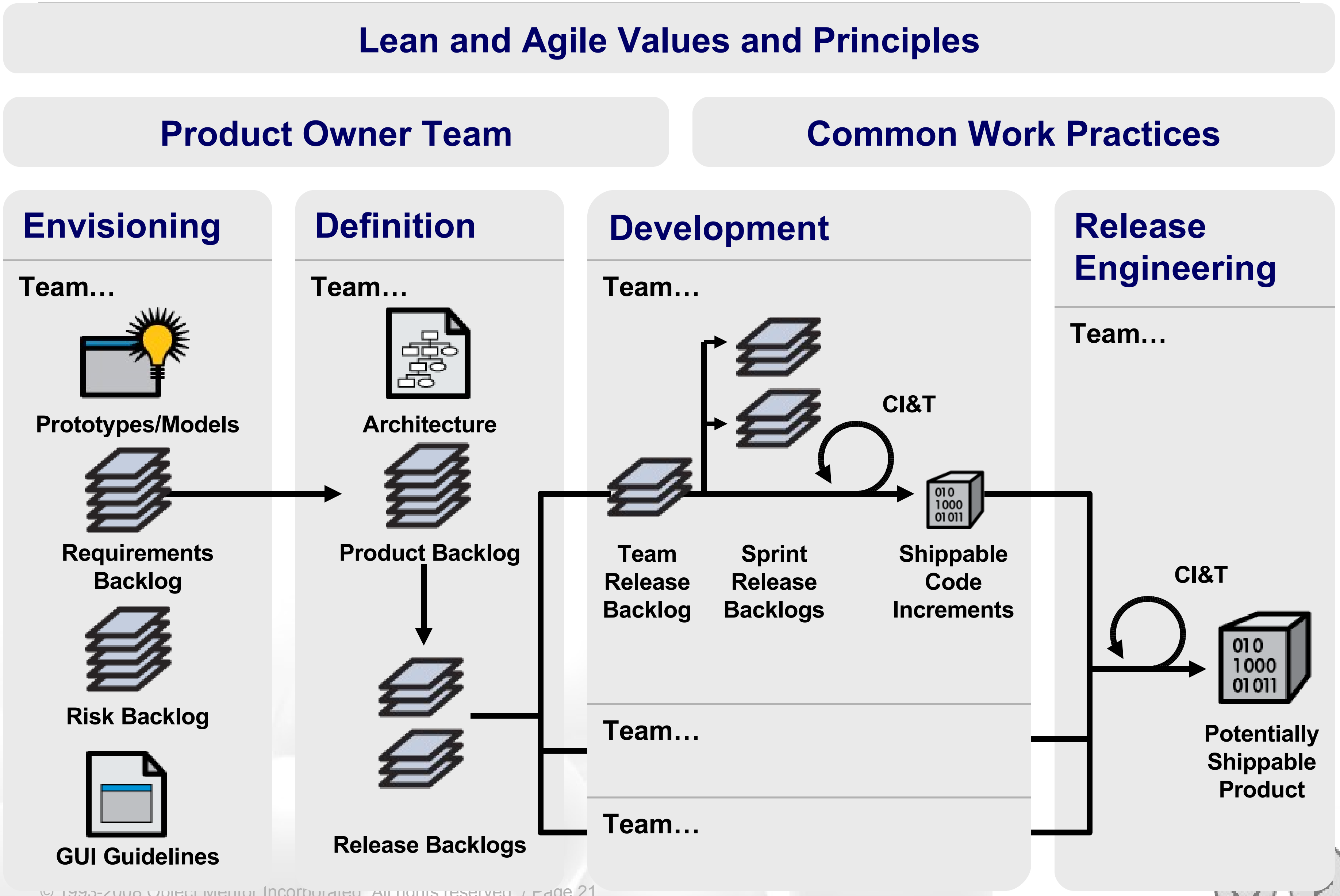
Build It 3 Times; you get it right the third time!

- Once to understand what it is (**Envisioning**)
- Once to understand how it goes together (**Definition)**
- One last time to make it for real (**Development**)
- Make sure everyone shares the big story, has the right infrastructure and the parts make the whole and ship it (**Release Engineering**)

Ship It Three Times and Then Start Re-Design

- 1st Release make it useful and robust
- 2nd Release make it more useful and faster
- 3rd Release make it more useful and smaller
- 4th Release Start Redesigning to avoid Legacy

Large-Scale Lean Development Activities At-A-Glance



Lean Product Team

- Agile values people over processes and tools
- Agile recognizes people are “heart and soul” of the development process
- Product Team is assembled with individuals who represent all of the skill sets required to successfully deliver a software product
- Structure can be easily integrated into any existing corporate model (e.g. staffing, performance objectives, performance reviews)
- Typical team consists of Product Management, Development and Release Engineering, Release Architecture
- Triages the Risk Backlog

Product Team Responsibilities

- Maximize business value that the product delivers to the customer
- Communicates the overall product vision
- Communicates the architectural vision
- Creates, owns and maintains all Product backlogs
- Creates tangible requirements
- Creates use cases and acceptance tests
- Creates and schedules releases
- Creates development teams (Scrums)
- Assigns work items to internal teams and external teams
- Manages third-party suppliers

Common Work Practices and Rhythms

Regardless of the activity, each team is effectively self managed through the use of four key practices

Backlogs
Used to Organize The Work By Value

Sprints
Used to Execute The Work In Small Increments

Continuous Test
Used to Ensure Quality

Metrics
Used to Track and Maintain Visibility

Sprint Rhythm (2 wk)
- Sprint Planning (1) , Daily Stand-ups .., Sprint Retrospective (1)
- Features => Stories => Tasks (1 – 2 days)

Development Rhythm
- Design Acceptance Test, Design Unit Test, Design Code, Build and Test

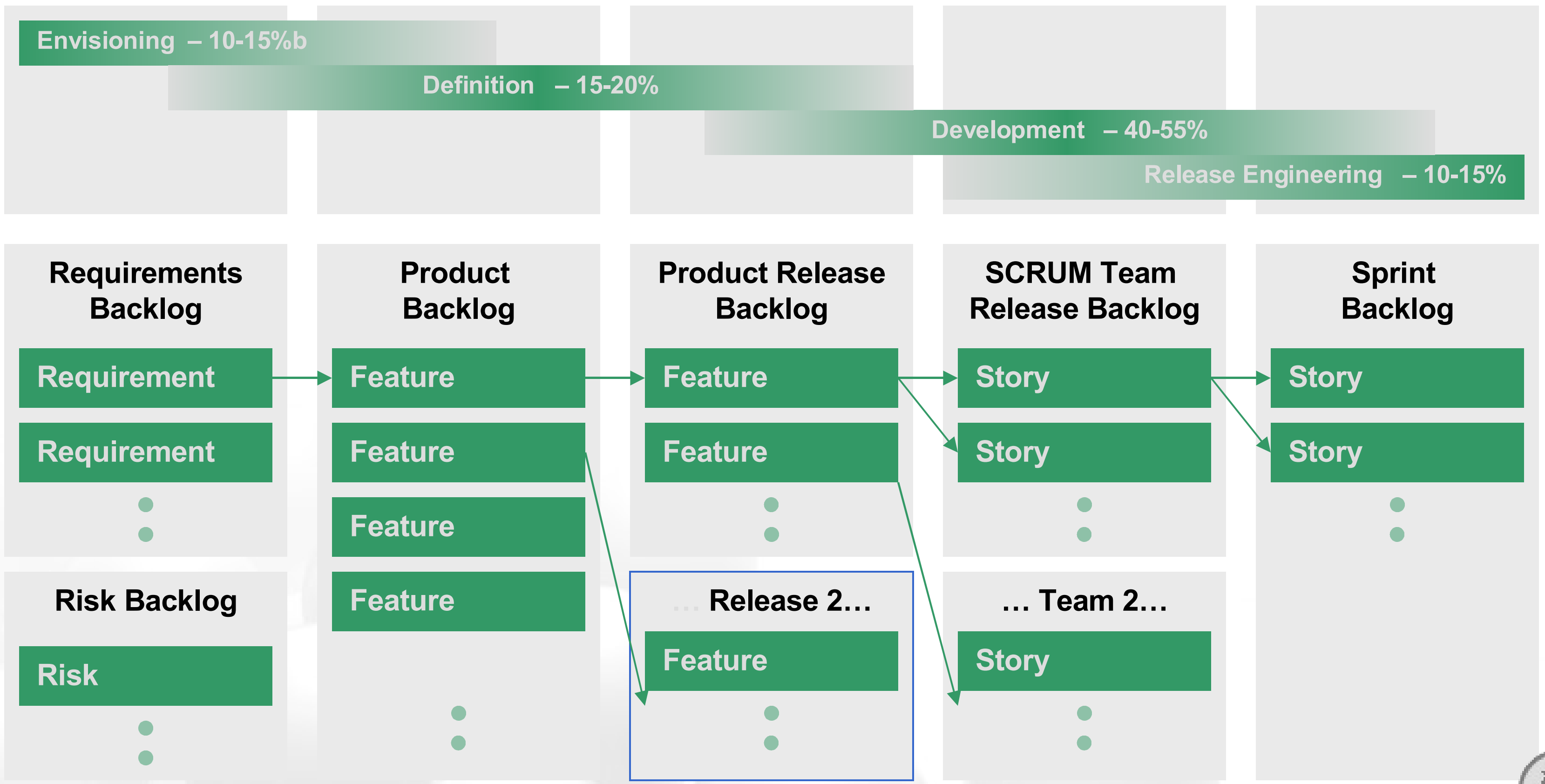
Release and Product Rhythm
- 6 - 8 sprints per internal release; 3 - 4 internal releases per product release
- Envision (3 – 6 m), Definition (1 – 3 m), Development (6 – 12), Freeze (1 – 3)
- Component and Platform Rhythm is 3 – 6 months ahead of Product Release

Backlogs

- Requirements Backlog
  - A list of *must have*, *should have* and *nice-to-have* customer requirements
- Risk Backlog
  - List of potential product, market and development risks
- Product Backlog
  - List of prioritized features with estimated times (all releases, all teams)
- Product Release Backlog
  - List of prioritized features with estimated times (one release, all teams)
- Scrum Team Release Backlog
  - List of stories to be completed by the team in one release (one release, one team)
- Scrum Team Sprint Backlog
  - List of stories to be completed by the team in one sprint (one sprint, one team)

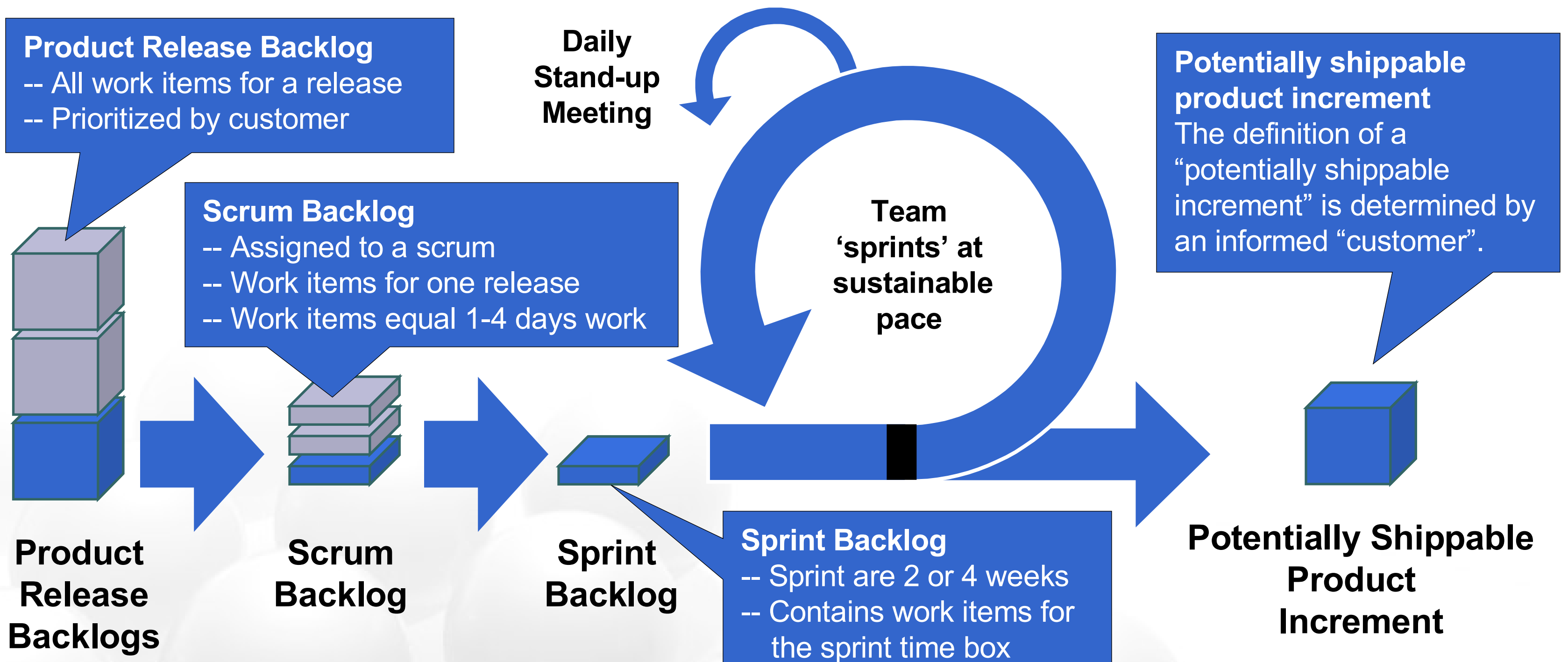
Backlogs are Used To Organize The Work

- BACKLOG is a list of work items – a few backlogs cover life cycle
- All work items must be entered into one of the backlogs
- All work items are prioritized by customer value
- Backlogs are reviewed and triaged as necessary



Scrums used to organize work

- Product teams consist of 4 - 20 people called SCRUMs
- Scrum is guided and facilitated by leader acting as coach/facilitator
- Scrum has the whole team needed to execute their backlog
- 2 week Sprints provide a common rhythm for whole life cycle
- All work including defect repair, training is in the backlog



Product Release Backlog
-- All work items for a release
-- Prioritized by customer
Daily Stand-up Meeting
Scrum Backlog
-- Assigned to a scrum
-- Work items for one release
-- Work items equal 1-4 days work
Team 'sprints' at sustainable pace
Potentially shippable product increment
The definition of a "potentially shippable increment" is determined by an informed "customer".
Sprint Backlog
-- Sprint are 2 or 4 weeks
-- Contains work items for the sprint time box
Product Release Backlogs
Scrum Backlog
Sprint Backlog
Potentially Shippable Product Increment

Schwaber, B

Types of Scrums

- Scrums are used for all activities in the product life cycle
- Envisioning Scrums
  - Collect and analyze market, customer, product, and technology requirements
  - Develops proof-of-concept and vision prototypes
  - Verifies prototypes with customers
  - Capture requirements in a Requirements Backlog
  - Capture risks in a Risk Backlog
- Definition Scrums
  - Translate Requirements into Features by creating User Cases and Acceptance Tests
  - Define the Architecture and Component Breakdown Structure for the product
  - Sort Features into the Product Backlog and Product Release Backlogs
  - Speculative Estimates
- Development and Release Engineering Scrums
  - Translate Features in Scrum Release Backlog into Stories plus associated unit & acceptance tests
  - Stories include estimates that are owned by the team and refined over time
  - Stories are prioritized based on their associated business value
  - Converts Stories into Working Code

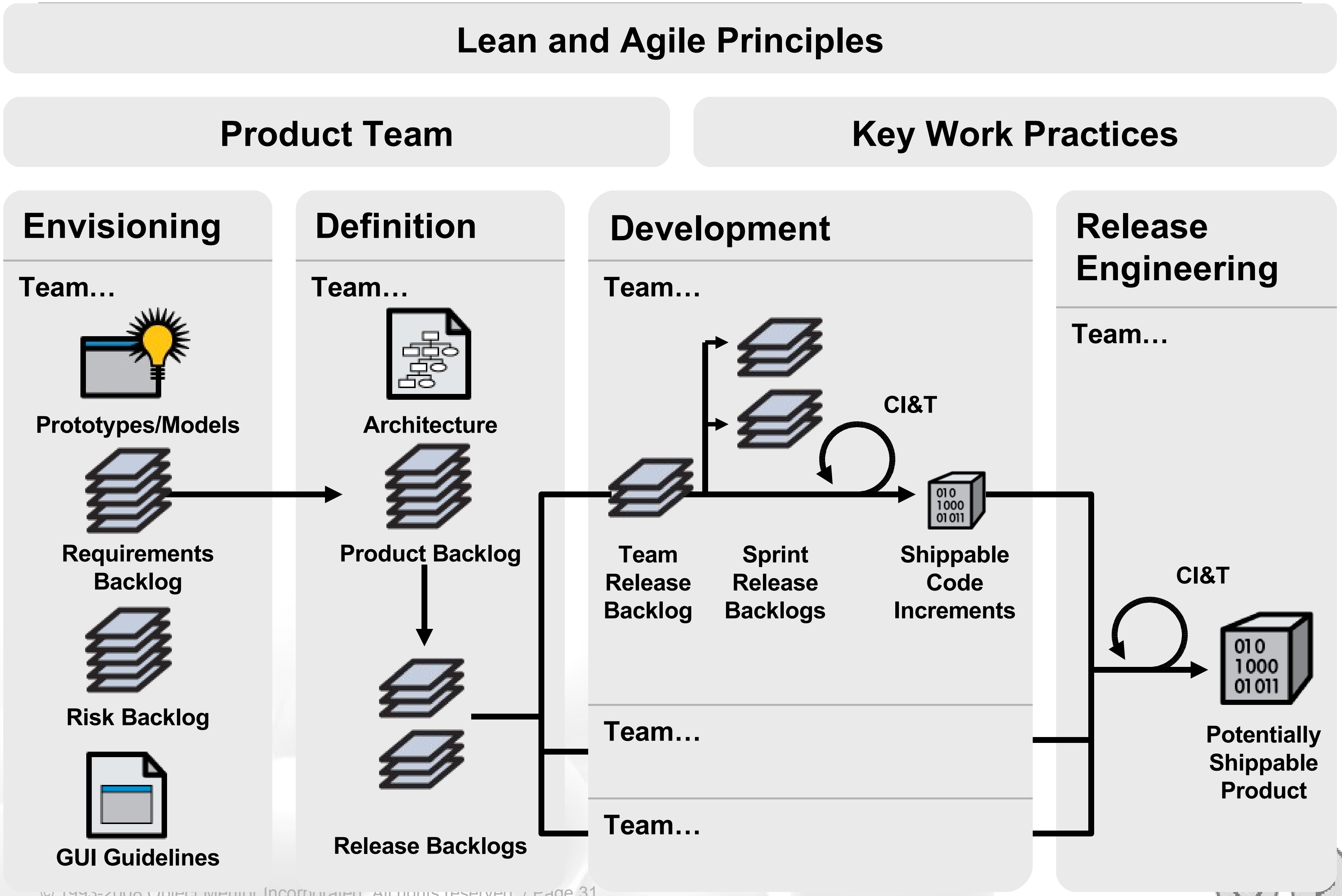
Measurements are used to Track Progress

- Project management not a separate entity in an Agile team
- Project management owned collectively by team
  - Team members contribute to the estimating process and commit to the deliverables
  - Team members own estimates, schedules and deliverables
- Team responsible for maintaining visibility
  - Team uses wall charts or online charts to make status visible to all stakeholders
  - Metrics derived automatically and continuously from the build process
  - Metrics are used to assess performance of the process, not people
- Typical Measurements
  - Efficiency: Are resources being optimally deployed?
  - Progress: Is the project on track for time and budget?
  - Productivity: How much code per unit of labor?
  - Rhythm or heartbeat: How active is the project day-to- day?
  - Quality: How good is the software being produced?

Team-Oriented Metrics

- Features/Stories Delivered (Velocity)
- Features/Stories Planned (Estimated Velocity)
- Features/Stories Remaining (Burn Down)
- Unit and Acceptance Tests Run (Progress)
- Backlog adds, changes, deletions (Feature Flux and Creep)
- Continuous Check-ins and Builds (Rhythm)
- Classes and Methods adds, changes, deletions (Impact on Code Base)
- Code Coverage due to UTs and ATs (Quality)
- Defects and Defect Density (Quality)

Lean and Agile For Large-Scale Development



Envisioning and Definition “BIG SPRINT 0”

The initial sprint is often called sprint 0 and it is used to as a exploratory/high level planning activity which seeks to explore, understand, swag, identify dependencies using practices such as:

- Spikes!
- Thin Slices (Tracer Bullets)

Unfortunately in large scale software new features or components contain so many unknowns and so much risk that development teams have little chance of delivering these in a release time box!

What Is Envisioning?

Envisioning is the process of developing a clear product vision and roadmap…

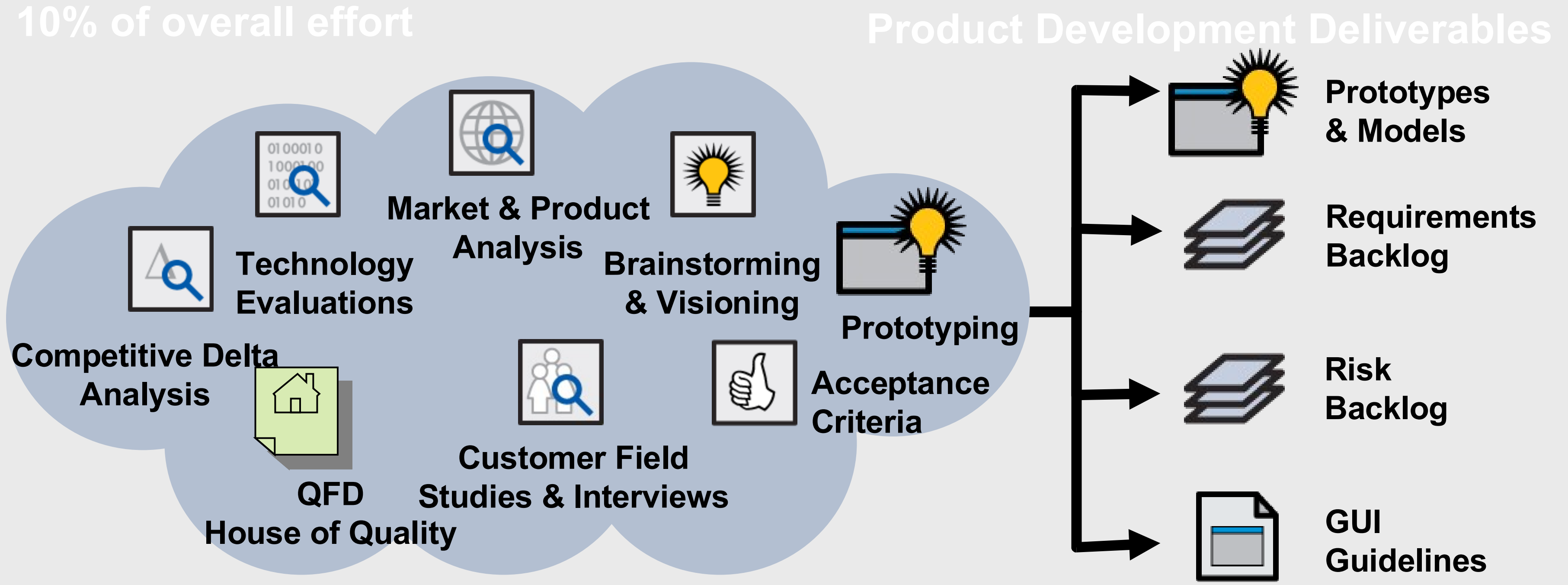
through consultation with users and choosers…

to ensure that we are building
the right product using the right technology
for the right market.

Envisioning

Envisioning ensures that you target the right features, technology and markets through customer, market, and technology research

10% of overall effort


Product Development Deliverables
Technology Evaluations
Market & Product Analysis
Brainstorming & Visioning
Prototyping
Competitive Delta Analysis
Acceptance Criteria
Customer Field Studies & Interviews
QFD House of Quality
Prototypes & Models
Requirements Backlog
Risk Backlog
GUI Guidelines

Practices

■ Brainstorming & visioning ■ Competitive analysis (SWOT) ■ Delta analysis ■ QFD ■ Customer studies ■ Hardware, platform & component evals ■ Prototyping/modeling

Deliverables

■ Requirements backlog ■ Risk backlog ■ Analysis & Verification Reports ■ Prototypes/Models ■ Look-and-Feel Guidelines

Product Vision – Voice of The Customer

Essence = Vision = The Really Big Story

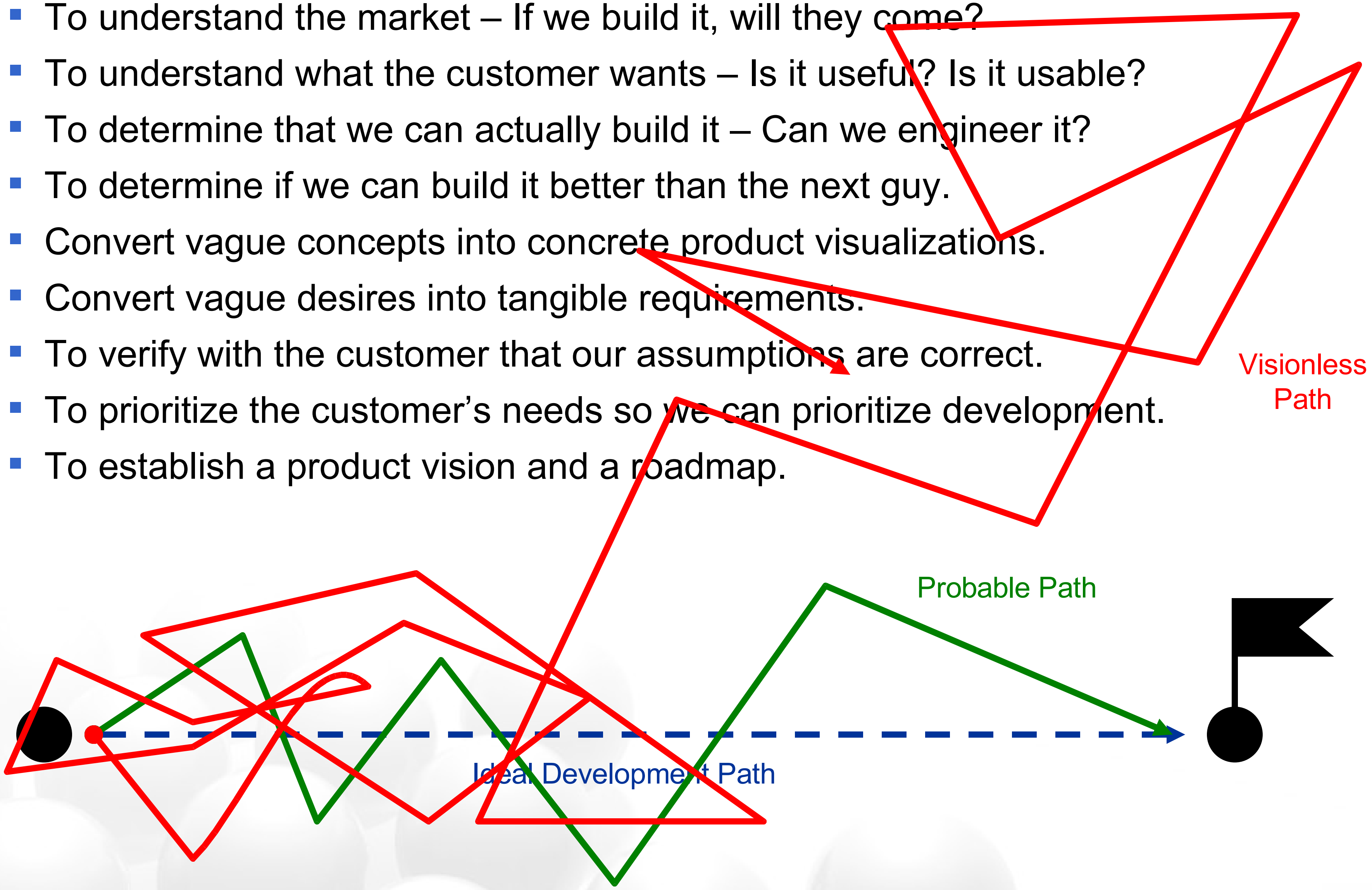
- Customers, Business Analysts, Product Owners and Developers need a simple story (s) which provides a shared vision of what is being developed
- Story Telling carries the essence, humans render the details according to the story, be it complex software, knowledge management or animated films
- A Story is elaborated by
  - Glossary to define terminology
  - Customer stories in the form of narrative and/or video
  - Customer stories in terms of features relative to current or competitors product or our legacy product – is like, is not like
  - Envisioning, House of Quality (QFD) …
  - Domain Model(s) to show essential relationships
  - Prototypes, Simulations

The Requirements Tangibility Imperative!

- Requirements MUST be UNDERSTANDABLE!
  - Need the conversation, not just the text
  - Tangibility is directly correlated to domain knowledge and understanding the customers stories.
  - Acceptance Criteria make requirements testable
- Requirements Must Have Business Value
  - Need to have a business value, and an estimated effort when defined, revised for release backlog
- Requirements MUST be TESTABLE!
  - TDD Acceptance Tests
- Non-Functional Requirements ARE Requirements – TCO, ilities…
  - Need TDD Acceptance Tests as well

So… Why Do We Envision?

- To understand the market – If we build it, will they come?
- To understand what the customer wants – Is it useful? Is it usable?
- To determine that we can actually build it – Can we engineer it?
- To determine if we can build it better than the next guy.
- Convert vague concepts into concrete product visualizations.
- Convert vague desires into tangible requirements.
- To verify with the customer that our assumptions are correct.
- To prioritize the customer's needs so we can prioritize development.
- To establish a product vision and a roadmap.



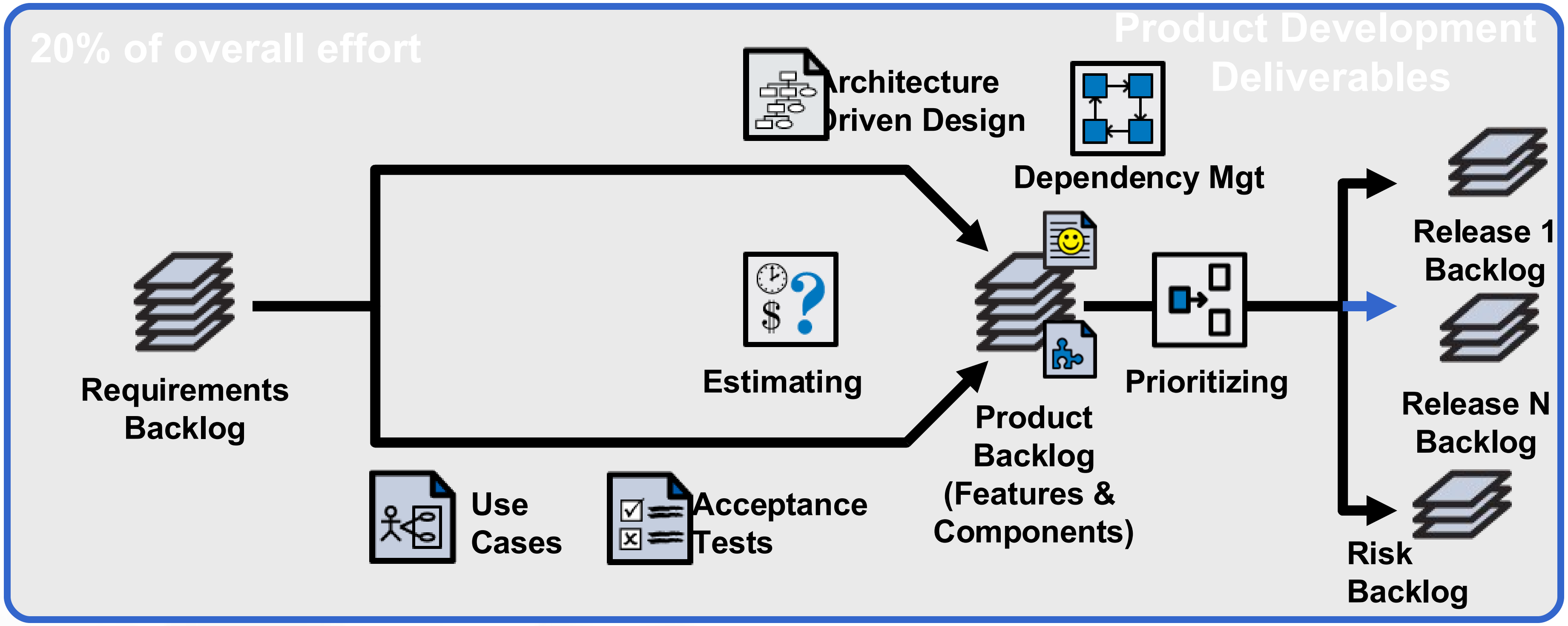
What Is Definition?

Definition is the process of ensuring that the product can be composed of its constituent parts and allocated the backlogs to appropriate teams

- Built to last through the use of appropriate internal and external components all of which have well defined interfaces
- Features are mapped across the architecture and allocated to feature and component teams
- Features are roughly sized/estimated as an input to downstream development teams

Definition

Definition involves transforming requirements into features, creating a high-level architecture and determining an initial work allocation for teams.



Practices

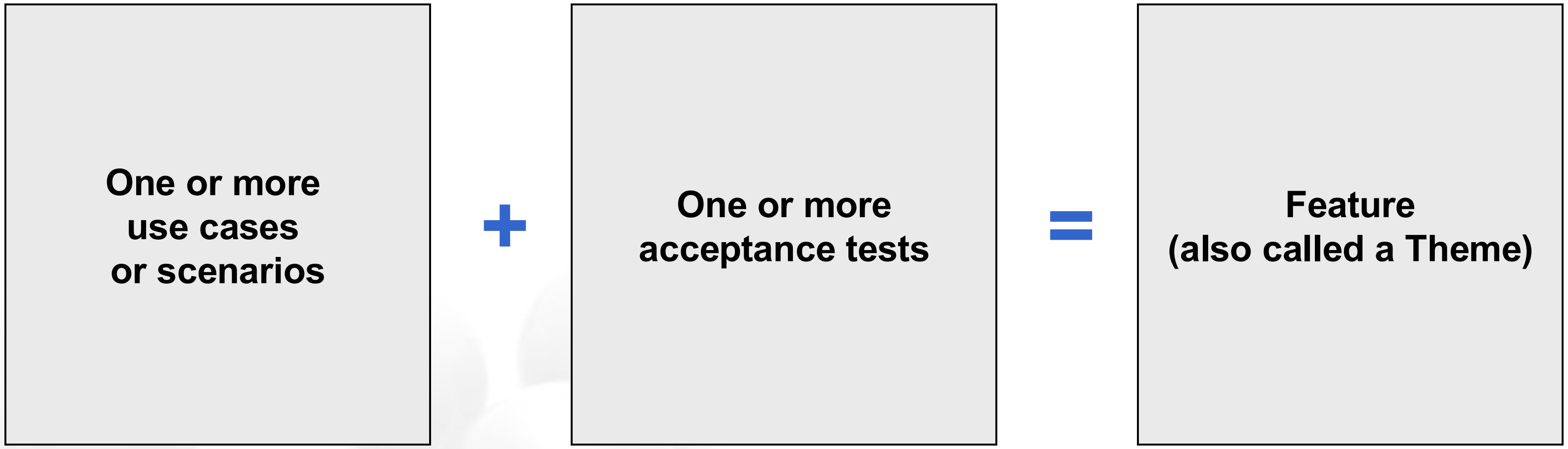
■ Feature Definition ■ Estimating ■ Dependency Management ■ Vendor Management ■ TDD ■ Architecture-Driven Design ■ Component & Feature Break Downs

Deliverables

■ Feature docs ■ Architecture docs ■ Product, component and feature backlogs ■ Risk backlog

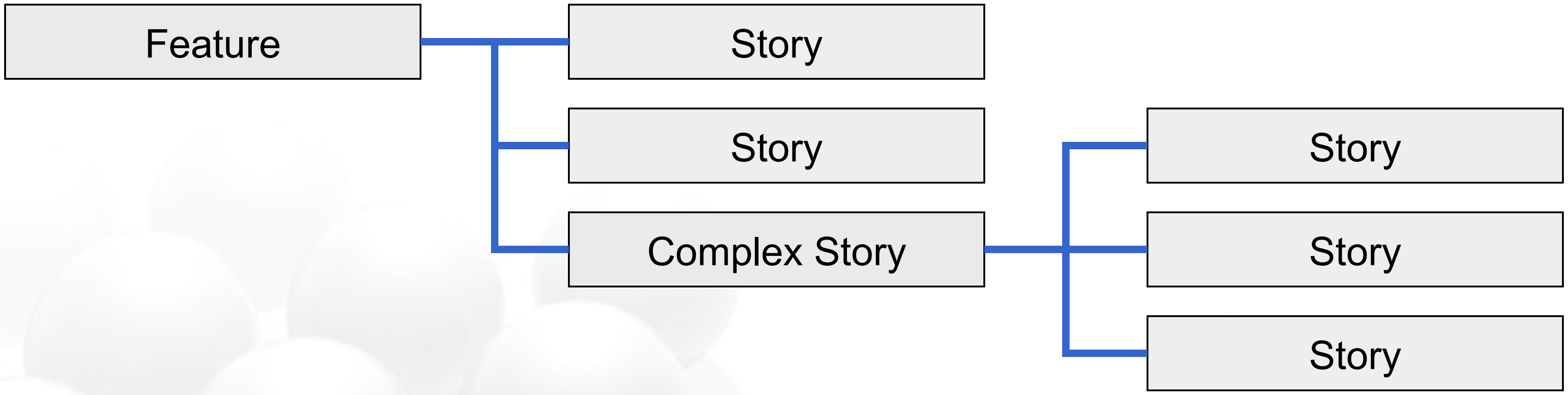
Features

- A feature is a customer requirement that has been translated into something that can be implemented by the development team
- Typically consists of one or more use cases and acceptance tests
- The use cases describe the feature’s operation
- The acceptance test describes the acceptable outcomes


One or more use cases or scenarios
+
One or more acceptance tests
=
Feature (also called a Theme)

Stories

- Stories are specific descriptions of the work to be done
- A feature use case will generally result in many stories
- A story represents one path through a use case
- A story should be able to be implemented in a single sprint by one or two developers in a few days. If this is not possible, it should be split into smaller stories.
- Stories have associated unit and acceptance tests
- Stories have estimates

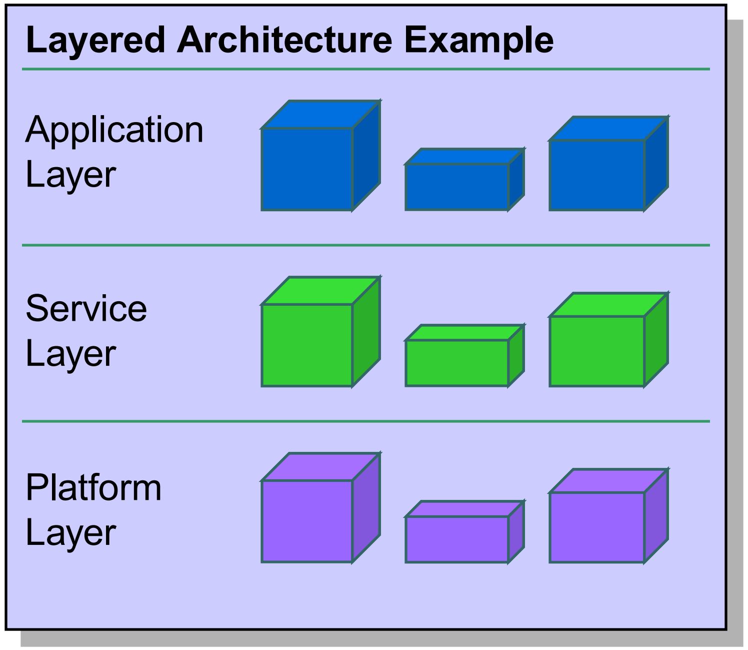


Architecture Driven Design (ADD) - *API First*

- Object Orientation [Simula 67] is an approach to software architecture, design and implementation which is based on building simulation models of the system. These models are expressed in code.
- Architecture Driven Design Benefits
  - Compartmentalizes the work into logical divisions
  - Creates stability within the individual parts of the product
  - Establishes ownership and accountability for individual parts
  - Communicates the essence of the product more easily
  - Provides a single expression of the system
  - Model can be version managed easily
  - Model can evolve
- Expressed through a layered architectural approach

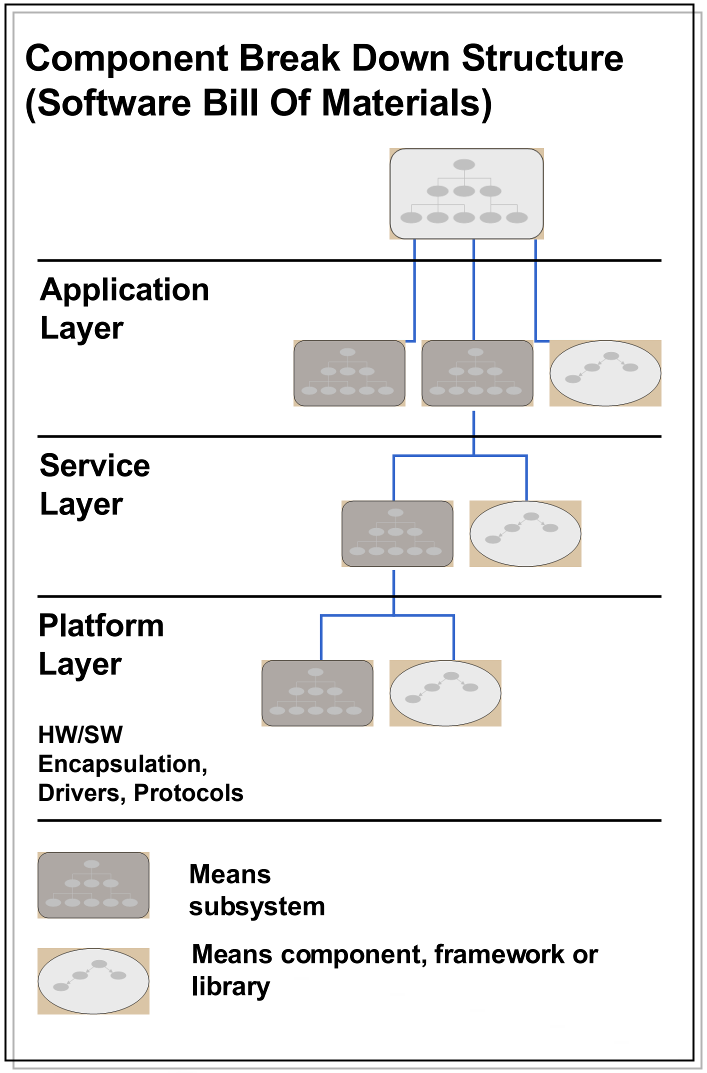
Layered Architectural Model

- The layered model organizes the architecture into 3–5 layers which communicate through an API.
- This model is essential for the use of third-party platform or applications and often is heavily influenced by the use of third-party offerings.
- Agile organizes work into feature breakdown structures and component breakdown structures.
- These structures are organize, plan and allocate resources into ***Feature Teams, Component Teams*** and ***Platform Teams***.
- The structures are also used to organize, plan and allocate Features and Releases.



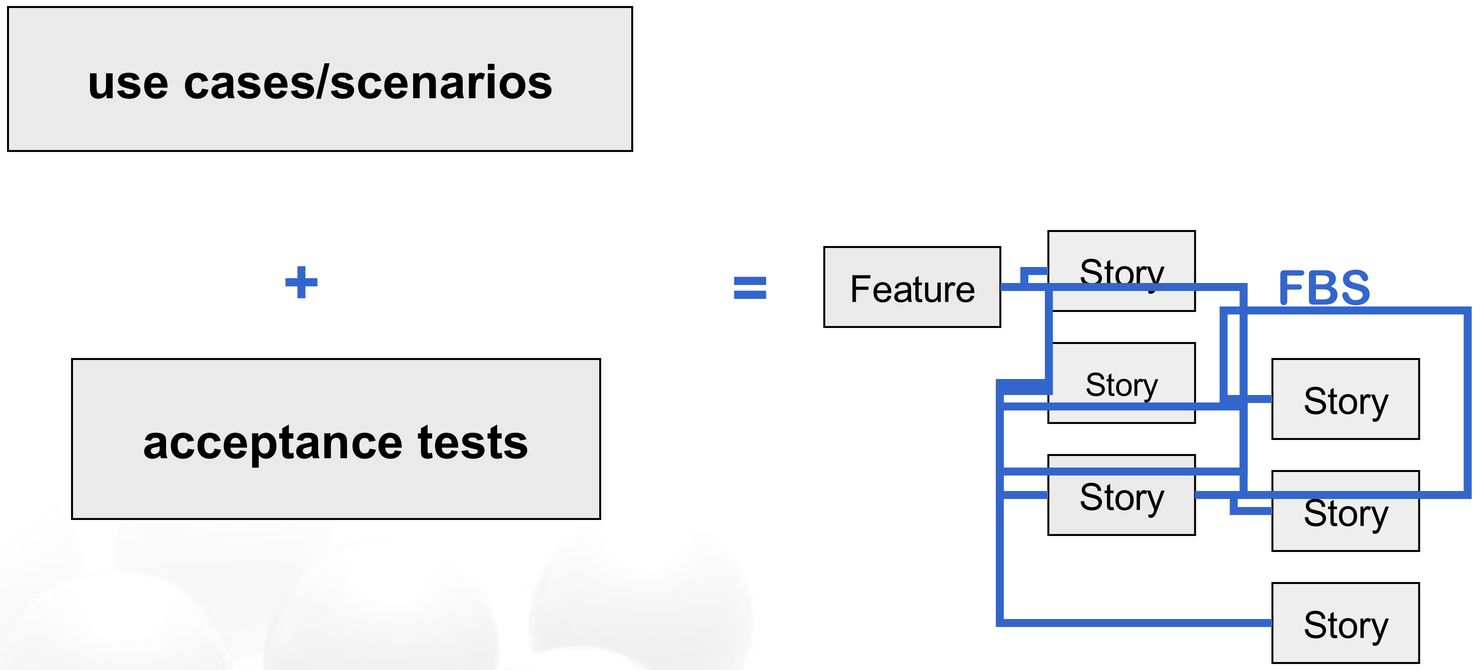
Component Breakdown Structure (CBS)

- The CBS provides a software bill of materials for the organization calling out new, modified and existing software components.
- The presence of a CBS is the hall mark of a Product Line Architecture and enables building platforms and associated components and products.
- The CBS is developed bottom up often as a core platform or framework on which to build other products.
- Ideally one would just configure the components and deliver a product but the reality is that features need additional code hence applications.


Component Break Down Structure
(Software Bill Of Materials)
Application Layer
Service Layer
Platform Layer
HW/SW Encapsulation, Drivers, Protocols
Means subsystem
Means component, framework or library

Feature Breakdown Structure (FBS)

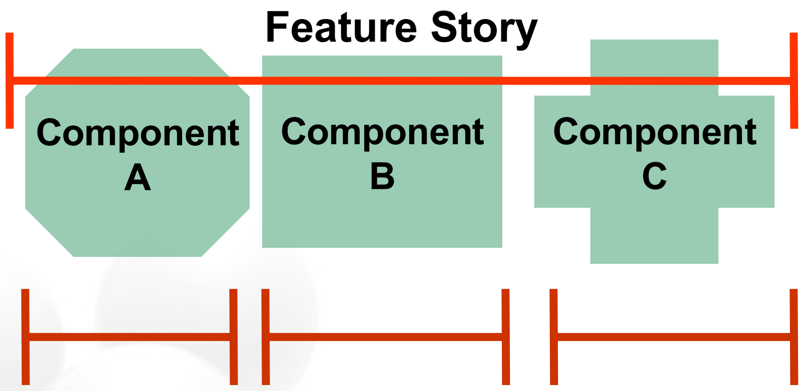
- Features are organized in an FBS
- The FBS relates features to other features (feature dependencies)
- The FBS also cross cuts components (component dependencies)
- Dependencies are captured in a Dependency Structure Matrix



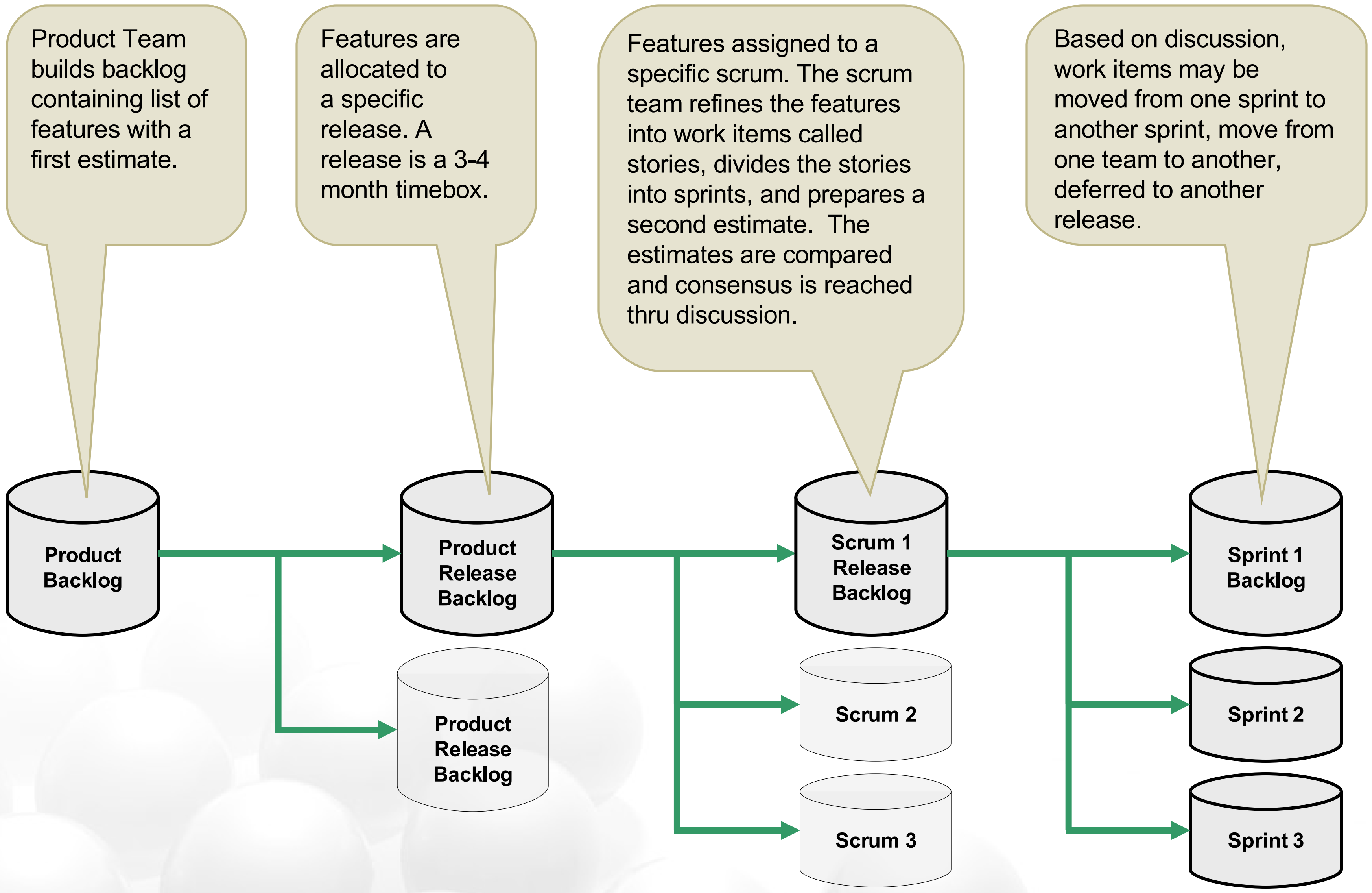
Feature and Component Teams and Backlogs

- There is a natural tension between components and features a critical part of definition is the identification of dependencies and the allocation of work to appropriate teams.
- Feature cross-cuts components hence require either product specific feature development or additional component development to support that feature. Often it is a combination of both with the need to move feature specific extensions into components in a later release.

Feature Backlogs and Component Backlogs



Product and Release Planning


Product Team builds backlog containing list of features with a first estimate.
Features are allocated to a specific release. A release is a 3-4 month timebox.
Features assigned to a specific scrum. The scrum team refines the features into work items called stories, divides the stories into sprints, and prepares a second estimate. The estimates are compared and consensus is reached thru discussion.
Based on discussion, work items may be moved from one sprint to another sprint, move from one team to another, deferred to another release.
Product Backlog
Product Release Backlog
Product Release Backlog
Scrum 1 Release Backlog
Scrum 2
Scrum 3
Sprint 1 Backlog
Sprint 2
Sprint 3

Product Estimates

- Estimates are “negotiated”, not “assigned”
- Two-stage iterative estimating process conducted over 4-8 weeks
- Strong emphasis on the collective ownership

Stage One: Estimate Preparation

Definition Team Release Estimates

The Product Team makes initial estimates for each feature in each of the Product Release Backlogs.

Development Team Release Estimates
Each SCRUM team make estimates for each feature in their Scrum Release Backlog.

Stage Two: Estimate Convergence

The two sets of estimates are compared. Consensus is reached through dialog between developers, product release engineers, architects, customers and owners.

Based on the discussions, features may be shuffled, re-allocated, and re-prioritized.

Based on the discussions, development schedules are updated and product release dates are set.

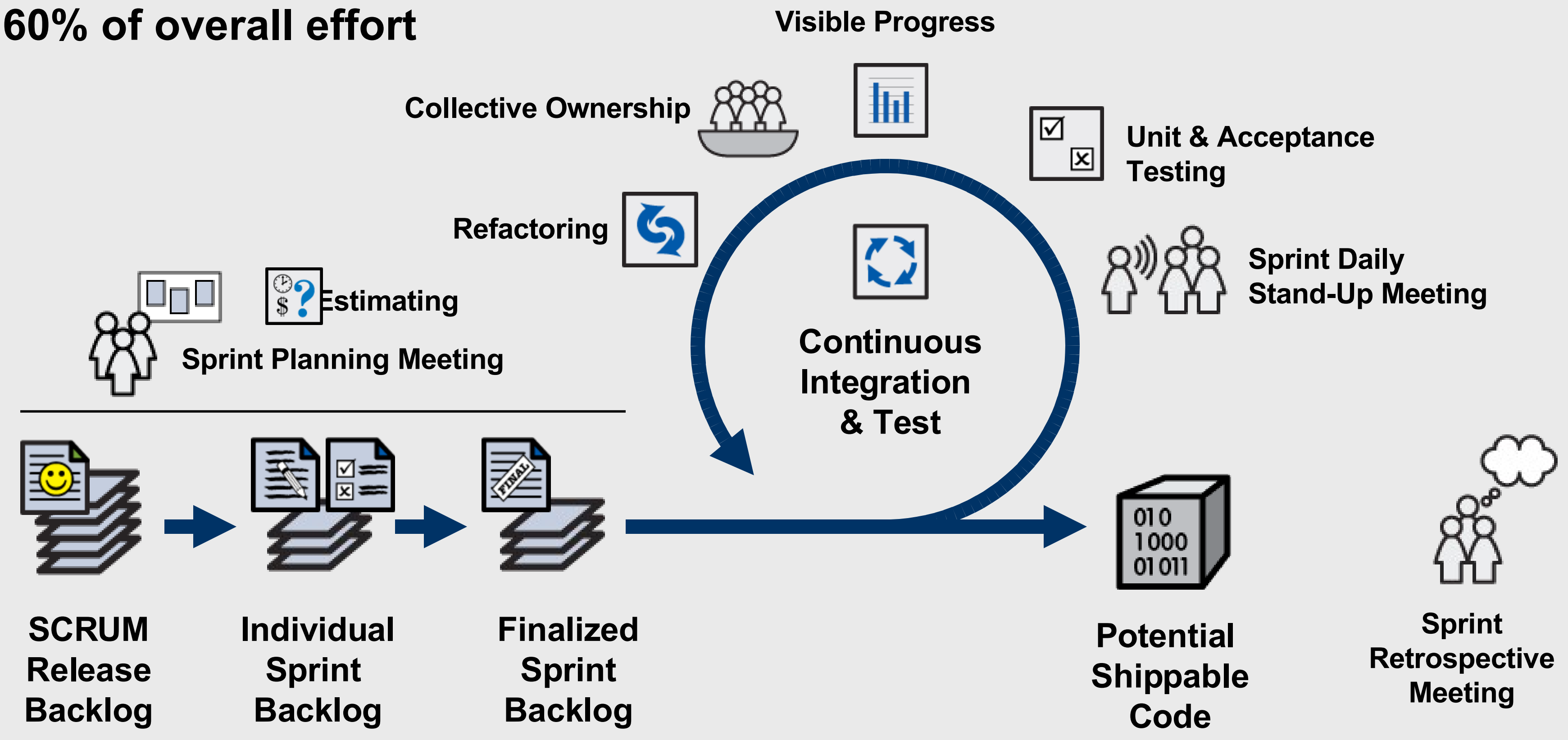
Estimation and Planning –The Best Wrong Answer

- Estimates Are Owned By Team
  - (e.g. multiple estimates by multiple people to encourage discussion)
- Use Relative Estimating Techniques
  - (e.g. this story is half as difficult as that story so it will take half the time)
- Use Range or 3 Point Estimates
  - (e.g. use ranges or)
- Use Multiple Units of Measurement
  - (e.g. multiple units – ideal days, story points, classes/methods – to improve accuracy)
- Learn From Previous Estimating Experience
  - (e.g. comparing previous estimates with previous actual outcomes)
- Learn from Expert Experience
  - (e.g. If you have never used J2EE bring in someone who has!)
- Improve Estimates Using Wide band Delphi (Planning Poker)
- Cross Check with Activity Based Estimates
- Estimate Resources with Activity Based Estimates

Development

Development transforms requirements into tested code.

60% of overall effort



Practices

■ Small Releases ■ Simple Design ■ Collective Ownership
■ Refactoring ■ Continuous Integration and Test
■ Test-Driven Development ■ Architecture-Driven Design

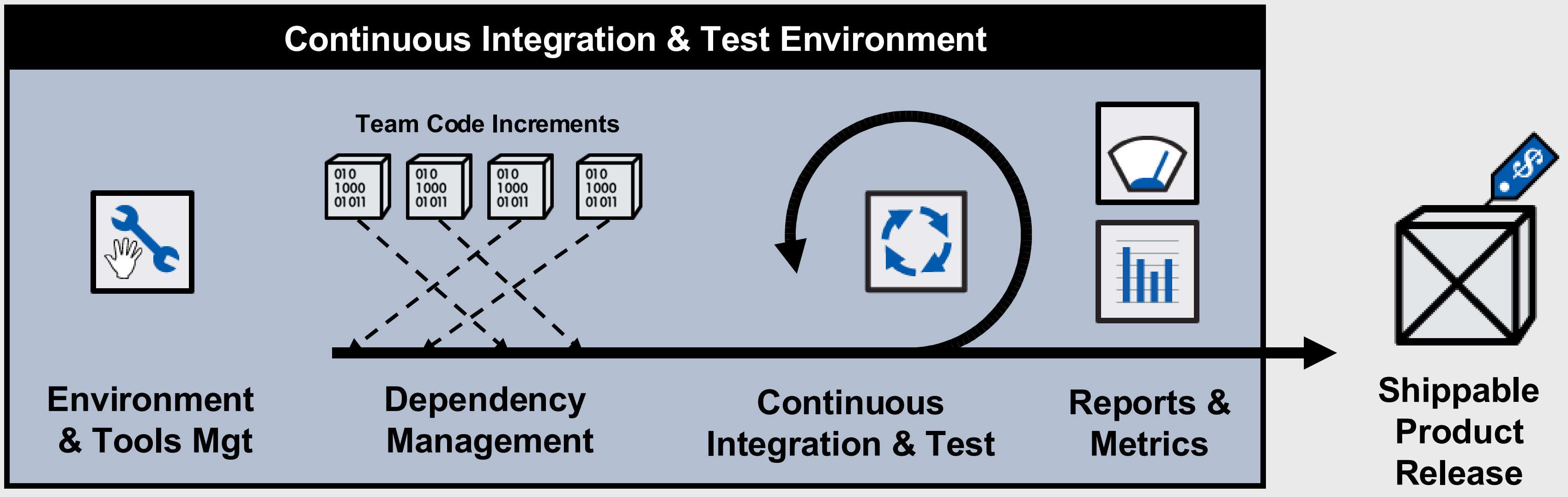
Deliverables

■ Robust code
■ Status reports
■ Readiness

Release Engineering

Release Engineering ensures correct integration of disparate code units into a cohesive product release. Performed concurrently with development.

10% of overall effort



Continuous Integration & Test Environment
Team Code Increments
010 1000 01011
Environment & Tools Mgt
Dependency Management
Continuous Integration & Test
Reports & Metrics
Shippable Product Release

Practices

■ Dependency Management ■ Vendor Management ■ Test-Driven Development ■ Architecture-Driven Design ■ Continuous Integration and Test

Deliverables

■ Continuous Build and Test Environment ■ Stable, Robust Code ■ Progress Reports & Metrics

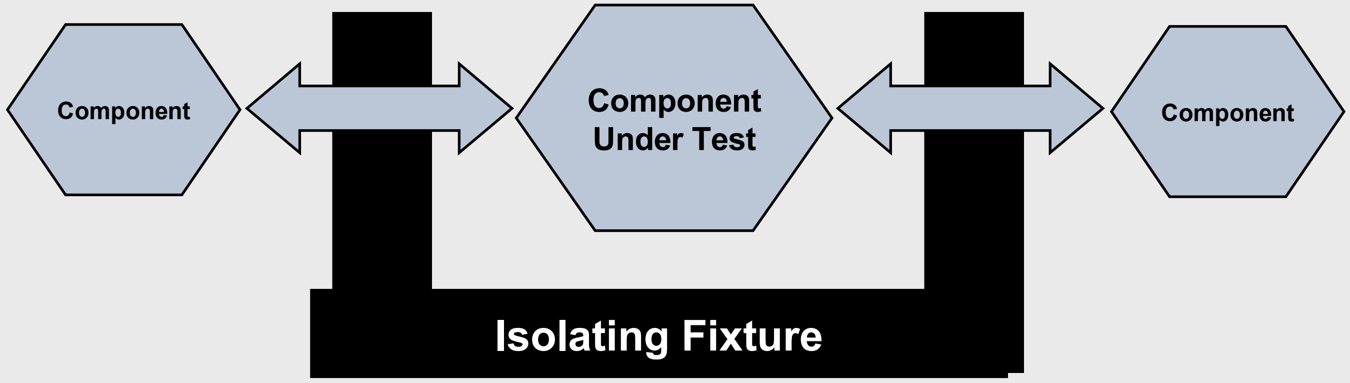
Release Engineering - *Start With a Winning End Game*

- Continuous Integration and Test Infrastructure
- Development Tools Infrastructure
- Transparency – Electronic Information Walls and Dashboards
- Dependency Management
- Readiness and Quality
- Documentation
- International Language and Accessibility
- Manual Acceptance and Test Automation
- Freeze, Thaw and Fix Management

Dependency Management

- Dependency identification is a critical success factor because teams can become deadlocked waiting on code from other teams
- Dependency management is responsibility of Release Engineering on behalf of the Product Team
- Thin Slices for each Feature reduce surprises
- Can be reduced by planning, sequencing and test-driven development
- Individual and pair wise component testing reduces integration problems
- Integration sprints used to ensure features come together properly

Test fixtures are used to support parallel code development by isolating code components from one another for unit testing



Architecture Dependency Models

Dependency Model = DSM + Design Rules*

What is a Dependency Model?

- An ordered hierarchical decomposition of the System into its Subsystems and their Components (modules)
- Display of current dependencies
- Design Rules for allowable dependencies between Subsystems
    - Enforce Layering
    - Enforce Encapsulations
    - Keep Components Independent

** DSM – Dependency Structure Matrix*

Just Enough Documentation

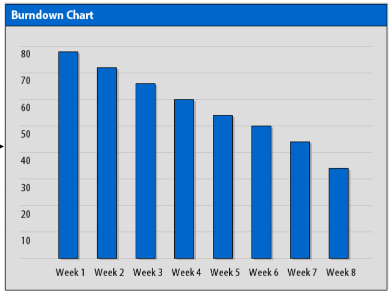
- Literate Programming for code artifacts
- Wiki or equivalent for non code artifacts
- Product Documentation derived from conversations, Envisioning, Acceptance Tests and UI artifacts in the code
- Technical writers typically lag the development team by one or two sprints sprint
- Release Engineering responsible for ensuring documentation acceptance

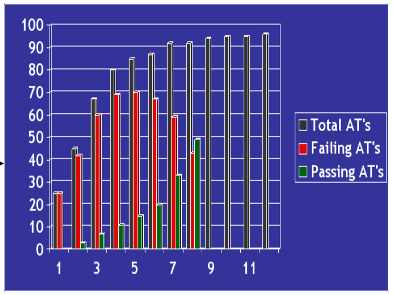
Establish A Product Development Dashboard

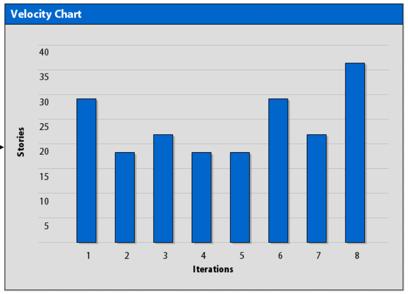
Automate the linkage between features/stories, code and tests

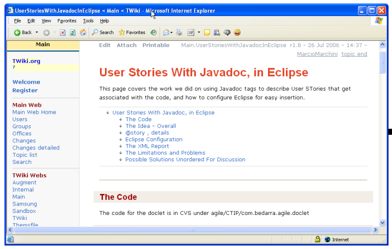
Measure true progress based on the state of the continuous build

Produce AT, UT, Velocity and Burndown Charts Automatically









UserStoriesWithJavadocInEclipse < Main < TWiki - Microsoft Internet Explorer

Edit Attach Printable Main.UserStoriesWithJavadocInEclipse r1.8 - 26 Jul 2006 - 14:37 - MarcioMarchini topic end

Main

TWiki.org

Welcome
Register

Main Web
Main Web Home
Users
Groups
Offices
Changes
Changes detailed
Topic list
Search

TWiki Webs
Augment
Internal
Main
Samsung
Sandbox
TWiki
Themsfile

User Stories With Javadoc, in Eclipse

This page covers the work we did on using Javadoc tags to describe User STories that get associated with the code, and how to configure Eclipse for easy insertion.

- User Stories With Javadoc, in Eclipse
  - The Code
  - The Idea - Overall
  - @story , details
  - Eclipse Configuration
  - The XML Report
  - The Limitations and Problems
  - Possible Solutions Unordered For Discussion

The Code

The code for the doclet is in CVS under agile/CTIP/com.bedarra.agile.doclet

Load wiki with requirements, features, use cases, stories, acceptance tests and unit tests

Link code in library to stories in wiki

See http://www.jot.fm/issues/issue_2007_03/column4

Social Engineering

Common Culture

- Values
- Vision
- Standardize Vocabulary and Practices e.g. points vs. ideal days…
- Spread Experienced People Across Teams/Locations
- Use Playing Coaches to share vision and experiences
- Common Tools
- Empower Distributed Teams – VOIP, IM, electronic White Boards
- Provide company wide visibility – publish charts to web
- Peer Reviews, Technical Seminars…

Align Individual/Team Compensation with Desired Behavior

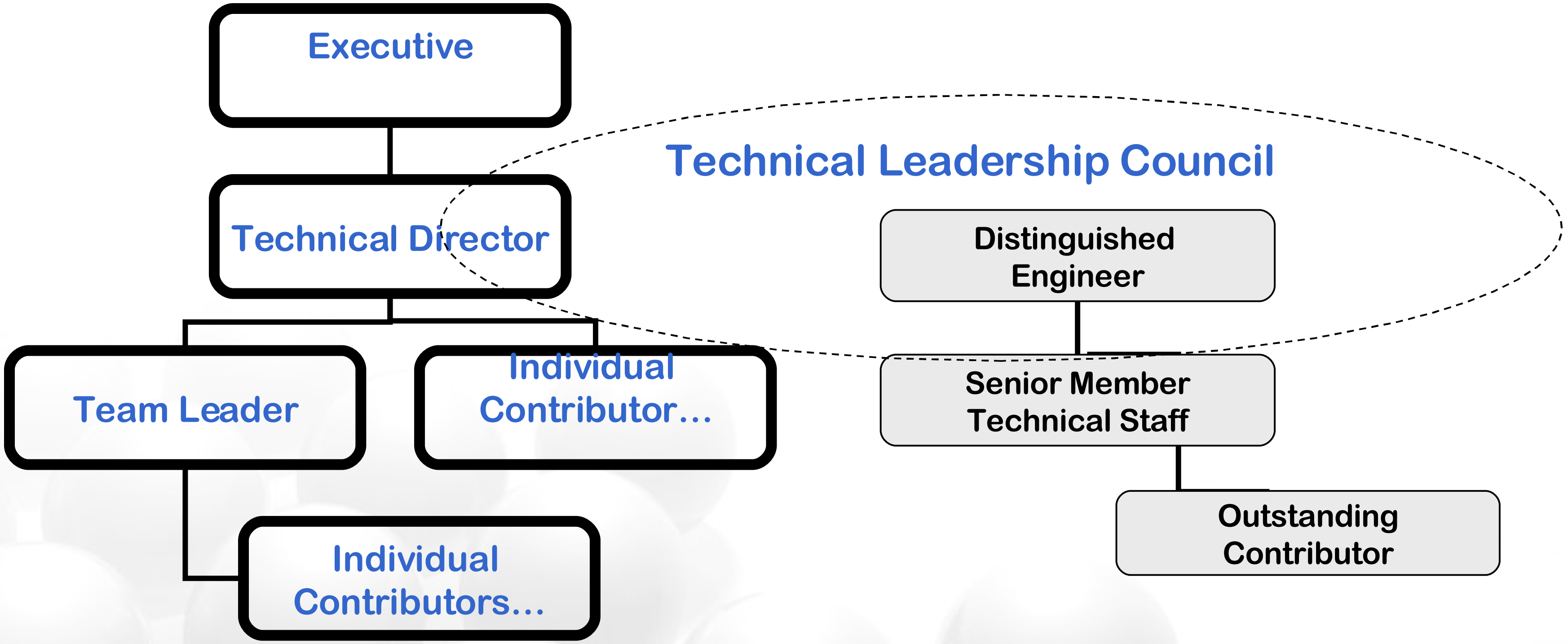
- Predictable Delivery
- Quality Delivery
- Teamwork
- Early Problem Identification and Resolution

Multi-site Development

- Stop using words like Remote, Headquarters learn their names! Keep their maps, calendars and pictures on your walls!
- Co-location is clearly ideal, but not reality in today's large scale software development.
- Open Source Development clearly demonstrates that co-location isn't essential.
- Shared Mental Space is more important than shared physical space
- Work must be allocated based on Component Interfaces, components should not be split across teams/locations.
- Live with your colleagues! Hire their grads, hire their manager… Ship people at least every 2 – 3 months.
- Release engineering and global continuous build make progress and problems visible and manageable. Wiki, VOIP, Shared White Boards enable dialog at a distance as a compliment to face to face

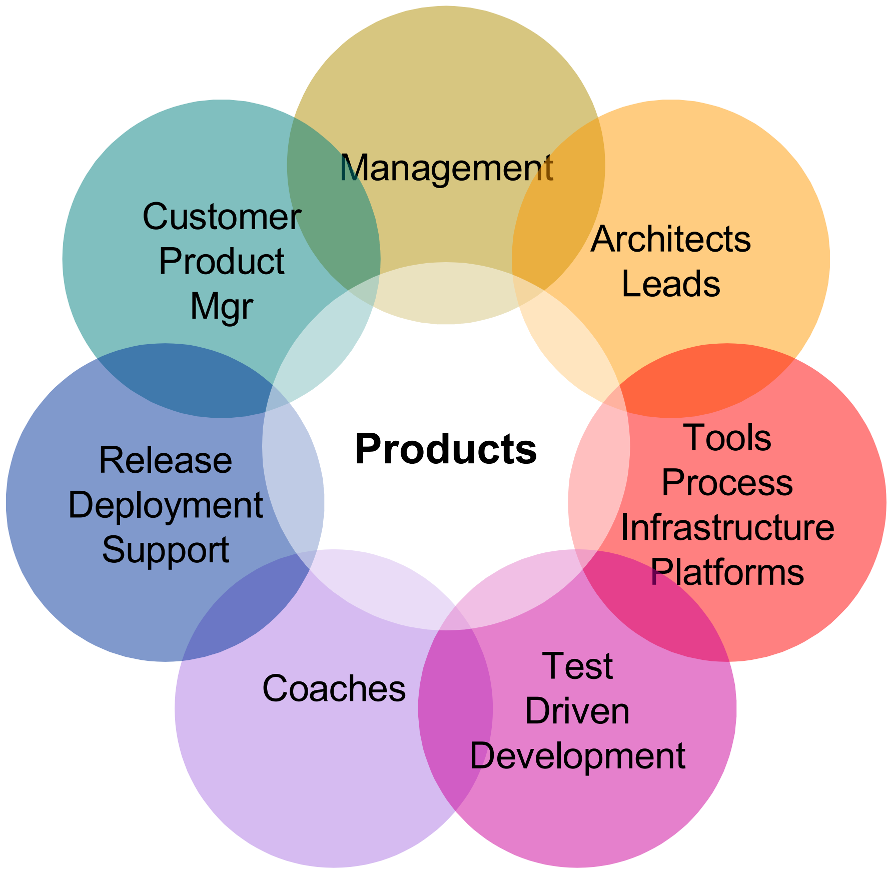
Playing Coaches and Technical Ladders

- Responsible for "walking the floor" to provide independent leadership and expertise – need time to do so!
- Responsible for identifying software waste and communicating the really big story – need time to do so!
- Need a real technical ladder in so that top technical talent has a career path …



Communities of Interest – Some Examples

- Technical Communities
  - Definition Community
  - Envisioning Community
  - Infrastructure and Tools Community
  - Testing Community
- Leadership/Coaching Communities
  - Technical Director/Management Community
  - Team Leader/Scrum Coach Community
  - Customer/Field Community



Key Changes in a Typical Lean and Transition

1. **Software = Product Design and Manufacturing**
   - Increase the understanding of the software value chain at all levels in the organization
   - Identify and reduce waste in software value chain
   - Understand the importance of component, platform and application life cycles
   - Understand the benefits of investment in tangible requirements and architecture
   - Understand how to design quality in (versus test defects out)

2. **Directing and Managing => Leadership and Coaching**
   - Work With versus *Work For* - Coaching versus Directing
   - Increased self discipline for teams and individuals who own deliverables, quality and schedule
   - Increased individual ownership with associated responsibility and accountability
   - Leadership proactively identifies and manages risk

Key Changes in a Typical Lean and Transition

3. My Way versus The Best Same “Wrong Way”

Everyone! needs to change a little for the organization to change a lot!

Common vocabulary, practices and tools applied sensibly and metrics aligned with practices

Make sure everyone knows the same way before fixing it - Improve process each release of the company i.e. triage process/practice/tools defects like other defects

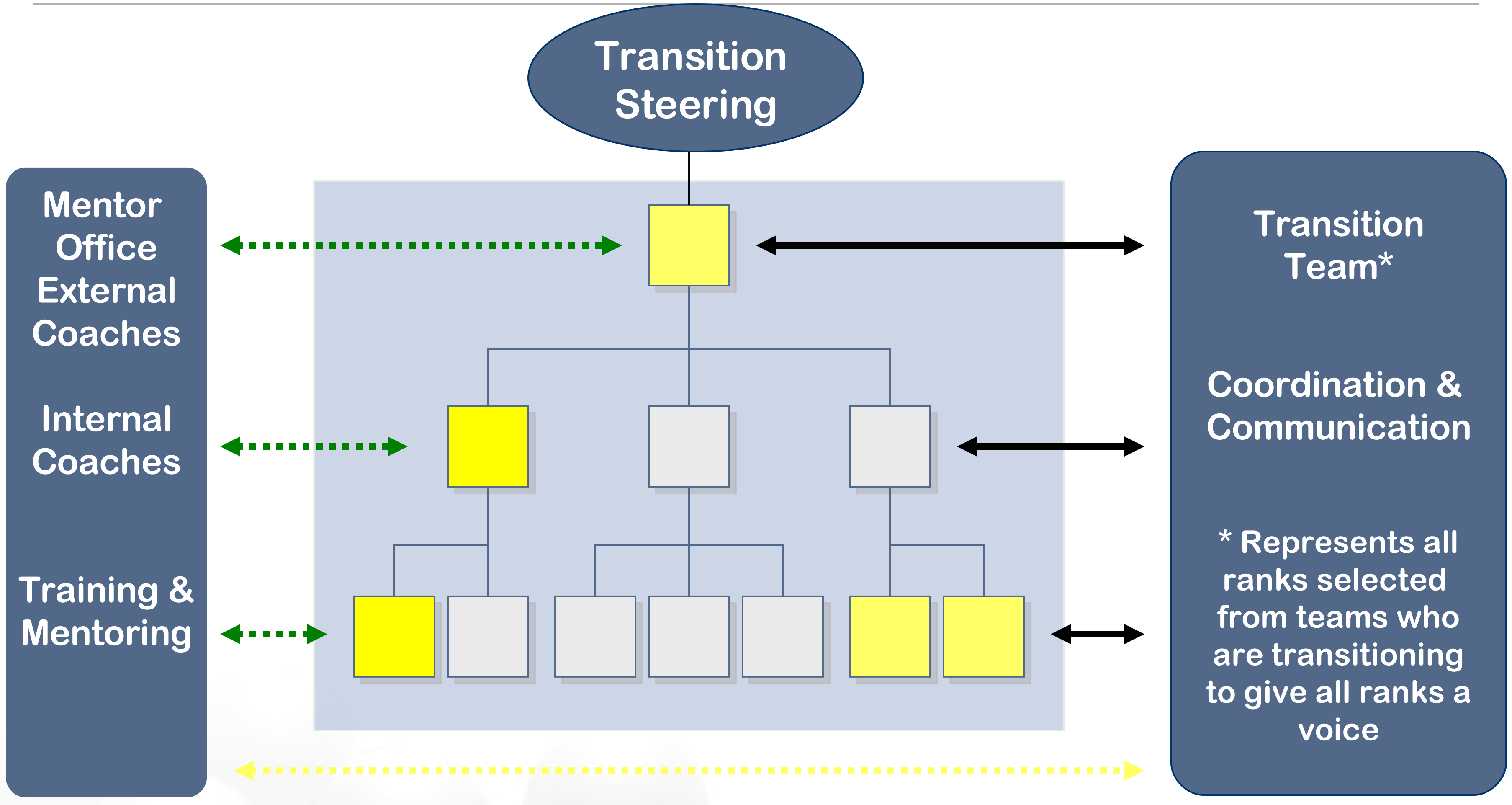
4. Strengthen Technical and Coaching Ladders

Coaches valued for people skills; Technical leaders valued for technical skills

Peer evaluation is an important promotion metric for both

Mandatory constructive annual reviews

Transition Management


Transition Steering
Mentor Office External Coaches
Internal Coaches
Training & Mentoring
Transition Team*
Coordination & Communication
* Represents all ranks selected from teams who are transitioning to give all ranks a voice

⇠⇢ Mentoring/coaching to learn and improve

⟷ Communication at level maintains the trust and provides feedback

🟨 Teams being transitioned

Successful Software Development is about a Winning Culture

- Software is a team sport, and like all team sports practice, constructive peer feedback and coaching are essential.
- Winning teams need to implicitly know the moves of each player, as well as the movements of the team as whole.
- The ultimate expression of process is a culture where building software is more like playing jazz. People Just Do It!