Software Process Improvement cnd.

Peter Dolog
dolog [at] cs [dot] aau [dot] dk
E2-201
Information Systems
March 27, 2007

SPI as organizational change

Motivation

Software Engineering is a continued struggle to improve quality and fight schedule and cost overruns

Piles of literature have emerged over the last 15 years on how to improve software processes (SPI)

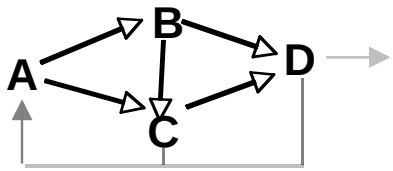
Many practical improvement efforts seem to fail

The challenges of SPI seem poorly understood

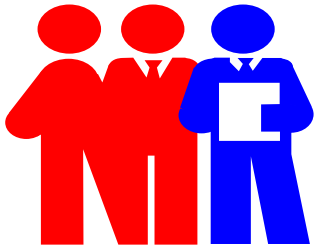
Agile approaches seem to offer alternative strategies for SPI but cannot be understood from within the existing SPI framework

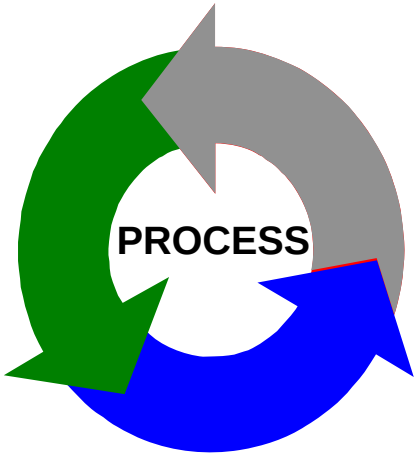
A Definition of Process

The means by which people, procedures, methods, equipment, and tools are integrated to produce a desired end result.



Procedures and methods defining the relationship of tasks





People with skills, training, and motivation

Tools and equipment

A Defined Process Can:

Help guide the work in an orderly way

Improve the understanding of what should be done

Provide organizations with a consistent working framework while permitting individual adjustments to particular needs

The Benefits of Process Standards

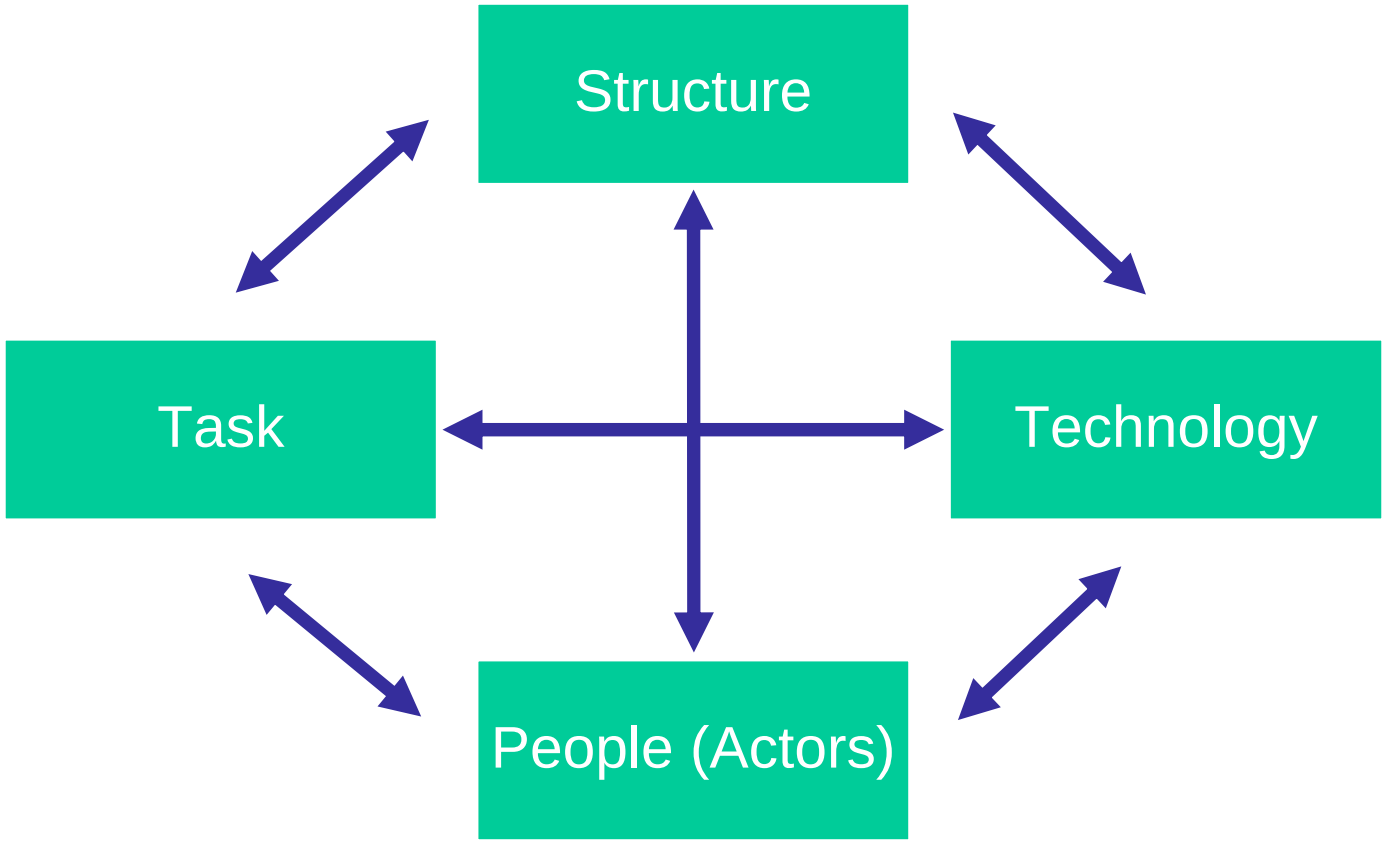
Help reduce the problems of training, review, and tool support.

With standard methods, each project's experiences can contribute to overall process improvement.

Process standards provide the basis for process and quality measurements.

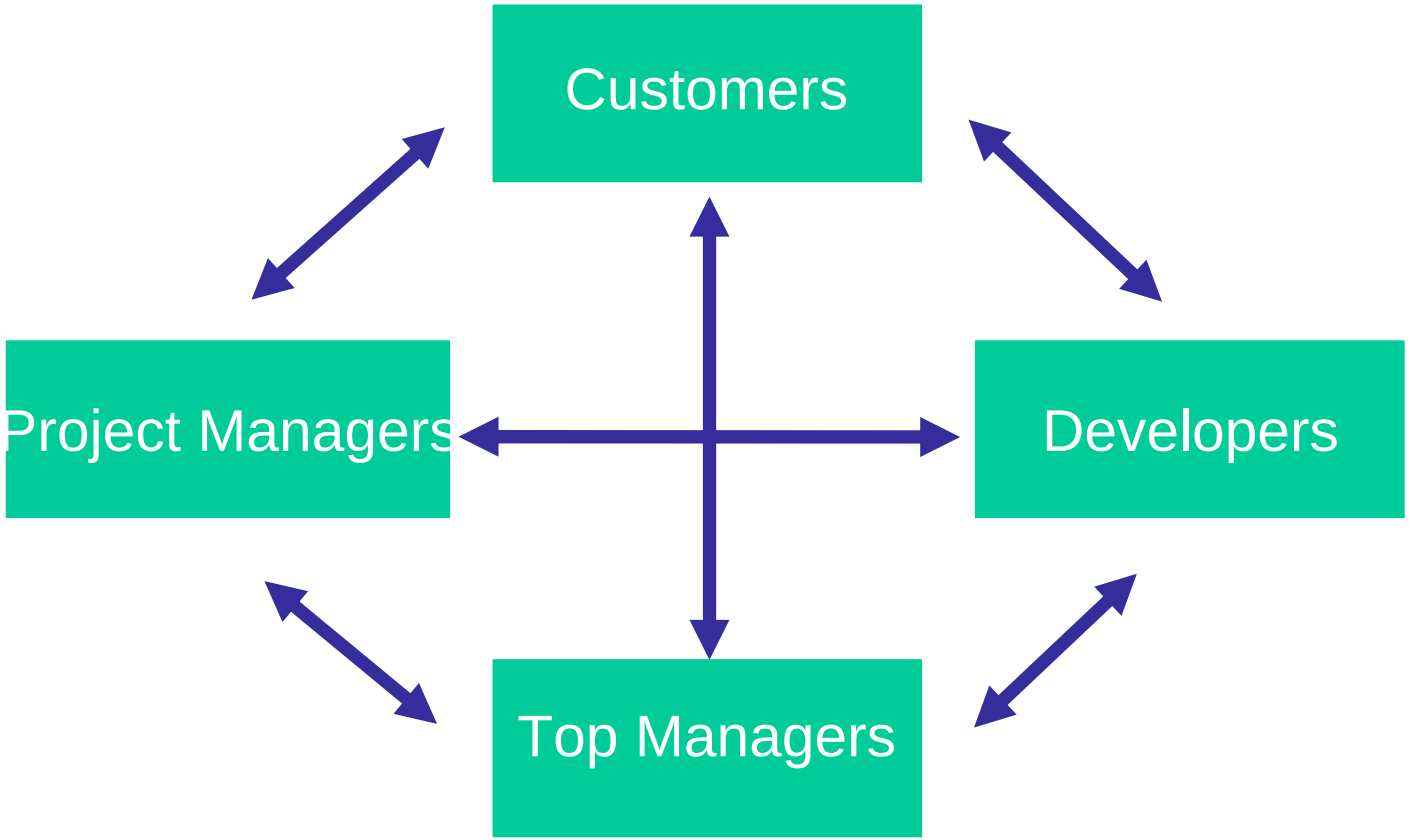
Since process definitions take time and effort to produce, it is impractical to produce new ones for each project.

Organizational System

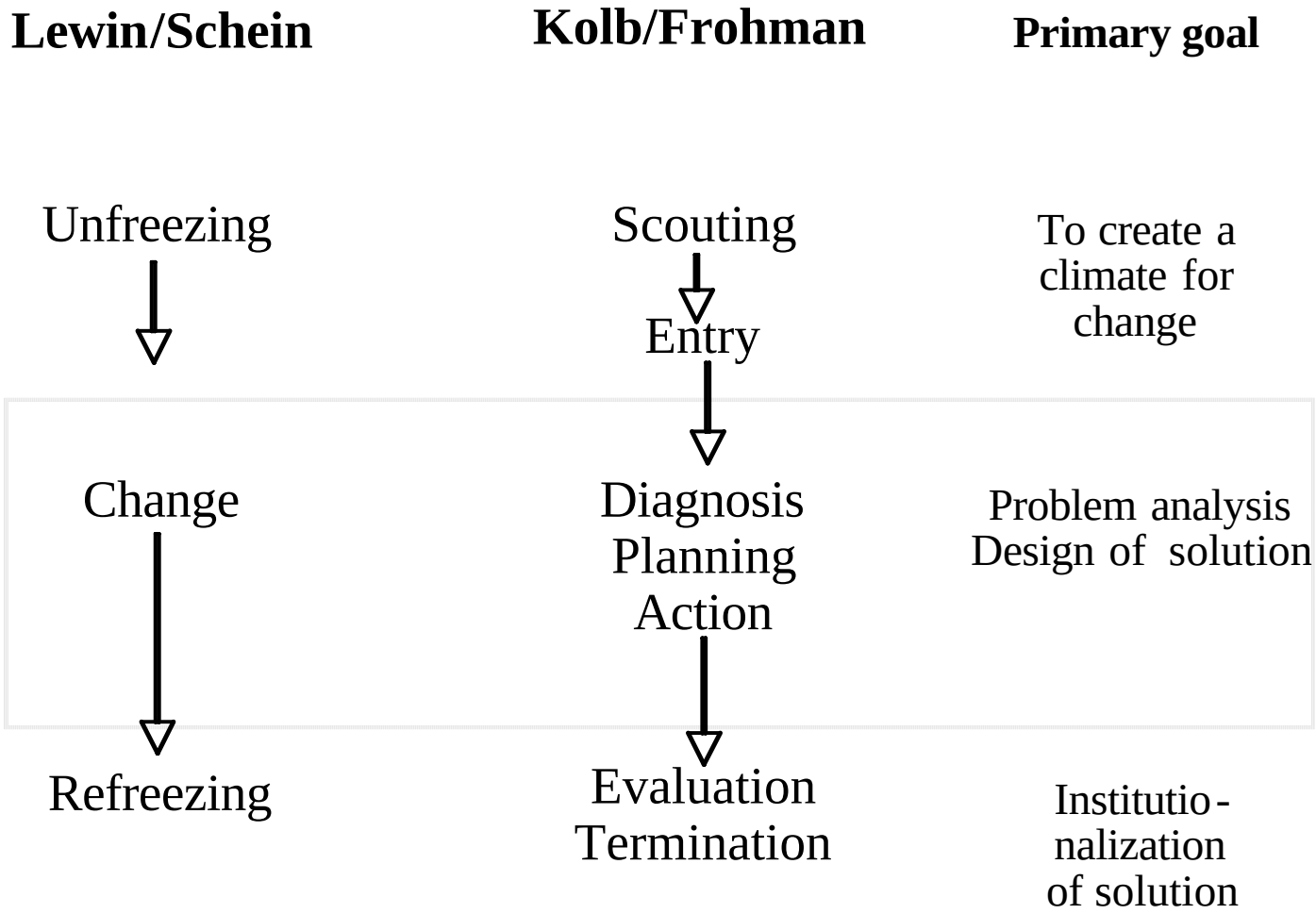


Leavitt, H. J. (1965) "Applied Organizational Change in Industry: Structural, Technological and Humanistic Approaches." In James G. March, Ed., Handbook of Organizations, 1144-1170. Chicago: Rand McNally.

SPI - Complex Interactions



Organizational Change Strategies



P. G. W. Keen: Information Systems and Organizational Change. Comm. Of ACM, 1981

SPI processes: IDEAL

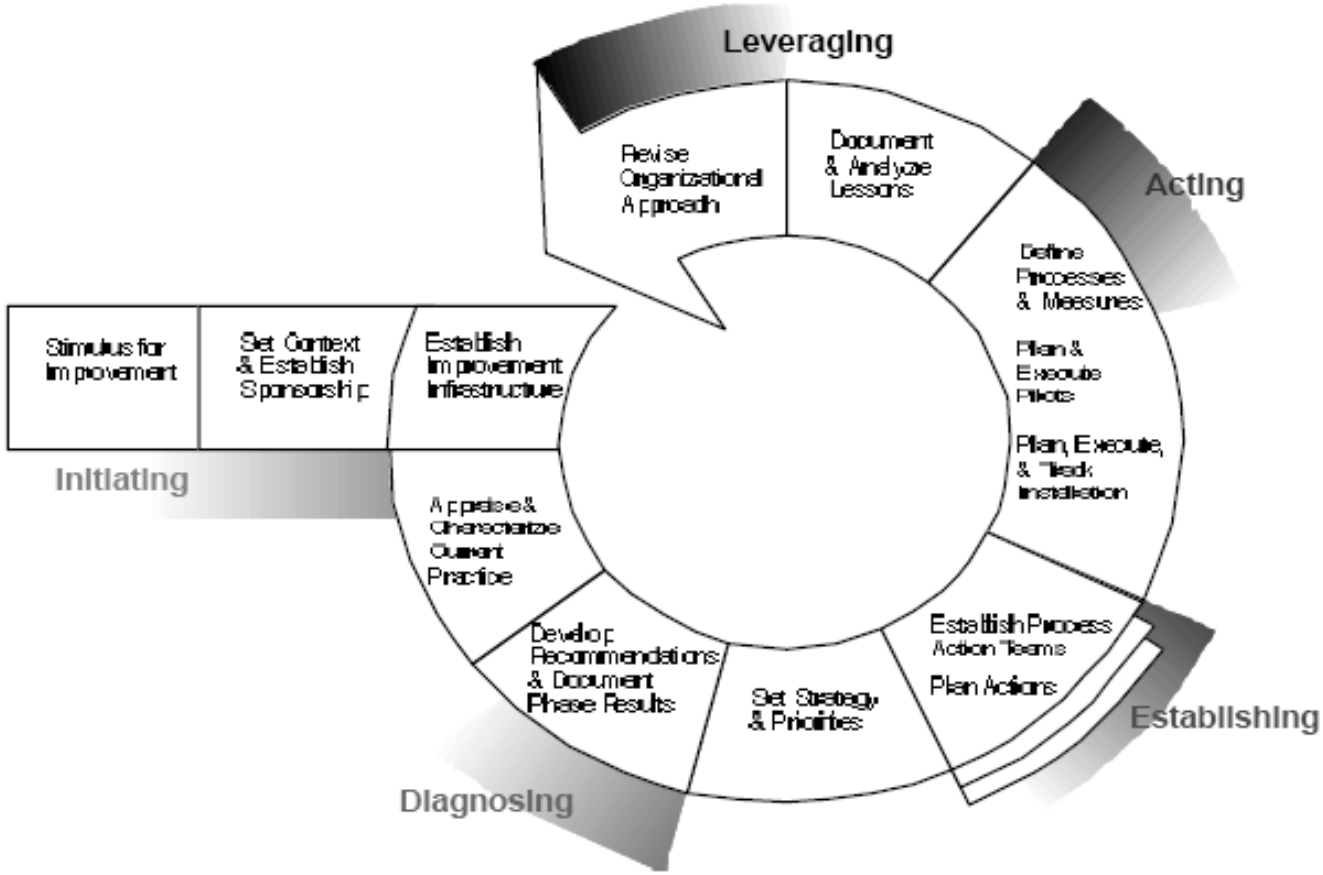


Figure Intro-1: The IDEAL Model

© Robert McFeeley: **IDEAL: A User's Guide for Software Process Improvement. SEI**

Software Process Measurement

Process-related measures
Project-related measures
Product-related and customer-related measures

Zahran, S. (1998). Software Process Improvement: Practical Guidelines for Business Success. Reading, Mass., Addison-Wesley.

Process-related measures

Number of times the program failed to rebuild overnight
Number of defects introduced per developer hour
Number of changes to requirements
Hours of programmer time available and spent per week
Number of patch releases required after first product ship
Overhead of each inspection
Cost of first-time testing
Cost to fix code defects
Cost to fix design defects

Project-related measures

Productivity

Staff hours

Dates

Overrun for schedules

Progress vs. Plan

Number of units completed

Number of units tested

Problem Counts

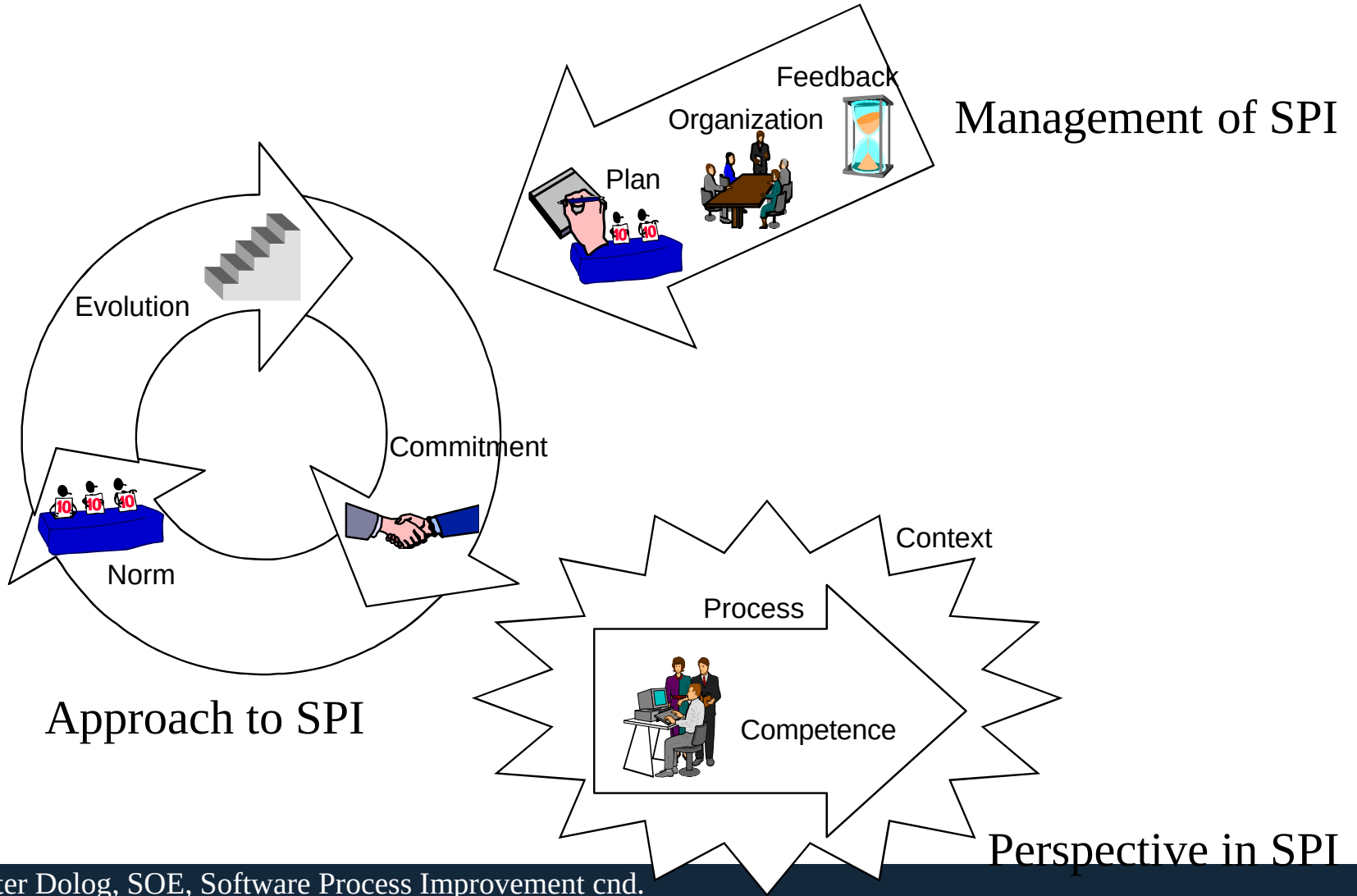
Product/customer-related measures

Cyclomatic Complexity
Lines of Code
Comments Percentage
Cyclomatic complexity Method
Weighted methods per class
Response for a class
Lack of cohesion of methods
Coupling between objects
Depth of inheritance tree
Number of children
Number of change requests

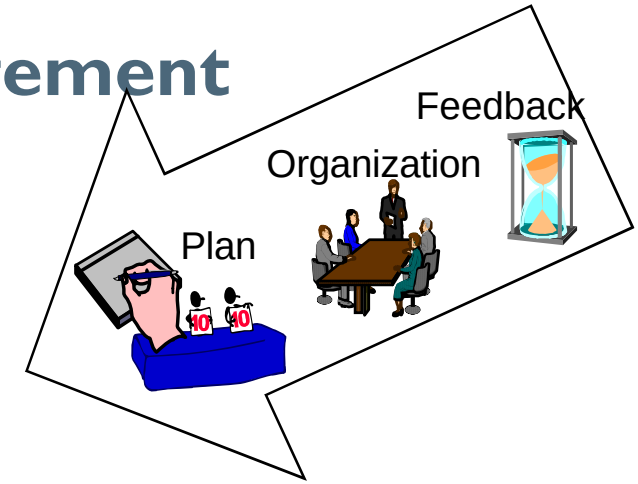
A MAP of SPI

1998-2001

MAP of Software Process Improvement

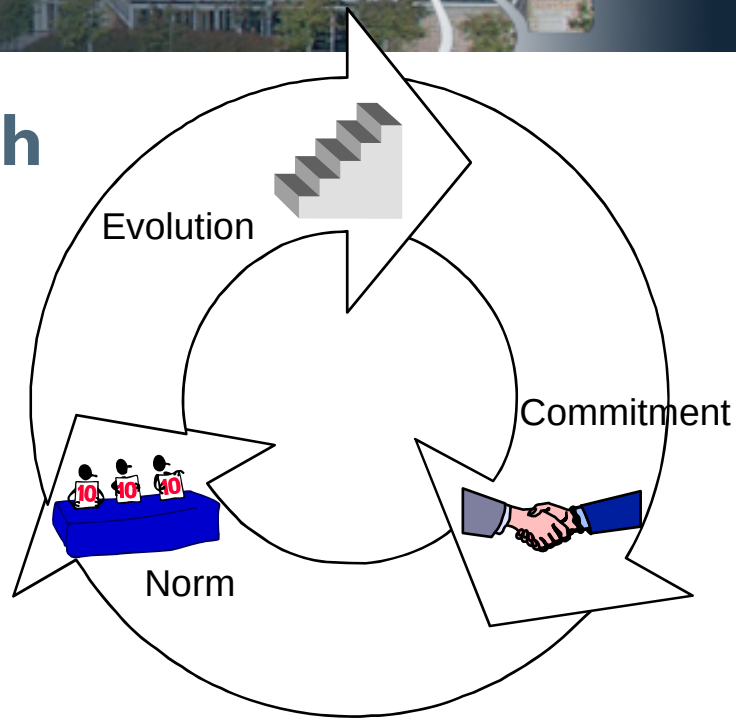


Key elements in SPI management



Concern	Idea	Aspiration	Pitfalls
Management of SPI	Organization	Dedicated and adapted effort	Inadequate resources, emphasis and coordination
	Plan	Plan goals, activities, re-sponsibilities and coordination	Inability to improve. Diversity or deadlock
	Feedback	Measure and assess benefits	Opportunism, and loss of relevance

Key ideas in SPI approach



Concern	Idea	Aspiration	Pitfalls
Approach to SPI	Evolution	Experimental learning and stepwise improvement	Wearing and inertia
	Norm	Seek guidance in ideal processes	Hastiness and fundamentalism
	Commitment	Ensure dedication and legitimacy	Politics and gold plating

The general MAP

Concern	Idea	Aspiration	Pitfalls
Management of SPI	Organization	Dedicated and adapted effort	Inadequate resources, emphasis and coordination
	Plan	Plan goals, activities, responsibilities and coordination	Inability to improve. Diversity or deadlock
	Feedback	Measure and assess benefits	Opportunism, and loss of relevance
Approach to SPI	Evolution	Experimental learning and stepwise improvement	Wearing and inertia
	Norm	Seek guidance in ideal processes	Hastiness and fundamentalism
	Commitment	Ensure dedication and legitimacy	Politics and gold plating
Perspective in SPI	Process	Integrate people, management and technology	Disinterested customers
	Competence	Empowerment through competence building	Turf guarding
	Context	Establish sustainable effort	Machine bureaucracy

Diagnosis (case example)

MAP	How	Case
Management	Organization	+
	Plan	+
	Feedback	—
Approach	Evolutionary	+
	Norm	+
	Commitment	—
Perspective	Process	+
	Competence	+
	Context	—

No One-Size-Fits-All to SPI

Normative SPI concerns	Correct effort	Ambitious effort	Grass-root effort	Adolescent effort
SPI Organization	+			
SPI Plan	+	+		
SPI Feedback			+	
Evolution	+		+	+
Norm	+	+	+	
Commitment	+	+	+	+
SE Process	+	+		
SE Competence		+	+	+
SE Context	+	+		

Blueprints versus recipes

2002-2003

The Blueprint approach

Externalization

- What is a software process? Where is it?

Separation

- Who designs the software process? How are they designed?

Structuration

- When are software processes designed? Why are they designed?

Improvement by Design threats

Separation of knowledge from use	Externalization of knowledge
Gold-plating process models at the expense of knowledge flow.	Confusing information with knowledge.
Lack of reflective dialogue between process users and designers.	Reifying process knowledge.
Process rollout by distributing process information.	Paying little heed to the role and importance of tacit knowledge.
Focusing on the past and the present and not the future.	Process ossification - downplaying thinking and reasoning.
Standard processes leaving little room for experimentation.	Read it on the intranet - substituting technological contact for human interface.

Eleven Deadly Sins

Not Developing a Working Definition of Knowledge

Emphasizing Knowledge Stock to the Detriment of Knowledge Flow

Viewing Knowledge as Existing Predominantly Outside the Heads of Individuals

Not Understanding that a Fundamental Intermediate Purpose of Managing Knowledge Is to Create Shared Context

Paying Little Heed to the Role and Importance of Tacit Knowledge

Disentangling Knowledge from Its Uses

Downplaying Thinking and Reasoning

Focusing on the Past and the Present and Not the Future

Failing to Recognize the Importance of Experimentation

Substituting Technological Contact for Human Interface

Seeking to Develop Direct Measures of Knowledge

Fahey, L., and Prusak, L. (1998) The eleven deadliest sins of knowledge management. California Management Review, 40, 3, 265-276.

Design: Verb or noun?

Software process design as architecture	Software process design as improvisation
A software process is a blueprint.	A software process is a recipe.
A software process is constructed at a single point in time.	A software process is continually reconstructed.
Software processes produce order through intention.	Software processes produce order through attention.
A software process creates planned change.	A software process codifies unplanned change after the fact.

Inspired by: Weick, K. E. (1993) "Organizational redesign as improvisation." In G.P. Huber and W.H. Glick, Eds., Organizational Change and Redesign, 346-379. New York: Oxford University Press.

Recipes - the concept

As an alternative to Blueprints, we can conceive of software development being supported by recipes—guidelines that we can tailor to specific and shifting conditions.

Using a recipe, process users collectively design the software processes through facilitation, reflection, and improvisation.

Externalization -> Facilitation

Process information

- Methods, procedures, experiences, patterns

Tools

- Enhance capabilities of individuals and teams

Organization & competence

- Facilitate sharing of knowledge (pair programming, collective code ownership, mentoring, ...)

Separation -> Reflection

Engagement

- See activity as a context for learning - use reviews, tracking and project post-mortems

Imagination

- Create shared visions - use project kickoff meetings

Alignment

- Coordinate energies and activities - use standards, policies, and values

Structuration -> Improvisation

Focus on interactions
See everyday projects as experiments
Ensure the ability and latitude to improvise under common standards
Empower teams to manage the unexpected
Encourage creativity through interaction