Philosophical Patterns In Software Development

The patterns community has several examples of organizational patterns. There have been many organizational patterns written that have a philosophy behind them that is not explicitly identified. Here we are attempting to document patterns we have used to develop and verify approaches to projects. We also believe these patterns lay the foundation for many organizational and other patterns or justify some of the things we do to build effective teams and processes. Since these are *philosophical* patterns they are (at least intended to be) about *mindsets*, not concrete *how-tos*. The how-tos are the organizational patterns and such.

This collection of patterns is intended for people who desire to be a change agent for more effective software development. The basic premise is that without a solid foundation on which to frame the use of organizational patterns, it is easy to choose the wrong patterns or miss the forest for the trees. One should avoid building their approaches on vain philosophies or faulty world-views that do not stand under the pressures of reality (see Matthew 7:24-27 and Colossians 2:6-8).

This is a work in progress. We have seen the effectiveness of the use of each of these patterns across many projects both directly and indirectly. We believe the patterns are tied together, but we don't have a good feel yet for how to show their connectivity.

In the meantime, here are the list of the philosophical patterns we've identified:

Valued Individuals

How do you maximize employee productivity?

- Everyone has their own unique combination of gifts and talents and are created by God.
- Today, the current workplace experiences a lot of turnover… people are not loyal to their employers.
- However, if you treat the employee as if you think they are short-timers, you may be the creator of a self-fulfilling prophecy.
- Turnover has a very high cost that often goes unnoticed by bean counters.
- Employees who do not feel valued, tend to do the minimum amount of work necessary to stay employed and are often distracted from the common good.
- Pay and benefits alone are not sufficient incentives for most people to say on jobs, at least in the software world.

Value each person on or related to the project. Figure out their strengths and weaknesses. Find roles, whenever possible, that take advantage of their strengths and publicly recognize how their unique strengths applied to that role are very important to your project. Recognize their unfulfilled potential and value them as if it was fulfilled. Help cover for them in their areas of weaknesses by not demanding too much of them when placed in unnatural roles and providing more assistance in those roles. Emphasize their value as an individual human being with unique needs and help meet those needs whenever possible without compromising the integrity of the project (e.g. flexible work hours, special tools, etc.). Apply the Golden Rule (Matthew 7:12) and the Great Commandment (Matthew 22:36-40).

Encourage others on the team to recognize the value of the individual contributors. When a person does not fit on a team, make every effort to help them find a new position. Don't just live with a bad fit and hope that by avoiding the issue it will go away. This is a sure cause for low morale not only for the person in the wrong role but the others around them. By allowing a person to suffer in the wrong role, you are sending the signal that the role is of more value than the individual in the role. Make this priority one. Jesus Christ gave his life for individuals, you can at least go an extra mile to fix this problem.

It is important to note that you cannot just "treat them as if you value them", but you must truly value them. Because, if you don't, under times of stress (or even just day to day goings on) the truth will come out. (Proverbs 4:23). If you find you do not truly value them, you must first humble yourself (Philippians 2:3-8)

The result is that team members will spend more of their time on work-related efforts, the output will be much higher quality, the individuals will be more willing to stretch themselves and grow. Through your example, others will begin putting others interests ahead of their own. You'll sleep better at nights knowing that you are honoring the individuals that God loves. They'll go the extra mile when they see that there is someone who values and appreciates them. You'll gain friends.

On one project (UI project) in which the author was project manager, we started with a kick-off meeting. Instead of the "this is what we're going to do" speech, we asked each individual to answer the questions:

- What do you expect to get out of the project?
- What are you willing to put into the project?
- What special talents and skills do you think you can uniquely contribute to the project?

We then immediately started sorting out roles and when there was not perfect fits, we looked for willing volunteers. When reluctance was sensed, we assure the volunteers that if the role became a burden to them, to raise the issue and we'll try a different approach. As the needs of the project changed and skills of the individuals became apparent we made adjustments in roles. Every time I (as project leader) felt we needed to make an adjustment, I'd ask everyone involved in the potential adjustment how they felt about it and if they did not like it, whether they had other suggestions as to how I could get the job done. I then offered the adjustments as a proposal to the rest of the group and we discussed it. This raised the awareness that matching people and roles was extremely important. Morale was very high. It became apparent after a while that there was one individual who didn't really fit into any of the necessary roles. We gave him as much notice as possible, helped him find a new job, and wrote letters of recommendations highlighting the unique talents and character traits the individual had. Productivity and morale rose even higher, and the individual was disappointed but grateful and relationships remained intact as he found another job with my help.

On the largest and most difficult project of which I've ever played a part, I was brought in to help after a huge mess was already well established from both a technical and project management perspective. Eventually, I was asked to lead an "integration team" to try to pull things together. I knew that I'd need someone to oversee the configuration management who would pay a lot of attention to technical details if we were ever going to get the stuff with which we were starting to co-exist. I knew there was one young man who was very frustrated in the difficult positions he had previously been put in on the project and did not have the maturity to respond appropriately. Although everyone knew he was very bright, no one wanted to put in the energy it would take to develop him into a real asset, and many had discussed dismissing him. I believed he had the raw material necessary to excel at this critical role. I requested him to be transferred to my team and told him I was confident he would do a great job although he had a lot to learn. I told him that I needed someone to fill this position well in order to free up my time to tackle other issues that needed to build on top of it and I would help him along the way. He was thrilled to have the opportunity and poured his energy into doing a great job. Unfortunately, he often generated problems due to his lack of tact in communication and his visible lack of patience with others less knowledgeable who did not seem inclined to learn. I probably invested at least 4-6 hours per week managing this young man and putting out the fires around him. After several months, all involved realized how valuable a resource he was in spite of his volatility and he learned a lot about how to handle himself in less inflammatory ways. Soon after, his help was being sought by many. Although he still wasn't everyone's favorite person, he was well-respected and no one even entertained the thought that there could be anyone better suited for the job. The 4-6 hours per week seemed like a lot, but I can't imagine how I could have accomplished anything else we did without him there. He has gone on to much greater heights since this time.

In an earlier situation when I was a younger, brasher man in the midst of dealing with a very difficult client, making every effort to do my best job while the client kept changing the rules, I was at the end of my rope. My manager (who was also the owner of the company) took time after working hours to talk to me. His goal was to find a strategy that would work, but during the conversation he said, "Ken, I want to do everything we can to make this work, but if it comes down to choosing either you or the client, it will be you every time." I went back to work with a completely different attitude the next day, and although the client didn't get much easier to please, it just didn't seem to matter as much.

Organizational patterns which seem to correspond with this pattern include *Domain Expertise in Roles* and *Self-Selecting Team* [Cope] and - somewhat - *Diversity of Membership* [Harrison]

Shared Vision AND Shared Reality

How do you keep employee morale high?

- There has been much written about the importance of a shared vision
- Often issues come up to cloud or challenge that vision
- When challenges are ignored or discounted as "bad attitudes" individuals not only question the value of the vision, but their own value on the team

Therefore, develop a vision, sharing the development and maintenance of that vision with all involved. When it is challenged, take the time to determine what the root cause of the concern is and acknowledge when the realities found may require an adjustment to the vision.

Later on the same project, due to some late delivery of some critical path items, we were asked to step up our efforts to hit a hard deadline. We did not agree to "pull out all the stops working days, nights, and weekends until the deadline was upon us". We instead took a long lunch (paid for by the company) and discussed how much work was left and how much time we were willing to put in for how long. We discussed the fact that we all had families and that we did not want to jeopardize our relationships at home, yet wanted to do a good job. We also agreed that each should commit to a certain number of hours up front and not expect anyone to do more than he was willing to do. We all agreed to 55 hour weeks for 6 weeks which we felt could get us to delivery two weeks later than requested and cost about 20% more than originally estimated. We also discussed that there was little room for error and that we had to track things so we would know as soon as we were in trouble. We delivered this commitment along with the potential risk to the client which we committed to manage via weekly updates to project plans. We also identified all of the reasons we felt the overrun was occurring (our contribution and the client's). Since the client accepted the reality of the situation, we could proceed and we hit our commitments while keeping morale fairly high. When tempers got a little short near the end of long weeks, we acknowledged that we needed a break if we were going to sustain our momentum and did something to lighten the pressure.

Organizational patterns which seem to correspond with this pattern include *Unity of Purpose* [Harrison]

Progressive Foundation

No building is better than its foundation. How do you ensure your project's foundation is sound?

- Software (and many other things in life) must be developed in changing environments
- Software development is currently more of a craft than a science, and as such, has no "first principles" in which to define a foundation
- As a craft, it is not totally chaotic and layered architectures have proven very valuable. Typically, one layer builds upon another.
- If people don't understand the foundations upon which the project is laid, whether in the way people are handled or the way software is built, they are bound to stray off it.

Therefore, develop the foundation of your software project, processes, and organization a little at a time. As the small parts are proven effective (or holes are identified) and well understood, more can be added. However, never let the foundation erode. It should always be steadily improving as each improvement adds more leverage to the big picture. Never let people forget the basic principles (Colossians 2:6-8).

During this same project, we found that the time pressure was often causing developers to hack together changes to the underlying and evolving framework that made the framework harder to understand. We found that as we added functionality the underlying framework had bigger holes. So we decided to create the role of the framework czar. His job was to both keep the framework clean and to make sure the framework provided the functionality any developer thought that it should and could justify by having at least two examples where a feature was needed. By the time the project was near completion, new windows could be designed and implemented in a matter of hours.

On another project, multiple products which were supposed to work together were developed almost completely independently although there was a small database access architecture which several groups shared but some refused to use because of some inherent problems. When duplication of effort in other areas was recognized, a small "core team" was formed. The goal was to have no duplication of effort. But, the core team had to start slowly. After a few key shared components were identified, roles were defined but there were few teams willing to devote resources to fill them. The core team became a service team which helped the individual teams incorporate the "core" into their products. As the individual teams began to see the benefit of establishing a core architecture, they began to be willing to commit a small amount of resources. The database access architecture was revamped and incorporated into some of the cooperating teams products with great success. Eventually, the communication between the core and individual teams was exemplary and more efficient than could have been designed up front. The individual teams were praising the benefits of the leverage they were getting from the core team and shed their old ways of ignoring the benefits of a solid foundation in both the architecture and communication techniques. The individual teams that had originally rejected the database access architecture then decided to incorporate it and join in on the group efforts as the benefits to the "improved foundation" were obvious.

Organizational patterns which seem to correspond with this pattern include *Architect Controls Product* [Cope] and *Technical Memo* [Ward],

Added Value

There are so many tools/techniques/approaches… how do we know which to use?

- Like individuals, each tool, technique, and approach has its own strengths and weaknesses.
- Unlike humans, the tools, techniques, and approaches don't care how they are used and won't tell you when they are being used inappropriately
- Each tool, technique, or approach takes time to learn and apply

Therefore, before using a tool, technique, or approach, determine what it is intended to produce and whether it fits the developer using it (see Valued Individual). If you believe that the result will produce a valuable artifact, try it on a limited basis. Never blindly use the tool, technique, or approach across the board. Instead, before each use, ask the question, "Is the cost of using this tool, technique, or approach obviously outweighed by the benefit of the artifact produced by it". Obviously there are times when not enough is known to make a confident decision about it. In these cases, do your best at calculating risk and reward and make a decision to go or not go. Don't assume the best approach for one case is that of another. As the tool, technique, or approach is better understood, help others know when the tool is most likely to provide added value. Beware that tools that don't naturally fit the bent of a particular developer whom you think should use it may require a costly adoption process. If you are sure the benefit is there, take the time it takes to provide the proper education to have the developer see the benefit.

In the UI project, it was determined that we needed to convey key design information. However, because we were writing in Smalltalk, it was often easier to browse code than read design documents to understand certain details of a design. Rational Rose was used to give a high level overview of the pieces of the design, but creating the diagrams were cumbersome. Therefore it was determined that it would be easier to communicate the design via a PowerPoint presentation which became the design document, a Cookbook, and a browser. Because the cookbook and design document only contained a handful of Class Diagrams and one or two Interaction Diagrams, only one developer on the project bothered to learn the tool in detail. No one ever complained about the lack of documentation with respect to either of these two diagrams. The developer didn't mind being the only one to draw the diagrams because there weren't many to do.

Organizational patterns which seem to correspond with this pattern include *Developer Controls Process* [Cope] and the discussion on developers being at the center of the picture in *Patterns of Productive Software Organizations* [Cope/Harrison].

Necessary Failures

How should one respond when things don't go as planned/hoped/expected?

- No human is infallible. Neither is any process or tool developed by a human.

- Success is desired and must be the common goal

Therefore, treat "failures" as opportunities to learn and ensure that the final product will be better because we learned. Edison found over a thousand ways that would not result in an electric light and thought of these failures as learning opportunities. In fact, behind most successful men, there are a lot of admitted failures. If you don't have many failures (wrong decisions) you may not be tackling the problem aggressively enough and you'll find that you may have hidden failures (missed opportunities) which might be a bigger problem in the long run.

In the UI project, there was a lot of effort put into the design of "editors" that would allow the user to make temporary edits of an object, but not make those edits final until the user said so. Mindlessly trying to apply the *Command* [GOF] pattern would have created more problems than it solved for a variety of reasons. We had to take a more aggressive attack to avoid these problems. We came up with what we thought was a powerful solution that seemed to work seemlessly well in early tests. The design was reviewed by several developers outside of the project and a large team of developers at the client and all of them thought it was excellent. Only a few short weeks before final delivery, a fundamental flaw in the approach was uncovered. It was determined that an entire new approach had to be taken. Instead of firing all the people that said it would work, those that created, reviewed, and approved the first design were brought back together to take a 2nd stab at it. Additionally some outside assistance was brought in to bring in any new insight. As the various proposals were entertained, the original designer had a suggestion that seemed far superior to the others. It was tried and again a completely different fundamental flaw was determined. After a 3rd series of brainstorming, the original designer came up with a 3rd strategy which seemed better than other proposals. It was implemented and proved to be a very simple yet powerful approach that far outweighed in value any of the other proposals that might have worked.

Revised, Realistic Resource Application

What do we do when we find ourselves limited by our budgeted resources?

- Projects are started with limited resources.
- There is political pressure to stick to these resource limitations.
- Team members using resources know best what is possible using them, even though this is an imprecise measure.
- Terminated projects often net a higher cost than completed projects since there is nothing to recoup costs with at the end of the day.
- Cost estimates are just that. If we adhere to them too strictly we can end up causing projects to fail because we apply resources in the wrong way or prevent critical resources from being applied at all.
- Once money is spent, there is typically no way to recover it.
- Because of the competitve marketplace, there is also pressure to promise the moon.

When determining whether to add more resources or pull the plug, make the decision based on where you are today, not whether you made mistakes in estimation in the past. You've learned and are hopefully in a better position to determine whether you'll get a return on "future" spending. Unlike Congress, people on most software projects can't spend money or time they don't have. However, also unlike Congress, we have a potential to get a return for our spending. Do your best to be realistic about what can be accomplished with the resources available and find creative ways to shift resources from one bucket to another.. But if more are needed to reach a desired goal, state so and reevaluate as suggested. If there is a way to get additional resources in order to get an eventual return, do it and don't agree to continue with resources that won't do the job. If your superiors won't deal with reality, it's time to find another job rather than to participate in their folly. Your foolish superiors are not the people who really provide for you (see Matthew 6:25-33). You can only promise the moon once. When you don't deliver it will cost you more than the deals you landed got you (Proverbs 22:1-3)

Note that this is not an excuse to be unrealistically low in original project estimates. Estimates should also be made with integrity. The better they are, the less often you will find yourself having to deal with some of the undesirable consequences of applying this pattern. Also, every effort should be made to live within the constraints of the estimates if it can be done with integrity.

It is often good to reapply Shared Vision AND Shared Reality when found in a position in which you had hoped not to find yourself. When temporary patch jobs are done as stop-gap measures, label them as such but continue to make sure the real ramifications are understood by all. Determine whether these measures are good enough to get the return you want without suffering other consequences (e.g. poor quality image). Of course, don't be afraid to pull the plug on the project if it makes sense. Don't fall into the temptation of saying "we've already spent $5 million, so we better finish". (Proverbs 11:2).

In the UI project, there were two windows that had some very unique behavior. It was thought that this behavior should be incorporated into the underlying framework. Unfortunately, by the time these windows were started, we were right up against the (revised) deadline. We hacked the two windows together and met our deadlines. We labeled the hacks as such and put it on a list of things to fix when we got around to it. We never have. There were other things more important and no one has really been able to justify spending the money to fix this problem of which no one else has really taken notice except the original programmer.

When making the decision to spend more money to complete the UI project, the manager realized that he had a choice of killing the project and taking a loss of funds to date (~$200K) or spending an additional $50K to get the original $250K return he expected. The decision at this point was not whether he should proceed with a "break even" project. He wisely chose to make an investment of $50K that would gain him $200K from this point on rather than spend $0 to get $0… the original $200K was already gone whether he proceeded or not.

Organizational patterns which seem to correspond with this pattern include *Size the Schedule* [Cope] and *Recommitment Meeting* [Ward].

Studio Approach

How do we produce great software and increase in our ability to do so?

- Software is more of a craft than a pure science or pure art
- Building non-trivial software is a collaborative effort
- There is much to learn via self-study
- In virtually every craft, the accepted best way to learn is under the guidance of a mentor as an apprentice
- It is very difficult to assemble a team of master craftsmen
- If you want more master craftsmen, you often have to develop them.

Rely on the studio approach, craftsmen working together to produce an excellent product through a collaborative effort toward a common goal whose whole is greater than the sum of its parts. If the required expert craftsmen are not already part of the team, bring them in as guests from the outside to assist your efforts. Assume that the best way for people to learn to become craftsmen is by learning in a studio, but don't expect the apprentice to produce what the master can when he starts. In fact, expect that the productivity of the apprentice to the master craftsmen is orders of magnitude difference at the start. Yet, expect that gap to close by making sure everyone in the studio knows why the apprentice is there. This may mean that the master at times stops producing at his typical rate in order to impart wisdom to the apprentice(s). Build this time into the schedule. Expect the apprentices to be pointed to individual self-study by the masters, and dialogues between the apprentices and the masters about this self-study is also a necessary part of effective learning. Realize masters are still learning themselves. Don't be unrealistic about the impact the number of apprentices vs. masters will have in the schedule. People are not interchangeable parts.

In the UI project, the framework czar was also the most experienced application developer. We were sure not to give him many deliverables, this kept him free to be able to help another developer with individual problems. Frequently, 30 minutes from the framework czar could move application developers over a hurdle they've been struggling with for hours. Each person on the project learned and contributed including a co-op programmer who had very little programming experience before this project. At the same time, there were some specialty roles that only one person had to learn "from the book". If someone else needed to know something they learned it in a fraction of the time by getting a few minutes of the other developer's time.

The C3 project at Chrysler has had great success via an approach they call eXtreme Programming. Although there are specialists, they claim that at least as much of their success is due to the studio environment under the tutelage of the master craftsmen Kent Beck and Ron Jeffries and applying techniques that focus on the product rather than the individual artist.

Knowledge Systems Corporation has offered its Smalltalk Apprentice Program for many years and most who have made the investment in this "studio approach" have firmly believed they have far exceeded their return on investment.

Organizational patterns which seem to correspond with this pattern include *Team Validation* [Cope] and *Validation By Teams* [Harrison].

Acknowledgements

Special thanks go to Neil Harrison who shepherded this paper and made many excellent suggestions not the least of which were adding the examples for each and pointed to many of the related organizational patterns. Most importantly, I'd like to thank Jesus Christ for being the author of the only philosophy/world-view that really matters, and many throughout my life that have encouraged me to keep my eyes on Him in all I do.

---

last updated 3-Aug-98

Ken Auer <kauer@rolemodelsoft.com>
RoleModel Software, Inc.
5004 Rossmore Dr.
Fuquay-Varina, NC 27526

(v) 919-557-6352
(f) 919-552-8166