Organizational Patterns For Software Architecture

Draft for Comment

by

- David Dikel, Applied Expertise, ddikel@aecorp.com
- Christy Hermansen, Eidea Labs, christy@eidea.com
- David Kane, Applied Expertise, dkane@aecorp.com
- Raphael Malveaux, Eidea Labs, raphael@eidea.com

Abstract: This pattern language describes software development in an organization in which the principles of software architecture are firmly in place. The patterns focus on organizational and management concerns and solutions, and are especially concerned with the informal practices used by managers to enable their organizations to be successful with software architecture.

The pattern language currently includes the patterns:

1. Informal Network
2. Rotation
3. Airing the Laundry
4. Risk Agent.

We have sketched out at least one future pattern, Strong Architect, and we expect to present others as this pattern language develops.

Introduction

This pattern language describes software development in an organization in which the principles of software architecture are firmly in place. The patterns focus on organizational and management concerns and solutions, and are directed towards the informal practices used by managers to enable their organizations to be successful with software architecture. This pattern language is not yet complete. We have fleshed out four of the patterns, and have identified the first of several other candidates for inclusion in the pattern language.

The initial pattern descriptions were derived from Applied Expertise's case studies, a set of benchmarking and consulting services that involved both commercial and government organizations. [DIKEL][WILSON] The patterns were revised based on the experience and knowledge of the authors. Contributions were also made by the DC Patterns Discussion Group. [DCPATTERNS]

Context

While a number of different settings were observed, the overall context for the pattern language contains the following characteristics.

- The organization has an established software product-line architecture.(1)
- The architecture supports an existing base of products, and the product-line is being extended to support new markets. As such the organization's size is fairly stable.(2)
- There are specific owners for each architecture component. This is an example of Coplien's ArtifactOwnership pattern. [COPLIEN] Components may be of different sizes, and there may be some level of hierarchical organization. For example, a team may have responsibility for a large component which is composed of smaller components, for which individual team members have responsibility.
- Each product-line has its own release schedule. Also, owners of shared components maintain regular release schedules.
- The software architecture is supported by strong configuration management. For example, one organization we studied could rebuild the software of any configuration of any product in their product-line.
- The organization has some minimal level of process and component documentation. This varied substantially among and within the organizations we observed. These patterns do not prescribe the level of detail appropriate for software architecture in a given organization, but assume that the organization has at least the minimal documentation appropriate for their context.

Forces

In addition to the context shared across the pattern language there are also shared forces.

Cloning and the pressure to develop new product features

The pressure to develop new product features quickly and the increased cost of maintaining code when code grows are forces that come to a head when organizations clone software. We believe that cloning is both a cause and a symptom of architecture breakdown. Cloning is a process in which an architectural component or group of components is duplicated and modified to provide an additional capability, and then maintained by a separate owner. While cloning offers a means to quickly respond to pressure to develop new features quickly, cloning often has far-reaching consequences. When a developing organization clones, it duplicates the code, and complicates product tracking and management. It also dramatically increased the maintenance burden for each product and reduced the possibility of reusing components that were maintained externally. This tension underlies many of the patterns in this language.

Building architectural capability to support future uses

There is pressure to build general capability in an architecture to support potential future uses. When this force proceeds unchecked, components that are too slow and too complex are built. These unwieldy components are bypassed by engineers who build their own point solutions. When the architecture's integrity is violated in such a manner, the software becomes more complex and expensive to maintain.

1. Informal Networks

Problem

How do owners recognize and avoid delivering inadequate solutions to their users, especially those resulting from decisions made early in the life-cycle?

Context and Forces

Quality, volume and timeliness of communication among component owners ("owners") and component users ("users") is low. As a result, users (who may also own other components) are either unaware of existing components at their disposal, or they misuse a component due to a lack of understanding of its intent or construction.

Often, users prefer to solve a problem independently either by cloning existing functionality or by writing an original solution, showing no interest in requesting modifications to existing components. Users who claim "It's easier just to do it myself" or "I have to do it myself to for it to be done right" are symptoms of a communication failure that could have started with either the component owner or the component user. The origin of the communication failure is likely no longer discernible or relevant.

For example, component users who are unable to get modifications to existing components may clone and modify those features to get the desired capability. When components are poorly documented and hard to understand, users tend to clone larger and larger chunks of code, or start from scratch. Both outcomes dramatically increases the amount of code to be maintained.

A system of components has an architect who is responsible for the product-line as a whole. (Future Strong Architect pattern) Owners tend to give priority to requests from their chain of command over those of other customers. They are also unlikely to have both the budget and motivation to document components for reuse or keep them up-to-date.

As a result of these communication barriers, components fit together poorly at integration time, and multiple solutions may exist for a single problem. Delivered software is often brittle.

In addition to the documented structure of the organization, an informal organization exists as well. (3) The documented organization includes reporting structures, role assignments, and the assignment of employees to groups. The informal network can include the friendships, social groups, and other unplanned interactions, and rarely shows up on any organization chart. [MORGAN]

Solution

Managers and practitioners should work together to use, strengthen and shape this informal network to bring about "cultural" support for partnering. Specifically, they should encourage, reward and support developers for their participation in and contribution to social networks; that is, they are responsive to their users, and provide solutions to solve the problems shared by multiple groups. Managers whose staff participate effectively in these networks should be similarly acknowledged and rewarded.

The absence of such encouragement can stifle the organization's productive use of architecture. In one such instance an organization had two projects for the same customer, using the same technologies, building similar products. The projects had been budgeted with the assumption that reuse would reduce their costs. After some initial delays the manager of one team instructed his group not to discuss the project with the other team because it would distract them from their project goals. Without such communication, the projects spent time and effort solving the same problems twice. The projects were both very late and over budget .

Rewards may be formal or informal. For example, a manager may acknowledge the contributions of an engineer towards solving a shared problem at a meeting or through a group email message.

Managers and practitioners should encourage the following behaviors among their staff and peers.

- Owners have a clear understanding of which users are depending upon them.
- Users have a clear understanding of which owners they depend upon.
- Owners are discouraged from introducing late surprises to users.
- Owners are evaluated by their managers and peers for their ability to negotiate feature requests and manage changes that affect multiple components.
- All developers are rewarded for their ability to use their informal networks to meet the needs of their customers.
- Owners are sanctioned for introducing changes that cause breakdown in other portions of the system.

There are several fundamental rules of the informal network:

- Common use
- Reciprocity
- No surprises

Common use means sharing a common component whenever possible, not just reusing an existing component, but working to increase a component owners ability to serve your needs as well as others now and in the future. Reciprocity means being ready to return the knowledge and insights from architects and component owners who have explained and/or adapted their structure or components. No surprises means alerting all stakeholders to needs and changes as soon as possible. This applies to both owners and users of components.

This pattern does not describe how these behaviors should be specifically introduced, since different approaches may be appropriate for different organizational cultures. However, there are examples of both managers and practitioners successfully using their informal network.

- Engineers were promoted based on their effectiveness in working with other groups with different reporting structures. Engineers who had been promoted all had stories of how the engineer had been able to use a common solution to solve problems for different groups.
- Practitioners can also take action to build their network. One architect makes it a point to help out other engineers, reviewing and commenting on their designs, and helping to solve problems. When the architect needs to call someone else for information or request a change, the phone is answered because of the architect's reciprocity.

Developers are rotated throughout the organization to increase the chances that when something goes wrong, one team member will at least "know someone" who is knowledgeable about a particular component which requires modifications. (see Rotation)

< FONT COLOR="#000000">In large-scale developments, the social network can complement more formal customer-supplier relationships. Large-scale developments often contain hundreds, even thousands of components. In these situations, it would be impossible for owners to keep track of and respond to all of their customers without formal support.

Formal customer-supplier relationships can take several forms. The approaches are complementary. We observe that both approaches work better when supported by effective informal networks. In both cases, the informal networks make asking questions of owners and informing users a less threatening and more rewarding experience.

- Owners can formalize their component's APIs by documenting the interface and "contracting" to support and maintain it. Owners are expected to notify users of proposed changes to APIs through established as well as informal network channels.
- Ownership can also be organized hierarchically. This organization can be implemented as a command and control or a technical hierarchy. In either case, owners of high-level components take on the primary obligation of understanding how their components interact with other high-level components and educating owners whose low level components comprise his or her high-level component. Low-level component owners take on the obligation of supporting peers who also contribute to the high-level component, as well as supporting the high-level component owner.

Benefits

Benefits include:

- Communication improves across the organization and over time. Customers and suppliers know more about each others' needs earlier. Even with expert individual component developers, integration can be painful without a good fit that has been validated via communication throughout the development process.
- Employee satisfaction is higher. An effective informal network increases an engineer's ability to solve problems and get recognition for his or her achievements. Because reciprocity is in place, engineers don't feel used. Conversely, engineers can expect to be reimbursed for the information they offer and adjustments they make.
- Cloning is minimized. Developers are strongly encouraged not to build from scratch and not to clone by social rules (as well as management rules). As a result, changes are localized to shared components, and managed by Owners.
- Because cloning and custom code are reduced, there is less code to maintain.
- Surprises are reduced. This means, the wrenching experience of finding out a week before delivery that your assumptions are wrong about a component you depend on are less likely to occur.
- Customer-supplier relationships are enhanced. It is much easier to talk with someone you know, than someone you don't -- particularly when the news is bad.
- Helps the organization make productive use of the cliques and informal networks that exist in all organizations.
- Learning about culture, code and technology is expedited. As component owners become more educated, their overall competence improves. Also, communication between component owners allows them not only to build their own confidence as their knowledge increases, but to gain confidence in the developers that support them.

Drawbacks

When management and technical staff see the power of informal networks, there is a strong danger of letting complementary formal mechanisms slide. Informal networks, by themselves, do not scale well. The rate of growth of the organization may be limited, because the informal networks take time to develop and expand. Also, the network may break down if the organization experiences substantial turnover, or reductions in staff. If the organization does not have strong configuration management practices in place, this pattern will not stop architectural deterioration. A period of adjustment is likely necessary for component owners priorities to shift from individual schedule requirements to development practices that ease component integration scheduling and problems. Support for this shift should be evident at all levels of an organization.

The pattern may be less effective if it is used across organizational or geographic boundaries; however, it has been observed crossing both, and may even mitigate some of the problems associated with managing across such boundaries.

Related Patterns

- Airing the Laundry
- Artifact/Code/Component ownership [COPLIEN]
- Rotation
- Unity of Purpose [HARRISON]
- Lock ’em Up Together [HARRISON]

2. Rotation

Problem

How does a manager reduce the number of people who if lost would prevent a project from succeeding, (the truck number 4), especially when the organization’s stability is suffering as increasing dissatisfaction of both key and non-key individuals is making developer turnover unpredictable and more likely?

Context and Forces

Detailed knowledge about components is hard to retain and disseminate. The component owner may feel roped-in by becoming the "expert" for a given component. If such employees are dissatisfied, the likelihood of losing them increases. When the only person who knows a component well leaves the organization, it is very expensive for the new component owner to reacquire that knowledge from scratch.

Other component owners may hoard information about the component to maintain their unique value. Since component owners often need to work with each other to get necessary revisions and additions from other components, key people are in a position to take advantage of their irreplaceable status. Organizations may make decisions to avoid offending key people, even if those decisions are not the best for the organization as a whole. For example, owners may try to maintain and expand their position and/or block the progress of anyone whom they feel threatened by, often at the expense of losing a large number of promising, but more replaceable engineers.

Solution

Periodically rotate component ownership by means of apprenticeship [COPLIEN]. It is critical that the organization and individual managers allow and encourage ex-owners to take time out of their schedules to support new owners. (See Informal Network pattern) To the extent possible, rotations should be synchronized with release schedules, so that new owners aren't thrown directly into the fire, and so that former owners will have time to act as mentors. The rotation should enable engineers to spiral, so that as people rotate they also move up into areas of more difficult technical expertise and/or greater responsibility. As a rule of thumb, rotate engineers about every 12 months, but this may vary due to the size and complexity of the components.

Benefits

Staff is more content because they get to work in a variety of areas, and turnover is reduced. [COUGER] The organization has more depth so if a component owner leaves the organization, there are other previous component owners who also have detailed knowledge of the component. This makes it easier and less expensive to replace the departed component owner. Since component owners will have worked in a variety of areas, and apprenticed with a number of different people, they will develop stronger social networks with other component owners.

Component owners will be better trained to assume positions of greater responsibility; as people rotate and move up in an organization, they will increase the portion of the system with which they have hands-on knowledge. They will then be more qualified to make decisions that affect larger portions of the system.

By rotating in spirals instead of flat rings, component owners who tend to resist change will have more incentive to move from their existing component to those with more challenge and responsibility. Component owners who are able to pick-up new components quickly, and will not feel "roped-in" by becoming the "expert" in a given area, because there would be a clear path to greater challenges and responsibilities.

Drawbacks

There is an overhead of training and retraining that results from this pattern. While the organization does not need to recreate detailed component knowledge from scratch, effort is required to routinely train new component owners as they are rotated to new assignments. There may be a schedule impact as well.

While component owners who seek to learn and grow will be more content and less likely to leave, some may feel threatened with starting all over again in a new area. As a result, they may leave or subvert the implementation of the pattern. Some other approach may be needed to accommodate their interests.

The organization may not be able to grow quickly because it takes time to apprentice new component owners. Similarly, if the organization is shrinking, for example, through staff cuts, then this pattern may break down.

There may be resentment generated from the addition of new people from outside the organization. For example, if new people are brought in and are given ownership responsibilities in the middle of the rotation spiral, there may be resentment from long-time organization members who started at the bottom of the spiral.

Related Patterns

- Apprenticeship [COPLIEN]
- Artifact/Code/Component ownership [COPLIEN]
- DevelopingInPairs [COPLIEN]
- Informal Network
- Diversity of Membership [HARRISON]

3. Airing The Laundry

Problem

When owners take shortcuts, how does a manager responsible for a product avoid possible resulting problems before they appear as surprises during integration?

Context and Forces

Component owners are reluctant to discuss problems and risks that impact schedule - they talk within their own team, but not with other teams. They fear retribution. Since owners are reluctant to discuss problems that affect schedule, these issues don’t show up until integration, and so component integration exceeds schedule projections. Component integrators spend long, tense hours during the last phase of a release cycle resolving late arising conflicts.

Component owners build and adapt components for their specific project group as well as for other external project groups whose code will be integrated into a final product. Under pressure from their managers to meet schedule to reach a milestone, owners and their managers may decide to cut out or modify features. They change assumptions that are not visible to other project groups. Side effects may include differences in performance, reliability or interaction with other components. Both owners and their managers feel discouraged from discussing these decisions with their peers.

Even small changes that may appear insignificant to owners may have substantial side effects for users outside of their project team. For example, their users may be working in different technical areas with different standards and protocols. The risk of this occurring is complicated by the lack of widely accepted standards that cross technical areas.

As a result, applications are fragile or inflexible due to poorly mapped component interfaces held together by supplemental code that was not part of their design. Translators between components with the same role begin to accumulate.

Solution

Product managers should:

- Ask owners or their managers to regularly (for example, weekly or monthly) surface any changes they have made, even those that do not appear significant.
- Ensure that the information is shared across all owners and users.
    - For example, in one organization, management established "no surprises" as the cardinal rule of meetings. When reporting problems at status meetings, engineers and other managers were expected to have had previously communicated the problems with others affected, regardless of where the affected groups were in the organization. The result of this approach

was extensive communication among the many groups

- Probe to make sure that when components are reused from outside the organization, that users have made sure they understood all assumptions and limitations associated with those components.
- Protect from retribution those who surface risks.
- Provide additional resources, and convey risks upward in the organization.
- Assign responsibility for identifying risks and assumptions of a decision to the individual who made the decision.
- Assign responsibility for exposing the underlying risks and assumptions of a component to its owner when he or she makes the component available for reuse.

Recording assumptions and risks should follow a consistent, but not inflexible, outline to minimize the amount of effort required to effectively convey and understand information. A consistent outline also allows staff to easily find analogies from previous development to better predict the effects of current decisions.

Benefits

This pattern has both long term and short term benefits; it increases an organization’s ability to manage risk.

- A larger pool of resources is able to contribute to solutions for resolving differences in assumptions and managing risks. More alternatives may then be considered so the organization can select the solution that is most consistent with overall organizational goals.
- Both the quality and the quantity of component reuse increase, during both design and integration. This effect becomes more pronounced with each iteration of development, because component mismatch and excess replication will be minimized during subsequent release cycles.
- Greater visibility of assumed risks and early identification of conflicts between assumptions minimizes surprise conflicts arising late in the development cycle.
- Conflict resolution guidelines reduce the potential for emotion to enter solution negotiations. Also, because designers and developers know in advance the criteria for resolving integration conflicts, they are more able to make design and implementation decisions that facilitate resolution or prevent conflicts.
- An organization will over time develop a risk management database and a mature tradition of accurate risk assessment as its members gather and record their experiences. The longevity of these overall benefits has the potential to exceed that of individual participants or programs.

Drawbacks

The organization may be lulled into a false sense of security if the process is expected to capture all risks and assumptions. A component owner may become dependent upon the system and fail to recognize risks the system does not anticipate. If a significant risk is missed, the organization could be blind-sided. Component owners may also fail to assume responsibility for conflicts that arise because the system did not take into account contributing factors.

Related Patterns

- Risk Agent

4. Risk Agent

Problem

How does the organization reduce the number, improve the speed, or even avoid elevating decisions to higher levels of the organization?

Context and Forces

Engineers and low-level managers are reluctant to negotiate decisions with peers who interface with their components, even though the best knowledge about the system or component architecture resides at this level. As a result, too many decisions are made by senior managers who are too busy to grasp details.

This pattern assumes there is a history of conflict between organization units in working on different parts of the same project. One or more leaders have in the past ignored or tried to subvert decisions made in different parts of the organization. Leaders hesitate to put time and resources into recommendations and decisions due to lack of confidence in other groups.

If the organization is very bureaucratic, engineers and first level managers may hesitate to make decisions because they are not confident the decisions will be enforced across the project.

Engineers who would like to be solving problems are frustrated when decisions get pushed to a higher level that is responsible for both or all differing groups. Their productivity decreases while they wait for higher level manager to be educated about the risks and assumptions associated with different solutions, and to then make a final decision. Higher level resources are consumed making engineering decisions, adding expense to the project.

Solution

This pattern requires a strong and talented "risk agent"; it is not for the faint-hearted. The agent must have the trust of the authoritarian regime and the ability to protect the engineers and managers who follow his or her advice. At the point of conflict, the agent should provide resources and support to resolve the source of the conflict and to shift the focus from conflict to problem solving. One of the key talents is to get engineers and managers to see what would have been perceived as their potential or actual screw-up as an engineering problem. Encourage partnering across organizational units to achieve common goals; and encourage low and mid-level managers to resolve conflicts without involving more senior management. Senior management should invest in developing these skills in lower level managers, whether through formal training or a more informal system of providing constructive feedback.

Educate engineers, and low- and mid-level managers about the project's business and architectural goals and priorities. Make sure that consistent information is disseminated across the project so that decisions by different groups are based on the same high level information.

Implementing the Informal Network pattern contributes to the success of this pattern. By developing and improving relationships among owners across organizational boundaries, they will be better able to resolve risks among themselves.

This pattern should be implemented with the pattern, Airing the Laundry. By keeping a record of considerations that affect decisions, managers develop confidence in their ability to defend their decisions and become less hesitant to make decisions. Also, managers are able to look back and evaluate what did and didn't work more objectively, improving their ability to make future decisions. This also allows a more objective exchange of information between groups, enabling confidence in the decisions of other groups to grow.

Benefits

Engineers and managers are empowered to make decisions and solve problems. They are able to make decisions that do not conflict with other organizations with which they do not share reporting relationships.

Delays caused by elevating decisions are minimized. Engineers are able to continue working sooner in response to more immediate decisions, reducing frustration also.

Fewer high level resources are consumed making engineering decisions. Low-level managers gain decision-making experience gradually, improving their ability to handle more responsibility and better preparing them for advancement.

Emotional responses to judgment errors are minimized, improving overall morale within a group and between different parts of a project.

Drawbacks

While this pattern can result in getting better decisions made faster, the solution will "run out of steam" if management does not implement a broad-based risk and opportunity management approach. Running out of steam can manifest itself in burn-out of the agent, and loss of confidence from those with whom the agent has gained trust. Because the pattern often produces impressive results for small "cost," it can be misused, even abused by managers. Misused, when the results mask a broader set of systemic risks; and abused, when a senior manager brings in one idealistic agent after another convincing each that "we are really going to change now." This pattern offers a first step towards resolving a complex set of forces; however, ultimate resolution may call for long-term solutions discussed in risk management and general management literature.

When decision making is delegated to low-level organizational units, the project can be affected more severely by inconsistencies in information reaching these units. If engineers and first level managers are making decisions based on inconsistent information, their decisions may expose the organization to risk. The importance of making assumptions and risks visible may not be stressed at a sufficient level, leading to conflicting decisions. (See Airing the Laundry) This may be related to other patterns that describe the role of architects, such as Coplien's ArchitectAlsoImplements. [COPLIEN]

Related Patterns

- Airing the Laundry
- Informal Network
- Strong Architect

5. Strong Architect (Under Development)

A strong architect has the trust and support of both developers and high-level management. This comes from his deep understanding of the product-line or lines, underlying technology, the organization, its people, its customers, and the business it is in. Unlike an architect with title and role, but no power, or an architect who is acknowledged for knowing the technology but who is ignored "when push comes to shove," the strong architect has "real power." This power may be granted explicitly or implicitly by the organization.

Concluding Remarks

All of the preceding patterns encourage a consistent system of development that strongly correlates to the organizational structure of a software project. While they emphasize informal procedures for enabling economical and effective development, they are meant to complement more formal policies and used with equal consistency. The authors and contributors consider these practices to be priorities key to the longevity and success of software development units.

References

[COPLIEN] The OrgPatterns web site, http://www. bell-labs.com/cgi-user/OrgPatterns/OrgPatterns, as viewed on May 18th, 1997.

[COUGER] D. Couger, "Motivating IS Personnel" Datamation, September 1988, reprinted in Software Management, 4th ed, Reifer, D., ed. IEEE Computer Society Press, Los Alamitos, CA, 1993.

[DCPATTERNS] The participants from the DC Patterns discussion group who participated in the discussion for this pattern language were: Frank Brauns, Rick Cunningham, David Dikel, Christy Hermansen, David Kane, Ajay Khater, Neil London, Raphael Malveau, Thad Sheer, Gary Smith, Theresa Smith, and Andy Wilkerson.

[DIKEL] D. Dikel, D. Kane, et al., Software Architecture Case Study: Organizational Success Factors; ARPA STARS, December, 1995.

[GARLAN] D. Garlan, D. Perry, "Introduction to the Special Issue on Software Architecture," IEEE Transactions on Engineering, April 1995, pp 269-274.

[HARRISON] N. Harrison, "Organizational Patterns for Teams" Pattern Languages if Program Design, Vlissides, J., Coplien, J., and Kerth, N., eds., Addison-Wesley, Reading, Massachusetts, 1996.

[MORGAN] G. Morgan, Images of Organization Sage Publications, Newbury Park, 1986, p41.

[WILSON] J. R. Wilson, D. Dikel, D. Kane, et.al., "Software Reuse Benchmarking Study: Learning from Industry and Government Leaders," Written for the Defense Information Systems Agency, by Applied Expertise under subcontract to EDS, Document DCA 100-93-D-0066, Delivery Order 0037, Nov. 10, 1995.

---

1 David Garlan and Dewayne Perry define software architecture as: "The structure of the components of

a program/system, their interrelationships, and principles and guidelines governing their design and evolution over time." [GARLAN] A software architecture that is shared across more than one system or product, or family of products is a software product-line architecture.

2 A different pattern language may be appropriate for organizations building new architectures, or for organizations experiencing substantial growth or shrinkage.

3 The notion of an informal organization was recognized by Elton Mayo in his Hawthorne studies in the 1920’s and 1930’s. [MORGAN]

4 "The ’truck number’ is the number of critical people on a project; if any one of them were to get hit by a truck, the project would lose essential knowledge and not be able to continue." [HARRISON]