Agile is ...

- a way of thinking
- a mental inclination or disposition
- a frame of mind
- a mindset of learning & discovering how to manage uncertainty
- an adjective not a noun

Describing an Agile Mindset

To people who don't develop software for a living, an agile mindset involves two key aspects:

1. Deliver maximum outcome with minimum output.
2. Seek short feedback cycles so you can learn and adapt quickly.

Embracing Agile should not be your end goal!!!

Agility Defined...

- "Agility is the ability to both create and respond to change in order to profit in a turbulent business environment.
- Agility is the ability to balance flexibility and stability"

Jim Highsmith (2002)

Agility - What to Do

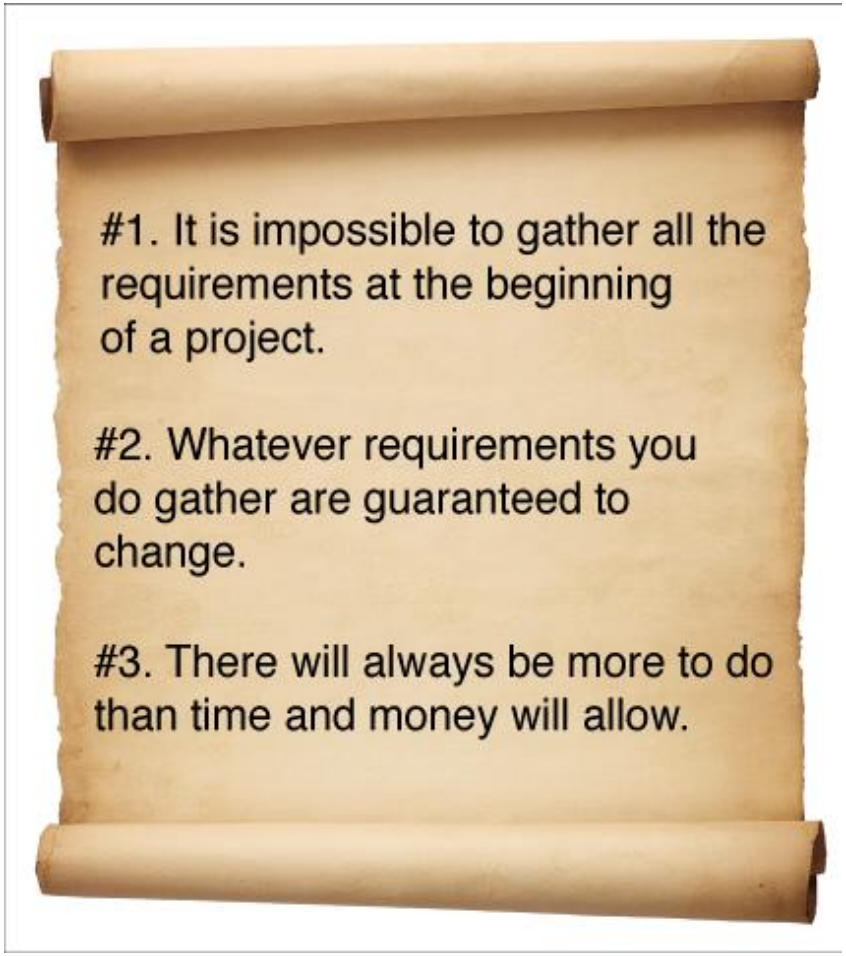
- Find out where you are
- Take a small step toward your goal
- Adjust your understanding based on what you learned
- Repeat

Agility - How to Do It

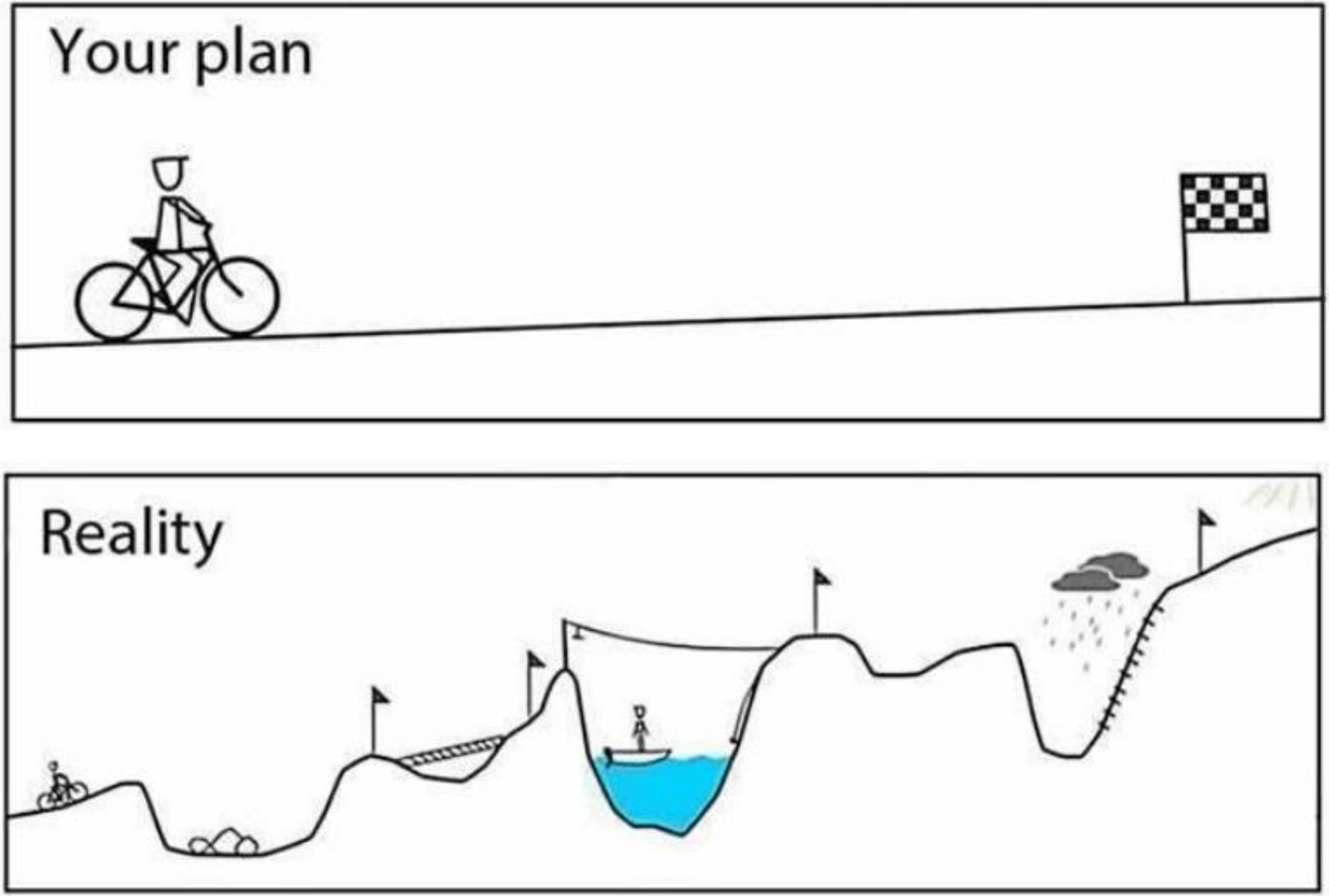
- When faced with two or more alternatives that deliver roughly the same value, take the path that makes future change easier

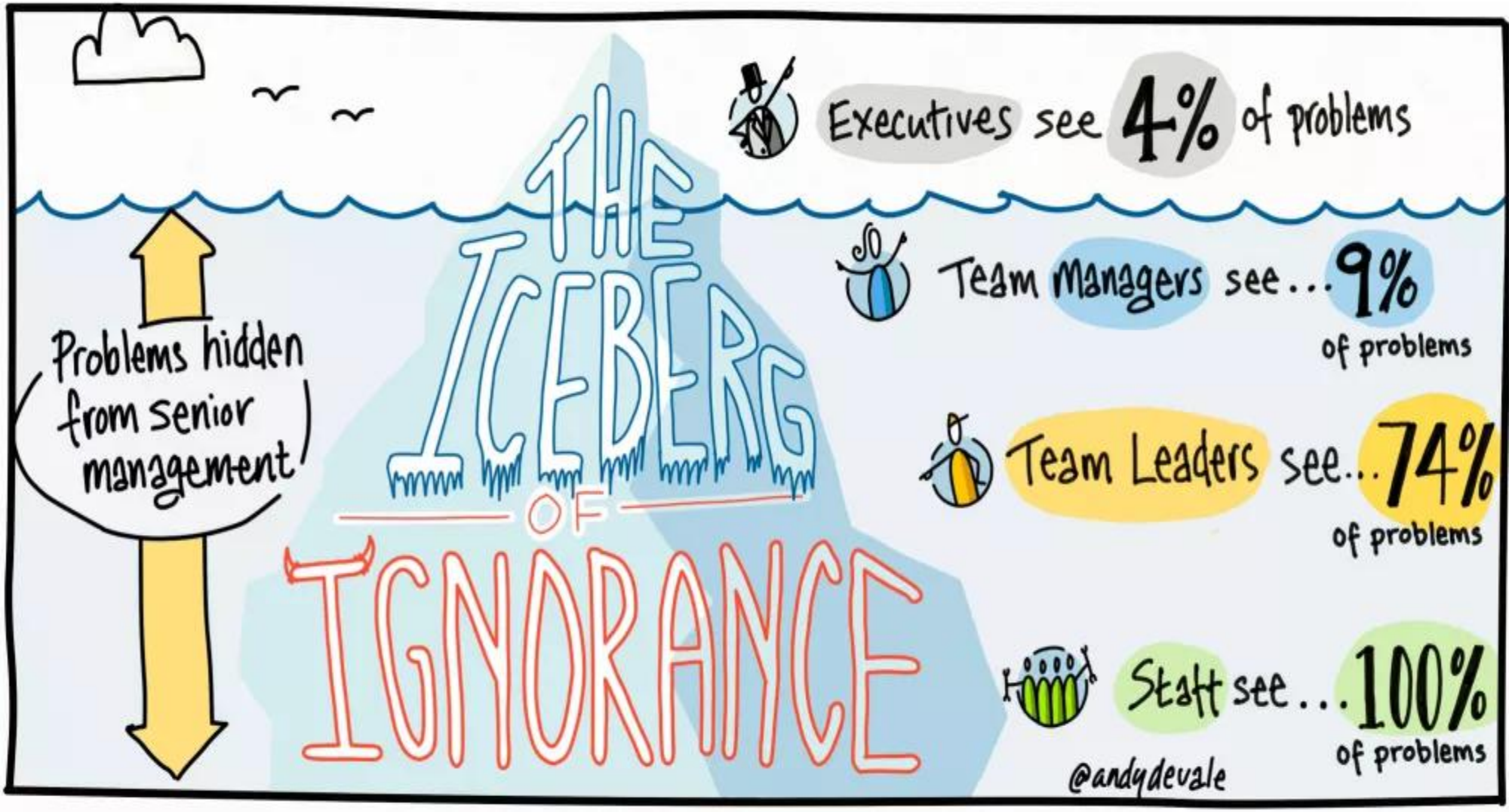
Dave Thomas, speech delivered at GOTO Amsterdam 2015

Three Simple Truths



- Accepting the first truth means you are not afraid to begin your journey without knowing everything up front. You understand that requirements are meant to be discovered and that not proceeding until all are gathered would mean never starting.
- Accepting the second means you no longer fear or avoid change. You know it is coming. You accept it for what it is. You adapt your plan when necessary and move on.
- And by accepting the third, you no longer get stressed when your to-do list exceeds your time and resources to deliver. This is the normal state for any interesting project. You do the only thing you can—you set some priorities, get the most important stuff done first, and save the least important for last.
- Once accepted, much of the stress and anxiety traditionally associated with software delivery disappears. You are then able to think and innovate with a level of focus and clarity that escapes most in this industry.


Your plan
Reality



https://corporate-rebels.com/iceberg-of-ignorance/

The Dead Horse Theory

The tribal wisdom of the Dakota Indians, passed on from generations to generations, says that, “When you discover that you are riding a dead horse, the best strategy is to dismount.”

However, in modern business, education and government, a whole range of far more advanced strategies are often employed, such as:

1. Buying a stronger whip
2. Changing riders
3. Threatening the horse with termination
4. Appointing a committee to study the horse
5. Arranging to visit other countries to see how others ride dead horses
6. Lowering the standard so that dead horses can be included
7. Re-classifying the dead horse as “living imparied”
8. Hiring outside contractors to ride the dead horse
9. Harnessing several dead horses together to increase the speed
10. Providing additional funding and/or training to increase the dead horse's performance
11. Doing a productivity study to see if lighter riders would improve the dead horse's performance
12. Deciding that as the dead horse does not have to be fed, it is less costly, carries lower overhead, and therefore contributes substantially more to the bottom line of the company than do some other horses
13. Re-setting the expected performance requirements for all horses
14. Promoting the dead horse to a supervisory position

HOW WELL CAN YOU PREDICT THE RESULTS OF YOUR ACTIONS?

complexity

Characteristics: The situation has many interconnected parts and variables. Some information is available or can be predicted, but the volume or nature of it can be overwhelming to process.

Example: You are doing business in many countries, all with unique regulatory environments, tariffs, and cultural values.

Approach: Restructure, bring on or develop specialists, and build up resources adequate to address the complexity.

volatility

Characteristics: The challenge is unexpected or unstable and may be of unknown duration, but it's not necessarily hard to understand; knowledge about it is often available.

Example: Prices fluctuate after a natural disaster takes a supplier off-line.

Approach: Build in slack and devote resources to preparedness—for instance, stockpile inventory or overbuy talent. These steps are typically expensive; your investment should match the risk.

ambiguity

Characteristics: Causal relationships are completely unclear. No precedents exist; you face "unknown unknowns."

Example: You decide to move into immature or emerging markets or to launch products outside your core competencies.

Approach: Experiment. Understanding cause and effect requires generating hypotheses and testing them. Design your experiments so that lessons learned can be broadly applied.

uncertainty

Characteristics: Despite a lack of other information, the event's basic cause and effect are known. Change is possible but not a given.

Example: A competitor's pending product launch muddies the future of the business and the market.

Approach: Invest in information—collect, interpret, and share it. This works best in conjunction with structural changes, such as adding information analysis networks, that can reduce ongoing uncertainty.

HOW MUCH DO YOU KNOW ABOUT THE SITUATION?

MANIFESTO FOR AGILE SOFTWARE DEVELOPMENT

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.

http://www.agilemanifesto.org/

Principles behind the Agile Manifesto

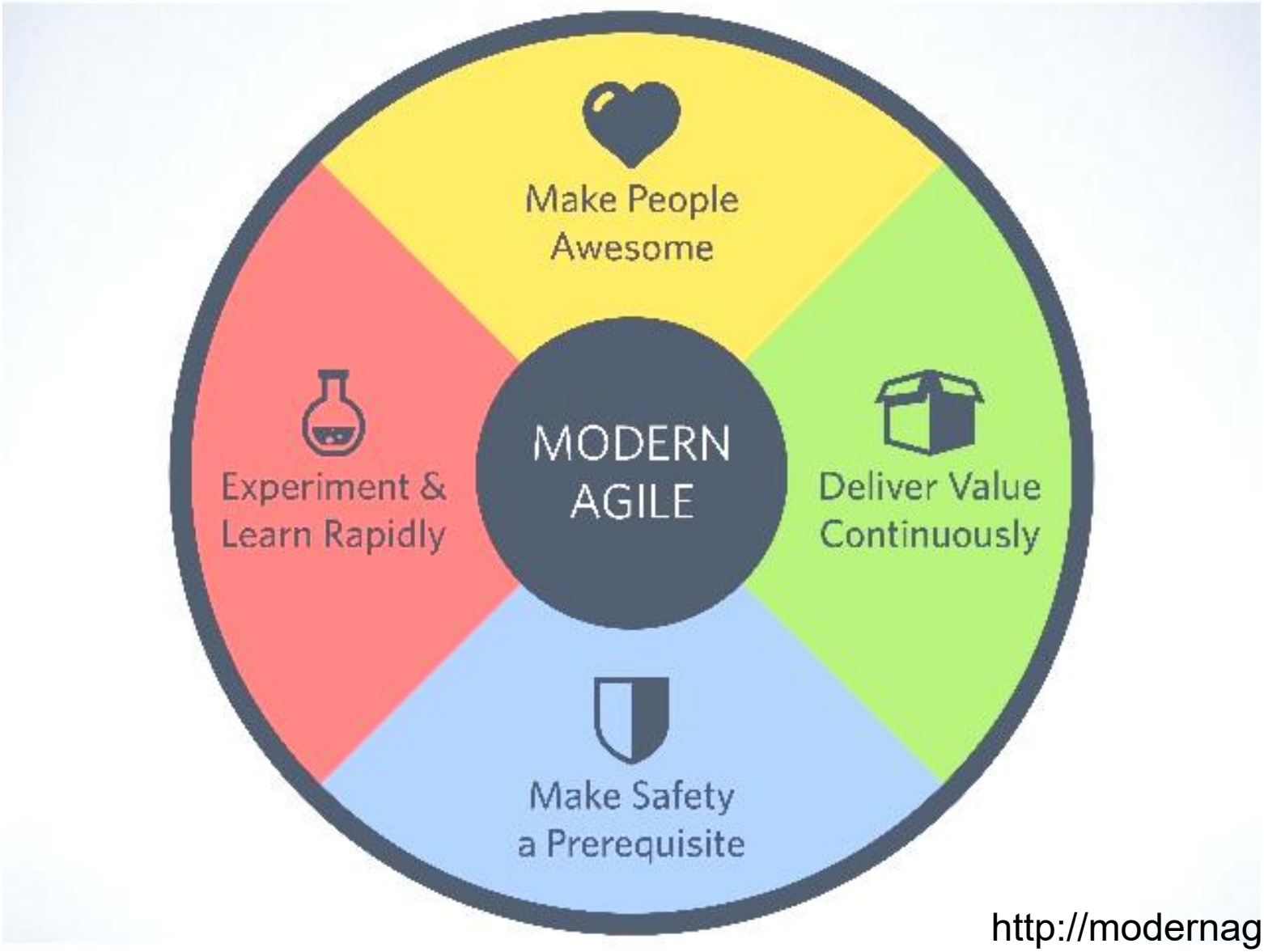
1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a 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 is face-to-face conversation.
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 effective, then tunes and adjusts its behavior accordingly.

http://www.agilemanifesto.org/principles.html

The 12 Principles Paraphrased

1. Deliver value through satisfying your customer's needs.
2. You don't know everything when you start a new endeavor. Structure your approach so that you can learn and adjust.
3. Strive for short feedback cycles to aid the learning mentioned in #2.
4. Don't construct silos. Keep your teams cross-functional.
5. Pay attention to the people doing the work. Trust them. Give them the environment and support they need.
6. Remove as many barriers to collaboration as you can for that cross-functional team so that they can have short feedback cycles.
7. Determine the outcome you're trying to achieve and measure progress and success based on whether you're delivering that outcome.
8. Keep a sustainable pace. Working overtime is not effective. People can't multitask. Instead of trying to do a bunch of things at once, focus on the most important thing.
9. Follow the appropriate practices for building your product or delivering your service. Pick the ones that allow you to learn and adapt.
10. Simplicity — the art of rationalizing the amount of work not done — is essential.
11. Let the people who do the work figure out how they're going to do the work based on their knowledge and experience.
12. Use short feedback cycles to reflect and adapt.

Modern Agile Guiding Principles



http://modernagile.org/

Make People Awesome

Steve Jobs used to ask his colleagues, “What incredible benefits can we give to the customer? Where can we take the customer?" In modern agile we ask how we can make people in our ecosystem awesome. This includes the people who use, make, buy, sell or fund our products or services. We learn their context and pain points, what holds them back and what they aspire to achieve. How can we make them awesome?

Make Safety a Prerequisite

Safety is both a basic human need and a key to unlocking high performance. We actively make safety a prerequisite by establishing safety before engaging in any hazardous work. We protect people’s time, information, reputation, money, health and relationships. And we endeavor to make our collaborations, products and services resilient and safe.

Experiment & Learn Rapidly

You can’t make people awesome or make safety a prerequisite if you aren’t learning. We learn rapidly by experimenting frequently. We make our experiments “safe to fail” so we are not afraid to conduct more experiments. When we get stuck or aren’t learning enough, we take it as a sign that we need to learn more by running more experiments.

Deliver Value Continuously

Anything that isn’t delivered isn’t helping anyone become more awesome or safe. In modern agile we ask ourselves, “How could valuable work be delivered faster?” Delivering value continuously requires us to divide larger amounts of value into smaller pieces that may be delivered safely now rather than later.

Manifesto for Agile Software Development

We are uncovering better ways of ~~developing software by doing it and helping others do it.~~

We are *uncovering better ways of getting awesome results.*

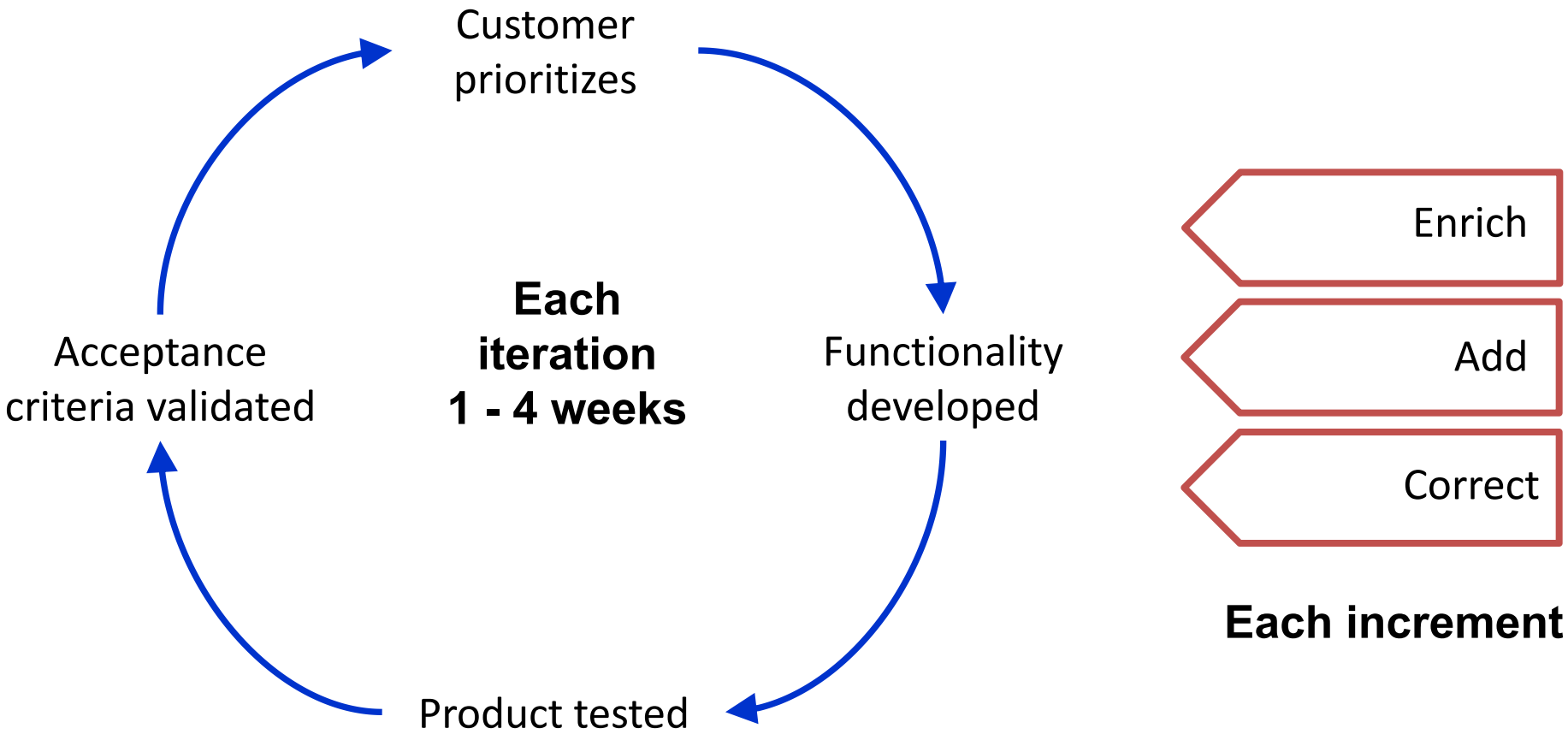
Characteristics that Make Up the Agile Mindset

- ***Positive attitude***
  - There are always challenges on projects; people are human and make mistakes, and everything is not always going to go well.
- ***Thirst for knowledge***
  - Agile is about learning and adapting.
- ***Goal of team success***
  - It is more important for the team to succeed than for the individual to have completed their tasks.
- ***Pragmatism***
  - It is critical that the team understands what is important to the business and then deals with things sensibly and realistically in a way based on practical rather than theoretical considerations.
- ***Willingness to fail***
  - Innovation often comes from trying things that you may not have thought of carrying out during your normal tasks.

Agile Business Objectives

1. **Continuous innovation** - to deliver on current customer requirements
2. **Product adaptability** - to deliver on future customer requirements
3. **Improved time-to-market** - to meet market windows and improve ROI
4. **People and process adaptability** - to respond rapidly to product & business change
5. **Reliable results** - to support business growth & profitability

Agile development is iterative & incremental



Traditional vs. Agile

TRADITIONAL

Production ready

First: Completely work out an idea
Then: Extremely accurate estimation

Time

AGILE

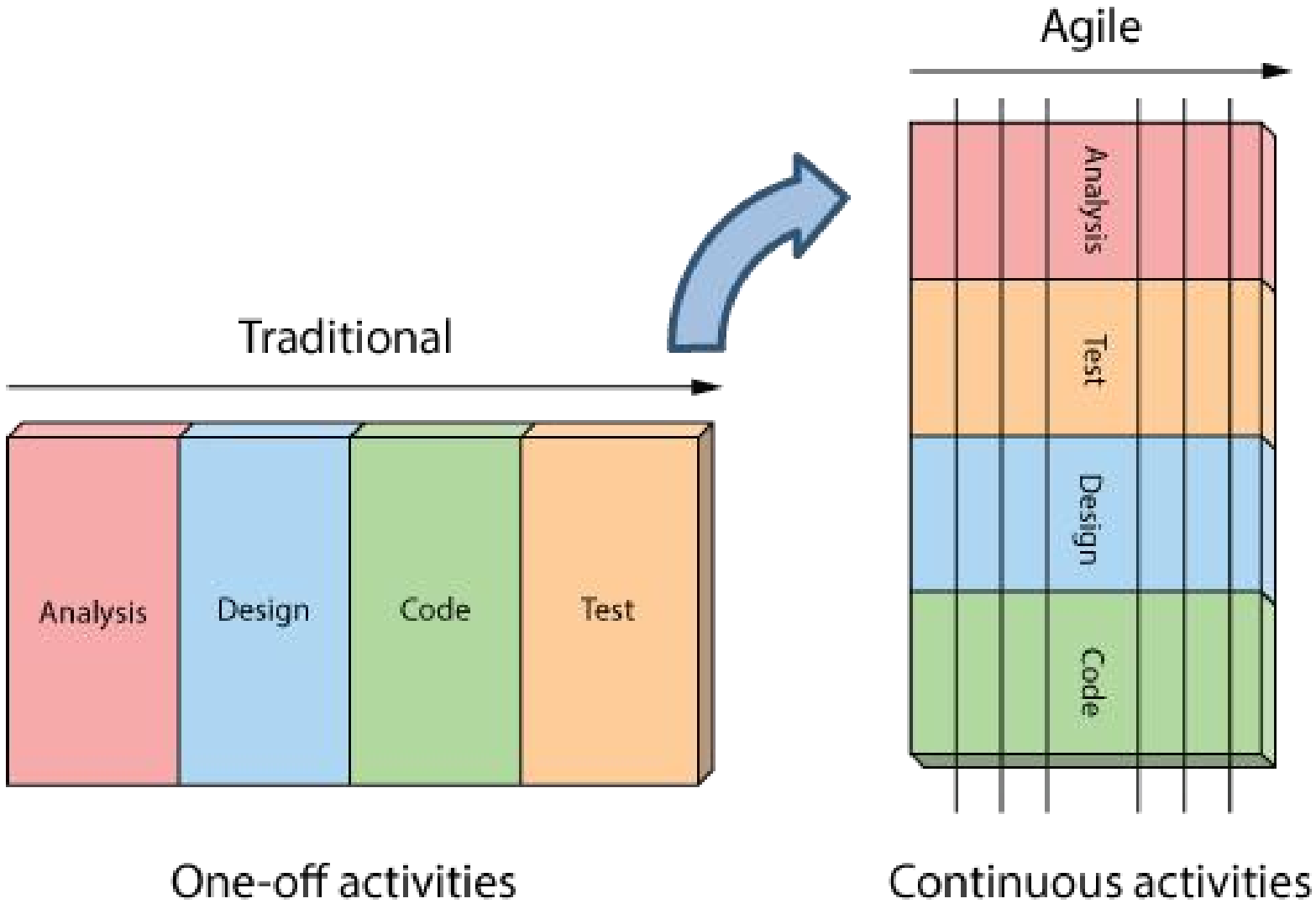
Maybe this was already sufficient!!

Always production ready

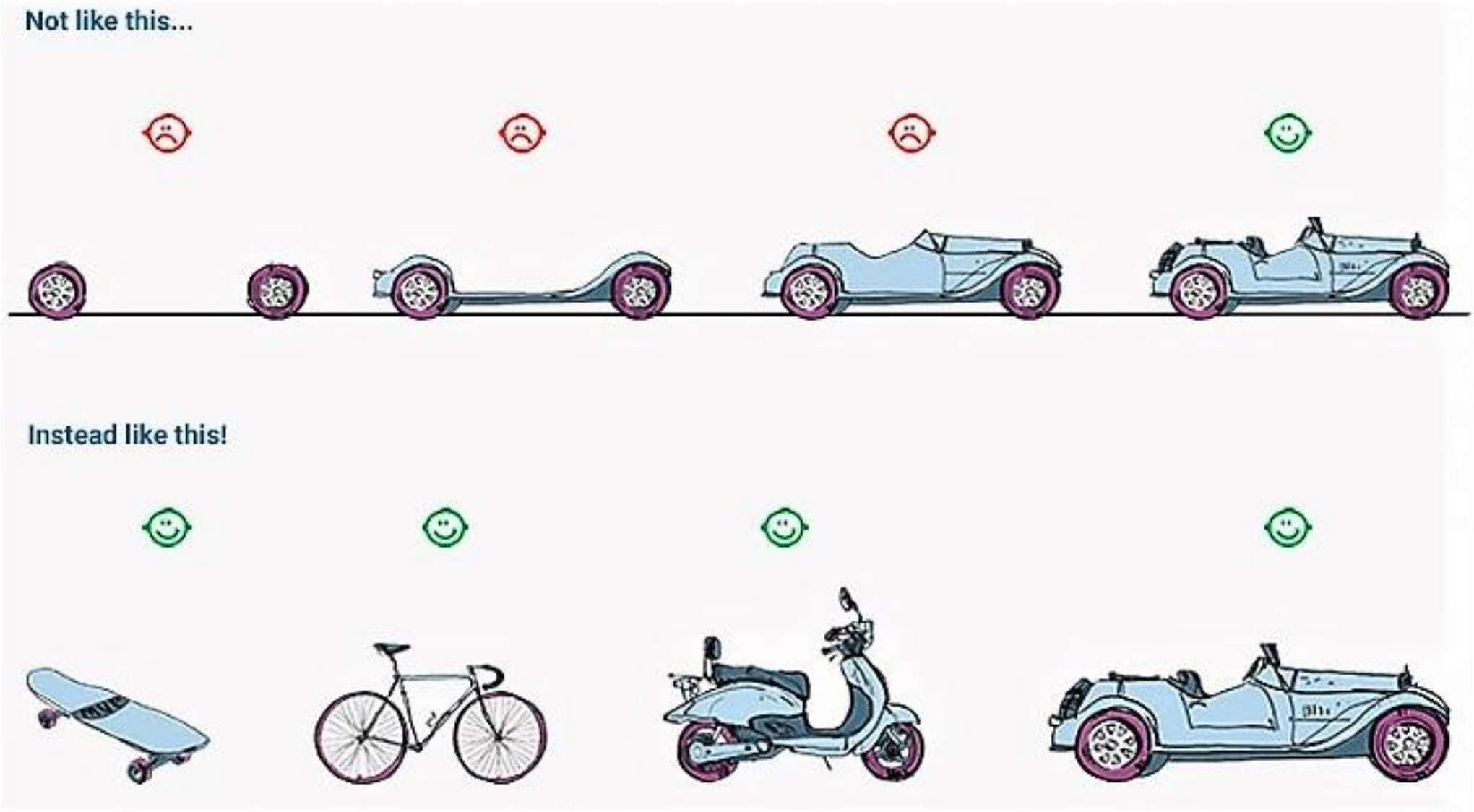
First: Think of an idea - outline
Then: Work out the idea, try out and adjust

Time

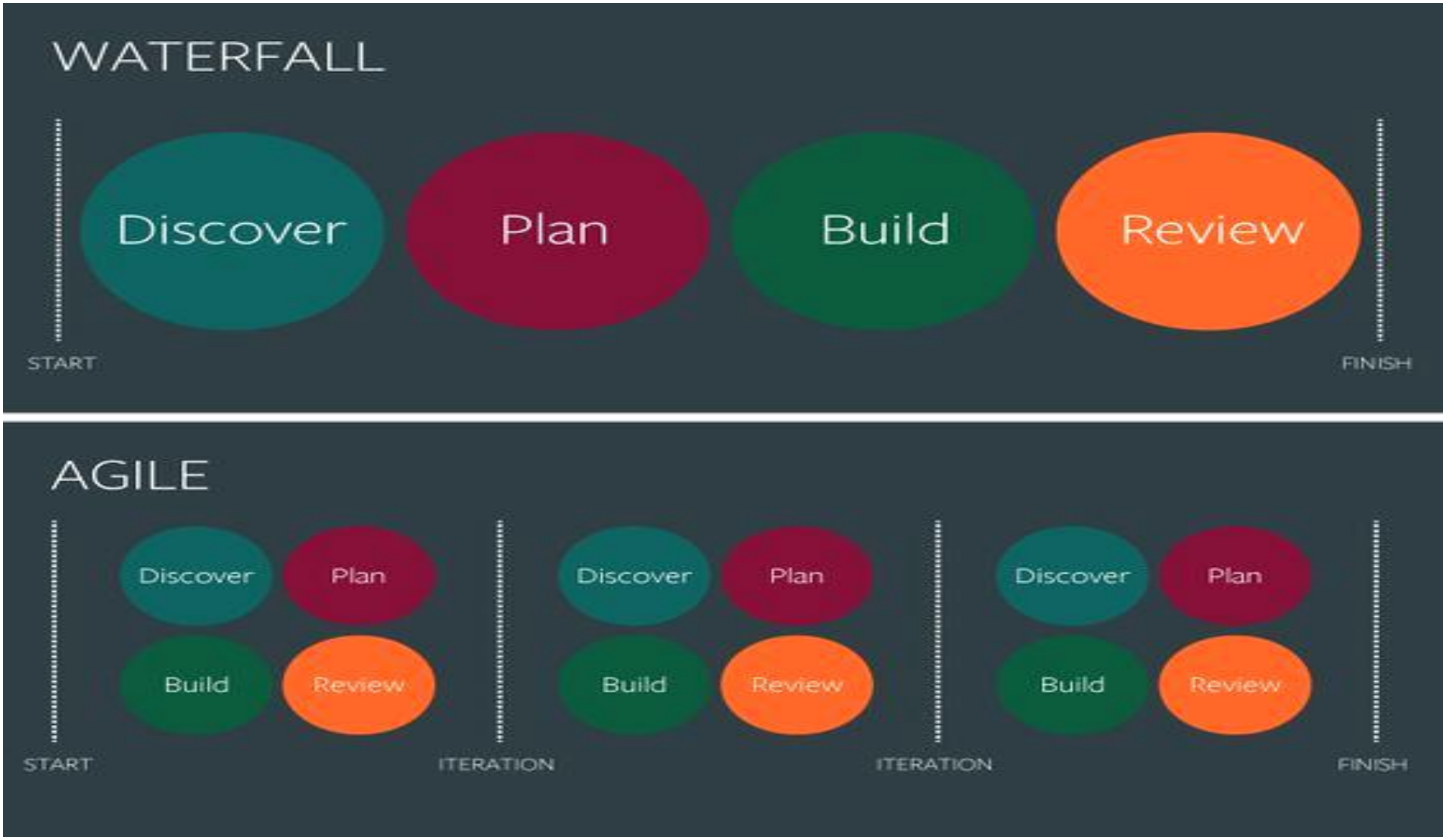
The Agile Approach



Not big bang but evolutionary

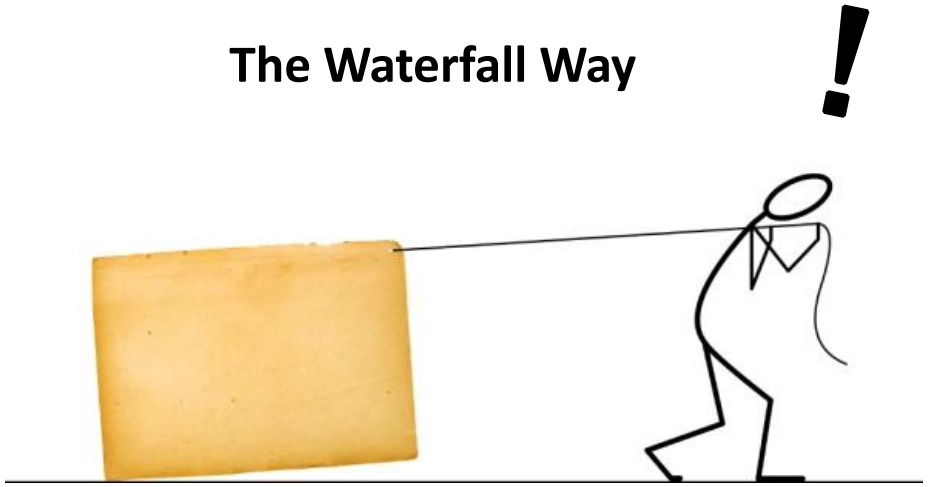


Waterfall vs. Agile



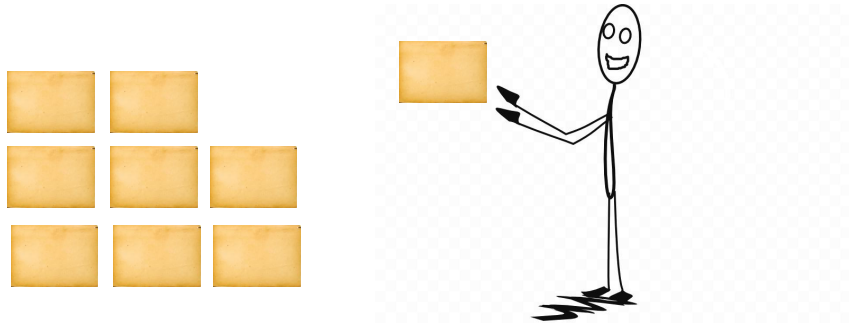
Waterfall vs. Agile

The Waterfall Way



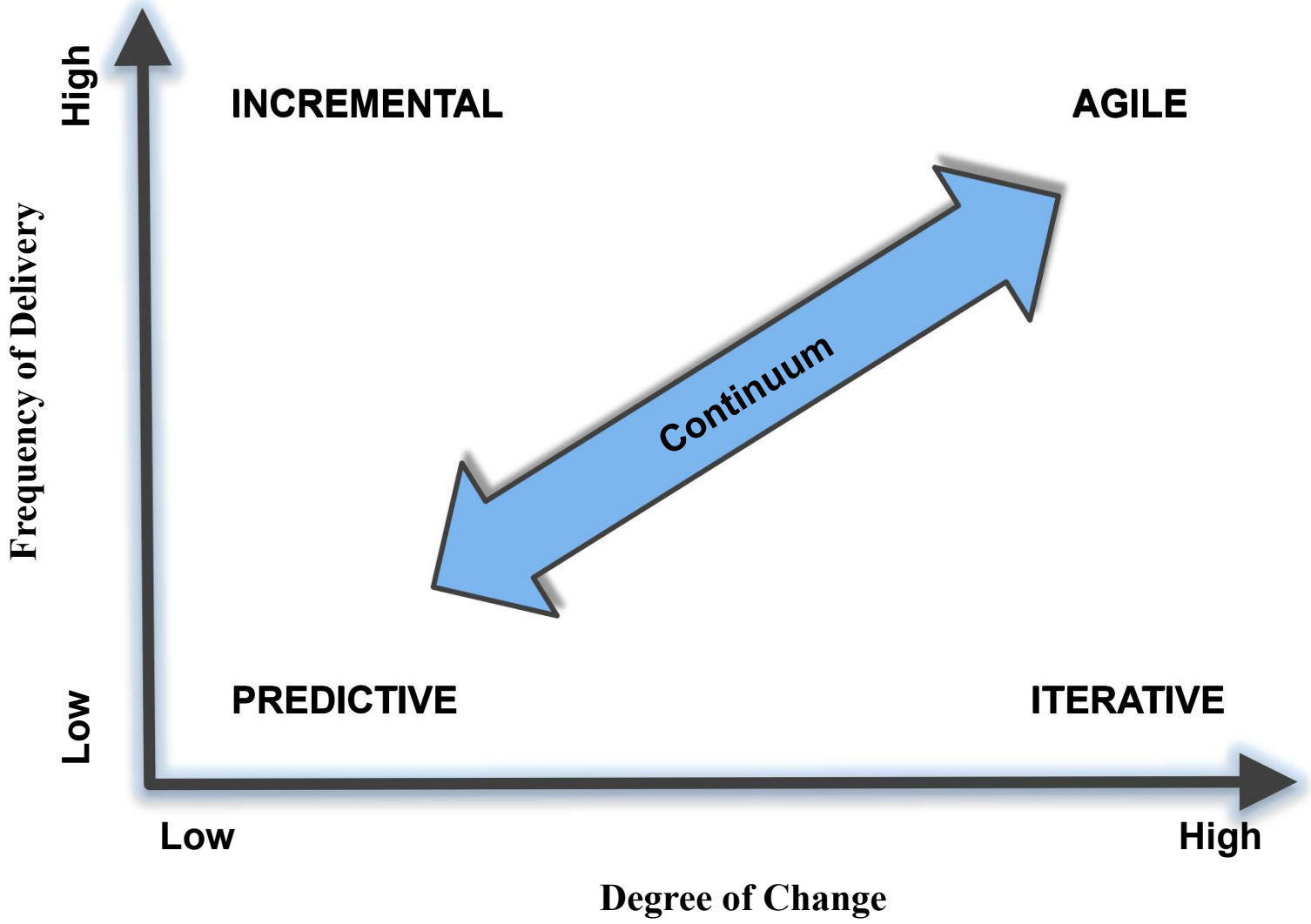
"This project has gotten so big and complicated, I'm not sure if the team can deliver it at all!"

The Agile Way



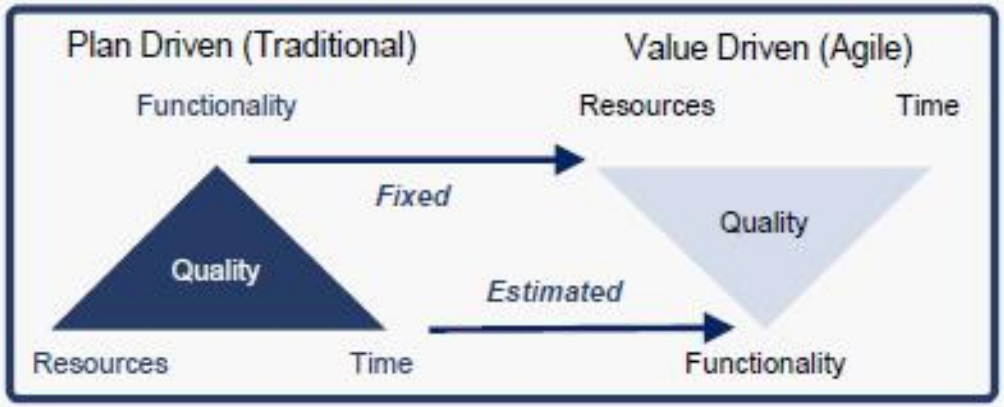
"It's so much better delivering this project in bite-sized portions regularly to my customers."

The Continuum of Life Cycles



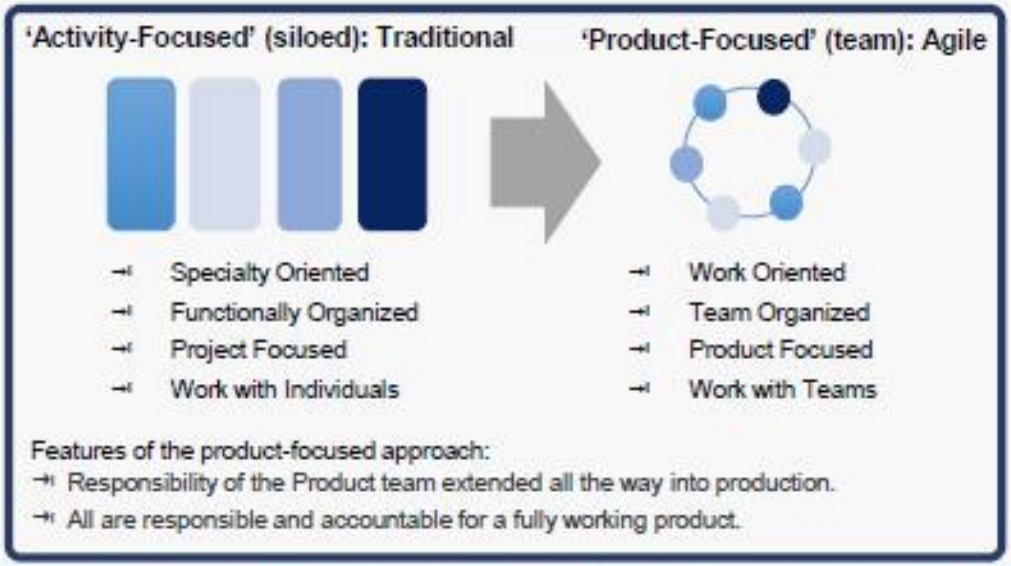
Agile: Satisfy the Customer

Agile movement provides alternative to traditional project development.



Traditional Development

- Start with a complete design
- Building is followed by testing the final product
- Finally, testing in practice
- No feedback loops
- Plan driven



Agile Development

- Every Sprint delivers working software that can be used in practice
- Starts with delivering basic functionality to which features are added
- Value driven

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

7 ELEMENTS OF AGILE CULTURE DNA

ADAPTABILITY TO CHANGE

A strong core that provides stability with flexibility to adapt and change

WELLBEING AND FULFILMENT

Happy and positive over fear based, stress, fatigue and burn-out. Providing a deep sense and feeling of achievement to individuals

COLLABORATIVE COMMUNITIES AND DISTRIBUTED AUTHORITY

A network of collaborative teams with more autonomy for decision-making as appropriate

AGILE LEADERSHIP

Supportive leadership over directive, command-and-control style

TRUST AND TRANSPARENCY

Loyalty, integrity and commitment to transparency, openness and honesty on day-to-day working

UNLEASHED PURPOSE AND MEANINGFUL RESULTS

A clear, compelling and inspiring purpose which focuses on results that matter to all stakeholders

INNOVATION, LEARNING AND PERSONAL MASTERY

Psychological safety, thoughtful experimentation, learning and reflective practice moving towards personal strengths and mastery

Types of Positions on Change

The support-resistance scale

Each person in your organization will be somewhere on the support-resistance scale, with passives sitting in the middle. If you can work out where people are on the spectrum, it will help you work out what you need to do to gain their support.

Champions

Vocal supporters that push for change and actively recruit other supporters. Key to success.

Change agents

Active and highly energetic supporters, committed to pushing the rock to the top of the hill.

Passives

Fence-sitters conserving energy. They wait until they see if it's worth the risk before they act.

Skeptics

Stubborn cynics, jaded by the failure of previous business transformations, who believe, *"It'll never work."*

Protesters

Energetic resisters that push against change; actively recruiting other protesters and skeptics.

Oath of Non-allegiance

I promise not to exclude from consideration any idea based on its source, but to consider ideas across schools and heritages in order to find the ones that best suit the current situation.

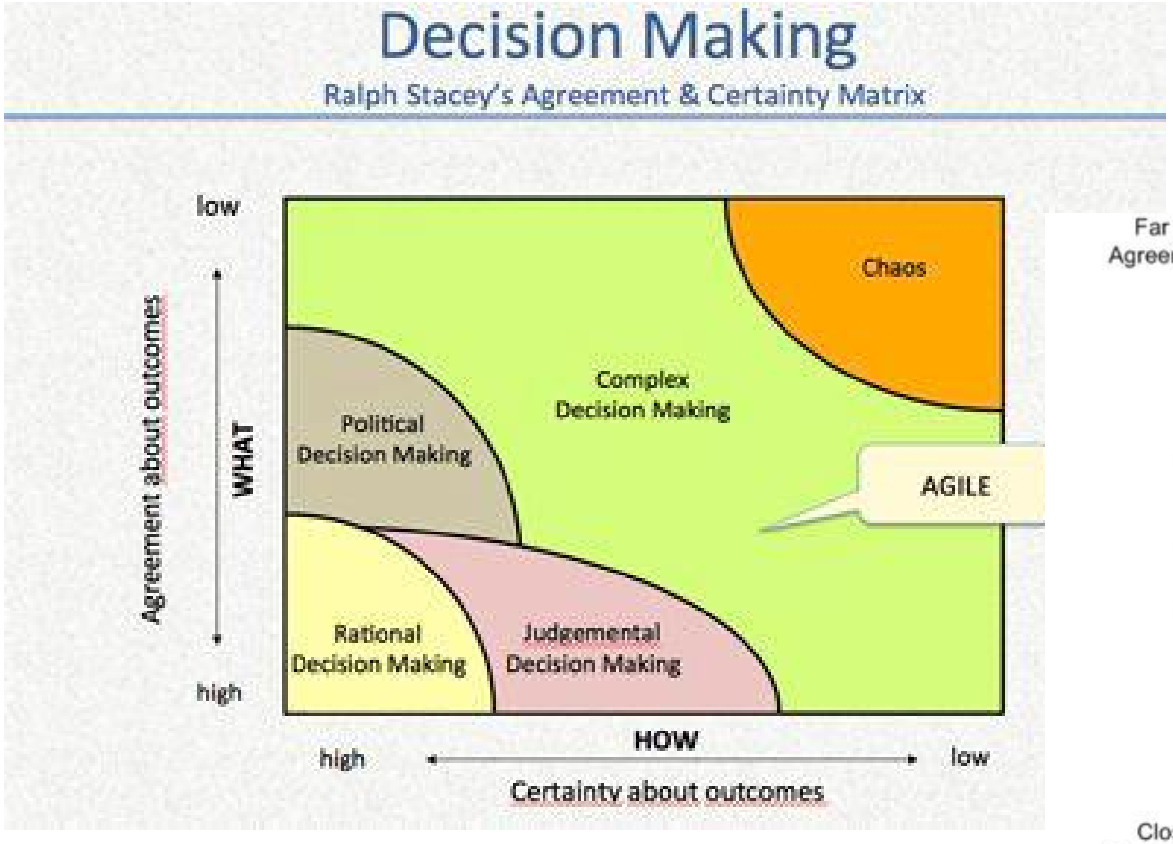
Alistair Cockburn

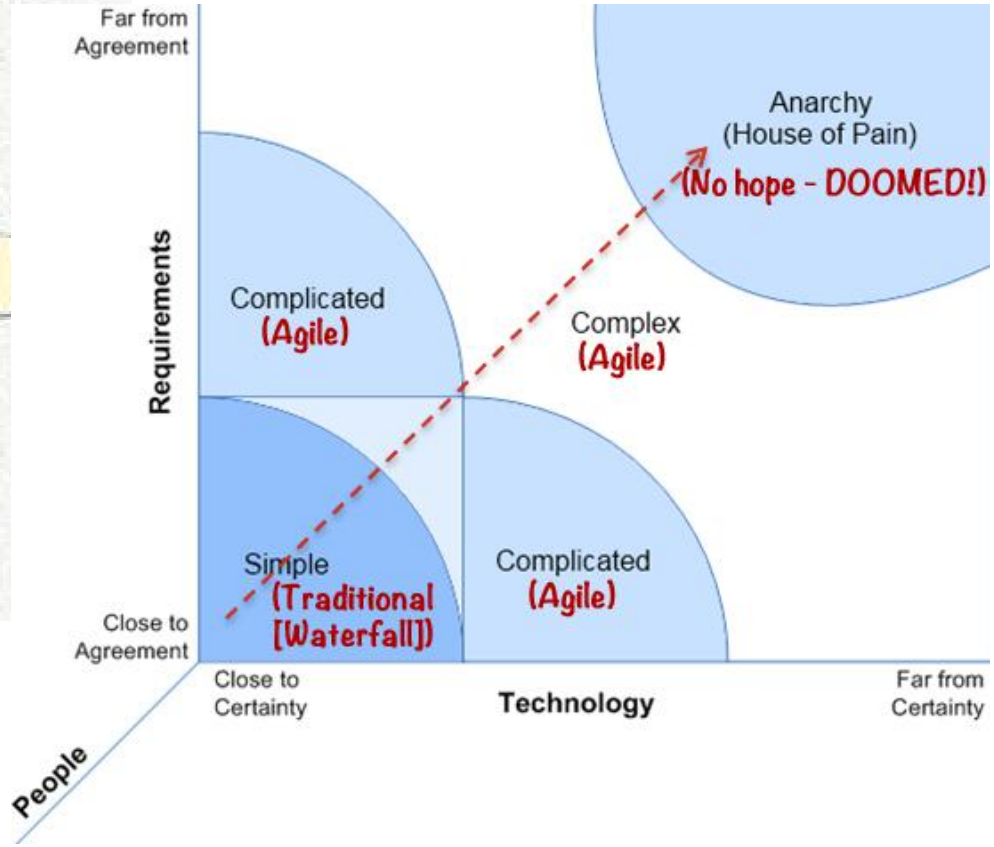
The Edge of Chaos

- “Complex systems tend to locate themselves at a place we call ‘the edge of chaos.’ We imagine the edge of chaos as a place where there is enough innovation to keep a living system vibrant, and enough stability to keep it from collapsing into anarchy. . . . Too much change is as destructive as too little. Only at the edge of chaos can complex systems flourish.”

M. Crichton [1995], pp. 2-3.

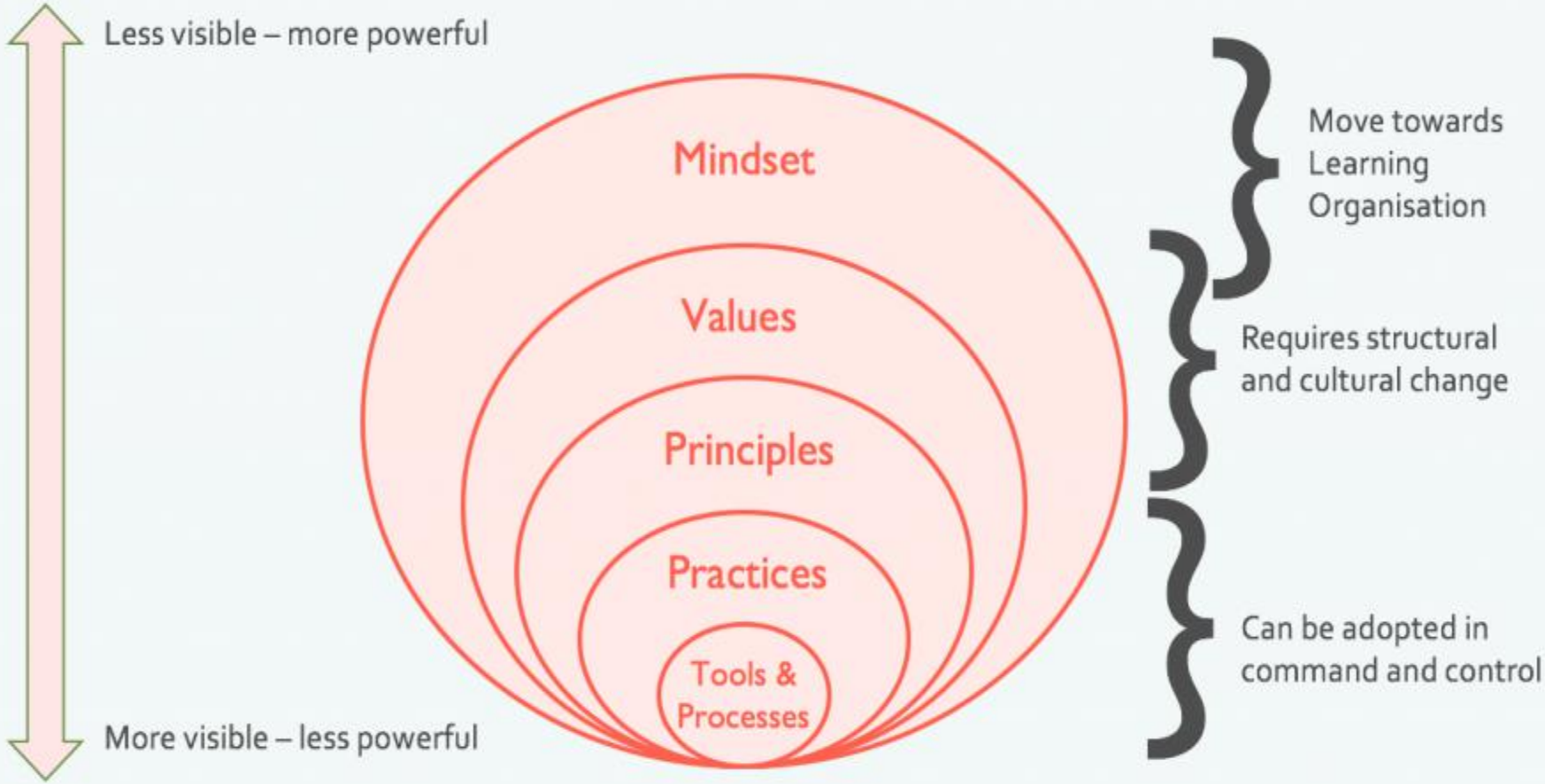
https://cognitive-edge.com/blog/cynefin-as-of-st-davids-day-2019/





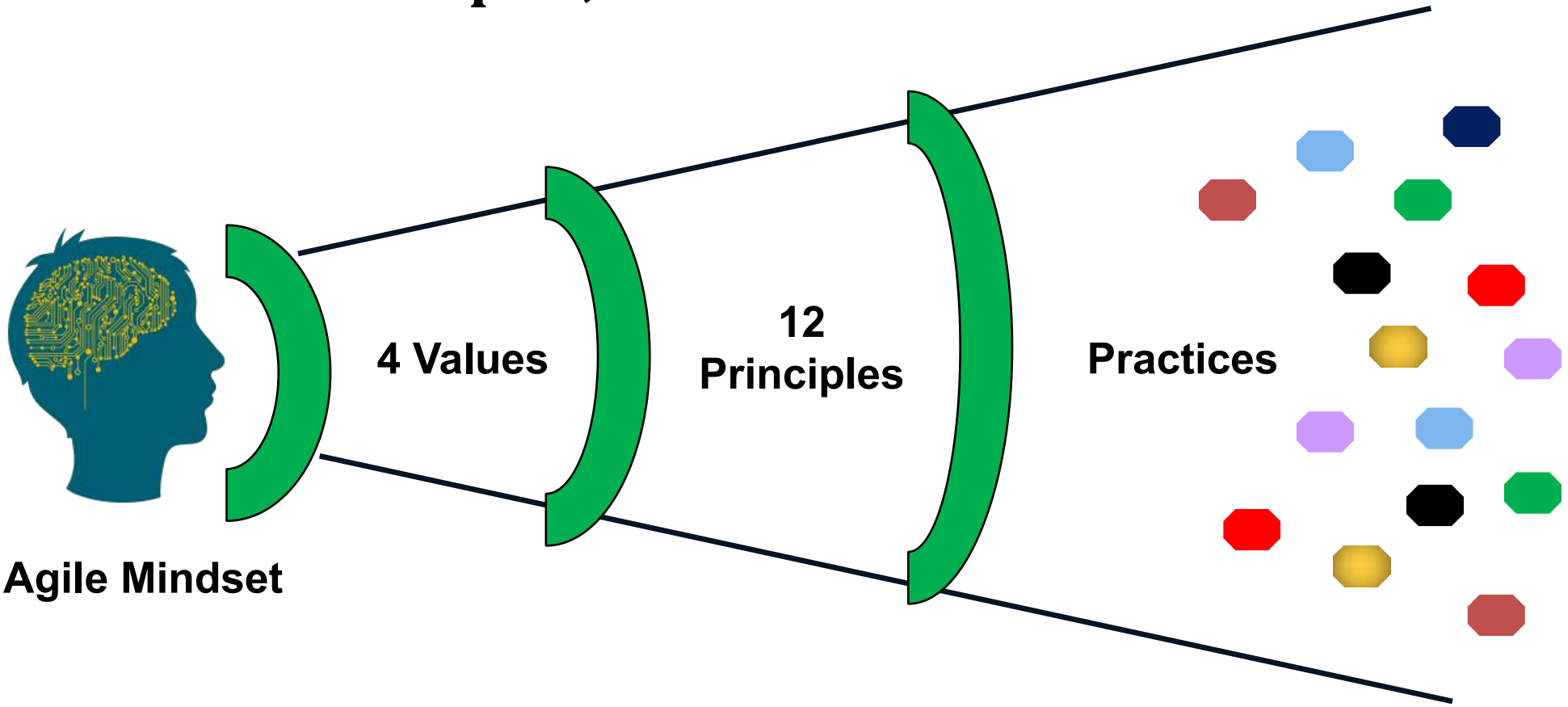
The Agile mindset is applicable in situations where there is a uncertainty and disagreement about outcomes. The challenge is to foster innovation and individual talent, encourage autonomy and intuition, and to stay away from chaos and anarchy.

What is Agile?



Simon Powers (2016) at https://www.adventureswithagile.com/2016/08/10/what-is-agile/

The Relationship between the Agile Manifesto Values, Principles, and Common Practices

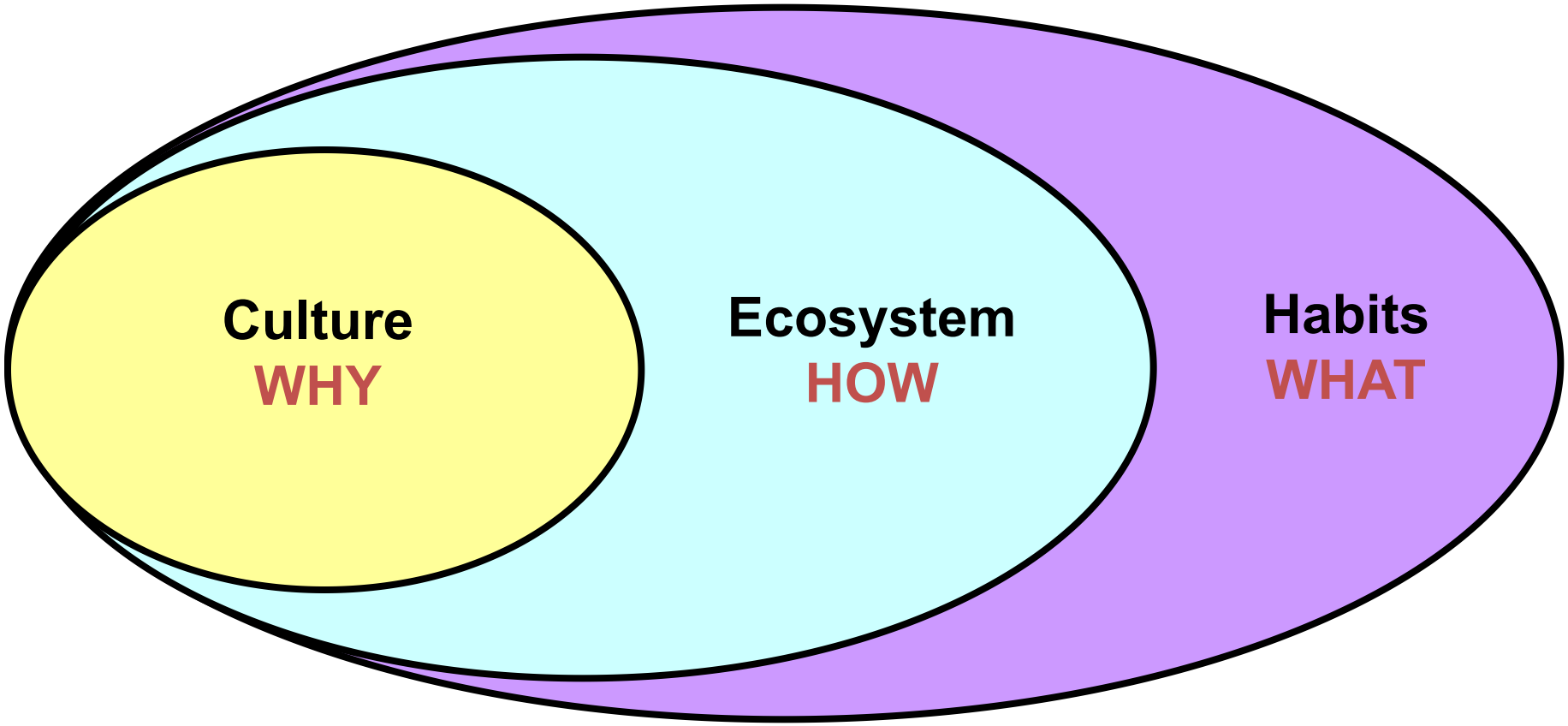


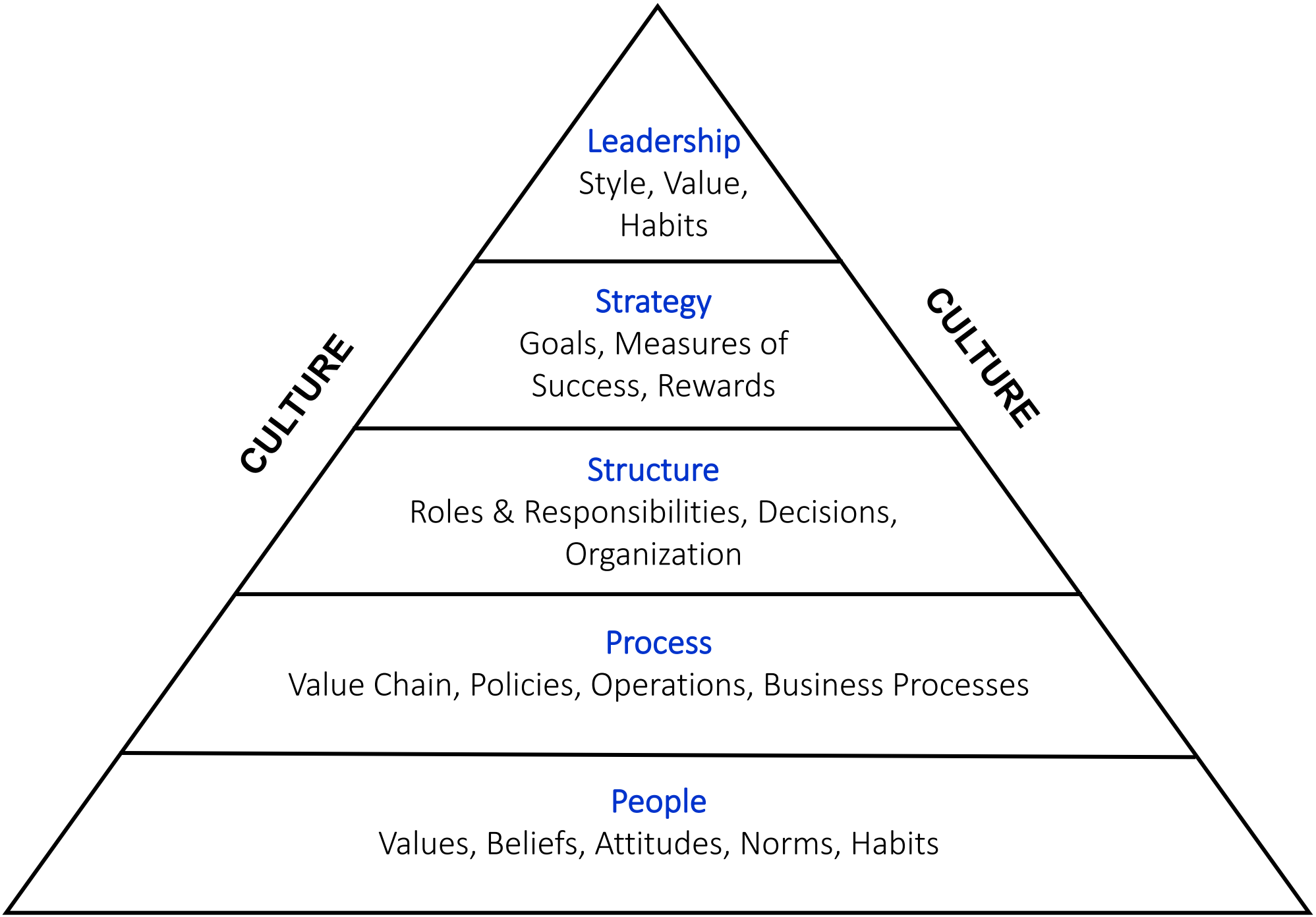
Agile is a mindset defined by values, guided by principles, and manifested through many different practices. Agile practitioners select practices based on their needs.

Agile Practice Guide (2017), PMI, p. 10

Organizational Agility

- ... is a culture which is in line with the values & principles of agile, supported by the organizational ecosystem & manifested through personal & organizational habits.




Leadership
Style, Value,
Habits
Strategy
Goals, Measures of
Success, Rewards
Structure
Roles & Responsibilities, Decisions,
Organization
Process
Value Chain, Policies, Operations, Business Processes
People
Values, Beliefs, Attitudes, Norms, Habits
CULTURE
CULTURE

FIXED VS. GROWTH MINDSET*

based on the work of Dr Carol Dweck

I believe my [intelligence, personality, character] is inherent & static, locked-down or fixed. My potential is determined at birth; it doesn't change.

FIXED MINDSET

- Avoid failure
- Desire to look smart
- Avoid challenges
- Stick to what they know
- Feedback & criticisms are personal
- Don't change or improve

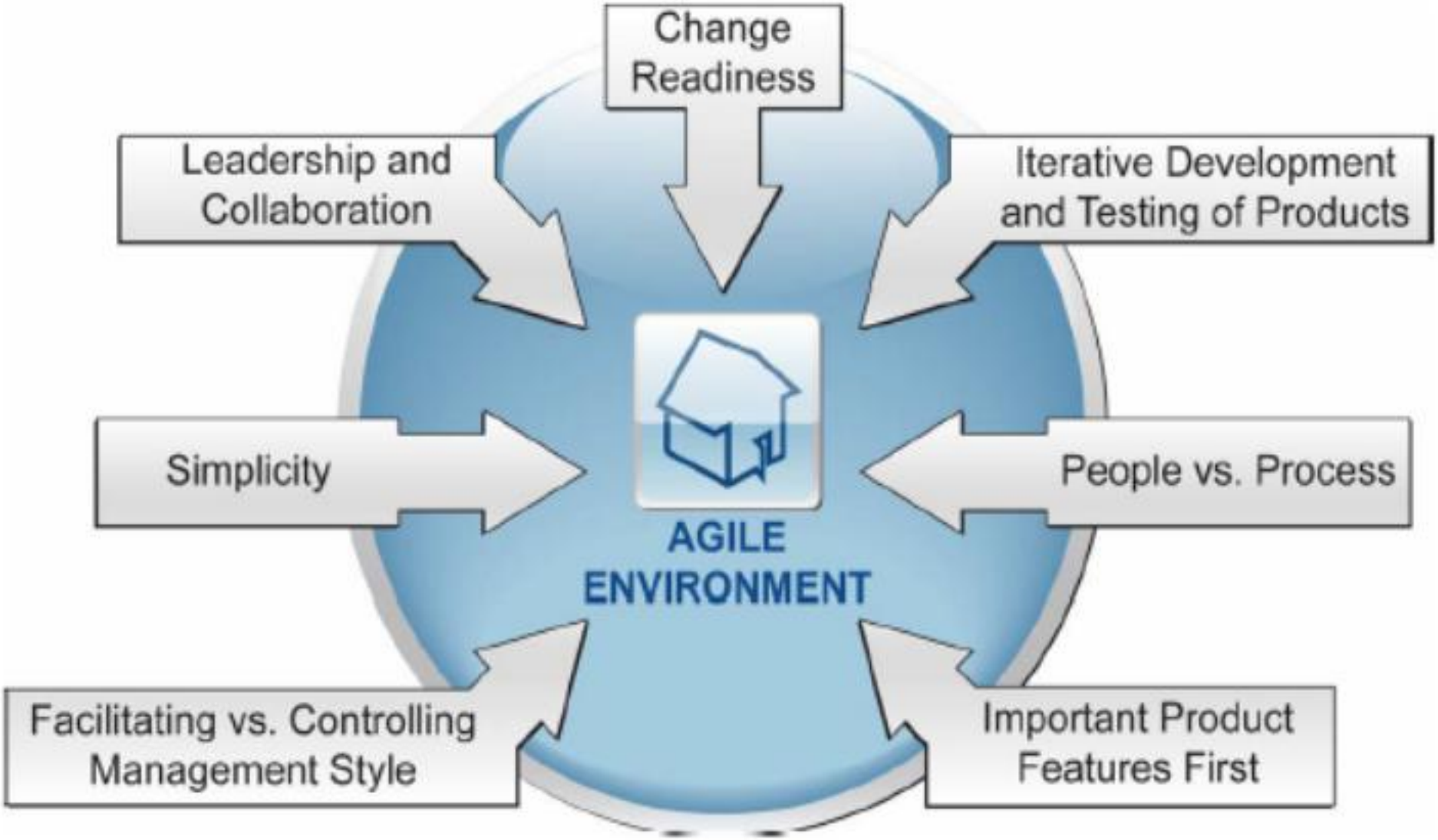
I believe my [intelligence, personality, character] can be continuously developed. My true potential is unknown & unknowable.

GROWTH MINDSET

- Desire continuous learning
- Confront uncertainties
- Embrace challenges
- Not afraid to fail
- Puts in lots of effort to learn
- Feedback is about current capabilities

*A mindset is the established set of attitudes held by someone

THE AGILE CULTURE



Agile Development Methods

- Agile software development is a group of software development approaches based on iterative and incremental development, where requirements and solutions evolve through collaboration between self-organizing, cross-functional teams.
- It promotes adaptive planning, evolutionary development and delivery, a time-boxed iterative approach, and encourages rapid and flexible response to change.
- It is a conceptual framework that promotes foreseen interactions throughout the development cycle.

Agile Myths

- Agile means “no planning”
- Agile means “no documentation”
- Agile means “no discipline or rules”
- Agile is just for the technology team
- Agile will solve all of our problems – it’s a “silver bullet”
- Agile is a quick fix
- Agile = Scrum
- Agile is easy
- Agile = cheap
- Visual management doesn’t work

Characteristics of Predictive and Agile Lifecycles

APPROACH	REQUIREMENTS	ACTIVITIES	DELIVERY	GOAL
Predictive	Fixed	Usually performed once for the entire project	Single delivery	Manage cost, scope and schedule
Agile	Dynamic	Repeated until correct or complete	Frequent small deliveries	Customer value via frequent deliveries and feedback

Agile Practice Guide (2017). PMI., p. 18

What is a team?

"A team is a **small number of people** with **complementary skills** who are committed to a **common purpose**, set of **performance goals**, and **approach** for which they hold themselves **mutually responsible**."

Katzenbach & Smith, The Wisdom of Teams, 1003

Agile Habits Critical for Realizing the Benefits of Agile

1. Put some skin in the game
   - In an agile environment, some business-unit leaders will be tapped as product owners—that is, the business-unit stakeholders most accountable for shaping the products.
   - These leaders must make the development and success of a product their highest priority—and they must be given the leeway to do so.
2. Shape the product together
   - Agile product development is less about taking orders and more about sharing information.
   - The business and the IT organization must codevelop products every day, side by side, in an ongoing process.

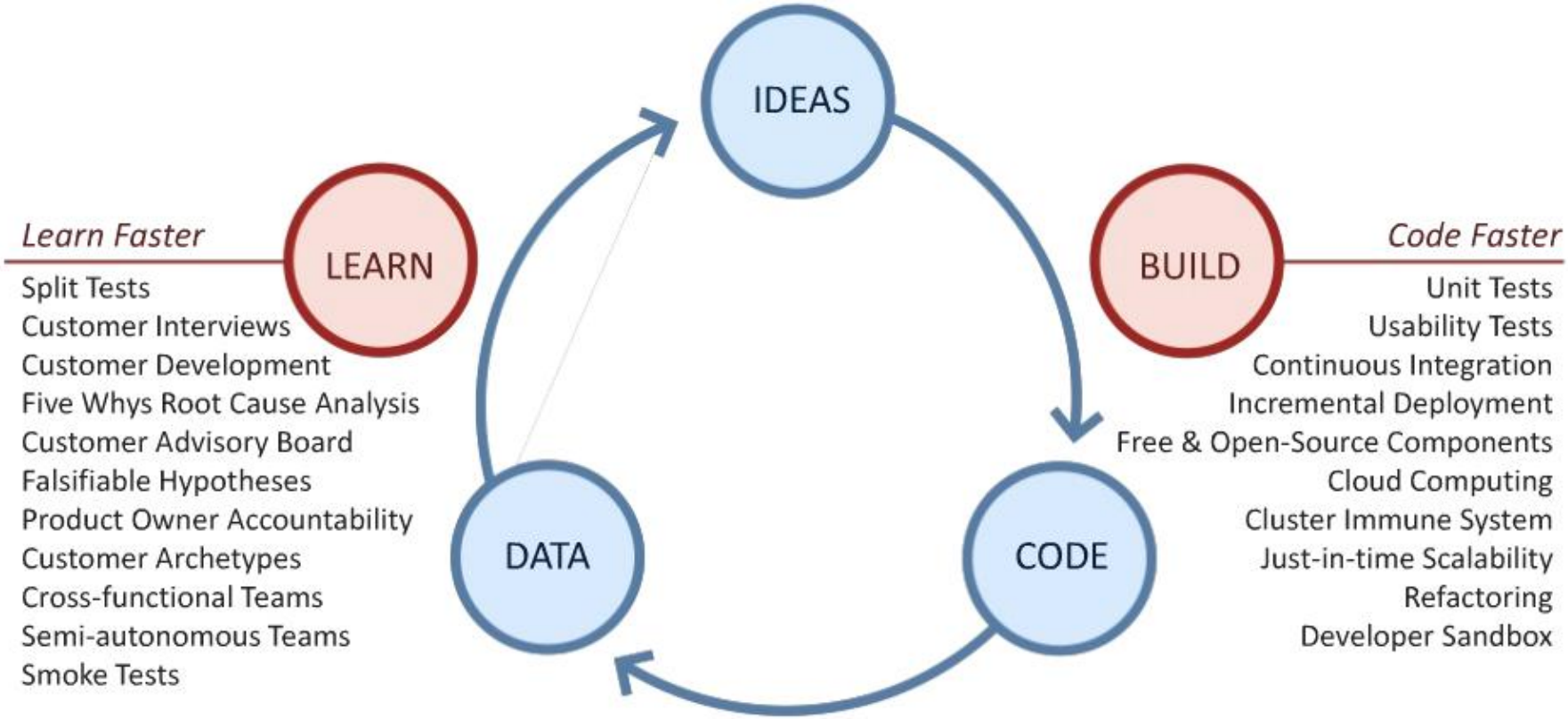
Agile Habits Critical for Realizing the Benefits of Agile

3. Cheer for your own team
   - Leaders in the C-suite and the heads of business units have a critical role to play as evangelists for the software products they codevelop: they must hold product owners from the business units accountable for the successful rollout of any new release and its effect on the business.
4. Think like a user
   - To help build software and products that transform the way a company operates or appeal to customers, product owners from the business must be unwaveringly committed to users' needs.

Agile Habits Critical for Realizing the Benefits of Agile

5. Learn to live with ‘good enough’
   - Agile development emphasizes a test-and-learn approach—for instance, releasing a minimally viable product that delivers value to end users in the short term but is expected to change on the fly.
   - In this case, chief information officers may need to help senior business executives come to terms with the release of a good-enough product by redefining their expectations and thresholds for risk.
6. Broaden the mandate
   - To maximize the impact of agile methods, senior leadership must consider ways to transfer lessons from agile teams to different areas of the company.

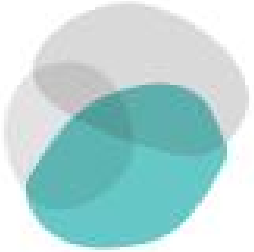
The Lean Startup



Human-Centered Design

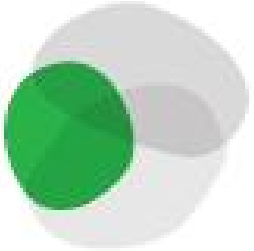
INSPIRATION

In this phase, you'll learn how to better understand people. You'll observe their lives, hear their hopes and desires, and get smart on your challenge.



IDEATION

Here you'll make sense of everything that you've heard, generate tons of ideas, identify opportunities for design, and test and refine your solutions.



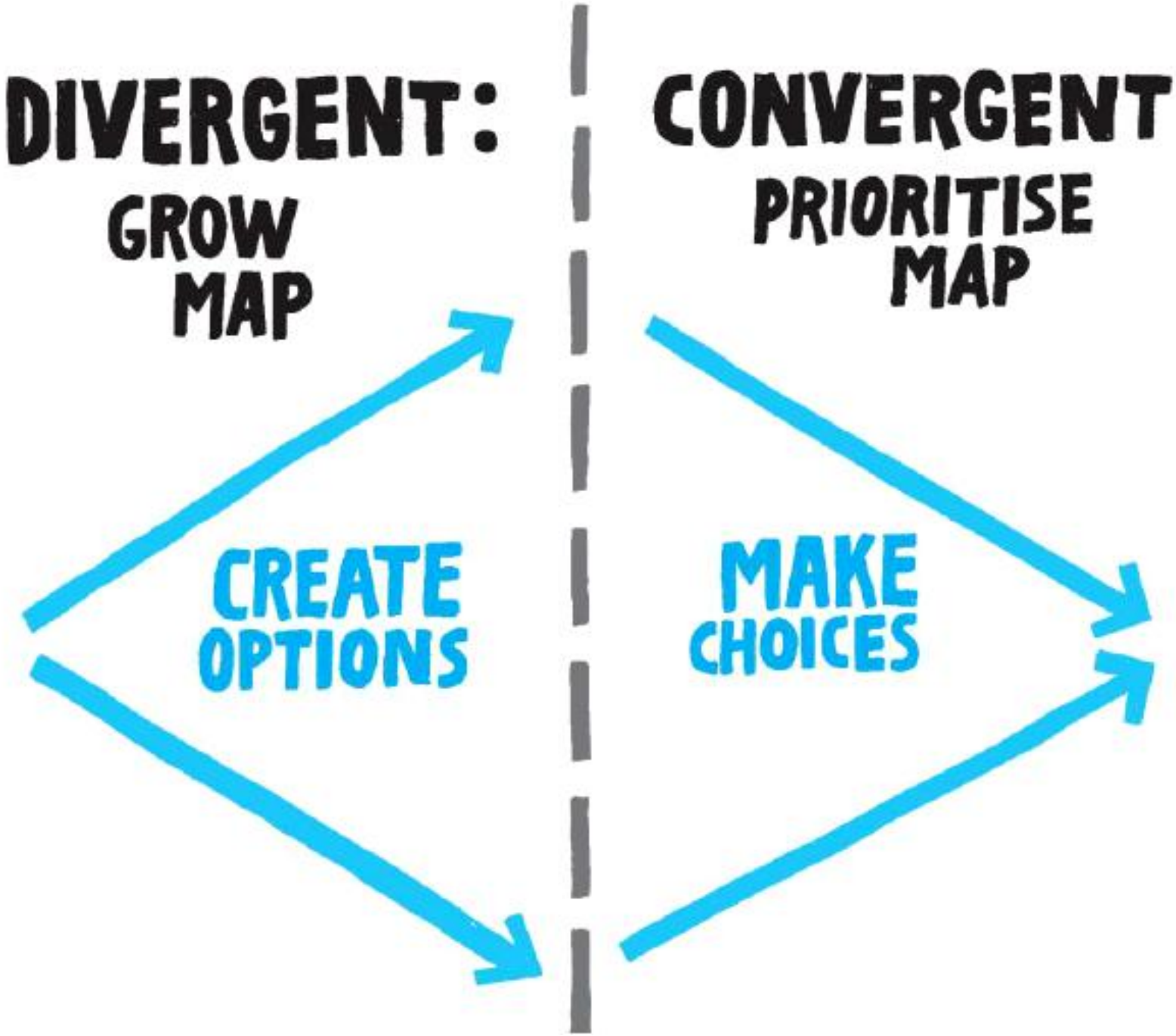
IMPLEMENTATION

Now is your chance to bring your solution to life. You'll figure out how to get your idea to market and how to maximize its impact in the world.

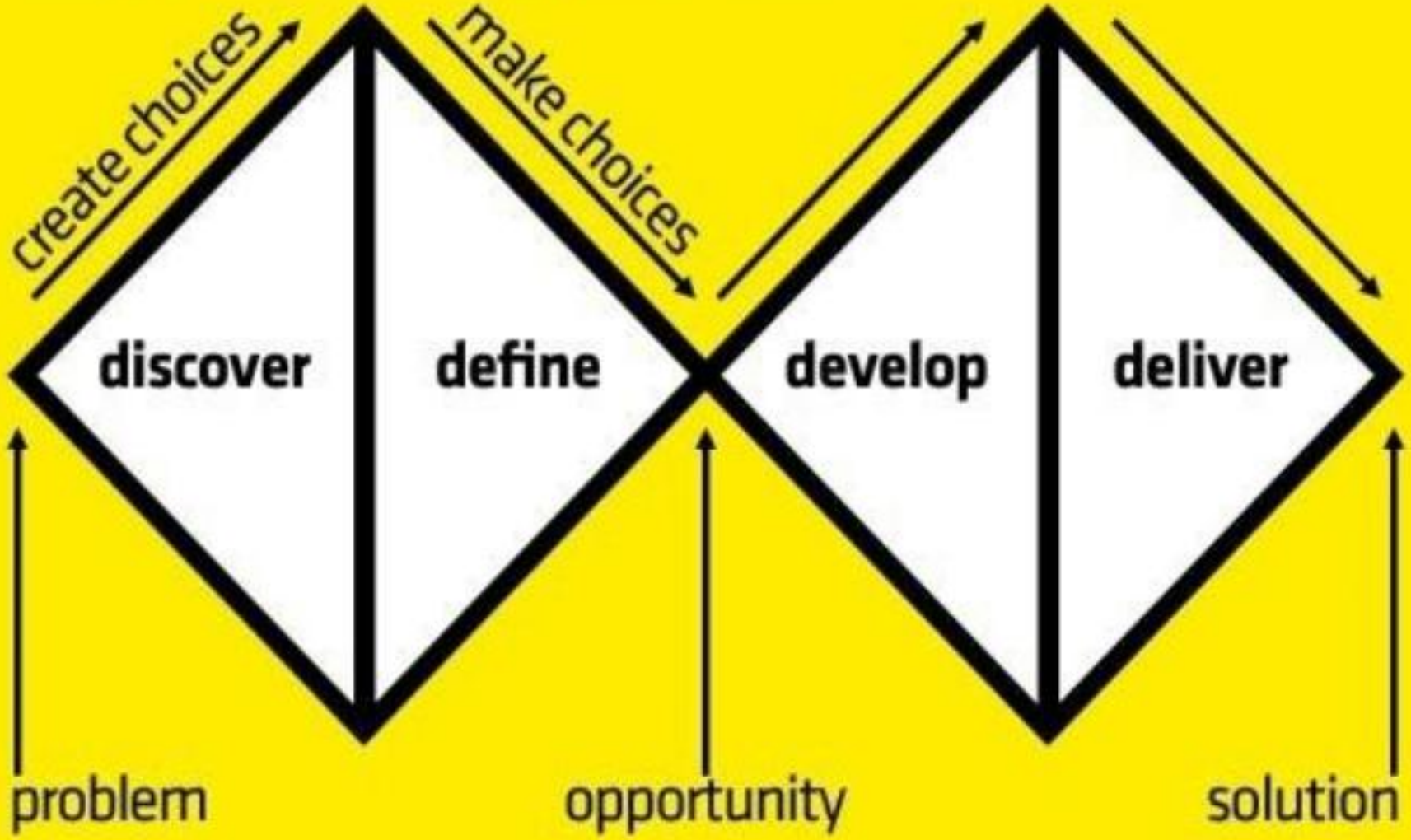
The Field Guide to Human-Centered Design by IDEO.org. (2015), p. 11

Human-Centered Design

- Consists of three phases
  - In the ***Inspiration*** Phase you'll learn directly from the people you're designing for as you immerse yourself in their lives and come to deeply understand their needs.
  - In the ***Ideation*** Phase you'll make sense of what you learned, identify opportunities for design, and prototype possible solutions.
  - And in the ***Implementation*** Phase you'll bring your solution to life, and eventually, to market. And you'll know that your solution will be a success because you've kept the very people you're looking to serve at the heart of the process

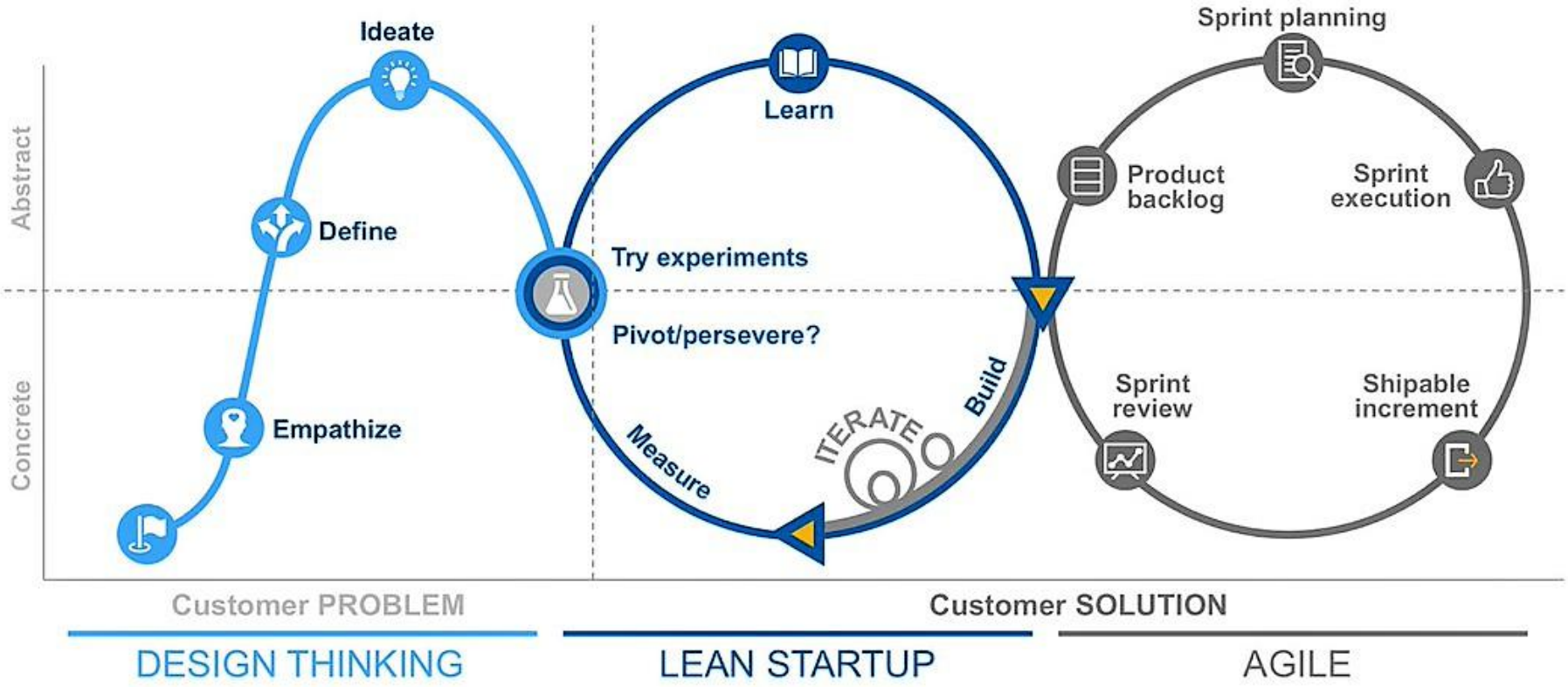

DIVERGENT:
GROW
MAP
CONVERGENT:
PRIORITISE
MAP
CREATE
OPTIONS
MAKE
CHOICES

design thinking process



Recruiting Social

Combine Design Thinking, Lean Startup and Agile



Gartner

Critical Success Factors for Change Management

- Stakeholder Collaboration, Empowerment, and Engagement
- Allocate Time for Acceptance of the Change
- Ensure System Alignment with the Change Initative
- Provide Focus for the Change Initiative
- Identify, Select, and Develop Talent Based on Change Management Competencies
- Formalize Philiosophy and Policy of Change Management
- Develop and Deploy Change Management Measurement Processes and Tools

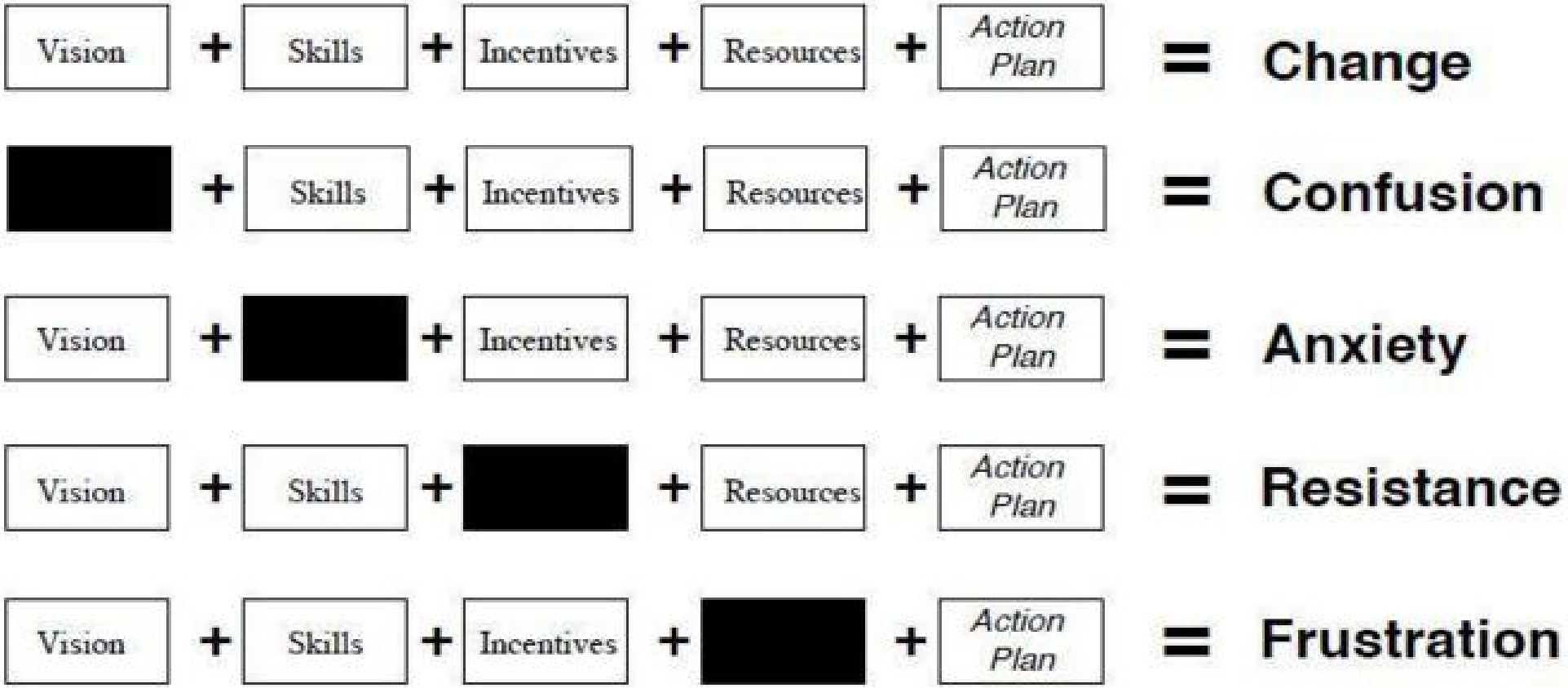
Potential Barriers and Change Derailers

- Lack of Strong, Committed Sponsorship
  - lack of a change sponsor
  - lack of commitment to funding and/or resources
- Cultural Resistance to Change
  - *inertia* due to a false sense of threatened sacifices
  - lack of *trust*
  - lacking the *competencies* to change
  - *bureaucratic* decision processes
- Failure to Build Change Readiness
  - lack of recognition for the need to continuously change
  - lack of knowledge/learning in a change process

Potential Barriers and Change Derailers

- Insufficient Time Allocated to Change
  - lack of time
  - poor follow through
- Poor Vision of the Future
- Poor Access to Technology by All Stakeholders
- Poor Measures and/or Measurement Process
  - lack of appropriate performance metrics

Managing Complex Change



This chart demonstrates that all 5 factors: Vision, Skills, Incentives, Resources and an Action Plan must be present for real CHANGE to take place. If any of these factors is missing an alternative outcome such as confusion, anxiety, gradual change, frustration or false starts will occur. (Ambrose, 1987)

Kotter's 8 Step Model of Change

ADKAR: A Goal Oriented Change Management Model

Awareness of the business reasons for change. Awareness is a goal/outcome of early communications related to an organizational change

Desire to engage and participate in the change. Desire is the goal/outcome of sponsorship and resistance management

Knowledge about how to change. Knowledge is the goal/outcome of training and coaching

Ability to realize or implement the change at the required performance level. Ability is the goal/outcome of additional coaching, practice and time

Reinforcement to ensure change sticks. Reinforcement is the goal/outcome of adoption measurement, corrective actions and recognition of successful change

Shu-Ha-Ri

Shu-Ha-Ri is a way of thinking about how you learn a skill or technique.

Shu: Following Stage

守

- Look for one procedure that works
- Measure success by (a) whether the procedure works and (b) how well they can carry out the procedure

Ha: Detaching Stage

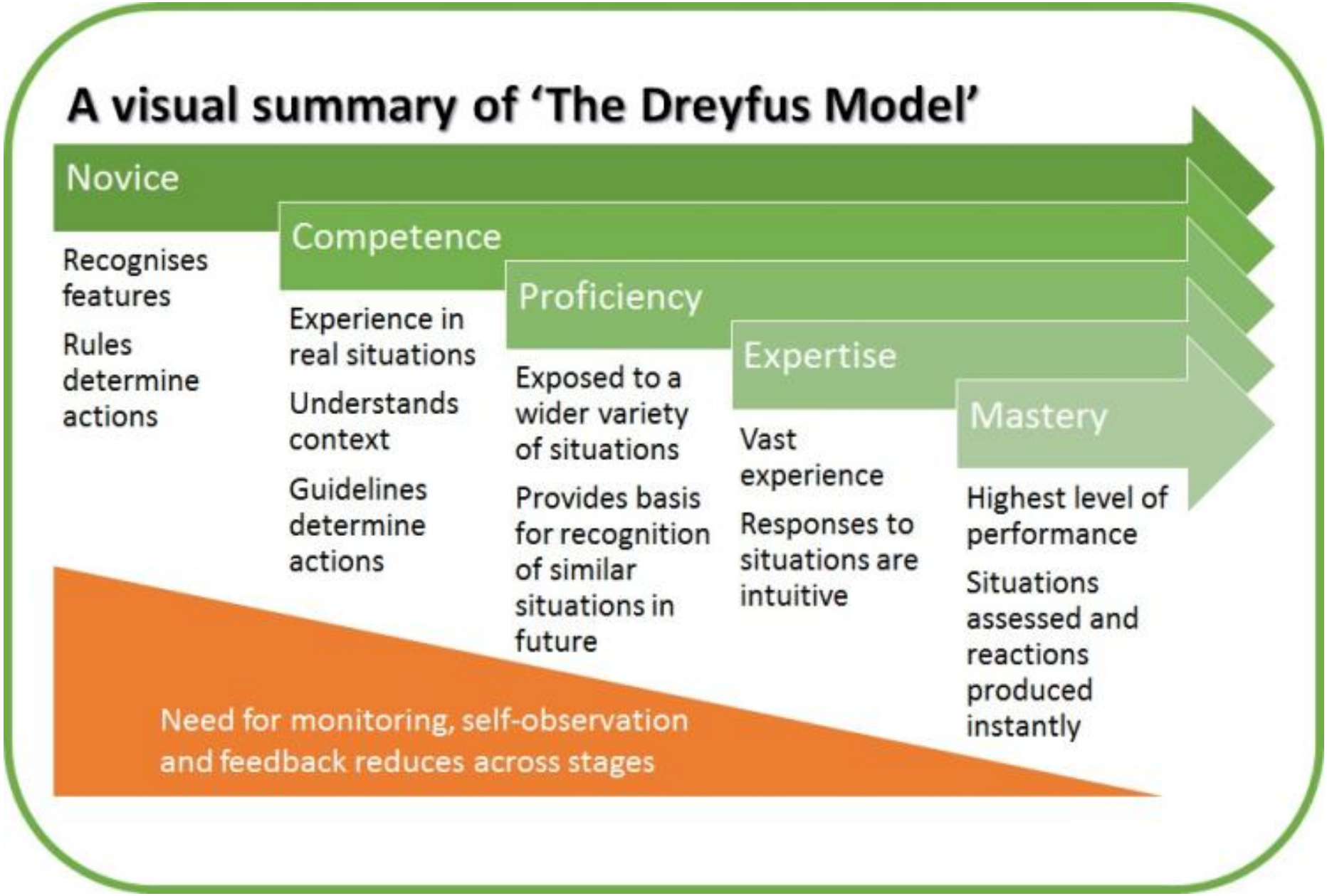
破

- Locate the limitations of the single procedure & look for rules about when the procedure breaks down
- Learns to adapt the procedure to varying circumstances

Ri: Fluent Stage

離

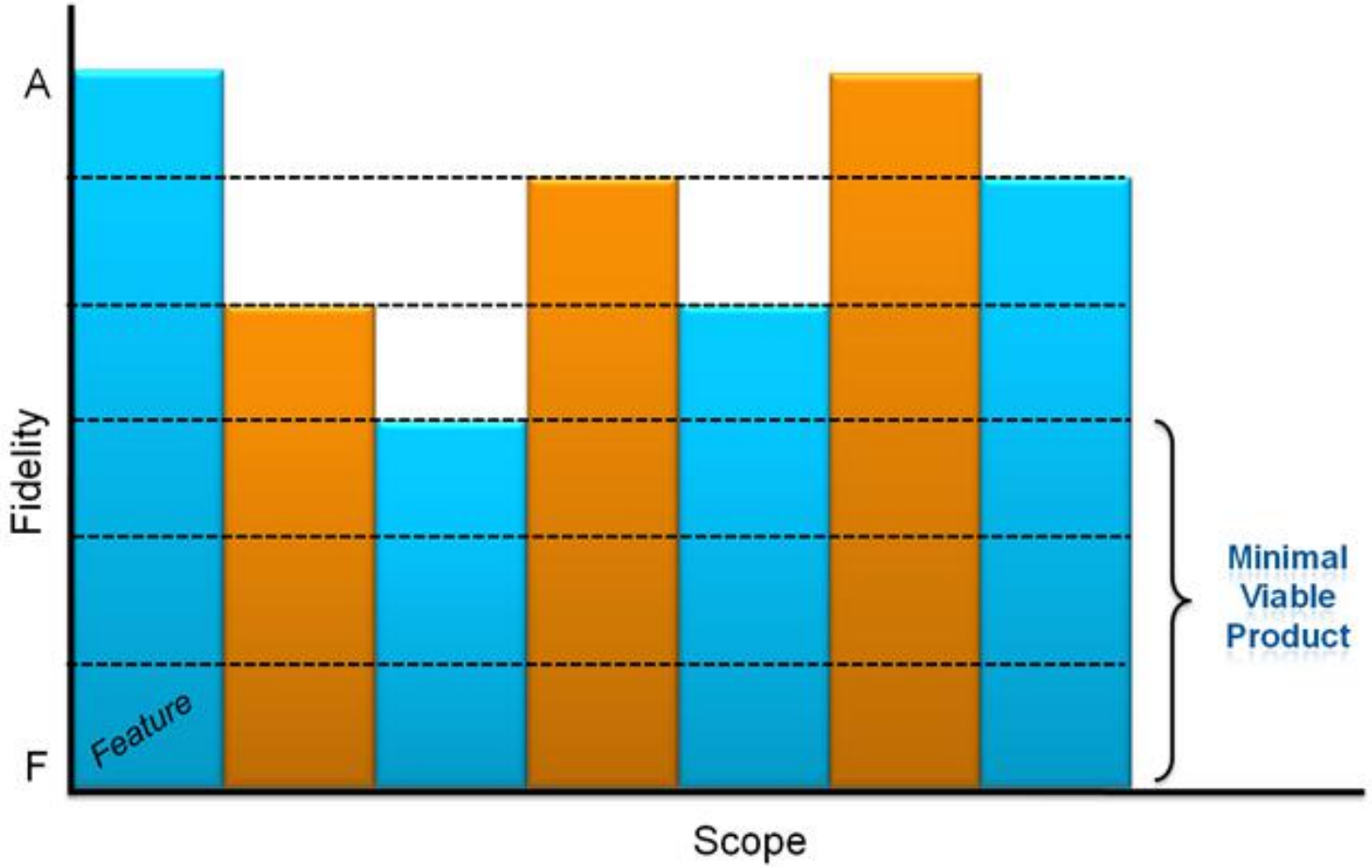
- Techniques become irrelevant; knowledge has become integrated throughout a thousand thoughts & actions
- Understands the desired end effect & simply makes his/her way to that end



https://richrtesting.com/the-dreyfus-model-a-visual-summary/

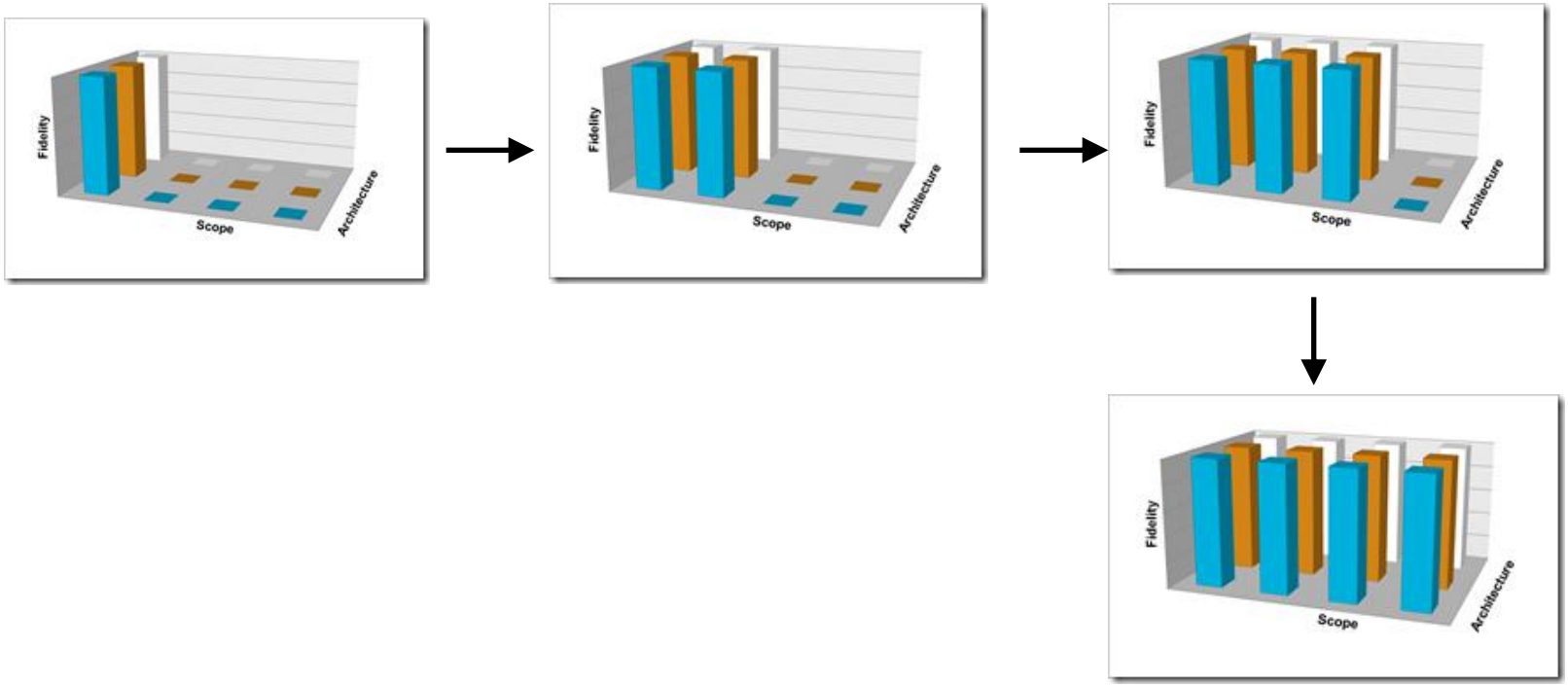
Fidelity of Features

Fidelity refers to the finesse of the feature, or solution. A low fidelity solution will be low in things like precision, granularity, or usability, but will still solve the original problem. As fidelity increases, so does the precision, granularity, usability etc.



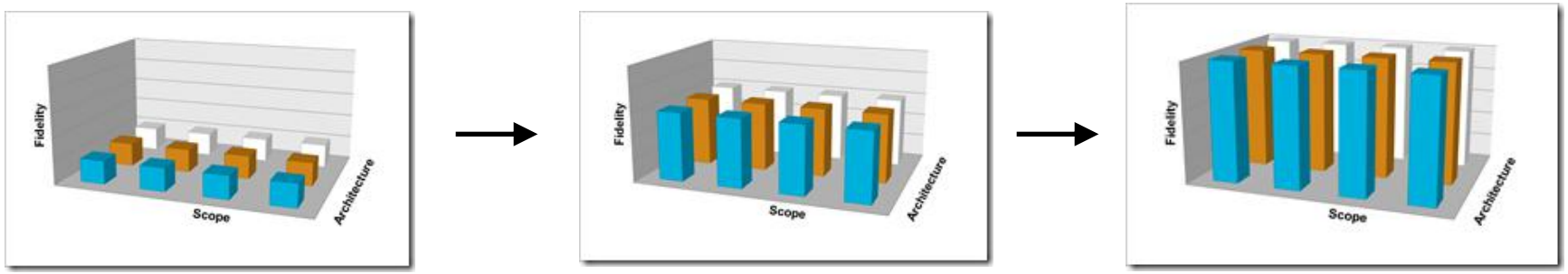
Incremental

- The purely incremental approach builds each feature, across all components, to full fidelity, one by one.



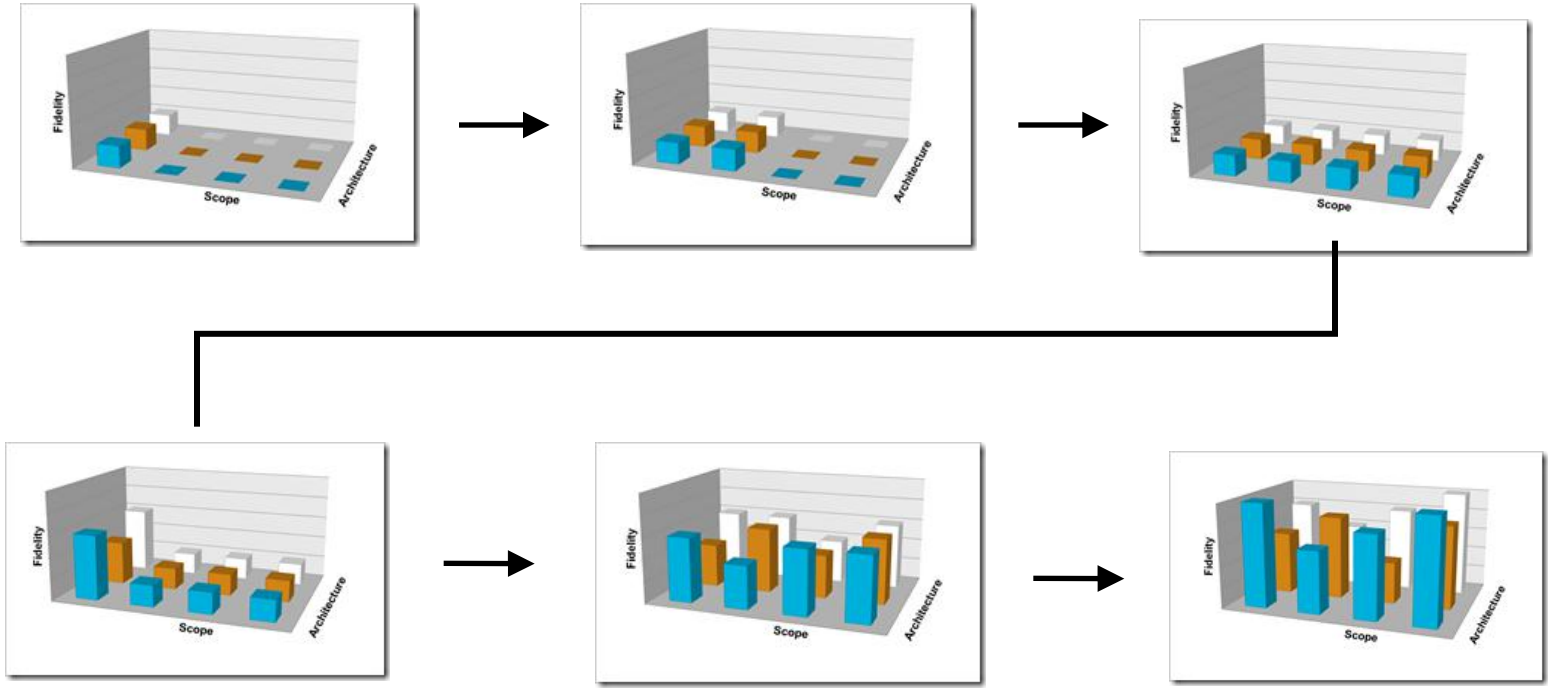
Iterative

- The purely iterative approach builds all the features, across all components, to the lowest fidelity, and then increases the fidelity to the highest level.



Agile

- An Agile approach combines the incremental and iterative approach by building each feature, one by one, at a low fidelity, and then both gradually adding features and increasing their fidelity until the right combination is achieved. Full fidelity is not always necessary. Fidelity provides a mechanism to effectively vary scope, while still meeting all the customers needs and solving their problems.



When to Use Agile?

- Agile methods are best suited for when:
  - Requirements are ambiguous & need continuous collaboration with customer
  - Project is complex and/or high risk & needs a high level of visibility
  - Teams are able to work together cross-functionally & co-located as much as feasible
  - Company culture is open to new ideas, new approaches, and change

Process Tailoring

Team should ask itself four key questions:

1. What practices should be required?
2. What other practices will we need?
3. What modifications (if any) do we need to make to these practices?
4. What level of formality should be used for documentation, approval, changes, etc?

Some Examples of Working Agreement Guidelines are:

- Show respect. Don't interrupt; let people finish what they're saying. It's OK to disagree with each other. No personal attacks, attack issues instead. We debate the merit of ideas, not people.
- Contribution. Everyone has equal voice and valuable contribution.
- Meeting. Be on time, end on time, have an agenda
- Be transparent. No hidden agendas. We will give feedback, we will receive feedback, and we will act on feedback.
- Impediments. Solve roadblocks within the team. If the impediment cant be solved within the team, give it to the Iteration Manager.
- We make commitments as a team. We will be held accountable to our commitments. – we work as a team to make a commitment and deliver on it.
- Incomplete stories are not good – it is better to help get an existing story to "done" than to start another story that can't be finished in the current sprint.

A sign seen at an automotive repair shop...

"We can do GOOD, QUICK and CHEAP work.

You can have any two but not all three.

1. GOOD QUICK work won't be CHEAP.
2. GOOD CHEAP work won't be QUICK.
3. QUICK CHEAP work won't be GOOD."

12 Failure Modes of Agile

1. Checkbook commitment doesn't support organizational change management. CEOs create within the company their own personal family dysfunction.
2. Culture doesn't support change. Reward plan, and a static and prescriptive standard of work. Try to keep cross-organizational uniformity and use PMO as enforcers.
3. Do not have retrospectives, or they are bad. Actions which come out get ignored or written off.

12 Failure Modes of Agile

4. In a race to finish features, the infrastructure gets worse and architecture becomes unstable. Distributed teams make this worse.
5. Lack of collaboration in planning. Needs to involve the entire team for release planning.
6. None or too many Product Owners. Both cases look the same. Agile is yet another hat to wear and the person is already too busy. They check out and ask the team to just do Agile. Can’t get past the ‘this sucks’ phase of adoption if the business is not bought in.

12 Failure Modes of Agile

7. Bad Scrum Master which uses a command and control style with the team to look faster, yet in reality slows things down. Low morale lowers IQ. Take decisions away and it actually makes people stupider!
8. No on-site evangelist. If the teams are distributed, need one at every site. Can’t reap the benefits of Agile or offshore without an on-site coach at each location.
9. No solid team. Empowered teams amplify learning.

12 Failure Modes of Agile

10. Tsunami of technical debt if don't pull tests forward.
11. Traditional performance appraisals. Individual heroics rewarded, glad you're not a team player!
12. Revert to traditional. Change is *hard*. Hit the threshold where this sucks. Revert back to old ways of doing business.

Jean Tabaka, keynote speech at Agile Australia 2009

12 Familiar Failure Patterns of Agile Transitions

1. The lack of a vision
2. Is ‘agile’ is a fad or a trend?
3. Projects, budgets, and stage-gates: my budget, my feature
4. The lack of a fail-friendly culture
5. The efficiency and utilization focus
6. No rapid build-test-learn culture
7. The ‘what is in for me’ syndrome
8. Taylorism and the resulting micromanagement
9. Misaligned incentives
10. Outdated technology stacks
11. Team building issues
12. Inadequate facilities

Source: https://www.youtube.com/watch?v=-RXQLVhIuJg&t=6s

Success Factors when Transforming to Agile

- Develop a strategy and roadmap for the transformation upfront. Developing a solid strategy that takes into account YOUR drivers for the transformation, what problems are you trying to fix or improvements are you seeking?
  - It should also consider both Process and People transformation needs.
  - The strategy results in a roadmap that guides the transformation execution led by the Agile Champion Team.

Success Factors when Transforming to Agile

- Need strong executive leadership commitment and active engagement.
  - The most successful transformations are the ones led by key C-Level executives and forming an Agile Champions Team comprising of cross-functional leaders.
  - The successful leaders rolled up their sleeves and are very engaged in the planning, review and tweaking of the roadmap.
  - They play an active part in communicating the message around the vision for the transformation to everyone in the company through various media.

Success Factors when Transforming to Agile

- Show measurable results early; getting excitement and positive energy.
  - This part is a must have for a successful transformation.
  - That's why it is important to start with a pilot.
  - Showing some results early, addressing concerns people will have about their roles and existing processes, and getting the business customers excited is instrumental to success.

Success Factors when Transforming to Agile

- Engage business partners and gain buy-in early.
  - Educate the business customers (internal or external) on how Agile can benefit them, why the organization is considering trying this out, what existing pain points we are trying to address, how they can engage effectively and manage their other duties.
  - Doing this early along with showing measurable results will go a long way towards a success transformation.
  - We've found that once the business customers try Agile the right way, they seldom want to go back to traditional methods of delivery.

Success Factors when Transforming to Agile

- Engage a strong expert partner.
  - Adopting Agile successfully is hard when you don't have any actual experience with it yourself.
  - Agile is easy in theory but hard in practice because of the change management involved so many teams revert to form quickly when facing challenges.
  - Use a combination of 'Transformation Workshops' and coaching.
  - The workshops help transform people's hearts and minds to being open to trying things differently and showing them 'how'.
  - The coaching helps guide the leaders and teams through this transformation every step of the way (dedicated or on demand).
  - Your partners should be there with you through every step of Piloting, Scaling and Enterprise Transformation while being as light touch or as dedicated with the help based on your needs.

Sample Agile Transformation Roadmap

Transformation Foundation

- Initial Assessment
- Transformation Vision and Objectives
- Design Initial Transformation Roadmap
- Educate Stakeholders, Gain Buy-in
- Standup Agile Champions team
- Build Backlog for Upcoming Iterations

Pilot Agile

- Initial executive and leadership team education on Agile
- Design, Identify, train and standup pilot teams
- Active Agile Coaching (Backlog, Release Planning, Velocity & Tracking)
- Engage and educate business customers
- Prove success, gain buy-in, measure outcomes
- Begin to develop Agile Toolkit (roles, processes..)
- Focus Championsteam on impediment removal
- Standup initial CoP
- Initial Agile Leadership, Collaboration, Facilitation transformation activities
- Initial technical assessment and transfomation roadmap

Scale to Several Teams

- Educate leadership and gain buy-in for scaling
- Design and Standup several teams for a specific program or by department/area
- Apply Agile practices for program management
- Begin Education on Agile Portfolio Management
- Roll out training, education, coaching accross the teams
- Roll out intermediate deep-dive agile training, soft skills and servant leadership
- Company wide communication
- Design and Execute Organization change management plan
- Launch Agile learning portal and internal training programs
- Launch more CoPs
- Execute technical transformation

Enterprise Transformation

- Educate leadership and gain buy in for enterprise transformation
- Design and standup the enterprise leadership triangle and initate their process cadence
- Design and Standup stable teams acorss the enterprise
- Stabalize CoPs
- Apply Agile to HR performance reviews, hiring, education/training, culture transformation
- Apply Agile to Finance budgetting and planning processes
- Apply Agile to Legal vendor contracts
- Support the Compliance/Audit department's understanding of Agile
- Execute enterprise technology transformation roadmap
- 'Make it Stick' Plan

Transformation Planning, Execution, Measurement, Inspection and Adaption

Stages of Agile Transformation

1. GETTING STARTED

Set the direction of your Agile transformation:

- Organizational Assessment & Awareness
- Management Commitment
- Goals Set
- Address Organizational Barriers
- A Few Driving the Initiative

2. PILOT AGILE

Begin to define how Agile will be incorporated within the organization, including:

- Executive Alignment
- Conduct Basic Training
- Adopt Agile Culture
- Break Down Silos
- Build Collaboration

3. IMPLEMENT AGILE

With a practice in place, this stage involves rolling out Agile to new teams and adding more advanced Agile practices, including:

- Role-based Training
- Increasing Improvement
- Growing Level of Standardization
- High Performing Teams
- Removing Organizational Impediments
- Relevant Metrics and Reporting

4. ENTERPRISE AGILE

Scaling Agile to the enterprise can be extremely difficult. This stage involves reevaluating your organization, including:

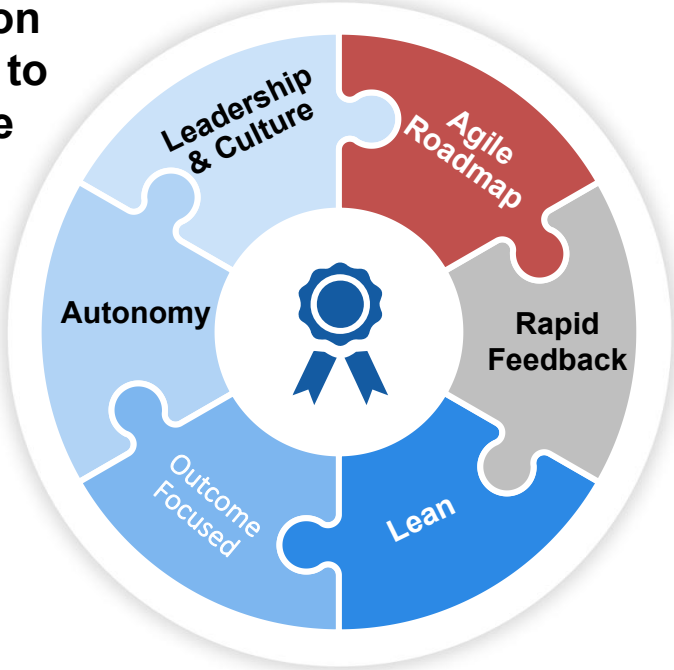
- Cross Functional Teams
- Real Time Metrics
- Continuous Feedback
- Automation and Reuse
- Culture Adopted

Six Broad Challenges of Agile Transition

1. Successful transformation needs strong leadership to support and protect agile culture.

2. Teams and stakeholders need to self-organize.

3. Manage your portfolio with Outcomes (not Output)

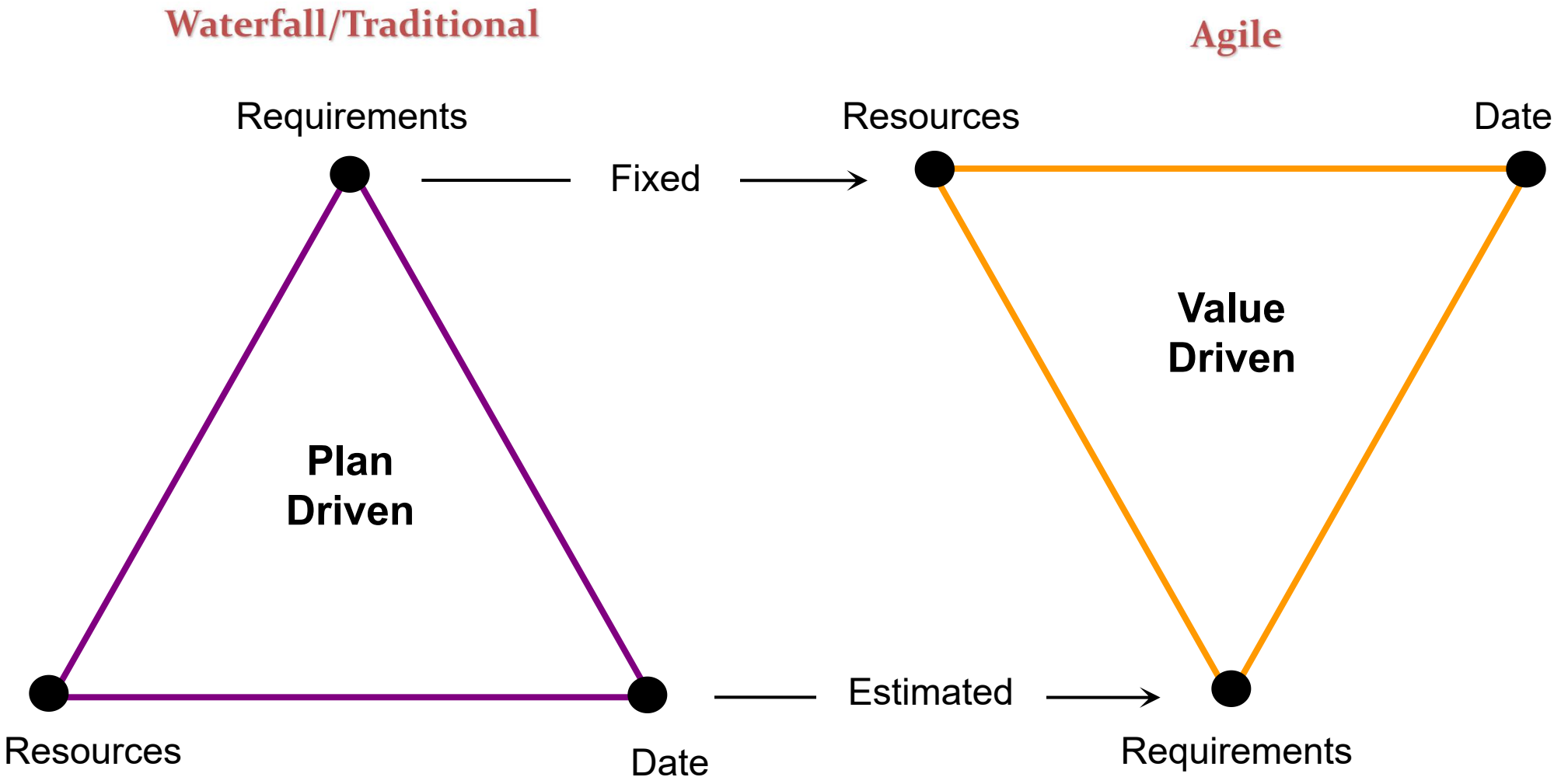


6. A clear agile roadmap properly aligned with business goals

5. Measure and improve value delivered with frequent feedback (Inspect and Adapt).

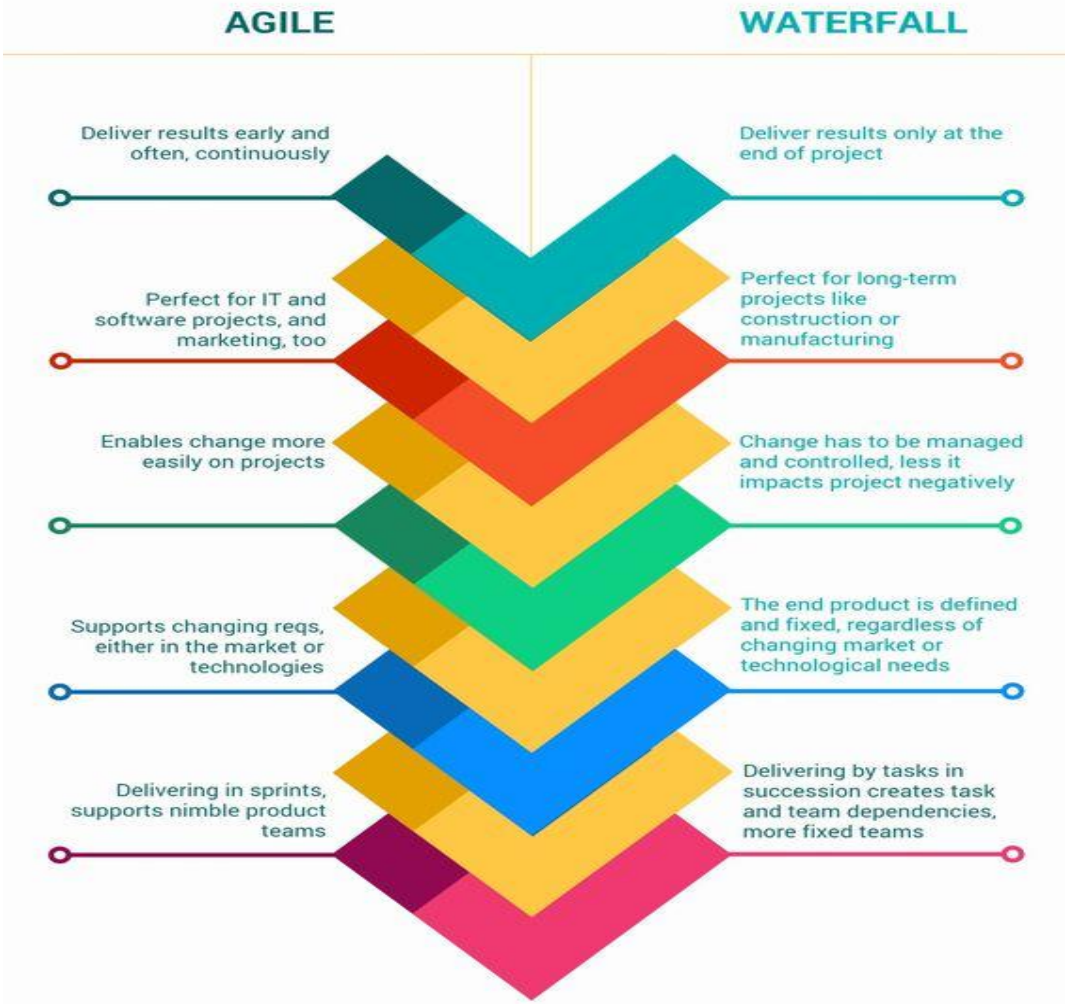
4. Systematically remove sources of waste and delay faced by Agile teams

Agile fixes the date & resources & varies the scope



AGILE VS. WATERFALL

FOR PROJECTS



Project vs Product

- A project is a temporary endeavor undertaken to create a unique product, service or result.

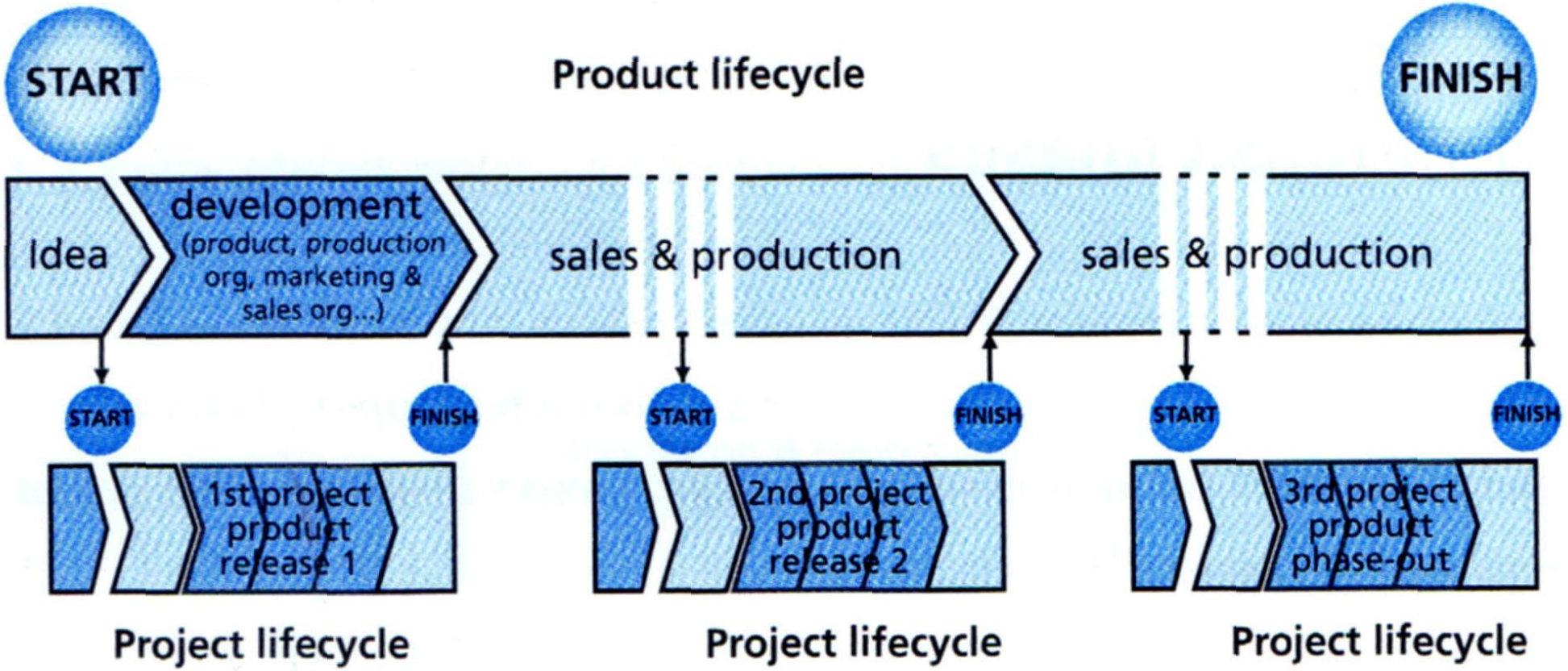
versus

- Product management is the discipline and business process governing a product from its inception to the market or customer delivery and service in order to generate the largest possible value to a business.

Project vs Product *(cont'd)*

- A project is all about the creation of something but when the something is created the project is done.
- Projects are not concerned with the ongoing improvement or enhancement of a product over the entire lifecycle of that product.
- Product management is concerned with the life of the product; from conception, through development and eventually to discontinuation.
- Product management is the sweet spot for agile.

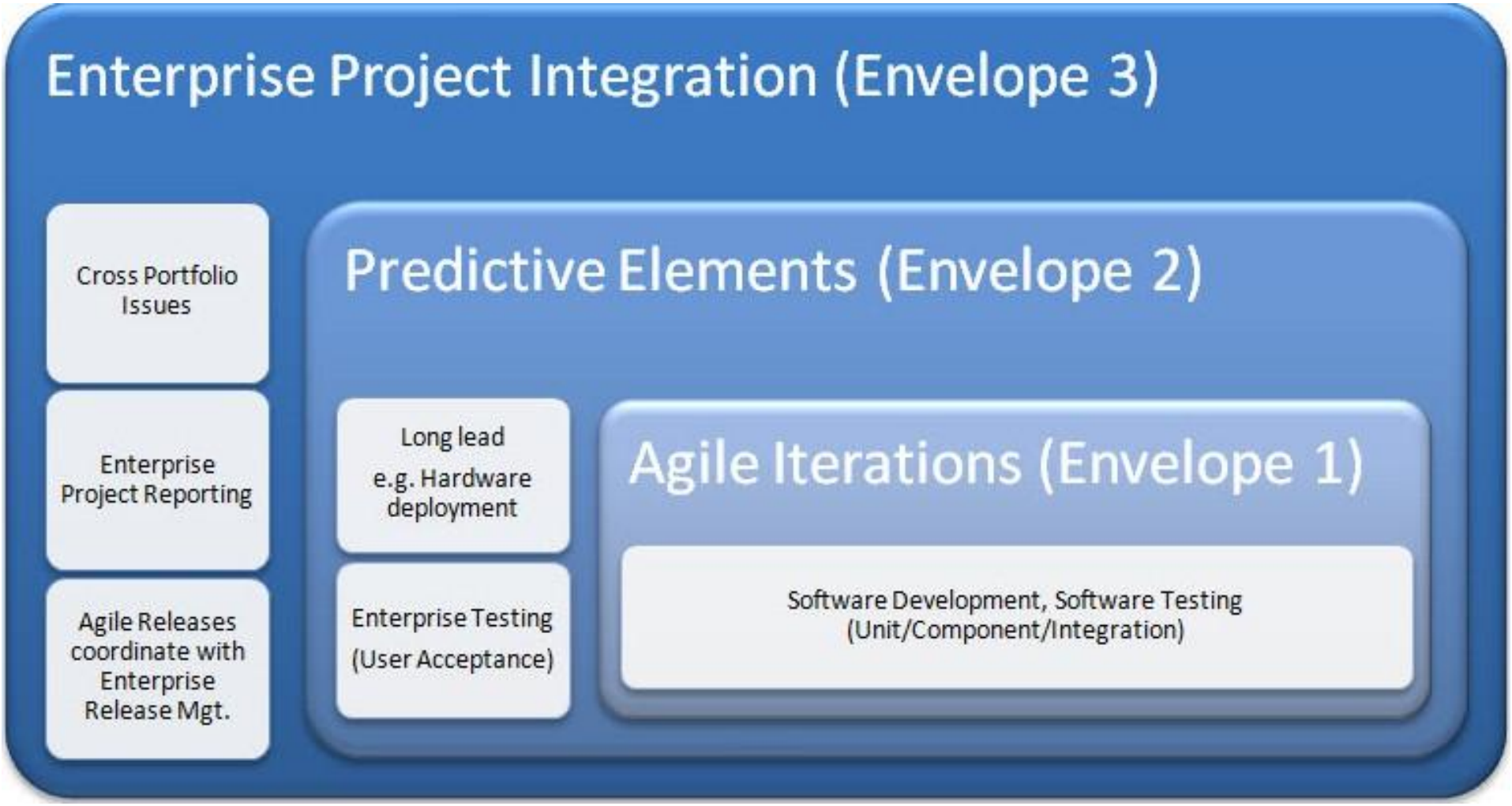
Project Lifecycle as part of a Product Lifecycle



Differences in Triple Constraints between Project & Product Management

	Project Management	Product Development
Scope	Fixed set of features	On-going prioritized list of features
Schedule	Start and end date	multiple releases
Budget	Allocated once	Cyclical

A Framework for Mixing Agile and Waterfall in the Same Project - The Envelope Method



The Envelope Method

- Using this approach the Project Manager is responsible for keeping the agile team protected, particularly during the iteration.
- Nothing should interfere with the team's delivery of working software by the end of the iteration.
- The project owner may change the backlog, the platform services group may make modifications to the environments.
- But every effort should be taken by the project manager(s) to protect the team so they can do what they do best without interruption.

The Envelope Method

- *Envelope 1*
  - The innermost Envelope is where the agile team works to execute the software development, unit testing, component testing, integration testing and defect resolution.
  - The Agile Team Lead (in Scrum the Scrum Master, or Iteration Manager) is the first line of issue resolution.
  - The Agile Team Lead is responsible for intra-team communication, for coaching the development team, for helping prioritize work when the team is stuck.
  - Generally the Agile Team Lead is responsible for the health and wellbeing of the agile team.
  - Impediments that the Agile Team Lead cannot resolve for the team are escalated to the Project Manager.

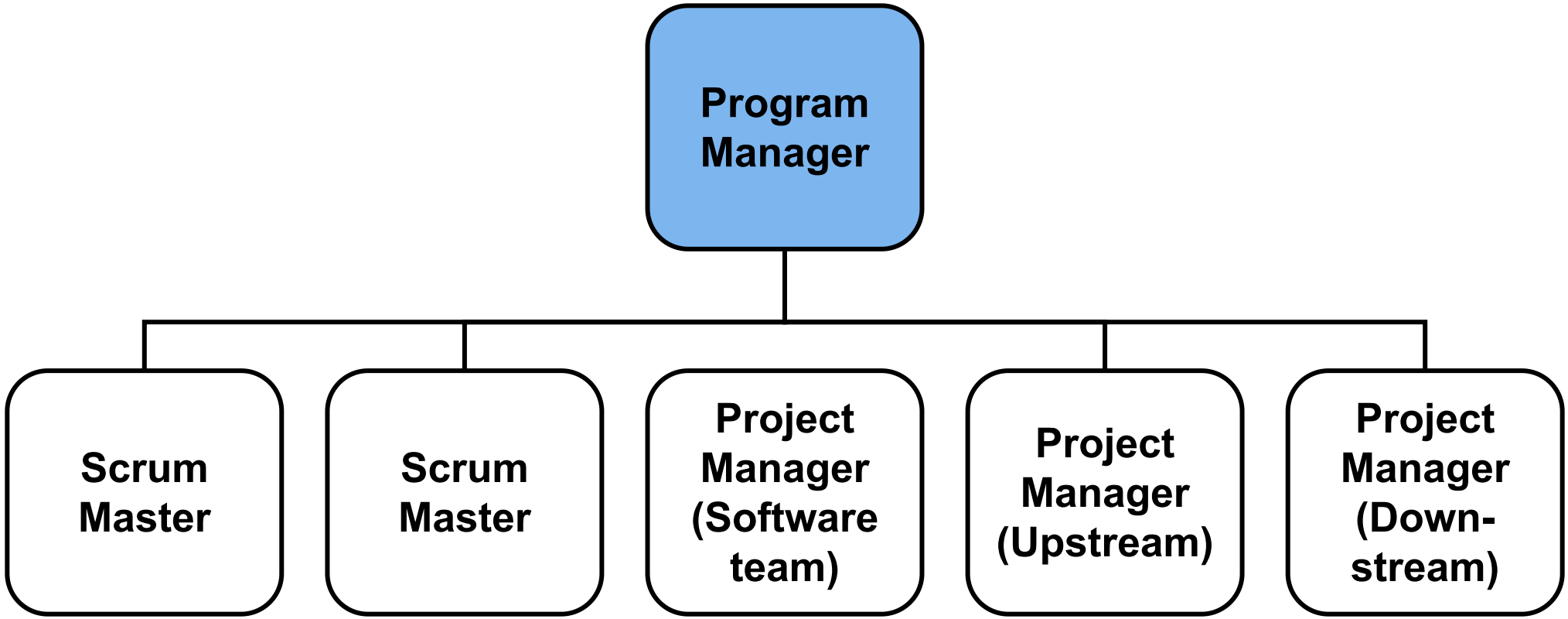
The Envelope Method

- *Envelopes 2 and 3*
  - Predictive elements lie at the second and third layers of the envelope framework.
  - The second layer is where the waterfall elements of the project are planned and managed.
    - These elements could be anything but one prime example is hardware delivery.
  - The third envelope is where the project interfaces with the rest of the enterprise portfolio.
    - At this layer the project manager coordinates with external projects, with the enterprise release planning and enterprise reporting.

2 Key Sources of Conflict in Hybrid Projects

	AGILE	WATERFALL
Culture	Organic, fluid, open	Command & control, structured, rule-bound
Cadence	Rapid and cyclical	Slow and steady

Agile PMO Setup



Program Manager must be equally adept at agile and non-agile projects

Differences in the PM Roles

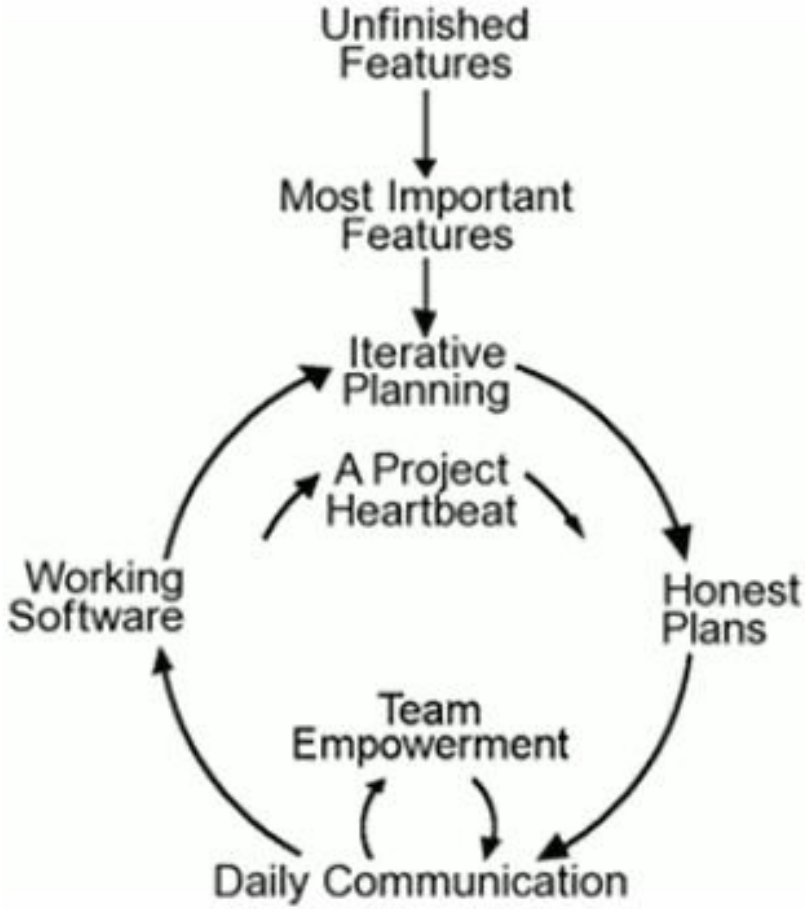
Agile PM	Traditional PM
Focuses on team communication and interaction	Focuses on processes and tools
Places priority on developing software and/or solutions that will be progressively modified and improved	Anticipates limited changes and requires comprehensive documentation
Emphasizes the importance of customer — project team collaboration and daily communication	Emphasizes the importance of contract negotiation and tasks delineated in the contract
Features flexibility and response to change	Works the plan; follows the plan to the end

Assessing Project Manager Readiness to Execute Agile Software Development

- Degree to which the project manager focuses on the customer rather than on following standard project management procedures
- Degree to which the project manager values innovation and practical processes over sticking with the plan
- Degree to which the project manager is comfortable with an uncertain and changing environment
- Project manager's skill and commitment to sharing information as needed with all stakeholders
- Project manager's level of commitment to the team and the willingness to promote team collaboration
- Project manager's ability to motivate the team, delegate, and then get out of the way

XP - Extreme Programming

- improve **software quality** and **responsiveness** to changing customer requirements
- **frequent releases** in short **development cycles**
- **improve productivity** and **regular checkpoints** with the customer
- **paired programming**



Uncle Bob's Original Three Rules of TDD

1. You are not allowed to write any production code unless it is to make a failing unit test pass.
2. You are not allowed to write any more of a unit test than is sufficient to fail; and compilation failures are failures.
3. You are not allowed to write any more production code than is sufficient to pass the one failing unit test.

Uncle Bob is Robert C. Martin

Uncle Bob “Refactored” Three Rules of TDD

Uncle Bob's Three Rules of TDD

- Write **no** production code, except to pass a failing test.
- Write only **enough** of a test to demonstrate a failure.
- Write only **enough** production code to pass a test.

Benefits in following the 3 TDD Laws

- **Debugging**. What would programming be like if you were never more than a few minutes away from running everything and seeing it work? Imagine working on a project where you never have several modules torn to shreds hoping you can get them all put back together next Tuesday. Imagine your debug time shrinking to the extent that you lose the muscle memory for your debugging hot keys.
- **Courage**. Have you ever seen code in a module that was so ugly that your first reaction was "Wow, I should clean this." Of course your next reaction was: "I'm not touching it!" Why? Because you were afraid you'd break it. How much code could be cleaned if we could conquer our fear of breaking it? If you have a suite of tests that you trust, then you are not afraid to make changes. You are not afraid to make changes! You see a messy function, you clean it. All the tests pass! The tests give you the courage to clean the code! Nothing makes a system more flexible than a suite of tests — nothing. If you have a beautiful design and architecture, but have no tests, you are still afraid to change the code. But if you have a suite of high-coverage tests then, even if your design and architecture are terrible, you are not afraid to clean up the design and architecture.

Benefits in following the 3 TDD Laws

- **Documentation**. Have you ever integrated a third party package? They send you a nice manual written by a tech writer. At the back of that manual is an ugly section where all the code examples are shown. Where's the first place you go? You go to the code examples. You go to the code examples because that's where the truth is. Unit tests are just like those code examples. If you want to know how to call a method, there are tests that call that method every way it can be called. These tests are small, focused documents that describe how to use the production code. They are design documents that are written in a language that the programmers understand; are unambiguous; are so formal that they execute; and cannot get out of sync with the production code.
- **Design**. If you follow the three laws every module will be testable by definition. And another word for testable is decoupled. In order to write your tests first, you have to decouple the units you are testing from the rest of the system. There's simply no other way to do it. This means your designs are more flexible, more maintainable, and just cleaner.
- **Professionalism**. Given that these benefits are real, the bottom line is that it would be unprofessional not to adopt the practice that yields them.

Three Types of Trust

- **Contractual** (roles & responsibilities)
- **Communication** (transparent, open, honest, and frequent)
- **Competence** (respect for each other's skills, recognition of interdependence)

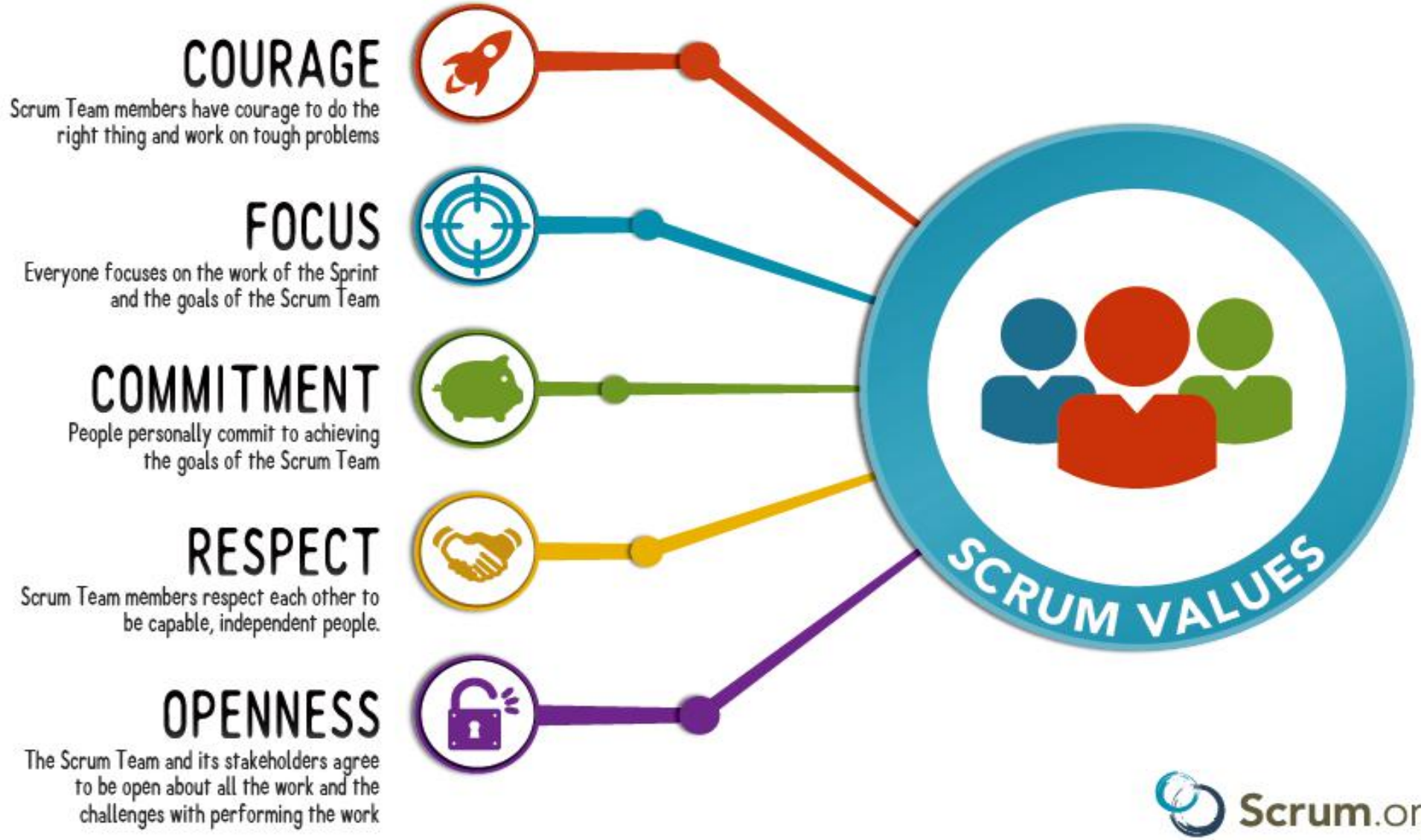
Behaviors that Build Trust		
1)	**Talk Straight**	Be honest, leave no doubt about what you are thinking
2)	**Demonstrate Respect**	Fairness, kindness, civility
3)	**Create Transparency**	Be open and authentic
4)	**Right Wrongs**	Apologize quickly, humility
5)	**Show Loyalty**	Give credit to others
6)	**Deliver Results**	Establish a track record of making things happen
7)	**Get Better**	Continuously learning, growing
8)	**Confront Reality**	Problems don't get better with time – address issues, lead courageously
9)	**Clarify Expectations**	Create shared vision and agreement up front
10)	**Practice Accountability**	Hold yourself and others accountable
11)	**Listen First**	Genuinely understand others opinions
12)	**Keep Commitments**	Do what you say you are going to do
13)	**Extend Trust**	Extending trust to others creates reciprocity

https://www.speedoftrust.com/#the-book

The Three Pillars of Scrum

Transparency	Inspection	Adaptation
• Scrum process must be visible to everyone in the team, and the standards must be common & understood by everyone on the team • The language used to describe the process must be shared by the team	• Development work & artifacts being produced must be inspected frequently in order to identify any variations from the desired goal • These inspections should be done in such a way that they do not obstruct the ongoing work	• Following an inspection, if it is deemed that one or more aspects of the process have deviated outside of acceptable levels of tolerance & that the resulting product would be unacceptable, the process or the product will need to be adjusted • That adjustment has to be made as soon as possible to reduce the risk of further deviation

Scrum Values



Scrum.org

Core Values of Scrum

- **Commitment**
  - Each individual must commit to fulfilling their role to the best of their ability. For example, the Team must commit to delivering the Sprint goal that they have promised to deliver.
  - And the Product Owner must commit to striving to provide the best value to the stakeholders.
  - The ScrumMaster, of course, must commit to the Scrum process and being a facilitator and coach.
- **Focus**
  - Means that we concentrate on and are answerable for doing the things that we have committed ourselves to do, rather than allowing ourselves to become distracted or diverted.

Core Values of Scrum

- **Openness**
  - Open communication at all levels without fear of blame or retribution.
  - Openness at all level because it is the right thing to do, not because it is required by management or someone else on the receiving end.

Core Values of Scrum

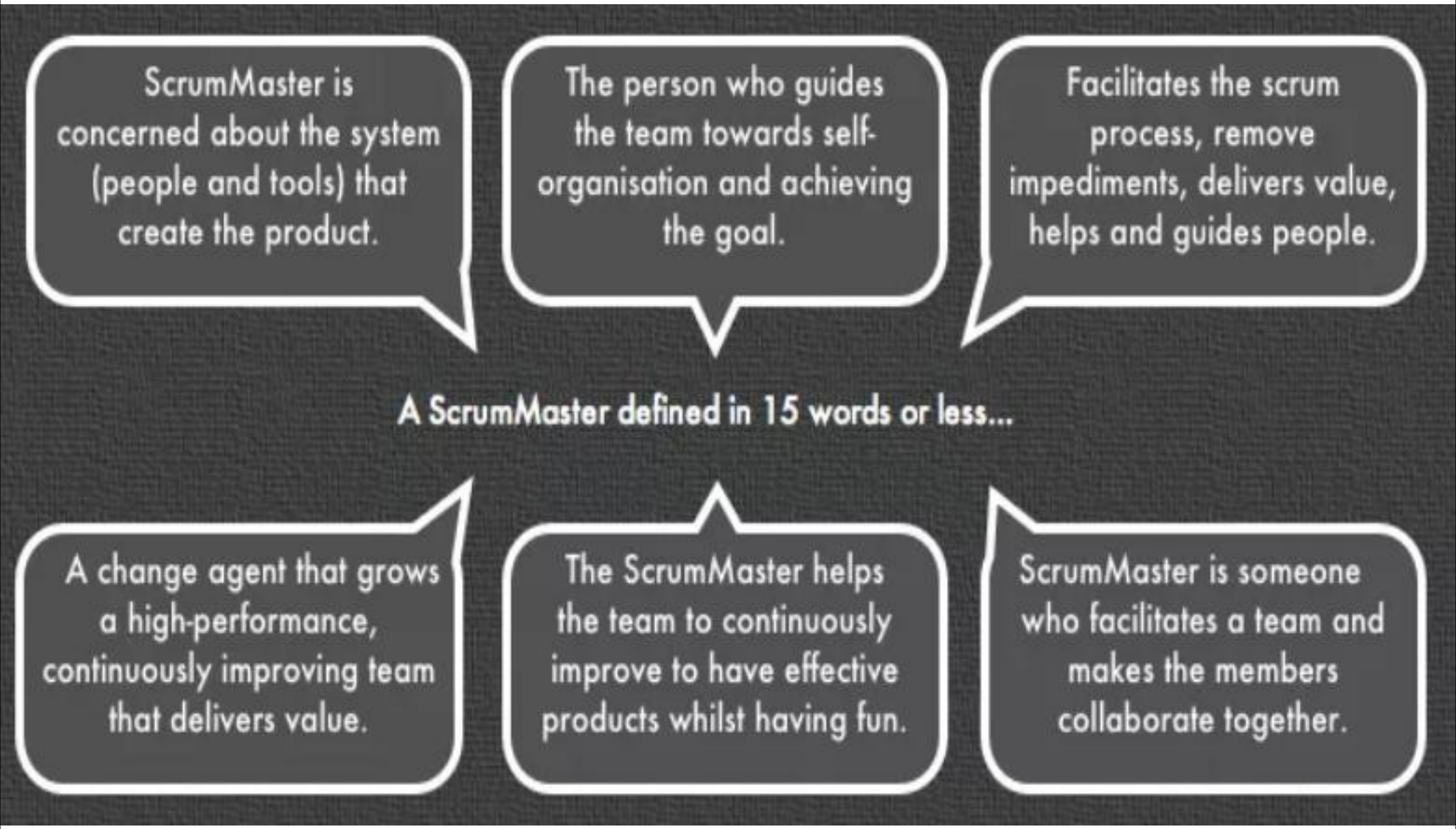
- **Respect**
  - Respect means that we show a high regard for ourselves, others, and the resources entrusted to us.
    - Resources entrusted to us may include people, money, reputation, the safety of others, and natural or environmental resources.
  - An environment of respect engenders trust, confidence, and performance excellence by fostering mutual cooperation and collaboration — an environment where diverse perspectives and views are encouraged and valued.
  - Each individual has something to contribute and be treated with respect.

Core Values of Scrum

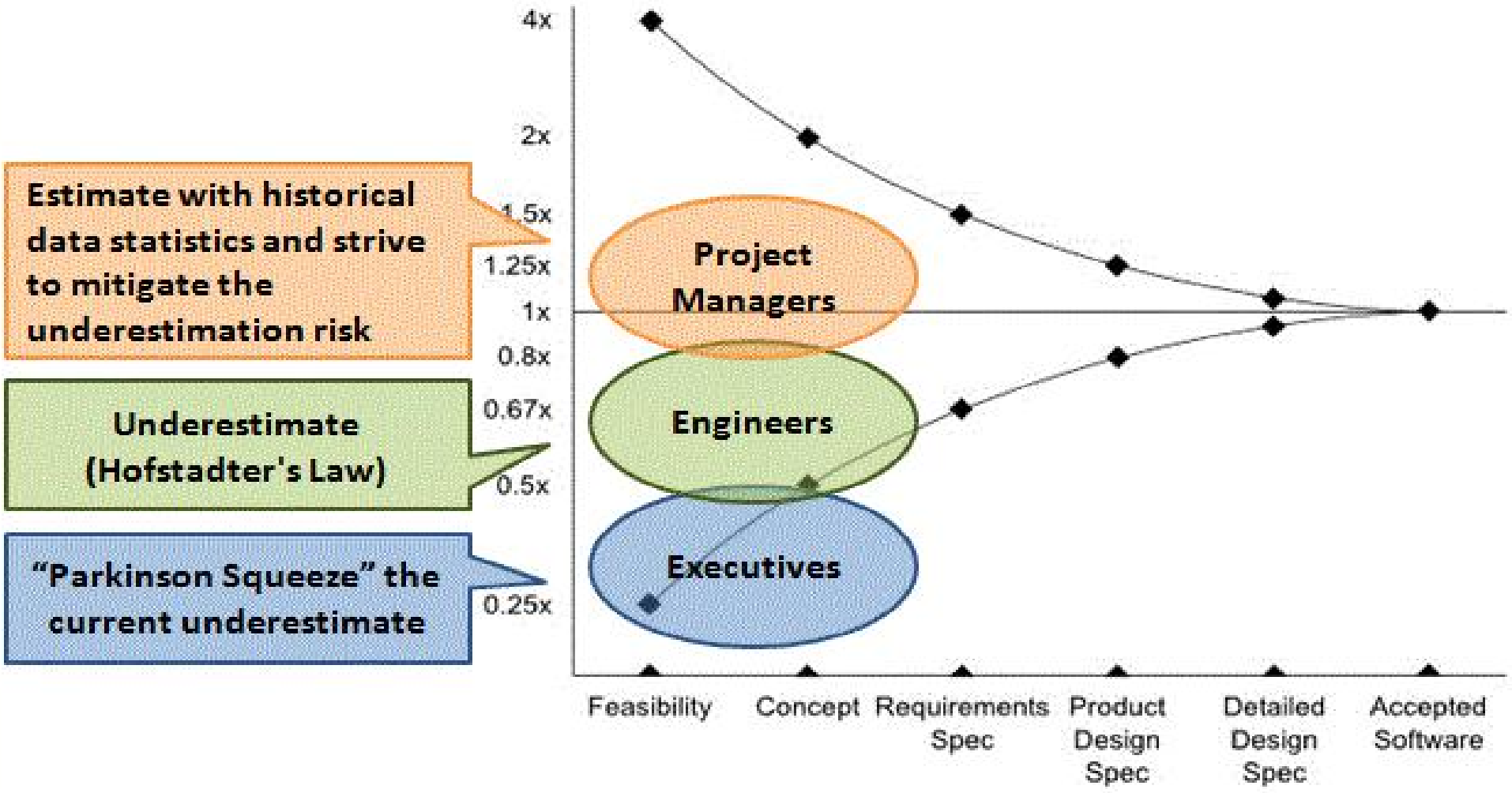
- **Courage**
  - Courage means that we have the daring to do the best that we can and the endurance not to give up.
  - We have the determination and resolution to take ownership of the decisions we make or fail to make, the actions we take or fail to take, and the consequences that result.
  - It also takes courage to communicate truths that don’t always want to be heard by sponsors or management.

The Scrum framework in 30 seconds

- A product owner creates a prioritized wish list called a product backlog.
- During sprint planning, the team pulls a small chunk from the top of that wish list, a sprint backlog, and decides how to implement those pieces.
- The team has a certain amount of time — a sprint (usually two to four weeks) — to complete its work, but it meets each day to assess its progress (daily Scrum).
- Along the way, the ScrumMaster keeps the team focused on its goal.
- At the end of the sprint, the work should be potentially shippable: ready to hand to a customer, put on a store shelf, or show to a stakeholder.
- The sprint ends with a sprint review and retrospective.
- As the next sprint begins, the team chooses another chunk of the product backlog and begins working again.


ScrumMaster is concerned about the system (people and tools) that create the product.
The person who guides the team towards self-organisation and achieving the goal.
Facilitates the scrum process, remove impediments, delivers value, helps and guides people.
A ScrumMaster defined in 15 words or less...
A change agent that grows a high-performance, continuously improving team that delivers value.
The ScrumMaster helps the team to continuously improve to have effective products whilst having fun.
ScrumMaster is someone who facilitates a team and makes the members collaborate together.

The Cone Of Uncertainty



Tuesday morning, 10:30 am: Status notification from the project lead.

"Hello team, please review the attached updated requirements specification document (filename: Project_Wishful_Thinking_v259.doc). Requirement 1.45 changed, see page 512 in red (ignore the blue and green revision marks from last week and be sure to read the embedded comments I've retyped from my conversation with our customer).

Sorry for the earlier confusion to the Testing team – you'll need to rewrite your test cases, as I attached the wrong version last week. Please make sure you are writing your test cases from this document, not the previous version. Thanks. - Barry"

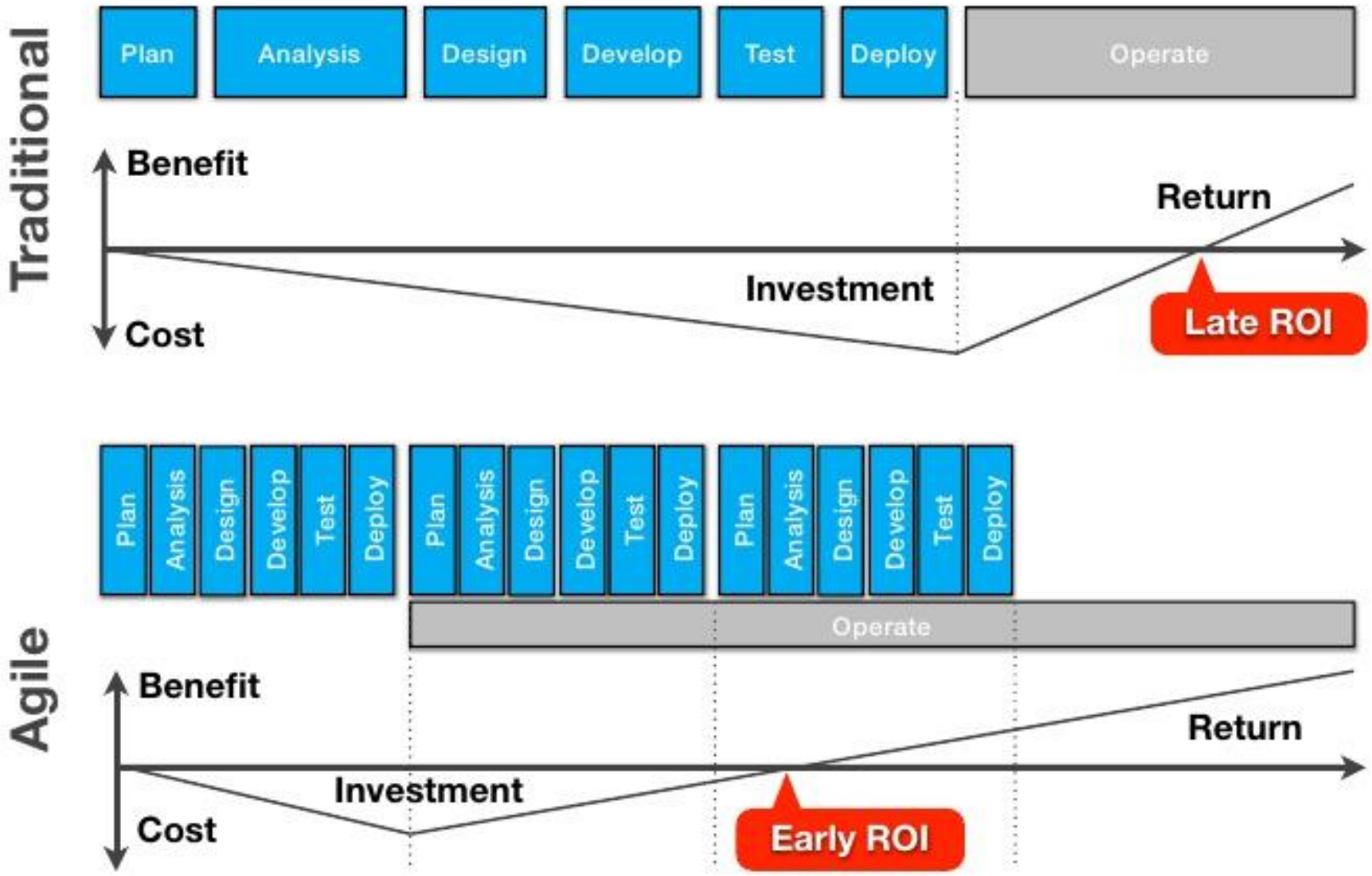
10 minutes later: "Me again, Craig found an error and fixed it, see the revised attachment."

15 minutes later: "People are asking what changed, sorry I wasn't more specific – see page 45, the blue section with my comments."

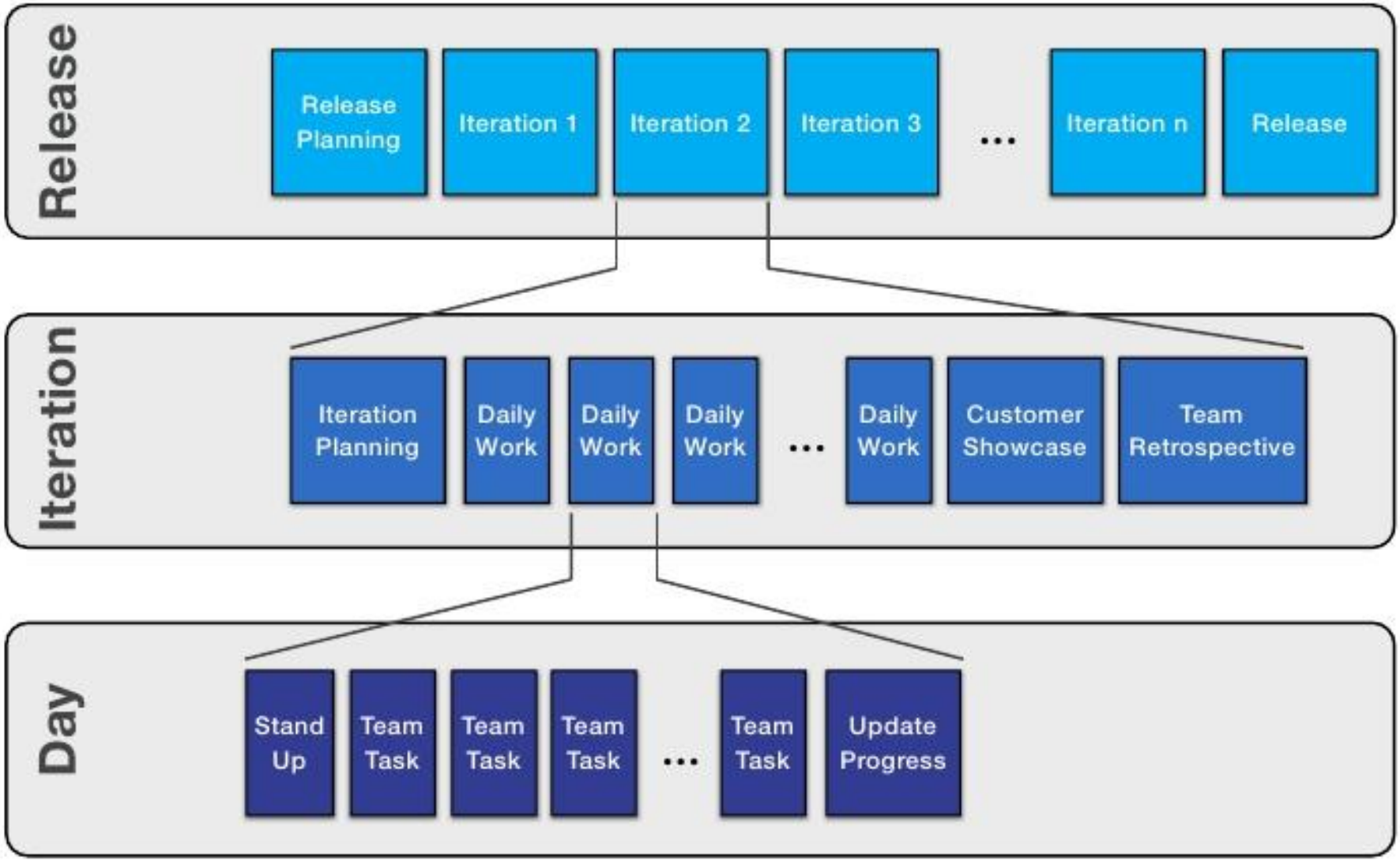
2 hours later: "Team, we need to stop all work on the interface with the front-end customer system and change to interface with the back-end system. I need to figure out the impact by the end of the day (no pun intended), so if you know anything about this – let me know. Updated requirements doc is attached. Project_Wishful_Thinking_v261.doc."

4 hours later: "The steering committee met, and needs a report with all the requirements being delivered in Phase I for a meeting tomorrow with Executive Staff. Can one of the analysts compile a list, get status from the developers and get this to me ASAP. Actually, double ASAP. Thanks."

Traditional vs Agile



Agile Delivery Layers



Key Issues Implementing Agile

1. Mindset change – when you catch yourself in the trap of saying “this is just like what we’re already doing except…” you’ve missed the key point. Agile Software development requires a complete mindset change. While some of the events (i.e. Daily Scrum) may be superficially similar the mindset is different.
2. Self Organization – self organization is messy and takes time to achieve yet its at the heart of a high performing Scrum team. So its sad to see Scrum Master, an Architect or other team member with a leadership position taking control. This problem can be obvious or subtle but either way they’re not giving the team room to grow.

Key Issues Implementing Agile

3. Meetings – aka Mechanical Scrum/Agile, you can hold all the right meetings (Planning, Daily Scrum/Standup, Review and Retrospective) if the joy and spirit is missing then they will be wooden events that don't but help the team collaborate and grow.
4. No Engineering Practices – Agile processes (Scrum, OpenAgile, Kanban etc.) tend to lead to an immediate speed up in the development process. But if there is no focus on working on Engineering practices (TDD, ATDD and Pair Programming), then the team will soon end up hip deep in Technical Debt.

Key Issues Implementing Agile

5. Retrospectives without improvement – a retrospective who's action items are not acted on quickly becomes a meeting people will have no respect for. Take the SMART actions from your retrospective, post them on your task board and check in on them in stand-up everyday.
6. Command and Control Agile – A Scrum Master, XP Coach or even Open Agile Process Facilitator isn't a Project Manager. They don't assign tasks or tell team members what to do. Instead they're there to help the team achieve its goal and to help them grow into a high performing team.

Key Issues Implementing Agile

7. Lack of Real Commitment – an Agile transition is hard work and requires time. There will be some bumpy moments during this time. Full commitment (not just $$$) is required from Management who have to demonstrate they support the change and will help to remove impediments.
8. Nothing Ever Changes Around Here – the team raises impediments but either doesn't work on removing them or isn't able to remove them without more help. They will quickly get the message that the organization isn't on board. Management must support the team by helping to expose impediments and get them resolved.

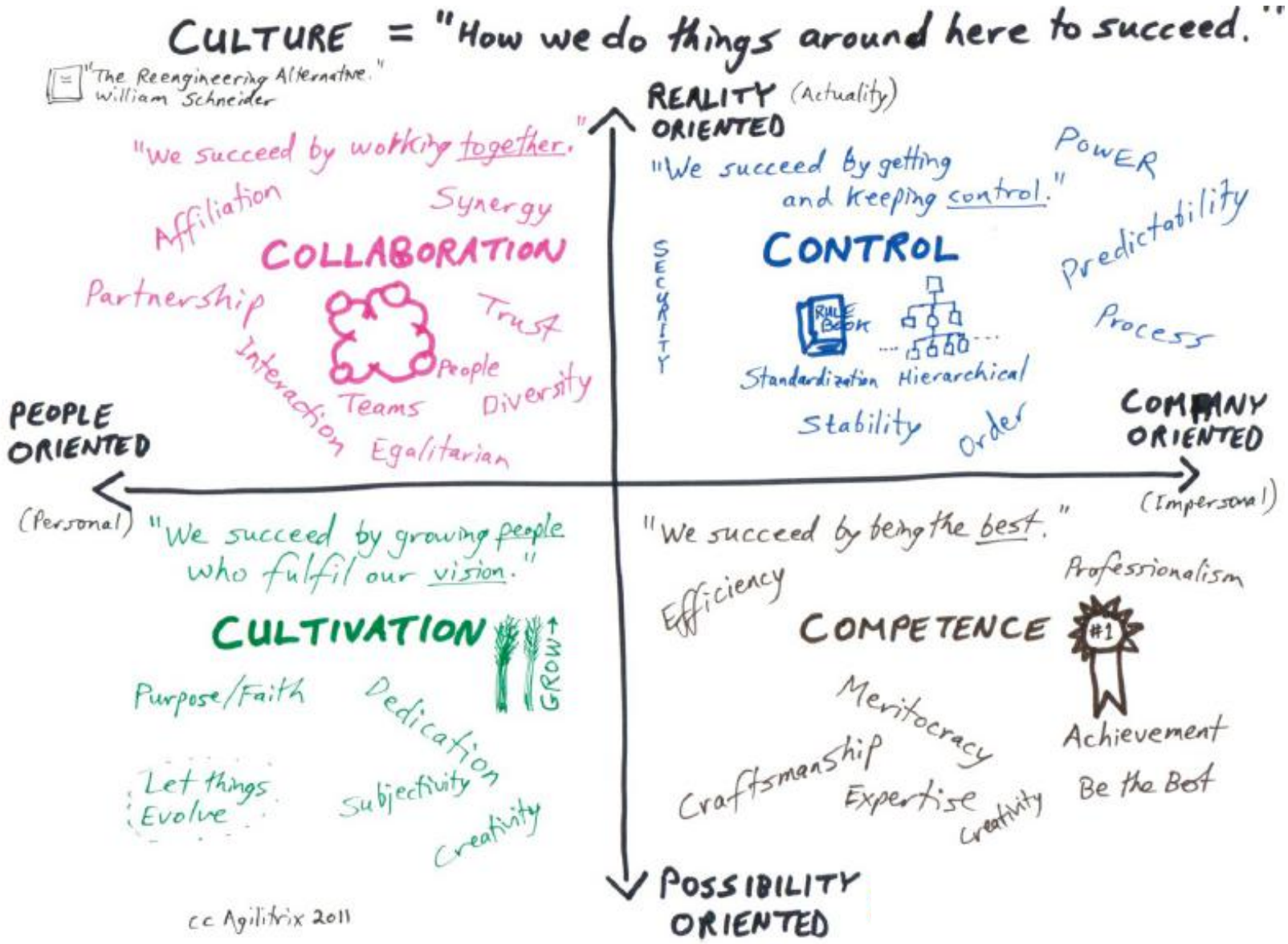
Key Issues Implementing Agile

9. Rushing to make the Planning Commitment – Teams that are new to Agile often overcommit in their first few sprints (aka iterations), sometimes by as 200%. When this happens they often cut corners to get all the work done, with classic statements like: “We don't have time to write Unit Tests”, “We don't have time to test that properly”. A good Agile team starts only a few User Stories at a time and then gets them to Done. That often means completing fewer stories than originally committed, its better to do this then have to rework previously “completed” stories because of corners cut.
10. Attempting to Scale at the Start – Organizations that are adopting Agile often want to work at scale (more than 3 teams) right off the bat. Its difficult to get a few teams off to a good start, without complicating matters with additional communication problems. Launching at scale can be done, but it tends to require a lot of coaching support. If you're starting on your own, start small.

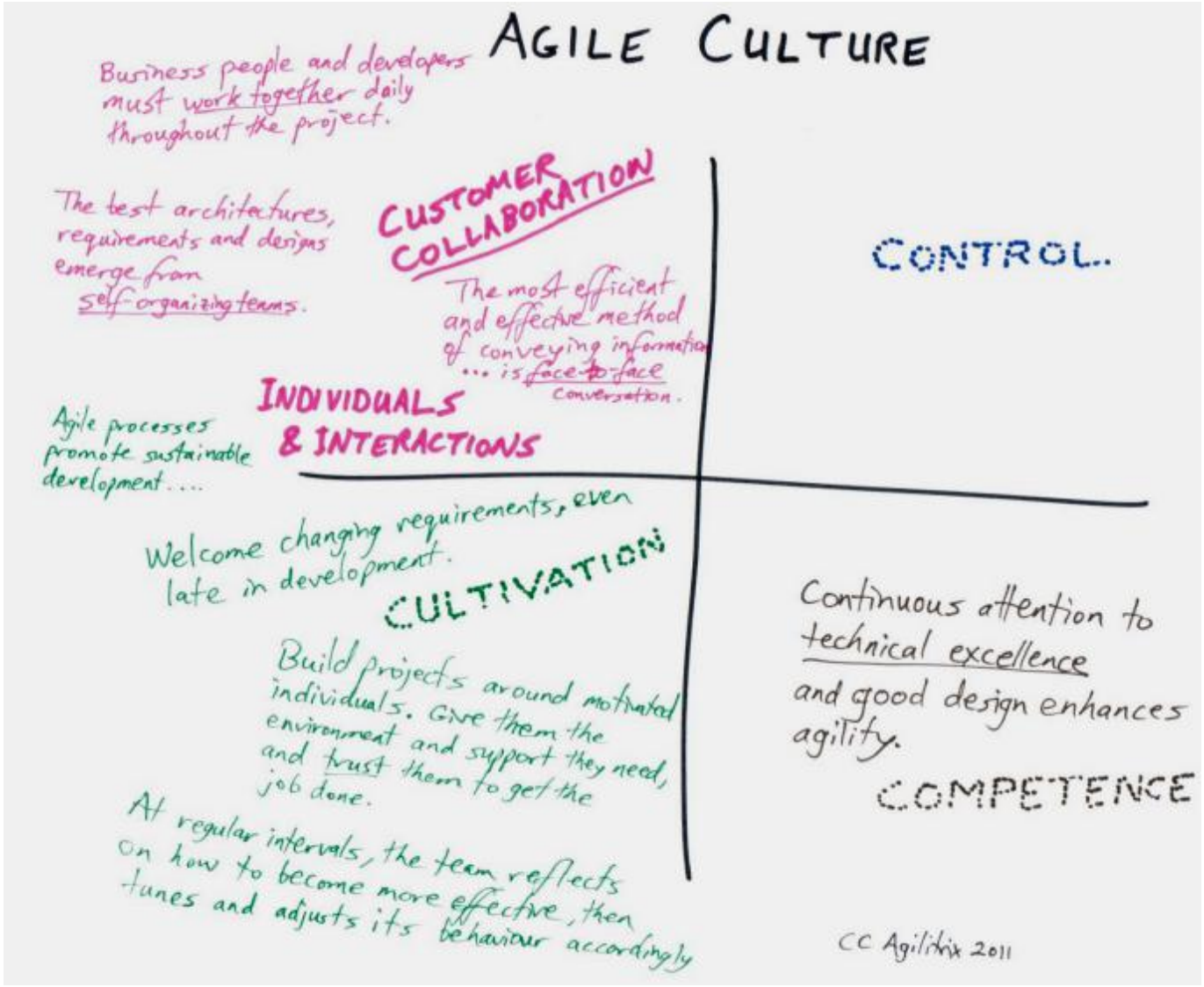
Key Issues Implementing Agile

11. Believing that you're done becoming Agile – An Agile transition is a journey, one that you never really complete. You will always find ways to improve and new experiments to be run. If you keep an open mind and are always looking for new ideas you will undoubtedly scale new heights.

Schneider Culture Model



Mapping Agile Values and Principles to the Schneider Model



Westrum's Typology (2004)

Table 1 How organisations process information

Pathological	Bureaucratic	Generative
Power oriented	**Rule oriented**	**Performance oriented**
Low cooperation	Modest cooperation	High cooperation
Messengers shot	Messengers neglected	Messengers trained
Responsibilities shirked	Narrow responsibilities	Risks are shared
Bridging discouraged	Bridging tolerated	Bridging encouraged
Failure→ scapegoating	Failure→ justice	Failure→ inquiry
Novelty crushed	Novelty→ problems	Novelty implemented

Agile Project Management	Traditional Project Management
Teams are self-directed and are free to accomplish deliverables as they choose, as long as they follow agreed rules.	Teams are typically tightly controlled by a project manager. They work to detailed schedules agreed at the outset.
Project requirements are developed within the process as needs and uses emerge. This could mean that the final outcome is different from the one envisaged at the outset.	Project requirements are identified before the project begins. This can sometimes lead to "scope creep," because stakeholders often ask for more than they need, "just in case."
User testing and customer feedback happen constantly. It's easy to learn from mistakes, implement feedback, and evolve deliverables. However, the constant testing needed for this is labor-intensive, and it can be difficult to manage if users are not engaged.	User testing and customer feedback take place towards the end of the project, when everything has been designed and implemented. This can mean that problems can emerge after the release, sometimes leading to expensive fixes and even public recalls.
Teams constantly assess the scope and direction of their product or project. This means that they can change direction at any time in the process to make sure that their product will meet changing needs. Because of this, however, it can be difficult to write a business case at the outset, because the final outcome is not fully known.	Teams work on a final product that can be delivered some time - often months or years - after the project begins. Sometimes, the end product or project is no longer relevant, because business or customer needs have changed.

Fixed-scope planning

When will all of these be done?

1. Sum all the backlog items the customer needs
2. Measure or estimate velocity as a range
3. Divide total story points by high velocity
   - This is the shortest number of iterations it could take
4. Divide total story points by low velocity
   - This is the “most” iterations it could take

Fixed-scope planning: an example

Total story points desired	120
Low velocity	15
High velocity	20

120 ÷ 20 = 6 iterations

or

120 ÷ 15 = 8 iterations

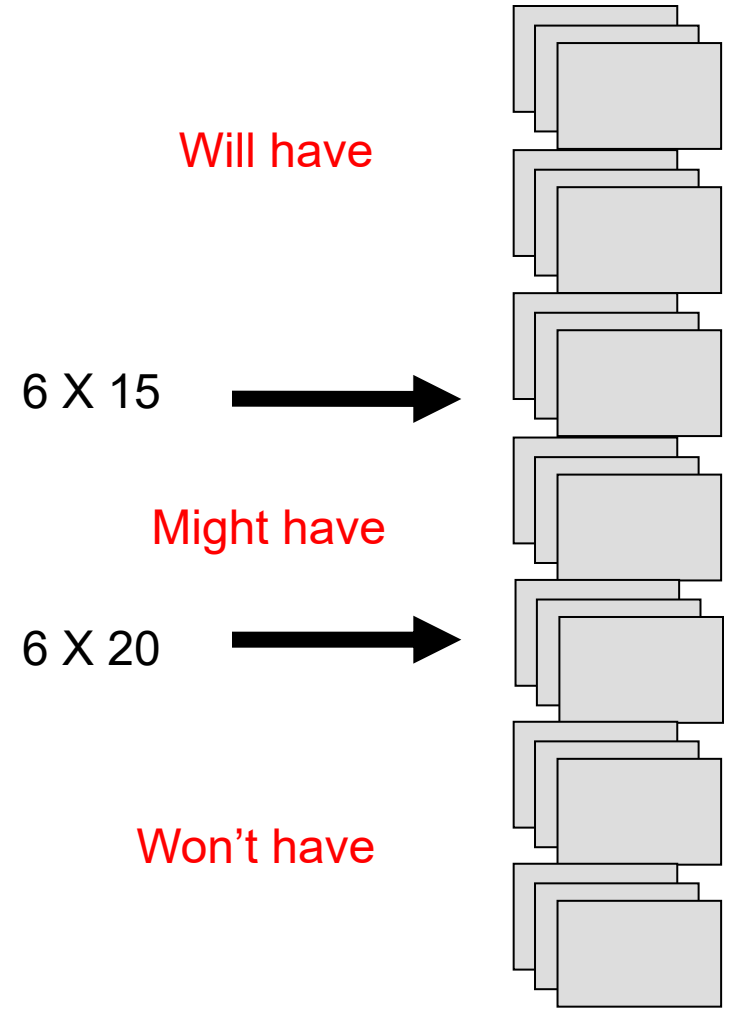
Fixed-date planning

How much can I get by <date>?

1. Determine how many iterations you have
2. Measure or estimate velocity as a range
3. Multiply low velocity by number of iterations
   - Count off that many points
   - These are “Will Have” items
4. Multiply high velocity by number of iterations
   - Count off that many more points
   - These are “Might Have items”

Fixed-date planning: an example

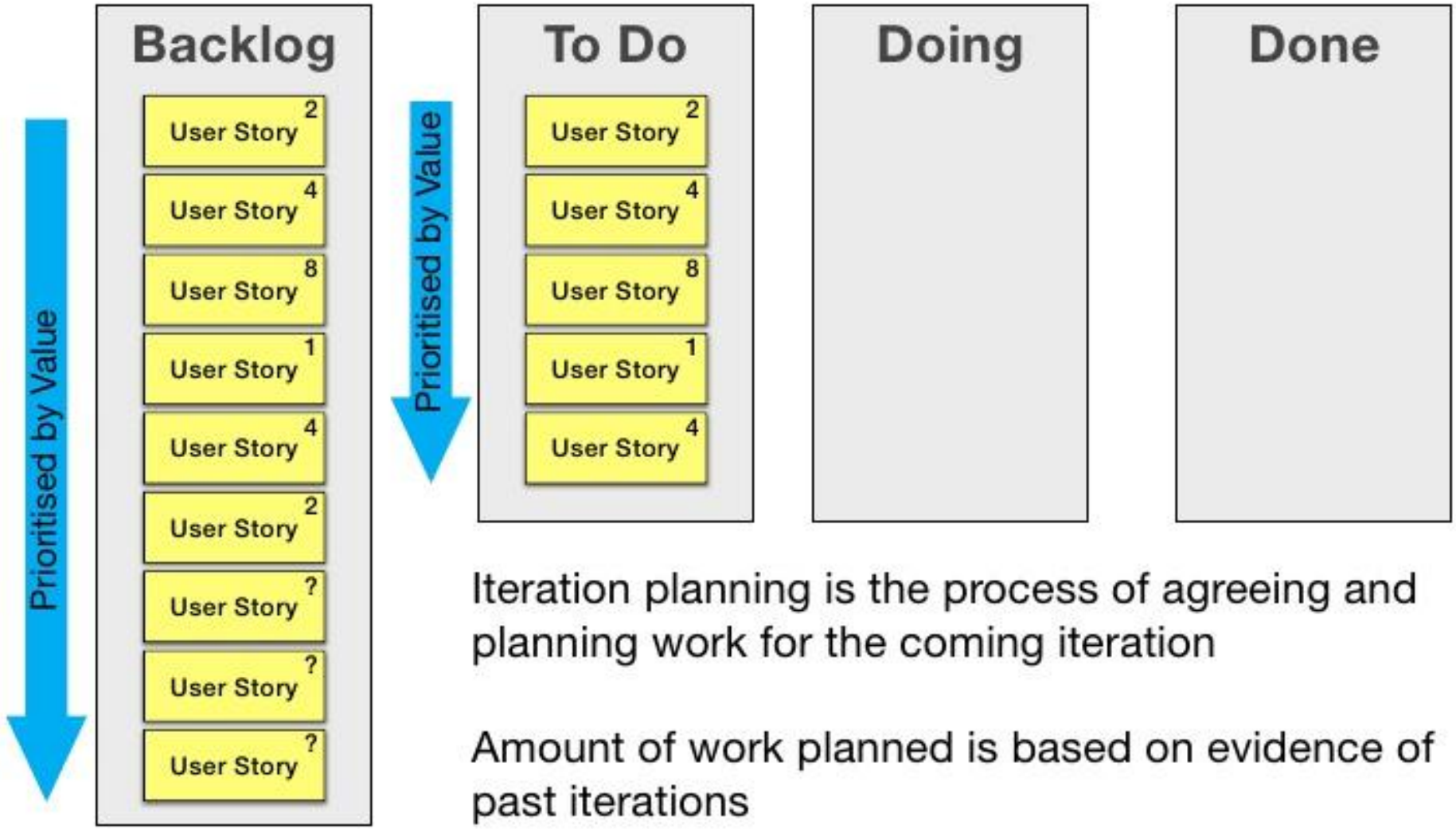
Desired release date	30 June
Today's Date	1 January
Number of iterations	6 (monthly)
Low velocity	15
High velocity	20



Scrum Product Backlog Items

- A typical Scrum backlog comprises the following different types of items:
  - User stories
  - Features/functionalities
  - Defects (Bugs)
  - Technical work
  - Knowledge acquisition
  - Enhancements/stablization
  - Spikes (feasibility analysis, POCs)

Iteration Planning



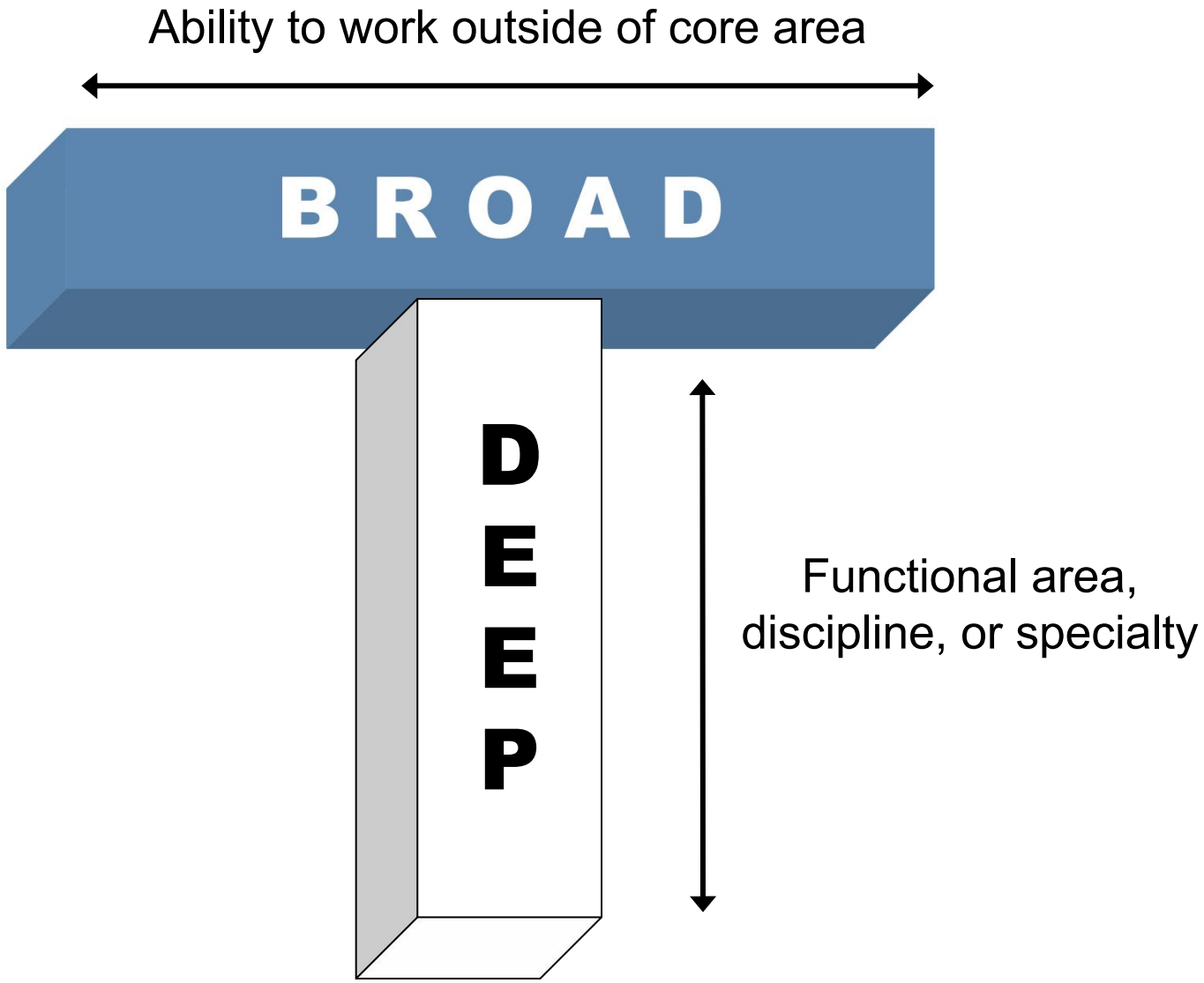
Iteration planning is the process of agreeing and planning work for the coming iteration

Amount of work planned is based on evidence of past iterations

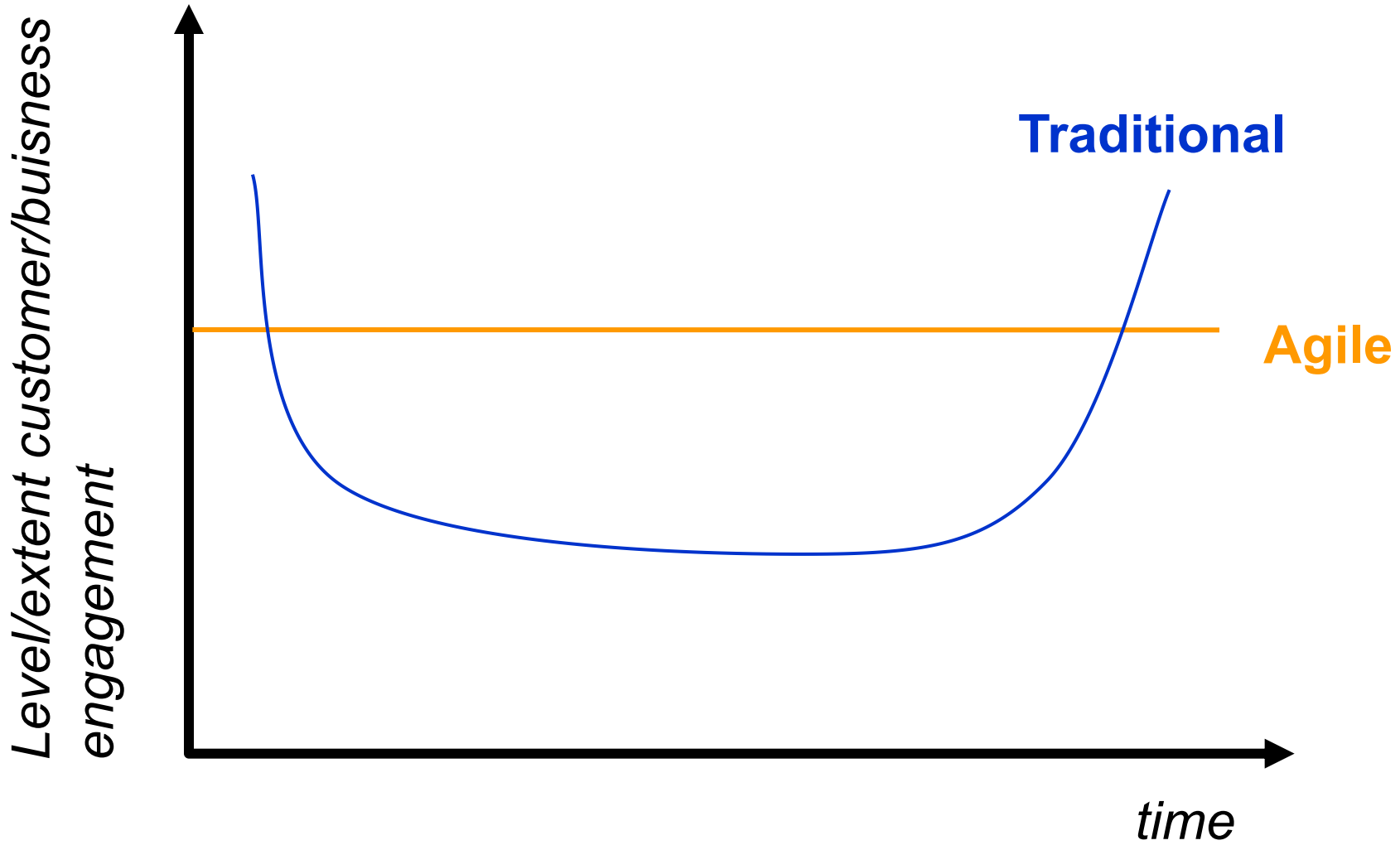
Scrum Framework – Cheat Sheet

Roles	Artifacts	Ceremonies
Product owner What should we work on and why? (Vision)	Product Backlog What are we doing and in what order?	Sprint Planning What are we doing next?
Team members Who will do it and how?	Sprint Backlog What are we doing right now? How will we do it?	Daily Scrum How are we doing today?
Scrum Master Who will help us do it (Process Facilitator)?	Burndown Chart How are we doing this Sprint? How we are doing this release?	Sprint Review How did we do?
	Product Increment This the product ready piece of software released as result of your sprint	Sprint Retrospective How do we get better?

T-shaped Skills



Comparison of Customer/Business Engagement Over Time



5 Deadly Sins in Projects

Point 1: Overambition

Overambition is a strong desire to execute a significant project to gain fame, fortune, or power through the impact of overreaching goals. Overreaching means to defeat oneself by going too far or by doing or trying to gain too much too soon. Sometimes overreaching is good, but both overambition and overreaching should be recognized and the risk considered and managed.

Point 2: Arrogance

Arrogance is the unwarranted, overbearing pride or hubris evidenced by a superior manner toward superiors, peers, and inferiors. Hubris is often associated with a lack of knowledge, interest in, and exploration of history, combined with a lack of humility. In Ancient Greece, "hubris" referred to actions taken in order to shame and humiliate the victim, thereby making oneself seem superior. Really smart people do not need to make themselves feel better at the expense of others.

5 Deadly Sins in Projects

Point 3: Ignorance

Ignorance is the condition of being unaware, uninformed, uneducated, and/or unsuspecting about the project and stakeholder goals, directions, details, issues, and opportunities. Many stakeholders practice "rational ignorance," which is when the cost of educating oneself about the issue sufficiently to make an informed decision can outweigh any potential benefit one could reasonably expect to gain from that decision. Therefore, it would be irrational to waste time doing so.

Point 4: Abstinence

Abstinence, in the context of project management, is the act or practice of refraining from participation and contribution to the project. Nonparticipation means withdrawing from the activities of the project and from the project in general. It also means that the person or persons do not want to be identified with the project or its activities.

Point 5: Fraudulence

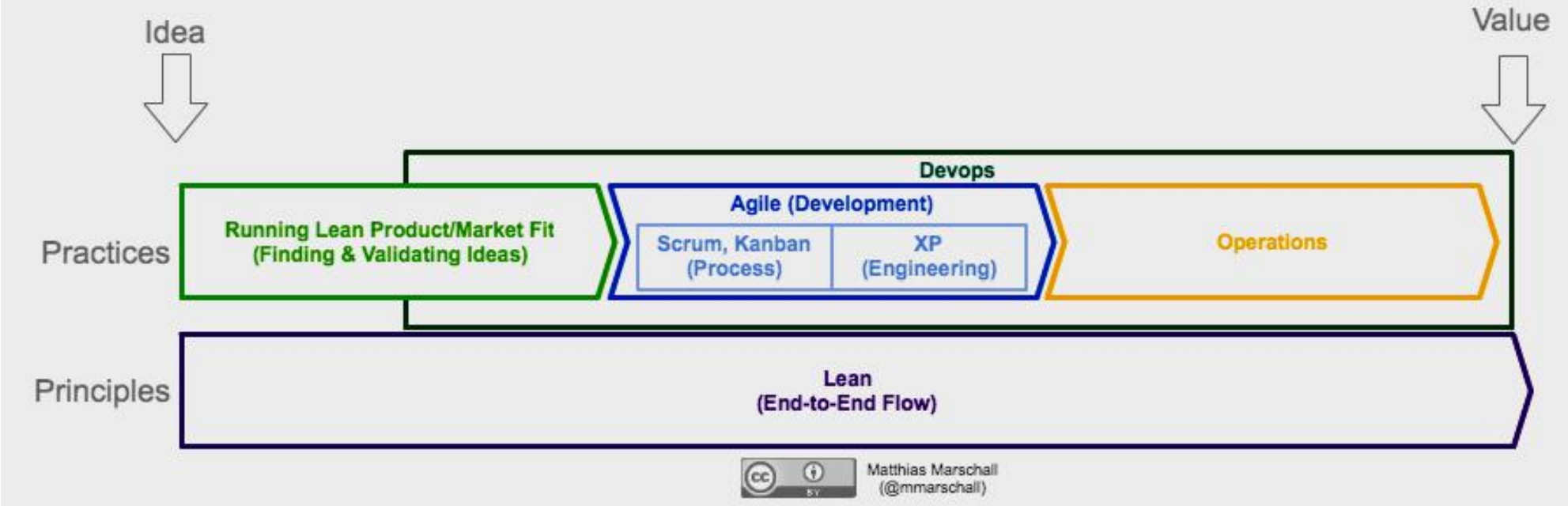
Fraudulence is an action intended to deceive; it is deliberate trickery intended to gain an advantage or to avoid confrontation. Fraudulence is being contrary to the truth or reality through speech or action. It is falsely portraying true intentions by deceptive words or action. Fraudulence, in the project management sense, is lying to oneself.

Watts Humphrey's Requirements Uncertainty Principle

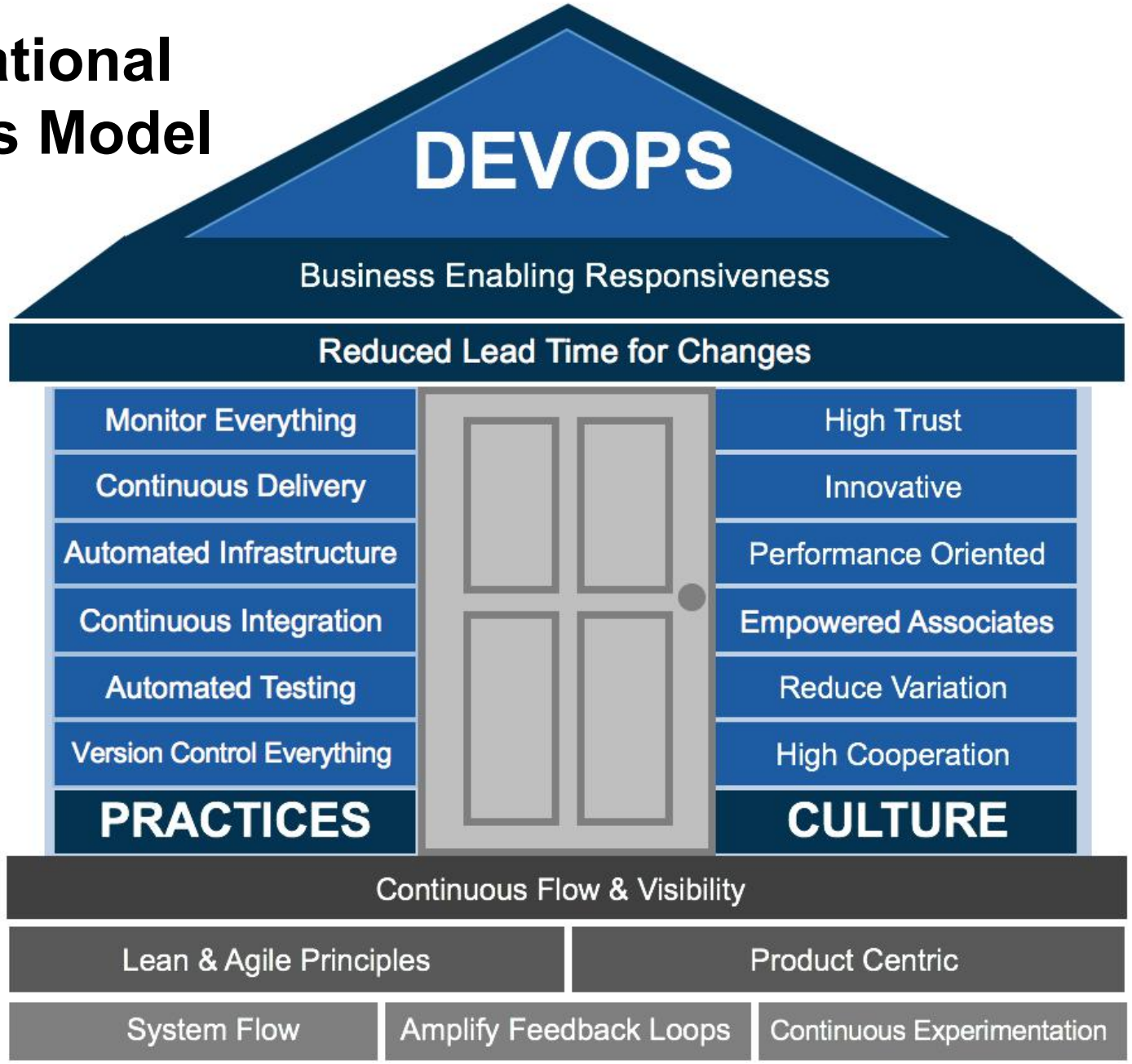
For a new software system, the requirements will not be completely known until after the users have used it.

- Agile addresses Humphrey's Requirements Uncertainty Principle by:
  - Capturing what users want in user stories
  - At the time of development, collaborating face-to-face and through whatever documentation is required to fully understand what is to be developed
  - Designing & developing features to address the requirement(s)
  - Then immediately thereafter providing what's been developed to the user(s) so the requirements can thus be fully known
  - Repeat

Relationship of Lean, Agile, and DevOps

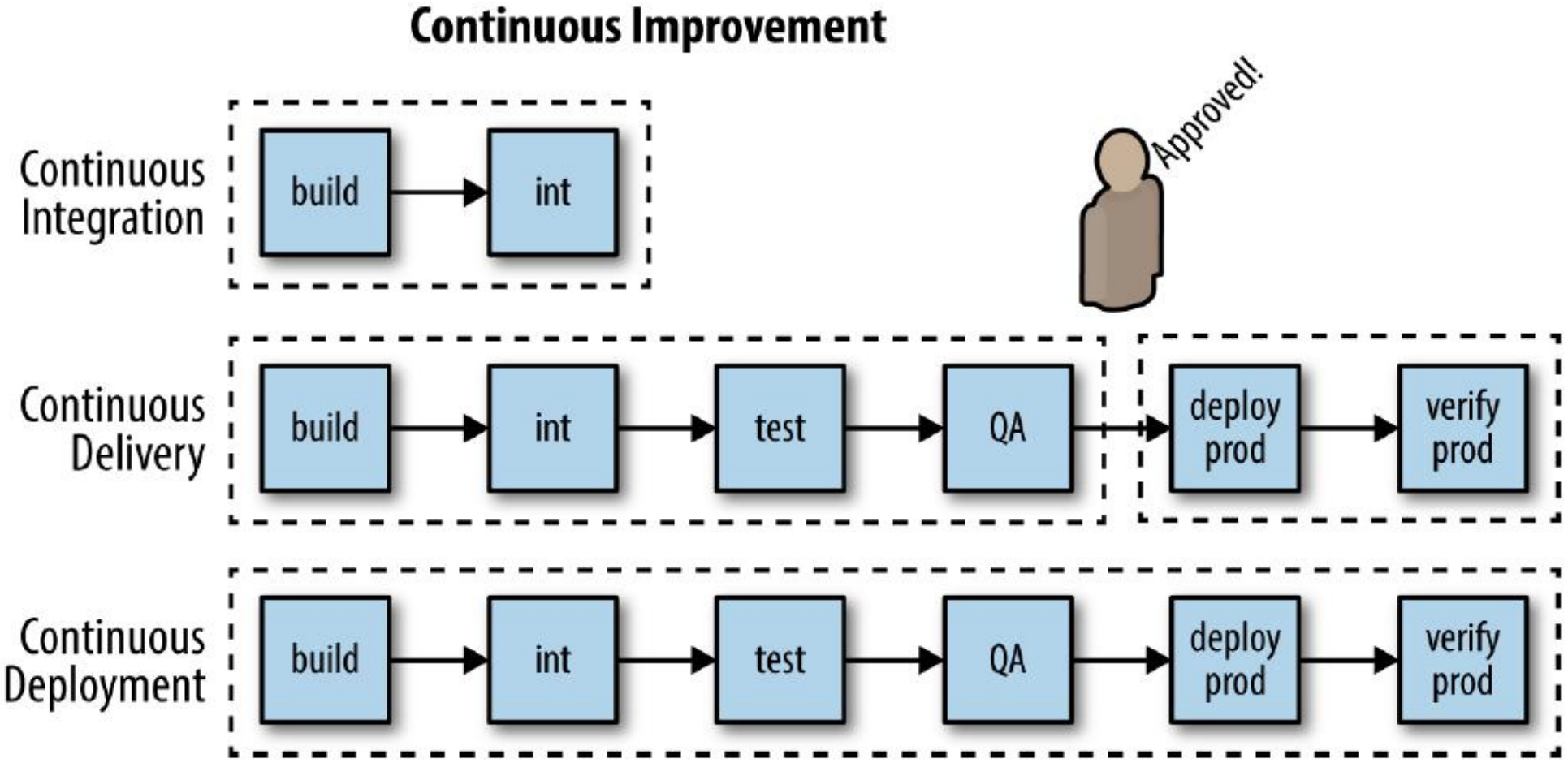


Foundational DevOps Model



Source: Nationwide Insurance, USA

Continuous Delivery vs. Deployment



Difference between CI and CD

DevOps vs Agile

What's the difference?

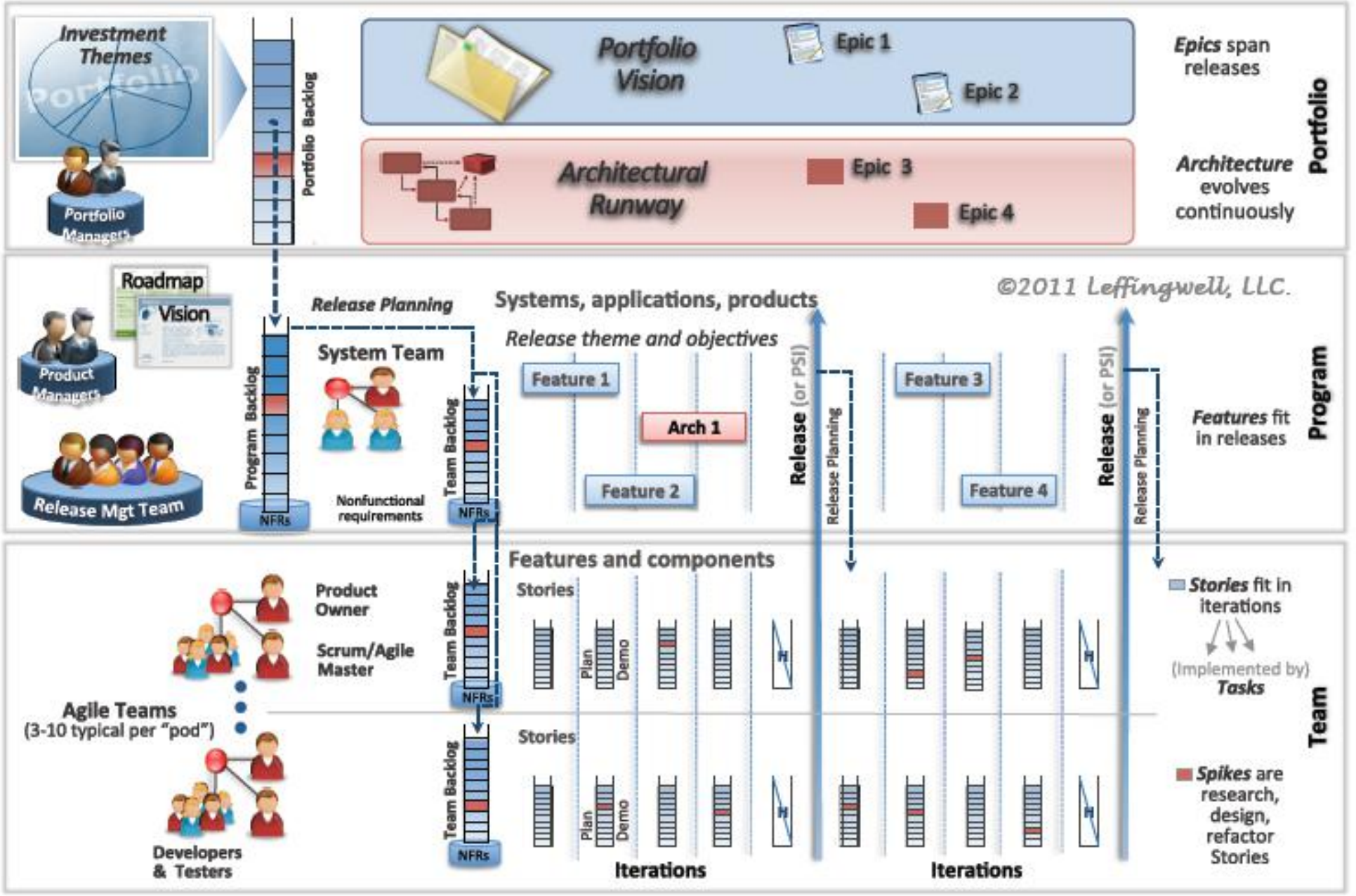
DevOps	Agile
A technical practice to achieve end to end solution	It is more about processes like scrum methodology and sprints
Operations as deployment is just a small part of ops. Other being configuration management etc.	It is the link between Design Thinking and DevOps
Aligns Ops with stability, such as scripting deployment and standardizing app builds	Aligns development with business priorities, thus focusing on speeding up the development process
DevOps helps to bridge the line between several cross functional teams (Development, QA, Monitoring, Deployment, Infrastructure etc.) by simplifying the process	In agile, the gap between these teams are bridged in daily scrum calls where all roadblocks are discussed and resolved with agile scrum methodologies.
Aims at faster deliveries through automation via various applications and tools like Docker, Chef, Jira, Configuration Management and Splunk. These tools are used for automating deployment and tracking bugs in real-time for quick fix.	Aims at streamlined delivery via tools like Kanban Board, Scrum, Extreme Programming (XP), Test Driven Development (TDD), Feature Driven Development (FDD), and Lean. These practices help in project management, planning and collaboration.

https://dzone.com/articles/is-devops-agile

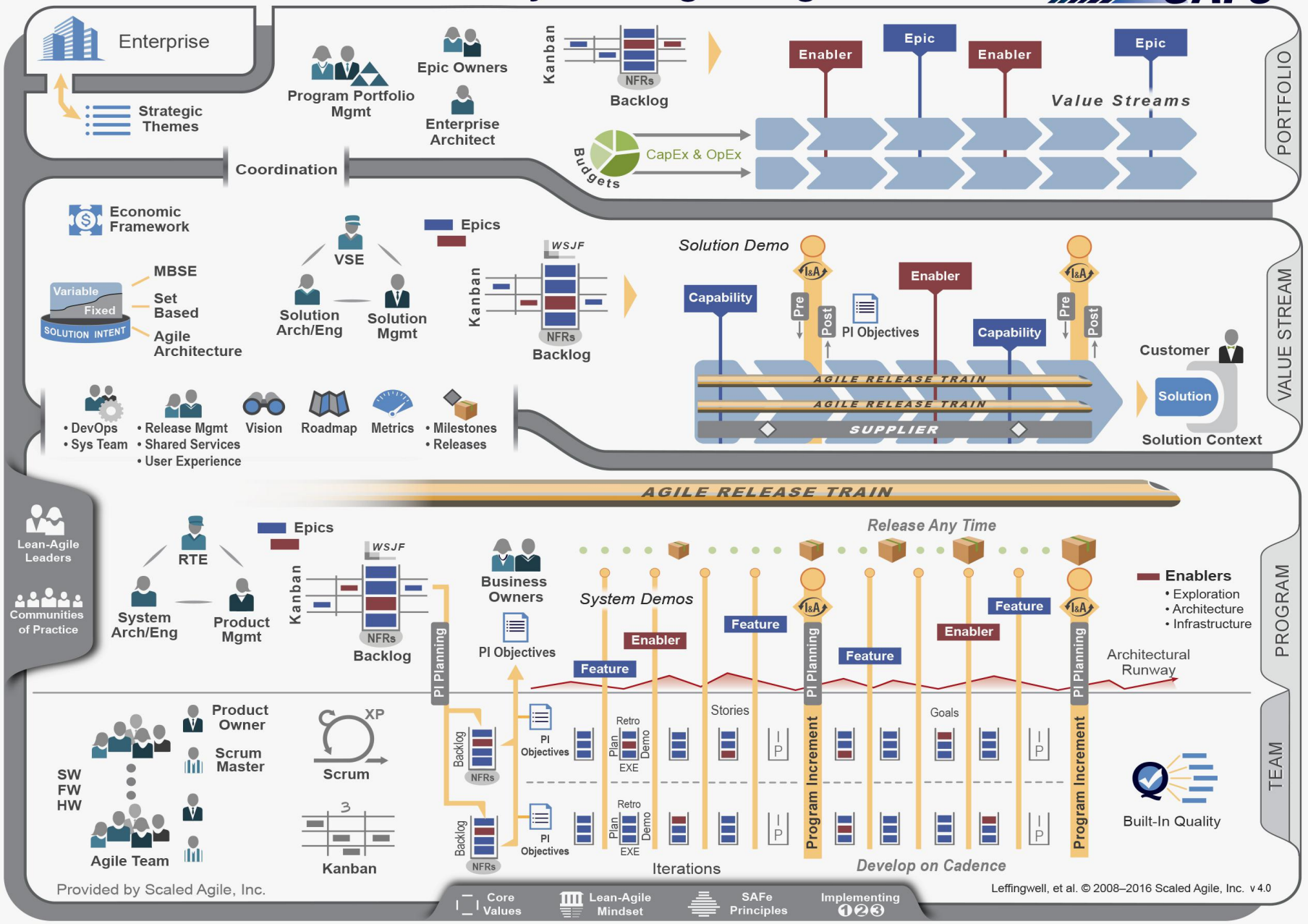
"Another flaw in the human character is that everybody wants to build and nobody wants to do maintenance."

- Kurt Vonnegut, Jr.
American novelist

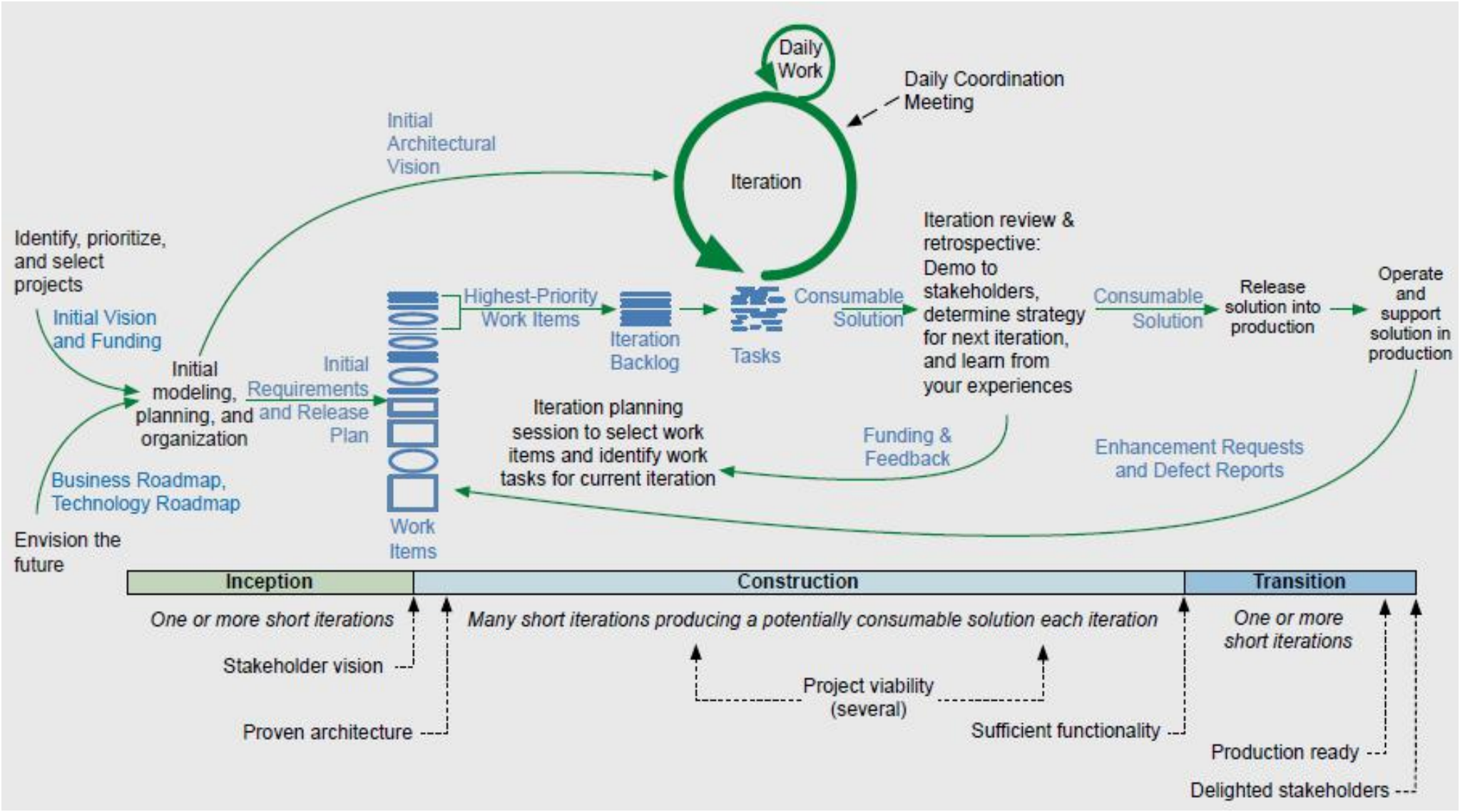
A Scaled Agile Framework



SAFe® 4.0 for Lean Software and Systems Engineering



The Disciplined Agile Delivery Lifecycle



From: DisciplinedAgileConsortium.org

Beyond Budgeting
- the adaptive management model

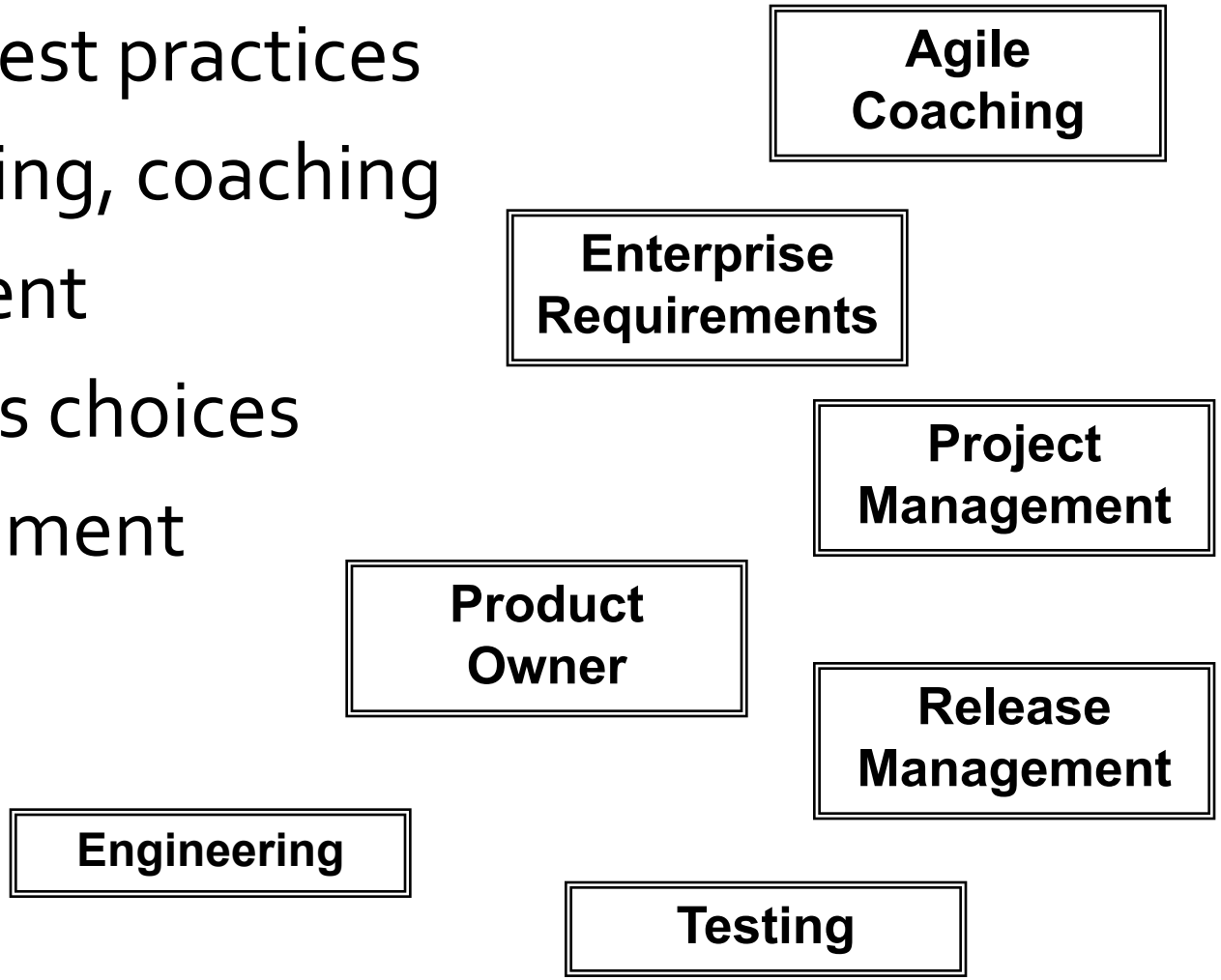
Leadership principles
1. **Purpose** - Engage and inspire people around bold and noble causes; ***not*** *around short-term financial targets*
2. **Values** - Govern through shared values and sound judgement; ***not*** *through detailed rules and regulations*
3. **Transparency** - Make information open for self-regulation, innovation, learning and control; ***don't*** *restrict it*
4. **Organisation** – Cultivate a strong sense of belonging and organise around accountable teams; ***avoid*** *hierarchical control and bureaucracy*
5. **Autonomy** - Trust people with freedom to act; ***don't*** *punish everyone if someone should abuse it*
6. **Customers** - Connect everyone's work with customer needs; ***avoid*** *conflicts of interest*

Management processes
7. **Rhythm** - Organise management processes dynamically around business rhythms and events; ***not*** *around the calendar year only*
8. **Targets** - Set directional, ambitious and relative goals; ***avoid*** *fixed and cascaded targets*
9. **Plans and forecasts** - Make planning and forecasting lean and unbiased processes; ***not*** *rigid and political exercises*
10. **Resource allocation** - Foster a cost conscious mind-set and make resources available as needed; ***not*** *through detailed annual budget allocations*
11. **Performance evaluation** - Evaluate performance holistically and with peer feedback for learning and development; ***not*** *based on measurement only and* ***not*** *for rewards only*
12. **Rewards** - Reward shared success against competition; ***not*** *against fixed performance contracts*

https://bbrt.org/the-beyond-budgeting-principles/

Communities of Practice

- Standards and best practices
- Knowledge sharing, coaching
- Skills development
- Technology/tools choices
- Process improvement
- Innovation



Center of Excellence

- A team of people that promote collaboration and using best practices around a specific focus area to drive business results.
- This team could be staffed with full- or part-time members.

Center of Excellence

- Responsibilities: CoEs should serve five basic needs:
  - **Support**: For their area of focus, CoE's should offer support to the business lines. This may be through services needed, or providing subject matter experts.
  - **Guidance**: Standards, methodologies, tools and knowledge repositories are typical approaches to filling this need.
  - **Shared Learning**: Training and certifications, skill assessments, team building and formalized roles are all ways to encourage shared learning.
  - **Measurements**: CoEs should be able to demonstrate they are delivering the valued results that justified their creation through the use of output metrics.
  - **Governance**: Allocating limited resources (money, people, etc.) across all their possible use is an important function of CoEs. They should ensure organizations invest in the most valuable projects and create economies of scale for their service offering. In addition, coordination across other corporate interests is needed to enable the CoE to deliver value.

Center of Excellence

- Common examples of CoE's are:
  - **Process**: this is the most strategic of all CoEs since all businesses are made up of processes
  - **Project Management Office (PMO)**: Many fail to think of themselves as a CoE and act as little more than a governance body
  - **Quality Assurance**: Whether for new product or software development the complexity of the roles, tools and techniques needed for quality often get formalized into a CoE
  - **Business Analysis (RMO)**: Some organizations have embraced the idea that getting business requirements, especially around software development, is a problem best addressed by a CoE

Center of Excellence

- **Communications**: Corporate communications, employee and customer relations are activities that are often supported by a centralized support process or function. At a basic level, their role is to support the line business around this focus area
- **Risk and Compliance**: Many organizations have created this capability without formally calling it a CoE. Insurance and financial institutions without exception will have this function. They almost always have veto power on changes to business processes or external communications. In ideal cases, they will help deploy standards and facilitate understanding throughout the organization.
- **Human Resources**: Another “function” or support process many businesses have embraced at a strategic level that meets the definition of a CoE.

Product Vision

- The product vision acts as the boundaries of the project in which the iterative, incremental work takes place.
- The product vision should answer the following three questions:
  1. What describes the product?
  2. Why is the product useful?
  3. What features will attract customers to this product?

Product Vision

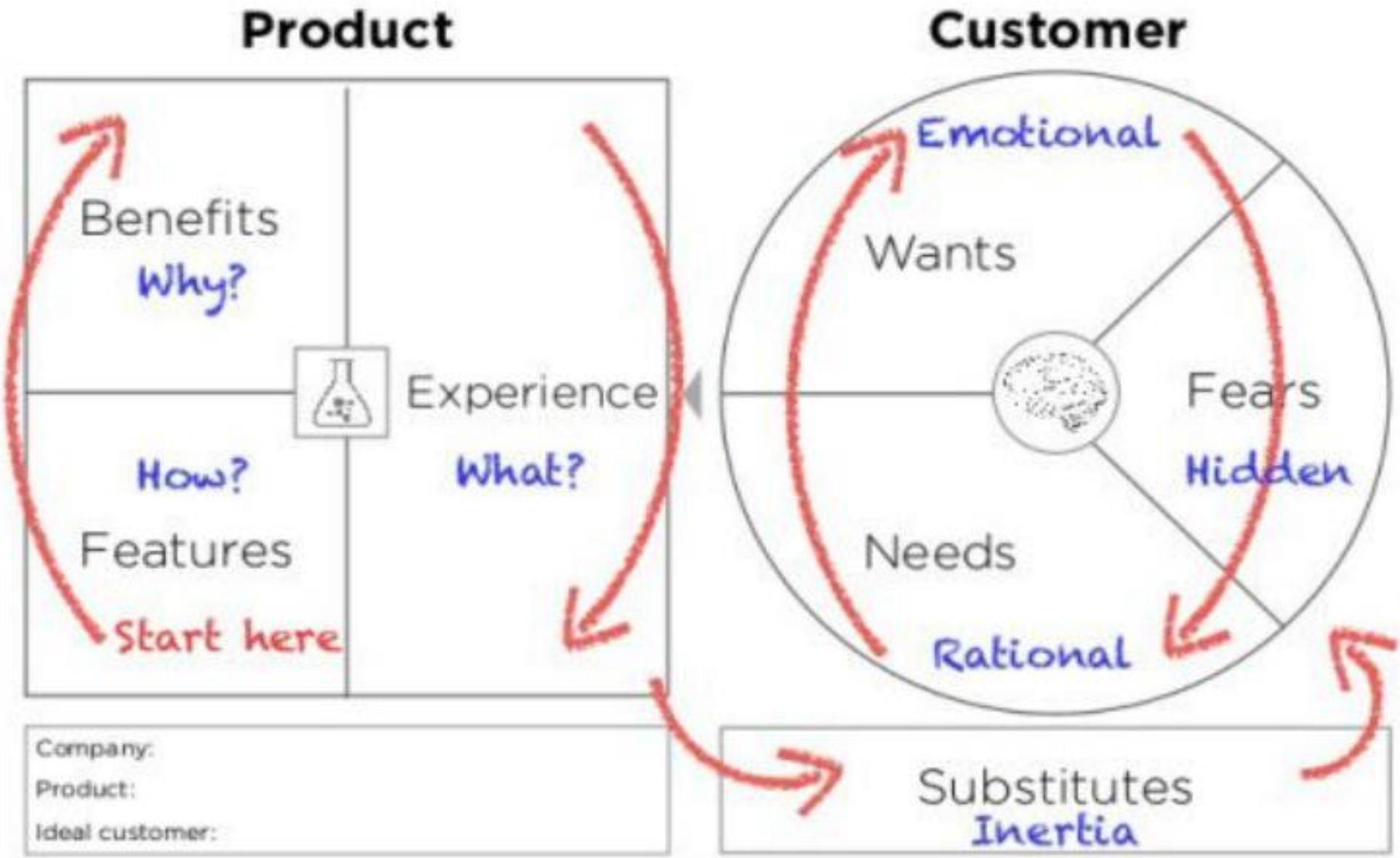
- Example of a product vision[1]:
  "For [target customer] who [statement of need] the [product name] is a [product category] that [product key benefit or compelling reason to buy]

[1]Geoffrey A. Moore, Crossing the Chasm, Marketing and Selling High-Tech Products to Mainstream Customer (revised edition), HarperCollins Publishers, New York, 1999

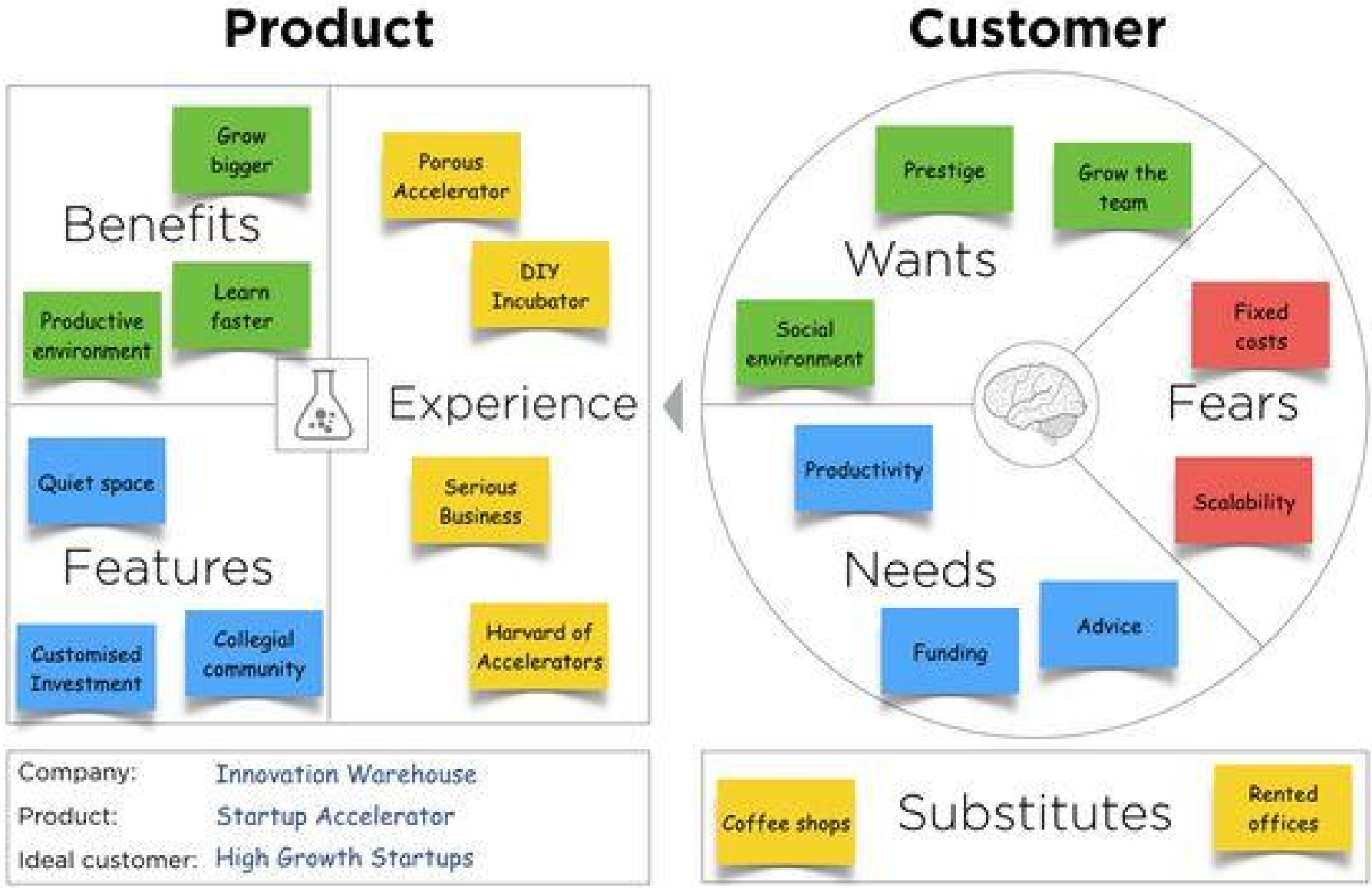
Product Vision Example

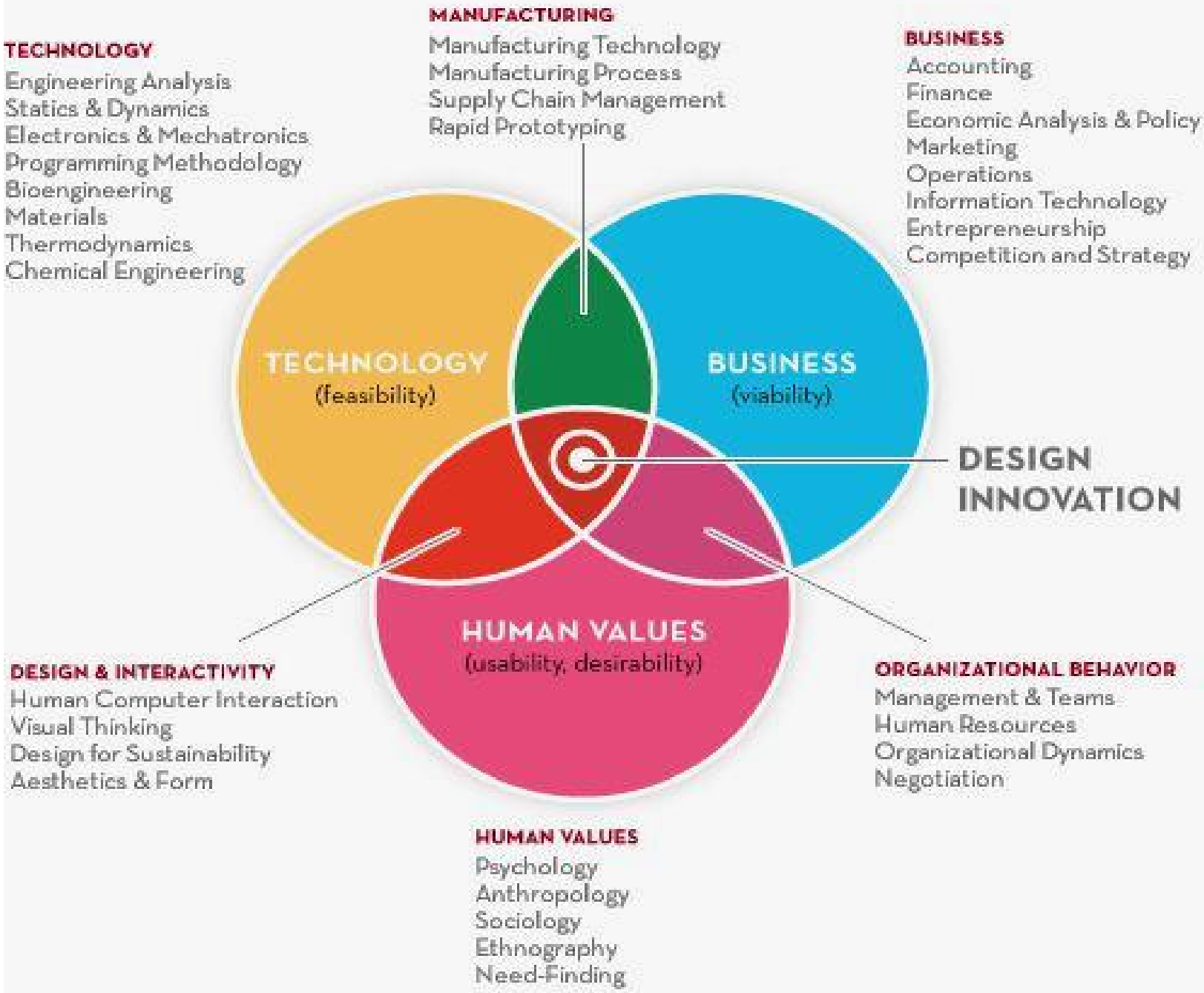
- FOR our large and mid-sized customers
- WHO need inventory lookup and ordering
- THE Community Catalog IS A web-based service
- THAT provides product lookup, reviews, availability and bulk ordering from our domestic warehouses
- UNLIKE our existing e-catalog that can't be easily updated and customers can't collaborate with us or each other,
- OUR PRODUCT will cost 50% less to maintain over the next 3 years and improve customer Net Promoter Score by 2 points by doubling interaction with sales support

Value Proposition Canvas

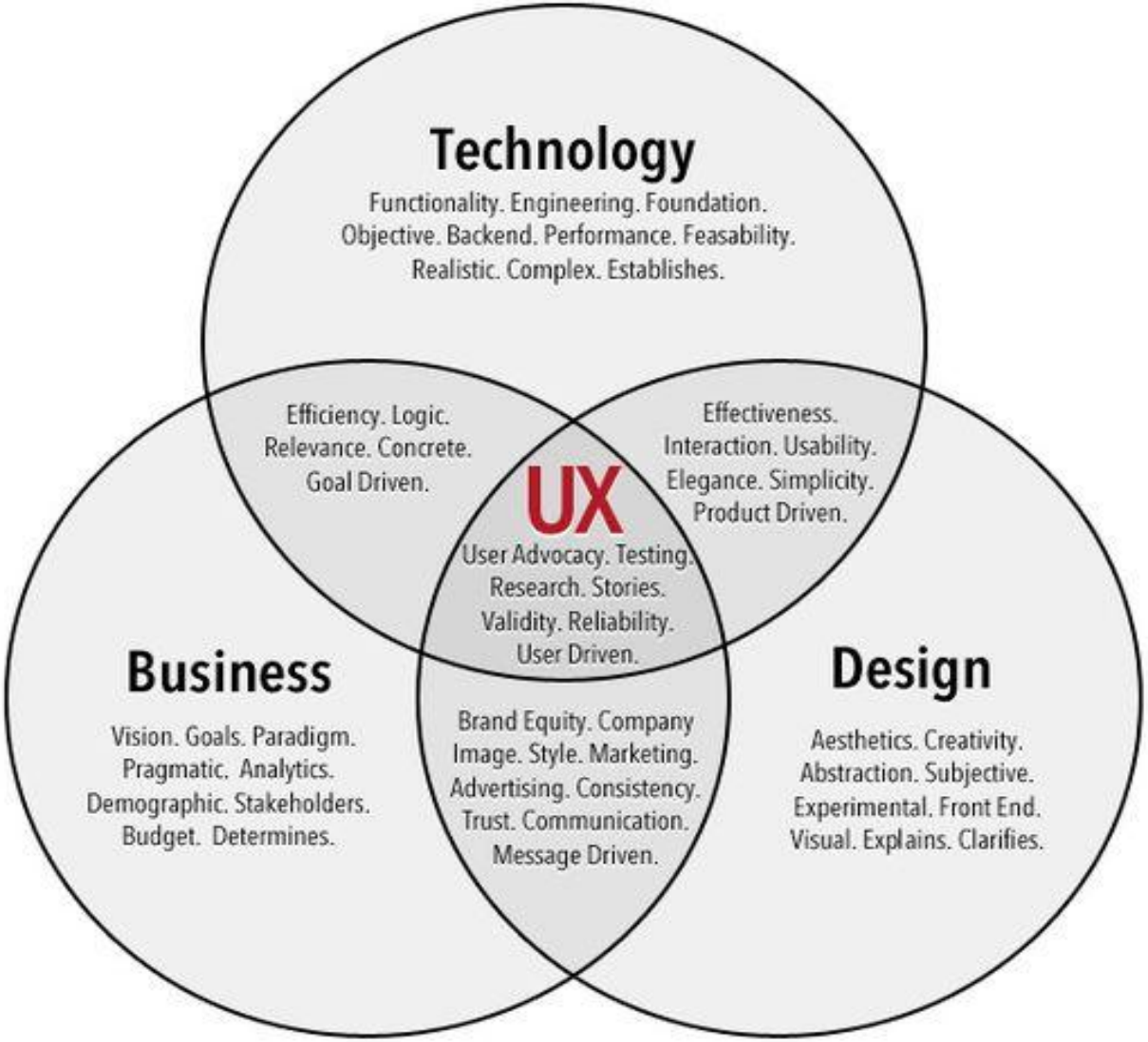


Value Proposition Canvas




TECHNOLOGY
Engineering Analysis
Statics & Dynamics
Electronics & Mechatronics
Programming Methodology
Bioengineering
Materials
Thermodynamics
Chemical Engineering
MANUFACTURING
Manufacturing Technology
Manufacturing Process
Supply Chain Management
Rapid Prototyping
BUSINESS
Accounting
Finance
Economic Analysis & Policy
Marketing
Operations
Information Technology
Entrepreneurship
Competition and Strategy
TECHNOLOGY
(feasibility)
BUSINESS
(viability)
DESIGN
INNOVATION
HUMAN VALUES
(usability, desirability)
DESIGN & INTERACTIVITY
Human Computer Interaction
Visual Thinking
Design for Sustainability
Aesthetics & Form
ORGANIZATIONAL BEHAVIOR
Management & Teams
Human Resources
Organizational Dynamics
Negotiation
HUMAN VALUES
Psychology
Anthropology
Sociology
Ethnography
Need-Finding

User Experience Design is the liaison between the three areas of technology, business, and design

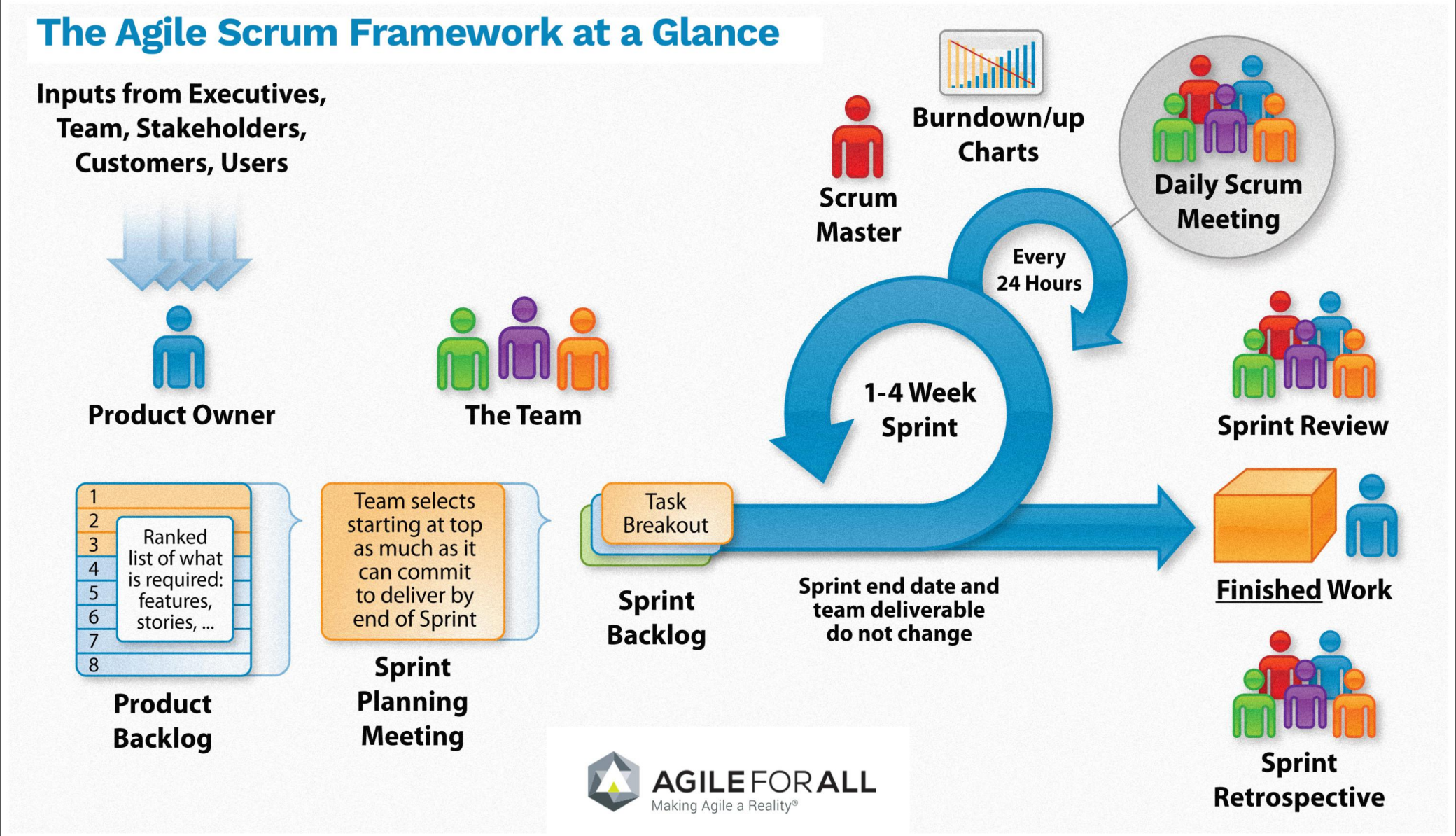


"Design is not just what it looks and what if feels like. Design is how it works."

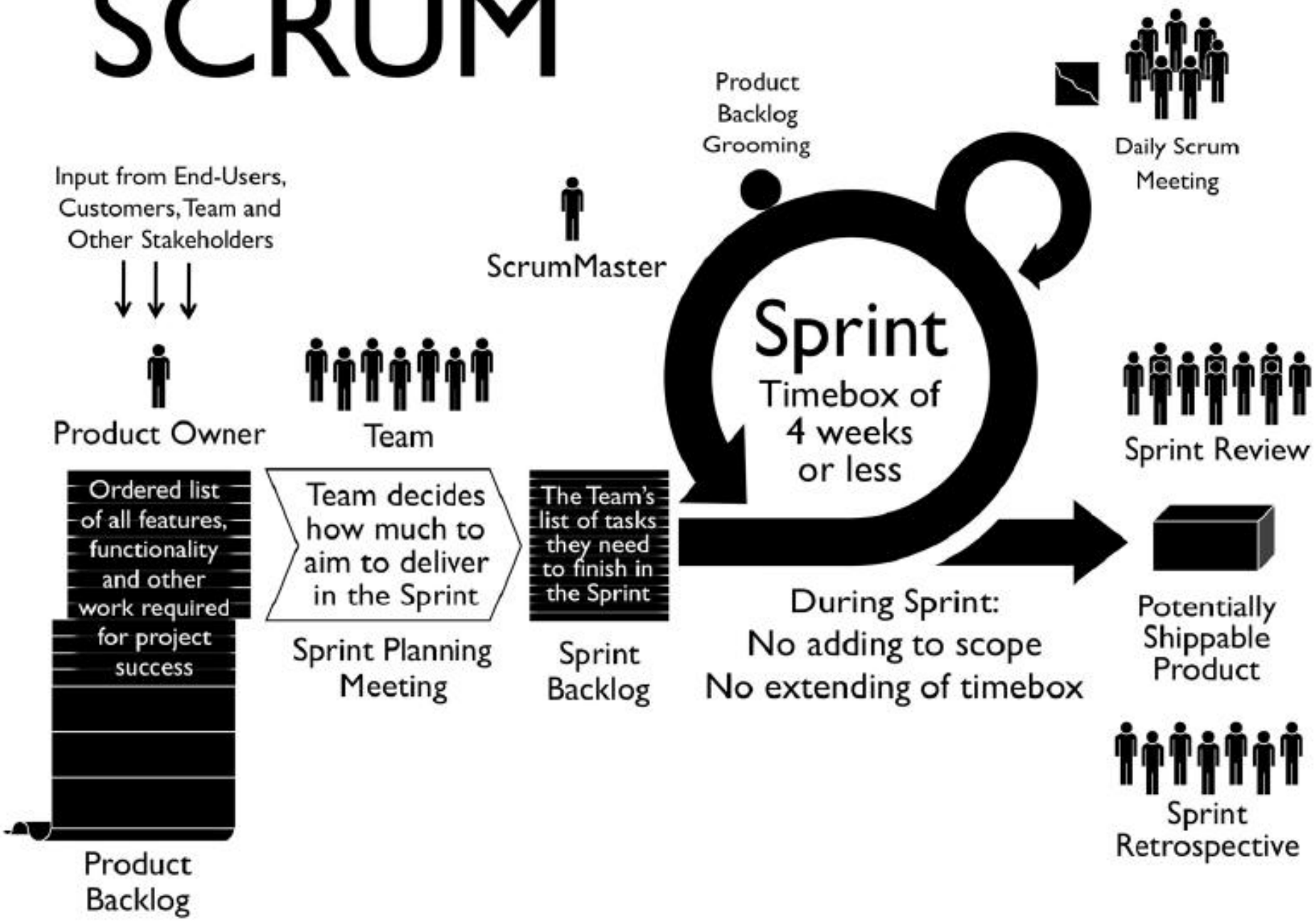
Steve Jobs

www.interaction-design.org

The Agile Scrum Framework at a Glance



SCRUM


Product Backlog Grooming
Daily Scrum Meeting
Input from End-Users, Customers, Team and Other Stakeholders
ScrumMaster
Sprint
Timebox of 4 weeks or less
Product Owner
Team
Sprint Review
Ordered list of all features, functionality and other work required for project success
Team decides how much to aim to deliver in the Sprint
The Team's list of tasks they need to finish in the Sprint
During Sprint:
No adding to scope
No extending of timebox
Potentially Shippable Product
Sprint Planning Meeting
Sprint Backlog
Sprint Retrospective
Product Backlog

SCRUM DEVELOPMENT PROCESS

SOFTHOUSE

1 Product Backlog

- List of requirements and issues
- Owned by Product Owner
- Anybody can add items
- Only Product Owner prioritizes

Backlog Estimation

- Product Owner describes new items
- All discuss
- Team estimates

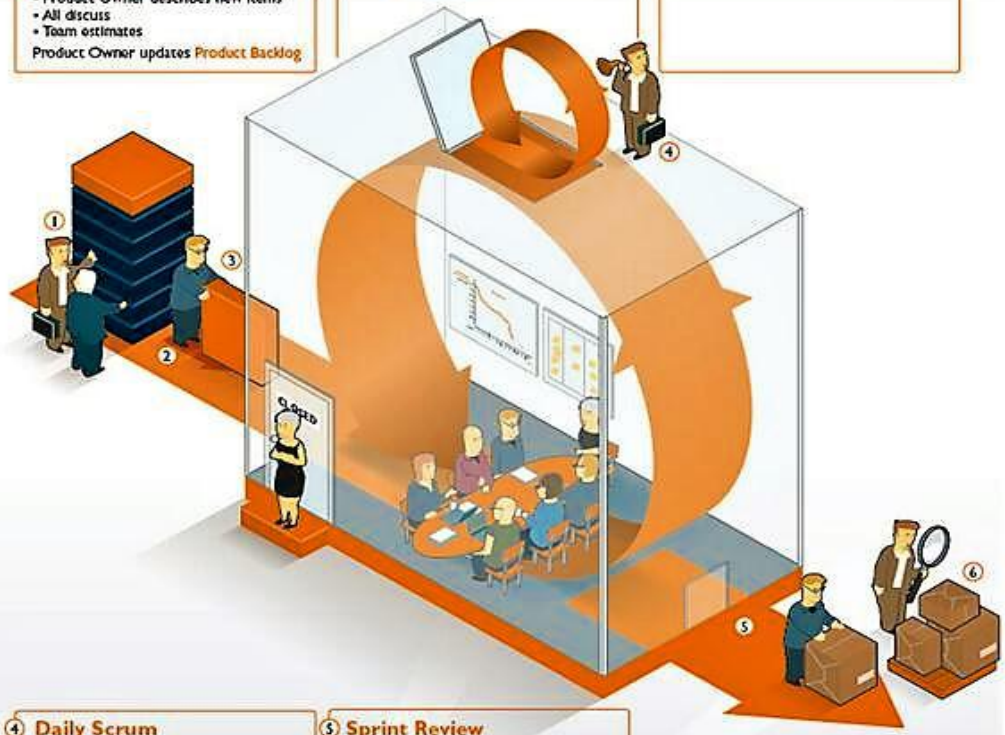
Product Owner updates Product Backlog

2 Sprint Planning

- Declare Sprint Goal
- Select highest priority items from Product Backlog that supports the Sprint Goal
- Team turns selected items into Sprint Backlog and estimates them

3 Sprint Backlog

- List of tasks
- Owned by Scrum Team
- Only Scrum Team modifies it



4 Daily Scrum

Same time every day

- What did you do yesterday?
- What will you do today?
- What's in your way?

Team updates Sprint Backlog
Scrum Master updates Impediment List & Sprint Burndown Chart

Sprint Burndown Chart
- How much work left until done

5 Sprint Review

Sprint Demonstration

- Team demos Increment
- All discuss
- Announce next Sprint Planning Meeting

Sprint Retrospect

Team discusses the sprint to answer the following:

- What should we keep doing?
- What should we stop doing?
- What should we start doing?

6 Increment

- Version of the product
- Potentially shippable (tested, documented etc)

Product Owner

Sets priorities and manages the product backlog

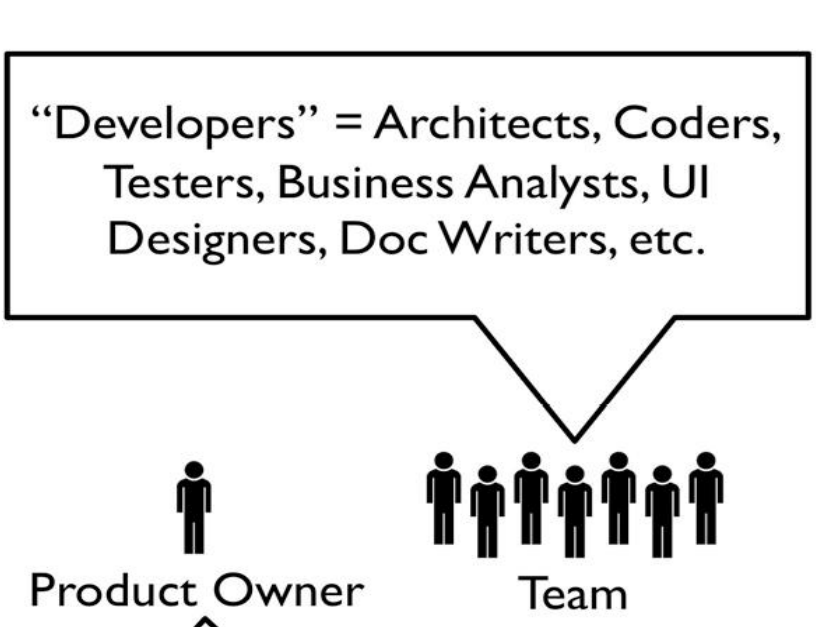
Scrum Master

Manages the process and removes impediments

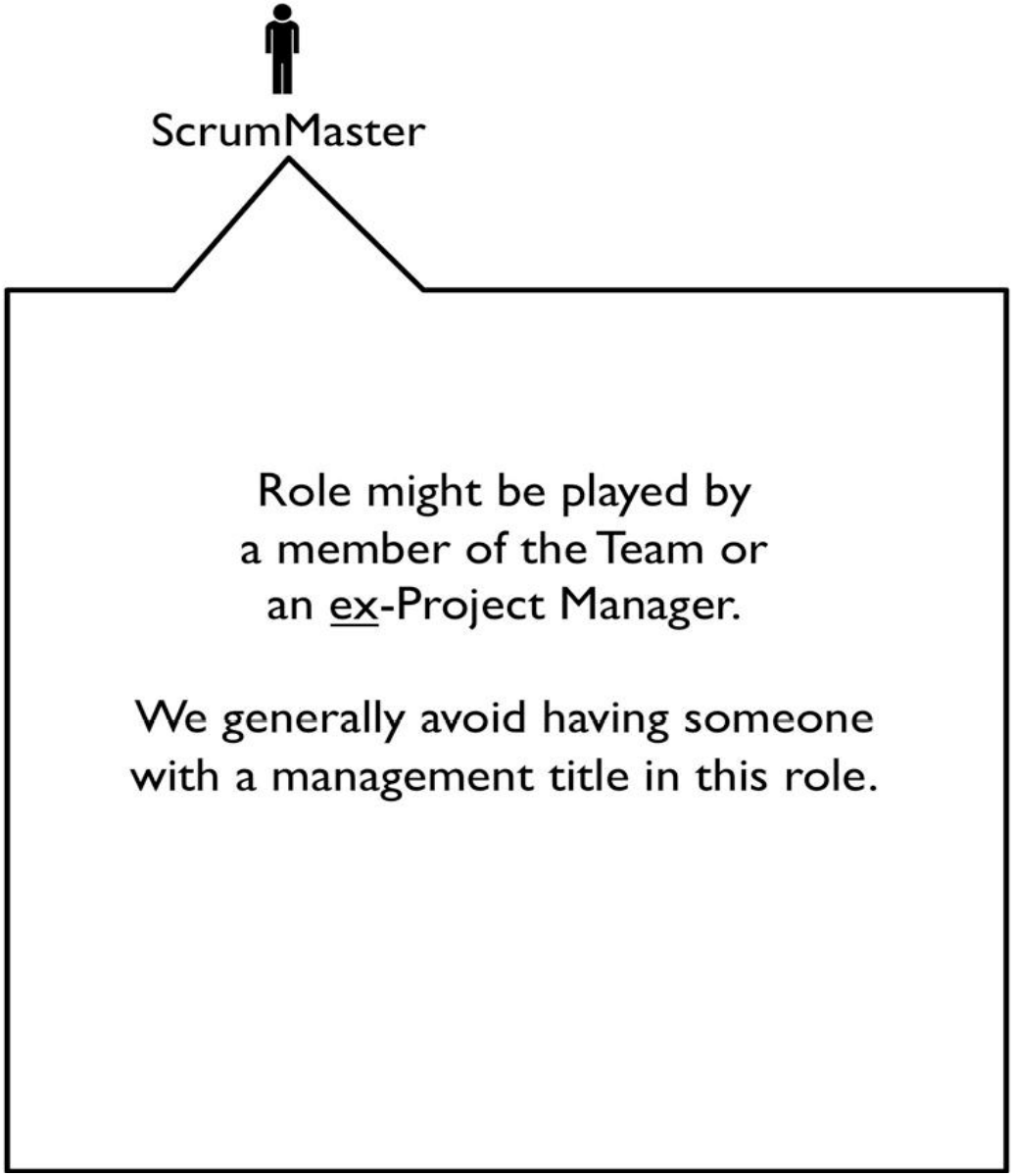
Scrum Team

Develops the product and self-organizes

SCRUM



Role might be played by Customer, Customer Representative, Product Manager, Product Marketing Manager, Program Manager, Project Manager.



"Coming together is a beginning;
keeping together is progress;
working together is success."

Henry Ford

The Definition of Done

- D.o.D. defines what it means for a Product Backlog Item to be "Done" (=potentially shippable) at the end of a Sprint

Our Team's Definition of Done

Product Backlog Item is done at end of Sprint if:

- **Code complete**
- **Code reviewed**
- **Unit Tested**
- **Integration Tested**
- **Acceptance Tested**
- **System Docs and User Docs updated**
- **No Priority 1 or Priority 2 defects remaining**

- Product Owner and Team create D.o.D. before first Sprint
- Can be changed, based on Product Owner and Team decision
- There may be Product Backlog Items that are outside the D.o.D.
  - For example, "Investigate Performance Problem" – needs its own D.o.D.

Definition of Done by the user

- ✓ Code produced
- ✓ Unit tests written and passed
- ✓ Builds without errors
- ✓ Code checked in source control
- ✓ Peer reviewed
- ✓ Deployed to system test environment and passed system tests
- ✓ Any build / deployment / configuration changes documented and communicated
- ✓ Relevant documentation produced and/or updated

The Definition of Done (DoD) in an agile project is a checklist of all the work that must be completed before the story can be considered complete, incoporating risk, compliance, and information and data privacy considerations.

It typically includes:

- Functionality coded and delivered.
- Functionality tested with all high severity defects resolved and plans in place to resolve low severity defects
- All nonfunctional requirements such as performance, scalability and security tested and validated
- Required documentation and audit trail created

Team "Done" List

...With a Story

- All Code (Test and Mainline) Checked in
- All Unit Tests Passing
- All Acceptance Tests Identified, Written & Passing
- Help File Auto Generated
- Functional Tests Passing

...With a Sprint

All Story Criteria, Plus...

- Product Backup Updated
- Performance Testing
- Package, Class & Architecture Diagrams Updated
- All Bugs Closed or Postponed
- Code Coverage for all Unit Tests at 80% +

...Release to INT

All Sprint Criteria, Plus...

- Installation Packages Created
- MOM Packages Created
- Operations Guide Updated
- Troubleshooting Guides Updated
- Disaster Recovery Plan Updated
- All Test Suites Passing

...Release to Prod

All INT Criteria, Plus...

- Stress Testing
- Performance Tuning
- Network Diagram Updated
- Security Pass Validated
- Threat Modeling Pass Validated
- Disaster Recovery Plan Tested

DONE (for each level)

Project/Solution =

All defined work completed

All compliance approved

Comm plan executed

Training completed

Documentation completed

Rollout completed

Release +

Required iterations complete

Compliance tests completed

UAT/Performance/Security/DR Testing passed

Documentation updated

Comms plan updated

Rollout plan instigated

Iteration +

Kick off held

Backlog updated

Iteration plan generated

Stories elaborated, written, tested, documented

Documents commenced

Training plan commenced

Rollout plan commenced

Comms plan Commenced

Showcase delivered

Retro run

Story

story identified

Elaborated → Acceptance Criteria

Quality defined

Sys Test cases identified

Auto Tests built

Coded to standard

Unit tests produced

Unit tests passed

sys tests passed (M+A)

Test summary Report produced

No defects

Quality criteria met

Release note written

"How to" written

UAT passed

Translated

Installable

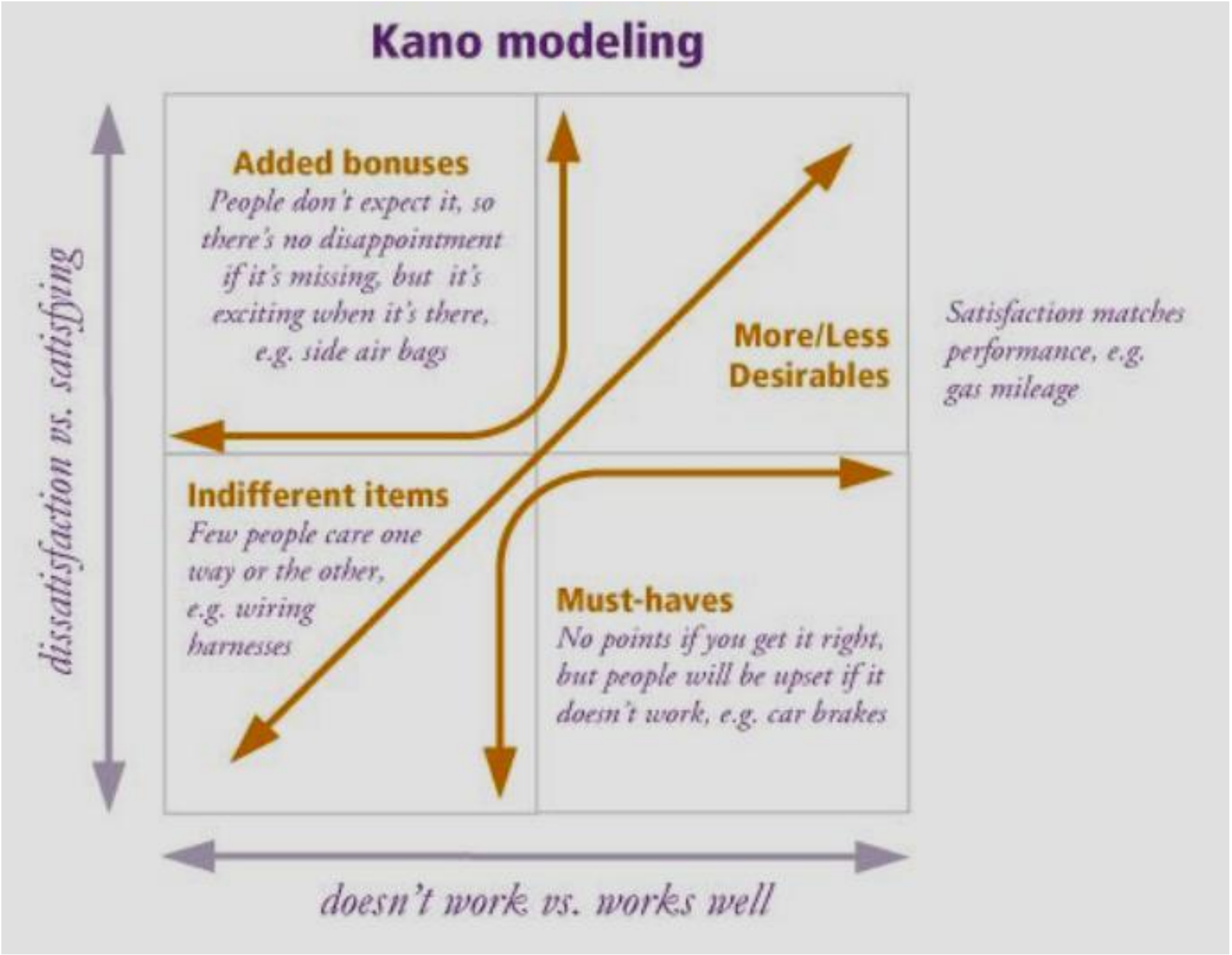
Explicit Hypothesis Test Card

I/We speculate that with <this capability>.
We would see <this result>.
These observations support our speculation
<raw observation 1>
<raw observation ...>
We should prioritize development of <this capability> because
<interpreted observation(s) listed as supporting indicators>
<...>

Scrum's Empiricism

Event	Inspection	Adaptation
Sprint Planning	• Product Backlog • (Commitment Retrospective) • (Definition of Done)	• Sprint Goal • Sprint Backlog • Forecast
Daily Scrum	• Sprint Progress • (Sprint Goal)	• Sprint Backlog • Daily Plan
Sprint Review	• Product Increment • Product Backlog • (Release Progress)	• Product Backlog
Sprint Retrospective	• Team & collaboration • Technology & engineering • Definition of Done	• Actionable improvements

Kano modeling



THE SERVANT AS LEADER

While servant leadership is a timeless concept, the phrase “servant leadership” was coined by Robert K. Greenleaf in The Servant as Leader, an essay that he first published in 1970. In that essay, Greenleaf said:

> *“The servant-leader is servant first… It begins with the natural feeling that one wants to serve, to serve first. Then conscious choice brings one to aspire to lead. That person is sharply different from one who is leader first, perhaps because of the need to assuage an unusual power drive or to acquire material possessions … The leader-first and the servant-first are two extreme types. Between them there are shadings and blends that are part of the infinite variety of human nature.*
>
> *“The difference manifests itself in the care taken by the servant-first to make sure that other people’s highest priority needs are being served. The best test, and difficult to administer, is: Do those served grow as persons? Do they, while being served, become healthier, wiser, freer, more autonomous, more likely themselves to become servants? And, what is the effect on the least privileged in society? Will they benefit or at least not be further deprived?“*

A servant-leader focuses primarily on the growth and well-being of people and the communities to which they belong. While traditional leadership generally involves the accumulation and exercise of power by one at the “top of the pyramid,” servant leadership is different. The servant-leader shares power, puts the needs of others first and helps people develop and perform as highly as possible.

https://www.greenleaf.org/what-is-servant-leadership/

A Day in the Life of a ScrumMaster

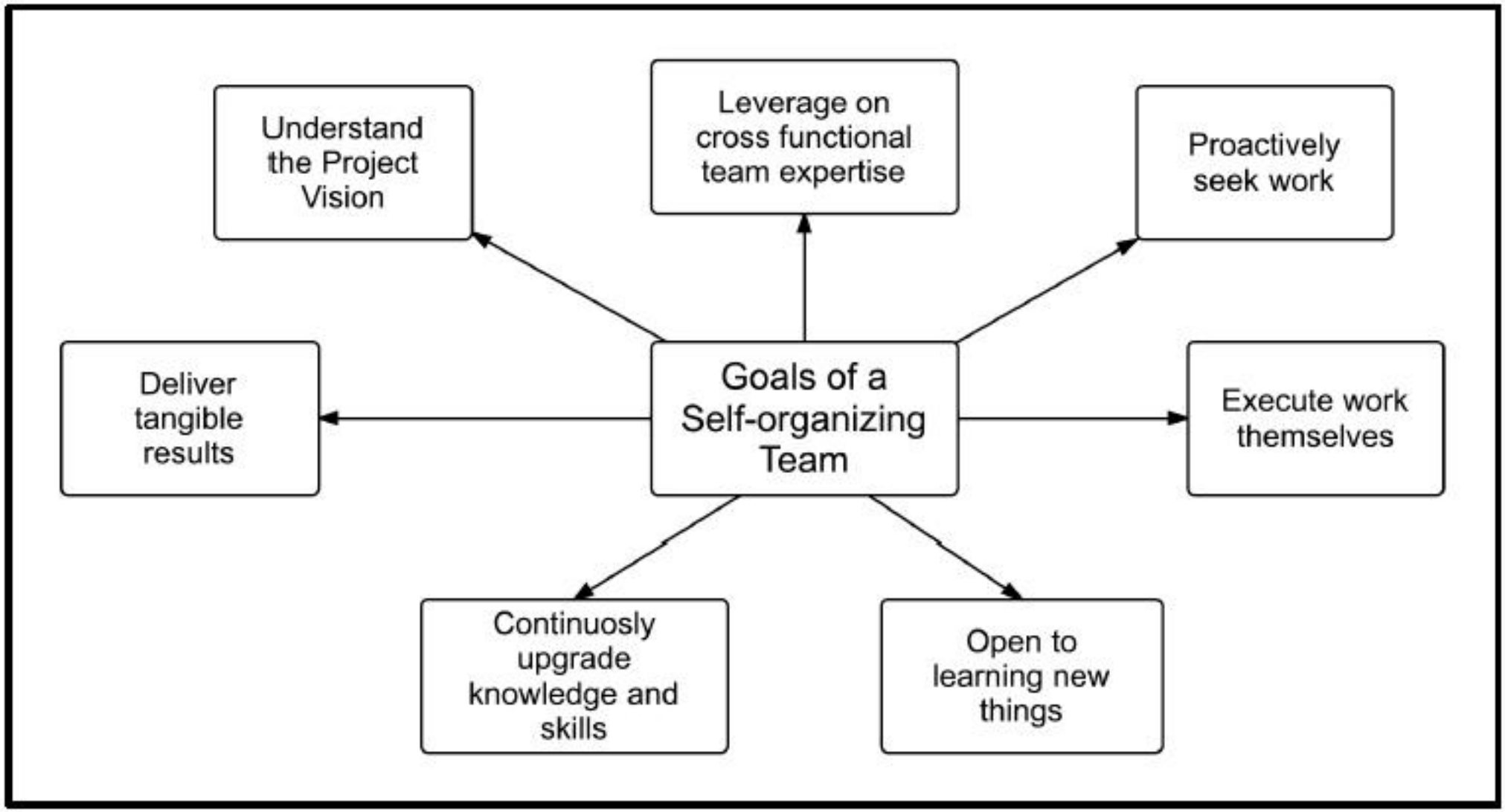
Morning

- Phone call to the Product Owner in the US to get a fast response to team's questions from the end of the previous day (30 mins)
- Update team-mates on the answers from PO (30 mins)
- Work on a coding task (90 mins)
- Hold the daily scrum meeting (15 mins)
- Continue to work on the coding task (60 mins)
- Facilitate between a design disagreement between 2 team-mates (20 mins)

Afternoon

- Meeting with purchasing department to request hardware the team will need in next Sprint (30 mins)
- Follow-up meeting with dept head to try to get budget for TDD training for team (20 mins)
- Take new team-mate to the admin group to sort out database access issues (20 mins)
- Facilitate a team discussion on how we're going to take action on the technical debt we're seeing (40 mins)
- Continue to work on the coding task (90 mins)
- Done for the day!

Goals of a Self-Organizing Team



FORMING

Behaviors

- The purpose and goals for the team are unclear.
- Members feel varying degrees of commitment.
- Members are cautious, don't initiate and avoid responsibility.
- Communication is low and a few members often dominate.
- Members are dependent on directive leadership.

Tasks

- Build a common purpose. Clearly establish the expectations of the customers or sponsors.
- Understand personal expectations and interests.
- Clarify accountability, recognition, and rewards.
- Assess resources; see who has what to contribute.
- Leader provides direction and drives the team process.

STORMING

Behaviors

- Differences and confusion arise over goals and roles.
- Struggles erupt over approaches, direction, and control.
- Team members react toward leadership with counterproductive behaviors.
- Team is uncertain about how to deal with issues openly.
- Team wrestles with issues of communication.
- Members act from an independent stance.

Tasks

- Involve everyone in the discussion.
- Inquire into differences; include all ideas and opinions.
- Seek further clarity about purpose and develop a common approach to meeting project objectives.
- Assess and test resource needs; make necessary adjustments.
- Define operational agreements (norms).
- Leader raises difficult issues and coaches team through struggles.

NORMING

Behaviors

- Team gains confidence, feels a sense of momentum.
- "What," "How," "Who," and "When" become clarified.
- Team develops agreements on approaches, goals, communication, and leadership roles.
- Team builds relationships with externals (customers, key stakeholders).
- Members begin to relate interdependently.

Tasks

- Develop processes for information sharing, feedback, and resource distribution.
- Have open forums on tasks and relationships, both internal and external.
- Build appropriate feedback loops with external relationships.
- Work toward consensus on overarching issues. Negotiate where appropriate.
- Leader uses a facilitative style to create the opportunity for others to lead.

PERFORMING

Behaviors

- Members take full responsibility for tasks and relationships.
- Team achieves effective and satisfying results.
- Team takes the initiative to continually assess external forces.
- Team facilitates itself easily through the various stages.
- Members work proactively for the benefit of the team.

Tasks

- Continuously seek to improve tasks and relationships.
- Assess and evaluate results against purpose and external forces.
- Celebrate successes—reward and recognize both team and individuals wins.
- Continuously test for better methods and approaches.
- Leader focuses on purpose, interdependent relationships, and conditions that shift the stages.

Tuckman's Model of
TEAM FORMATION

POTENTIAL WELL-FORMED TEAM

FORMING

A short-lived phase; the Team gets acquainted, learns roles and responsibilities.

STORMING

A challenging period; as the Team experiences disagreements, power struggles and conflict emerge.

NORMING

The Team discovers the light at the end of the tunnel, establishing guidelines and understanding process.

PERFORMING

The Team gets it; collaborating, anticipating and adjusting. Work is efficient, and the Team is motivated.

Teamwork: A Problem of Team Role Ambiguity

- There were four team members named Everybody, Somebody, Anybody and Nobody.
- There was an important job to do and Everybody was asked to do it.
- Everybody was sure Somebody would do it.
- Anybody could have done it, but Nobody did.
- Everybody was angry about that, because it was Somebody's job.
- Everybody thought Anybody could do it, but Nobody realized that Everybody wouldn't.
- In the end, Everybody blamed Somebody when Nobody did what Anybody could have done.

The Five Dysfunctions of a Team

Dysfunction #1: Absence of Trust
The fear of being vulnerable with team members prevents the building of trust within the team.

Dysfunction #2: Fear of Conflict
The desire to preserve artificial harmony stifles the occurrence of productive ideological conflict.

Dysfunction #3: Lack of Commitment
The lack of clarity or buy-in prevents team members from making decisions they will stick to.

Dysfunction #4: Avoidance of Accountability
The need to avoid interpersonal discomfort prevents team members from holding one another accountable.

Dysfunction #5: Inattention to Results
The pursuit of individual goals and personal status erodes the focus on collective success.

Source: The five dysfunctions of a team, a leadership fable, Patrick Lencioni

The Five Dysfunctions of a Team

- Dysfunction #1: Absence of Trust
  - The fear of being vulnerable with team members prevents the building of trust within the team.
- Dysfunction #2: Fear of Conflict
  - The desire to preserve artificial harmony stifles the occurrence of productive ideological conflict.
- Dysfunction #3: Lack of Commitment
  - The lack of clarity or buy-in prevents team members from making decisions they will stick to.
- Dysfunction #4: Avoidance of Accountability
  - The need to avoid interpersonal discomfort prevents team members from holding one another accountable.
- Dysfunction #5: Inattention to Results
  - The pursuit of individual goals and personal status erodes the focus on collective success.

Characteristics of High Performing Teams

- Teams willing to address the five dysfunctions can experience the following benefits. High performing, cohesive teams:
  - Are comfortable asking for help, admitting mistakes and limitations and take risks offering feedback
  - Tap into one another's skills and experiences
  - Avoid wasting time talking about the wrong issues and revisiting the same topics over and over again because of lack of buy-in
  - Make higher quality decisions and accomplish more in less time and fewer resources
  - Put critical topics on the table and have lively meetings
  - Align the team around common objectives
  - Retain star employees

6 CHARACTERISTICS OF A GOOD AGILE TEAM MEMBER

Having an agile team is integral to success of a project. Here are a few characteristics of the perfect agile team member. Learn how you can foster a great environment for your team.

Talk It Out

Communication is one of the most important aspects of teamwork. The best team members actively talk to each other and solve problems face to face.

Plan It Out

Being agile requires more planning. Project members should devote at least 20% of their work time to planning in order to run things smoothly.

Speak Up

A good agile team member isn't afraid to speak up in the hour of need. An environment where team members don't fear the blame for failure should be encouraged.

Listen Up

Great agile team members are good listeners and always value different opinions. They understand user's needs, groom the backlog, plan and evaluate feedback continuously.

Fully Engage

The best agile team members are insatiably curious. They always look for ways to hone their skills, learn new things and share their knowledge. Invest time and money in employee development.

Get Things Done

A great agile team member understands the long standing principle of getting things done. Involve them in the planning process and empower them to take ownership of the project.

Role of the BA in Agile Projects

There are a variety of ways a business analyst can be engaged on an agile project:

- The analyst might be the facilitator in more complex environments, bringing divergent business stakeholders together and helping them speak with a single voice so the project team is not confused by contradictory and conflicting perspectives.
- The analyst might act as the product owner/customer representative where they are empowered by the business to make decisions on product features and priority.
- The analyst could act as a surrogate product owner, in situations where the business product owner is not available.
- The analyst might act as the second in command to a business product owner with limited availability.

Role of the BA in Agile Projects

- The analyst could take the role of coach in an environment where the business product owner is competent and committed, but has limited IT project experience and the rest of the development team are lacking in domain knowledge.
- The analyst can play a central role in defining and communicating the acceptance criteria prior to development work commencing.
- The analyst can be involved in creating and executing acceptance tests.
- The analyst can ensure that the team remains focus on the business value of the project.
- The analyst can play a role in identifying important requirements that might not have been actively represented by stakeholders.

Example of one way to create a Sprint Backlog

				New Estimates of Effort Remaining at end of Day...					
Product Backlog Item	**Sprint Task**	**Volunteer**	**Initial Estimate of Effort**	**1**	**2**	**3**	**4**	**5**	**6**
As a buyer, I want to place a book in a shopping cart	modify database		5						
	create webpage (UI)		8						
	create webpage (Javascript logic)		13						
	write automated acceptance tests		13						
	update buyer help webpage		3						
	. . .								
Improve transaction processing performance	merge DCP code and complete layer-level tests		5						
	complete machine order for pRank		8						
	change DCP and reader to use pRank http API		13						

Product Backlog

As a frequent flyer, I want to…	3
As a frequent flyer, I want to…	5
As a frequent flyer, I want to…	5

Sprint Backlog

TASKS	HRS
Code the UI	8
Write test fixture	6
Code middle tier	12
Write test	5
Automate tests	4

Creating this list is sprint planning

Example of a Sprint Backlog

User Story	Tasks	Day 1	Day 2	Day 3	Day 4	Day 5	...
As a member, I can read profiles of other members so that I can find someone to date.	Code the ...	8	4	8	0		
	Design the ...	16	12	10	4		
	Meet with Mary about ...	8	16	16	11		
	Design the UI	12	6	0	0		
	Automate tests ...	4	4	1	0		
	Code the other ...	8	8	8	8		
As a member, I can update my billing information.	Update security tests	6	6	4	0		
	Design a solution to ...	12	6	0	0		
	Write test plan	8	8	4	0		
	Automate tests ...	12	12	10	6		
	Code the ...	8	8	8	4		

Technical Debt

- Refers to the shortcuts developers take & also to the many bad things that plague software systems including:
  - Unfit (bad) design: a design that once made sense but no longer does, given important changes to the business or technologies we now use
  - Defects: known problems in the software that we haven't yet invested time in removing
  - Insufficient test coverage: areas where we know we should do more testing but don't
  - Excessive manual testing: testing by hand when we really should have automated tests
  - Poor integration & release management: performing these activities in a manner that is time-consuming & error-prone
  - Lack of platform experience: having mainframe applications written in COBOL but don't have many COBOL programmers around

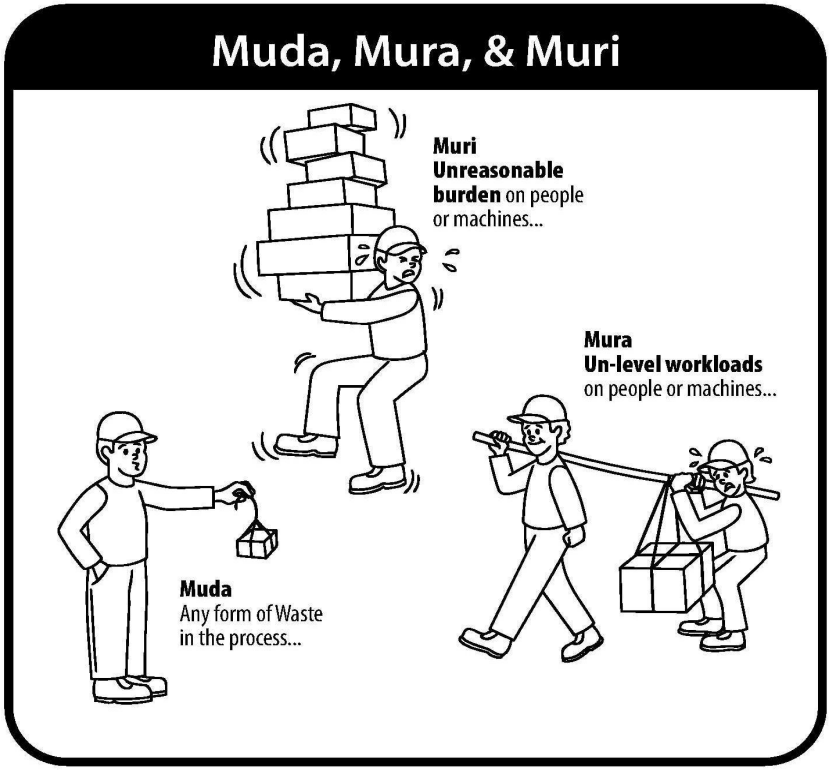
What is Smoke Testing?

- Is performed after software build to ascertain that the critical functionalities of the program is working fine.
- It is executed "before" any detailed functional or regression tests are executed on the software build.
- Purpose is to reject a badly broken application, so that the QA team does not waste time installing and testing the software application.
- In Smoke Testing, the test cases chosen cover the most important functionality or component of the system.
- Objective is not to perform exhaustive testing, but to verify that the critical functionalities of the system is working fine.
- E.g. a typical smoke test would be – Verify that the application launches successfully, Check that the GUI is responsive ... etc.

Lean Philosophy

- Muri (overburdening of people or equipment)
- Mura (unevenness in workload)
- Muda (waste or non-value adding activities)

- Muri creates mura leading to the inability to reduce muda.
- Thus, waste can be reduced through solving problems that involve imbalanced loading and overstraining of people



Muda

無駄

"Waste" or failures of people or processes to efficiently deliver product.

Mura

斑

“Unevenness”, or failures due to inconsistent or unpredictable outputs.

Muri

無理

“Overburden”, or failures of standardization to create efficient process.

Types of Wastes

- **Correction**
  - Corrections are and time you redo, rewrite, rework, repair, or scrap something. This can be as simple as rewriting a grocery list. Say you have a grocery list but you want to rearrange the items on it in the order you will encounter them in the store. Even though it will speed things for you shopping it had to be redone instead of thinking of making the list ordered in the first place. Redoing the list did not add any value to you; it took longer to write it a second time instead of doing it right the first time.
- **Overproduction**
  - Overproduction is when you make too much of something or you perform too much of a service for some one. Have you ever held a meeting and made copies for that meeting? Most people make a few extra, do you? That is overproduction they will end up in the trash. Or have you every as a question about something in a store and the salesman goes on an on answering your question when all you wanted was the simple answer? That salesman was overproducing.

Types of Wastes

- **Movement of material or information**
  - This type of waste is when you take any material for information and have to move it from one place to another. You may ship it or carry it your self but that movement does not create any value for the customer in fact it is lost time because it delays your product or service from getting to your customer.
- **Motion of employees**
  - This type of waste is when you or an operator has to get up and walk or travel to get something to do their job. Just like movement of materials and information, motion of the operator does not create value. In fact the “thing” in the process is not changing at all.

Types of Wastes

- **Waiting**
  - This type of waste is when you, other employees, customer, material, or equipment sits idle waiting.
- **Inventory or other resources**
  - This type of waste is not just supplies and materials on shelves but also any recourse your company has that is not being utilized. We normal see inventory as parts and supplies sitting on a shelf like boxes of cereal in the grocery store. But here inventory also include equipment that is standing idle or in storage.

Types of Wastes

- **Processes**
  - Doing more than required by the customer. This is a hard one to understand because sometimes doing more for free has a WOW factor for your customers. That is why it is important to know what is of value and what is not. You see sometime you do sometime more that you think the customer wants and they do not care. That is when it becomes a waste.
- **Confusion**
  - Missing or misinformation.
  - Confusing goals & metrics.
- **Underutilized human potential** (skills, talents, and creativity)

Type of Waste	Description
Defects	Any nonconformance that leads to redoing, reworking, recontacting, or reviewing. Examples include missing critical data on forms and not sending out a debit card in time.
Waiting	Any time during which work is not being performed on the customer request. Examples include waiting for approval and waiting for branch feedback.
Overproduction	Producing more than required or more than a process step has the capacity to handle, resulting in the building of inventory. An example is batch processing of applications.
Unnecessary transportation	Movement of files, data, or customer requests. With every movement, there is a risk of loss or delays in processing.
Inventory	Work-in-process, representing unrecognized potential revenue. An example is applications waiting for processing.
Overprocessing	Doing more than is required from a customer's perspective.
Motion	Movement to transport information or data. An example is extra steps taken by employees to accommodate an inefficient process layout.

Lean Principles

Specify Value

Define **value** from the customers perspective and express value in terms of a specific product or service

Map the Value Stream

Map all of the steps...value added and non-value added...that bring a product or service to the customer

Work to Perfection

The complete elimination of waste so all activities create value for the customer by breakthrough and **continuous improvement** projects

Establish Flow

The continuous **flow** of products, services and information from end to end through the process

Implement Pull

Nothing is done by the upstream process until the downstream customer signals the need, actual demand **pulls** product/service through the value stream

The Product Owner

Product Definition

Product Vision

Strategy

Competitive Landscape

Market Segmentation

User Personas

User Research

Design Sense

Analytics

Domain Knowledge

Marketing

Entrepreneurship

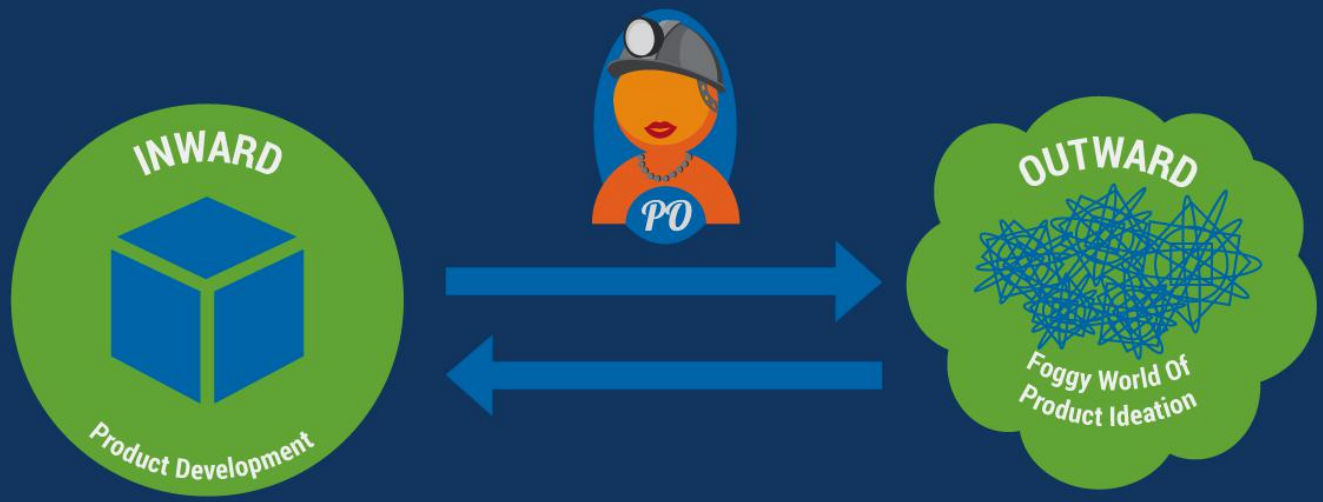
Leadership

Agile / Lean

Innovation

Curiosity, Passion, & Perserverence

The PRODUCT OWNER Dilemma



Have we built a viable thing? (Results/Reward/ROI)

Are we building the thing right? (Product Management)

What is the right thing? (Interactive Exploration)

The Product Owner's role is a balancing act between the inward and outward forces of product management. Learning to finesse that balance is the mark of a truly agile Product Owner. By applying Scrum to determine if the Team is building the right product at the right time for the right release, Product Owners can confidently navigate between the push of product development and the pull of product ideation to successfully leverage their Team's performance.

Top 7 Scrum Product Owner Responsibilities

1. Leads the development effort by conveying his or her vision to the team and outlining work in the product backlog.
2. Prioritizes work based on Business Value.
3. Negotiates work with the team.
4. Must remain available to the team to answer questions and deliver direction.
5. Must resist the temptation to micromanage.
6. Must resist “raiding the team’s spirit.”
7. Must not be afraid to make tough decisions.

Top 10 activities of a product owner:

1. Creates and MAINTAINS the Product Backlog
2. Prioritizes and sequences the Backlog according to business value or ROI
3. Assists with the elaboration of Epics, Themes and Features into user stories that are granular enough to be achieved in a single sprint
4. Conveys the Vision and Goals at the beginning of every Release and Sprint
5. Represents the customer, interfaces and engages the customer
6. Participates in the daily Scrums, Sprint Planning Meetings and Sprint Reviews and Retrospectives
7. Inspects the product progress at the end of every Sprint and has complete authority to accept or reject work done
8. Can change the course of the project at the end of every Sprint
9. Communicates status externally
10. Terminates a Sprint if it is determined that a drastic change in direction is required

Jack Milunsky (https://dzone.com/articles/product-owner-top-10)

A Day in the Life of a PRODUCT OWNER

1. Attend Daily Standup.
2. Sidebar with some devs to clarify a Story they're working on.
3. Get update from SM on resolved impediment with 2 pair programming developers.
4. Skype with a remote Stakeholder who was unable to attend Sprint Review.
5. Have lunch with Stakeholders to discuss their desirements for a future product release.
6. Begin prioritizing Backlog items in terms of their value to the organization.
7. Meet with SM to discuss findings from Retro and discuss how to better include remote Stakeholders in Review.
8. Update Information Radiators to track Velocity and progress of release.
9. Meet with Business Owner on status of next product release.
10. Grab an afternoon coffee with SM and discuss how to recognize and deal with technical debt that built up during the rush feature release.
11. Help SM extract Stories from the devOps Chore Epic.
12. Swarm with Team Members on results from release Performance Testing.

Attributes of Servant-Leaders

Functional Attributes		*Accompanying Attributes*	
1. Vision	6. Modeling	10. Communication	16. Persuasion
2. Honesty	7. Pioneering	11. Credibility	17. Listening
3. Integrity	8. Appreciation	12. Competence	18. Encouragement
4. Trust	9. Empowerment	13. Stewardship	19. Teaching
5. Service		14. Visibility	20. Delegation
		15. Influence	

Motivation in the New Era

Motivation 3.0

Motivation 3.0 moves the focus from the "*reward-punishment*" compliance approach of the 20th century to an "*engagement*" model that is better suited to workplaces of 21st Century economies.

Motivation 3.0 is a concept of Daniel Pink

Motivation 3.0

Autonomy
- a sense of self-direction
- having control over your work

Mastery
- getting better at something
- making progress

Purpose
- contributing to something bigger than yourself

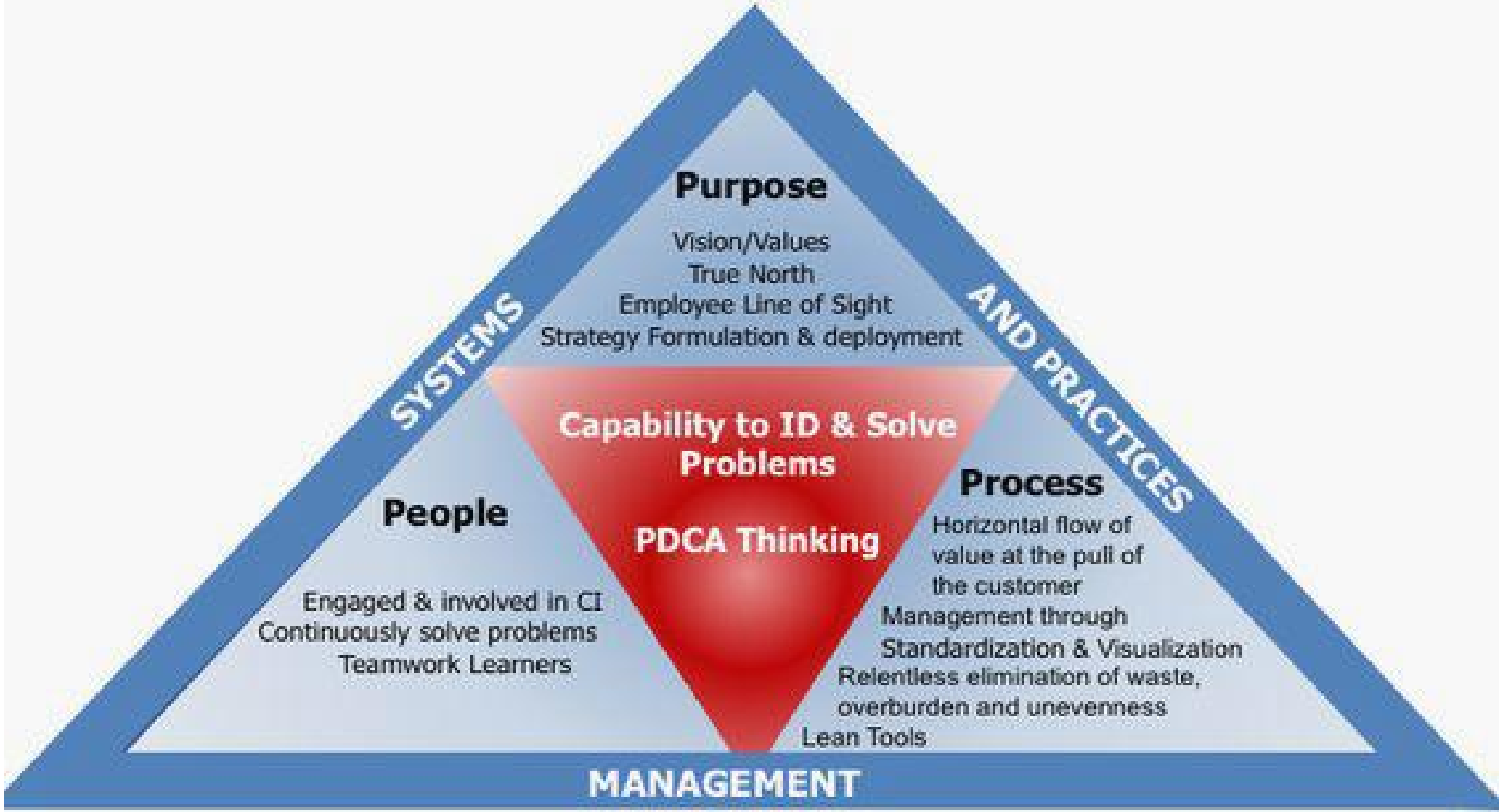
Motivation 3.0 is a concept of Daniel Pink

Based on the work of Daniel Pink

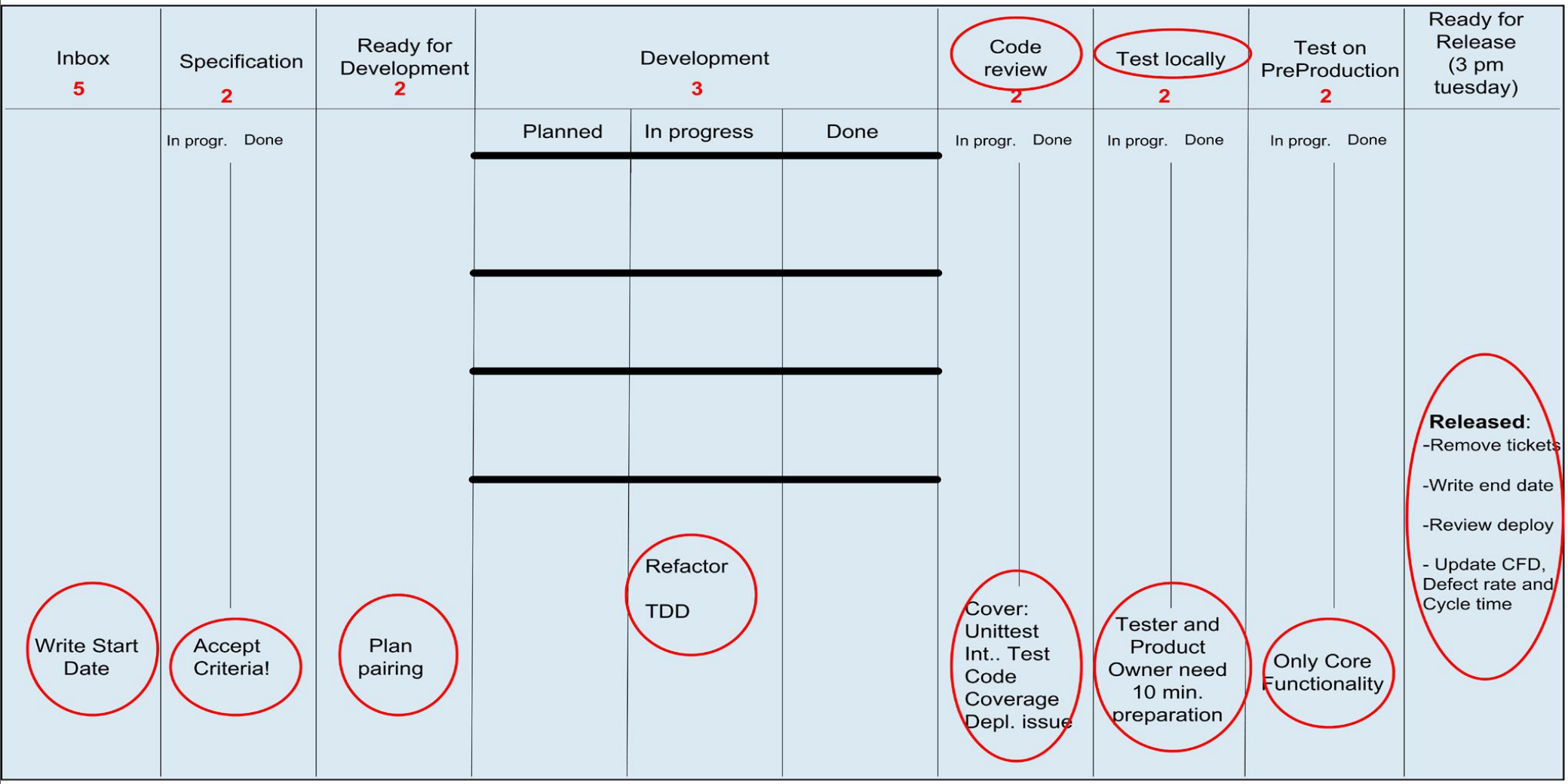
Three Lessons on Visual Management

- Make your work and the status of that work as visual as possible
- Design simple and inexpensive techniques to error-proof processes
- Catch mistakes and errors before they turn into customer defects

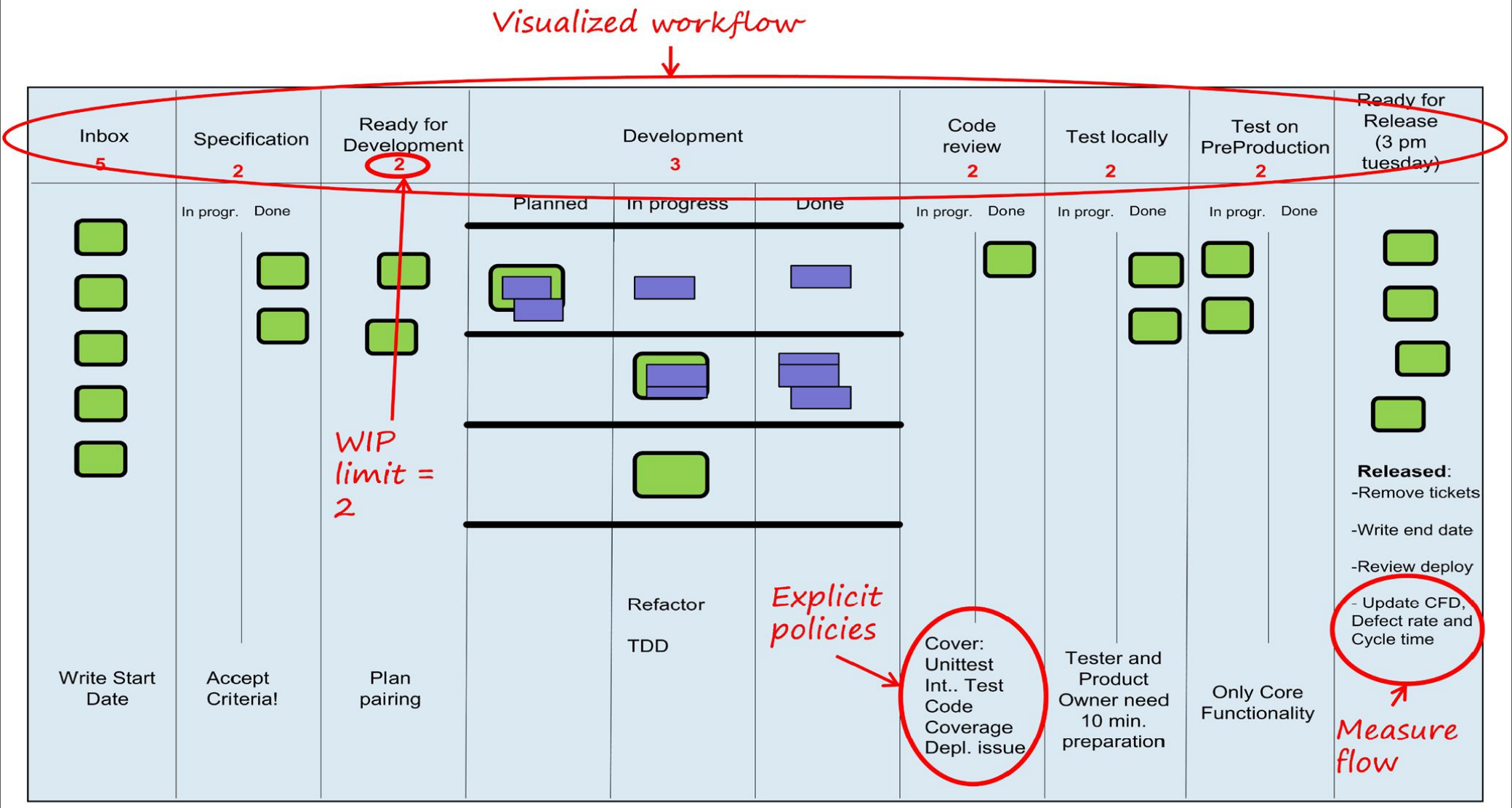
Lean Organization



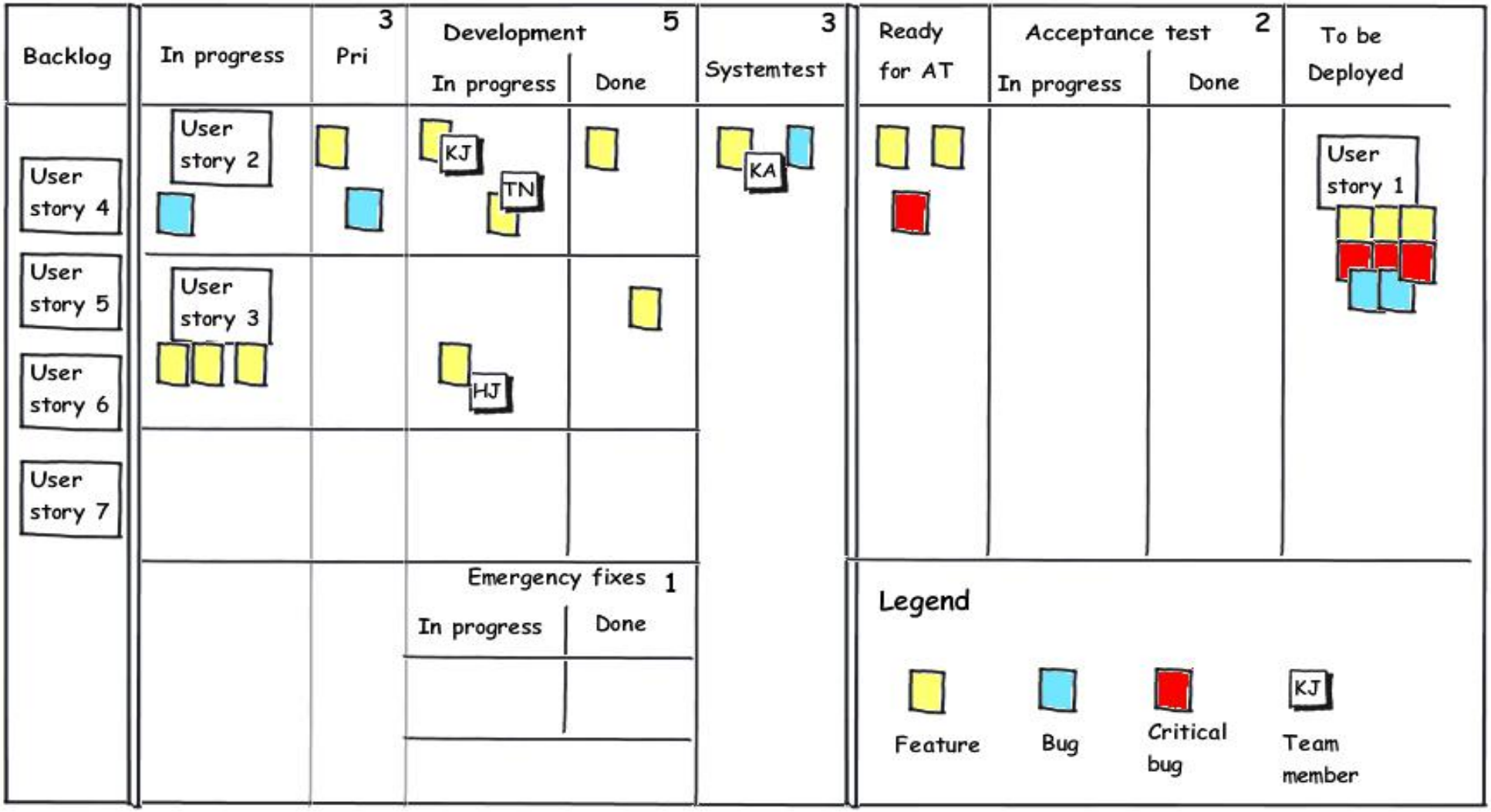
Explicit Policies Visualized on Kanban Board



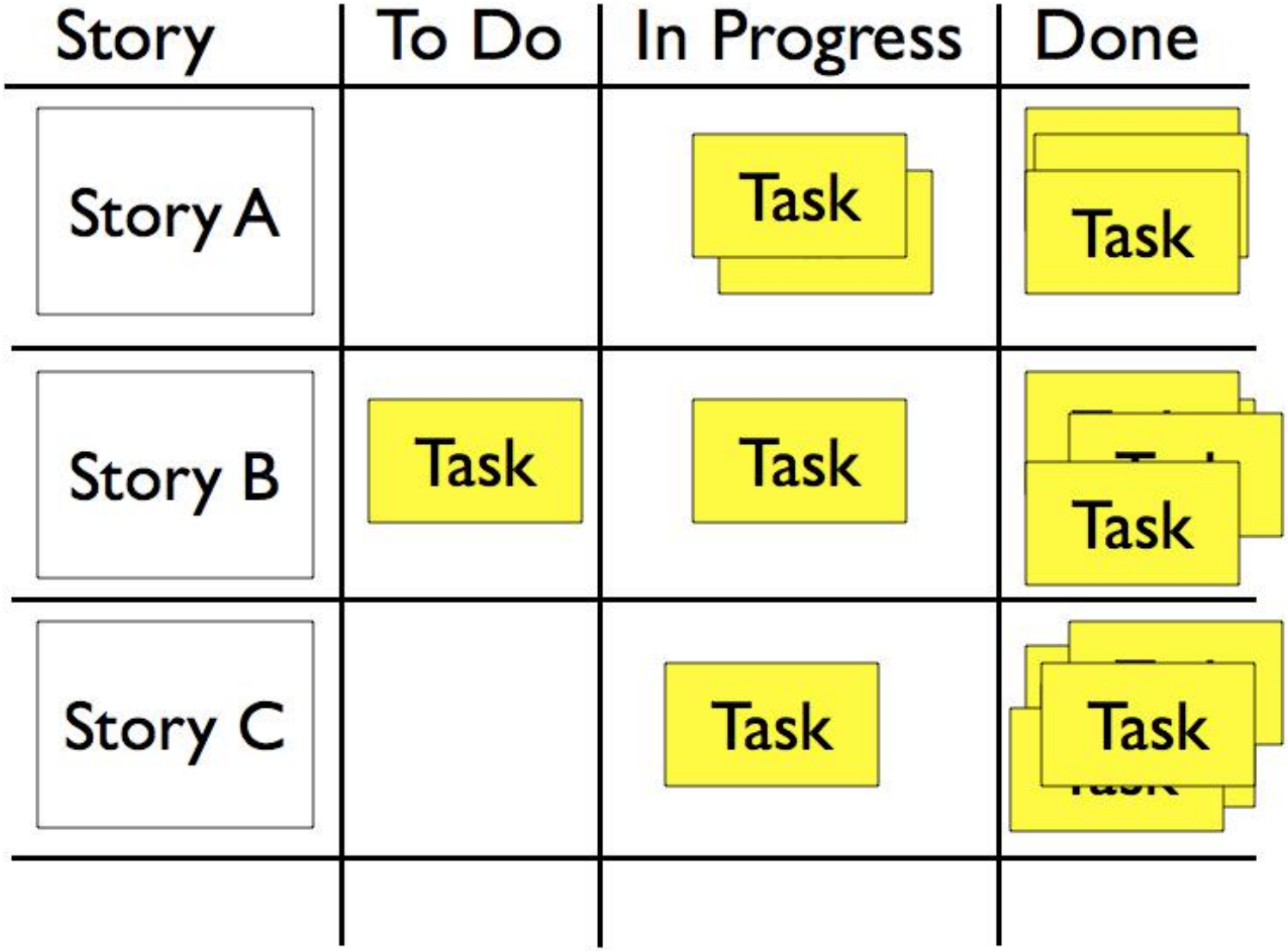
Kanban Principles in Action



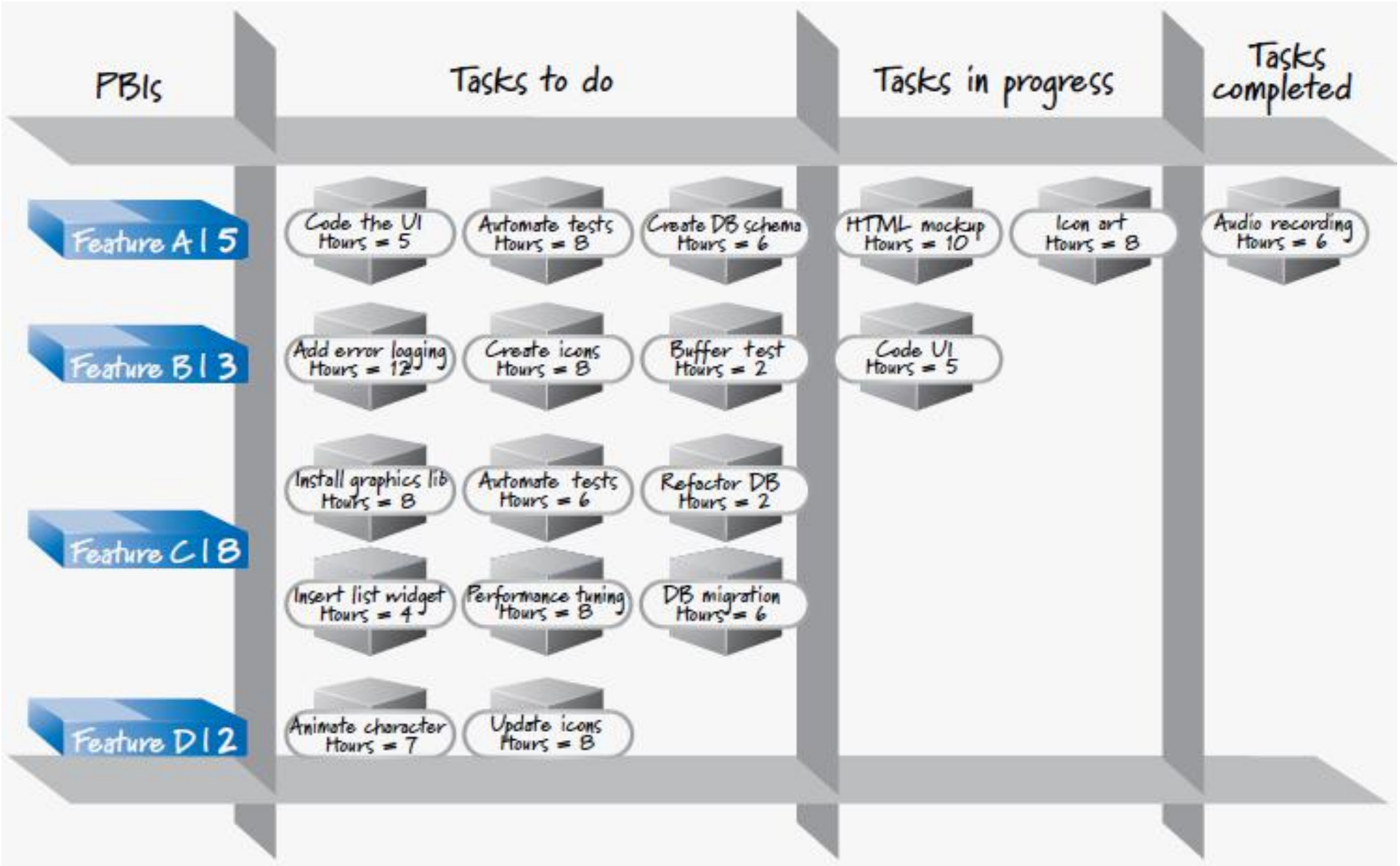
Kanban board



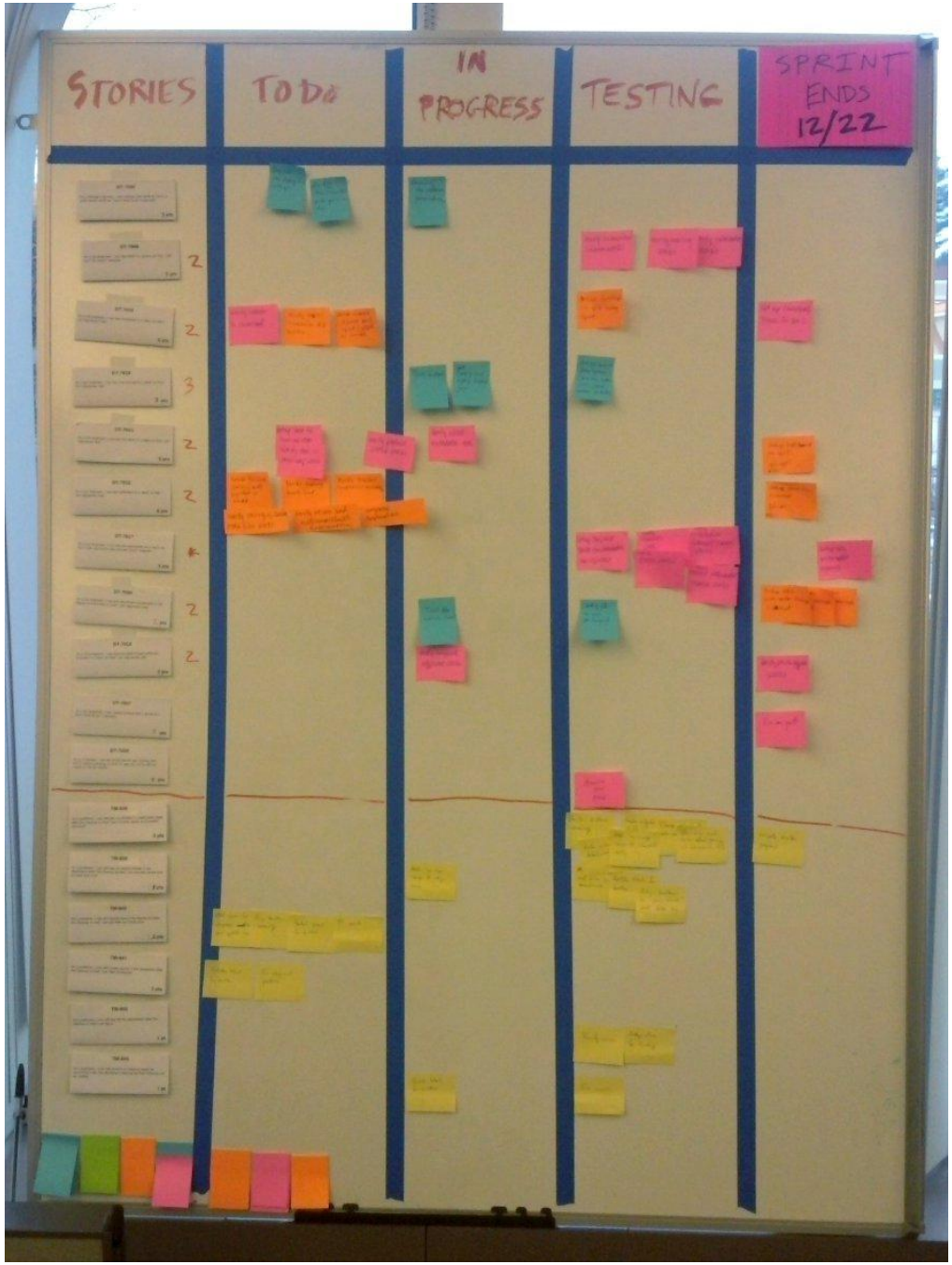
Task Board Format



Example Task Board



Scrum Task Boards

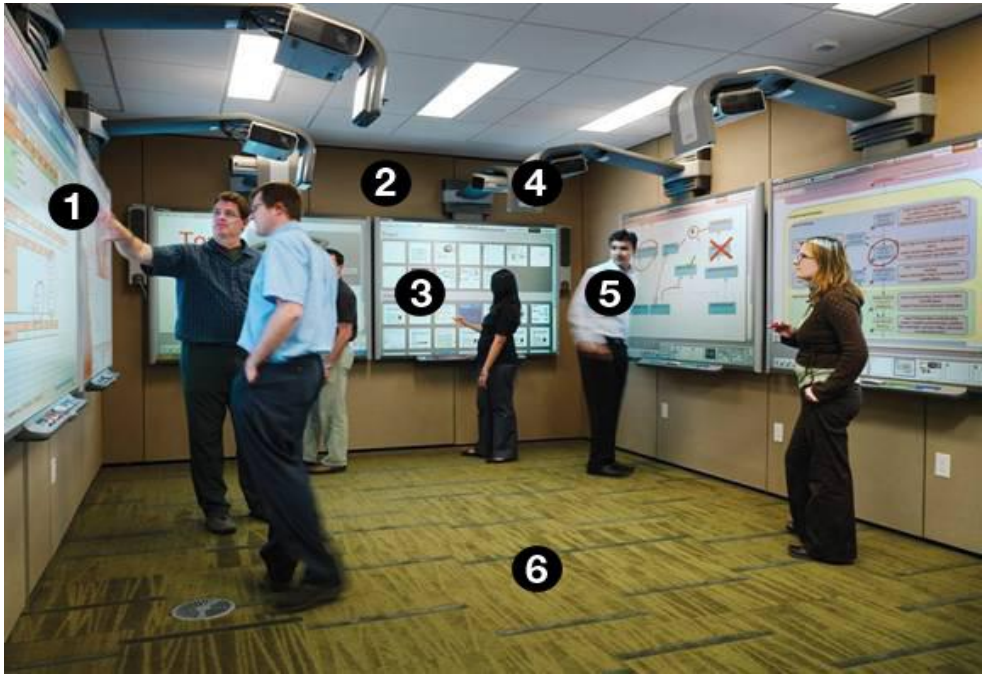


A Daily Standup Meeting

The Three Questions Raised in the Daily Scrum

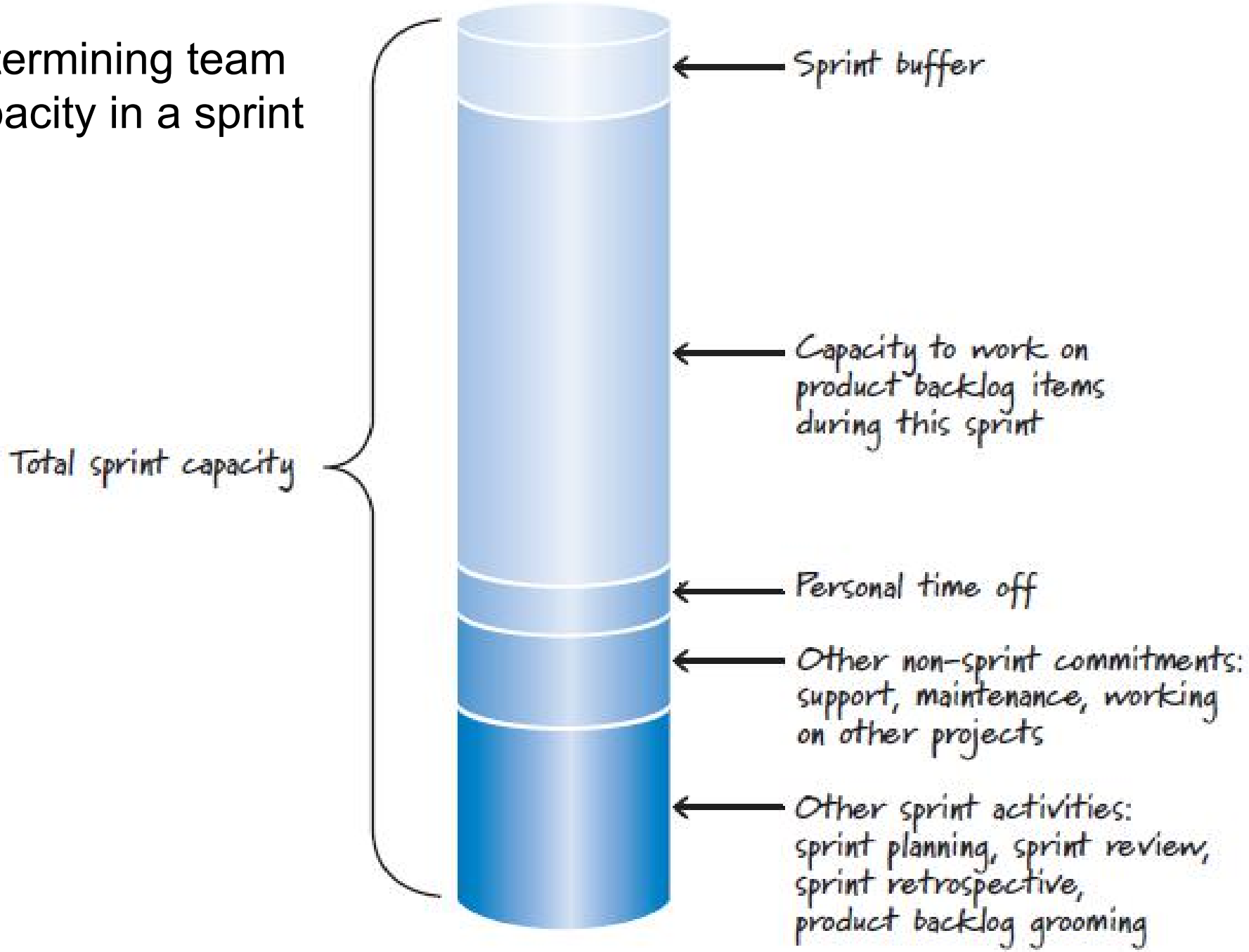
- What did I do yesterday that helped the Development Team meet the Sprint Goal?
- What will I do today to help the Development Team meet the Sprint Goal?
- Do I see any impediment that prevents me or the Development Team from meeting the Sprint Goal?

Collaboration Rooms



Sample collaboration rooms for distributed or co-located teams

Determining team capacity in a sprint



Sprint Review

- A Sprint Review is held at the end of the Sprint to inspect the Increment and adapt the Product Backlog if needed.
- During the Sprint Review, the Scrum Team and stakeholders collaborate about what was done in the Sprint.
- Based on that and any changes to the Product Backlog during the Sprint, attendees collaborate on the next things that could be done.
- This is a four-hour time-boxed meeting for one-month Sprints.
- Proportionately less time is allocated for shorter Sprints. For example, two week Sprints have two-hour Sprint Reviews.

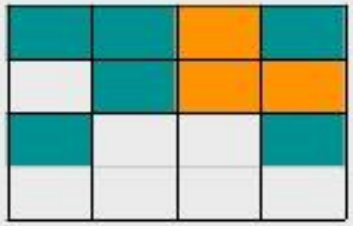
Customer Showcase

Iteration Update

- What did we achieve
- Demo / show work completed
- What issues did we face

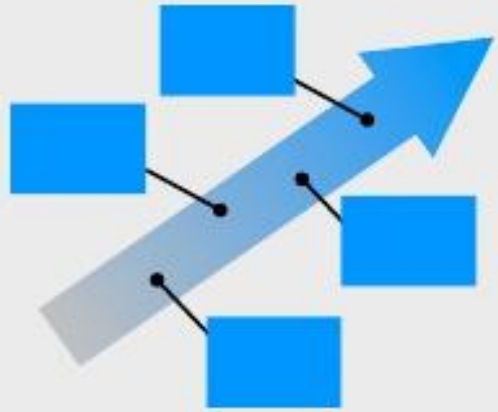
Progress Update

- How far are we thru the work / project
- What is the path to completion

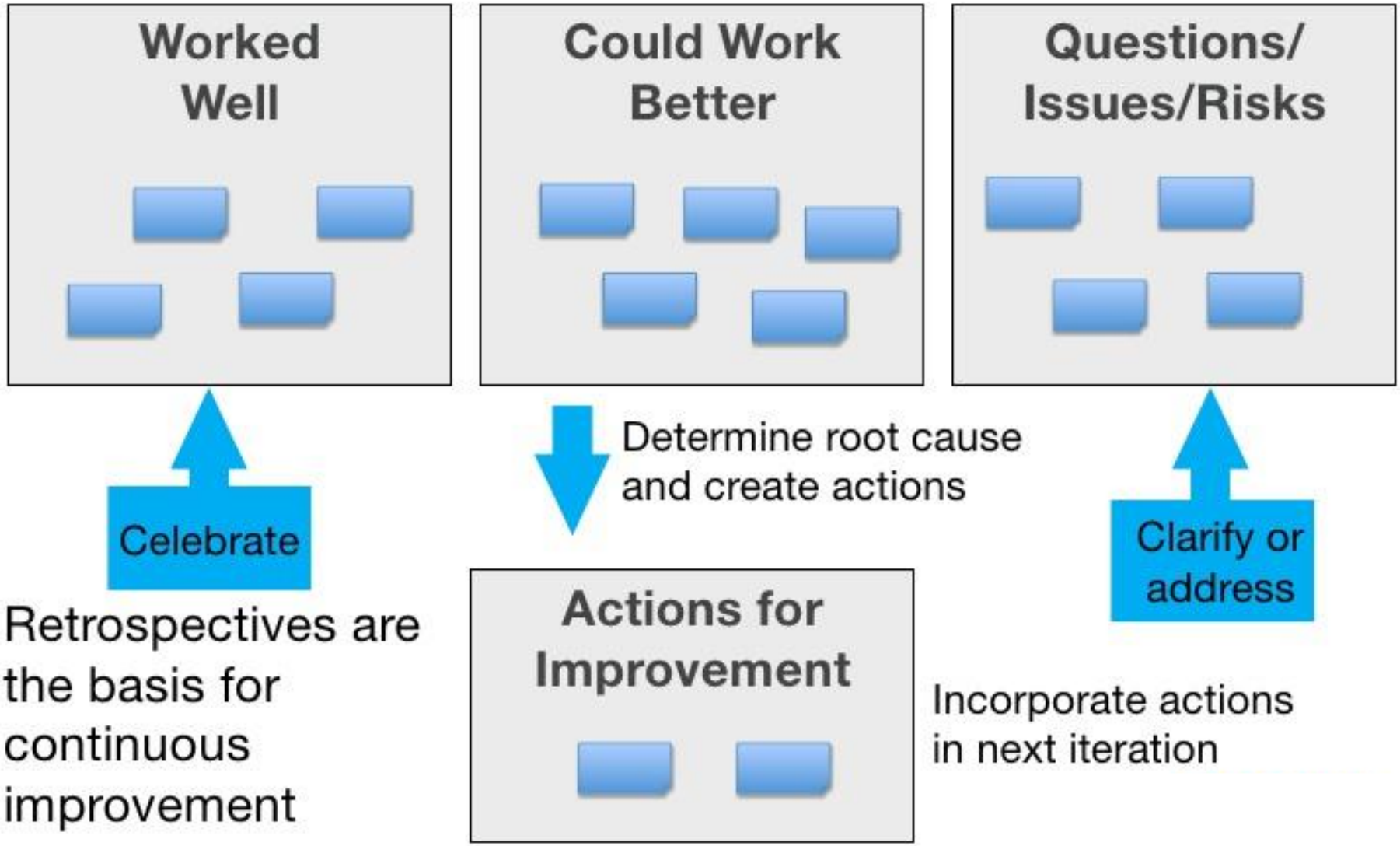


What Next?

- Focus for next iteration based on customer feedback
- Key milestones



Retrospectives



The Retrospective Prime Directive

Regardless of what we discover, we understand and truly believe that everyone did the best job they could, given what they knew at the time, their skills and abilities, the resources available, and the situation at hand.

- Norm Kerth

Start

Start getting a better understanding of the Product Backlog Items during Sprint Planning

Start doing more communication between Team and Product Owner during the Sprint

Stop

Stop waiting until the end of the Sprint to begin testing

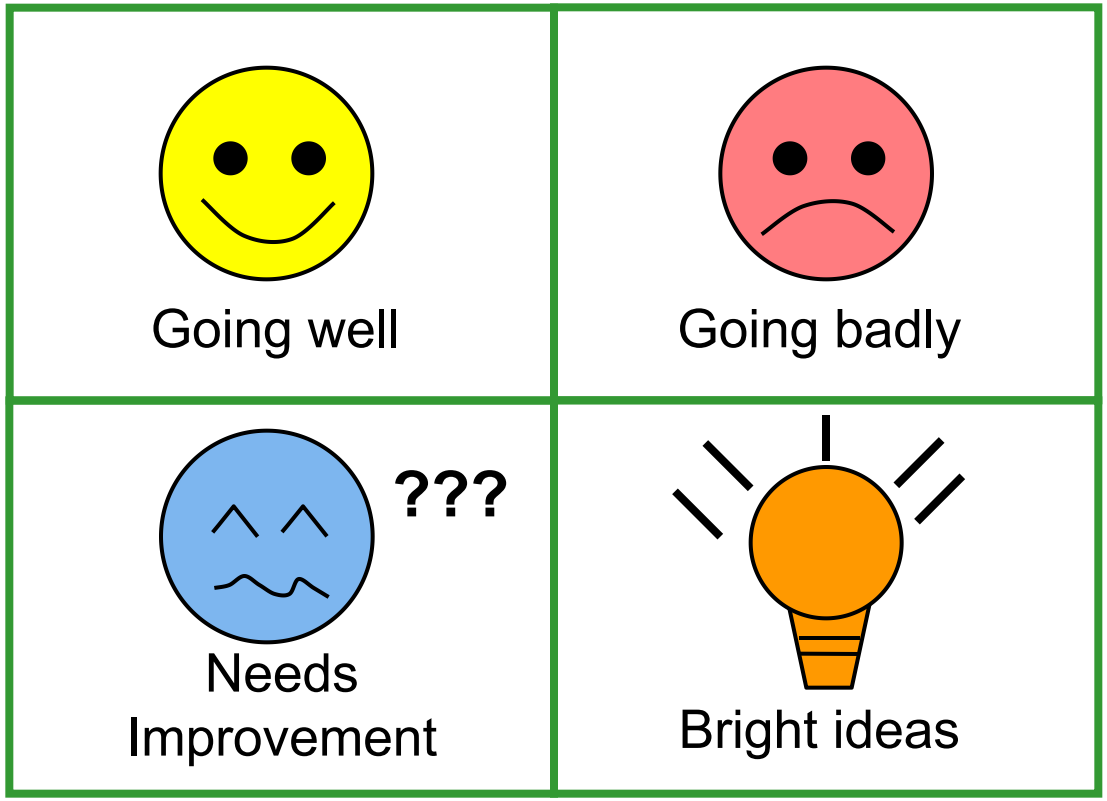
Stop allowing changes to our target during the middle of the Sprint

Continue

Continue the good teamwork and positive attitude

Continue putting a little extra buffer in our plan for each Sprint

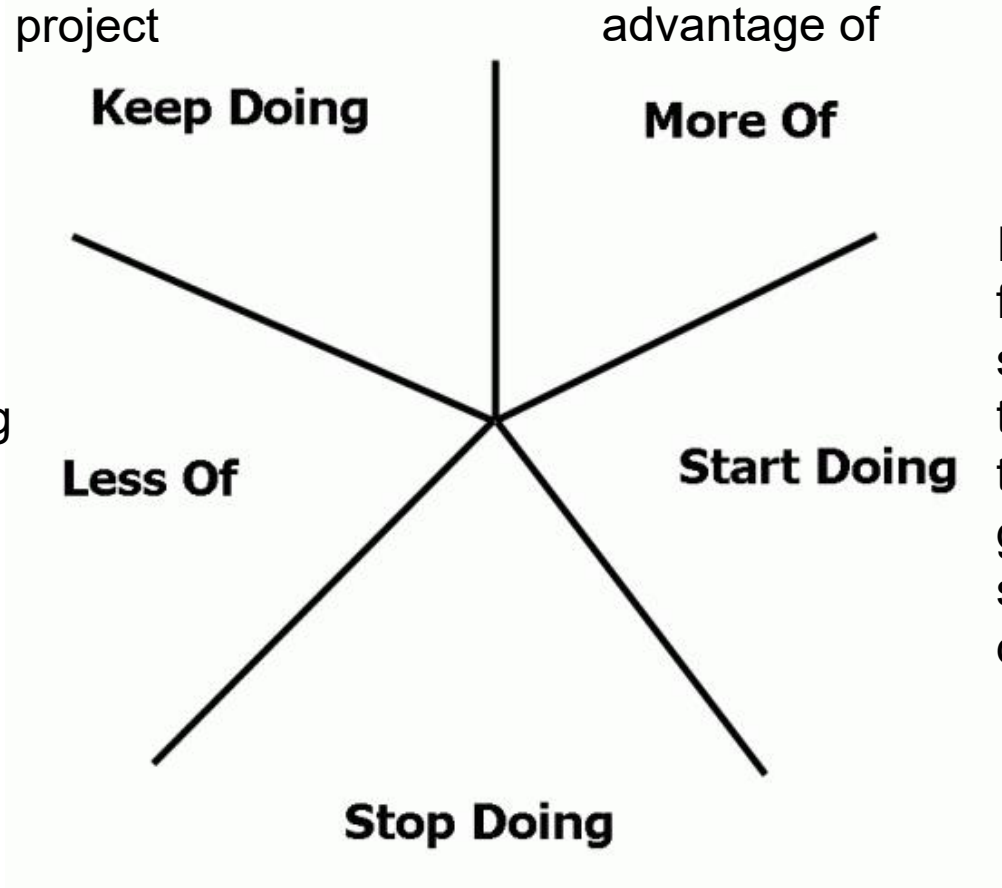
Esther Derby's 4 Quadrant Retrospective



The Retrospective Starfish

A good starting point for team members to focus on typically all the good things that they liked about a project

Another type of focus that helps further refine or highlight practices, technologies, etc that team members might want to try more and are not necessarily taking full advantage of



Helps to focus on practices that might need a bit more refining or that were simply not helpful in the current circumstance

Is a great opportunity for team members to suggest new things to try because of things that may not have gone so well or just for simply keeping things dynamic and fun

Obviously for things that are not very helpful to development practices or not adding much value

A Chain of Goals

Ultimate purpose, positive change
Brief statement or slogan

Value proposition and business benefits
Captured in the **product strategy**

Benefit of a major release, product version
Stated on the **product roadmap**

Benefit of a sprint
Shown on the **sprint backlog** or **task board**

https://www.romanpichler.com/blog/leading-through-shared-goals/

THE GO PRODUCT ROADMAP

romanpichler

DATE The release date or timeframe	*Date or timeframe*	*Date or timeframe*	*Date or timeframe*	*Date or timeframe*
	When will the release be available?			
NAME The name of the new release	*Name/version*	*Name/version*	*Name/version*	*Name/version*
	What is it called?			
GOAL The reason for creating the new release	*Goal*	*Goal*	*Goal*	*Goal*
	Why is it developed? **Which benefit does it offer?**			
FEATURES The high-level features necessary to meet the goal	*Features*	*Features*	*Features*	*Features*
	What are the 3-5 key features?			
METRICS The metrics to determine if the goal has been met	*Metrics*	*Metrics*	*Metrics*	*Metrics*
	How do we know that the goal is met?			

Sample GO Product Roadmap

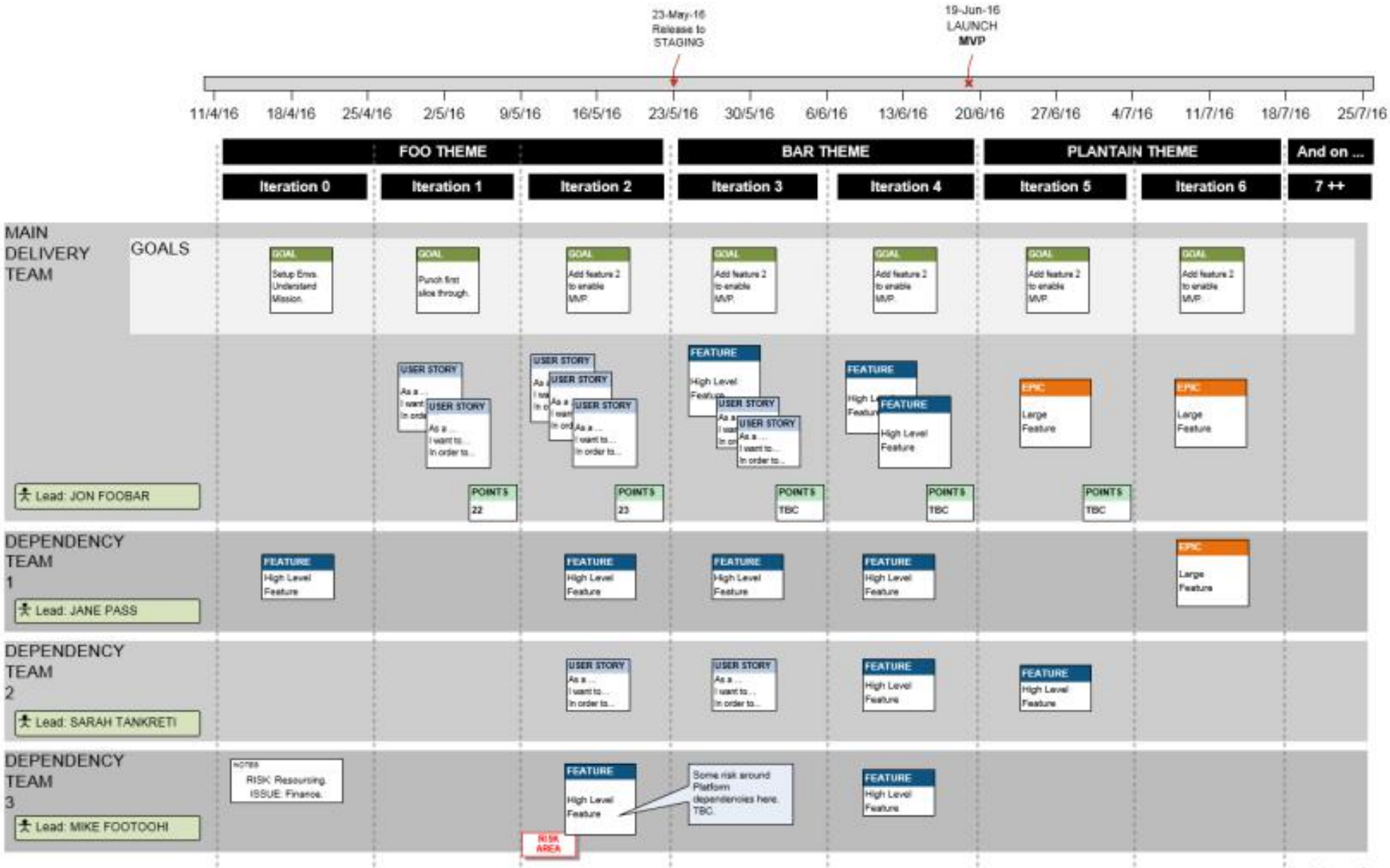
	1st quarter	2nd quarter	3rd quarter	4th quarter
	Version 1	Version 2	Version 3	Version 4
	Acquisition: Free app, limited in-app purchases	Activation: Focus on in-app purchases	Retention	Acquisition: New segment
	• Basic game functionality • Multiplayer • FB integration	• Purchase dance moves • Create new dances	• New characters and floors • Enhanced visual design	• Street dance elements • Dance competition
	Downloads: top 10 dance app	Activations, downloads	Daily active players, session length	Downloads

http://www.romanpichler.com/blog/goal-oriented-agile-product-roadmap/

Agile Release Planner – Timeline

Version 2

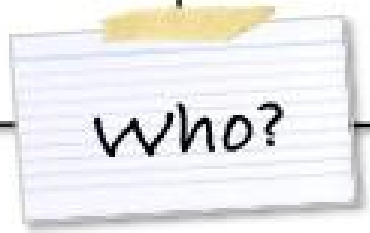
Tuesday, 08 March 2016

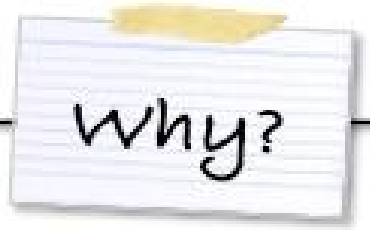


A Persona Template

A persona, first introduced by Alan Cooper, defines an archetypical user of a system, an example of the kind of person who would interact with it. The idea is that if you want to design effective software, then it needs to be designed for a specific person. For a bank, potential personas could be named Frances Miller and Ross Williams. In other words, personas represent fictitious people which are based on your knowledge of real users.

Name and Picture	Characteristics	Need
What does the persona look like? What is the persona's name?	What are the persona's relevant characteristics? For instance, demographics, job, and lifestyle information.	Why would the persona want to buy or use the product?





A Persona - An Example

Peter

Works as product manager for a mid-sized company.

Is 35 years old, holds a marketing degree.

Has got experience working as a product owner on software products with agile teams.

Has had some Scrum training.

Has managed mature products successfully. Now faces the challenge of creating a brand-new product.

Wants to leverage his agile knowledge but needs advice on creating innovative product using agile techniques.

PERSONA PROFILE

Name:
Fictionalize it

Profession:
Be very specific

Age:
Choose a number, Not a Range

Personal BG:
Mini Life story - Hometown, Family, Schooling, Work, etc

INTERESTS:
What does she spend her time on?
What enlivens her?
What does she read, watch, listen to?

NEEDS:
What does she Need, in her Life, in her work?
What's essential for her, what does she require?
WHAT's MISSING?

VALUES:
What carries Meaning For Her?
WHAT's her MORAL COMPASS?
What makes her INDIGNANT, Satisfied, FrustrATED?
What does she WANT for the World?

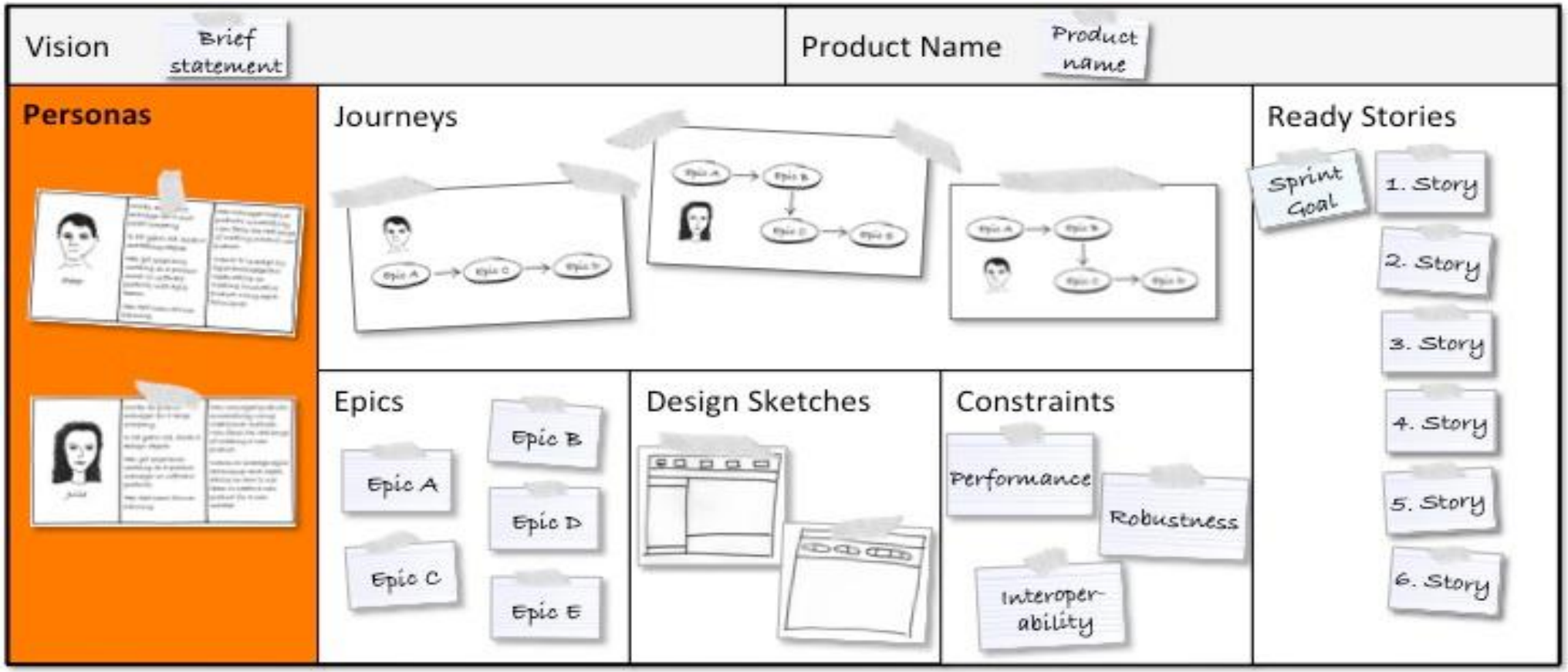
POWERS:
What Resources does she control?
What can she do, muster?
Who does she have INFLUENCE OVER?

BEHAVIORS:
What ARE her
- ROUTINES
- BUYING
- Free Time
- Rituals

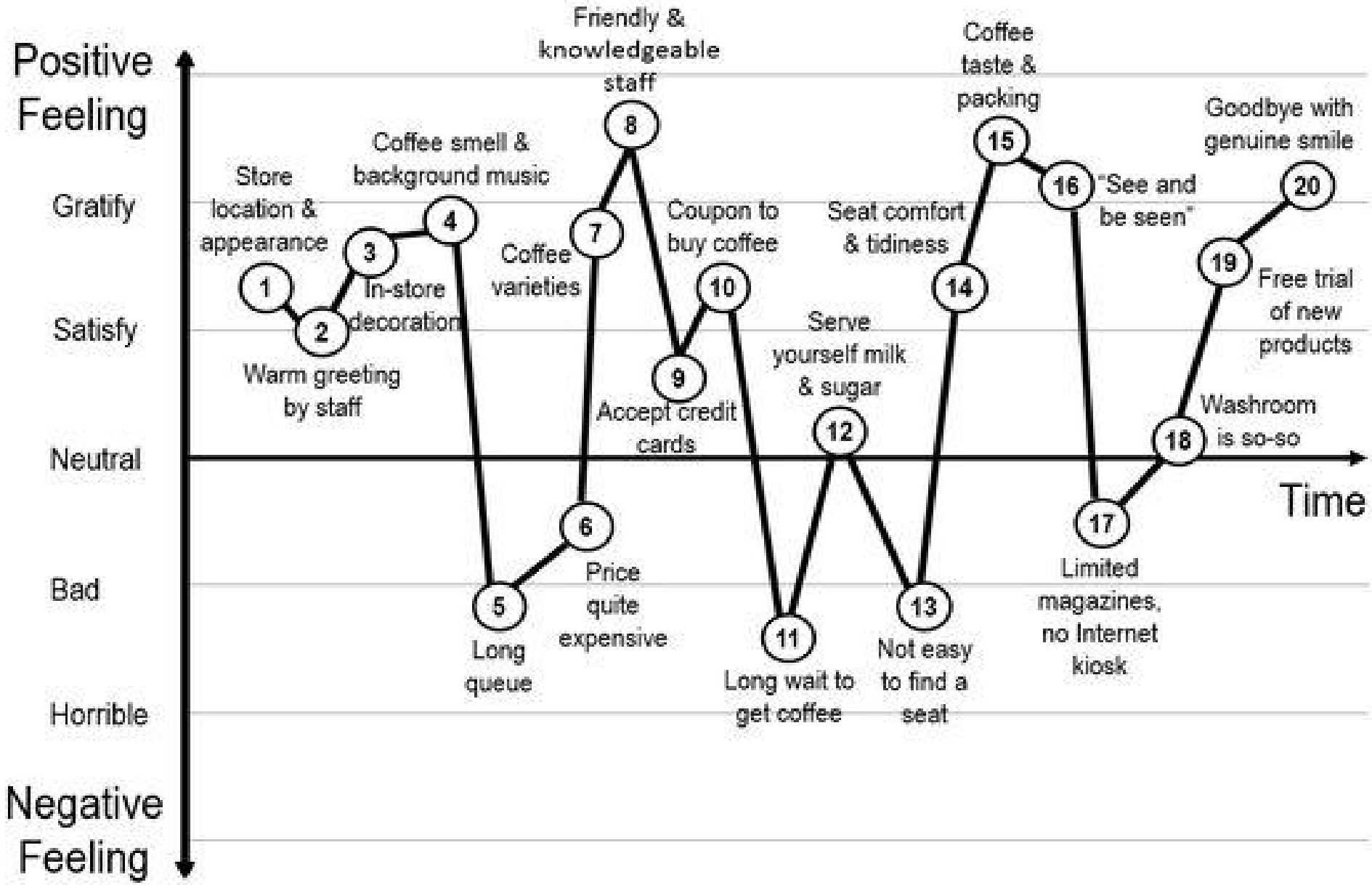
ASPIRATIONS:
WHAT KIND of PERSON DO THEY WANT TO BE?
BRANDS?
SPENDING PATTERNS?
Life Dreams? Heroes? Role Models?

http://www.openlawlab.com/2014/09/19/persona-template-user-centered-design-process/

Product Canvas



An Example of a Customer Journey Map



ESTIMATING STORY POINTS USING COMPLEXITY BUCKETS

- The Elatta Method -

This approach provides a **consistent way** for teams to size stories by discussing each story in terms of pre-defined buckets of complexity before deciding on the final points. The steps are simple:

- **Decide on the buckets of complexity** you think match your project. For example, many software development efforts have the buckets used below, but a reporting or BI project could have different ones.
- **Discuss the story in each bucket** and determine if the team can agree if the work it has a **Light**, **Medium**, **High** or **Complex** level of complexity.
- **Add up the points** and see which Fibonacci Story Point bucket it falls into. If it falls between two buckets, have the team do a gut check and decide on which ones it falls into.

User Interface	*Business Logic*	*Data / Integration*	*Testing*
L = 1	*L = 1*	*L = 1*	*L = 1*
M = 2	*M = 2*	*M = 2*	*M = 2*
H = 3	*H = 3*	*H = 3*	*H = 3*
C = 4	*C = 4*	*C = 4*	*C = 4*
Helpful Considerations:	*Helpful Considerations:*	*Helpful Considerations:*	*Helpful Considerations:*
- Number of screen fields	*- Number of business rules*	*- Number of data stores*	*- User testing complexity*
- Screen validation logic	*- BR complexity*	*- Complexity of StoredProc*	*- Data setup complexity*
- Number of screens		*- Number of tables*	*- Test automation*

***Example**:*
As a customer, I want to browse the list of products.
User Interface: M = 2
Business Logic: N/A
Data: L = 1
Testing: L = 1
Total is 4 points, which is between 3

Theme, Epic, User Story

Estimating Scope with Story Points

- A story point is an integer number that represents an aggregation of a number of aspects, each of which contributes to the potential "bigness" of a story.
  - **Knowledge**: Do we understand what the story does?
  - **Complexity**: How hard is it to implement?
  - **Volume**: How much of it is there? How long is it likely to take?
  - **Uncertainty**: What isn't known, and how might that affect our estimate?
- A story point estimate combines all of these facets into one number, which estimates the size of a story compared to other stories of a similar type.
  - A 2-point story should expect to take twice as long as a 1-point story

User Stories

- User stories are short, simple description of a feature told from the perspective of the person who desires the new capability, usually a user or customer of the system. They typically follow a simple template:

As a <type of user>, I want <some goal> so that <some reason>

- User stories are often written on index cards or sticky notes, stored in a shoe box, and arranged on walls or tables to facilitate planning and discussion.
  - As such, they strongly shift the focus from writing about features to discussing them.
  - In fact, these discussions are more important than whatever text is written.

Users Stories are ...

- A promise for a conversation
- Short description of functionality
  - from the user's perspective
- Provides value to the user or customer
  - consider all types of clients
- Must be testable
- Provides enough information for developers to make rough estimates
- Often written on index cards

Splitting Epics

As a power user, I can specify files or folders to backup based on file size, date created and date modified

As a user, I can backup my entire hard drive

As a user, I can indicate folders not to backup so that my backup drive isn't filled up with things I don't need saved

Adding Details to User Stories

- Consider the following as user story example:
  - *As a vice president of marketing, I want to select a holiday season to be used when reviewing the performance of past advertising campaigns so that I can identify profitable ones.*
- Details could be added to that user story example by adding the following conditions of satisfaction:
  - Make sure it works with major retail holidays: Christmas, Easter, Mother's Day, Father's Day, Labour Day, New Year's Day.
  - Holiday seasons can be set from one holiday to the next (such as Christmas to New Year).
  - Holiday seasons can be set to be a number of days prior to the holiday.

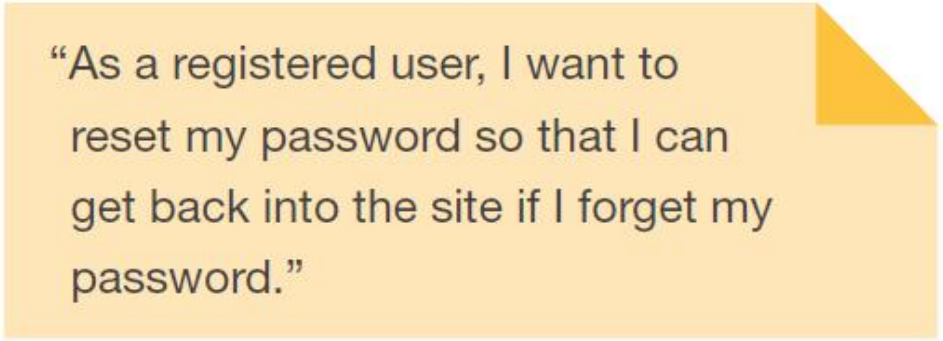
Further Examples of User Stories

- As a **user**, I want to **upload photos** so that **I can share photos with others**.

- As an **administrator**, I want to **approve photos before they are posted** so that **I can make sure they are appropriate**.

CARD

The card is an overview of who, what, and why of a user story



CONVERSATION

Through conversation, the Product Owner & Delivery Team develop a shared understanding of the goals for functionality as well as the constraints.

For our password reset example, questions might include:

- What type of authentication do we need?
- What information do we need to collect about the user?
- Are there different types of users that we have to worry about?

CONFIRMATION/CRITERIA

The results of the conversation should be captured as Acceptance Criteria. A well-defined user story is testable. How will the Product Owner confirm that the story was implemented properly? That is, when is 'DONE'?

Acceptance Criteria for our Reset Password story might include:

- Username, password, and email address are all required. Display name can be provided, but is optional.
- Passwords can be 6 – 200 characters in length.
- Passwords should be stored encrypted and cannot be easily decrypted.

Requirements as Stories

A story is a multi-purpose artefact used to collate requirements and facilitate planning and tracking of the project

Story is written from perspective of the user in simple language

Acceptance criteria ensures clear definition of done

Additional details may be captured in electronic system

FRONT

Points

User Story:

As a... user (of the system)
I want... (a particular facility)
So that... (I receive some benefit)

BACK

Acceptance Criteria:

Given... (a particular scenario)
When... (I perform some action)
Then... (I expect some outcome)

User Story Acceptance Criteria

As a customer, I want to be able to see my daily energy usage so that I can lower my energy costs & usage.

Acceptance Criteria:

- Read DecaWatt meter data every 10 seconds & display on portal in 15-minute increments & display on in-home display every read.
- Read KiloWatt meter for new data as available & display on the portal every hour & on the in-home display after every read.

Common Mistakes When Writing User Stories

- Example: As a user I want to be able to manage ads, so that I can remove expired and erroneous ads.

- The mistake?
  - User Story for a “User”
  - Is that a portal administrator, who wants to have possibly of database clean-up and ads moderation?
  - Or maybe it is an advertiser, who wants to have list of all ads he submitted to the side and have possibility of removing ads when they are not needed any longer or there was error found in the content?
  - What is missing is persona or role mentioned.

Common Mistakes When Writing User Stories

- Example: As a Product Owner I want the system to have possibility of deleting ads, so that users have possibility of deleting ads.

- The mistake?
  - User Story for Product Owner
  - Person writing this User Story did that only for sake of doing it
  - Also no role or persona to give you clues about user expectations.

Common Mistakes When Writing User Stories

- Example: As a developer I want to replaced the folder widget, so that I have maintained folder widget.
- The mistake?
  - This a technical story (usually consists of necessary maintenance tasks like software updates, re-factoring, changing frameworks and so on)
  - They are totally rightful to be implemented, but represented like in the example above they do not represent value for the customer and you will not get a buy-in from Product Owner.
  - Rewrite it from a user point of view with persona on role: ‘As a commercial user I want the system to allow me run multiple searches at the same time, so that I can do my job faster.’
  - Tasks to go with it can be: ‘Re-factor search handling mechanism to allow multiple threads for single user.’, ‘Update java version to 64-bit one.’
  - Acceptance criteria need to define some measurable and testable definition of improvement like: ‘Single user can run 5 searches at the same time.’ and ‘Search returns results in less than 4 seconds.

Common Mistakes When Writing User Stories

- Example: As an commercial advertiser I want to have filtering option.

- The mistake?
  - No Business Value or Benefit for Customer
  - We have the role, we have the need, but reason and business value are missing.
  - Why does an commercial advertiser want to have filtering option?
  - What does he/she want to achieve?

Technical Story Defined

- A Technical Story is one focused on non-functional support of a system.
  - For example, implementing back-end tables to support a new function, or extending an existing service layer.
- Sometimes they are focused on classic non-functional stories, for example: security, performance, or scalability related.
- Another type of technical story focuses more towards technical debt & refactoring.
- And still another might focus on performing technical analysis, design, prototyping & architectural work.
- All of these are focused towards underlying support for base functional behavior.

Types Of Technical Stories

- **Product Infrastructure** – stories that directly support requested functional stories. This could include new and/or modified infrastructure. It might also identify refactoring opportunities, but driven from the functional need.
- **Team Infrastructure** – stories that support the team and their ability to deliver software. Often these surround tooling, testing, metrics, design, and planning. It could imply the team “creating” something or “buying and installing” something.
- **Refactoring** – these are stories that identify areas that are refactoring candidates. Not only code needs refactoring, but also it can often include designs, automation, tooling, and any process documentation.
- **Bug Fixing** – either clusters or packages of bugs that increase repair time or reduce aggregate of testing time. So this is more of an efficiency play.
- **Spikes** – research stories that will result in learning, architecture & design, prototypes, and ultimately a set of stories and execution strategy that will meet the functional goal of the spike. Spikes need to err on the side of prototype code over documentation as well, although I don’t think you have to “demo” every spike.

User Stories and Technical Stories in Agile Development

User Story:

- As an online shopper at ACME Widget Company, I want to ensure that my order is complete and valid when I submit it at the online store, so that I can get the products I ordered in a timely manner without mistakes or delays.

Technical Story:

- In order to ensure that only valid orders are accepted by the system, System A must map the Order XML to the main canonical and provide the main canonical to SYSTEM B to run order validation rules.

User Stories and Technical Stories in Agile Development

User Story:

- As a user requesting authentication, I need to be able to login via the web app, so that I can manage my account details via the web

Technical Story:

- In order to deliver a new authentication mechanism for web based applications XXXX will need to extend the kiosk authentication code in the SSL including 2-layer authentication: passwords and user-centric questions

TECH STORY FORMAT

In order to <achieve some goal> <a system or persona> needs to <some action>

- Example 1: In order to split the collection curve wizard story, the tech lead needs to do some detailed design. (Just enough to split the story.)

- Example 2: In order to estimate the “user connection report” story, the tech lead and BA need to research capabilities of MSTR and have some conversations with the PO. (Just enough to estimate the story.)

A good Product Backlog is "DEEP"

- **Detailed appropriately.** The top priority items are more fine-grained & detailed than the lower priority items, since the former will be worked on sooner than the latter. For example, the top 10% of the backlog may be composed of very small, well-analyzed items, and the other 90% much less so.
- **Estimated.** The items for the current release need to have estimates, and furthermore, should be considered for re-estimation each Sprint as everyones learns and new information arises. The Team provides the Product Owner with effort estimates for each item on the Product Backlog, and perhaps also technical risk estimates. The Product Owner and other business stakeholders provide information on the value of the product requests, which may include revenue gained, costs reduced, business risks, importance to various stakeholders, & more.
- **Emergent.** In response to learning and variability, the Product Backlog is regularly refined. Each Sprint, items may be added, removed, modified, split, & changed in priority. Thus, the Product Backlog is continuously updated by the Product Owner to reflect changes in the needs of the customer, new ideas or insights, moves by the competition, technical hurdles that appear, and so forth.
- **Prioritized.** The items at the top of the Product Backlog are prioritized or ordered in a 1-N order. In general, the highest-priority items should deliver the most bang for your buck: lots of bang (business value) for low buck (cost). Another motivation to increase the priority of an item is to tackle high risks early, before the risks attack you.

The INVEST CRITERIA

For 'Good' Stories

INDEPENDENT	STORIES SHOULD BE INTERNALLY INDEPENDENT DURING THEIR EXECUTION. THE SUCCESS OF ONE STORY SHOULD NOT DEPEND ON THE SUCCESS OF ANOTHER BEING WORKED ON AT THE SAME TIME.
NEGOTIABLE	STORIES ARE THE NEGOTIATION UNITS IN SCRUM. STORIES ARE AGREED TO IN PLANNING AND ARE DELIVERED. THE TEAM NEGOTIATES ACTUAL CONTENT OF STORIES DURING DEVELOPMENT.
VALUABLE	STORIES ARE, BY DEFINITION, UNITS OF VALUE THAT ARE REQUESTED BY STAKEHOLDERS OR TEAM MEMBERS. THE VALUE CAN BE EXTERNAL OR INTERNAL - FOR STAKEHOLDERS OR THE TEAM.
ESTIMABLE	THE TEAM NEEDS TO BE ABLE TO AGREE TO THE STORY, WHICH IMPLIES THAT THE STORY'S EFFORT COULD BE ESTIMATED. HOWEVER, SOME STORIES ARE AMBIGUOUS AND MUST BE TIME-BOXED.
SMALL	STORIES SHOULD BE SMALL ENOUGH THAT THERE IS LITTLE CONFUSION ABOUT WHAT THEY MEAN AND CAN BE COMPLETED RELATIVELY QUICKLY. WE RECOMMEND A SINGLE FOCUS PER STORY.
TESTABLE	STORIES SHOULD BE TESTABLE. EACH STORY NEEDS TO BE VERIFIABLE, SO THAT THE TEAM CAN DETERMINE WHEN IT IS DONE. DONENESS TAKES DIFFERENT FORMS FOR DIFFERENT STORIES.

I	**I**ndependent: Ensure the story can stand alone, thus avoiding the creation of complex dependencies with other stories.
N	**N**egotiable: Assume there will be ongoing negotiation about the priority, form, and function that the story represents.
V	**V**aluable: Stay focused on features that are valuable to the user and not on the technical details meaningful only to developers.
E	**E**stimatable: Keep the story discreet and clear enough to develop an order-of-magnitude estimate for effort in resource requirements.
S	**S**mall: For clarity, estimating, and planning purposes, ensure the story does not become an epic.
T	**T**estable: Explicitly describe in the story the feature that is expected so that it can pass a simple "yes" or "no" test.

Independent

- Stories are easiest to work with if they are independent.
- That is, we'd like them to not overlap in concept, and we'd like to be able to schedule and implement them in any order.
- An independent story can also be independently *valued*.

Independent - Example of Functional Dependency

As an administrator, I can set the customer's password security rules so that users are required to create and retain secure passwords, keeping the system secure.

As a customer, I am required to follow the password security rules set by the administrator so that I can maintain high security to my account.

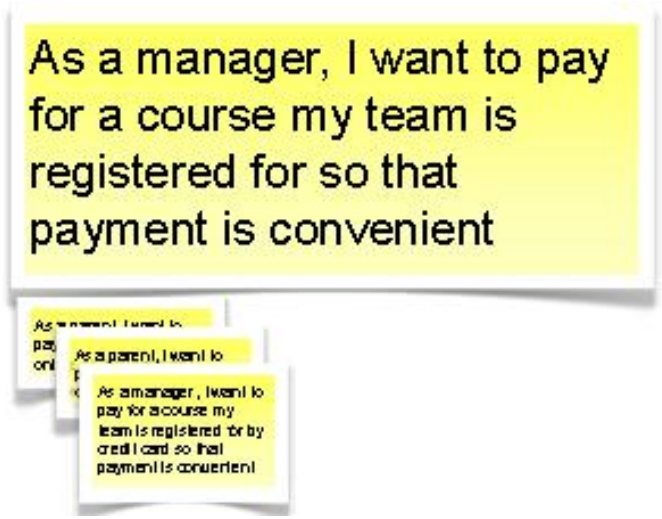
Independent - Removed Dependency

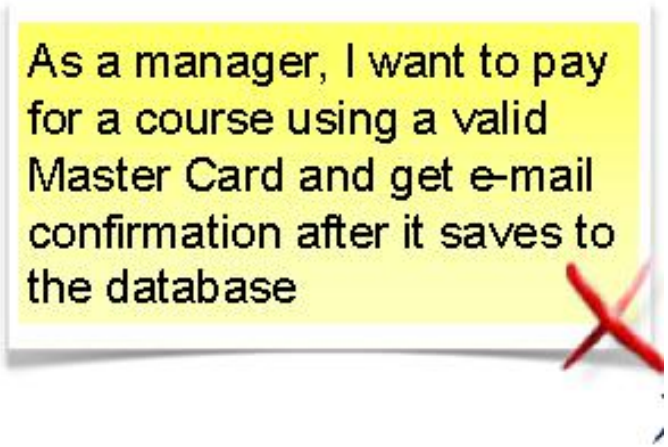
As an administrator, I can set the password expiration period so that users are forced to change their passwords regularly.

As an administrator, I can set the password strength characteristics so that users are required to create difficult-to-hack passwords.

Negotiable

- A good story is negotiable.
- It is not an explicit contract for features; rather, details will be co-created by the customer & programmer during development.
- A good story captures the essence, not the details.
- The lack of overly constraining and too-detailed requirements enhances the team's & business's abillity to make trade-offs between functionality & delivery dates.





Valuable

- The focus here is to bring actual project-related value to the end-user.
- Coming up with technical stories that are really fun to code but bring no value to the end-user beats one of the Agile Principles, which is to continuously deliver valuable software.

- As a utility marketing director, I can present users with new pricing programmes so that they are more likely to continue purchasing from us.

Valuable

- Value depends on what we're trying to achieve.
- One old formula is IRACIS:
  - Increase Revenue: Add new features (or improve old ones) because somebody will pay more when they're present.
  - Avoid Costs: Much software is written to help someone avoid spending money. For example, suppose you're writing software to support a call center: every second you save on a typical transaction means fewer total agents are needed, saving the company money.
  - Improve Service: Some work is intended to improve existing capabilities. Consider Skype, the voice network: improving call quality is not a new feature, but it has value. (For example, more customers might stay with the service when call quality is higher.)

Valuable

- Meet Regulations: The government may demand that we support certain capabilities (whether we want to or not). For example, credit card companies are required to support a "Do Not Call" list for customers who don't want new offers. If the company didn't provide the capability by a certain date, the government would shut down the company.
- Build Reputation: Some things are done to increase our visibility in the marketplace. An example might be producing a free demo version of packaged software, to improve its marketing. In effect, these are an indirect way to increase revenue.
- Create Options: Some things give us more flexibility in the future. For example, we may invest in database independence today, to give us the ability to quickly change databases in the future. The future is uncertain; options are insurance.
- Generate Information: Sometimes we need better information to help us make a good decision. For example, we might do an A-B test to tell us which colour button sells more.

Estimable

- If a user story size cannot be estimated, it will never be planned, tasked, and, thus, become part of an iteration.
- So there's actually no point in keeping this kind of user story in the Product Backlog at all.
- Most of the time, estimation cannot be executed due to the lack of supporting information either in the story description itself or directly from the Product Owner.

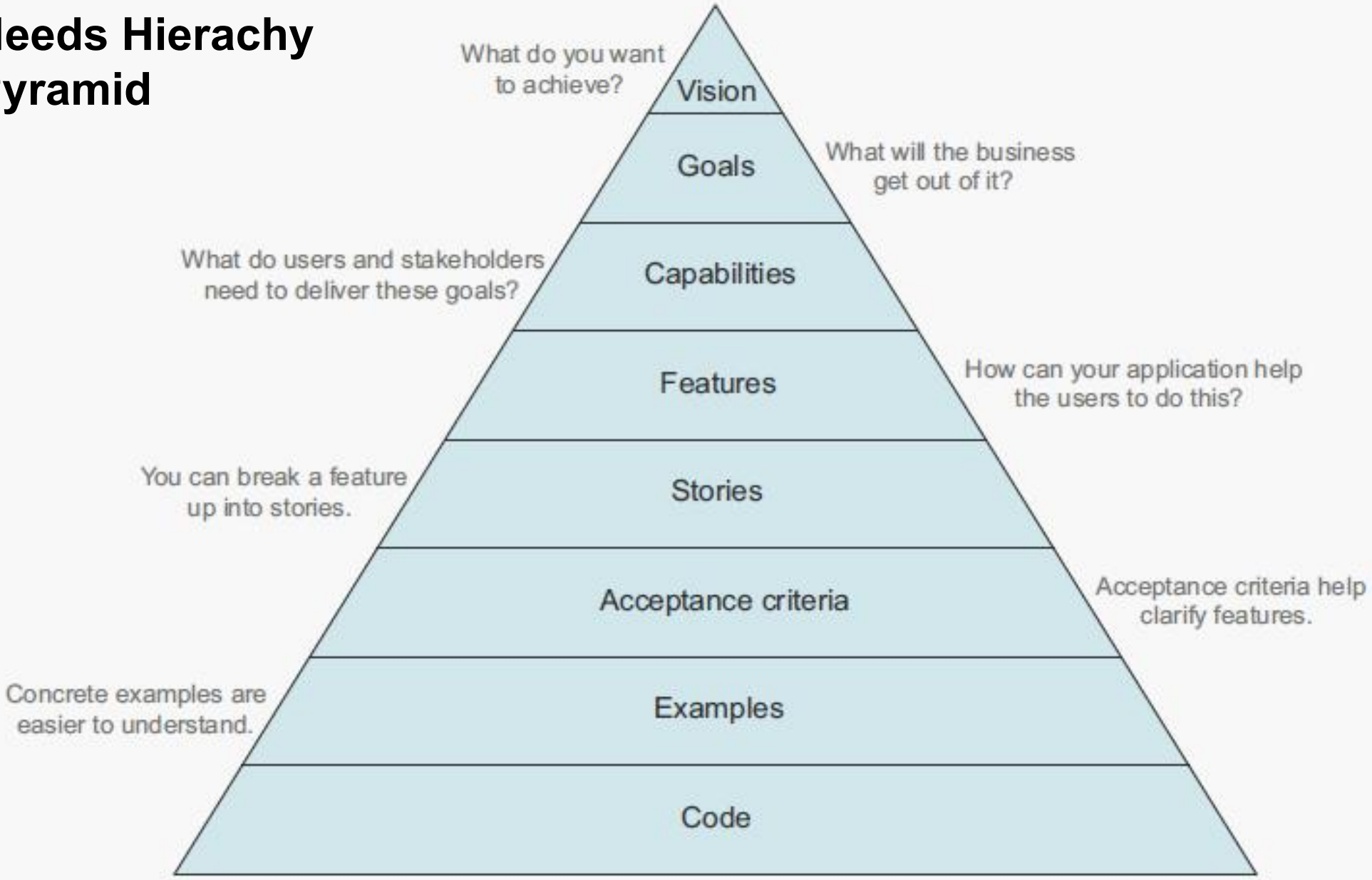
Sized Appropriately (Small)

- Try to keep your user story sizes to typically a few person-days & at most a few person-weeks.
- Anything beyond that range should be considered too large to be estimated with a good level of certainty or even "epic" & broken down into smaller user stories.
- There's no problem in starting with epic stories, as long as they are broken down when the time to place them in an iteration backlog becomes closer.

Testable

- Bear in mind that a story should only be considered DONE, among other things, if it was tested successfully.
- If one cannot test a story due to lack of information (see "Estimable" above), the story should not be considered a good candidate to be part of an iteration Backlog.
- This is especially true for teams employing TDD - Test Driven Development

Needs Hierachy Pyramid



All features, and ultimately all code, should map back to business goals and the project vision.

Needs Hierarchy (I)

- At the highest level is the project *vision*, a short statement that provides a high-level guiding direction for the project. This statement helps ensure that all the team members understand the principal aims and assumptions of the project. For the Flying High Frequent Flyer website, the vision could be to create a website that will enable Frequent Flyer members to consult and use their points and membership privileges more quickly and more easily than with the current manual system.
- More precisely, a project aims to support a number of business goals. *Business goals* are executive-level concepts that generally involve increasing revenue, protecting revenue, or reducing costs. For the Flying High Frequent Flyer website, one of the primary business goals might be “Earn more ticket sales revenue through repeat business from Frequent Flyer members.”

Needs Hierarchy (II)

- As a software developer, your job is to design and build *capabilities* that help the business realize these goals. A capability gives your users the ability to achieve some goal or fulfill some task, regardless of implementation. A good way to spot a capability is that it can be prefixed with the words "to be able to." For example, Flying High Frequent Flyer members need the capability to be able to book flights and benefit from their member privileges while doing so.

TIP Capabilities don't imply a particular implementation. For example, "the ability to book a flight" could be provided online or over the telephone.

- You design and implement *features* to deliver these capabilities. Features are what you actually build, and they're what deliver the value. One feature that helps deliver the capability to book flights might be "Book flights online using Frequent Flyer points."

TIP Features are pieces of deliverable software functionality, such as an "Online Frequent Flyer account summary."

Needs Hierarchy (III)

- To build up your understanding of the features, you can use concrete *examples* of what the system should do in different situations.
- These examples can form the basis for the *acceptance criteria* of a feature. Once you have a set of features and scenarios to work with, you can start designing a user interface and writing the *code* that goes behind it.

TIP Examples illustrate how a feature works, such as "Account summary should display a list of recently earned points."

Acceptance Criteria

Acceptance Criteria – An Example

- **Given**: I have selected a date to purchase tickets
- **When**: I submit my selection
- **Then**: I receive a confirmation for each seat

Acceptance Criteria – An Example

Given:

Origin	*Destination*	*Pass*	*Date*
Hong Kong	New Zealand	1	16 January 2014

When: Submit

Then: Display “Congratulations! You are ready for your trip. Your confirmation numbers are NNNNNN”

ConfirmationNo

000001

Acceptance Criteria – A More Complex Example

Given:

#	*Origin*	*Destination*	*Pass*	*Date*
1	Hong Kong	New Zealand	1	16 January 2014
2	Hong Kong	New Zealand	3	16 January 2014
3	New Zealand	Hong Kong	4	23 January 2014

When: Submit

Acceptance Criteria – A More Complex Example

Then: Display “Congratulations! You are ready for your trip. Your confirmation numbers are NNNNNN”

#	Confirmation
1	000001
2	000002, 000003, 000004
3	000005, 000006, 000007, 000008

Table 7.1 *Types of Testing That Benefit from Automation*

Test	Definition	Agile Considerations for Automation	Example
Microtests	Individual unit tests that run while the code is being developed. They test an individual behavior in a single object.	Pro: Fast and automated check for quality while the code is being developed. Con: Will not detect system-level defects.	Tests a small portion of the code (less than 12 lines) and only a few classes in an object.
Acceptance	Acceptance tests are performed to ensure that the product functions meet the agreed-upon requirements. Acceptance tests can be performed by the test team when acting as a mock customer or by a customer before formally accepting a contracted product.	Pro: Can be read by nonprogrammers. Con: Slow to execute.	Behavior-Driven Testing (e.g., Gherkin, JBehave)—a tool that tests the behavior of a product. The test code is written in "business readable language," meaning the test language can be read and understood by team members who are not familiar with programming languages. See "Gherkin Example."

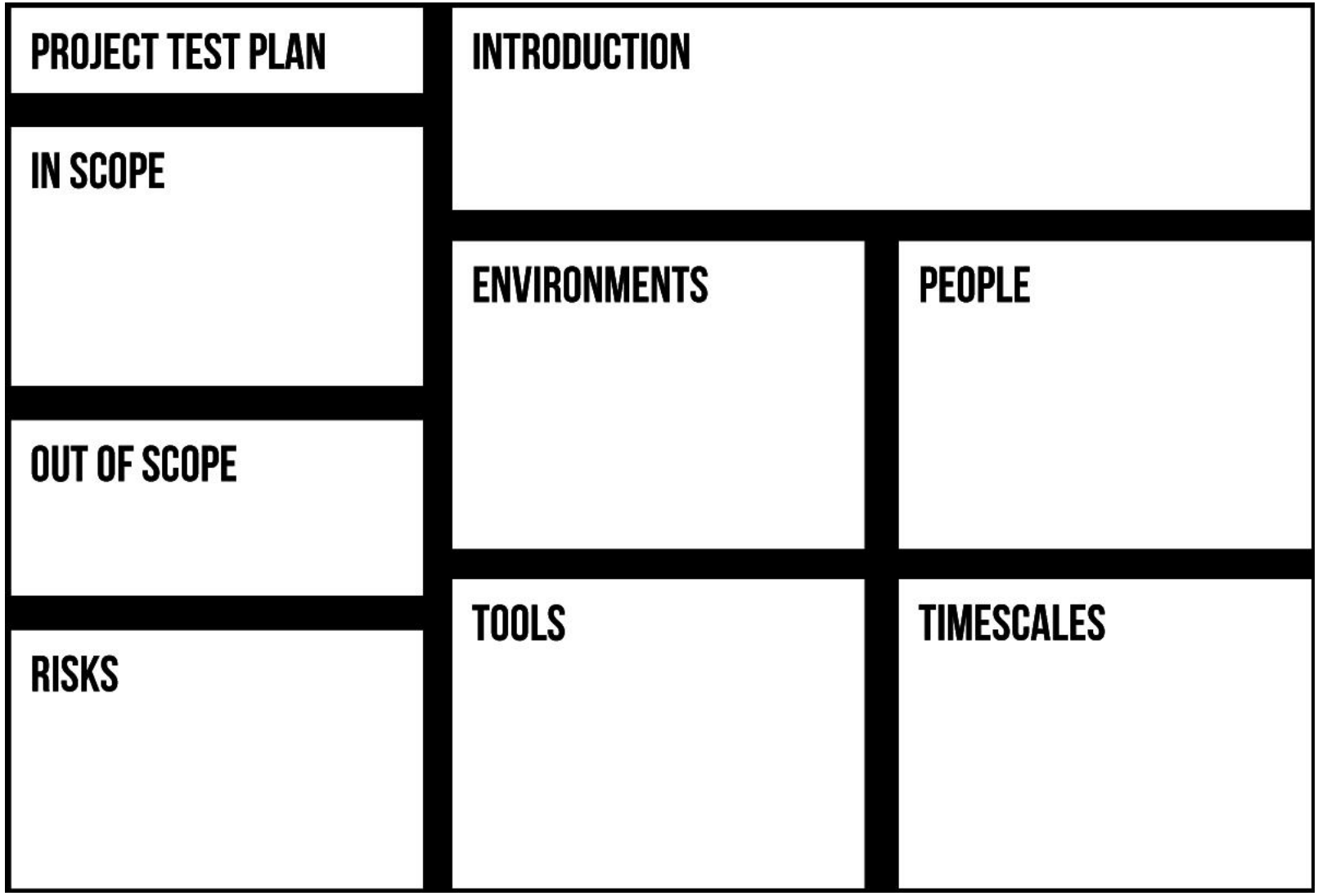
Regression	Testing to see if new code has broken code that already exists in the product.	Pro: Quickly determines if new code has broken existing code. Typically the code is run against a select set of tests (sometimes referred to as a "regression bucket") that provide good coverage of the product functions. Con: Usually does not cover all test cases and cannot detect system-level problems.	JUnits tests can be run against every new build to verify that previously working code still functions properly.
System	Testing all components of the product together on all supported platforms emulating the customer experience.	Pro: Tests the full system. Con: Tests can take significant time and money and may require external dependencies such as special hardware.	Testing an app to make sure it works on both Android and iOS operating systems.
User Interface	Automating the clicks through a graphical or command-line interface.	Pro: Allows user interface testing to be part of the regression and speeds up user interface testing. Con: Some features are difficult to automate and may miss some critical bugs.	Selenium (docs.seleniumhq.org) will automatically test web pages to make sure all features (e.g., buttons, links, mouse overs) are functioning.

Table quoted from Ashmore S., and Runyun K. (2015) Introduction to Agile Methods. Addison-Wesley, pp. 200 - 201

One Page Test Plan

PROJECT TEST PLAN	AUTHOR: A TESTER	
INTRODUCTION	IN SCOPE	OUT OF SCOPE
RISKS ASSUMPTIONS	RESOURCE TIMESCALES	ENVIRONMENTS + TOOLS

One Page Test Plan



10 Patterns for Splitting a User Story

1. Workflow Steps

Identify specific steps that a user takes to accomplish a specific workflow, and then implement the workflow in incremental stages.

As a utility, I want to update and publish pricing programs to my customer.

...I can publish pricing programs to the customer's in-home display.

...I can send a message to the customer's web portal.

...I can publish the pricing table to a customer's smart thermostat.

2. Business Rule Variations

At first glance, some stories seem fairly simple. However, sometimes the business rules are more complex or extensive than the first glance revealed. In this case, it might be useful to break the story into several stories to handle the business rule complexity.

As a utility, I can sort customers by different demographics.

...sort by ZIP code.

...sort by home demographics.

...sort by energy consumption.

10 Patterns for Splitting a User Story

3. Major Effort

Sometimes a story can be split into several parts where most of the effort will go toward implementing the first one. In the example shown next, processing infrastructure should be built to support the first story; adding more functionality should be relatively trivial later.

As a user, I want to be able to select/change my pricing program with my utility through my web portal.

...I want to use time-of-use pricing.

...I want to prepay for my energy.

...I want to enroll in critical-peak pricing.

---

4. Simple/Complex

When the team is discussing a story and the story seems to be getting larger and larger ("What about x? Have you considered y?"), stop and ask, "What's the simplest version that can possibly work?" Capture that simple version as its own story, and then break out all the variations and complexities into their own stories.

As a user, I basically want a fixed price, but I also want to be notified of critical-peak pricing events.

...respond to the time and the duration of the critical-peak pricing event.

...respond to emergency events.

10 Patterns for Splitting a User Story

5. Variations in Data

Data variations and data sources are another source of scope and complexity. Consider adding stories just-in-time after building the simplest version. A localization example is shown here:

As a utility, I can send messages to customers.

...customers who want their messages:

...in Spanish

...in Arabic, and so on.

6. Data Entry Methods

Sometimes complexity is in the user interface rather than the functionality itself. In that case, split the story to build it with the simplest possible UI, and then build the richer UI later.

As a user, I can view my energy consumption in various graphs.

...using bar charts that compare weekly consumption.

...in a comparison chart, so I can compare my usage to those who have the same or similar household demographics.

10 Patterns for Splitting a User Story

7. Defer System Qualities

Sometimes, the initial implementation isn't all that hard, and the major part of the effort is in making it fast or reliable or more precise or more scalable. However, the team can learn a lot from the base implementation, and it should have some value to a user, who wouldn't otherwise be able to do it all. In this case, break the story into successive "ilities."

As a user, I want to see real-time consumption from my meter.

...interpolate data from the last known reading.

...display real-time data from the meter.

---

8. Operations (Example: Create Read Update Delete (CRUD))

Words like *manage* or *control* are a giveaway that the story covers multiple operations, which can offer a natural way to split the story.

As a user, I can manage my account.

...I can sign up for an account.

...I can edit my account settings.

...I can cancel my account.

...I can add more devices to my account.

10 Patterns for Splitting a User Story

9. Use-Case Scenarios

If use cases have been developed to represent complex user-to-system or system-to-system interaction, then the story can often be split according to the individual scenarios of the use case.*

I want to enroll in the energy savings program through a retail distributor.

Use case/story #1 (happy path): Notify utility that consumer has equipment.

Use case/story #2: Utility provisions equipment and data and notifies consumer.

Use case/story #3 (alternate scenario): Handle data validation errors.

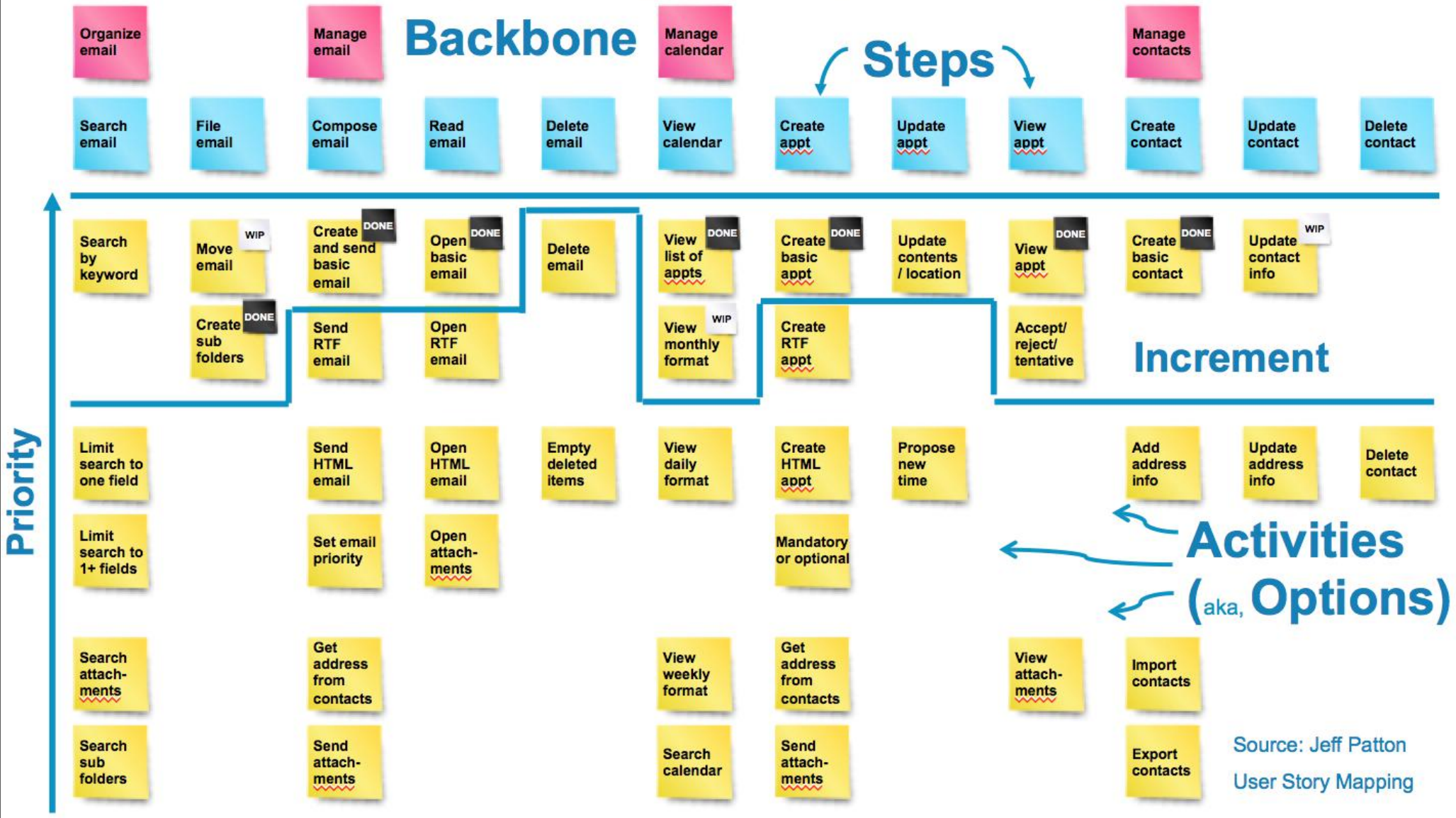
---

10. Break Out a Spike

In some cases, a story may be too large or overly complex, or perhaps the implementation is poorly understood. In that case, build a technical or functional spike to figure it out; then split the stories based on that result.

Example of a Story Map

Story Mapping Example



http://cmforagile.blogspot.com/2015/07/story-telling-with-story-mapping.html

Reasons
A BACKLOG ITEM
May Be An Epic

COMPLEX	THE ITEM MIGHT BE TOO COMPLEX TO BE UNDERSTOOD WELL ENOUGH TO BE AGREED TO.
UNKNOWN	MAYBE NOBODY ON THE TEAM KNOWS ENOUGH ABOUT THE ITEM TO EVEN MAKE A GUESS WHETHER OR NOT IT CAN BE AGREED TO.
RISKY	THERE ARE TOO MANY UNKNOWNS; IT IS TOO RISKY TO AGREE TO THE ITEM WITHOUT FURTHER INVESTIGATION OR A MITIGATION STRATEGY.
BIG	THE ITEM COULD JUST BE TOO BIG TO DO IN ONE SPRINT, EVEN THOUGH IT IS WELL UNDERSTOOD.

THE EISENHOWER BOX

"What is important is seldom urgent and what is urgent is seldom important."
-Dwight Eisenhower, 34th President of the United States

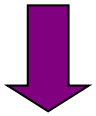
FURPS

- Refers to all the types of requirements that affect the system's fitness for use
  - **FUNCTIONALITY**: what the system does for the user [Capability (Size & Generality of Feature Set), Reusability (Compatibility, Interoperability, Portability), Security (Safety & Exploitability)]
  - **USABILITY**: how easy it is for a user to get the system to do it [Human Factors, Aesthetics, Consistency, Documentation, Responsiveness]
  - **RELIABILITY**: how reliably the system does it [Availability (Failure Frequency (Robustness/Durability/Resilience), Failure Extent & Time-Length (Recoverability/Survivability), Predictability (Stability), Accuracy (Frequency/Severity of Error)]
  - **PERFORMANCE**: how often, or how many of it, it can do [Speed, Efficiency, Resource Consumption (power, ram, cache, etc.), Throughput, Capacity, Scalability]
  - **SUPPORTABILITY**: how easy it is for us to maintain & extend the system that does it [(Serviceability, Maintainability, Sustainability, Repair Speed) - Testability, Flexibility (Modifiability, Configurability, Adaptability, Extensibility, Modularity), Installability, Localizability]

NFRs

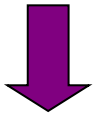
Expressing NFRs in User Story Form

All messages shall be displayed in less than one minute



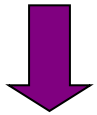
As a customer, I want to be notified of any messages from the utility in less than one minute of arrival so that I can take appropriate action quickly

All open source software must be approved by the CFO



As your CFO, I need to make sure we don't use any open source software that I haven't approved, so we don't have licence exposure

Update mobile app with new logo



As a product manager, I need to make sure we update the logo to satisfy marketing

Examples of NFRs

Accessibility
Audit and control
Availability
Backup
Capacity: current and forecast
Certification
Compatibility compliance
Configuration management
Dependency on other parties
Documentation
Disaster recovery
Efficiency
Effectiveness
Escrow
Extensibility
Failure management
Legal and licensing issues
Interoperability
Maintainability
Modifiability
Open Source
Operability
Patent-infringement-avoidability
Performance/response time
Platform compatibility
Price
Privacy
Portability
Quality
Recovery
Reliability
Resilience
Resource constraints
Response time
Robustness
Scalability
Security
Software, tools, standards
Stability
Safety
Supportability
Testability
Usability

Spikes

- Spikes are a type of story that are used for activities such as research, design, exploration and even prototyping.
- The purpose of a spike is to gain the knowledge necessary to reduce the risk of a technical approach, better understand a requirement, or increase the reliability of a story estimate.
- Originally defined within XP, spikes primarily come in two forms, technical and functional.
  - Functional spikes are used to analyze aggregate functional behaviour and to determine how to break it down, how it might be organized and where risk and complexity exists, in turn influencing implementation decisions.
  - Technical spikes are used to determine feasibility and impact of design strategies.
- Like other stories, spikes are estimated & are demonstrated at the end of the iteration.

Examples of Spikes

As a consumer, I want to see my daily energy use in a histogram so that I can quickly understand my past, current, and projected energy consumption.

In this case, a team might create two spikes:

- ***Technical spike***: Research how long it takes to update a customer display to current usage, determining communication requirements, bandwidth, and whether to push or pull the data.
- ***Functional spike***: Prototype a histogram in the web portal and get some user feedback on presentation size, style, and charting attributes.

Typical Risks and Responses

Risk	*How it is Addressed in the Agile Environment*
Schedule risk	• Execute pair programming. • Schedule daily meetings. • Determine the team's velocity.
Scope risk (scope creep)	• Maintain a collaborative and transparent nature to work. • Continually validate features with customers.
Requirements risk	• Perform an iterative discovery of requirements (user stories/features).
Technical and feasibility risk	• Attempt the riskiest technical elements early (depending on interdependencies). • Use early iterations as feasibilities and technical test cases.
Resource risk	• Recruit highly skilled individuals. • Integrate these individuals into the team. • Constantly collaborate and interact. • Team members are not swapped in and out.

‘Speedboat’ innovation game is helpful in gathering information from the team about success/failure factors as well as visualising inputs to fuel further discussion.