11 Laws of The System Thinking in Software Development

The System Thinking Laws from Peter Senge's book "The Fifth Discipline" applied to Software Development.

"I will work harder" - the horse Boxer (from George Orwell's Animal Farm)

bichuas

The System Thinking Laws from Peter Senge's book "The Fifth Discipline" applied to Software Development.

1. Today's problems come from yesterday's solutions.

We, humans, are happy when we solve problems. We often don't think much about consequences. Surprisingly, our solutions could strike back and create new problems.

- A company decides to reward few key members of the very successful team with bonuses and promotions. The rest of the team feel unfairness and loss of motivation. Eventually tension between members is increased. The following projects are no longer successful.
- A project manager frequently asks developers to fix a new bug or work on urgent requests from customers. Developers do their best to fulfil these requests. Frequent distractions prevent them from finishing their main tasks for the iterations. Project shows only little progress.

2. The harder you push, the harder the system pushes back.

We have this stubborn reaction to push our way through when things are not working out as we want. We charge without time to stop, think and find better alternatives. Sometimes we solve problems, but often we find ourselves up to ears in the swamp of other problems.

- Managers keep pushing people to work overtime and meet deadline when a system is far from completion. The number of bugs is increasing and overall quality is rapidly dropping causing more delays. More and more effort is required to launch the software system.
- Developers heroically stretch the same architecture for the new system requirements, which don't fit into the old rigid way. They are so busy doing it that don't have time to stop, analyze and change approach. The system degrades.

3. Behavior grows better before it grows worse.

Short-term solutions give us a short break and temporary improvement, but don't eliminate fundamental problems. These problems will make situation worse in the long run.

- A company gives customers hefty discounts and run expensive advertisement - many people buy the software. Customers are unhappy after purchase, because software is unusable and unreliable.
- Management promises development team big bonuses if they finish system in time. A team work hard, but soon realize that it is impossible. Developers becomes cynical and unmotivated.

4. The easy way out usually leads back in.

We learn few solutions in our life, which brought easy success earlier. We try to vigorously apply them in any situation disregarding particular context and people.

- Agile coach is forcing full Extreme Programming implementation when developers are not ready to accept some practices as pair programming or TDD. It creates stress, conflicts and allergy to any Agile approach.
- Developers apply design patterns everywhere unnecessarily complicating the system.

5. The cure can be worse than the disease.

Some familiar solutions could be even dangerous like drinking beer while programming to reduce stress for unreal deadlines.

- A company hires various contractors to work on core features, because doesn't trust full-time developers. As a result, the system doesn't have conceptual integrity, in-house developers don't understand and cannot change it. Domain knowledge, interpretation and concepts are missing from the brains of company employees.
- Developers make many shortcuts, copy and paste code for similar functionality to please management and deliver first version fast. They do quick progress at the beginning, but code eventually becomes Big Ball of Mud.

6. Faster is slower.

When we feel smell of success we start advance at the full speed without much caution. However, the optimal rate of growth usually is much slower than the fastest growth possible.

- Managers add many people to already successful project. The overall progress becomes slower, because of communication overhead and loss of team coherence.
- Developers quickly add new features to the system without proper refactoring and improving existing code. System becomes difficult to understand and modify.

7. Cause and effect are not closely related in time and space.

We are good at finding causes to our problems, even if they are just symptoms and far from real root causes.

- Development team stops accepting requirement changes from customers to finish a system in time. Customers are unhappy with delivered software.
- After few incidents with a live system, management compel developers to get approval and write detailed technical specification before implementing any change in the system. Developers lose motivation for any improvements in the system and start procrastinating.

8. Small changes can produce big results-but the areas of highest leverage are often the least obvious.

Most obvious grand solutions like changing company policy, vision or tag line often don't work. Small ordinary, but consistent changes could make a huge difference.

- Developers have everyday interactions with customer and make most decisions. As a result, customer needs are well understood, decisions are better and solutions are optimal.
- Developers build automated unit tests for each function in the system. As a result, design is flexible, people are confident, the system is fully tested after each change.

9. You can have your cake and eat it too - but not at once.

We often face rigid "either-or" choices. Sometimes they are not dilemmas if we change our perspective and rules of the system.

- Experience managers know that we cannot increase the number of features and reduce time and cost simultaneously. However, it could be possible to achieve if we just improve flow of ideas, find right people and avoid over-engineering.
- Developers believe that they should follow either Transaction Script or Domain Model architecture patterns. However high performance solution in the complex domain can combine both for the best effect.

10. Dividing an elephant in half does not produce two small elephants.

Inability to see the system as a whole could often lead to suboptimal decisions.

- A manger evaluates developers based on the number of lines of codes they produce or points they implemented during iteration. Developers produce a lot of useless code.
- Management promises testers $5 dollars for any found bug in the system. Testers are no longer interested in cooperating with developers and prevent root causes of the bugs. Good and productive relations between teams disappear.

11. There is no blame.

We like to blame and point fingers to other people or circumstances, sometimes we even believe in this. But we and cause of our problems are part of the System.

- It is Joe fault that the team didn't release system this morning. He didn't fix all the bugs overnight even though PM kindly gave him free pack of beer, T-shirt and pizza.
- People don't use a company's excellent Web 2.0 social application. Users are simply stupid and don't appreciate hard work.

So what?

These 11 laws of The System Thinking show that all our solutions have consequences, sometimes bad and unexpected. Systems around us are what they are, and we shouldn't blame, but learn them. To master The System Thinking and control these systems we should

1. understand what are the systems we are dealing with, either human or software.
2. consciously learn relations, cause and effect chains
3. perceive the systems as a whole and as the part of other systems

There are many challenges to the system thinking. Many we can defeat with gaining and using knowledge how

systems work. But the most serious challenge is our own contradictory human nature. Our passions, emotions and instincts could easily defy the rational and systematic way of thinking. First step to master the system thinking is to learn how to cooperate with ourselves.

Question: What is your experience with the system thinking (or absence) in software development?
