Lean and Agile In the Large
Principles, Practices and Experiences for Large Scale Software Development

Dave Thomas

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

www.davethomas.net
davethomas@objectmentor.com

Lean and Agile and Agility

- Both Lean and Agile and Death By Quality (TQM, Six Sigma, CMM) come from Toyota Just In Time Manufacturing
- Lean appeals to business types most of whom don't understand software and often feel hostage to it
- Agile appeals to development types many of whom don't understand business and large organizations and often feel hostage to it
- Agility appeals to business leaders but they confuse it with Agile Development (predictability + quality may not come with increased productivity or flexibility)
- 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.

 / Page 3

Lean and Agile in The Large is a Business Transformation

1. Adoption of new vocabulary
2. New Practices and Artifacts define a new process
3. New Development Infrastructure to support 1.
4. Transformation from directing to coaching
5. Usually accompanied by organizational refactoring
6. Typically takes 18 – 24 months

Not for the faint at heart!

Lean Principles

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

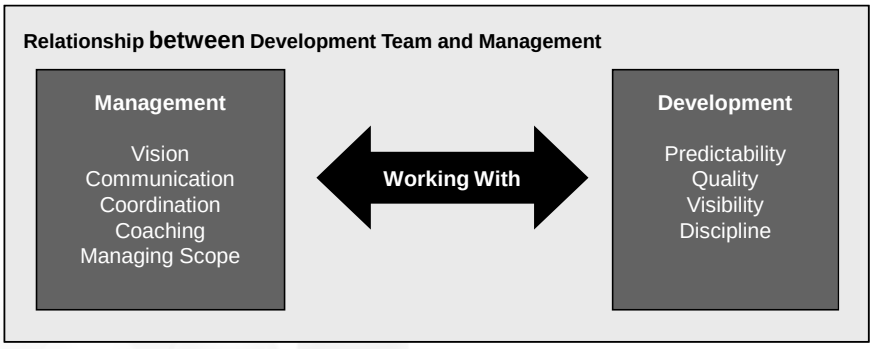
Seven Lean Principles	Core Shared Values
Eliminate waste	Client-focused
Amplify learning	Client-driven
Decide as late as possible	Incremental results
Deliver as fast as possible	Continuous questioning and introspection
Empower the team	Change is progress to a better solution
Build integrity in	
See the whole	

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

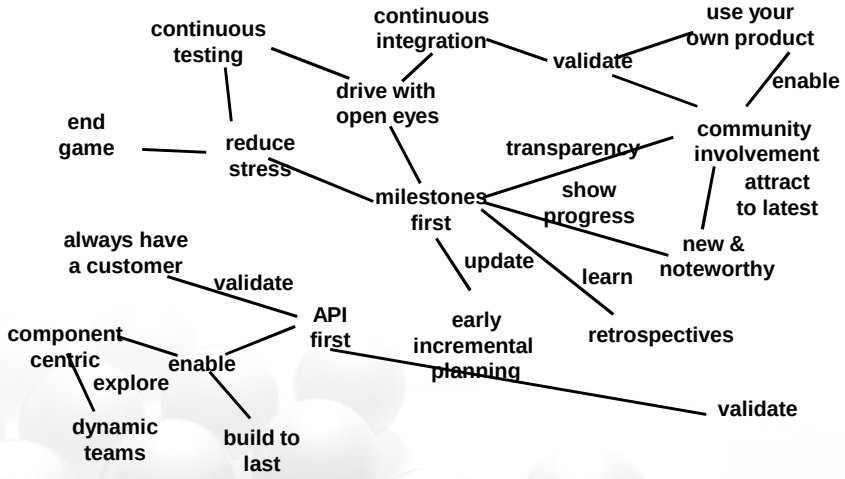
Lean/Agile Commitments

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

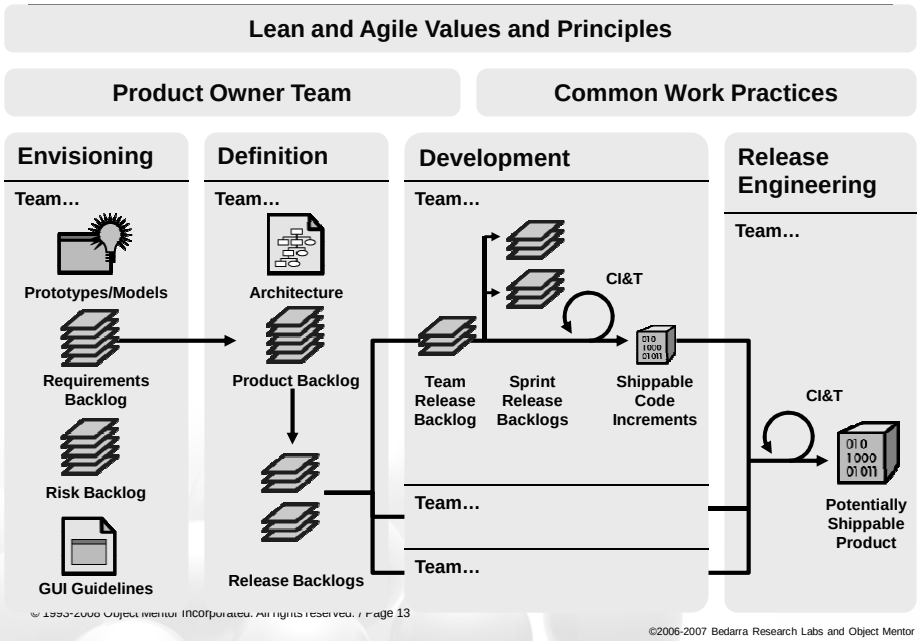


A Large Scale Lean and Agile Success Story

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



Large-Scale Lean Development Activities At-A-Glance



Good Software Comes in 3s

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

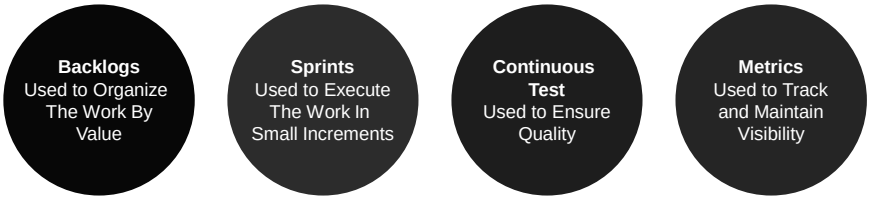
- Once to understand what it is (**Envisioning**)
- Once to understand how it goes together (**Definition)**
- One last time to make it for real (**Development**)
- Make sure everyone shares the big story, has the right infrastructure and the parts make the whole and ship it (**Release Engineering**)

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

Common Work Practices and Rhythms

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



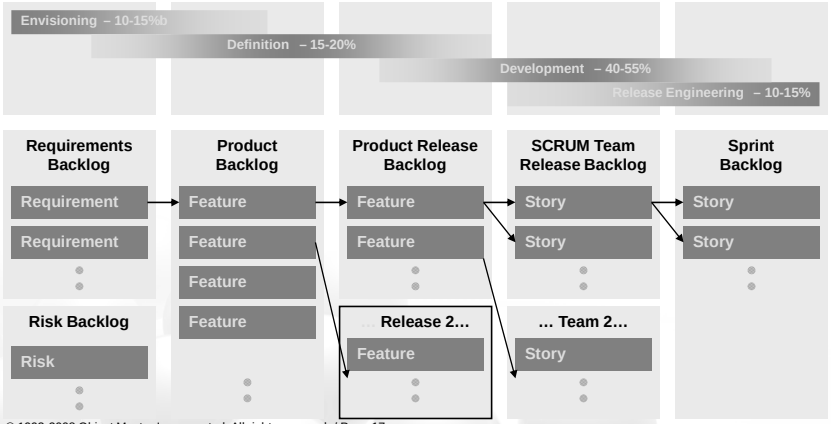
Sprint Rhythm (2 wk)
• Sprint Planning (1) , Daily Stand-ups .., Sprint Retrospective (1)
• Features => Stories => Tasks (1 – 2 days)

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

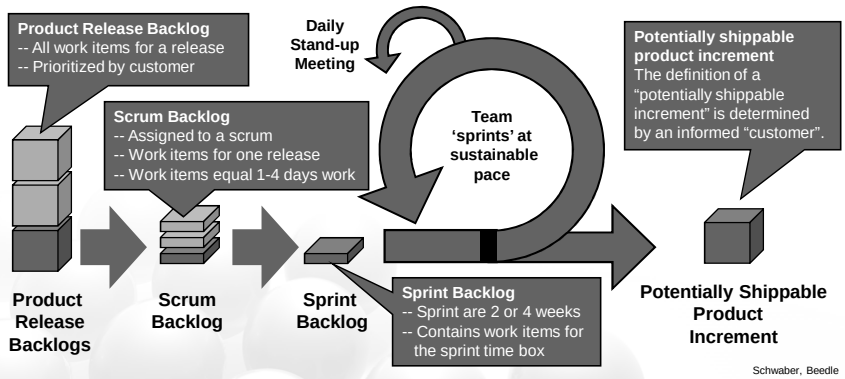
Backlogs are Used To Organize The Work

- 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



Schwaber, Beedle

Types of Scrums

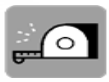
- Scrums are used for all activities in the product life cycle
- Envisioning Scrums
  - Collect and analyze market, customer, product, and technology requirements
  - Develops proof-of-concept and vision prototypes
  - Verifies prototypes with customers
  - Capture requirements in a Requirements Backlog
  - 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

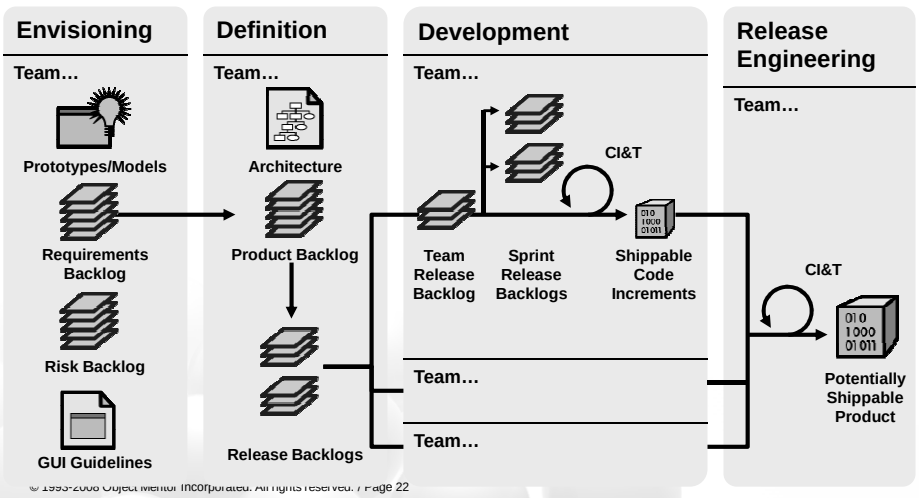
- Project management not a separate entity in an Agile team
- Project management owned collectively by team
  - Team members contribute to the estimating process and commit to the deliverables
  - Team members own estimates, schedules and deliverables
- Team responsible for maintaining visibility
  - 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”

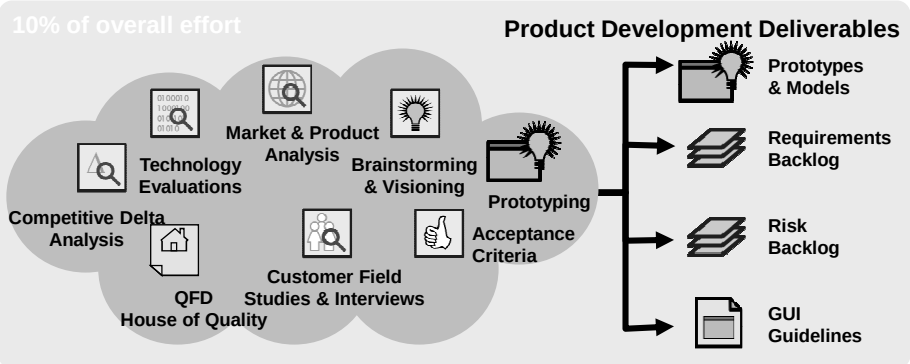
The initial sprint is often called sprint 0 and it is used to as a exploratory/high level planning activity which seeks to explore, understand, swag, identify dependencies using practices such as:

- 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!

Envisioning

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



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

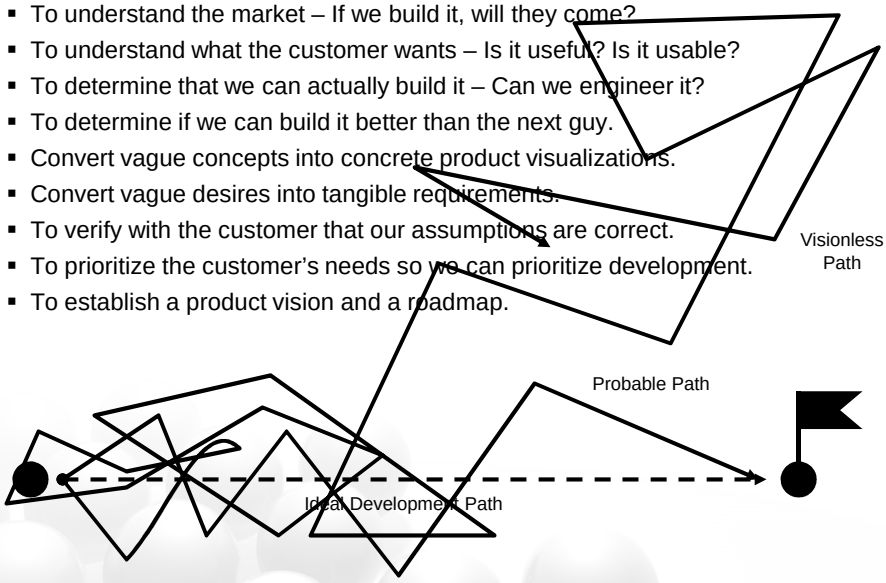
- Customers, Business Analysts, Product Owners and Developers need a simple story (s) which provides a shared vision of what is being developed
- Story Telling carries the essence, humans render the details according to the story, be it complex software, knowledge management or animated films
- Story 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.



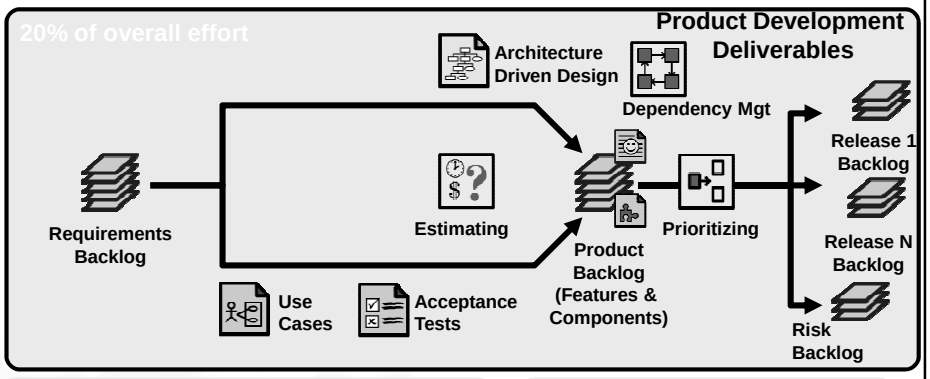
What Is Definition?

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

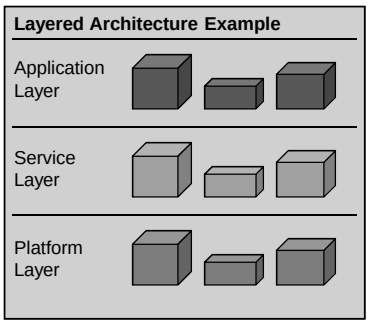
- A feature is a customer requirement that has been translated into something that can be implemented by the development team
- Typically consists of one or more use cases and acceptance tests
- The use cases describe the feature's operation
- The acceptance test describes the acceptable outcomes

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

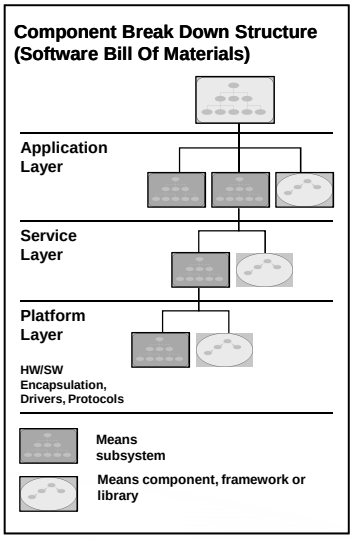
Layered Architectural Model

- 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.



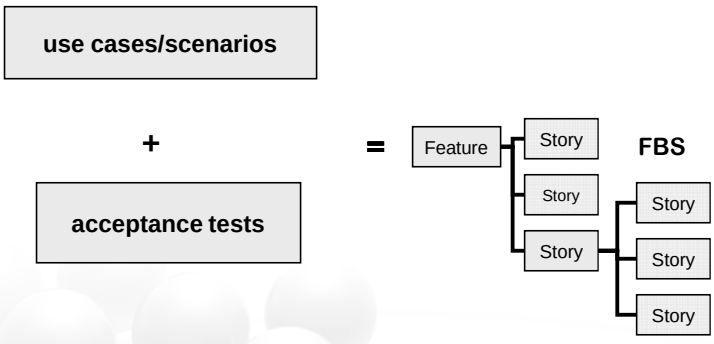
Component Breakdown Structure (CBS)

- 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.



Feature Breakdown Structure (FBS)

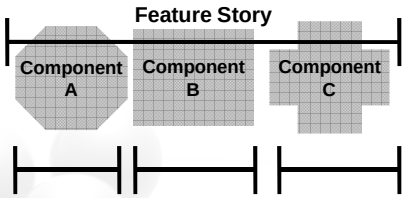
- Features are organized in an FBS
- The FBS relates features to other features (feature dependencies)
- The FBS also cross cuts components (component dependencies)
- Dependencies are captured in a Dependency Structure Matrix



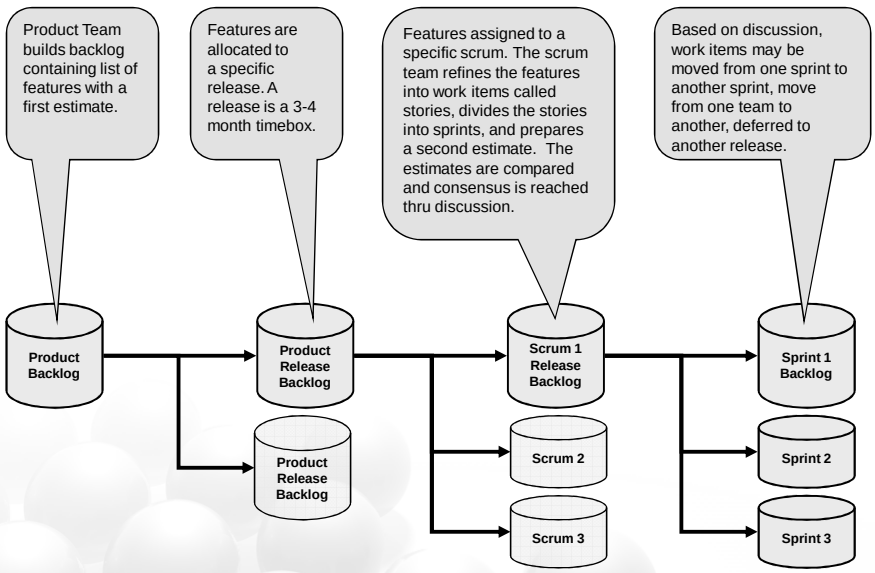
Feature and Component Teams and Backlogs

- 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 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.

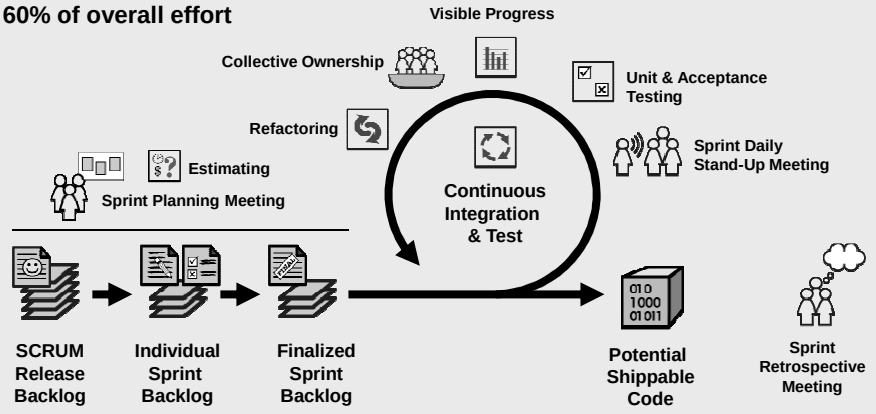
Planning –The Best Wrong Answer

- 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 wide band Delphi)
- 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!)

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

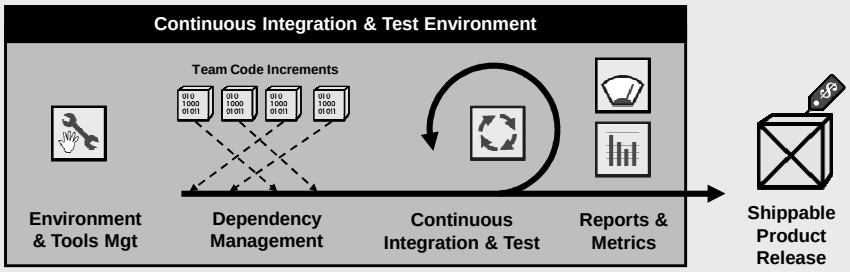
Deliverables

■ 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



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

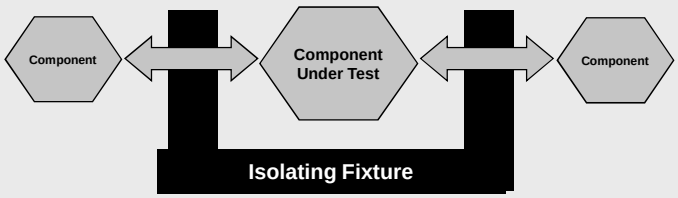
Release Engineering - *Start With a Winning End Game*

- 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

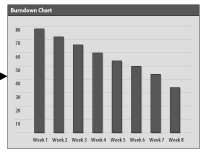


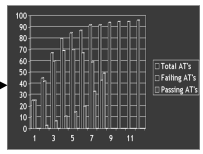
Establish A Product Development Dashboard

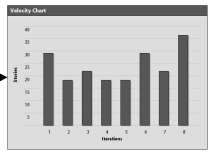
Automate the linkage between features/stories, code and tests

Measure true progress based on the state of the continuous build

Produce AT, UT, Velocity and Burndown Charts Automatically







@StoryMain

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

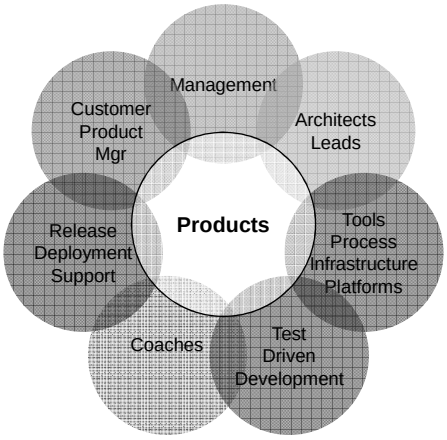
- Values
- Vision
- Standardize Vocabulary and Practices e.g. points vs. ideal days…
- Spread Experienced People Across Teams/Locations
- Use Playing Coaches to share vision and experiences
- Common Tools
- Empower Distributed Teams – VOIP, IM, electronic White Boards
- Provide company wide visibility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

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



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!

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
3. **My Way versus The Best "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