The Principle of Software Process Change

Alexandre Bergel
abergel@dcc.uchile.cl

Part of this lecture is based on “Managing the Software Process”, from Watts S. Humphrey (SEI Series in Software Engineering), 1990

Context of this lecture

There is a very strong wish from the Chilean IT society to become an off-shoring and outsourcing place

Getting a CMMI evaluation or ISO 9126 certification is crucial for opening the market

Proximity to the North American market make us think in terms of *software process*

Being agile and responsive to process changes is crucial to be competitive

Goal of this lecture

Getting a general sense of software process change

Especially focusing on:

resistance to change, champions, sponsors, unfreezing, refreezing

This lecture describes the principles for changing the software process and discusses some common misconceptions about software work. The objective of this lecture is to have a general sense of software process change, focusing on the issues as resistance to change, champions, sponsors, agents, unfreezing, and refreezing before launching a software process change program.

Software process in perspective

When some approach seems to fit a need, we often think it will solve all the problems

Software process management is a powerful mechanism

- to assess software problems
- to frame organizational improvement

However, it is not a cure-all

Process in perspective

Two other areas that need to be considered are people and design methods

People

- Software quality is extremely sensitive to the talent of its builders
- Large software teams must contain a mix of talent
- Not all errors are made by the least skilled professionals

People and skills

Better people clearly do better work

When focusing only on talent, you may bump into:

- short supply of the best people
- you probably have the best team you can get right now
- with a proper guidance, most people can do better work

Important to make better use of the talent we have

Design

- Superior products have superior designs
- Successes are always designed by people who understand the applications
- A software program can be viewed as executable knowledge
- Knowledge + creative design => a quality product

Some studies [1] showed that successes are always designed by people who understand the application best. For example, a well-designed program to control a missile was designed by someone who understood missiles.

[1] Curtis, W., H. Krasner, V. Shen, and N. Iscoe. “On building software process models under the lamppost.” IEEE Proceedings, ICSE’87

The Six Basic Principles

The basic principles of software process change are

1 - Major changes to the software process must start at the top

2 - Ultimately, everyone must be involved

3 - Effective change requires a goal and knowledge of the current process

4 - Change is continuous

5 - Software process changes need periodic reinforcement

6 - Software process improvement requires investment

1 - Changes must start at the top

- Major changes require leadership
- Conviction that long-term improvements are possible and essential
- Managers must set
  - challenging goals
  - monitor process
  - insist on performance
- Process problems are management's responsibility

2 - Everyone must be involved

A mature process help getting individual actions structured, efficient, and reinforced

As in professional sports, all the team members need to support each other

When they don't, they act like a mob of individual players resulting in a loose of synergistic cohesion

With an immature software process, software professionals are forced to improvise solutions

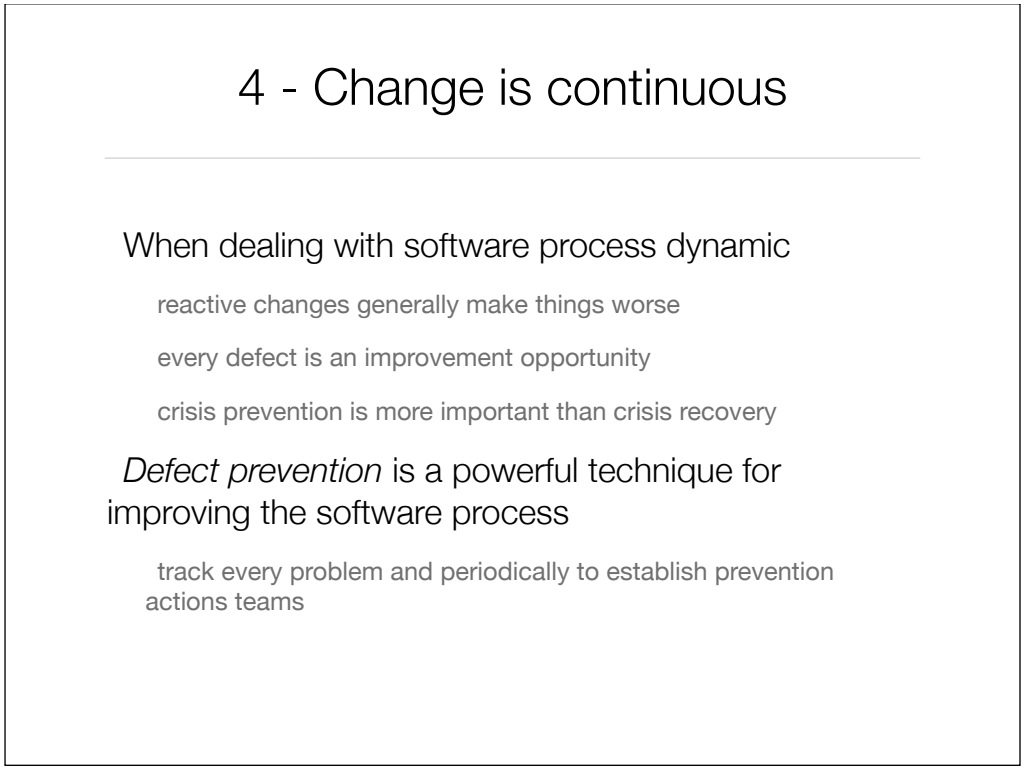
3 - Change is built on knowledge

An effective change program requires a reasonable understanding of the current status

People often feel that their problems are unique, however studies show this is not true

Experiences shows that problems were already addressed in the past, in the same organization

[1] Humphrey, W. S., T. Kasse, and D. Kitson. “State of Software Engineering Practice”, Software Engineering Institute Technical Report, CMI/SEI-89-TR-1



One of the most difficult things for a management team to recognize is that human intensive processes are never static. Both the problems and the people are in constant flux, and this fluidity calls for periodic adjustment of tasks and relationships. Even with a stable population, the people continually learn new skills and find different ways to solve problems.

The *Reactive changes generally make things worse* point comes from industrial engineering. In a crisis, the focus is on quickly fixing the immediate problem, on not on improving the process. It is wise to make permanent process changes in a disciplined way. By occasionally adding new review steps or requiring additional tests, the process can soon become an incoherent patchwork rather than an orderly improvement framework.

Defect prevention is a powerful technique for improving the software process. The general idea is to track every problem and periodically to establish prevention action teams.

One of the greatest conundrums of software management is maintaining sufficient priority on problem prevention. People gain greater visibility from putting out fires than from preventing them. These ‘‘heroes’’ are then more likely to be promoted than the quiet and effective professionals who stayed out of trouble. At lower maturity levels, management must consciously reward preventing as well as fixing.

5 - Changes won't stick by themselves

- Improvements are likely to deteriorate over the time
- Entropy refers to the steady increase in the randomness or disorder
- Precise and accurate work is so hard that is rarely sustained for long without reinforcement

Software engineering is a tough discipline, and it is getting tougher. There are endless opportunities for error.
The actions of even the best-intentioned professionals must be tracked, reviewed, and checked, or process deviations will occur.

6 -Investment

It takes times, skill, and money to improve the software process

- To improve the software process, someone must work on it!
- Improvements should be made in small, tested steps
- Train, train, train!

Common misconceptions

Effective changes depends on realism. Without an appreciation of the current state of the software process and a realistic view of the future, it is easy to succumb to wishful thinking

We must start with first requirements

There is a widespread but fallacious view that requirements are the customers' job and that development should not start until they are explicitly defined

Requirement must change as the software job progresses

Demand for firm and unchanging requirements is most wishful thinking
Requirements must change as the software job progress
Software development is a learning process
Incremental development process gradually increases the level of detail

If it passes test, it must be OK

If the generally dismal record of software quality problems doesn't prove this false, nothing will

The problem are technical

In spite of many improved languages, tools, and environments, the problem of software cost, schedule, and quality remain

We need better people

Since the software professionals make the errors, some people erroneously feel that they should be blamed for them

Implementing a Software Process Change

Effective change process

An effective change process has three phases:

unfreezing, moving, refreezing

Unfreezing is best initiated by an effort to understand organizational problems

Champions, sponsors, and agents

Champions are the one who initiate the change

Someone in authority needs to sponsor them

Next step is to identity the change agents

- lead change planning and implementation

Summary

This lecture describes the principle for changing the software process ...

... and discusses some common misconceptions that can inhibit the improvement efforts

The basic principles of software process change are

1 - Major changes to the software process must start at the top

2 - Ultimately, everyone must be involved

3 - Effective change requires a goal and knowledge of the current process

4 - Change is continuous

5 - Software process changes need periodic reinforcement

6 - Software process improvement requires investment

Common misconceptions

Need to be connected to the real World

- We must start with first requirements
- If it passes test, it must be OK
- The problem are technical
- We need better people