J. Favaro and P. Falcone, “The Agile Theater,” in: *Computer Programming*, CP130, Infomedia S.r.l., Ponsacco, December 2003, pp. 69-70.

The Agile Theater

John Favaro and Patrizia Falcone
Consulenza Informatica, Via Gamerra 21,
56123 Pisa, Italy
jfavaro@tin.it, Patrizia.Falcone@tin.it

They looked at each other in surprise. Nobody had expected this new scenario. It was late in the evening, and they had just finished dealing with the last story. Why now? Was it just a test to see whether they could handle it? No matter. Within minutes all the experience and technical preparation of the team began to surface. Two of the team members recognized the underlying structural patterns in the new story and began to elaborate the outlines of a solution. The other team members picked up quickly – they had worked together as a team and in pairs in every possible combination – and soon the solution emerged, with fully shared ownership.

A description of an Extreme Programming (XP) team working on a payroll processing package for a particularly demanding client? Or perhaps a Crystal team on a tight schedule deploying a new web-based life insurance policy management system? Neither: an improvisational theater group.

In recent years, software engineers have often been able to enrich their understanding of their own discipline by examining *other* disciplines, ranging from the more obvious, such as integrated manufacturing and architecture, to the rather less obvious, such as gardening and music (the members of the prestigious Atlantic Systems Guild once took choral singing lessons in a search for new insights into the nature of teamwork). In this article, with the help of a professional theatrical director and communications consultant, we will examine the relationship between software engineering – in particular the branch known as “agile methods – and the theater – in particular the branch known as “improvisational theater.”

At first glance, the theater and software development might seem as far away from each other as the sun and the moon. Consider, however, the following perspective: at the core of software development is the exploration, analysis, and formalization of behavior. And so it is with the theater – with the minor difference that the theater has been at it for centuries.

More surprising is perhaps the claim that the techniques of *improvisational* theater might be applicable to what, after all, has been associated with the rigid precision of an engineering discipline. But we are coming to recognize that programming is much more a social activity than previously appreciated (although some pioneering books having been telling us this for years) [1] [2]. We are also realizing that instead of being a linear, methodical activity, modern software development usually involves continuous reaction to continuously changing conditions.

Before we begin, let us first dispel a common misconception about the nature of improvisational theater. Most of us are familiar with a type of “online” improvisation we see on popular entertainment television, where an actor (Roberto Benigni is a good example) or group of actors asks the audience for an idea or theme and proceeds to improvise a scenario (usually humorous) around it. A loose analogy to a team of agile programmers dealing with changing requirements is evident, as we saw in the introduction. But there is another, even more important use of improvisation: as a tool for "offline" study. Here, the actor uses improvisation as a *technical tool*, to work on the subtle qualities of a complex situation or character without being led astray by a theatrical text, with its fixed, static structure. Working his way inside the character, the actor brings back outside what he has found – like the statue that Michelangelo famously discovered inside his block of marble. The actor's approach is to break up reality into its components, study each one individually, and then to put them all back together seamlessly. This is where we can begin to draw some lessons for the software developer.

Let's develop a concrete example, with an actor exploring a particular character. In the theater – as in software development – working against *constraints* can be an especially valuable source of insight. We now introduce a constraint: the character has a stutter (effectively both a physical and a psychological constraint). Exploring the pain and discomfort of the stuttering character is a way to arrive at his essence – the "what" and the "why" of the character.

Note that here the actor is working outside the context of any specific theatrical situation. Working out of context is an important technique for uncovering and understanding aspects that normally would not even be considered. It makes the actor aware of the choices he can make. It puts the actor in control of the situation, and not the other way around. And now the surprise: at this point, the actor *throws away everything he did*. That's right: he discards all of the improvisation. If the actor's exploration of the character had remained on the surface, he would have arrived only at an imitation of the sound of the words uttered by the stutterer. But his understanding of the pain and discomfort of the stutterer remain inside him, and he can use this understanding in a completely different context, say, the *Cherry Garden* of Chekhov – where his character may not be a stutterer at all, but, for example, might exhibit the same psychological discomfort as the stutterer.

This is the purpose of *spikes* in Extreme Programming: intense, focused investigations into the nature of particularly complex or subtle technical problems, outside the normal context of the project. By extracting himself from the normal context, the programmer is free to deepen his understanding of the true nature of the issue. Afterward he may throw away all of the exploratory work he has done, but the insights gained remain within him, and are brought back into the project. This sets agile methods apart from traditional methodologies, which generally frown on such departures from planning.

Now let us move from one to two actors. Although the most immediate analogy would be to the XP practice of *pair programming* – and certainly some value can be had from that analogy – the more important analogy is the joint exploration of a new situation in a project. For example, suppose that a waiter now enters the scene. The task of the actors is to discover the relationship between these two characters, the conflicts that may arise, and which solutions may be developed.

At this point, the composition of the conflict becomes important. For example, the stuttering customer reacts differently according to whether the waiter treats him with kindness or with impatience. If the waiter's impatient attitude leads only to more stuttering, then this could result in a negative solution (the customer decides to leave).

In this case, the waiter will have to discover what needs to be done to put the customer at ease in order to arrive at a positive solution.

It is possible that they discover that the situation that existed up to that point can no longer function, because the new character cannot be incorporated into the existing scenario. In improvisational theater, *this nearly always happens when a dialogue has not been successfully created.* That is, each character ends up only carrying on a monologue. There is only one way for a dialogue to emerge: at any given moment, there is one who leads, and one who follows. *And this must happen in alternation.* The more the actors are in synch, the more each understands intuitively when to lead and when to pass the baton to the other. This goes very much against the common notion that a great actor must always be at the center of attention. On the contrary, the greater the actor, the less he is afraid of seconding another – he knows that this, too, can be a valuable source of creative energy. The communication and feedback between the actors is constantly fed by the alternating leading roles.

What can the actors do to break an impasse if it arises? A violent confrontation between the characters – never desirable but always possible – might open up the way to a dialogue. Another solution is to step back and reconsider the entire scenario up to that point, and introduce new variables that permit a solution – for example, the waiter turns out to be an old school friend. And finally, if no solution is possible, the actors must summon up the courage to throw away all presuppositions and start over, with new character traits and even a clean slate of new characters if necessary.

Agile software development methods are among the few to eschew the notion of hierarchical team composition (for example, the common role of "chief architect" does not exist). The core values of **communication** and **feedback** are espoused in order to create a continuous dialogue among team members (both inside and outside of pair programming situations), with no fixed team leader. The continuous dialogue is an ideal environment for robust adaptation to a situation in flux. And the core value of **courage** is promoted in order to enable the team to take the tough decisions of throwing away all previous preconceptions and starting over – another thing that can happen with continuously changing requirements. Here, too, agile methods are unusual in their explicit and positive provision for abandonment.

Let us consider now a new team of actors, and a new scenario: the men are being sent off to war. But suppose that the entire team is male (something that occurs with perhaps too much frequency in software development). What does this mean? The female characters, which provide an important component of tension in such a scenario, are missing. A cardinal rule of improvisational theater is **simplicity**, the search for the most elementary solution possible, using the means available. There are no women, but the men can *receive telephone calls* from women – and thereby create the desired male-female dialogue. This technique for creating missing roles is a "theatrical pattern," well known to any technically prepared actor. Like their counterparts in object-oriented design, however, these theatrical patterns are only applicable in the appropriate *context*. For example, if the context is a sixteenth-century play, then telephone calls are hardly possible.

In this case, the team must dig even deeper in order to arrive at a solution. They discover that the essential **metaphor** of the scenario isn't "men versus women," but rather "who leaves versus who stays home." The all-important dialogue arises from the relationship between these two roles. Each actor must first use all the means at his disposition to understand his own role (an older one may realize that he stays home) and then contribute to the dialogue from which emerges the meaning of war (for example, a father and son must resolve an old conflict before the son's departure).

The discovery of the essential metaphor of a system (e.g. a desktop) is a central concept in agile development methods. It is needed to support the concept of emergent system behavior – that is, the sense of what the system is arises from the continuous dialogue of the team members.

Communication, feedback, simplicity, courage, metaphor – the theater has utilized these concepts for centuries to create some of mankind's greatest masterpieces. The ambitions of software engineers are perhaps a bit more modest, but their need to understand and implement these concepts is equally great. Let the one discipline learn from the other.

[1] Gerald M. Weinberg, *The Psychology of Computer Programming*, Dorset House, 1971

[2] DeMarco, Tom and Timothy Lister, *Peopleware*, 2nd edition, Dorset House, 1999

John Favaro is an independent consultant based in Pisa, Italy. He has published widely on principles of strategy and valuation for IT investment, and was one of the organizers of the first workshop on Economics-Directed Software Engineering Research (EDSER). He has published several articles on agile methods, particularly in relationship to economic considerations. He was the first to suggest that agile processes were options-driven and to analyze Extreme Programming concepts using option pricing theory.. He has an M.S. in Electrical Engineering and Computer Science from the University of California at Berkeley and a B.S. in Computer Science and Mathematics from Yale University.

Patrizia Falcone began her artistic career in 1979, specializing in the Strasberg and Stanislavsky methods and the techniques of the *Commedia dell'Arte*. In 1987 she added teaching and directing activities to her acting work, culminating in the foundation of a theatrical laboratory in Tuscany in 1994. Successively her innovative productions found recognition around Europe, such as her staging of the works of Ionesco, cited on the permanent ionesco.org site in Paris, and her production of *San Juan* of Max Aub, included in a scholarly text on the works of the playwright in Spain. In the late 1990s she began her activities as a communications consultant, providing seminars on interpretive reading, diction, and communication for professionals outside the theater, such as her recent collaboration with the *International Center for Information Management, Systems and Services* in Poland. Currently she is collaborating with the United Nations International Children's Emergency Fund (UNICEF) in projects using theatrical techniques for the dissemination of the International Convention on Children's Rights. She has a Doctor of Jurisprudence degree from the University of Pisa.