

# Process Affordances: Nudging Toward Change in your Organization

Michael Keeling

Net Health Systems Inc.

Pittsburgh, USA

[mkeeling@neverletdown.net](mailto:mkeeling@neverletdown.net)

**Abstract**—An affordance, in the context of human-computer interaction, is any action possibility that is readily perceivable by an actor. Good affordances gently nudge actors into performing the correct action at the appropriate time. A classic example is the so-called “Norman Door” first presented by Donald Norman in the *Design of Everyday Things*. Affordances are often considered when designing objects and user interfaces, but software processes too present action possibilities to actors. The best software processes rely on affordances to nudge teams into following best practices without even realizing it. The implication is that both psychology and sound engineering practices must be considered when affecting change. In this paper I will discuss the psychological factors behind affordances and demonstrate strategies for identifying and changing process affordances through examples based on my experiences.

**Keywords**—component; process improvement; affordance; process design

## I. INTRODUCTION

“The *affordances* of the environment are what it *offers* the animal, what it *provides* or *furnishes*, either for good or ill.”

– James Gibson, *The Theory of Affordances*

Ecological psychologists use the theory of affordances to help predict how animals will behave in the context of an environment. Artifact designers borrowed the idea to help make objects that are more readily usable by people. Software designers, in turn, built on what artifact designers discovered and applied similar ideas to user interface design. As software engineers one of our main charges is designing processes fit for a specific purpose within the context of a specific environment. We need process to help us build working software. I propose that process, like any designed artifact, relies on affordances for influencing how software engineers behave on a project. Thinking about a process’s affordances can help us better understand how software development processes work and show us what to change so teams can improve.

## II. THE THEORY OF AFFORDANCES AND SOFTWARE PROCESS

An affordance is a perceivable element, such as an object or idea, which directs an animal’s thinking toward a specific set of actions [1]. Sometimes an affordance is a real thing in the physical world while other times an affordance might exist as

an idea or perception. From a purely ecological perspective, affordances in an environment create niches which in turn place selection pressure on animals enabling evolution [2]. In other words, some affordances create advantages for animals while other affordances create disadvantages.

Donald Norman in *The Design of Everyday Things* extended this basic theory of affordances and proposed methods for leveraging affordances to better understand how to design objects. Norman proposed that by considering the physical attributes of an object, the purpose of the object, and the context in which the object is used as well as the abilities, background, goals, and culture of the user, designers can create objects that are easier and more intuitive to use [3]. Conversely, undesirable behavior can be discouraged by making objects more difficult to use. The idea that people can be “nudged” into taking certain actions by both the environment and the objects with which they interact has been exploited by object designers and entrepreneurs with great effect.

As an example, consider an oversized, comfortable chair. For an American, the comfort of a chair in a living room creates an affordance which encourages you to sit and relax. Consider that same oversized, comfortable chair in the dining room of a fast food restaurant. In this scenario, a chair which encourages patrons to remain in the dining room prevents new patrons from being served. In this environment, an oversized, comfortable chair is inappropriate and indeed, many fast food restaurants use uncomfortable chairs specifically to encourage diners to eat faster and leave, making room for more diners. Similarly an oversized, comfortable chair would seem extremely out of place in a traditional Japanese sitting room in which space is at a premium and other objects, such as pillows, are commonly used for sitting rather than oversized chairs.

I propose that process, generally, and software processes in particular, as designed artifacts, also rely on affordances for encouraging or discouraging behaviors among the teams that follow them. Steps in a process, though purely imaginary, directly influence teams’ behavior. For example, when was the last time you heard someone refer to a step in a process as “jumping through a hoop?” Though steps in a process exist only as ideas, processes are as much a part of the environment as any physical object.

Since a software process is part of a team's environment, a team will react differently to a process based on team members' backgrounds, experiences, and abilities. As software processes exist solely in our minds, the influence a process has over a team will vary based on the team's perception and understanding of that process. In other words, since process is an abstract idea, how well you understand a process will determine how a process's affordances influence you. If you don't understand a process well and execute it poorly, affordances designed into the process may be lost on you much like how red and green warning lights would not be easily perceived by a person with color blindness.

### III. AFFORDANCE DRIVEN PROCESS IMPROVEMENT

Studying the affordances of a software process presents interesting opportunities for team improvement efforts. The basic idea is to identify existing process affordances that are hurting or helping the team and replace poor or harmful affordances with better ones. The biggest problem is that when something is designed well, users shouldn't notice the affordances and how those affordances affect behavior. Therefore, to identify affordances for analysis it is necessary to consider an artifact (be it process or object) from the perspective of a designer.

Affordance Driven Design is a design methodology which shifts the focus from an artifact's functions to the affordances which nudge users to achieve a function [4]. By thinking about affordances in this way, they begin to look eerily similar to design choices affecting quality attributes from software architecture. By this I mean that an artifact's affordance might promote or inhibit specific functions in the context of an environment when performing a specific function.

In fact, thinking about process design in the same light as software design makes identifying affordances through reverse engineering a navigable task for software engineers. Rather than focusing on techniques and tools from ecological psychology or object design, we can reuse techniques already in our software engineering tool box, specifically the quality attributes scenario [5]. Quality attributes scenarios define a software system's behavior in the context of an environment while performing specific functions. Similar to software design, identifying functional requirements for a process and the qualities that process should promote or inhibit in the form of scenarios is one way to identify and evaluate process affordances.

The generic method for Affordance Driven Design has three steps. The result of this generic method is a list of affordances to be designed into an artifact [6]. As software engineers, we can apply principles learned from software architecture during the first two steps. The final step in the generic method is a simple prioritization using whichever technique your team desires. The steps of the generic method for Affordance Driven Design are the following.

1. Identify a user's needs in terms of functions.
2. Identify affordances which may help achieve the functions

3. Choose the affordances to design into the artifact which are most likely to help users achieve the function.

As an example, in one experiment focusing on affordances, subjects were asked to mix a cold drink using a blender [6]. The point of the experiment was to evaluate the affordances of the blender the subject used and how it influenced the person's use of the blender. The functions in this case were relatively simple. Example functions are preparing the blender, powering the blender, blending, and cleaning. The person's ultimate goal was to create a mixed drink. When analyzing the blender and how the person interacted with the device, designers considered several functional affordances, or what software engineers will quickly identify as quality attributes. These included "countertop-ability," "transportability," "clean-ability," and "mix-ability," among others.

In this experiment, all subjects were able to complete the task but blenders performed very differently in terms of the identified quality attributes. Some blenders had affordances which promoted cleaning while others inhibited it. And like software design, there are often trade-offs between qualities.

Extending this idea to software processes is relatively straightforward. All software processes have the same basic function, to build software, but different processes value different qualities. For example, Extreme Programming values "changeability," the ability to adapt a project quickly, while other processes might value "software requirements stability." "Plan-ability," the ability to see a certain distance into the future, might be more important in some scenarios than others. Once the quality attributes are known and quality attribute scenarios identified, finding the affordances in a process is usually simple.

### IV. CASE STUDIES

The remainder of this paper will discuss examples of process affordances taken from my experiences. For the purposes of discussion, I define a good affordance as any affordance which actively promotes desired behavior or inhibits undesirable behavior. Conversely, an affordance which actively inhibits desired behavior or promotes undesirable behavior is a bad affordance. A poor affordance is any affordance which passively promotes or inhibits behaviors, often resulting in inconsistent behavior. Remember that context, background, goals, and abilities are important when determining whether an affordance is good, bad, or poor. So a child-safety locking lid on a drug container might be a good affordance for a family with children (preventing accidental poisoning), a bad affordance for an elderly person with arthritis (unable to open the container to take needed medicine), or a poor affordance for a gorilla (who smashes all containers open with a rock regardless of the type of lid).

#### A. *Square Root: Changeability Affordances*

Square Root was a five person studio team from the Carnegie Mellon University's Master of Software Engineering (MSE) program [7]. SQUARE (Security Quality Requirements Engineering) is a nine step process for eliciting

security requirements [8]. The result of the project was a commercial-grade, web-based tool currently used by the SEI for research and education and by business customers using SQUARE on real projects [9]. During the spring semester (Elaboration Phase of the Agile Unified Process) the team was missing milestones, experienced confusion over tasking, and work was stalling in a variety of areas.

The team's planning process was relatively simple. At the beginning of the phase we examined all the activities and artifacts that needed to be completed by the end of that phase. For each identified milestone we specified entry criteria, general tasking, what it meant to be done (validation), and exit criteria. Milestones were estimated with planning poker and each milestone was assigned to a member of the team. The "milestone owner" was responsible for making sure the milestone was completed by either delegating tasks or working on it themselves.

As the phase progressed, more information became available and the plan needed to change. In spite of this, the team leader encountered resistance from team members about changes to the plan. Changeability was valued in the team's planning process, but there were affordances preventing change from happening.

First, our process encouraged us to plan more work than time allowed. This was due to a missing connection between day-to-day progress and the overall plan. Second, though the new team leader may have believed there was consensus, the team in fact did not wholly agree with the priorities for iterations. This behavior was not specifically discouraged by our planning process and so it was allowed to persist. In other words, there were poor affordances for communication and consensus. Third, leftover work was not addressed during planning. Some tasks might simply expire while others may change priority. This bad affordance created a sense of urgency for individuals carrying over work from iteration to iteration, encouraging individuals to resist change. Finally, assigning milestone owners had unanticipated side effects. The goal was to ensure that someone was taking responsibility for coordinating and monitoring milestone work. This worked so effectively that milestone owners exhausted themselves while attempting to finish milestones, and resisted changes that prevented them from finishing what they promised.

Once these bad and poor affordances were identified it was relatively easy to replace them with good affordances that promoted changeability and communication. To do this, the team abolished milestone ownership, created a task backlog, used task velocity from previous iterations to limit the amount of work any one person could take for an iteration, and planned iterations as a group rather than asking individuals to plan alone.

#### *B. Black Knight Technology: Turnaround-ability Affordances*

I was once a member of a 12 person analysis and testing team responsible for evaluating a real-time track manager of the US Navy. Software releases were made every six weeks so

it was important that feedback was returned to the development group as quickly as possible. To complicate matters we were the only beta-test group which regularly performed real-time, asynchronous tests. A side effect of this sort of testing, compared to flat file testing, was that test data varied from test run to test run depending on a multitude of unpredictable variables from the time of day to where a sensor simulator was pointing during a test run to network traffic. As a result, for any given test, multiple runs had to be performed and all the data analyzed for proper and consistent behavior. Identified anomalies required root cause analysis.

The test bed consisted of two separate networks, one for analysis and the other for testing, connected by a single shared server used for transferring data. The physical environment mirrored the virtual one with computers from each network running in separate rooms, joined by a single open doorway. The networks were disconnected in an attempt to isolate networking variables while running tests. Because of this environmental constraint, copying data (upwards of 4 GB at a time) from the test network to the analysis network took considerable time. Often test runs resulted in bad data, where portions of the system under test would silently crash. However since analysis was conducted exclusively on the analysis network, bad data wasn't discovered until hours later, resulting in wasted effort reconfiguring equipment for test runs which could have been run immediately had the testers known.

The turnaround-ability of the team's testing process was highly valued. In this case, there are several affordances which potentially inhibit turnaround-ability. Testers do not have sufficient computing power to quickly examine test data prior to transferring it to the analysis network. Configuring test equipment is difficult and error prone. Existing analysis tools are geared toward root cause analysis and not intended for peeking at data. There was a general attitude that "testers do the testing while analysts do the analysis."

Several affordances were altered to improve the turnaround-ability of the team's testing process. First we rearranged the physical environment, bringing two machines suitable for analysis onto the test network. Second, new analysis tools were commissioned capable of performing quick looks into the data. Third, traditional team roles were eliminated – analysts were given training on how to use the testing equipment while testers were taught how to do analysis. The result was a more cohesive team capable of performing more testing work with less time.

## V. CONCLUSION

When a process is not working for a team, it can be difficult to figure out what specifically needs to change. Focusing on the process's affordances is one technique for quickly identifying problem areas and shedding light on the qualities the team actually values. Affordances are subtle, usually by design, but by applying lessons from other design disciplines such as software architecture, it's easier for software engineers to apply psychological concepts which may not be familiar.

In each of the two presented cases, affordances in both process and environment nudged the teams to behave in ways that were contrary to their goals. Understanding those goals

and the qualities the team valued in meeting those goals was the key to identifying the problematic affordances and what new affordances to use.

Modern software processes are highly designed artifacts. Though thinking about affordances is not something generally applied to process, most process designers consider heavily how the techniques and activities in a process will influence a team. Before tailoring a process, taking time to understand the designer's intentions will help determine whether it's the affordance causing problems or the team's execution of the process.

#### REFERENCES

- [1] J. J. Gibson, *The Ecological Approach to Visual Perception*. Psychology Press, 1986.
- [2] A. Chemero, "An Outline of a Theory of Affordances," *Ecological Psychology*, vol. 15, no. 2, pp. 181-195, 2003.
- [3] D. Norman, *The Design of Everyday Things*. New York: Currency Doubleday, 1988.
- [4] J. R. A. Maier and G. M. Fadel, "Affordance-Based Design: Status and Promise," *Maier Design Works*, 2006.
- [5] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*. Addison-Wesley Professional, 2003.
- [6] A. B. Galvao and K. Sato, "Affordances in Product Architecture: Linking Technical Functions and Users' Tasks," in *Proceedings of ASME Design Theory and Methodology Conference*, Long Beach, CA, 2005.
- [7] D. Garlan, D. P. Gluch, and J. E. Tomayko, "Agents of Change: Educating Software Engineering Leaders," *IEEE Computer*, vol. 30, no. 11, pp. 59-65, Nov. 1997.
- [8] N. R. Mead, E. D. Hough, and T. R. Stehney II, "Security Quality Requirements Engineering (SQUARE) Methodology," *Software Engineering Institute*, Pittsburgh, Technical Report CMU/SEI-2005-TR-009, 2005.
- [9] (2009, Dec.) SQUARE Tool. [Online]. <http://www.cert.org/sse/square-tool.html>