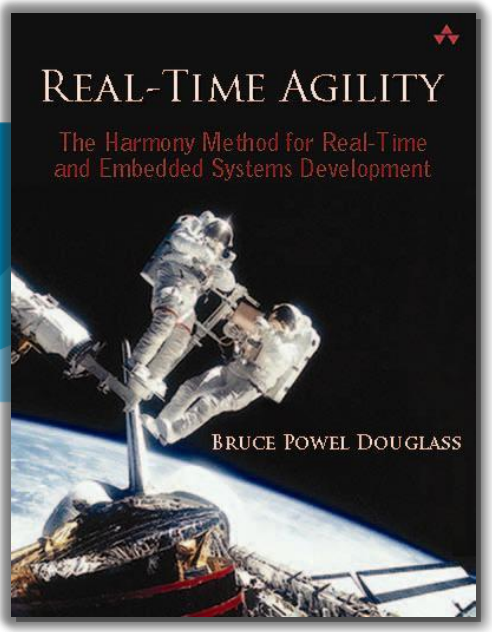
How to fail with Agility

Agile Antipatterns for the Masses

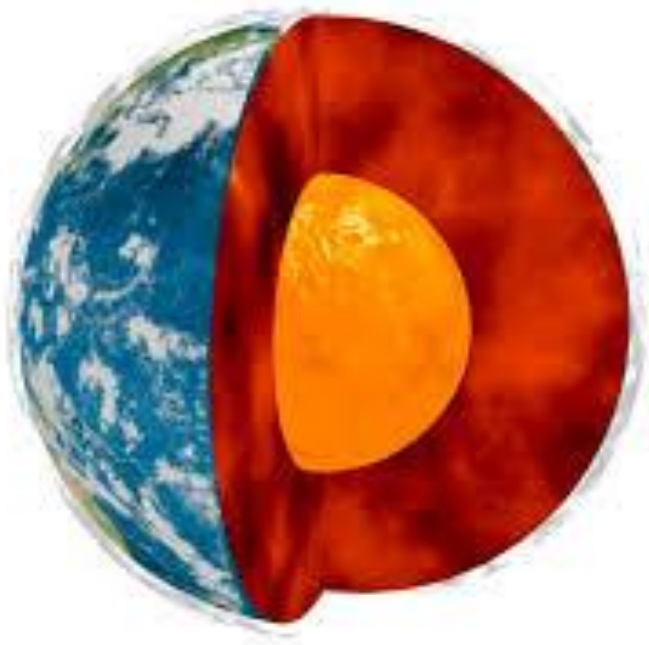
Bruce Powel Douglass, Ph.D.
Chief Evangelist, IBM IoT
Bruce.Douglass@us.ibm.com
Twitter: @IronmanBruce
www.bruce-douglass.com



Section

Core agile principles

Core principles constitute the key underlying ideas and philosophical tenets of agility



Agile Manifesto

The Manifesto for Agile Software Development, commonly known as the Agile Manifesto, is an intentionally streamlined expression of the core values of agile project management. Use this manifesto as a guide to implement agile methodologies in your projects.

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

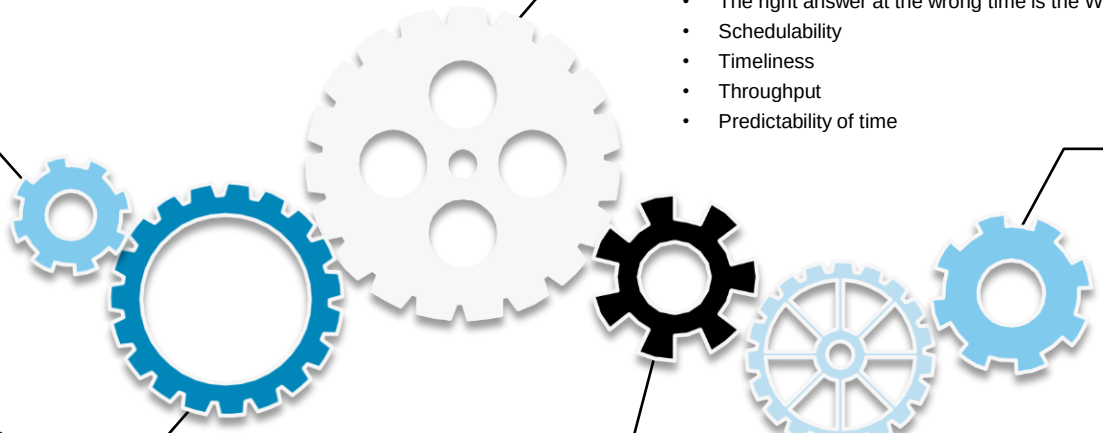
- **Individuals and interactions over processes and tools**
- **Working software over comprehensive documentation**
- **Customer collaboration over contract negotiation**
- **Responding to change over following a plan**

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

The 12 Agile Principles

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

Agile in the Embedded Systems Space



01
HW/SW Co-development

Often HW doesn't exist at the start of the project

02
Specialized HW

- Must develop custom HW, device drivers, and use RT OS
- More challenging test enviironment

03
Performance

- The right answer at the wrong time is the WRONG ANSWER
- Schedulability
- Timeliness
- Throughput
- Predictability of time

04
High Dependability

- High Safety
- High Reliability
- High Security
- Often subject to regulation

05
Resource Constraints

- Slow CPUs
- Reduced memory
- Low bandwidth
- No virtualization or mass storage

06
Shipped as Complete System

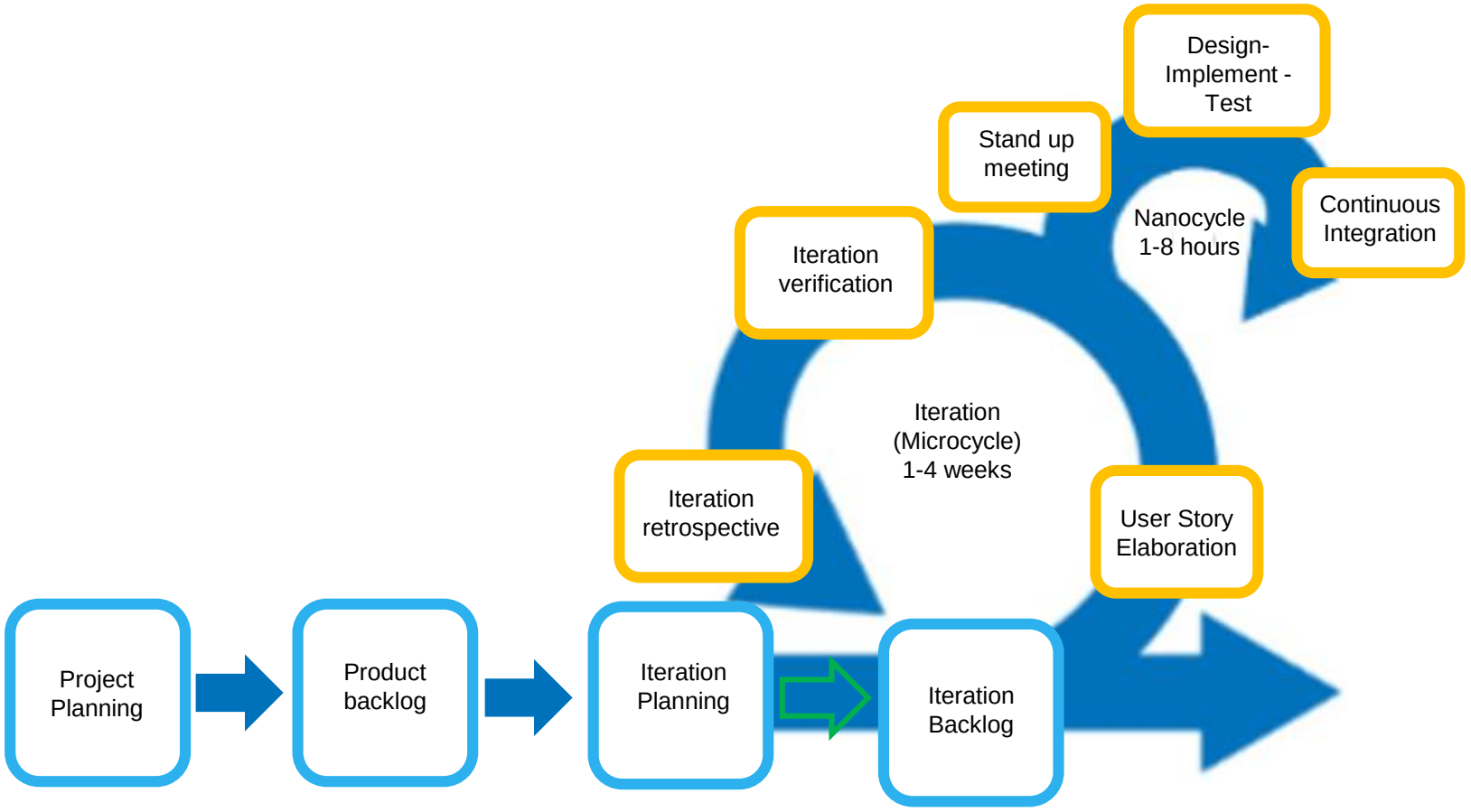
Must be shipped and certified with other engineering facets

Section

The agile lifestyle

Time is cyclic for the agile practitioner

The Agile Lifestyle



Project Planning

01

Product owner identifies the **product vision**

The product vision is a definition of what your product is, how it will support your company or organization's strategy, and who will use the product

02

Product owner creates the **product roadmap**

The product roadmap is a high-level view of the product requirements, with a loose time frame for when you will develop those requirements.

03

Product owner creates the **release plan**

The release plan identifies a high-level timetable for the release of working software. An agile project will have many releases, with the highest-priority features launching first

04

Team performs **iteration planning**

Iteration planning sessions take place at the start of each sprint, where the team determines what requirements will be in the upcoming iteration

05

Team performs **iteration review**

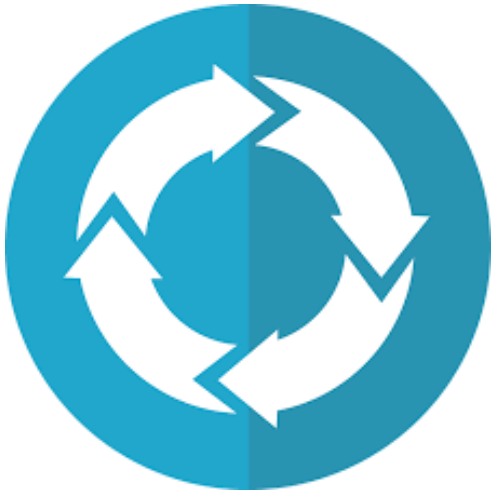
In the iteration review, at the end of every iteration, you demonstrate the working product created during the iteration to the product stakeholders

06

Team performs **iteration retrospective**

The iteration retrospective is a meeting where the team discusses how the iteration went and plans for improvements in the next iteration

Iteration Planning



01

Implement one or more **user stories**

Augments the existing system with code to implement new user stores. This is equivalent to implementing a set of requirements that is coherent around a use of the system

02

Remove a set of identified defects

Non-critical defects may be deferred to later iterations for resolution

03

Reduce a set of risks

Project risk is always about things that are not known. Risk are reduced through activities known as spikes than uncover important project information.

04

Target one or more **platforms**

Multiple platforms may be supported for the client, as well as development, testing and sandbox environments. Development platforms may include simulated or hand-built hardware.

05

Implements one or more **architectural aspects**

Architecture does not spring forth fully formed. It can be developed over a set of iterations

06

Results in **running software** which may be deployable to the customer

You only know about quality properties that you can verify.

Artifacts

Product Vision

The product vision is to define the essential properties and direction of the product

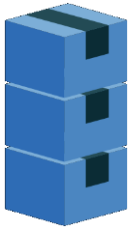
Product Backlog

The prioritized set of things to do including

- Use cases, user stories, and requirements
- Defects identified from previous iterations
- Experiments to perform to reduce project risk (spikes)
- Technical work items, such as technical infrastructure and planned refactoring

Product Roadmap

The roadmap is a high level, strategic view of the series of deliverable systems mapped to capabilities and customer needs, usually with a 12-24 month planning horizon.



Iteration Backlog

The set of work elements from the product backlog allocated to an iteration

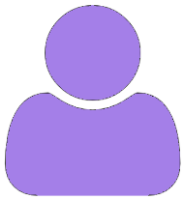
Release Plan

The release plan is a strategic plan, usually with a planning horizon of 3-9 months containing product backlog, features, and user stories.

Increment

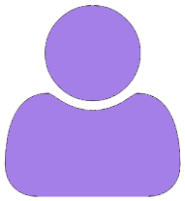
The system delivered at the end of an iteration, verified and validated against the set of user stories and requirements dealt with so far in the project.

Agile Roles



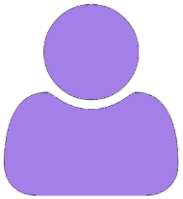
Product Owner

The person responsible for bridging the gap between the customer, business stakeholders, and the development team. The product owner is an expert on the product and the customer's needs and priorities.



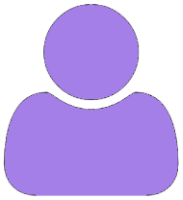
Project Facilitator

The person responsible for supporting the development team, clearing organizational roadblocks, and keeping the agile process consistent. May be known as **scrum master.**



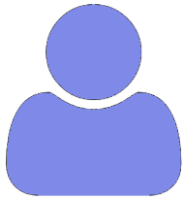
Agile Mentor

Someone who has experience implementing agile projects and can share that experience with a project team.



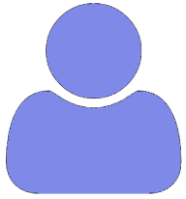
Stakeholder

Anyone with an interest in the project. Stakeholders are not ultimately responsible for the product, but they provide input and are affected by the project's outcome



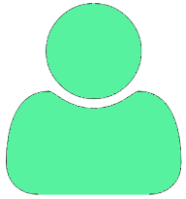
Architect

A person in charge of overall product design organization, technologies and approaches



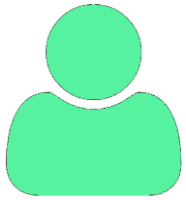
Developer

A person who designs, implements, and performs units/TDD product testing.



Integrator

A person in charge of integration of the work of multiple developers, using a practice known as **Continuous Integration**.



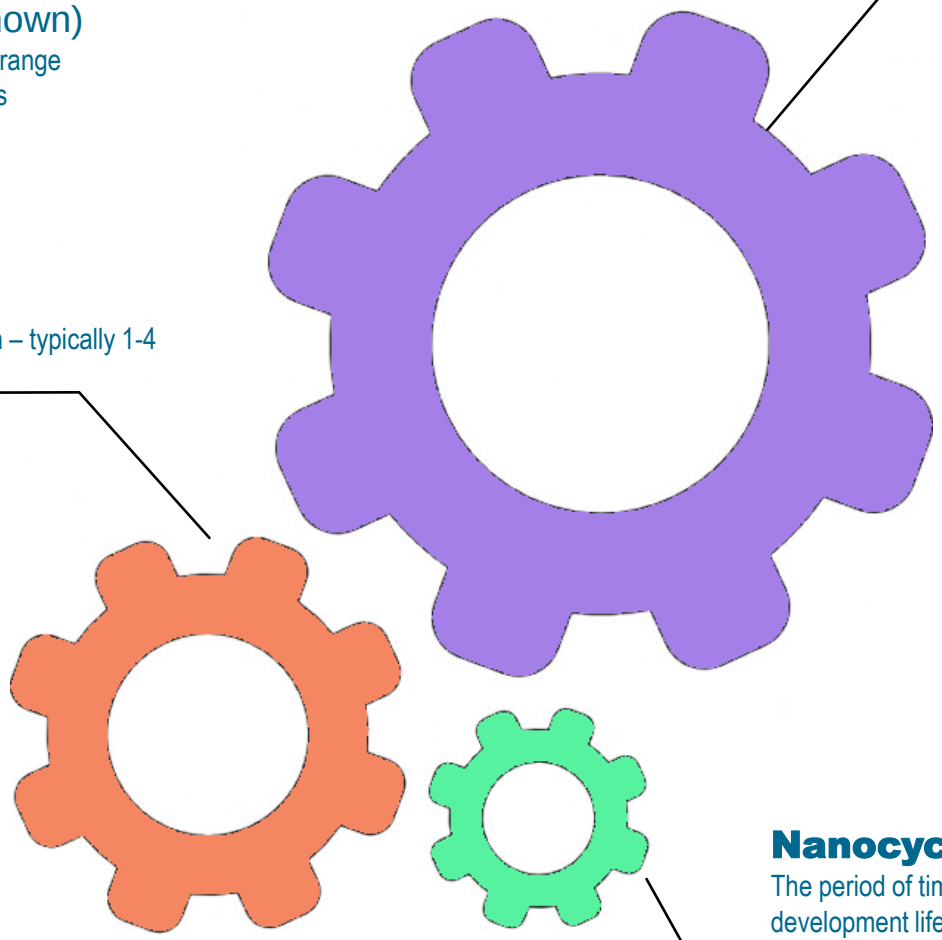
Validator

A person in charge of creation and execution of the integrated system, either incrementally or at the end of the iteration.

Agile Timescales

Macrocycle
The extent of a release plan for a product version, often 6-24 months

Megacycle (not shown)
Portfolio management and long-range portfolio planning often 2-5 years



Microcycle
The length of a single iteration – typically 1-4 weeks.

Nanocycle
The period of time between testing in a test driven development lifecycle, typically 30 minutes – 1 hour; also the time between integrations of a developing product, typically 1 – 8 hours.

Section

How to fail with Agility

Just because you want to do things in an agile fashion doesn't mean you will succeed

Lack of stakeholder involvement

- One of the key ideas in agile methods is to involve the stakeholders
  - For embedded products, the stakeholders often include
    - Marketing
    - Subject Matter Experts (SMEs)
    - Manufacturing
    - Customers (acquisitions)
    - Customers (end-users)
    - Testers
    - Maintainers
- The goal is to create value for the stakeholders
  - Clear, unambiguous, and consistent requirements are difficult to create
  - Requirements may not actually address the need (verification vs validation)

Being "Rigorously Agile"

- The *whole point* of agile methods is to be responsive to changing needs, conditions, and events and to adapt to them when necessary
- Understand that what works for a team of 5 building a web site might not work for a team of 50 building an aircraft engine
- Blindly sticking to an approach – no matter how "cool" – is anti-agile
- Being agile it not an excuse to stop thinking, evaluating, measuring, and changing
- Understand that while there are many more ways to fail than to succeed, there are still many ways to succeed and your job as a developer is to find a good way to satisfy your stakeholders needs in the most efficient, effective, and high quality way
- Examples
  - Some will claim requirements are not agile
  - Some deny the need to meet regulatory compliance standards as not agile
  - Declaring "That's not what <author> means by <term> so it must be wrong"

Not addressing project complexity factors

uses specialized hardware

Many engineering disciplines involved

often distributed teams across cultural boundaries

is often co-developed with the hardware

is often subject to fixed-price bids

constrains programming languages and tools

is delivered in a shipped, stand-alone product

challenging testing environment

is often subject to rigorous external regulation

must often be highly predictable

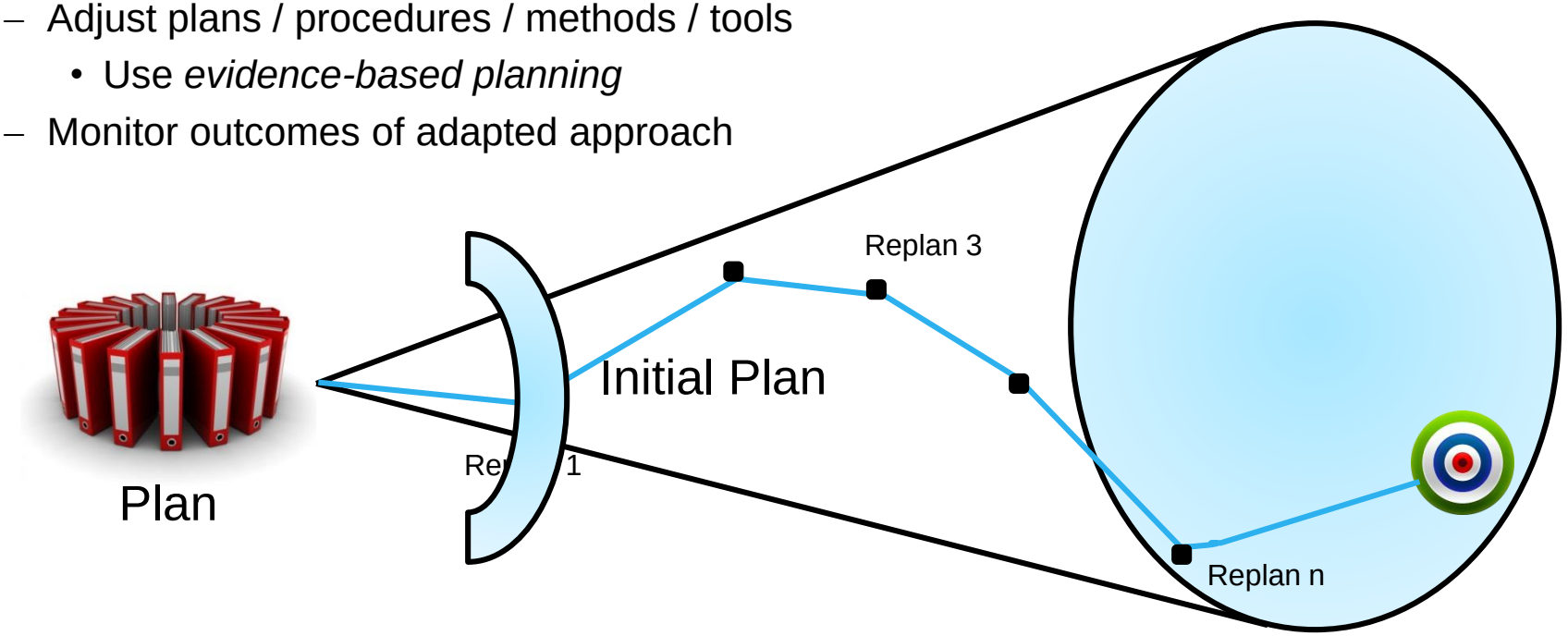
must often be very highly reliable and safety critical

is often subject to tight timeliness constraints

often has severe resource constraints

Failure to learn from mistakes

- Most organizations – agile or not – run an “open loop” (i.e. ballistic) fashion rather than adjusting based on “ground truth” (i.e. dynamic)
- To learn from mistakes
  - Monitor outcomes against plan (important)
  - Monitor outcomes against goal (more important)
  - Use root cause analysis to determine underlying problem and resolution
  - Adjust plans / procedures / methods / tools
    - Use *evidence-based planning*
  - Monitor outcomes of adapted approach



Not paying attention (metrics, people!)

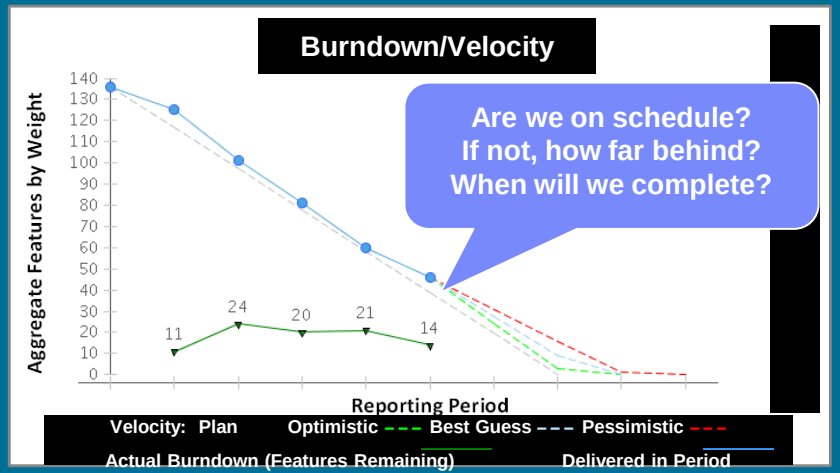
- The leading cause of project failure is not addressing project risk factors!
- Most of this is because people aren't measuring intermediate outcomes against goals

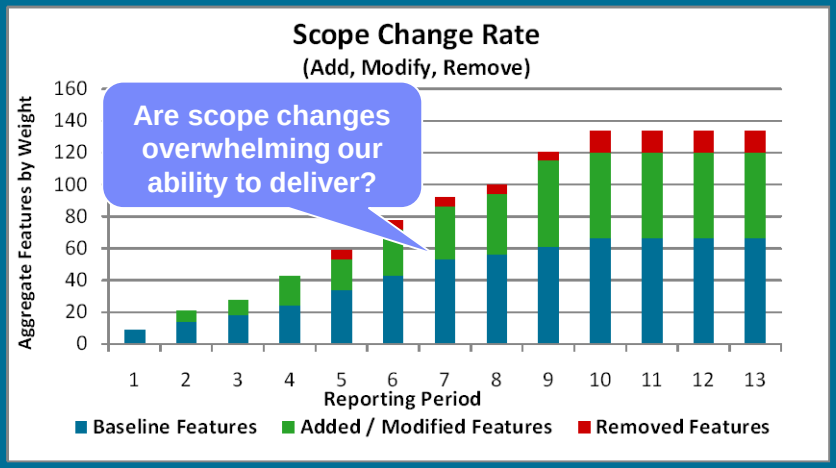
	Companies that measure	Companies that don't measure
On-time projects	75%	45%
Late projects	20%	40%
Cancelled projects	5%	15%
Defect removal	>95%	Unknown
Cost estimates	Accurate	Optimistic
User satisfaction	High	Low
Software status	High	Low
Staff morale	High	Low

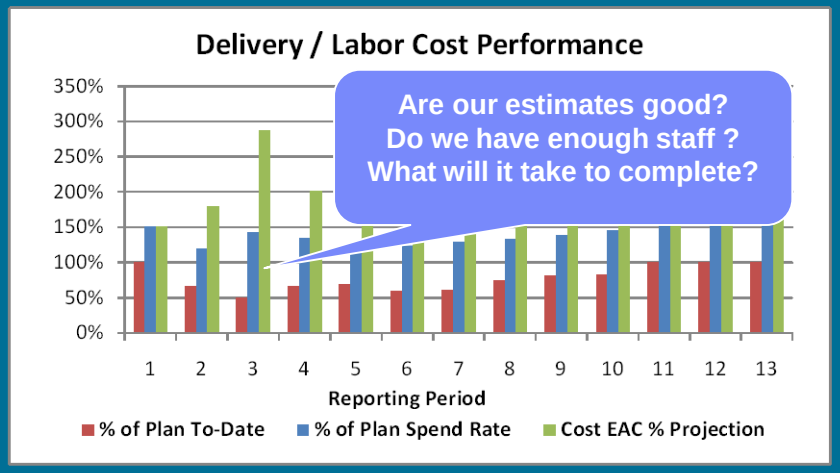
Source: Applied Software Measurement 3rd Edition by Capers Jones 2008

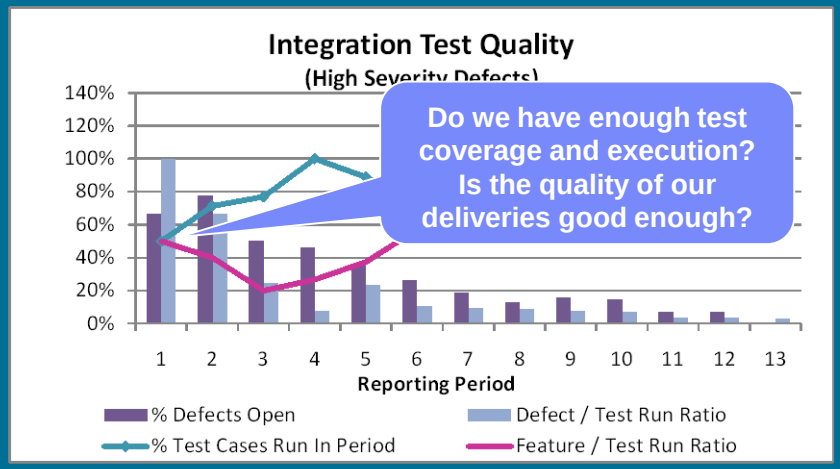
Not paying attention (metrics, people!)

(all reported by common reporting period)









Some Common Agile Metrics

Velocity

Progress measure per unit time, such as user stories per iteration. May be normalized by dividing by # of team members (productivity measure)

Burn Down Rate

Shows the reduction in the number of work items. or their accumulated story points. over time (completeness measure).

Done / Planned Ratio

Ratio of user stories completed / user stories planned for a set of iterations (adherence to plan measure)

Requirements Churn

Shows the rate of change and addition of new requirements, or user stories, as a function of time (stakeholder need stability measure)

Remaining Risk

Essentially a burn down chart for risk exposure over time, where risk exposure for each risk in a Risk List is assigned a number or categorical value (such as low, medium, high) (risk measure)

Team Happiness

Monitors the ability of a team to continue working at their current velocity.. A team that meets their productive plan but is unhappy is likely being burned out (sustainability measure)

Open Defect Age

Tracks the average age of defects in the work item list, measured from when they are discovered until when their resolution has been verified. (responsiveness measure)

Running Tested Features

Number of customer-defined system features that have verified

Escaped Defects

Number of defects reported by stakeholders (Quality measure)

Complexity

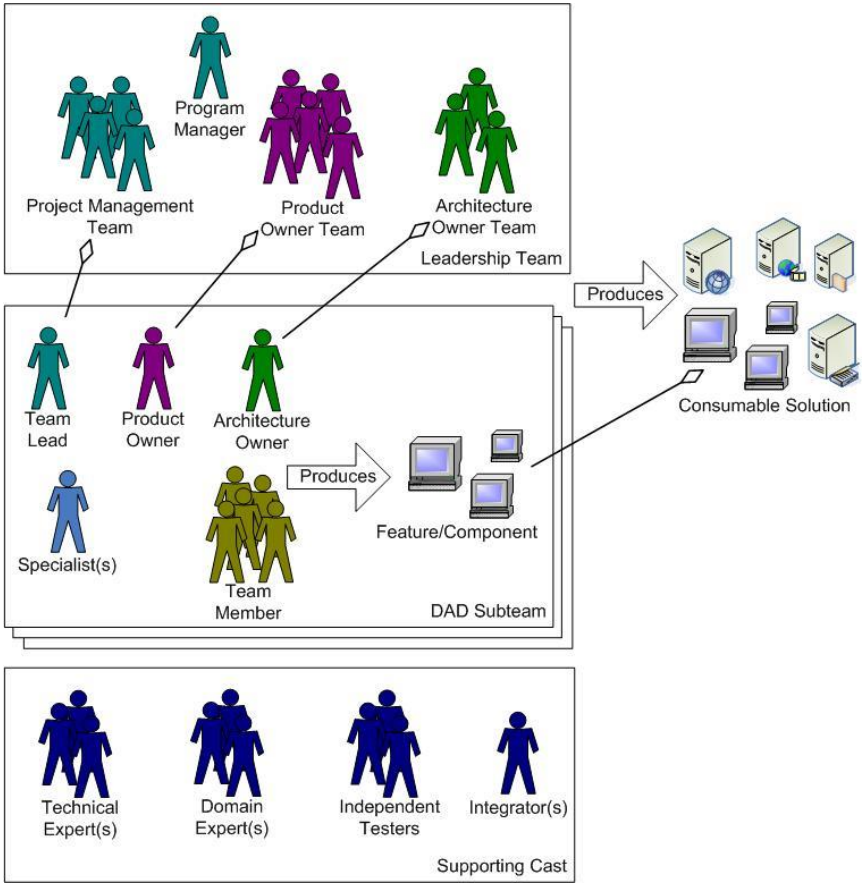
Measured complexity of the software, such as with McCabe Cyclomatic Complexity (maintainability measure).

Defect Density

Measured number of defects normalized per unit size (KLOC or modules, commonly) (quality measure)

Poor communication

- Embedded system development is a cooperative activity among a potentially large number of team members
- Critical communications include
  - Customer
  - Project Manager / product owner
  - Architect
  - Software engineer
  - Electronics engineer
  - Mechanical engineer
  - Systems engineer
  - Subject Matter Expert (SME)
  - Technical expert
  - Integrator
  - Independent tester



Ways to Communicate

Design

- Who: Architects, developers
- Why: capture design decisions, support analysis, understand design implications

Stand Up Meeting

- Who: team members
- Why: share progress and impediments

Documentation

- Who: developers, tech writers, customers
- Who: architects, developers
- Why: User docs: tutorial, reference; Product docs: developers, architects, maintainers

Planning Meeting

- Who: project manager/product owner, team members, customer
- Why: share information about project and iteration plans, schedules, and estimates

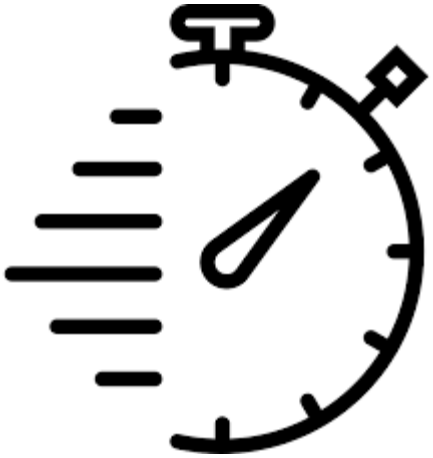
Customer demo, released build

- Who: customer, project manager/product owner, team members
- Why: deliver value to the customer, validate value to customer

Iteration Retrospective

- Who: project manager/product owner, team members
- Why: review outcomes against plan, outcomes against goals, perform gap analysis, identify risk factors, modify plan based on “ground truth”, update procedures, processes, and tools to improve outcomes

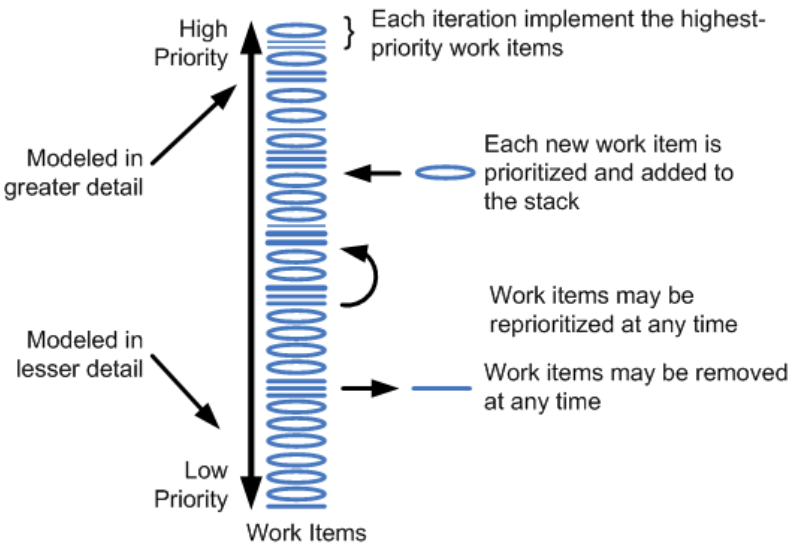
Poor estimation



- Systems development is inherently difficult to estimate because
  - It is about *invention* of things that do not yet exist
  - We do not have all the information we need
  - We do not have infinite fidelity of the information we do have
  - Some of what we know will change
  - Some of what we know is wrong
- Plans and schedules are *theories*
  - Theories need to be proven based on actual evidence
- Understand that estimates always contain error
- Absolute estimates (e.g. hours) allow costing to be done
- Relative estimates (e.g. story points, use case points) allow relative timing & costing to be done
  - Relative estimates can be converted to absolute if the appropriate velocity metric is available)

Iteration planning estimation

- Performed by the whole team, not just the team lead, at the beginning of each iteration in a self organizing manner
- Effectively just in time (JIT) detailed planning for the iteration
- Determine team velocity
  - Actual number of points completed in previous iterations / hours expended
- Estimate points per work item
  - Already done as part of Release Planning, but should revisit for the ones being addressed this iteration
- Estimate effort
  - Individuals sign up for tasks
  - Break work items into smaller tasks
  - Estimate number of actual hours to complete the tasks
  - Calibrate estimation using previous iterations' velocity and effort estimation
- Determine iteration scope
  - Adjust iteration objectives based on estimates



Poor testing

- The primary reason for teams to move to agile methods is to improve quality and testing is one of the key ways this is achieved
- Typical problems with testing in agile projects
  - Test Driven Development (TDD) is not employed
  - Integration among software units is deferred
  - Integration with other disciplines is deferred
  - Verification is not done by an independent team
  - Tests inadequately verify the System under Test (SUT) in terms of
    - Functionality
    - Performance
    - Out of range
    - Data equivalency class coverage
    - Robustness (Invariant violation)
    - Coverage (e.g. structural, decision, and modified condition/decision)

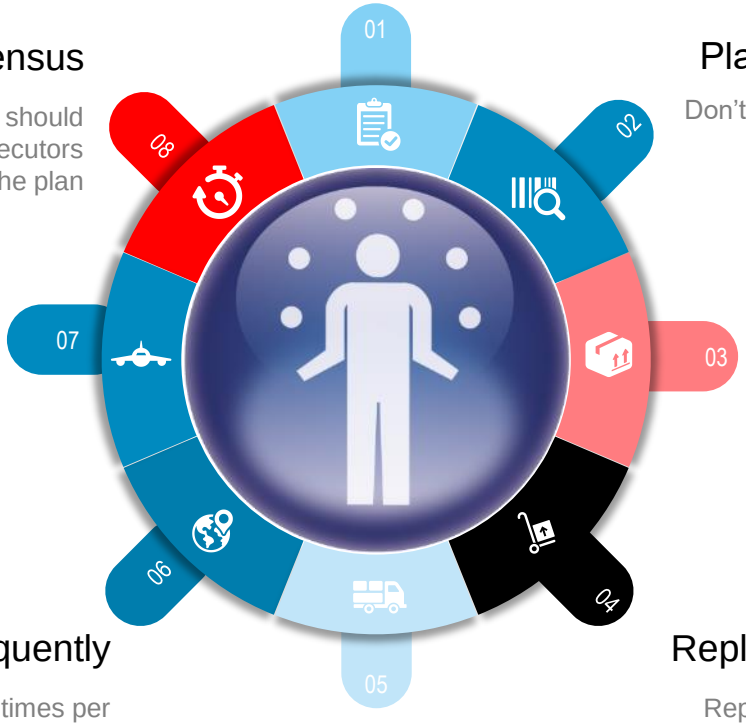
Ignoring risks

- Ignoring risks is the leading cause of project failure
  - In agile terms, risks are sometimes called “impediments”
- Risks are all about *what you do not know*
- Risks are addressed with “spikes” (risk mitigation activities) that uncover information to reduce project risk
- The outcome of these spikes are
  - Reviewed during Iteration Retrospective
  - Used to adjust plans, designs, and architectures (depending on the risk)
- **Risk List** (aka “risk management plan”) identifies the risk, its severity and likelihood
- **Spikes** are allocated to iterations (as work items) usually in a highest-risk-first fashion

Lack of Planning

- Some people think agile is about avoiding planning.
  - This is *not true!*
- Agile planning
  - Planning is not done beyond the scope and fidelity of knowledge
  - The **Release Plan** allocates key requirements, risks, and work items to iterations, and concerns their planned start, completion and efforts
  - An **Iteration Plan** is done *just in time* and identifies a more detailed plan to just the next iteration
  - At the end of the iteration, the outcomes are used to adjust the **Release Plan**
  - The result is a *steered* project that converges on the solution even though it started with incomplete information

Key Agile Planning Concepts



Be open to change

A key principle of Agile is to “embrace change”

Plan “just enough”

Don’t plan beyond your level of information

Plan “just in time”

Plan just before you need it to be sure your information is an accurate and complete as possible

Replan frequently

Replan as evidence is collected so that plans are as accurate and complete as possible

Decide how to measure success

Measure success against goals of the plan

Measure frequently

Measure multiple times per task scope.

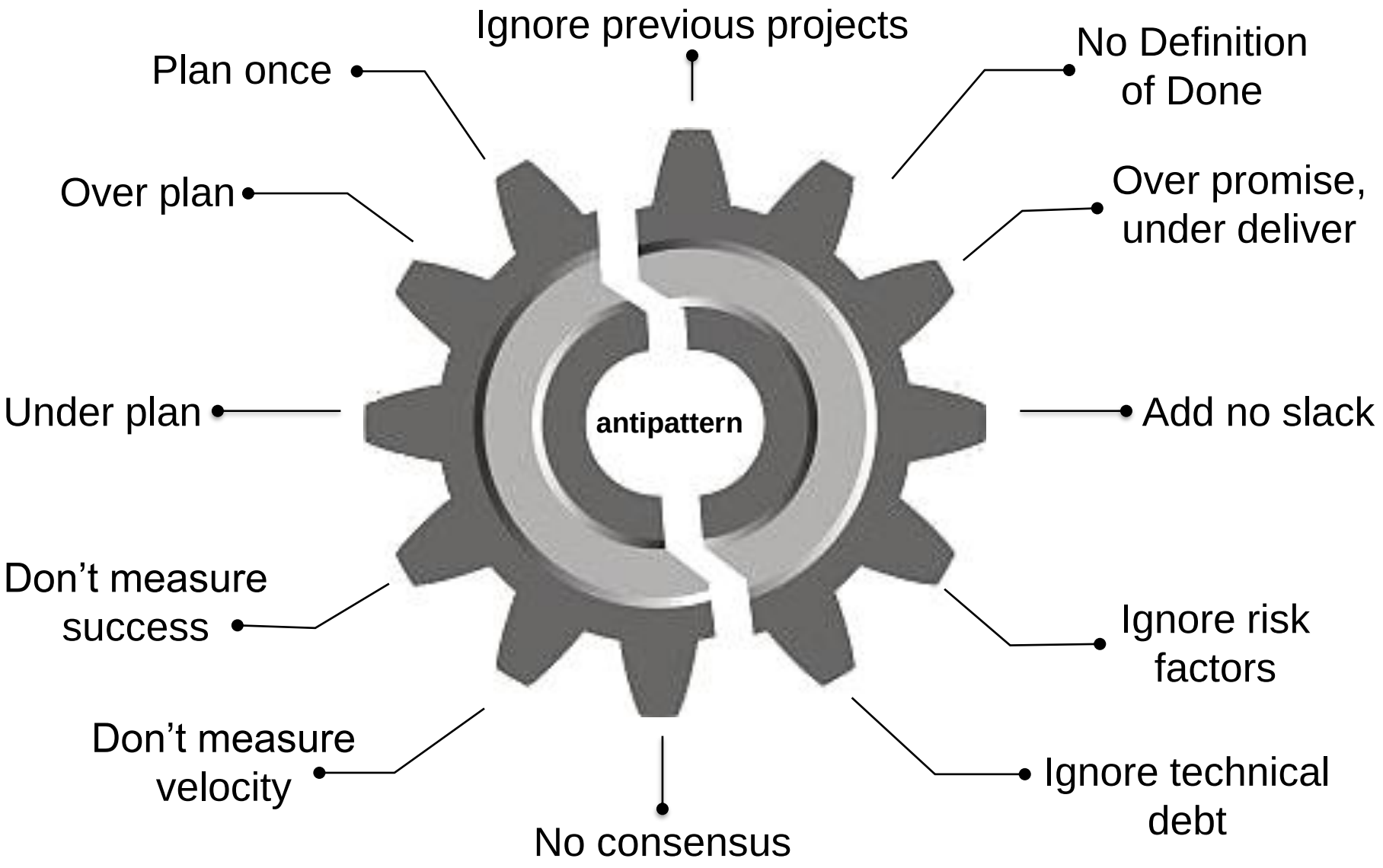
Time box or feature box

This ensures both progress and allows collaboration.

Achieve consensus

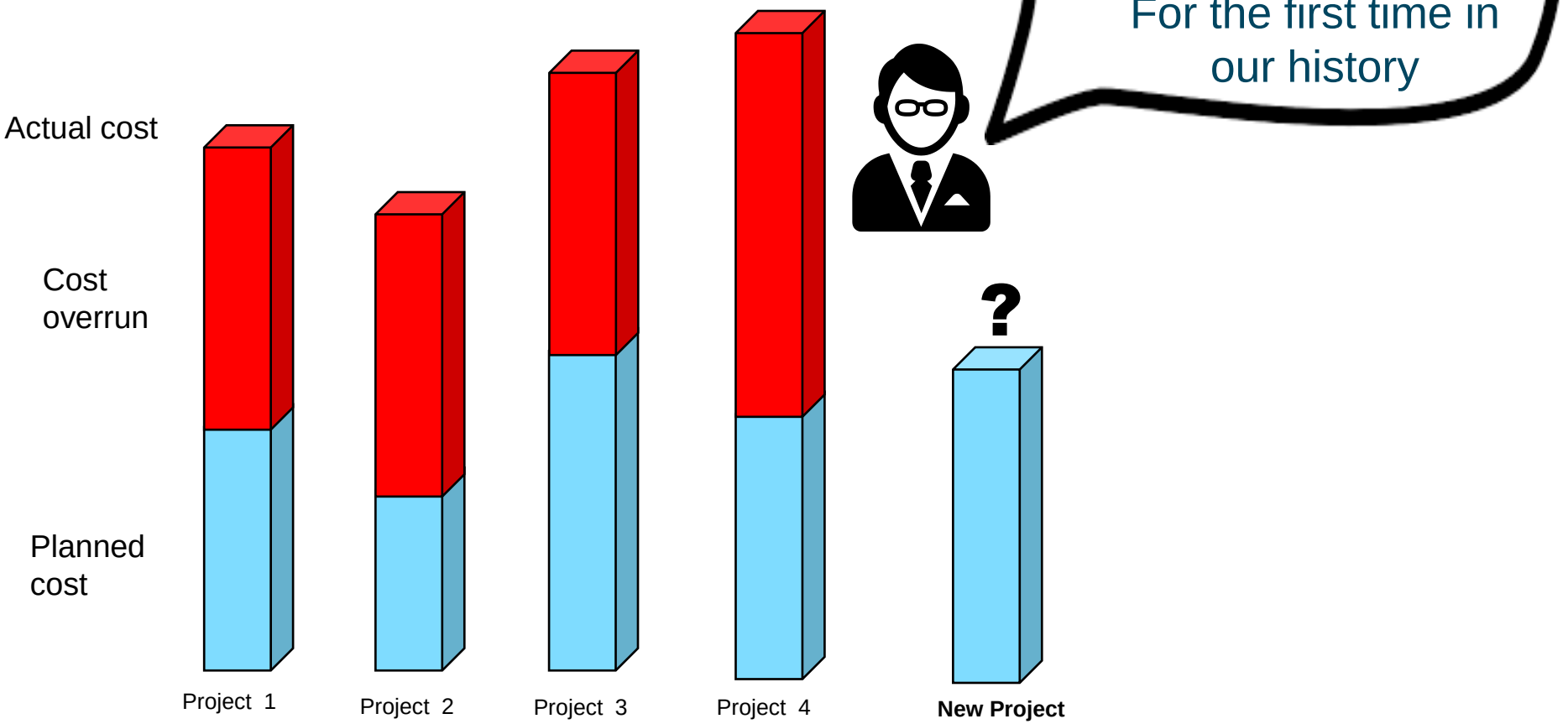
All involved parties should agree, including the executors of the plan

Planning anti-patterns

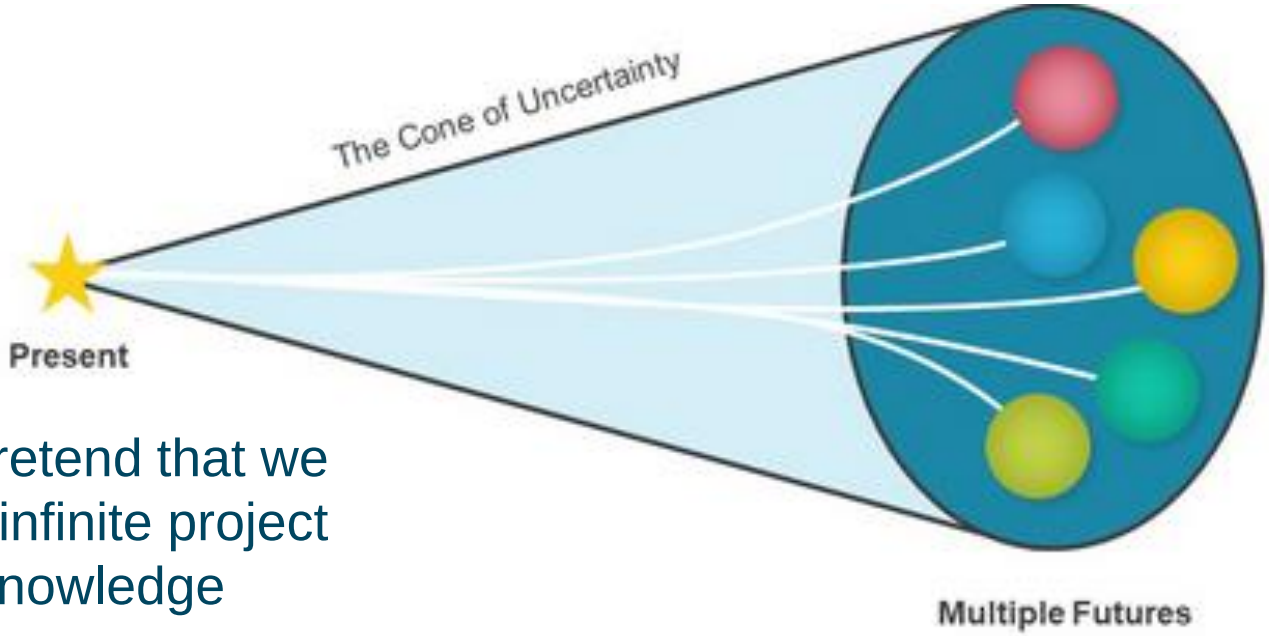


The 3rd most common mistake in planning

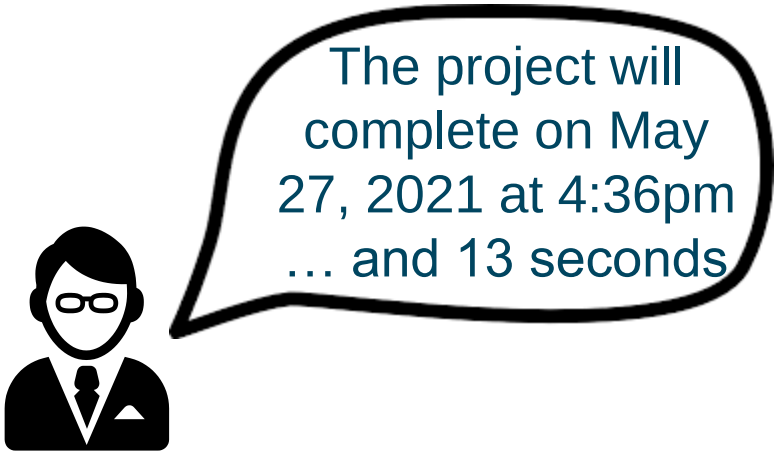
We ignore previous project outcomes



The 2nd most common mistake in planning



We pretend that we have infinite project knowledge



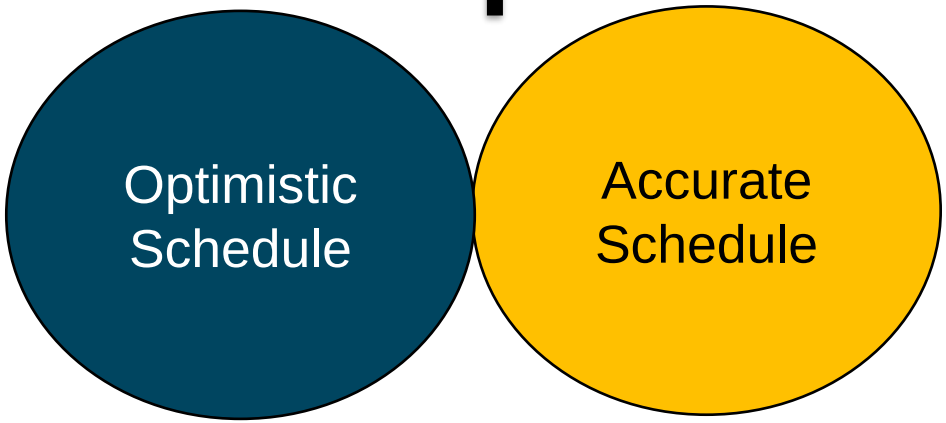
But really, we're estimating things we really don't know

The most common mistake in planning

A schedule may be use to motivate workers (this is known as an "optimistic schedule"

OR

A schedule may be use to accurately describe project cost, effort and time

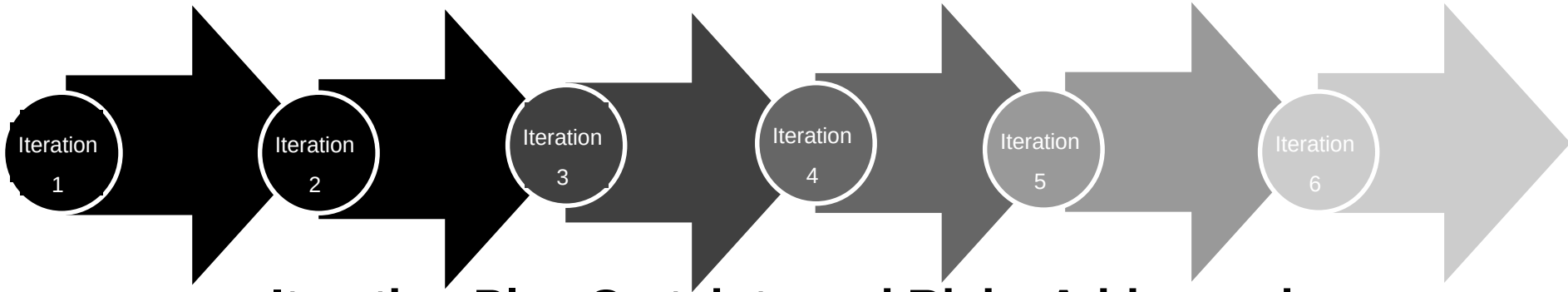


Motivational – Accurate schedule Venn diagram

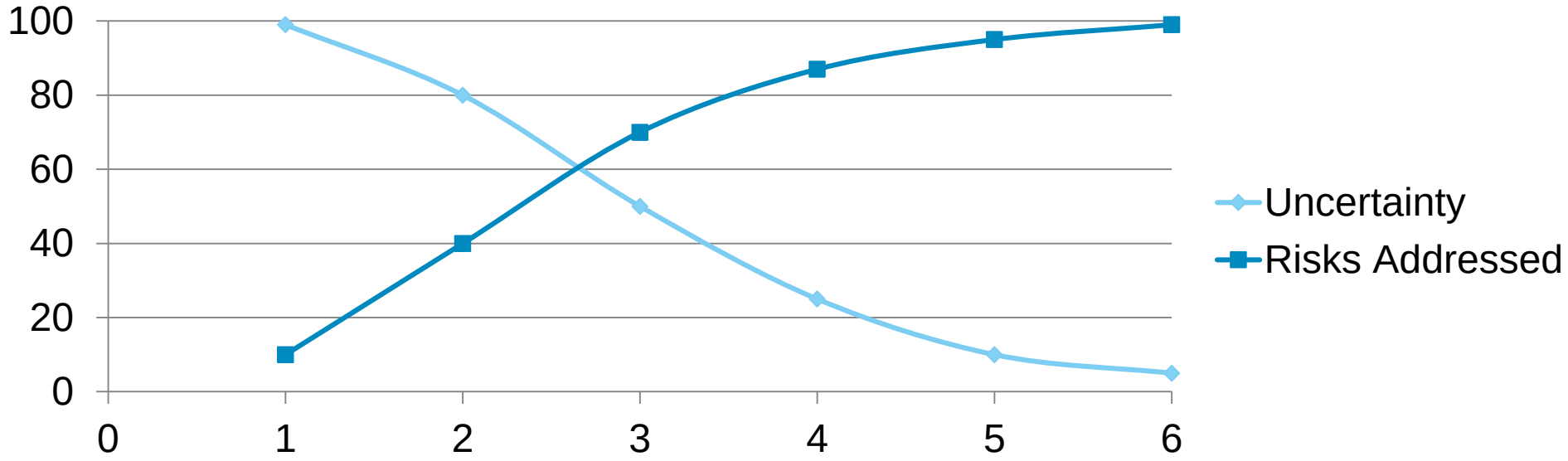
Optimism is the enemy of realism

Two Tier Planning: Release Plan

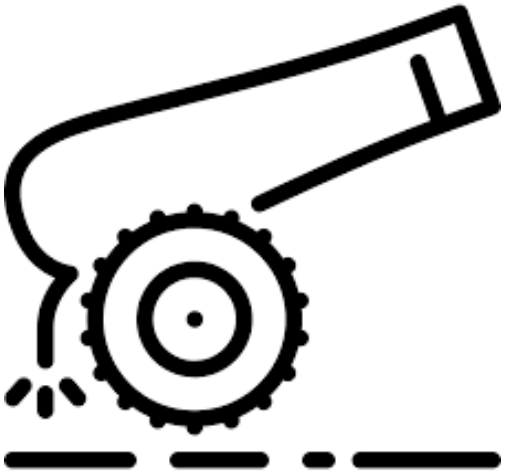
Release Plan



Iteration Plan Certainty and Risks Addressed



Poor agile adoption plan

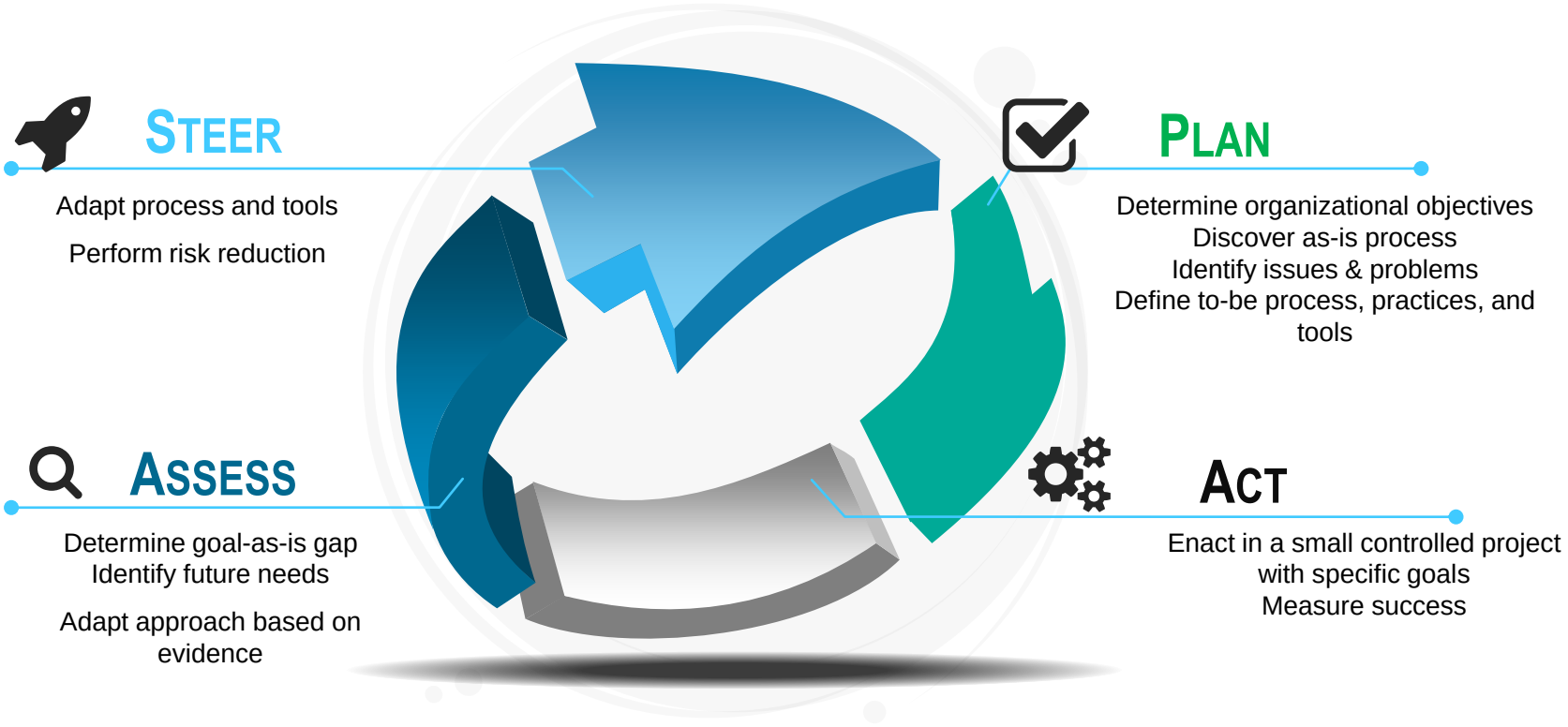


- Some people use *ballistic methods* to adopt agile, leading to
  - Solving the wrong problem
  - Breaking things that worked well before adoption
  - Not solving the problem well
  - Lack of instrumentation (metrics) prohibits customization/adaptation of methods to best meet organizational needs
  - Increase resistance on the part of engineering or management
  - Local optimization at the expense of the overall organization
  - Failed projects

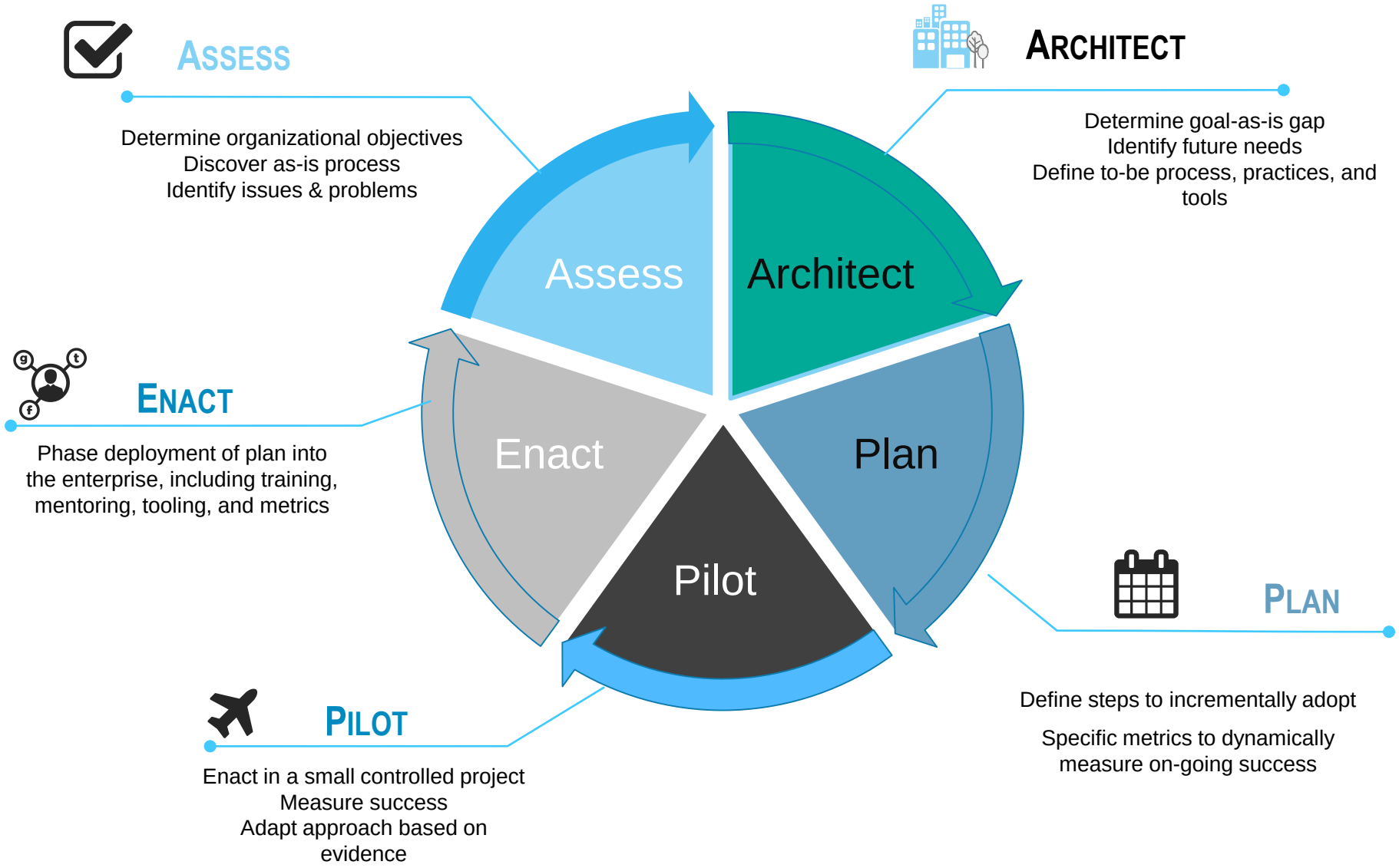
Poor agile adoption plan

- Tip: Use agile to adopt agile
  - Understand the needs
  - Create an incremental plan to meet the needs
  - Instrument the plan with success metrics
  - Steer the adoption
- Specifically
  - Capability Assessment
    - This step performs detailed interviews with all related technical and managerial staff and review of all organizational and project documentation to establish both strengths and weaknesses of the organization
  - Vision & Goal Establishment
    - This activity establishes the organizational goals and priorities them
  - Key Recommendations
    - Based on the previous steps, this activity identifies the necessary skills, processes, and technologies to achieve the vision and goals of the organization and the tasks required to achieve them at the necessary performance levels
  - Phased Plan Development
    - A detailed plan based on the previous steps is constructed, loaded with dates, staff, tooling, milestones, and success metrics

Use Agile to Adopt Agile



Use Agile to Adopt Agile



Download Papers, Presentations, Models, & Profiles for Free