126 strategies to implement practices to avoid 'Agile in name only' and 'Build better / risk less'

Based on Beyond Legacy Code: Nine Practices to Extend the Life (and Value) of Your Software.

Say What, Why and Whom before How

7 strategies for Product Owners:
- Be the SME
- Use development for discovery
- Help developers understand why and for whom
- Describe what you want, not how to get it
- Answer questions quickly
- Remove dependencies
- Support refactoring

7 strategies for writing better stories:
- See it as a placeholder
- focus on the 'What'
- Personify the 'Who'
- Know why a feature is wanted
- Start simple and add enhancements later
- Think about edge cases
- Use acceptance criteria

Build in small badges

7 strategies for measuring software development:
- Measure time-to-value
- Measure time spent coding
- Measure defect density
- Measure time to detect defects
- Measure customer value of features
- Measure cost of not delivering features
- Measure efficiency of feedback loops

7 strategies for splitting stories:
- Break down compound stories into components
- Break down complex stories into knowns and unknowns
- Iterate on unknowns until they're understood
- Split on acceptance criteria
- Minimize dependencies
- Keep intensions singular
- Keep stories testable

Integrate continuously

7 strategies for agile infrastructure:
- Use version control for everything
- One-click build end-to-end
- Integrate continuously
- Define acceptance criteria for tasks
- Write testable code
- Keep test coverage where it is needed
- Fix broken builds immediately

7 strategies for burning down risk:
- Integrate continuously
- Avoid branching
- Invest in automated tests
- Identify areas of risk
- Work through unknowns
- Build the smallest pieces that show value
- Validate often

Collaborate

7 strategies for pair programming:
- Try it, you'll like it
- Engage driver and navigator
- Swap roles frequently
- Put in an honest day
- Try all configurations
- Let teams decide on the details
- Track progress

7 strategies for effective retrospectives:
- Look for small improvements
- Blame process, not people
- Practice the five whys
- Address root causes
- Listen to everyone
- Empower people
- Measure progress

Create clear code

7 strategies for increasing code quality:
- Get crisp on the definition of quality
- Share common quality practices
- Let go of perfectionism
- Understand trade-offs
- Hide 'How' with 'What'
- Name things well
- Keep code testable

7 strategies for writing maintainable code:
- Adopt collective code ownership
- Refactor enthusiastically
- Pair constantly
- Do code reviews frequently
- Study other developers' styles
- Study software development
- Read code, write code, and practice coding

Write the test first

7 strategies for great acceptance tests:
- Get clear on the benefits of what you're building
- Know who it's for and why they want it
- Automate acceptance criteria
- Specify edge cases, exceptions, and alternative paths
- Use examples to flesh out details and flush out inconsistencies
- Split behaviours on acceptance criteria
- Make each test unique

7 strategies for great unit tests:
- Take the caller's perspective
- Use tests to specify behaviour
- Only write tests that create new distinctions
- Only write production code to make a failing test pass
- Build out behaviours with tests
- Refactor code
- Refactor tests

Specify behaviours with tests

7 strategies for using tests as specifications:
- Instrument your tests
- Use helper methods with intention-revealing names
- Show what's important
- Test behaviours, not implementations
- Use mocks to test workflows
- Avoid over specifying
- Use accurate examples

7 strategies for fixing bugs:
- Don't write them in the first place
- Catch them as soon as possible
- Make bugs findable by design
- Ask the right questions
- See bugs as missing tests
- Use defects to fix process
- Learn from mistakes

Implement the design last

7 strategies for doing emergent design:
- Understand Object-Oriented design
- Understand design patterns
- Understand Test-Driven Development
- Understand refactoring
- Focus on code quality
- Be merciless
- Practice good development habits

7 strategies for cleaning up code:
- Let code speak for itself
- Add seams to add tests
- Make methods more cohesive
- Make classes more cohesive
- Centralize decisions
- Introduce polymorphism
- Encapsulate construction

Refactor legacy code

7 strategies for helping you justify refactoring:
- To learn an existing system
- To make small improvements
- To retrofit tests in legacy code
- Clean up as you go
- Redesign an implementation once you know more
- Clean up before moving on
- Refactor to learn what not to do

7 strategies for when to refactor:
- When critical code is not well maintained
- When the only person who understands he code is becoming unavailable
- When new information reveals a better design
- When fixing bugs
- When adding new features
- When you need to document legacy code
- When it's cheaper than a rewrite

Henny Portman, Sept 2015, https://hennyportman.wordpress.com