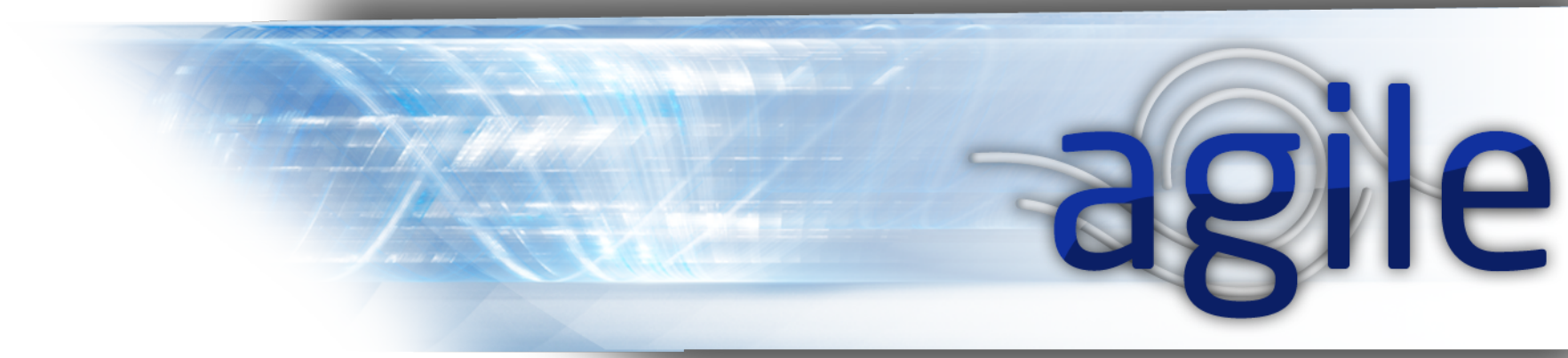
Tradecraft

Best Practices & Agile Development



Tony Alletag

Penn State
B.S. IST
M.S. MIS

Open Source Development

Virtual Infrastructure

Agile Coach

IT Project Manager

Chief Technology Officer

Small Business Owner

15 Years Software Development

Best Practices

Modern Tech Stack

open source

Logistics

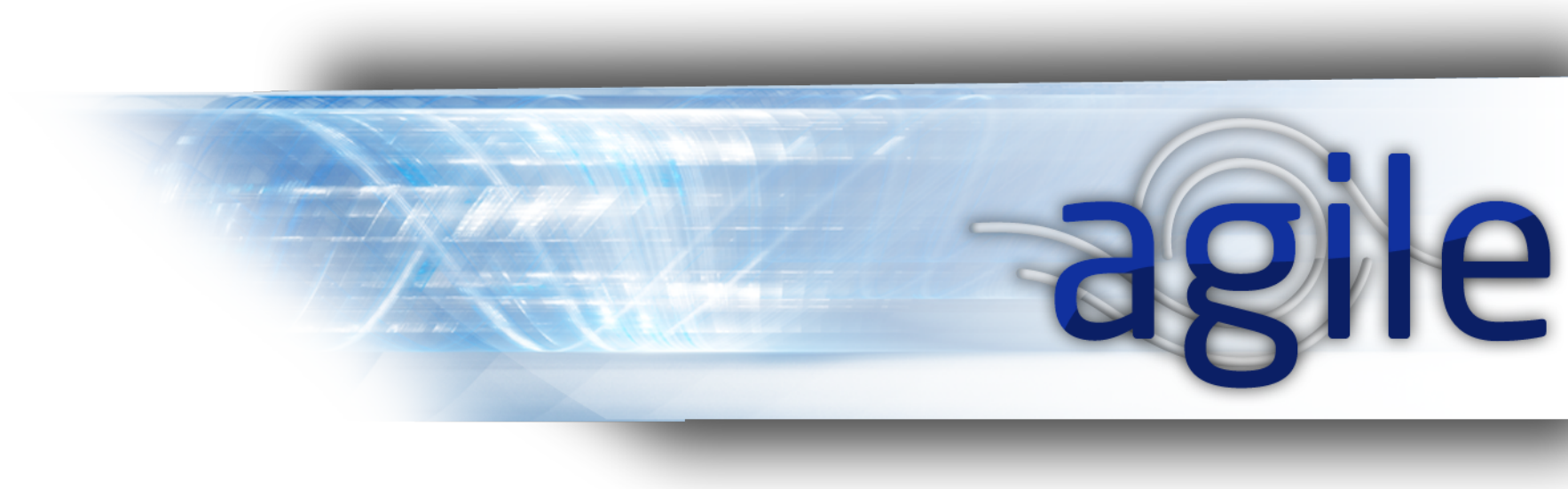
- Class will be given in **Iterations**
- Each Iteration will be followed by a **Break**
- Parking Lot Chart
- Definition of Done - “Fist to 5”
- Working Agreement

50	10	50	10	50	10	50

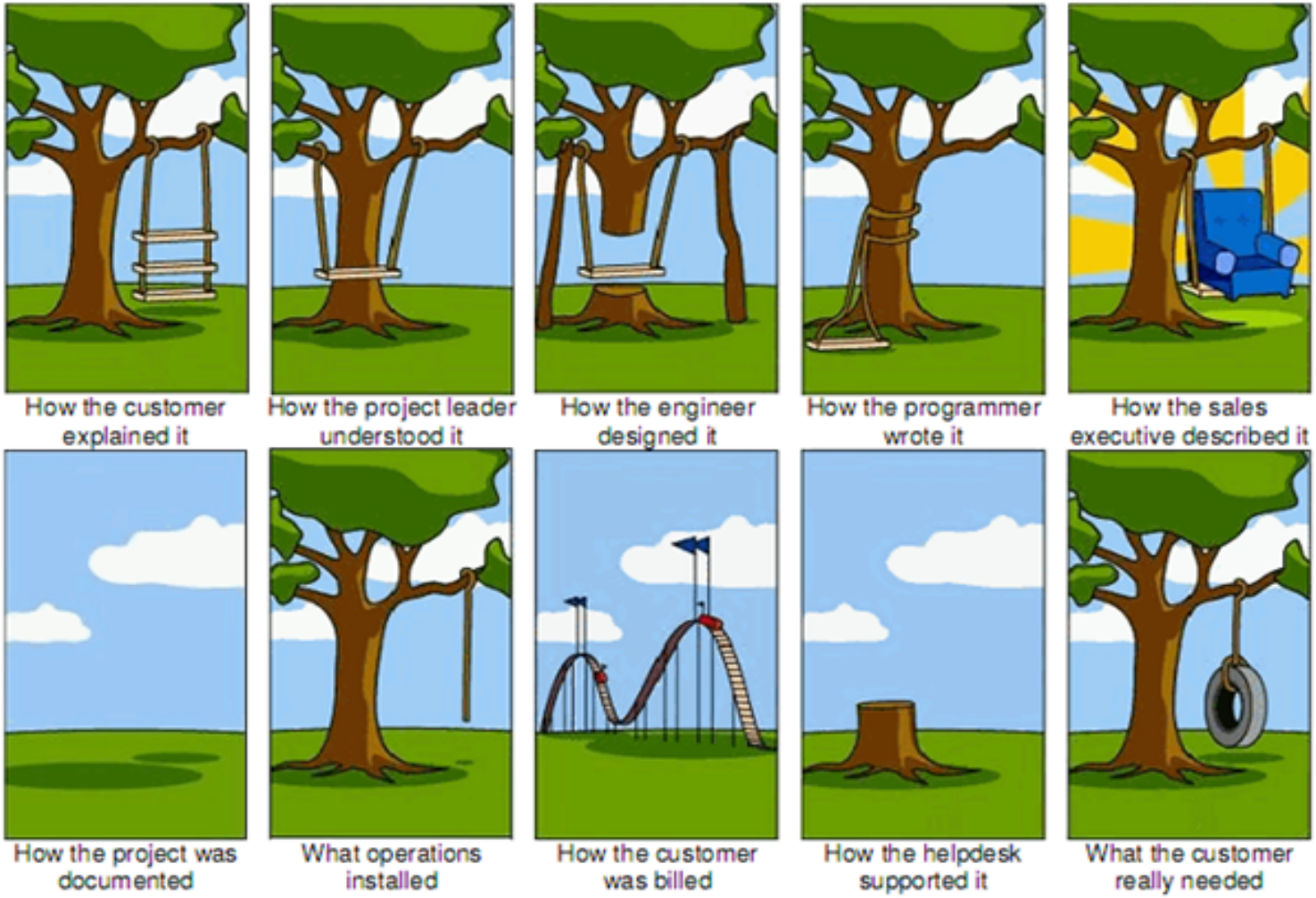
Backlog

- What is Agile
- Common Terminology
- Agile Teams
- Iteration 0
- Iteration N
- Iteration R
- Metrics
- Tools
- Continuous Learning

What is Agile



The Nature of Software Development



Introduction

Agile Development Process

- Agility and rapid delivery of capabilities
- Continuous planning, testing, in response to continuous change
- Agile does not equal scrum

Goals

- Deliver working tested software frequently with 2 week iterations
- Welcome change even late in development
- Continuous integration and automated unit test
- Trust team members to do their job and do it correctly
- Work off of a prioritized Backlog

Quality Quickly

“The Waterfall” Utopia

Royce, Winston (1970), "Managing the Development of Large Software Systems", *Proceedings of IEEE WESCON*

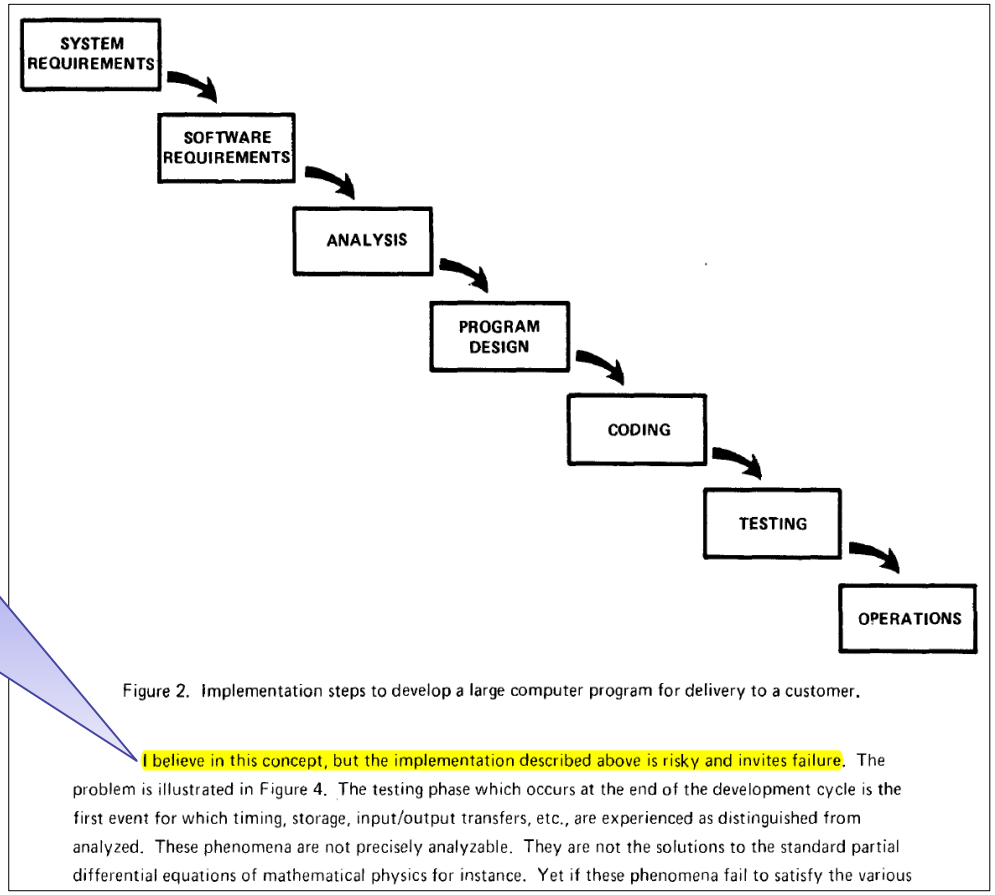


Figure 2. Implementation steps to develop a large computer program for delivery to a customer.

I believe in this concept, but the implementation described above is risky and invites failure. The problem is illustrated in Figure 4. The testing phase which occurs at the end of the development cycle is the first event for which timing, storage, input/output transfers, etc., are experienced as distinguished from analyzed. These phenomena are not precisely analyzable. They are not the solutions to the standard partial differential equations of mathematical physics for instance. Yet if these phenomena fail to satisfy the various

“I believe in this concept, but the implementation described above is risky and invites failure.”

Waterfall not for Software Development

The Agile Manifesto

We are uncovering better ways of developing software by **doing it** and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on the right, **we value the items on the left more.**

IKIWISI

I Know It When I See It

IRKIWIEI

I Really Know It When I Experience It

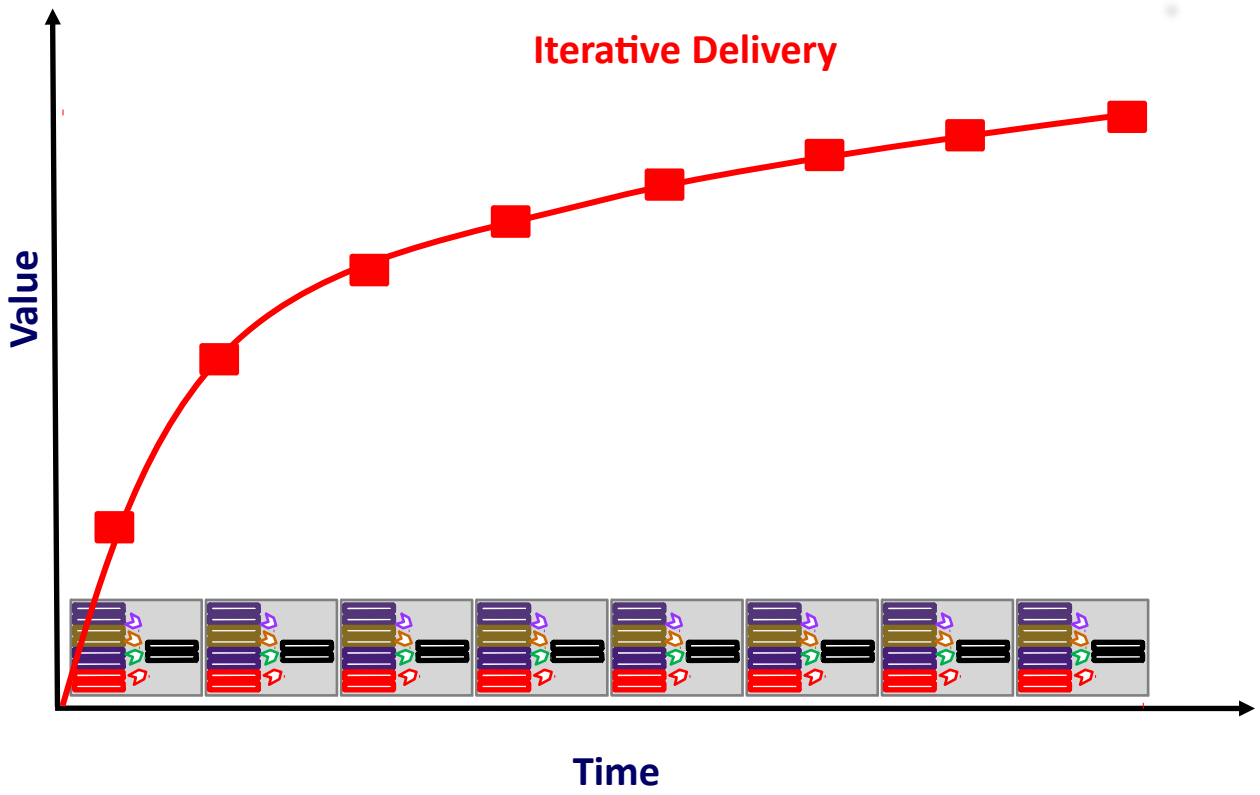
Minimal Viable Product

A minimum viable product (MVP) is the the most pared down version of a product that users are able to use

- It has enough value that people are willing to use it or buy it
- It demonstrates enough innovation and future benefit to retain early adopters
- It provides a feedback loop to guide future development

Get Real User Feedback Early

Prioritized Iterative Delivery

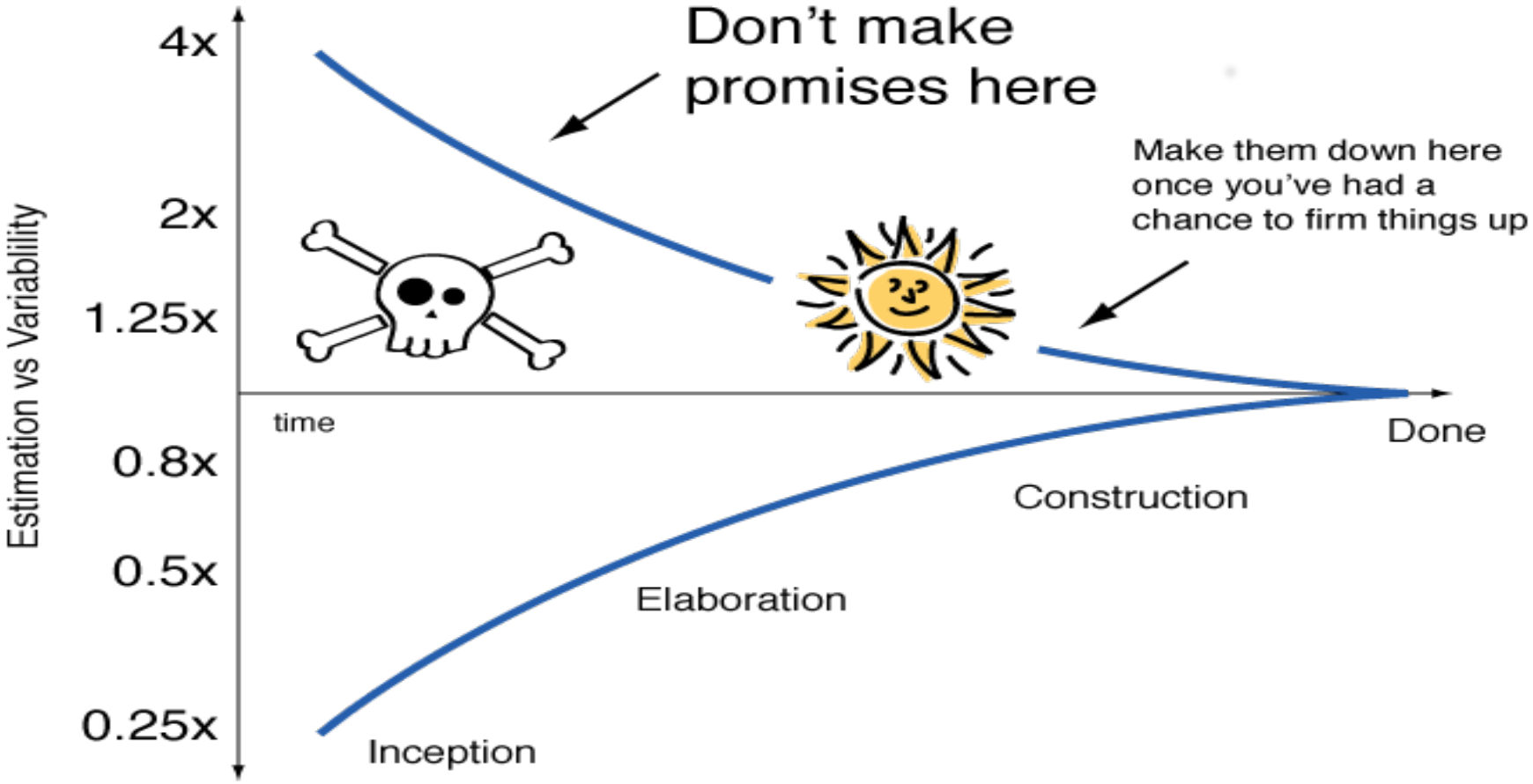


Iterative Value

Sustainable Pace

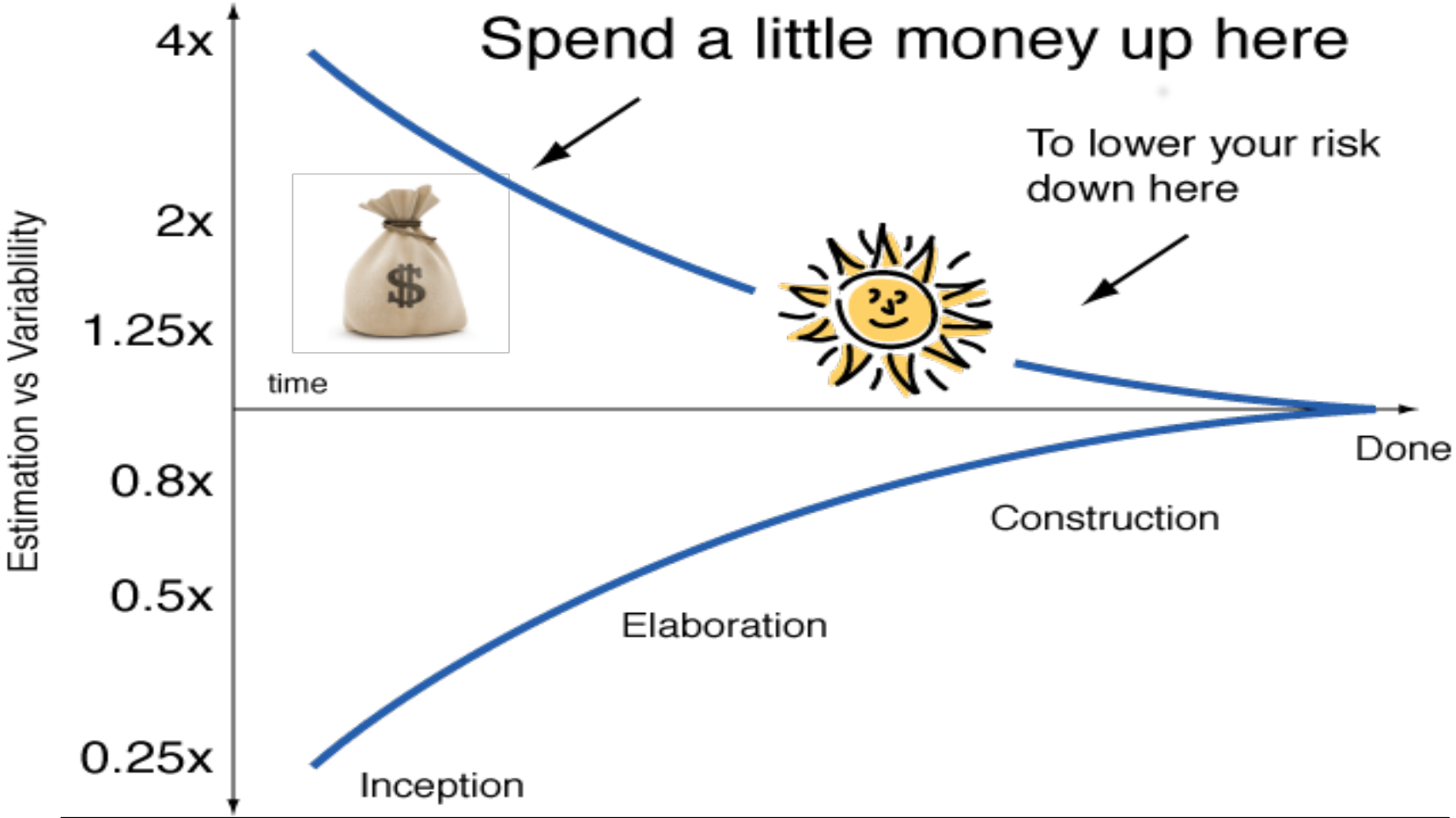
Heroes Developers Impair Agile Teams

Cone of Uncertainty



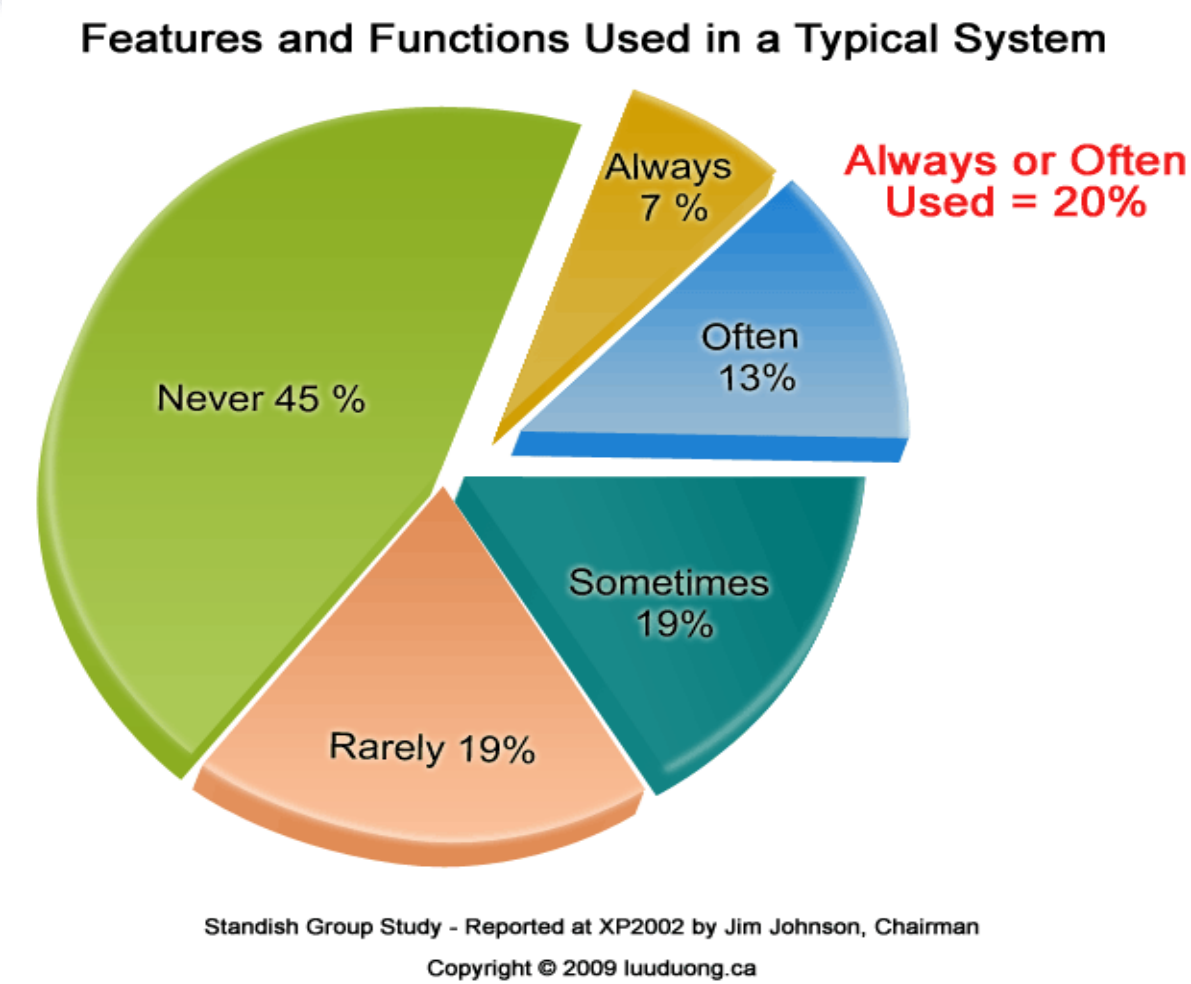
Devils in the Details

Cone of Uncertainty



Devils in the Details

Feature Usage



Over Engineering

The Agile Principles

1.Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

2.Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

3.Deliver working software frequently, from a couple of weeks to a couple of months, with preference to the shorter timescale.

4.Business people and developers must work together daily throughout the project.

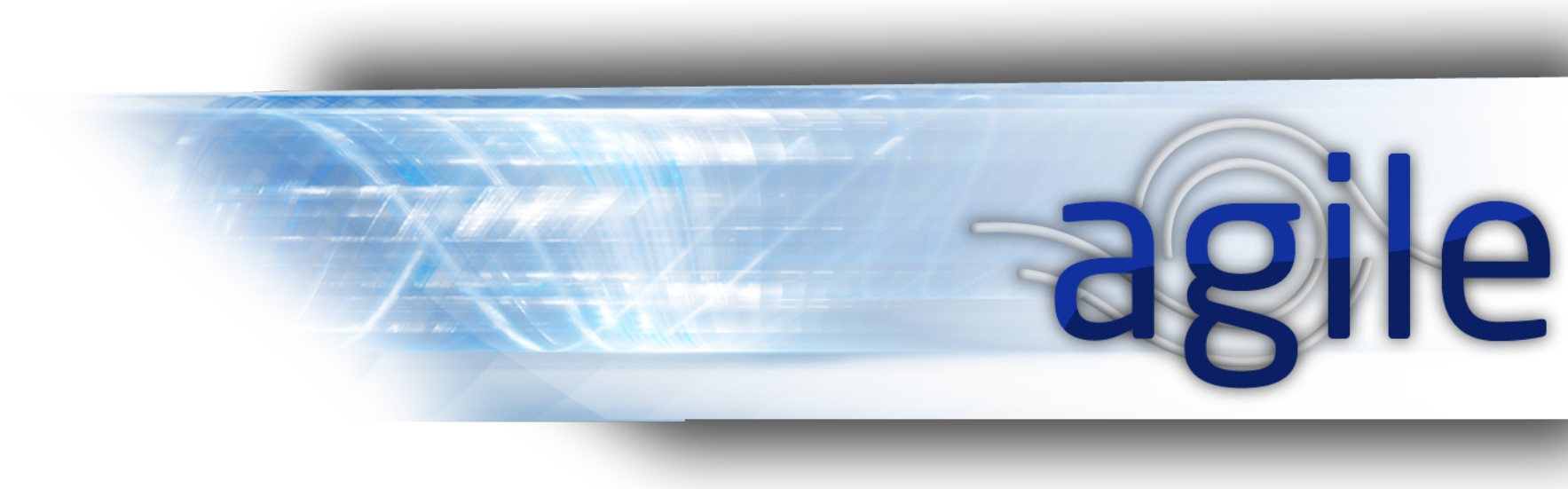
5.Build projects around motivated individuals. Give them the environment and support they need and trust them to get the job done.

6.The most efficient and effective method of conveying information to and within a development team id face-to-face conversation.

The Agile Principles

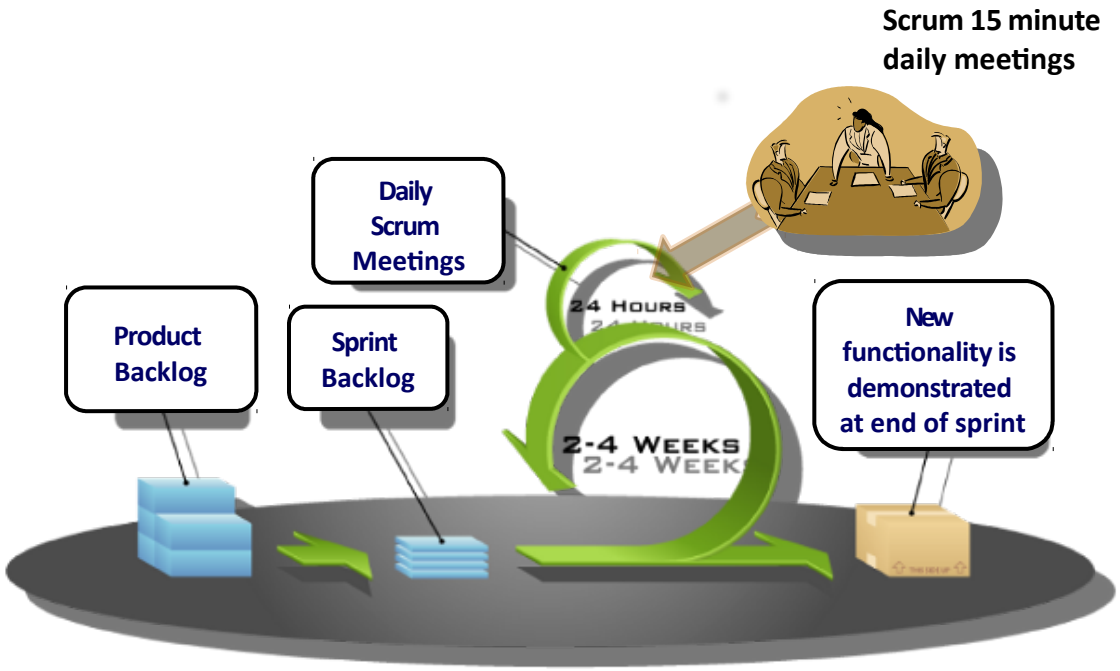
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity – the art of maximizing the amount of work not done – is essential.
11. The best architectures, requirements and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more efficient, then tunes and adjusts it's behavior accordingly.

Flavors of Agile



Scrum

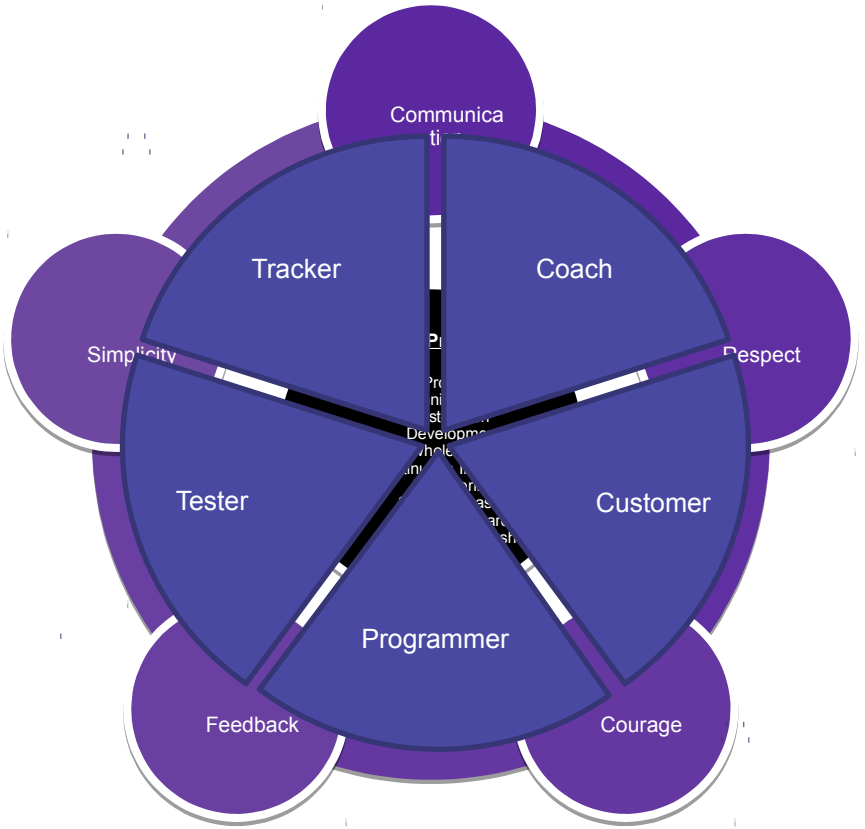
- Three Roles
  - Product Owner
  - Team
  - Scrum Master
- Artifacts
  - Product Backlog
  - Sprint Backlog
  - Burndown chart
- Ceremonies
  - Daily Standup
  - Sprint Planning
  - Sprint Review and Retrospective



Agile is more then Scrum

Extreme Programming (XP)

- Extremely Disciplined
- Five Values
- Five Roles
- 12 Practices



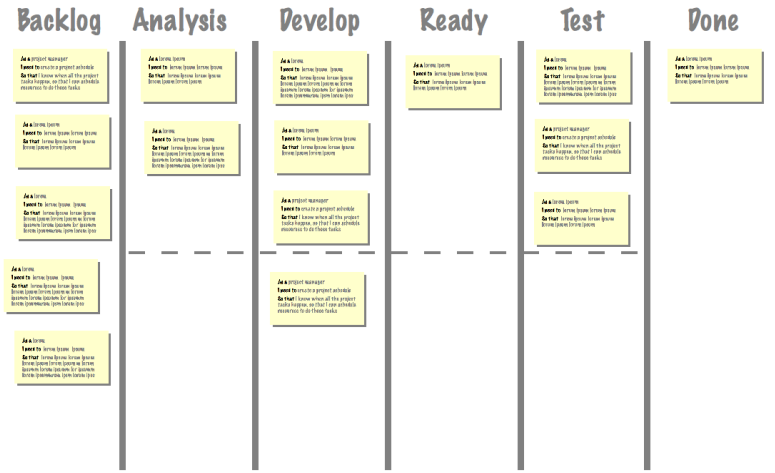
2X

Lean

- Reduce Work In Progress
  - Just In Time (JIT)
  - “Last Responsible Moment”
  - Actively Seek and Destroy Waste
- Translation of lean manufacturing to software development domain
- Adapted from the Toyota Production System

Kanban

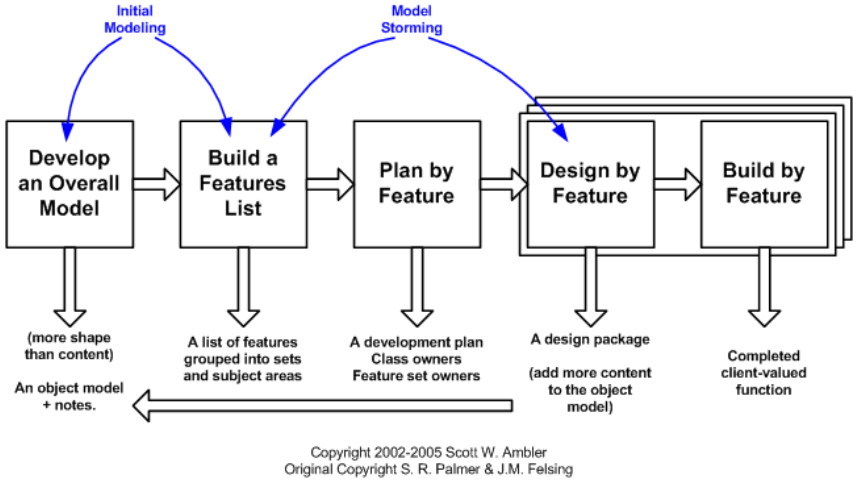
- Make Work Visible
- Limit Work in Progress
  - Explicitly limit the number of tasks, stories, features and epics
- Help Work to Flow
- Make Policies Explicit
- Evolutionary Change
  - Improve using performance data
- Distinguishable as “Iteration-less” – A continuous flow of capabilities



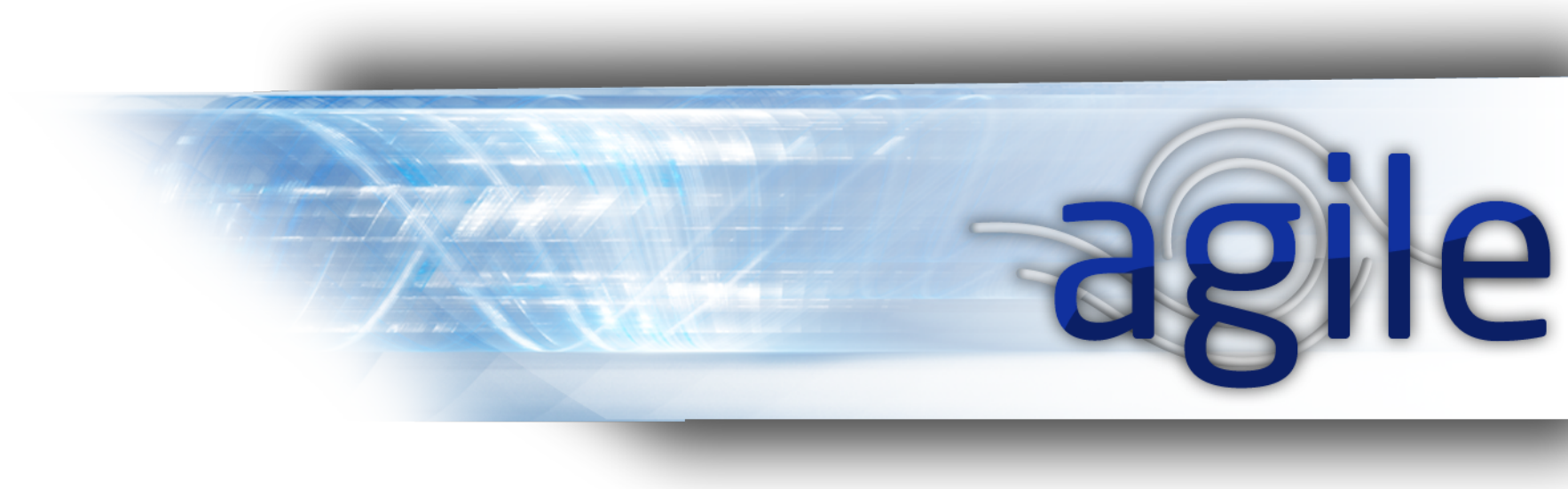
Work in Progress

Feature-Driven Development (FDD)

- Focus on End-to-End Working Software
- 5 Basic Activities
  - Develop Overall Model
  - Build Features List
  - Plan by Feature
  - Design by Feature
  - Build by Feature
- Heavily leverages Metamodeling techniques
  - Consistent with Model-Based System Engineering Practices



The Mindset



What Is Agile

Savior!

SCRUM?

Approach?

Methodology?

Chaos

Fad?

No Architecture

No Planning

Lean?

Process?

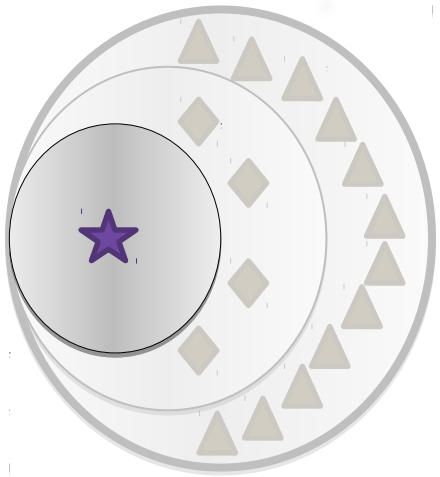
No Documentation

XP?

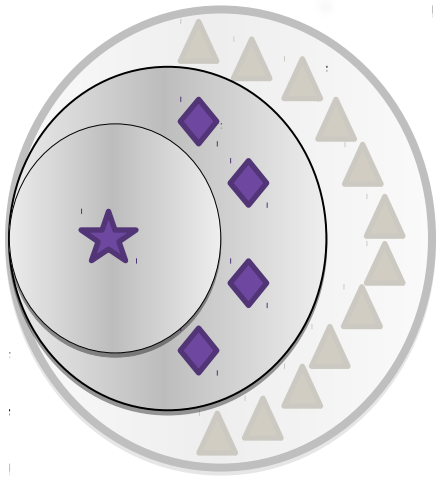
Framework?

No Discipline

Agile is a Mindset

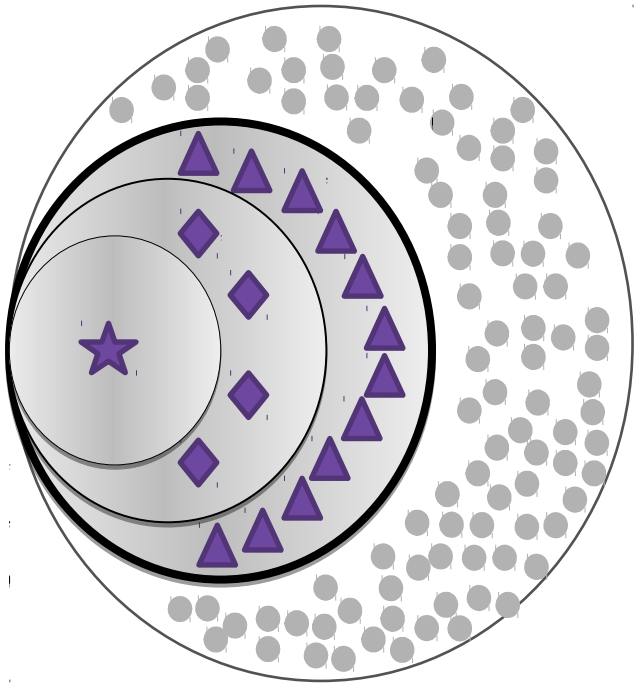


Agile is a Mindset



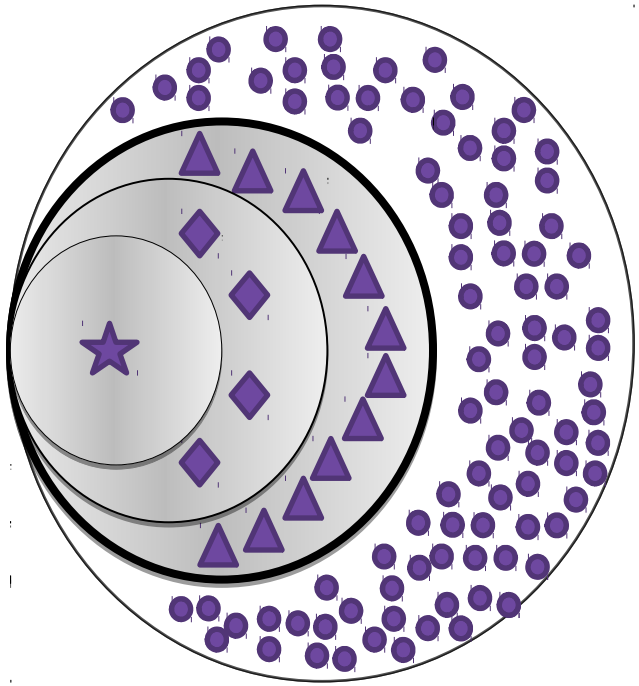
- defined by values

Agile is a ★ Mindset



defined by values ◆ , guided by principles ▲a

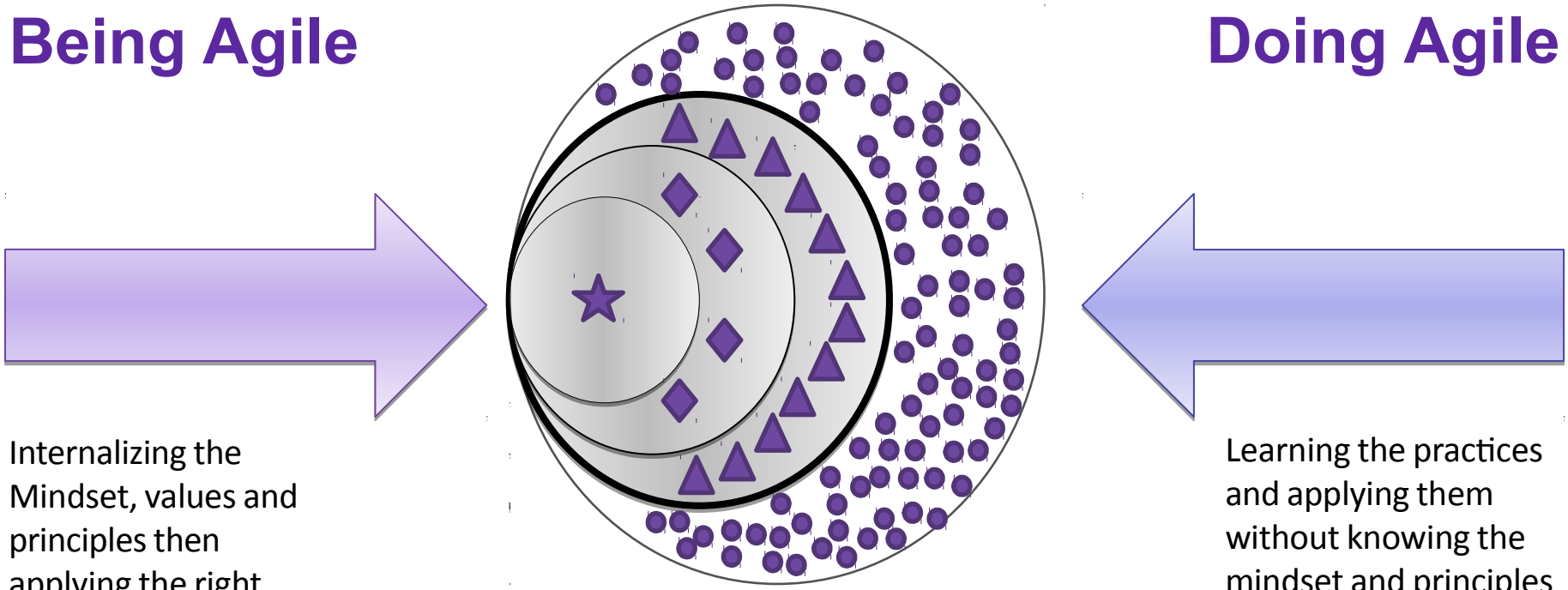
Agile is a Mindset



defined by values guided by principles and manifested through many different practices

What Is Agile

Being Agile

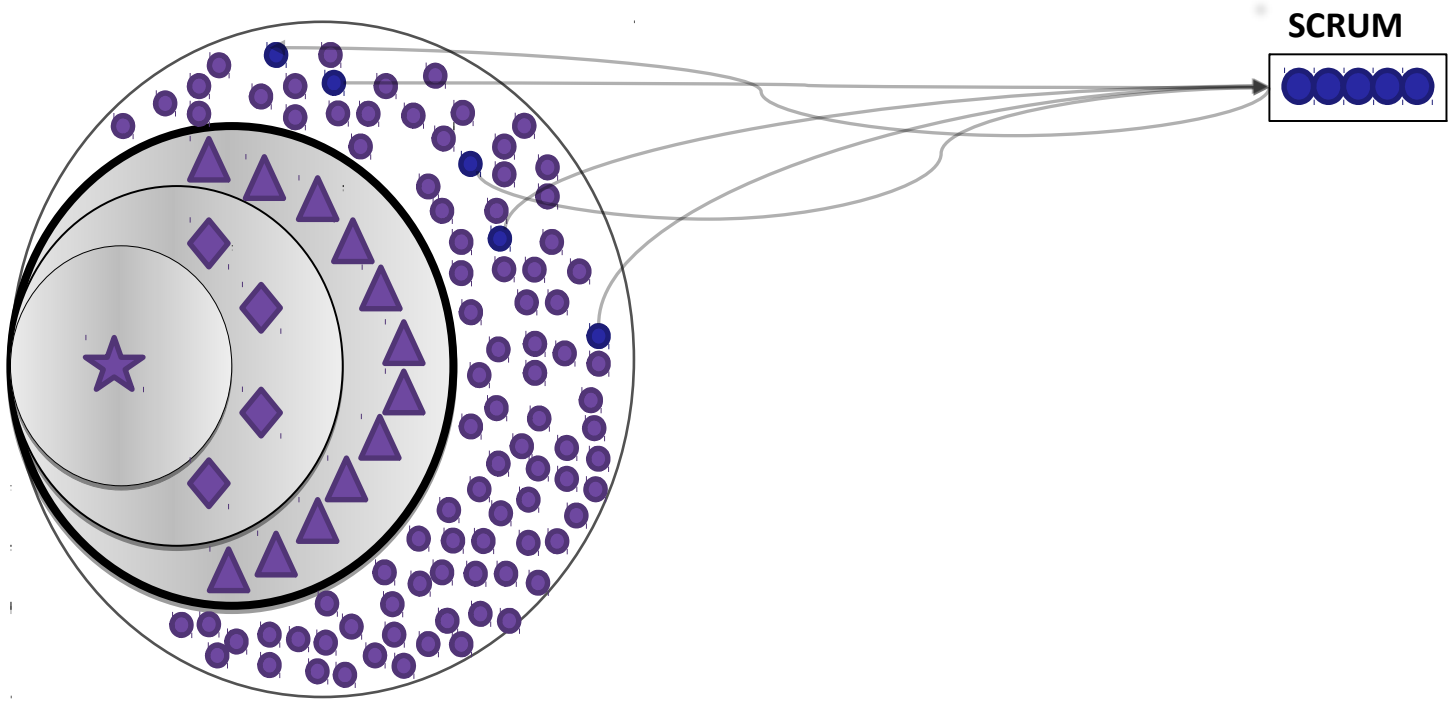


Internalizing the Mindset, values and principles then applying the right practices and tailoring them to different situations

Doing Agile

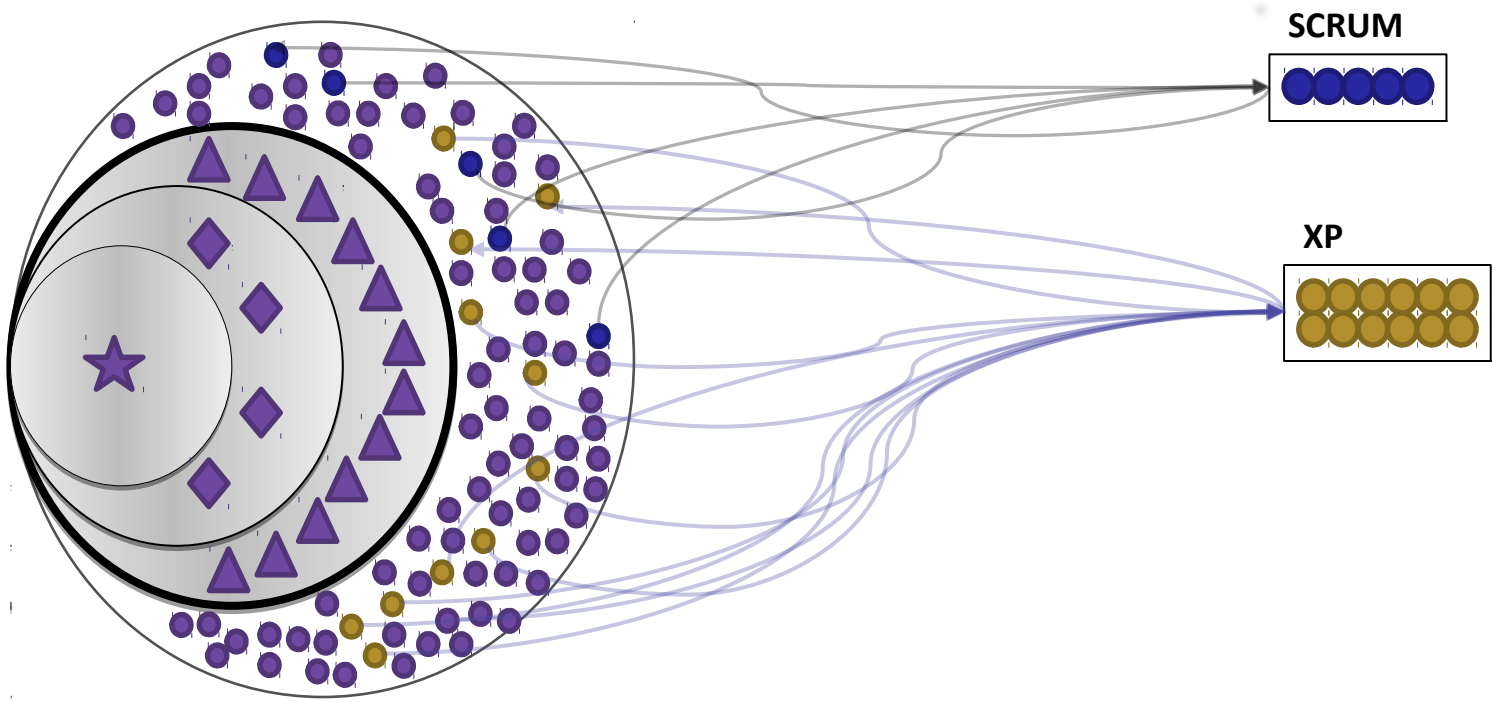
Learning the practices and applying them without knowing the mindset and principles to know when to tailor and how to select the appropriate practices

What Is Agile



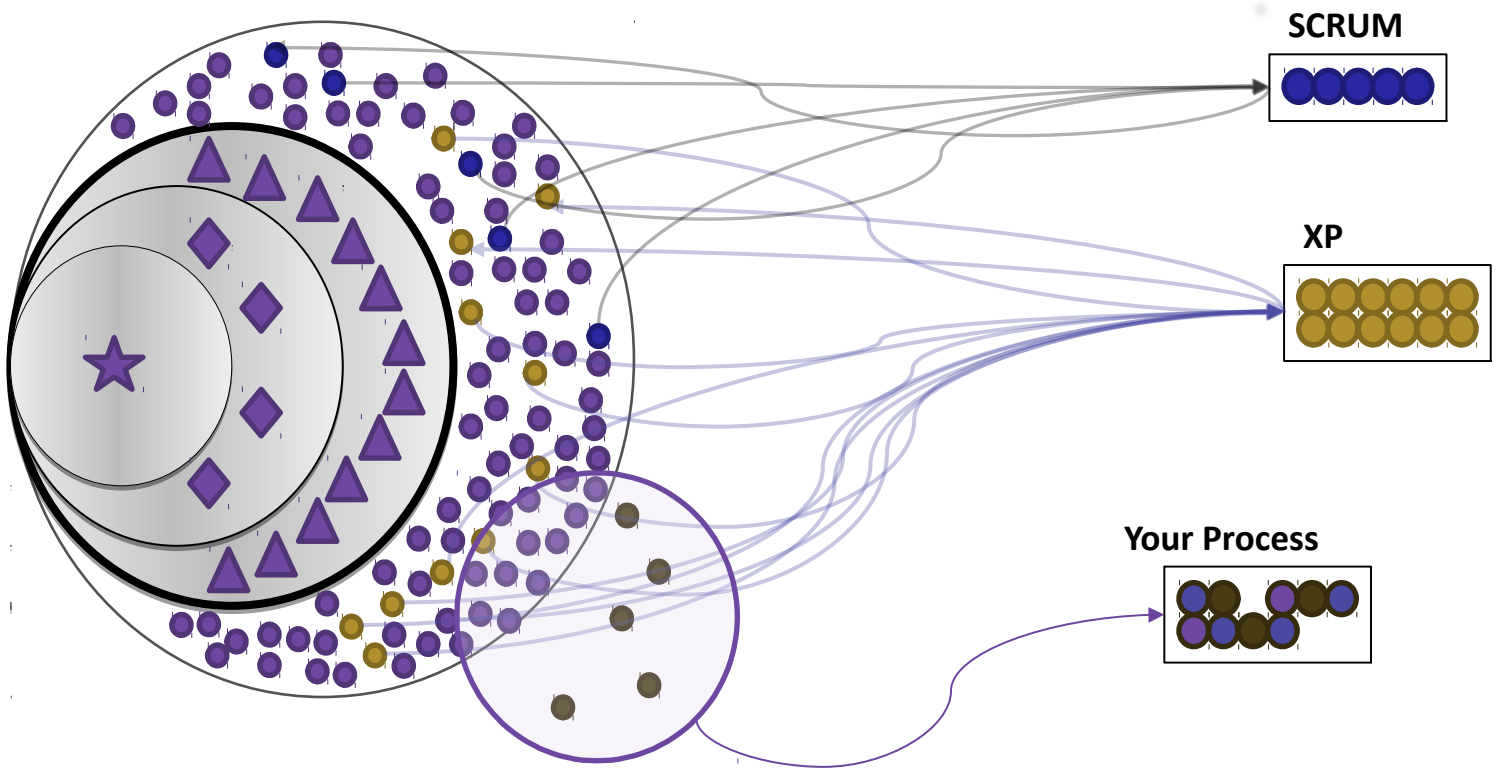
Myth: Scrum is Agile

What Is Agile



Myth: XP is Agile

What Is Agile

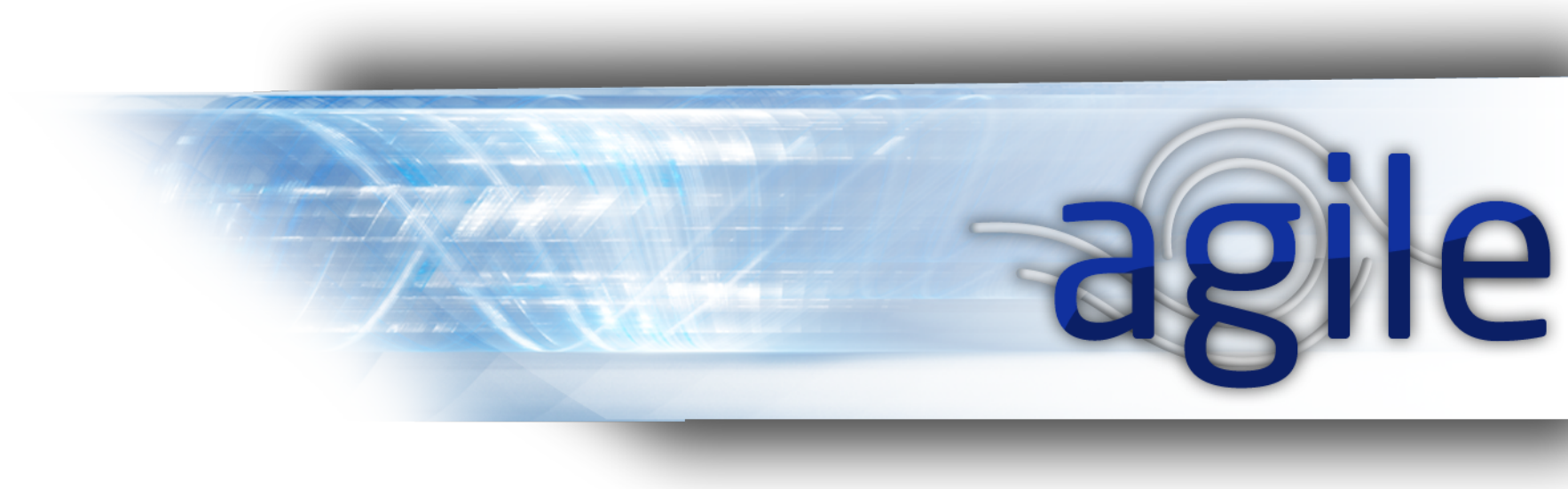


Making Agile Work for Your Team

What Is Agile

Agile Is A Mindset

Common Terminology



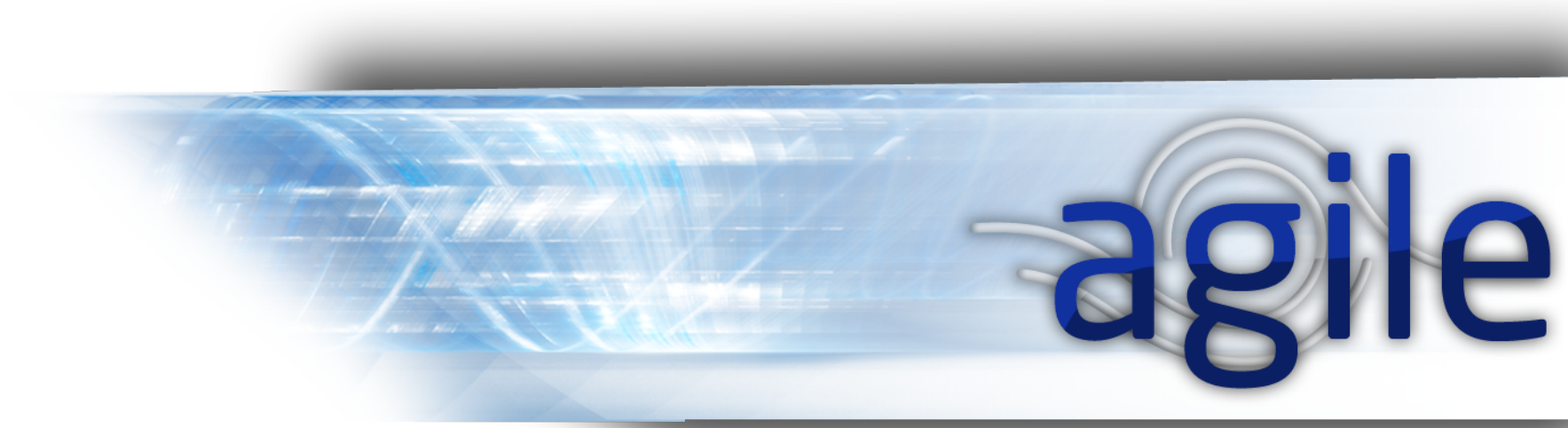
Common Terminology

- Roadmap
- Product Backlog
- Release
- Capability
- User Story
- Velocity
- Burn-down Chart
- Collective Code Ownership

- Bug
- Iteration
- Daily Stand-up
- Retrospective
- Done
- Version Freeze
- Technical Debt
- Sustainable Pace
- Continuous Integration

Communication

Agile Team roles



Agile Team Roles

Integrated Teams

Agile teams are self-organizing and self-managing

Consistent flow of information with “Information Radiators”

Communication

Agile Team Roles – User

Uses the capability that is being developed

Communicate the functional needs and capabilities

Participates in creation, prioritization, and review User Stories and Acceptance Criteria

Most Important Role for Success

Agile Team Roles – Customer

- Communicate what they want the delivered product to do and achieve
- Prioritize features and improvements
- Software and product acceptance
- Constantly provide feedback on features that are being developed
- Execute acceptance test
- Provide advice as the domain and subject matter experts
- Communicate the functional needs and requirements of the capability
- Being available to all of the agile team members for feedback

Most Important Role for Success

Agile Team Roles – Customer

Constant Customers availability to all Agile Team members is critical for collaboration and development success

Direct Access

Agile Team Roles – Coordinator

Project management Oversee all activities of the project

Foster and facilitates Customer relationships

Manages the project team's releases and iterations

Plans release, test, and deployment schedules

Maintain the Product Backlog

Agile Glue

Agile Team Roles– Project Coordinator

- Participate in prioritization of team workload
- Identifies and eliminates roadblocks
- Build trust and maintain customer relationship
- Roadmap features and plan releases with the coordinator
- Team leader, mentor, and problem solver
- Identify and eliminate roadblocks
- Software engineering experience

Agile Glue

Agile Team Roles– Project Coordinator

Communicates and collaborates with all project resources and stakeholders

Agile Glue

Agile Team Roles– Technical Lead

Provides overall technical guidance of capability development

Is the Agile Team's technical expert

Provides technical expertise in project's technology and implementation design

Mentor other Developers and Write Code

Agile Team Roles– Technical Lead

- Career software engineer
- Motivate and coach the team
- Preforms Code reviews as needed and when requested by other team members
- Technical problem solver
- System design architecture
- Works with the coordinator to ensure that project goals are clearly communicated to the project team and regularly measure progress toward such goals

Mentor other Developers and Write Code

Agile Team Roles– Technical Lead

Understands the team's technical capabilities and limitations

Motivates and provides technical coaching to the development team

Mentor other Developers and Write Code

Agile Team Roles– Developer

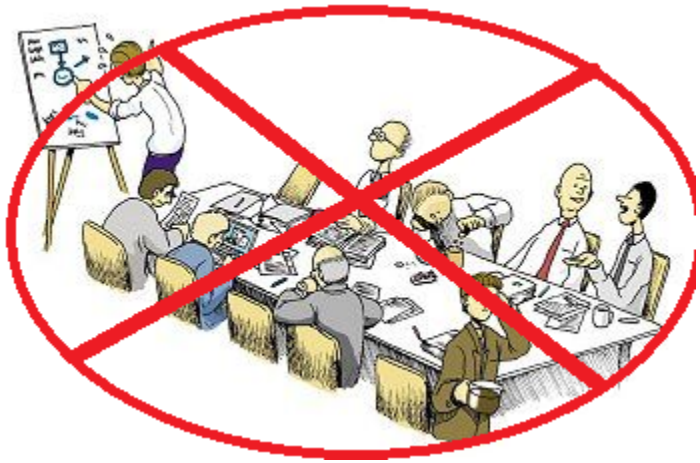
- Write users stories based on the feature backlog
- Write code to develop a releasable product at the end of every iteration
- Database design and development
- Implement test cases and unit test
- Self-motivated and self-organized
- Deploy the application to all server environments

Maximize Time Writing Code

Agile Team Roles– Developer

Owns and estimates the level of effort for User Stories

Collaborates at daily stand-up meetings



Maximize Time Writing Code

Agile Team Roles - Designer

- Design a product that is appealing and easy for the users to use
- Use of media queries and responsive design techniques device-agnostic design
- Ensure users are able to find relevant information and do what they want with the product
- Get feedback from real users and make design improvements
- Develop wireframes or primitive screen mockups
- Develop site maps and user flow diagrams

Simple Easy to Use Interface

Agile Team Roles - Tester

Developer

Understands the system but, will test "gently" and, is driven by "delivery"

tester

Must learn about the system, but, will attempt to break it and, is driven by quality

Daily Testing of Completed User Stories

Agile Team Roles - Tester

- Provides functional verification and validation of the User Story
- Collaborates in the development of the user stories and acceptance criteria
- Assist with writing acceptance tests for user stories
- Functional verification and validation of capabilities and user stories
- Writing test cases/scenarios for acceptance tests
- Performing acceptance and exploratory tests during each software iteration
- Collaborate during iteration planning
- Documentation of test results and reporting of bugs
- Learns how the system is used by the Customer

Daily Testing of Completed User Stories

Agile Team Roles - Tester

Performs acceptance and exploratory tests during each software iteration and release

Documents test results

Reports the discovery of bugs

Collaborates with developers to resolve bugs

Bug Free Software

Daily Testing of Completed User Stories

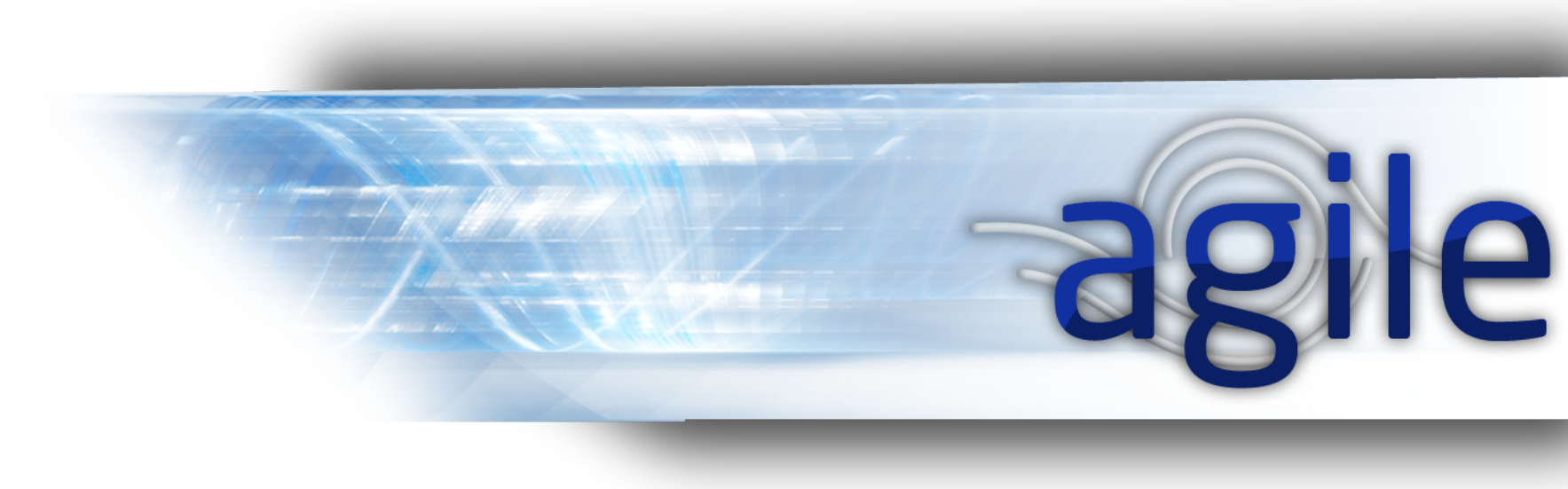
Team Roles Exercise

Identify who fills what role on your project and discuss any misunderstands among the team members

On the worksheet provided:

1. List your project name
2. For each role listed write the name of the person or persons that you believe fulfills that role
3. Once the worksheet is completed:
4. Gather as a group with your project and discuss the roles everyone has filled in
5. Discuss any misunderstandings between project members on who is in what role
6. The goal of the exercise is to have a better understanding of the roles and responsibilities of each team member

Iterations

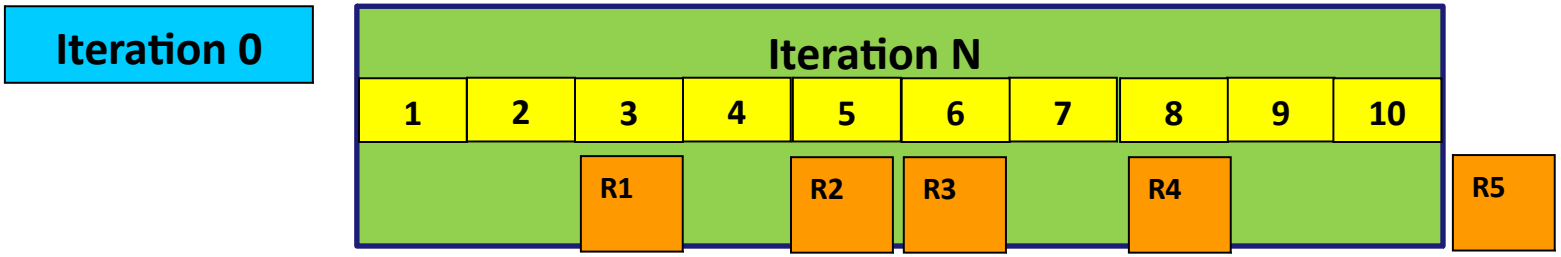


Iterations

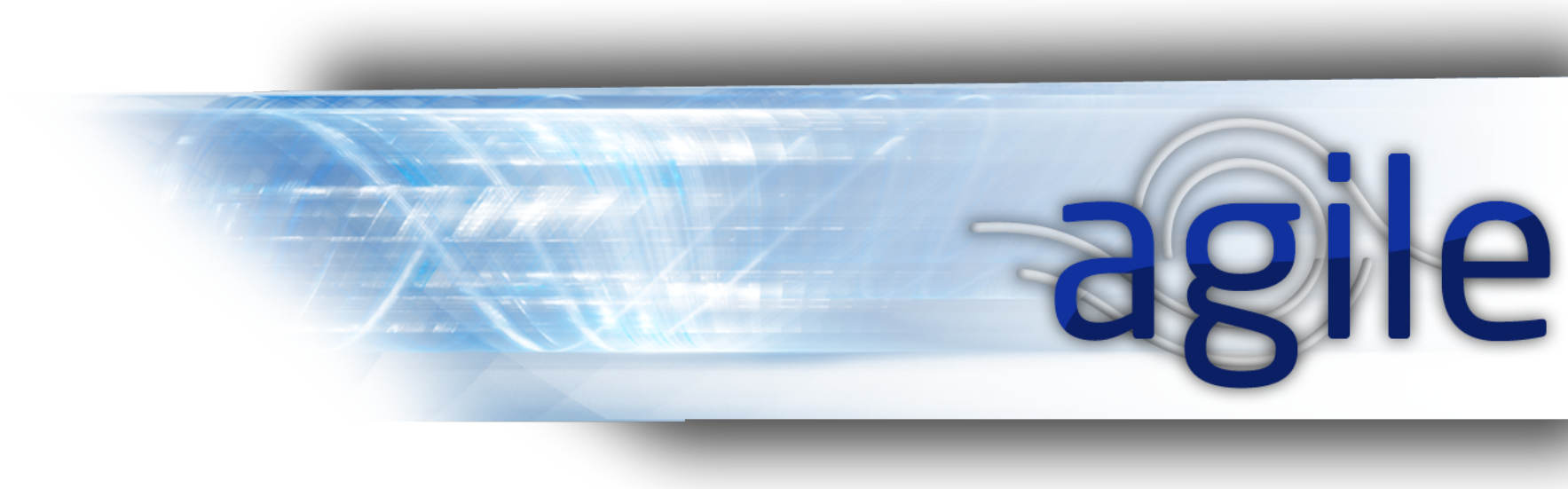
Iteration 0
Initiation and planning

Iteration N
Successive development

Iteration R
Periodic deployments



Iteration 0



Iteration 0

Finish the staffing profile and assemble the team

Create the project's issue tracking and wiki spaces

Create the initial Product Backlog of User Stories

Create a project Roadmap based off of the User Story Backlog that aligns with product and users goals

Kick Start Development

Iteration 0

Define infrastructure requirements and where the project will be hosted

- Development Servers
- Demo Servers
- Test Servers
- Production Servers
- Data feeds and System Connections

Kick Start Development

Iteration 0

Perform “Enough” architecture and system design to start development

Record initial documentation on project wiki

Work with the technical mentor to follow standards and best practices

Kick Start Development

Iteration 0

Three things everyone needs to agree on for agile to be successful:

- It is impossible to gather all requirements at the beginning of a project
- Requirements gathered up front are almost guaranteed to change
- There will always be more to do than time and money will allow

Successful agile projects:

- Gather just enough user stories in Iteration 0 to build an initial capability
- Constantly welcome change, evolve, and add new user stories to the backlog
- Frequently re-prioritize based on the highest value capabilities for users

Kick Start Development

Iteration 0

Requirements

- Difficult to gather all requirements at project beginning
- Early requirements are very likely to change
- Backlog will always be larger than what time and money allow

Agile User Stories:

- Gather “enough” information and User Stories in Iteration 0 for an Initial Capability Build
- Constantly Welcome Changing and Additional User Stories
- Frequently reprioritize Backlog based on the highest value capabilities

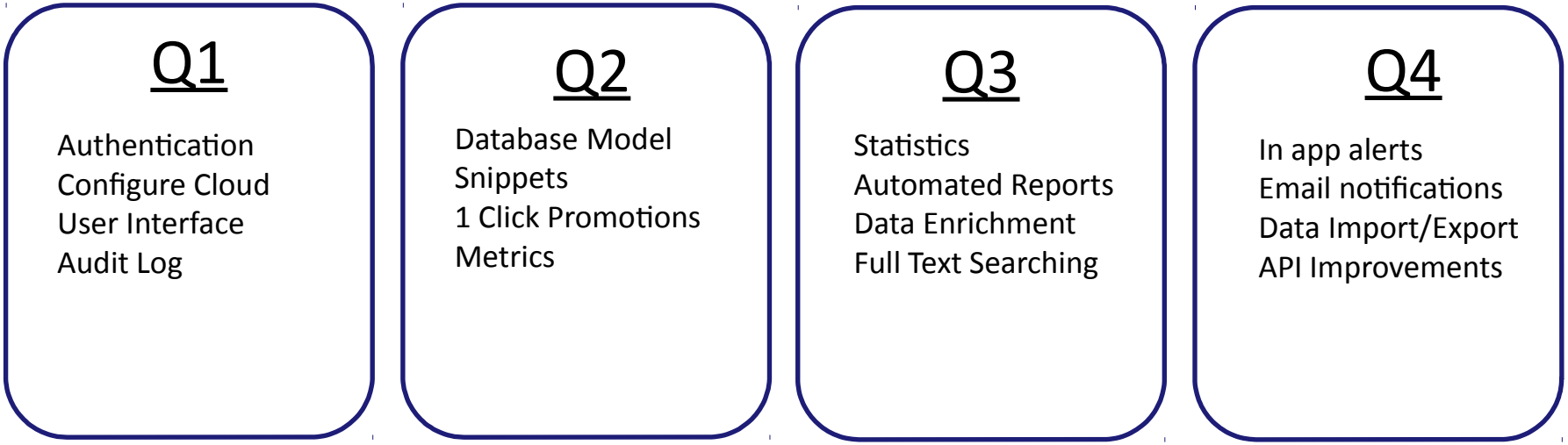
Progressive Elaboration

Iteration 0 – Capabilities

Backlog

- Contains the users highest priority request for the next six to eight months
- Use Issue tracking tool to track a constantly prioritized the Backlog

Roadmap



Progressive Elaboration

Iteration 0 – User Stories

User Stories

- As a <role>, I want <goal/desire> so that <benefit>
- As a user, I need automated notifications/email alerts on the system so that I can be notified of things I may be interested in when they are entered into the system.

Acceptance Criteria

- Update profile to include keywords for notification criteria
- Receive email notification when a new report is entered with profile keywords
- Click link in email to view report
- View report

Build product to fulfill Users needs

Iteration 0 – User Stories

Card

As an online shopper, I can ship to a friend

Conversation

Talk with Bob,
my stakeholder

Confirmation

Test: No more, No Less:

[x] Can retrieve friend's address

[ ] Can specify ship date and carrier

[ ] Can track order

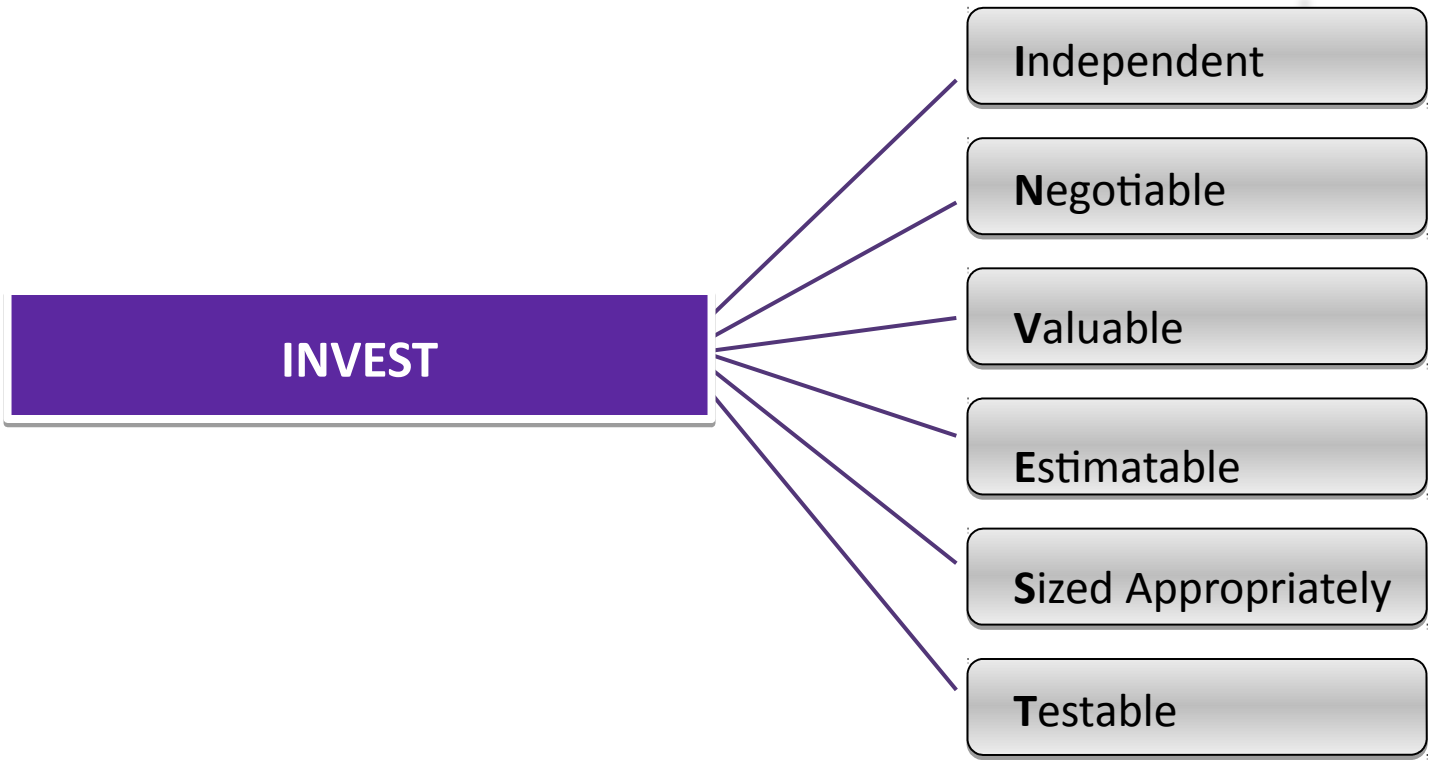
Build product to fulfill Users needs

RGB - Card details

Role	Goal	(Benefit)
As a doctor	I can see my patients medications	So I don't prescribe conflicting drugs

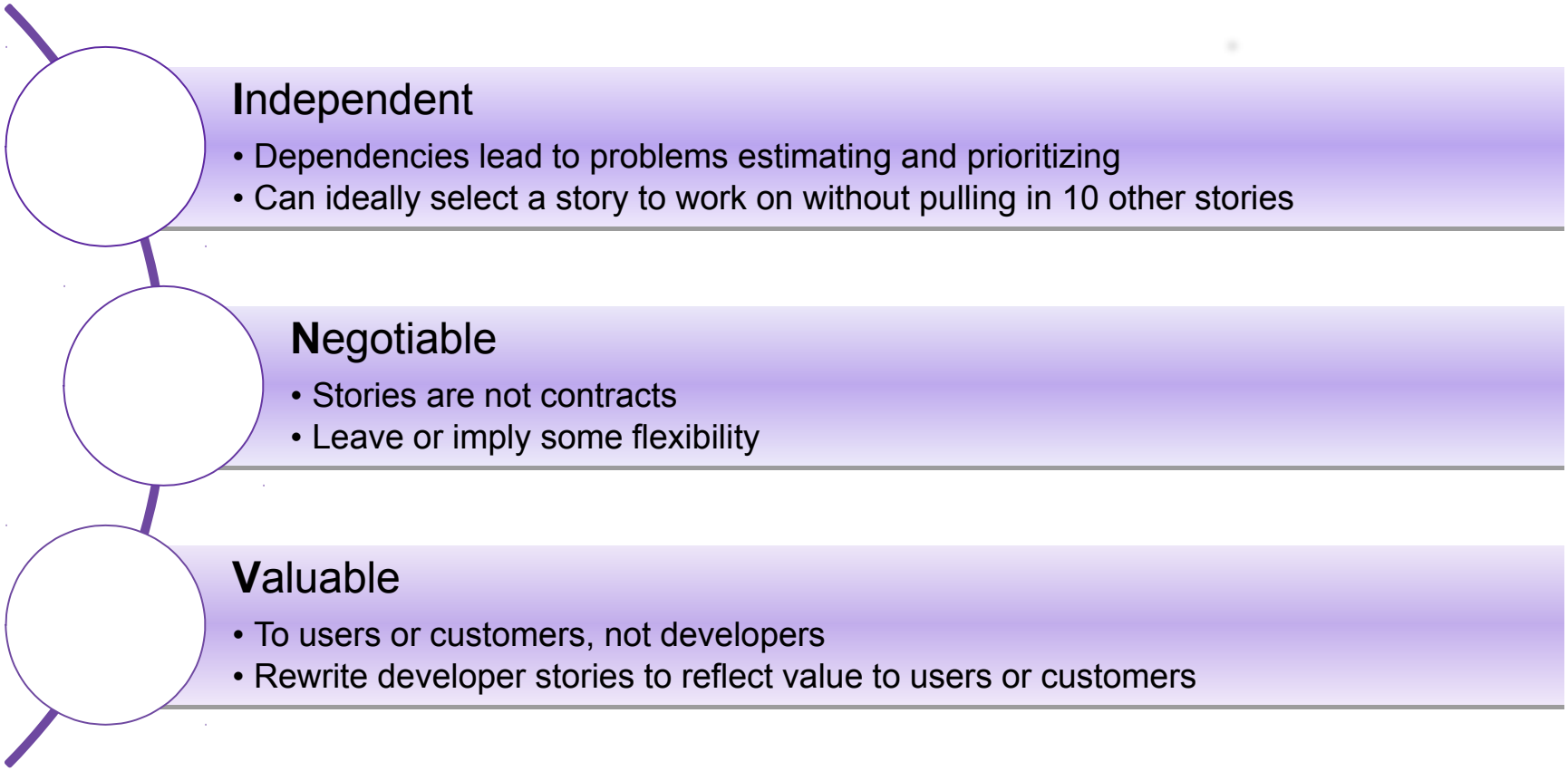
Build product to fulfill Users needs

INVEST In Good User Stories



Build product to fulfill Users needs

Attributes of a Good Story



Build product to fulfill Users needs

Attributes of a Good Story

Estimable

- Because plans are based on user stories, we need to be able to estimate them

Sized appropriately

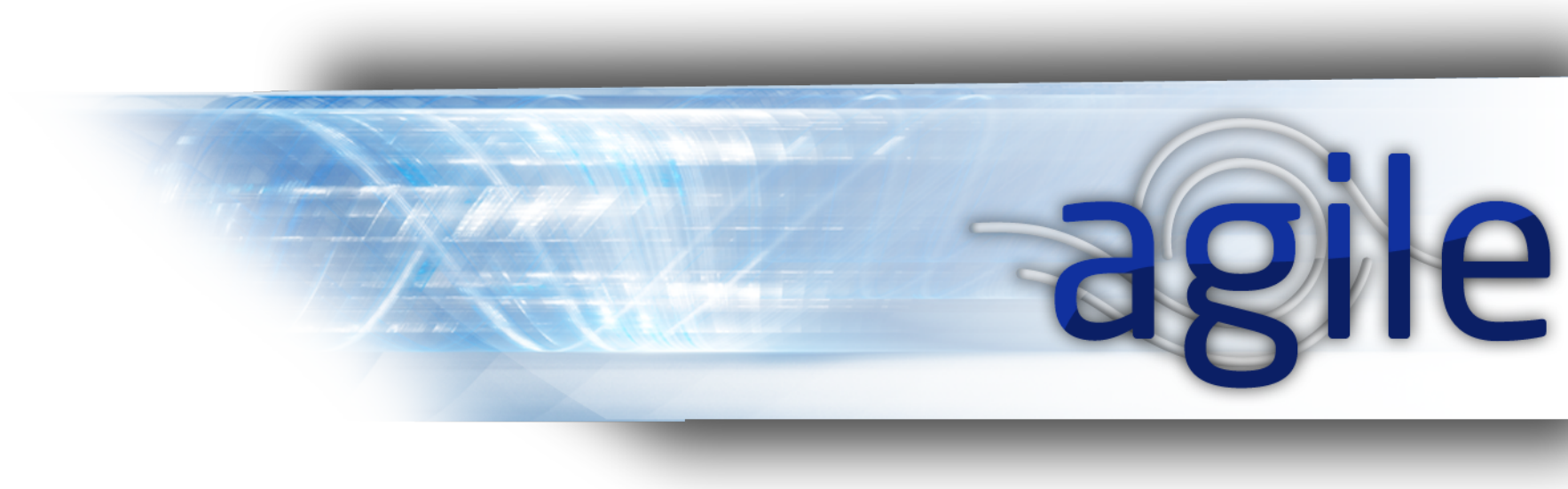
- Small enough to complete in one iteration if you are about to work on it
- Bigger if further off on the horizon

Testable

- Testable so that you have an easy, binary way of knowing whether a story is finished
- Done or not done; no “partially finished” or “done except”

Build product to fulfill Users needs

Iteration N



Iteration N

Continuously re-prioritize Product Backlog and write code

Iteration Repetition and Rhythm:

- Document project, design, development, test, and deployment information on the projects wiki
- Check in code regularly to support continuous integration and early testing
- Deploy the code to a demo or test area for user review and feedback
- Perform an Iteration Retrospective in order to continue to improve the project's processes

Repetition and Rhythm

Iteration N - Meetings

Necessity to communicate project needs, make informed decisions, and have technical discussions.

Developers should not be required to attend management or status meetings
Automated reporting tools can be used to provide status of the agile team

All team members are invited to attend meetings as long as they are getting something of value out of the meeting or giving something of value to the other attendees.

Communicate Needs

Iteration N - Meetings

Backlog Prioritization

- Customer focused
- Discuss feature improvement and new features to be implemented
- Prioritization of features and requirements
- Progressive elaboration 6-8 months out
- At a minimum once a quarter
- Developer Involvement

Communicate Needs

Iteration N - Meetings

Iteration Planning

- Routine and habitual - same time same place every other week
- Developers sign up for and commit to user stories and task
- User Stories and task are not assigned

Communicate Needs

Iteration N - Meetings

Daily Stand-ups 15 min or less

- Always starts on time
- Detailed technical discussion is deferred to after Daily Stand-up
- Focuses team on immediate task and deliverables

Ask everyone

- What did you accomplish yesterday?
- What do you plan on doing today?
- What is stopping or slowing your progress?

Communicate Needs

Iteration N - Meetings

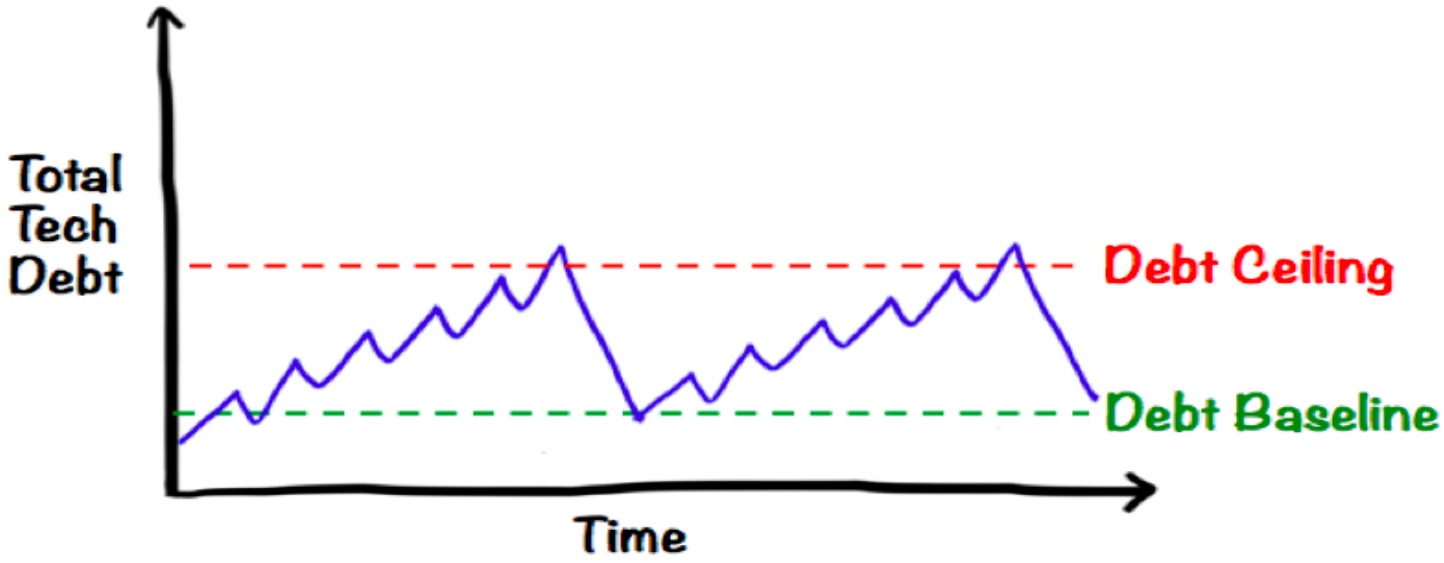
Daily Stand-up Exercise

Communicate Needs

Iteration N - Meetings

Technical Exchange

- Routine and habitual - same time same place every other week
- In the weeds technical discussion
- Improve code quality and reduce technical debt
- Developer focused



Communicate Needs

Iteration N - Meetings

Pre-Release coordination

- Coordinate with Testers, Customer Support, Deployment team

Demonstrations

- User focused to show and demonstrate the features that have been implemented

Retrospective

- What should we continue doing?
- What should we stop doing?
- What do we need to start doing to improve?

Communicate Needs

Iteration N – Task Estimation

Done

- User Story that passes its acceptance test criteria
- For code the User Story must be unit tested, code developed, code checked in to source control, and pass all unit test

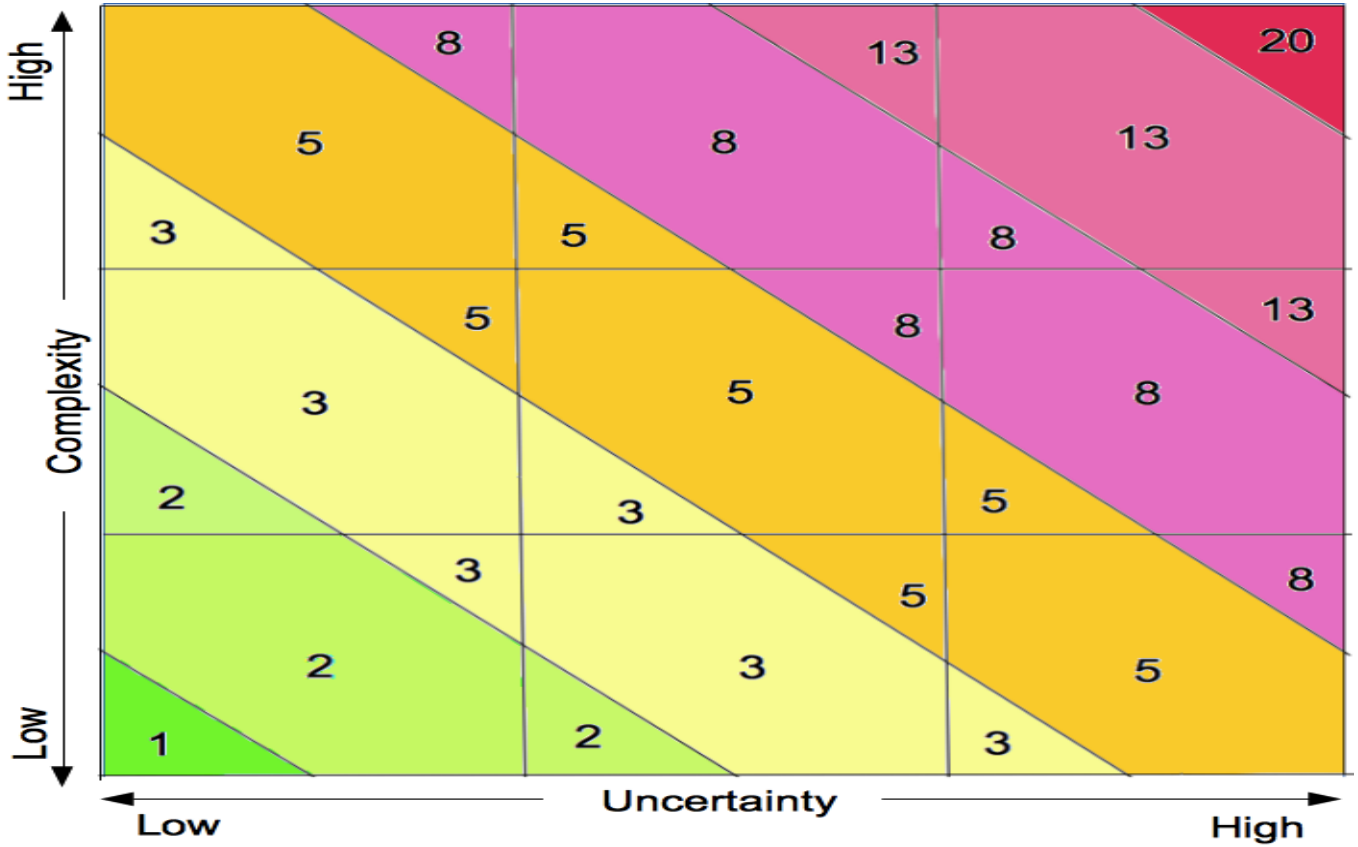
Simple Task Estimation

- Level of effort (LOE) 1-5 to each task
- Where a LOE of 1 is equal to 20% and 5 being equal to 100% of the available time the person has during that iteration

No estimation process is ever 100% accurate

Iteration N – Task Estimation

Fibonacci Task Estimation



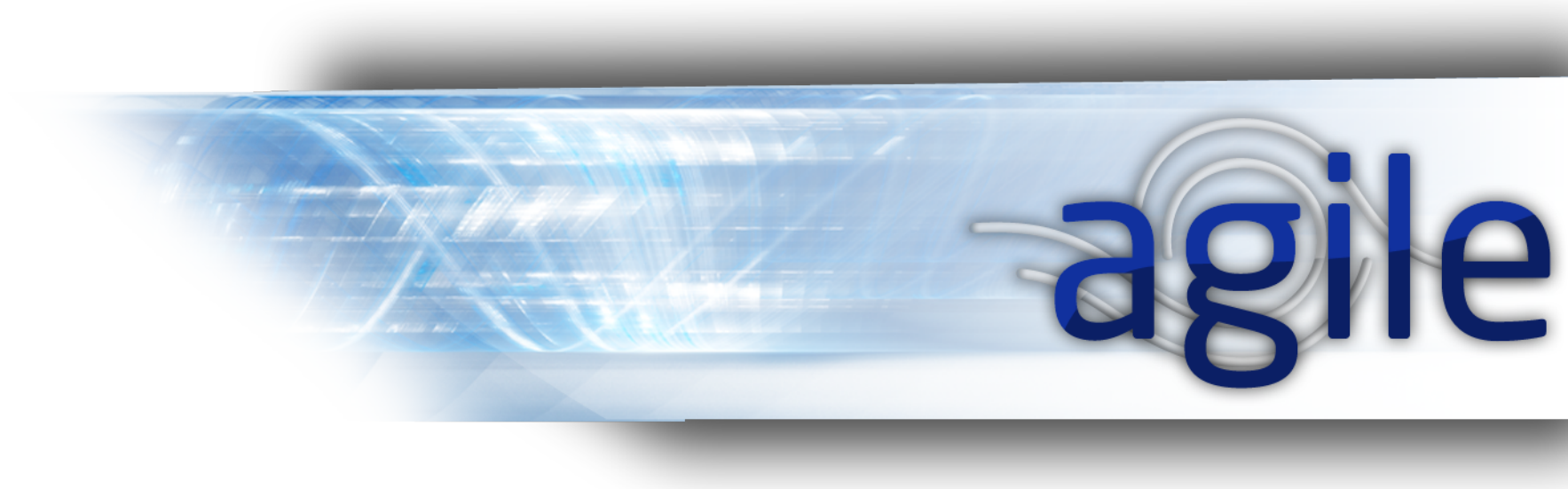
No estimation process is ever 100% accurate

Sample Schedule

Mon	Tues	Wed	Thurs	Fri
Code/Test	Code/Test Demo Retrospective	**Iteration Planning**	Code/Test	Code/Test
Code/Test	Code/Test	Code/Test	Code/Test	Code/Test
Code/Test	Code/Test Demo Retrospective	**Iteration Planning** Code/Test	Code/Test	Code/Test

Two Week Iteration

Iteration R



Iteration R

Release Management

Finalize documentation

- Users Guides
- Config Files
- Test Results
- API Documentation

Deploy the release to the test environment for integration testing

Record the test results in a wiki and enter any bugs into tracking tool

Deploy the new software version to the production environment

Development Continues

Iteration R - Version Naming Conventions

Major	Major software release when you make incompatible API changes to include database schema updates or changes to the technology stack
Minor	Minor Improvements, bug fixes, in a backwards-compatible manner
Patch	Version when you make backwards-compatible bug fixes
Iteration	*Once and Iteration - Identifies the current development cycle*
Build	*Multiple times daily - Identifies a working software build*

Major.Minor.Patch.Iteration.Build >> Webapp_v1.2.2.1

Naming Conventions – SemVer http://semver.org

Iteration R – Committing Changes

Guidelines for committing software:

- Make sure the local workspace matches the latest checked into SCM before doing a commit (most tools already do that)
- Compare changes with the latest checked in to SCM
- Keep commits small and logical
- All commits need to have comments describing what changed and the associated task
- Build and test affected modules before committing
- Ensure coding standards are followed (i.e., no commented code)
- Review other developer commits periodically

Source Control

Iteration R – Continuous Integration

Guidelines for Continuous Integration:

- Automated builds should be executed at every commit or at least once daily
- Sixty percent unit test code coverage is ideal; 70% or more is exceptional
- Avoid excessive unit testing
- Automated notification to the enter development team when a build or unit test fails

Automated Builds

Iteration R – Code Reviews

Guidelines for Code Reviews:

- No formalized code review meetings
- Work as a team, when a defect is found treat it as a learning opportunity
- Review major features
- Make request to have someone review your code
- Review small sets of code 500 lines limit per code review
- Focus on high impact code, reviews should be focused on core modules that most of the project depends on

Team Ownership of Code

Iteration R - Testing

Manual Testing
- Needed for major version updates
- Not scalable

Automated Testing
- Test Driven Development
- Code coverage reports
- Browser automation (Selenium)

Test Early Test Often

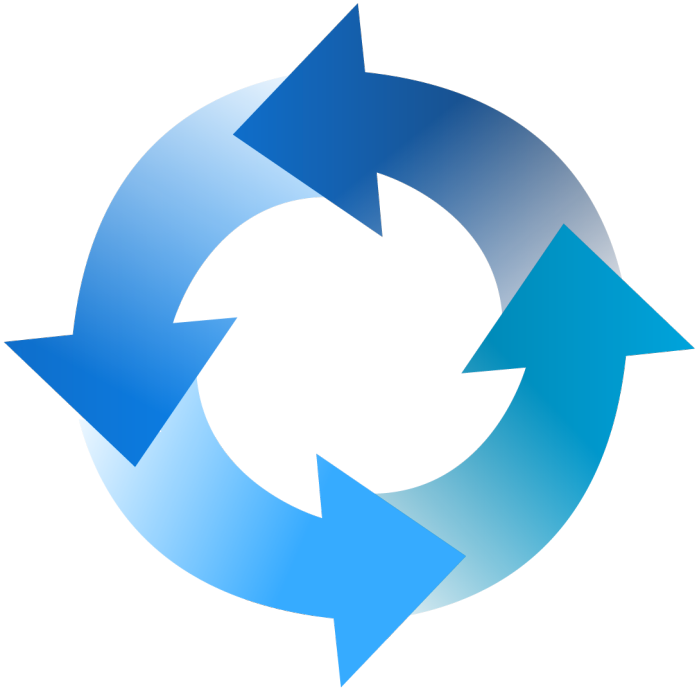
Iteration R - Testing

Acceptance testing

- Push feature updates to a staging area after every iteration
- Acceptance testing should be done by the users and systems engineers

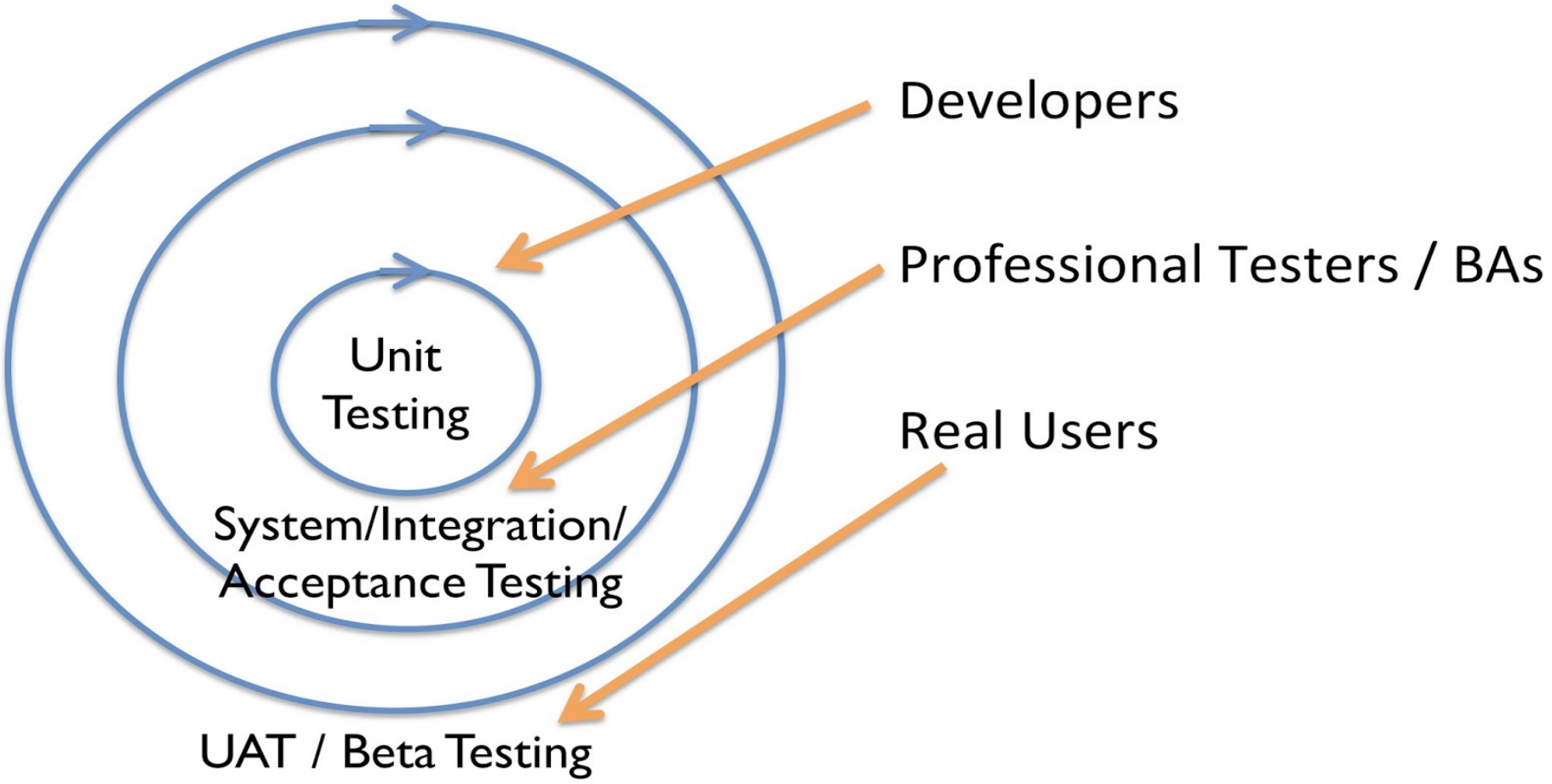
User Feedback

- Feature improvements
- New features request



Test Early Test Often

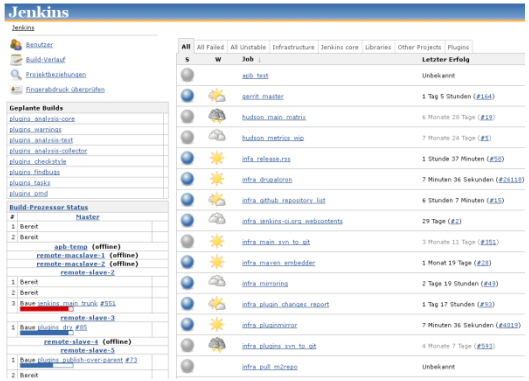
Iteration R - Testing



Test Early Test Often

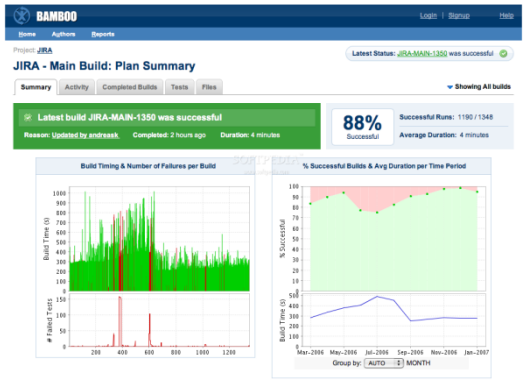
Iteration R - Deployments

Automated Software Configuration Management
- Subversion
- Git
- Nexus Repository



Continuous Integration
- Bamboo
- Jenkins/Hudson

Automated Deployments
- Custom Scripts
- Remote Package Manager (RPM)
- Capistrano

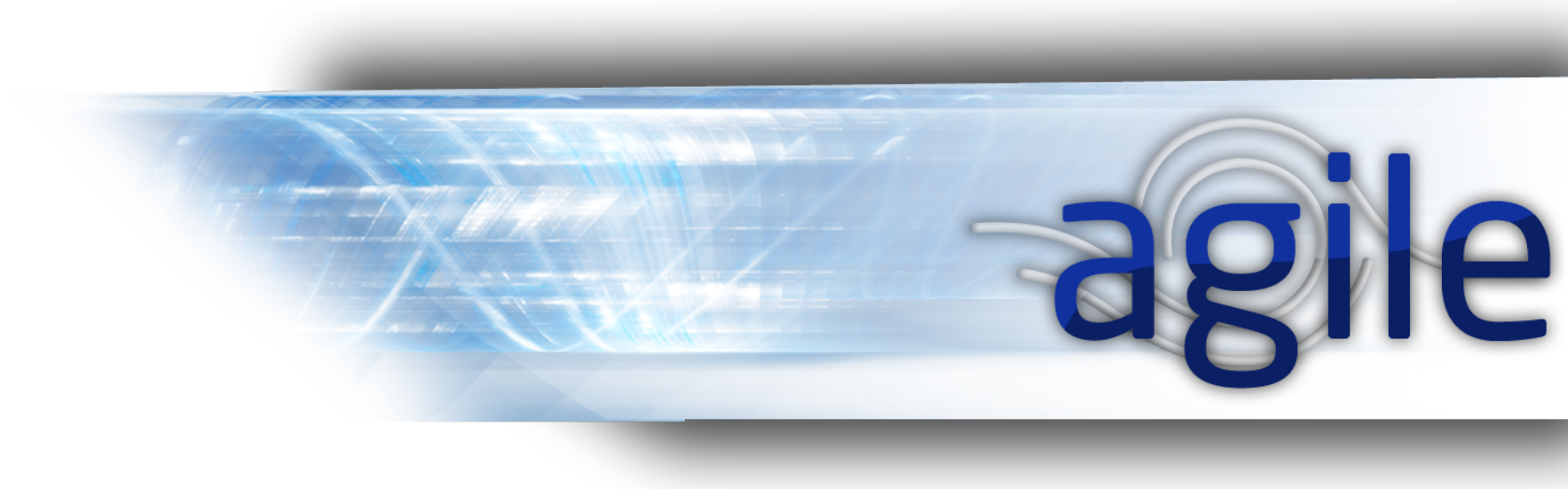


Deploy Early Deploy Often

Agile Exercise

1. Have a Backlog Prioritization Session with the Customer
2. Develop initial Backlog
3. Develop initial RoadMap
4. Define server environments
5. Define staffing plan with team roles
6. Define development cycles
   - How often will you have a Backlog Prioritization meeting?
   - What time will you have your daily stand-ups, will they be virtual?
   - How long will your iterations be?
   - When will iterations start, when will they end?
   - How will customers/users view your product and provide feedback
7. Develop three users stories based of the scenario with acceptance criteria

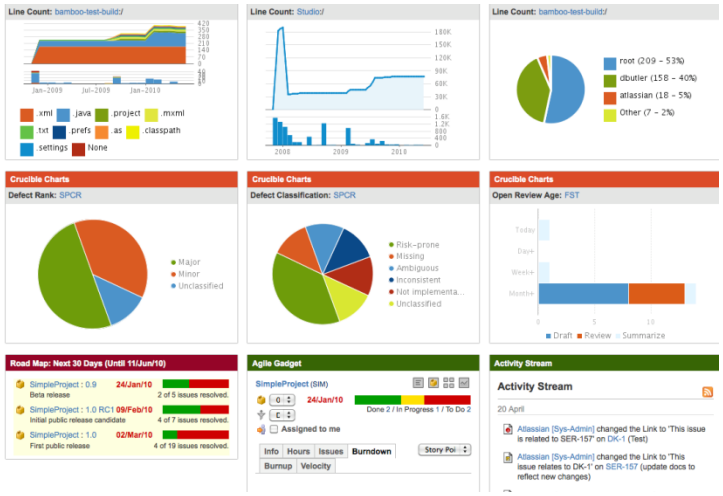
Metrics



Metrics

Automated Metrics

- Unit Test Coverage
- Burn-up and Burn-down charts
- Estimated / Actual Velocity
- Known issues
- Bugs closed per Iteration
- Escaped Defects



Information Radiators – No meeting required

Metrics

Non-Automated Metrics

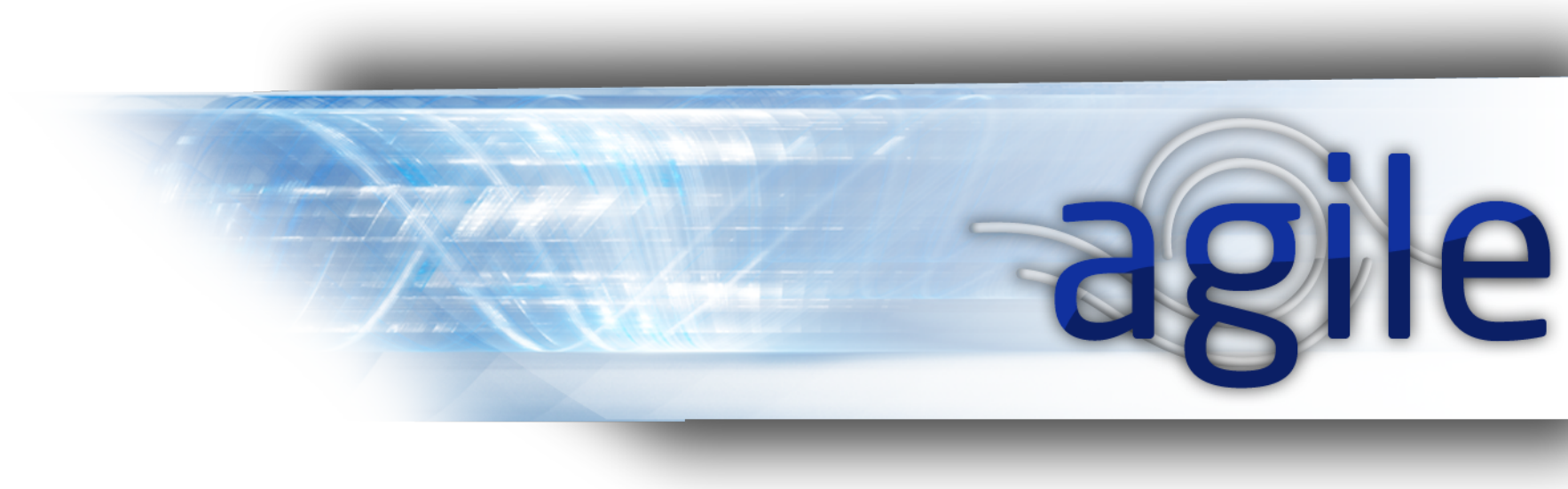
- Is the user community generally satisfied with the software
- Iterations per production release

Information Radiators – No meeting required

Agile Checklist

Best Practices		Best Practices
Unit Testing		Pair Programming
Continuous Integration		User Stories with Acceptance Criteria
Defined Coding Standards		Dedicated Users
Definition of Done		
Automated Builds		
Ability to Reduce Technical Debt		**Meetings**
Velocity Measures		Daily Standup
Working software after an Iteration		Iteration Planning
Use of Burndown/Burnup Charts		Backlog Prioritization with Users
Work Committed to not Assigned		Roadmapping Sessions
Those Doing the Work Provide Estimates		Release Planning
Use of a Prioritized Backlog		Retrospectives
Feature request maintained in a Backlog not in System Requirements Documents		Demonstrations
Collective Code ownership		Technical Exchanges

Tools



Tools

Agile Ticket Tracking	Free	Website
Trac	X	http://trac.edgewall.org/
Redmine	X	http://www.redmine.org/
Huboard	X	https://huboard.com/
Github	X	http://www.github.com/
JIRA		https://jira.atlassian.com/secure/Dashboard.jspa
FogBugz		http://www.fogcreek.com/fogbugz/

Build Management	Free	Website
Jenkins	X	http://jenkins-ci.org/
TravisCI	X	http://travis-ci.org/
CruiseControl	X	http://cruisecontrol.sourceforge.net/
Bamboo		http://www.atlassian.com/software/bamboo/overview
RPM's	X	http://en.wikipedia.org/wiki/RPM_Package_Manager
Capistrano	X	https://github.com/capistrano
Maven	X	http://maven.apache.org/

The Agile Practices

Product / Project Definition

- Product visioning
- Project charting
- Affinity (relative) estimation
- Size-base (point) estimation
- Group estimation
- Value-based documentation
- Prioritized product backlog
- User stories
- Progressive elaboration
- Personas
- Story Maps / MMF
- Story slicing
- Acceptance tests as requirements

Planning and Cadence

- Short iterations
- WIP Limits
- Early and frequent releases
- Roadmapping
- Velocity-based planning and commitment
- Iteration planning / Iteration backlog
- Release planning / Release backlog
- Time boxed iterations
- Sustainable pace

Continuous Learning

The Agile Practices

Communication

- Informative workspace
- Osmotic communication
- Team room
- Daily standup meeting
- Retrospectives
- User interface prototyping
- Team design
- Collective code ownership
- Definition of done

Testing

- UAT
- Automated functional tests
- Automated developer tests (unit tests)
- Exploratory testing
- Software metrics

Continuous Learning

Agile Practices

Human Centric

- Frequent face-to-face
- Team chartering
- Collaborative teams
- Self-organizing teams
- Cross-functional teams
- Servant leadership
- Task volunteering
- Generalizing specialist
- Human Centric
- Burn –up / burn-down charts

Delivery Practices

- Refactoring
- Incremental design
- Automated builds
- Version Control
- Configuration management
- Test driven development
- Pair programming
- Continuous integration
- Incremental deployment
- Simple design

Continuous Learning

Agile Resources on the Web

Simple Model for Agile Development - http://www.pmi.org/Knowledge-Center/Knowledge-Shelf/Agile-Practices.aspx
Agile Samurai book and website - http://agilewarrior.wordpress.com/
Continuous Integration - http://en.wikipedia.org/wiki/Continuous_integration
Behavior Driven Development - http://en.wikipedia.org/wiki/Behavior_Driven_Development
Test Driven Development - http://en.wikipedia.org/wiki/Test-driven_development
Requirements - http://www.scrumalliance.org/articles/398-agile-requirements-definition-and-management
User Stories - http://en.wikipedia.org/wiki/User_story
User Stories - http://www.mountaingoatsoftware.com/topics/user-stories
Refactoring - http://en.wikipedia.org/wiki/Code_refactoring
Technical Debt - http://en.wikipedia.org/wiki/Technical_debt
Code Smell - http://en.wikipedia.org/wiki/Code_smell
Software Rewrite- http://www.joelonsoftware.com/articles/fog0000000069.html
Software Rewrite - http://en.wikipedia.org/wiki/Rewrite_%28programming%29
Software Development - http://discursive.com/2013/10/15/the-7-rules-of-software-development

Continuous Learning