Outline

• Examples of Anti-Patterns
• Anti-Patterns on
  – New Agile Projects
  – Product Backlog Management and Requirements Management
  – Architecture and Design
  – Testing
  – Software Development Team
  – Project Management
Definition

Anti-Pattern:

“Antipatterns are common solutions to common problems where the solution is ineffective and may result in undesired consequences. An antipattern is different from bad practice when:

• It is a common practice that initially looks like an appropriate solution by ends up having bad consequences that outweigh any benefits
• There’s another solution that is known, repeatable, and effective.
• The concept of antipatterns was inspired by the concept of design patterns, which indicate common effective solutions to common problems.

Antipatterns were initially applied in the context of software development, but have extended to other aspects of software engineering, organizations, and project management.

Coaches and consultants like to invoke antipatterns as a way of pointing out behavior they often see in teams they coach and as an avenue of suggesting better patterns.”

Ref: https://www.agilealliance.org/glossary/antipattern
Examples of Anti-Patterns

Organizational
• **Analysis paralysis:** A project stalled in the analysis phase, unable to achieve support for any of the potential plans of approach
• **Groupthink:** A collective state where group members begin to (often unknowingly) think alike and reject differing viewpoints
• **Micromanagement:** Ineffectiveness from excessive observation, supervision, or other hands-on involvement from management
• **Mushroom management:** Keeping employees "in the dark and fed manure" (also "left to stew and finally canned")
• **Seagull management:** Management in which managers only interact with employees when a problem arises, when they "fly in, make a lot of noise, dump on everyone, do not solve the problem, then fly out"
• **Vendor lock-in:** Making a system excessively dependent on an externally supplied component

Project management
• **Cart before the horse:** Focusing too many resources on a stage of a project out of its sequence
• **Death march:** A project whose staff, while expecting it to fail, are compelled to continue, often with much overwork, by management which is in denial
• **Ninety-ninety rule:** Tendency to underestimate the amount of time to complete a project when it is "nearly done"
• **Overengineering:** Spending resources making a project more robust and complex than is needed
• **Scope creep:** Uncontrolled changes or continuous growth in a project's scope, or adding new features to the project after the original requirements have been drafted and accepted (also known as requirement creep and feature creep)
• **Brooks's law:** Adding more resources to a project to increase velocity, when the project is already slowed down by coordination overhead.

Ref: https://en.wikipedia.org/wiki/Anti-pattern#Examples
Anti-Patterns on New Agile Projects (1/3)

• “Do Agile” vs “Being Agile”
  – Going through motions without understanding what intended outcomes should be
    • Task 1 for requirements gathering, producing draft and final SRS IAW with IEEE std 830-1998; after three months, start Task 2, build product backlog, develop personas and user stories, sprint cadence etc.
    • SOW states the contract shall follow Agile methodology, shall define sprint cadence, etc. AND at kickoff, define the detailed capabilities and services to be delivered at the end of each sprint
    • Design review for each sprint
    • Death in CDRLs
    • Fluid Sprints – no time box or extended sprint length
    • Not planned well
    • Expectations that stories can move in and out or that sprint stays open if work is not completed
  – Recommendation
    • Be Agile, follow Agile manifesto values, study Agile lessons learned
Anti-Patterns on New Agile Projects (2/3)

• **Scrum-but**: “we’re doing Scrum, but we…” [do something that is completely the opposite of what it says to do in Scrum]
  – Examples: Extensive up-front design, Large User Stories (all use cases covered), Lots of hand-offs (versus cross-functional team)

• **Agile-on-the-fly**: If teams are new to Agile, it’s recommended that they adopt it properly first, then try and experiment with it once they’ve got the hang of it.

• **Recommendation**
  – Need training, mindset change, full team (including management) buy-in, on-going coaching
Anti-Patterns on New Agile Projects (3/3)

• Assuming agile planning is entirely ad hoc and as-you-go
  – Agile planning is intended to be flexible, but not chaotic
  – One program planned “Releases,” but had only the vaguest notion of what those Releases would contain
    • Resulting in lack of structure and priority in their sprint planning
  – **Recommendation**
    • Planning should be oriented to achieve a Minimum Viable Product or Minimum Operational Capability (focus on delivering features or capabilities rather than functional components) then enhance that capability incrementally in subsequent sprints and releases
    • NEVER be in a state where the product isn’t working. Plan EVERY iteration to enhance and deliver additional mission capabilities.

• Not building in quality
  – Not enough testing, especially regression testing, preferably automated regression testing
  – **Recommendation**
    • Start with the end in mind, use acceptance criteria, definition of done
    • Embed testers as part of the development team
Anti-Patterns on the Product Backlog Management or Requirements Management (1/3)

• Time/effort wasters
  – Not spending time during backlog management - impacts Sprint Planning efficiency
  – Too many items or items too old – clutter and difficult to prioritize
  – Review/estimate everything (and too early) - unnecessary effort by team
  – Too much information or acceptance criteria – leave room for discussion with the team for new perspectives and negotiation on scope; leaves team less engaged if everything is spelled out
  – Recommendation
    • Organize backlog grooming sessions to ensure that items are ready for next Sprint
    • Review/estimate the top priority items that are likely to be addressed in the next 1-2 sprints
Anti-Patterns on the Product Backlog Management or Requirements Management (2/3)

• Prioritization issues
  – Prioritization by proxy – someone else (external to Product Owner) dictates the priorities; no accountability of Product Owner
  – Prioritize full project up front
  – **Recommendation**
    • Appoint a Product Owner with authority
    • Priorities should expect to adjust as you observe working software and assess how much is enough
Anti-Patterns on the Product Backlog Management or Requirements Management (3/3)

• Failing to organize and prioritize the backlog by feature
  – *Item is horizontal component versus end-to-end feature*
    • May lead to completed components, but not operational or not delivering mission value
• Composition of a backlog item
  – *Items not decomposed from Themes or Epics*
• Assign a non-functional requirement to be developed in one Sprint
  – *E.g. scalability requirement can not be achieved in one Sprint*

• **Recommendation**
  – *Factor in or bake in non-functional requirements from day one*
  – *Start from requirements decomposition. Organize requirements in a capability-driven structure*
  – *Helpful to be able to filter by an epic and see what the features have been defined and help to convey scope of a particular release*
Anti-Patterns on Architecture and Design

• Architecture and Design are somewhat orthogonal to development methodologies
  – *Agile doesn't work well with stovepipes or monolithic hardware/software systems*
  – **Recommendation**
    • Modularity, layered architecture, abstracted dependencies
    • Small, self-contained, testable feature decomposition
    • Start with Just Enough Architecture

• Several space and ground software are developed by engineers from other domains without software engineering background
  – **Recommendation**
    • Be cognizant of technical debt
    • Start by defining some principles, tenets and architectural decisions upfront

• Lack of Government-owned software architecture
  – **Recommendation**
    • Define interface specifications
Anti-Patterns on Testing

• Accepting manual testing as suitable and effective
  – Recommendation
    • Design for test
    • Use Test-Driven Development at the unit level
    • Use Behavior-Driven Development at the integration level
    • Continue to enhance automated functional tests throughout the lifecycle

• Accepting automated unit testing covers everything
  – Recommendation
    • Need a good balance between automated and manual testing

• Not integrating Test with Development
  – Recommendation
    • Test as you go, continual testing, automated regression testing
Anti-Patterns on the Software Development Team

• Product Backlog Grooming done privately
  – **Recommendation**
  • Full team must be involved to ensure a shared understanding of the “why” and “what” since anyone on the team should be able to pick up a story and work on it

• Fill Sprint Backlog with 100% of the team capacity
  – **Lead to “I’m busy” attitude and no time to help others**
  – **Recommendation**
  • Leave room for collaboration and team support

• Recommend to identify a “Story Shepherd” who ensures the tasks and task dependencies are clear and moving along within the sprint
Anti-Pattern on Project Management (1/2)

• Planning failures
  – *Pre-planning iterations as though using traditional planning (IMS)*
  – *Failing to plan releases in terms of expected features/capabilities*
  – *Creating a too-detailed IMS, and evaluating progress (and EV) against it*
  – **Recommendation**
    • May continue to use high-level IMS
    • Use more of Product Roadmap and Architecture Runway
      – *Consider dependencies and constraints when developing Release Plan*

• Team organization failures
  – *Teams organized by functional decomposition, rather than by Feature*
  – **Recommendation**
    • Encourage multi-disciplinary teams organized by Feature/Epic
    • Focus on flat organization structure to speed up decision-making process

EV – Earned Value; IMS – Integrated Master Schedule
Anti-Pattern on Project Management (2/2)

• Tasks too large that it sits in “In Progress” the whole sprint
  – “Yesterday I worked on X and today I plan to work on X”
    • Team doesn’t really know what you are doing or whether progress is being made
    • It’s not apparent if you are blocked or stuck on a problem that possibly someone else can help with
    • Impacts dependent subtasks and likely the larger story
  – **Recommendation**
    • Scrum master should pay attention to progress, shared vision, and potential impediments

• Ineffective retrospective
  – *No action or follow-up taken to remedy issues*
    • Repetition of issues, deteriorating morale
  – *Team less likely to raise concerns if they feel nothing will change*
  – **Recommendation**
    • Team should identify actionable, measurable, and controllable items
    • Put action items in the backlog
    • Review past action items with the team