Prerequisites	Basics	Foothold	Accelerating	Pervasive	Mastery
Foundational items required.	Basic concepts that take significant steps forward.	Return on investment tipping point reached, driving continuing advancement	Cost of change notably reduced.	Cost of practices reduced due to continual and pervasive application.	Continuous delivery fully enables business agility.
Management Support o Slack for improvement o Iterative and incremental design over BDUF o Delaying design abstractions o Business & IT Tools o Source control o In Process Testing o User Interface Testing o CI Server Object Orientated Concepts o Abstraction o Encapsulation o Polymorphism o Inheritance Version Control o Everything o Frequent check in o Check in comments o Checking into main o Merging conflicts o Short lived branches Automated Builds o IDE and command line o Using source control o Dependency management o Everyone using Environments o Reasonably identical, representing production o Setup is documented o Includes: o Developer workstation o Integration o Test, staging, … o Production	Only write small incremental code that is o Needed to satisfy the current test o Simplest solution for current test Spike to first understand solution, then concentrate on test driving Write tests for o Primary case o Else case(s) o Boundary conditions o Exceptions Automated Testing o Red, Green, Refactor o Given input, expect output o Side effects (state) o Replacing manual testing o Using Test Doubles Story Delivery o Push to be very small o Maintain ability to demonstrate business value for each story and functionally test o Implement only defined acceptance criteria o Create separate tech (task) cards if necessary (no velocity) Design Principles o Descriptive Naming o Boy Scout Rule Code Smells & Refactoring o Stepwise refactoring under tests Design Collaboration o Whiteboard design sessions o “We” instead of “I” o Closely working together and sharing experience Continuous Delivery o Continuous integration as a practice, not a tool o Deployment is documented	Paying attention to code smells and refactoring to maintain clean code. Automated Testing o All new code under test o Test First used as verification o Interaction Points using Mocks o FIRST rule for tests including UI based tests o Separate exploratory testing (not static) Story Delivery o As proficiency grows iterate splitting stories into ½ to 2 day cycle times. Design Principles o Single Responsibility o Don’t Repeat Yourself o Delaying decisions, YAGNI o Dependency injection o Composition over inheritance o Interfaces over concrete implementation Code Smells & Refactoring o Duplicated code o Long method o Large class o Associated refactoring patterns Legacy Code o Characterization tests Design Collaboration o Collective code ownership o Trust team mates o Pairing API Design o Cross team collaboration Design Patterns o Speculative generality smell Continuous Delivery o CI automates source control, build, in process tests, and code coverage o Testing complete within sprint (dev, QA, business)	Caring for medium level design details and how to refactor to them. Automated Testing o Tests influence design by listening to “pain”; adjusting design to simplify testability o Clean tests o Client side/JavaScript o Database o Test pyramid “flipped” from UI/integration tests to unit tests o Acceptance criteria written as verifiable tests in business language Design Principles o Emergent design o Separation of concerns o Law of Demeter Code Smells & Refactoring o Comments o Long parameter list o Message chains o Feature envy o Data clumps` o Inappropriate intimacy o Associated refactoring patterns Design Collaboration o Enterprise architecture o DevOps o Trust beyond team API Design o Iterative bottom up design driven by incremental needs Legacy Code o Approaches to add behavior to legacy code without refactoring (sprout method, sprout class, wrap method, wrap class) Design Patterns o Observer o Iterator o Composite Continuous Delivery o CI automates UI tests, and code metrics o Virtualization o Scripted environments o Scripted deployment/ promotion o Velocity multipliers o Always ready to deploy to production	Testing truly drives design and care is given to maintaining comprehensive and fast feedback. Automated Testing o Tests drive design o Balancing use of in-process integration, unit, and isolation tests o Intentional use of fakes, stubs, mocks, and spies o Acceptance and functional tests automated o Refactoring database design o Performance testing Design Principles o Open/Closed Principle o Liskov Substitution Principle o Interface Segregation Principle o Dependency Inversion o Principle of least knowledge Code Smells & Refactoring o Case statements o Divergent change o Shotgun surgery o Temporary field o Primitive obsession o Refused bequest o Lazy class o Middleman o Associated refactoring patterns API Design o API contract defined by tests used by both sides o API stubbed Legacy Code o Sensing effects class has on other classes o Separating dependencies in order to get test to work without loading entire system o Identifying seams Design Patterns o Strategy o Decorator o Factory o Singleton o Null Object o Command o Facade Continuous Delivery o Infrastructure as code w/ TDD o Push-button promotion o Feature tags o Backwards compatibility o External interfaces have doubles and defined by tests	Automated Testing o Test feedback iteratively guides writing of code o Outside in approach o Determining purpose and value of a test o Testing behavior over design o Security and other non functional testing Design Principles & Code Smells o How to navigate conflicting concepts API Design o Versioning o Backwards compatibility Legacy Code o Refactoring patterns: o Irritating parameter o Hidden dependency o Construction BLOB o Global dependency o Include dependencies o Onion parameter o Aliased parameter o Hidden method o Undetectable side effect Design Patterns o Template o State o Proxy o Remainder of GoF and other patterns. Continuous Delivery o Deployment occurs multiple times per day (optionally production) o Complete automation o Post deployment tests and rollback o A/B testing
Journeyman →	Demonstrating		Assimilating		
Craftsman →	Leading		Demonstrating		Assimilating
Artisan →			Leading		Demonstrating

Assimilating = understands concepts and working to apply on professional work
Demonstrating = using effectively as matter of routine on professional work
Leading = considered expert; sees through eyes of both expert and novice and able effectively teach
* everyone teaches, as they openly share what they know

pillar
Doing IT Differently.