Harris Hierarchy of Software Development Support Needs

Michael Harris and Thomas M. Cagley, Jr.
David Consulting Group, Malvern, PA, USA

Abstract – This paper describes the authors’ hypothesis that software development organizations operate at different levels in a hierarchy of needs modeled on the Maslow Hierarchy of needs. The Harris Hierarchy of Software Development Needs has four levels representing operating states of the software development organization: Failure, Fear of Failure, Control and Value. The paper describes how the new model can be applied to software development organizations. It describes what an assessment might look like for an example organization with an actual assessment for a real organization. The authors suggest metrics for the assessment. Finally, the paper suggests events that might cause a positive or negative transition through the level of the model.

Keywords: Model based software engineering, Software Engineering Methodologies, Project Management Issues, Software Metrics

1 Introduction

This research paper describes a hypothesis that emerged, partially formed, from an internal research seminar conducted by the leading consultants of the David Consulting Group as part of continuing efforts to improve the software development of clients. The genesis of this particular hypothesis was an attempt to synthesize our many years of experiences with software development organizations into a set of categories that would enable us to better help existing and future clients to recognize their current and desired situation.

We have tested the hypothesis against our own experiences and found it to be a useful tool in most cases but the hypothesis needs to be validated against the experiences of others in the community. We will welcome feedback.

2 Harris Hierarchy of Software Development Needs

Recently, we were discussing our client’s reasons for needing help with their software development functions. As the need stories and examples flew, we found the same common thread kept coming up again and again: Failure, Fear of Failure, Control and Value. Moreover, it became clear that we were talking about different states that the same organization could find itself in at different times and in response to different internal and external changes. Further, that the model could be used at a more granular level to describe team level states within an organization.

With apologies to Maslow[1], we created the Harris Hierarchy of Software Development Needs (see Figure 1):

![Harris Hierarchy of Software Development Needs](image)

**Figure 1:** Harris Hierarchy of Software Development Needs

As with Maslow’s Hierarchy, software development teams can move up and down through the tiers as their circumstances change. Broadly, the more mature a software development organization is (Our Tier 4 or Value Tier), the more time it will spend being concerned about delivering value. Immature software development groups (Tier 1 - Failure) lurch from failure to failure dependent on heroes to pull success from the teeth of failure by working ridiculous hours or renegotiating the scope, duration or price of the project[2].
Between these extremes, the Tier 2 (Fear of Failure) Software Development (SD) organization experiences frequent, apparently random failures because they do not have repeatable processes - they continually operate in Fear of Failure mode in which estimates have ranges of +/- 50% or more. Our Tier 3 Control SD organization is mature enough to have repeatable processes but has not dealt with the governance issues that ensure it continues to deliver value to its internal (and external) clients. You can think of software development maturity in terms of CMMI®, Plan, measure, do or any other framework you’d care to choose.

It is important to recognize that software development organizations in each Tier could need support and coaching but that the nature of their goals is very different!

2.1 Does the Harris Model apply to the whole organization or some subsets?

In general, we suggest that the model can be applied at the level of granularity for which processes and metrics are, more or less, uniformly defined and implemented with vigor. In short, we have found that the model provide a rich source data when applied to teams within an organization because we have observed that different teams are often at different levels within even quite small organizations.

Implemented with vigor is an important qualifying characteristic of our framework which merits further discussion. Vigor incorporates people into the equation. In our experience good software development requires passion. That passion can be focused on the content, the method, the tools or even the process itself (for metrics work like us). We submit that an organization will never successfully achieve or sustain the Control level without some passion for process in the organization. We respectfully suggest that the success of agile development stands on several pillars but one of them is defining a process that developers can be passionate about.

By way of contrast, we considered a few other candidate characteristics such as simply, implemented or implemented with rigor but these definitions lacked the active engagement element that we consider to be essential at the higher levels of the model.

Identifying teams as the expected granularity at which the model can best be applied reinforces the involvement of people in the model since this tends to be the smallest unit of people interaction in a software development project. Interestingly, we have often observed situations in which even teams operating at the failure level can be seen to be implementing with vigor. Consider the plate-spinning magician rushing from pole to pole to keep the plates on top from crashing to the ground by adding more spin. No shortage of vigor there! Vigor in this case portends the ability to change.

We have seen several organizations in which the absence of process definition and the dependence on a few heroes was implemented with more vigor that most. It is precisely this combination that allows small organizations such as start-ups or a small group in a major organization to operate at the failure level by generating significant innovation. However, this combination tends not to be scalable and can make a slightly larger organization (or the same start-up after some success) operate at the failure level for a period of time.

2.2 So we have the model, how can we use it?

The first intended use of the Harris Hierarchy is as a diagnostic tool to segment software development organization teams into performance levels and to identify the appropriate improvement strategies. To enable this, we have included in this paper some characteristics of teams at different levels, some metrics that might support identification of a level and some causes of upward and downward transitions.

It is important to note at this point that the Harris Hierarchy is really a grey scale and that we can certainly see sub-levels within the four major levels. For example, we believe that within the Fear of Failure Level, close to the boundary with the Failure level is a sub-level that we call Fear of Imminent Failure. For a company, transitioning from the Control level, we often see that Fear of Failure does not capture management attention (it should!) anywhere near as much as Fear of Imminent Failure. Another example of these sub-levels occurs in the Control level where we have observed that teams operating in the lower half of the Control level close to Fear of Failure tend to focus on what we call, defensive control. Defensive control focuses on processes, metrics and reporting to avoid being caught out by management. By contrast, teams operating in the upper half of the Control level, are starting to focus more on processes, metrics and reports that try to measure the value being delivered even if this sometimes makes them look bad in the short term. We believe it is very possible to be at the Control level and not delivering much value!

We believe the Harris Hierarchy has another secondary value for software development leaders in building teams for different types of projects and detecting and interpreting the impact teams moving from one level to another which they will almost certainly do from time to time either upward or downward with without noticing the transition.

2.3 What are some characteristics of teams that are unique to the different levels?

Please see Table 1.
Table 1: Characteristics of teams at different levels of the hierarchy

<table>
<thead>
<tr>
<th></th>
<th>Failure</th>
<th>Fear of Failure</th>
<th>Control</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Schedule</strong></td>
<td>Partial deliverables of customer schedule</td>
<td>Deliverables close to customer schedule</td>
<td>Deliverables on customer schedule</td>
<td>Deliverables on customer schedule</td>
</tr>
<tr>
<td><strong>Content</strong></td>
<td>May be excellent but quantity or quality fall short of customer needs</td>
<td>Inefficiencies of process that are not under control mean that there is not time to deliver the desired quality of content</td>
<td>Exactly what the customer ordered</td>
<td>Exceeds customer expectations</td>
</tr>
<tr>
<td><strong>Quality</strong></td>
<td>• Growing defect backlog. • Tier 1 support under constant stress • Severity 1 defects in acceptance testing</td>
<td>• Growing defect backlog. • Tier 1 support under constant stress • Severity 1 defects in acceptance testing</td>
<td>• Stable defect backlog • Predictable defect levels in releases • Predictable turn-around time for bug fixes</td>
<td>• Most defects captured before delivery to customer • Rarely a subject of discussion with the customer</td>
</tr>
<tr>
<td><strong>Cost</strong></td>
<td>Estimation[4] is completely unreliable and often relies on SMEs</td>
<td>Frequent estimating disasters destroy confidence in process</td>
<td>Estimation based on historic data and predictable costs are achieved</td>
<td>Estimation based on historic data and predictable costs are achieved</td>
</tr>
<tr>
<td><strong>Customer Satisfaction</strong></td>
<td>Poor</td>
<td>Mixed</td>
<td>Satisfied</td>
<td>Wowed</td>
</tr>
<tr>
<td><strong>Team Members</strong></td>
<td>Heroes and villains</td>
<td>Nervous nellies</td>
<td>Efficient and effective professionals</td>
<td>Value Providers</td>
</tr>
<tr>
<td><strong>Release Management</strong></td>
<td>Unpredictable</td>
<td>Subject to occasional disasters</td>
<td>Sound</td>
<td>Sound</td>
</tr>
<tr>
<td><strong>Risk Management</strong></td>
<td>Non-existent</td>
<td>Too high-level</td>
<td>Structured</td>
<td>Customer-facing</td>
</tr>
</tbody>
</table>

Table 2: Harris Hierarchy Metrics

<table>
<thead>
<tr>
<th></th>
<th>Failure</th>
<th>Fear of Failure</th>
<th>Control</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Overall</strong></td>
<td>Consistent Performance Below Expectations</td>
<td>Inconsistent Performance</td>
<td>Consistent Performance ALL Metrics</td>
<td>Consistent Improvement of ALL Metrics</td>
</tr>
<tr>
<td><strong>Schedule</strong></td>
<td>Inconsistent Performance (generally below baseline)</td>
<td>Dates met and performance generally below baseline</td>
<td>Dates met and performance generally above baseline</td>
<td>Schedule performance consistently above baseline</td>
</tr>
<tr>
<td><strong>ROI</strong></td>
<td>Not Measured</td>
<td>Cost Side measured</td>
<td>Cost and Benefit predicted and measured</td>
<td>Full Life Cycle ROI monitored and actions taken to attain</td>
</tr>
<tr>
<td><strong>COST Per Unit of Work</strong></td>
<td>Inconsistent Performance (generally below baseline)</td>
<td>Cost performance generally below baseline</td>
<td>Cost performance generally above baseline</td>
<td>Cost performance consistently above baseline</td>
</tr>
<tr>
<td><strong>Defects Per Unit of Work</strong></td>
<td>Inconsistent Performance (generally below baseline)</td>
<td>Defect Density performance below baseline</td>
<td>Defect Density performance generally above baseline</td>
<td>Defect Density performance consistently above baseline</td>
</tr>
<tr>
<td><strong>Process Assessment</strong></td>
<td>Primarily Ad-Hoc or not formally assessed</td>
<td>Primarily Ad-Hoc but formally Assessed</td>
<td>CMMI (or similar) Level 2 or better</td>
<td>CMMI (or similar) Level 2 or better with improvement plans</td>
</tr>
<tr>
<td><strong>Customer Satisfaction</strong></td>
<td>Poor</td>
<td>Inconsistent</td>
<td>Consistent Average or Better</td>
<td>Satisfied Customers</td>
</tr>
<tr>
<td><strong>Re-organization Half-life</strong></td>
<td>3 -6 Months</td>
<td>6 - 12 Months</td>
<td>12 - 18 Months</td>
<td>18 Months or More</td>
</tr>
</tbody>
</table>
2.4 What Metrics might be indicators of one Harris Hierarchy level or another?

See Table 2. Pretty good metrics provide a spotlight to tell you where you are going because you are always changing. In building our model we have found that a robust pallet of metrics is required to meet the definition of pretty good metrics. To work, the pallet must be made up of metrics that change as a team or organization transits through the model and each metric needs to evaluate different aspects of the model than the other metrics in the pallet.

A final characteristic of any metric in the pallet is that it can actually be deployed (not just an academic idea). The metrics pallet supporting the model has two levels. The first level, designated overall, provides an indication of how the metrics should interact in aggregate. For example, the metrics of a team at the Failure level[5] would be consistently below expectations. As the team moves away from Failure they become more inconsistent. As a team moves from Failure to Fear of Failure, performance tends to swing between deliveries that are below expectations, meeting them and occasionally showing flashes of superior performance. Finally, arriving at the Value level, all metrics should be above expectations and still show consistent improvement.

Table 2 names each metric and describes how the performance varies at each level. The authors recommend creating a baseline for each metric. This will allow you to validate your appraisal of a team level and to determine whether changes to the team are effective.

2.5 What might a Harris Hierarchy Assessment look like?

We can anticipate a number of different ways of looking at the data from a team-by-team evaluation of Harris Hierarchy levels but, as a starting point, we always ask ourselves, What decision will be made when this information is presented to management? In this case, when we report to management that we see their teams operating at a variety of Harris Hierarchy levels, they tend to ask something like, Do I need to do anything about it today? and then How can I prioritize? Based on this type of reaction, our favorite initial chart is a bubble chart that captures the teams Harris Hierarchy levels, the business impact of the project they are working on and the size of the team in terms of full time equivalent people (see Figure 2).

An analysis of this this presentation clearly shows that teams B and C should be the first candidates for attention because of the potential to impact the business. Teams D & E need attention but it might be more efficient to disband them than to try to improve then given the relatively low business impact. Team A is interesting and a type of team quite often. The team is a reasonable size and it is operating at a high level in the hierarchy therefore it is usually the type of team that gets highlighted as evidence that we are doing just fine thanks. That certainly true for team A (they may have done CMMI, ISO 9001 or the like). However, especially given where it sits in terms of potential business impact, is that really true for the organization as a whole? A discussion of redirection would make sense for teams in this quadrant.

Figure 2: Example Harris Hierarchy Assessment

2.6 What evidence do we have of the applicability of the model so far?

2.6.1 Provider of large, mission-critical software products to the telecoms industry

When we were introduced to this company, they had some of their teams operating at the Control level (see Figure 3 teams A, B, C & D) while some were operating at the Failure Level (teams E & F). Unfortunately, teams E and F were very visible to customers because they were new products with heavy user interface elements. The teams were part of acquisitions for which the software development management team did not feel consulted and, hence, they did not regard themselves as accountable. Clearly, the solution here was to apply the Control practices to the new teams. However, there was resistance at first because of the fear that the new teams would mess up the nice metrics of the old teams. One of the ways that the Control level had been achieved was to virtually exclude the rest of the company from the development process. As a new organizational structure with new personalities was introduced, control over the content and budgets for software development moved outside of the software development organization and the resulting culture change dropped one team (team D) from Control level down to Fear of Failure level. We have seen this happen in other
organizations where the "Control" level is sustained through a rigid, almost bureaucratic, approach which does not deal with change well.

Figure 3: Real Harris Hierarchy Assessment

2.6.2 Provider of customized software solutions to the Department of Defense using SCRUM

In the eyes of their clients, this company clearly operated at the "Value" level which won them more business. However, as the size of their projects grew due to the absence of controls, they quickly dropped to the "Fear of Failure" level. One of the areas that exhibited the impact of the reduction of capability was release management. Without controls they quickly lost control of release management which was an essential part of their Agile SCRUM value proposition. Implementing a lean version of parts of CMMI enabled them to return to the "Control" level and from there to plan a return to the "Value" level.

2.6.3 Provider of large, mission-critical software products to the banking industry

Prior to implementing a CMMI program to provide a focus on structure and discipline, this company operated at the "Failure" level even though they were perceived as highly successful. The issue was that only one of forty clients would provide a reference and the relatively small group of employees who could successfully execute was run ragged as the company grew. After implementing the CMMI program, the company moved up to the "Control" level but soon dropped back to the "Fear of Failure" level as management focus on the CMMI program shifted elsewhere.

2.7 What keeps companies in one state rather than transitioning?

Our early sharing of this model outside of the research group most often generates a question of how stable is the "Failure" level. Our observations suggest, it is an unstable level. Organizations or teams must eventually move up or collapse. The rate of collapse and rebuilding can be measured in the average half-life of organizational restructuring or CIO turnover. The lower the half-life (use the last three reorganizations as a baseline) the larger the impact of failure caused by churn on the organization. Unfortunately, we see all too often that an excessive presence of passion in the form of heroes or an excessive absence of passion for the level of discipline that can stop the pain. In these cases organizational inertia can cause teams to wallow in the failure level far longer than we might like or expect. Either failure is "understandable" because the team tried so hard or "normal" because it's always been that way, hasn't it?

At the higher levels, companies stay in one state by keeping their discipline alive and relevant to the day to day work of the development teams. Here again, organizational stability is a quick means of measuring the impact of higher levels.

2.7.1 What can cause upward transitions?

- Intervention
  - Positive reinforcement of best practice
  - Change of non-productive behavior to best practice
  - Near death experiences
- Success
  - Innovative or accidental behavior that works (and is identified and reinforced)
- Removal of control from the team
  - Breaking of "traditional" behavior patterns for example moving away from a subject matter expert generated estimation model to a centralized parameter-based, historic data-driven estimation model.[6]
  - Implementing self-directed team models based on shared goals[7]
- Investment
  - In the implementation of specific best practices such as CMMI or Agile. While there is always an internal component to this investment, a key to success here is to bring in the right external consultants as software development experts and change agents.
- Outsourcing[8]
  - Sometimes the outsourcing of software development to a third party allows the organization to leverage the best practices of "experts" whose sole business is software development. Sometimes even the need to define processes to make outsourcing possible improves what had gone before.

2.7.2 What can cause downward transitions?

- Intervention
A sense of lack of control caused by the intervention of “management” can de-motivate the most passionate contributors.

Lack of consistency of purpose (intervention of the day)

- Success
  - Different size teams require different processes to generate the same results. This cannot be ignored as success causes the team to grow.
  - Past success based on ad hoc processes or heroism in a dynamic environment. Past performance does not ensure future success in this environment.

- Removal of control from the team
  - Successful software development requires detailed knowledge more than most engineering activities and the most detailed knowledge is usually at the bottom of a hierarchical team not at the top. Managers ignore this reality at their peril.

- Unrealistic expectations
  - Unrealistic expectations are a guarantee of failure and repeated failure is a trigger for a downward transition.

- Outsourcing
  - Outsourcing bad software development practices will not improve them
  - Introducing outsourcing necessarily involves changes to process flows and the threat of mistakes

3 Conclusions

The Harris Hierarchy of Software Development Needs is a new concept that we believe adds value and should be included in organizational tool sets to help managers to identify software development teams with issues through the prism of a disciplined framework.

The ideas that we have provided in this paper are intended as starting points for further development by practicing software development managers.

4 References

[1] Maslow’s Hierarchy of Needs:
http://en.wikipedia.org/wiki/Maslow’s_hierarchy_of_needs


