TAMUHack Leadership Model

Introduction

My personal leadership philosophy has been an abstract idea until the formation of TAMUHack. As a cofounder of TAMUHack my leadership technique has solidified over time based on research and, ultimately, trial and error. It is loosely based on key features and design principles found in computer programming, namely encapsulation and the separation of concerns, along with general concepts found in shared leadership models. This leadership model repurposes the aforementioned programming features and principles into metaphors for the social and relational concepts of shared leadership.

TAMUHack

This leadership philosophy was tested on an event called TAMUHack, a club with the specific aim to throw a “hackathon,” which is a computer programming competition. Hackathons bring people from around the country (and world) to team up and compete against other teams to build cutting edge innovations using computer programming. TAMUHack was founded by five Texas A&M students. These students created a leadership team and set out to throw a five-hundred-person hackathon on campus. TAMUHack was to be a 24-hour event, free for all attendees. This included meals, snacks, a venue, judging, prizes, and more.

Programming practices

In computer programming, encapsulation refers to the ability of a language to have self-contained structures that cannot be directly accessed or manipulated by outside parties (1). A main topic that is explained in conjunction with encapsulation is
information hiding, which minimizes the amount of information available to outside code. Bits of packaged code can work together via interfaces, which are ways for the structures to minimally interact with and retrieve data from one another (2). All of these factors relate to the level of dependence between different parts of code. The goal of these concepts is to increase independence between elements. Code that is fairly independent (i.e. has a “clean” interface) is called “loosely-coupled,” and it is known that loose coupling is an indicator of good design (3).

In order to properly use encapsulation and information hiding to achieve loosely-coupled code, a programmer must understand an important principle called the “separation of concerns.” While designing a program one must first identify the end goals and objectives (e.g., “What type of data will this program handle?” or “How should the input data be organized?”). The answers to these questions determine the program’s architecture. The reason code organization and independence is so important is because the need to modify code is inevitable. Code must be adaptable and coupled loosely enough so that, perhaps, a new feature can be added in without disrupting the rest of the program. Additionally, it is ideal to have code that can easily accept new implementation from other existing programs (4). If an interface is clean and code is loosely coupled, a new module can be added in without much additional manipulation or complication.

Shared leadership principles

Shared leadership, leading with a group of people, contrasts individual leadership, which involves a single person’s management over a group. Some scholars believe that successful shared leadership depends on the effectiveness of task
delegation and collaborative, but unique, roles. The group of leaders must not only be able to carry out individual tasks, but also act as one distinct unit. The ideal dynamics of the interactions between multiple leaders is an interesting paradigm. Pearce and Conger state that these interactions should be multidirectional and equal rather than one-sided and hierarchical (5).

**Bottom Line**

It is the aim of this paper to seamlessly integrate the two seemingly unrelated topics of programming practices and shared leadership principles to create a novel theoretical leadership model tested through the execution of TAMUHack.

**Methods**

*Identification and separation of concerns*

At the beginning of the leadership process, tasks were identified and separated into five roles. Each leader in the group was assigned to a specific role. The five roles were: Chief Officer, Treasurer, Webmaster, Designer, and Marketer. Throughout the leadership process, additional tasks were given to the appropriate leader, depending on the skills required to complete the task. Designation of responsibilities to individual roles had the purpose of allowing efficient separation of concerns. Each leader was to be independent in task execution, and tasks were carried out without the help of other team members, unless a particular task was not specifically fitted to any existing role. In this case, it was set aside for the time being in order to later be assigned to an interface
**Establishment of interfaces**

Interfaces comprised the bounds of partnership between the five leaders. Just like in computer programs, these interfaces allowed two or more team members to work together and exchange information in order to complete specific tasks. At any point a leader could call for the creation of an interface if additional help to complete a task was needed. The interface would connect that leader to another leader who had the necessary skills to help complete the assignment. The purpose of these interfaces was to avoid unnecessary burdens on the other leaders.

**Officer restructuring**

The previous methods included ways to implement sophisticated encapsulation and loose coupling of leadership roles. The effectiveness of these precepts were tested when one of the five leaders of TAMUHack stepped down. The tasks assigned to that leader were reassigned, and the interfaces between that member and the rest of the leaders were severed and relocated to form new dependencies.

**Analysis of effective leadership**

The leadership process throughout the planning and execution of TAMUHack was deemed successful if strong parallels could be drawn between the leadership team and a well-designed program.

**Results and Discussion**

**Modifying concerns and interfaces**

Because this was the first TAMUHack (and the first hackathon thrown by any of the leaders), the list of identified concerns was lacking quite a few important entries.
This means that most of the leader interfaces were insufficient and required remodeling at some point during the leadership process. A specific example included the addition of the task of volunteer recruitment. Due to inexperience, the need for volunteers during the event was overlooked. The task involved volunteer recruitment and sign-up, and it was deemed too large a task for any one leader. Therefore it was assigned to the interface between the Treasurer, who recruited students from various clubs, and the Webmaster, who created the online volunteer questionnaire and database. This interface allowed specific communication between the Treasurer and the Webmaster that did not involve any other team member.

A few months before the hackathon, the Designer of the group had to step down due to unforeseen circumstances. The officer restructuring method was used to redistribute the Designer’s concerns (duties) to other leaders. This officer restructuring in the form of subtraction required the deletion and addition of new interfaces. The Designer’s roles were added to the list of concerns of all officers.

Efficacy of encapsulation

The requirement to add and subtract interfaces and delete entire dependencies (like the Designer) tested the efficacy of the encapsulation of the leadership system. Leaders whose interfaces were untouched by changing events had consistent efficiency in all tasks given. The small amount of required reorganizing minimized the damage to the team as a whole. The team was also successful during the hackathon because each leader was effective at completing tasks individually as well as completing them as a team. The fact that unnecessary details of one leader’s concerns were kept secluded from all other leaders allowed each person to complete tasks free from distractions.
Each leader could trust the independence of the other parts of the team knowing that the leadership processes were effectively encapsulated.

*Parallels with shared leadership*

Shared leadership models provided a guide for the multidirectional nature of the TAMUHack leadership model. The emphasis that shared leadership theories place on task distribution and independence parallels with the programming design principles used to create this leadership theory (5).

*Conclusions*

This leadership model succeeded in paralleling the TAMUHack leadership process with the programming principles of encapsulation and the separation of concerns along with common shared leadership principles. As previously stated, code that does not require heavy manipulation during the addition or subtraction of functionality is considered loosely-coupled and well-designed. It was deemed that the TAMUHack team was a successful team because it was metaphorically loosely-coupled, allowing the team to work independently and permitting simple interface changes. It was also effectively encapsulated, meaning that each member was in charge of certain tasks and, due to information hiding, was not concerned with other tasks that were not a part of his or her concerns or interfaces. Finally, the intrinsic parallels between shared leadership theory and programming design principles provided an excellent framework for the leadership model used to run TAMUHack.
References


