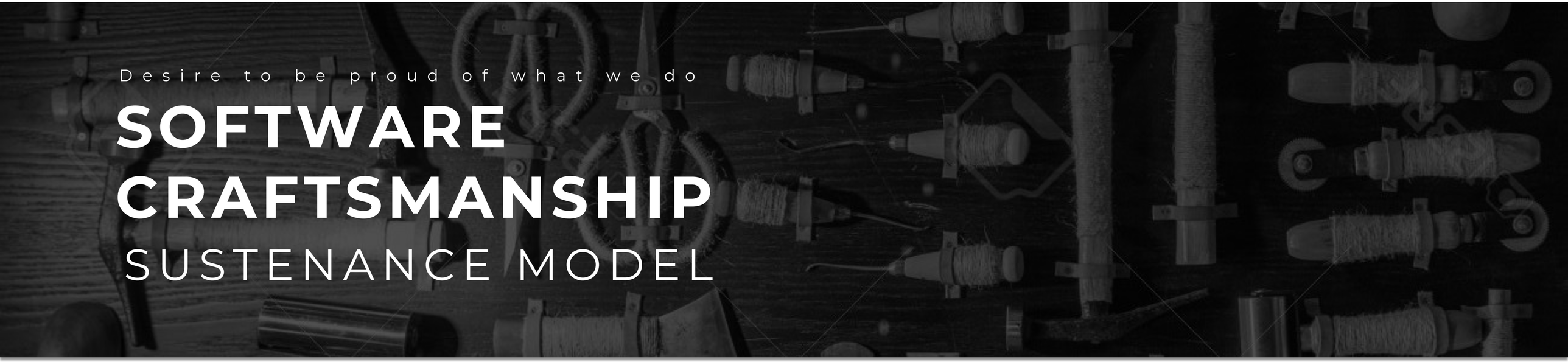
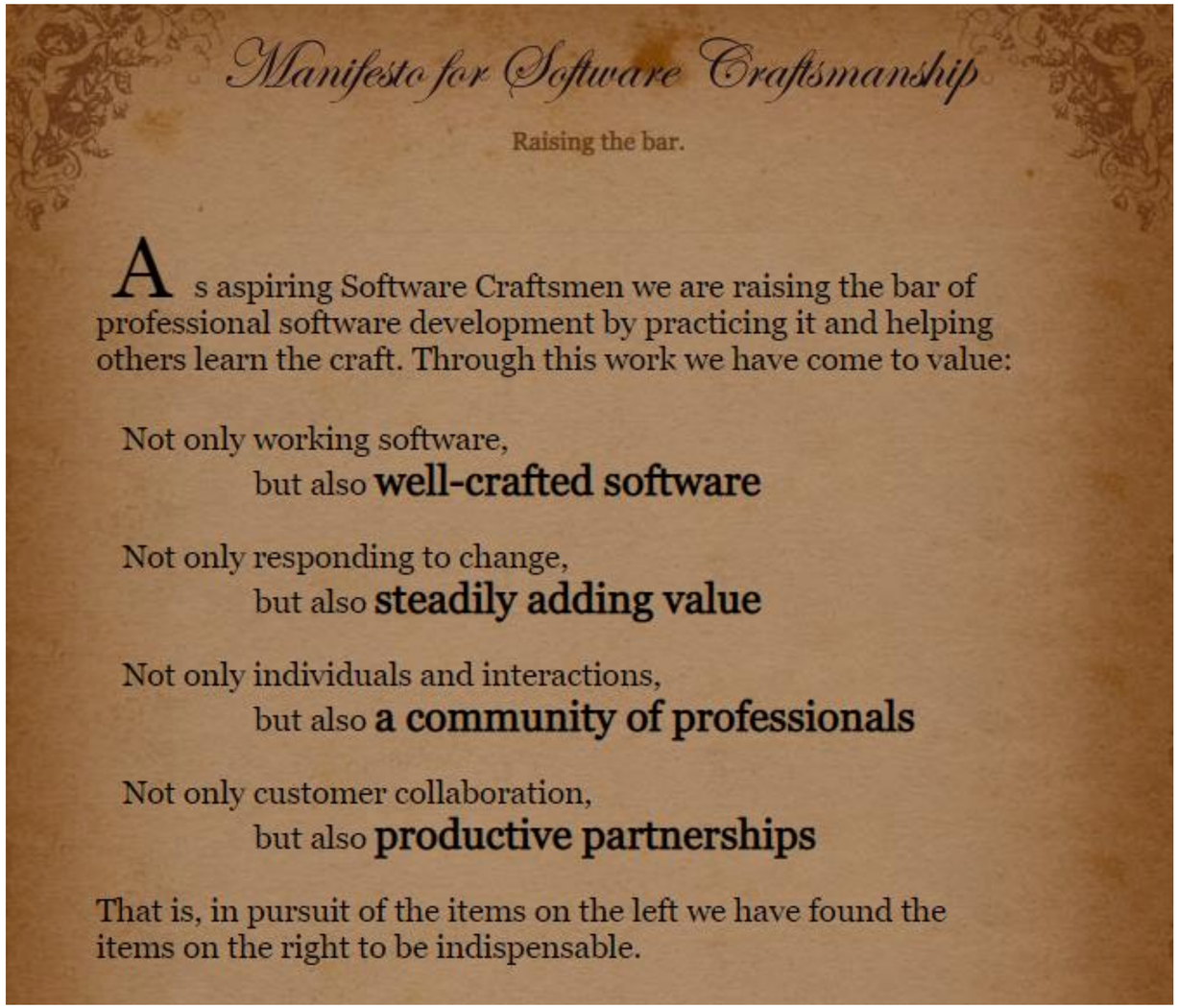


Desire to be proud of what we do

SOFTWARE CRAFTSMANSHIP

SUSTENANCE MODEL

BHARAT
MANE



https://manifesto.softwarecraftsmanship.org/

MANIFESTO FOR SOFTWARE CRAFTSMANSHIP

MOTIVATION

The origins of software craftsmanship came from the Agile software development movement which aimed to reform software project management in the 1990s.
In December of 2008, a summit was held in Chicago that resulted in the drafting of the Software Craftsmanship Manifesto:

THE PROBLEM DEFINITION

In spite of close to 2 decades , the adoption and sustenance is still not as much as it should be. A very small percentage of organization practice the software craftsmanship to it’s core.

Objective of this research is to propose the model that would help adopt and most importantly sustain SOFTWARE CRAFTSMANSHIP

The model that covers four areas and 40+ practices that can be measured over 5 key process indicators

NOTE: At present this model is evolving and being piloted at Society Générale. There are few others who are also contributing to the evolution of this model

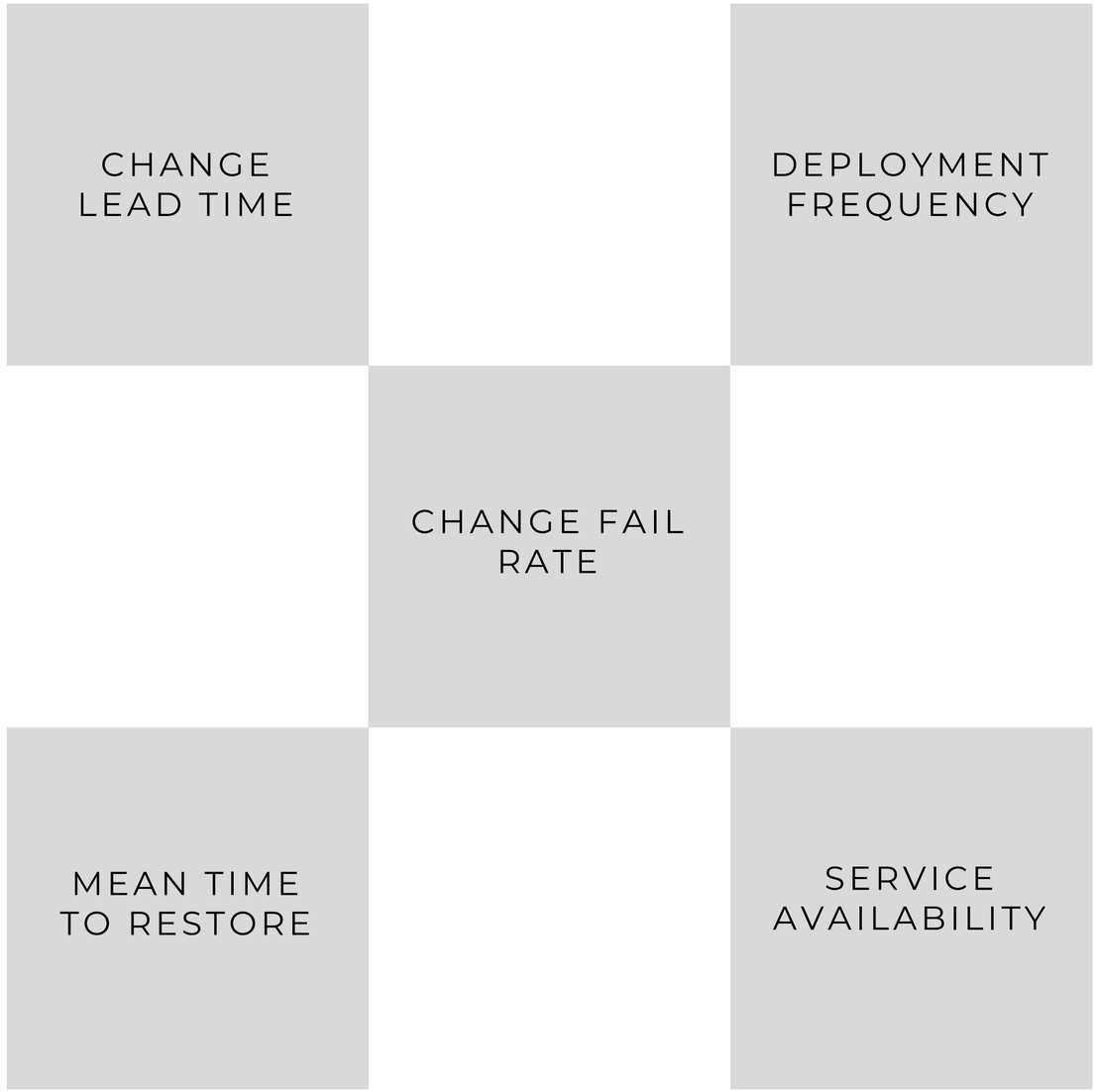
FOUR AREAS

PEOPLE

TOOL MASTERY

DEVELOPER EXPERIENCE

DEVELOPMENT PRACTICES



5 KEY PROCESS INDICATORS

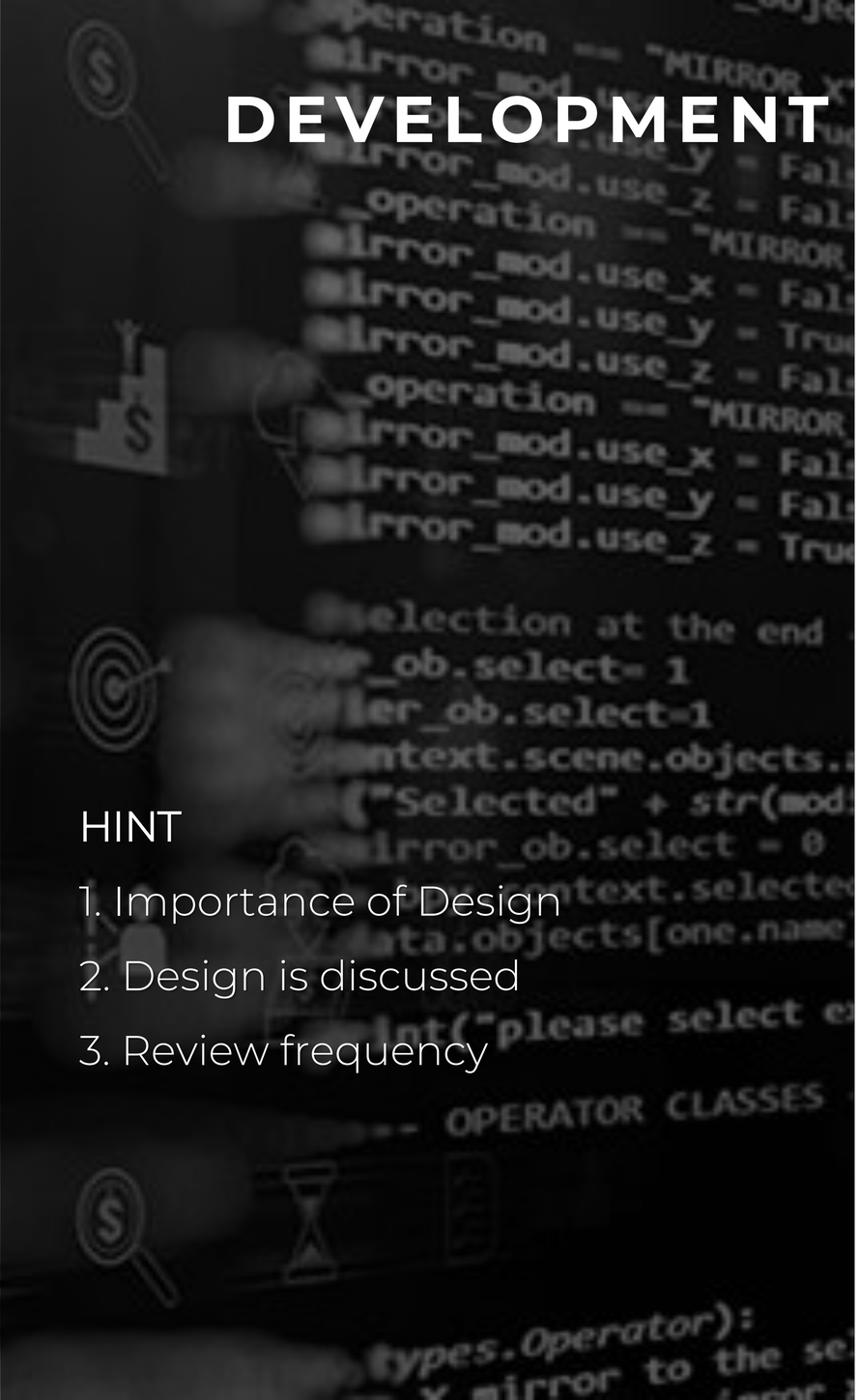
5 KEY PROCESS INDICATORS

- Reduced **LEAD TIME** by reducing wasteful activities, which, in turn, let us deploy more frequently.
- Increased **DEPLOYMENT FREQUENCY** forces the teams to improve their practices and automation.
- Better practices and automation leads to the improved **CHANCE FAIL RATE** and **TIME TO RESTORE**.
- The improved Change Fail Rate and Time to Restore lead to an improved **SERVICE AVAILABILITY**

FORTY+ PRACTICES

Development Practices	People	Tool Mastery	Developer Experience
Naming Conventions/Clean Code	Recruitment Evaluation	Development Environment (code editor, IDE)	Backing from PO & Senior Leadership on craft
API Naming Conventions	Growth Mindset	GIT or any other VCS	Trainings
Restfulness	Psychological safety	CI/CD Pipeline	access to information
Design Patterns	Re-usability	Tests are triggered automatically with metrics	onboarding process
Domain Definition (DDD)	Refuse complexity and over- engineering (YAGNI)	Code Quality SONAR	Administrative processes vs development time
Design Principles (SOLID, KISS, DRY, YAGNI, POLA)	Ownership Mindset	Code Security, Checkmarks	Softwares & Hardwares
Code Reviews	End-to-End value delivery	Deployment XLDeploy	
Pair Programming	Cross functional teams	Infra as a code	
Continuous Refactoring	The code is own by the community		
TDD			
Code Coverage			
BDD			
Business Involvement for BDD			
Mutation Testing			
Integration Testing			
Logging			
UX For developers			
Microservices Design			
Think of performance during design			
Vulnerability			

DEVELOPMENT PRACTICES



HINT

1. Importance of Design
2. Design is discussed
3. Review frequency

Design Principles

We apply best suited design and patterns, for example DDD with Hexagonal architecture only if relevant

1. We immediately start coding without taking design into consideration
2. We seldom apply design principles. When we do, it's an individual effort. It can result in design flaws that go unnoticed for a long time
3. We usually apply design principles. However still it's an individual effort and is not always collectively discussed
4. We often apply design principles. This time it's the result of a collective effort and is systematically discussed
5. Architecture is constantly reviewed and discussed collectively. Spotted design issues are addressed as soon as possible

Microsoft Excel
Worksheet

https://github.com/bharatmane/public-presentaions