The Agile Architect: Our Experience in Discovering a Successful Pattern

CHRIS EDWARDS, P.ENG., IHS INC.
SEAN DUNN, CD, P.ENG., IHS INC.

As our software development teams at IHS Inc. undertook an ambitious project to re-architect our desktop software platform, we struggled to reconcile the traditional role of Architect with Agile values. From our experience, we came to appreciate the vital role of leadership skills to a successful Architect. We began to view the responsibility of the Architect as that of a servant-leader, responsible for building self-organizing teams. We experimented with several variants to this approach. Ultimately our Architect role proved transient once the teams reached critical mass on technical skills and organizational clarity.

1. INTRODUCTION

The role of "Architect" is sometimes frowned upon in the Agile community; this role is often perceived as a central command-and-control authority who performs large up-front design, generates excessive documentation and bottlenecks decisions. We struggled with the question: is there truly a place for an Architect in an Agile organization? This paper chronicles our organization's trial-and-error approach in discovering how to achieve technical alignment across multiple teams while preserving team empowerment.

In pursuit of this balance between empowerment and alignment, we experimented with several different approaches to the Architect role. We discovered the vital need for clear technical direction early in the project, but that the leadership approach of the Architect had to evolve over time if we were to meet our goal of self-organizing teams. Eventually we learned that an "Architecture Coach" best achieved our objectives of maintaining technical alignment while promoting self-organization. The Architecture Coach was most effective as a servant-leader, responsible for developing technical skills in others and promoting cross-team alignment on design goals and values.

After approximately two years of experimenting with the Architect role, the responsibility of developing the architecture eventually shifted to the teams. We reached critical mass on technical skills and conceptual clarity, at which point the teams took ownership of evolving the architecture. The responsibilities of the traditional architect were now shared among all the teams and we were able to retire the role entirely.

2. BACKGROUND

This paper shares the experiences of a group of approximately 40 developers within a division of IHS Inc., a global company that supplies information and analytics to many industries including energy, aerospace and defense, chemicals, automotive and electronics. In early 2013, the company transitioned from a waterfall model towards Agile values and methodologies. At this time, developers were experienced in software engineering, but relatively new to Agile thinking. One of the first major changes was to re-organize into five Agile feature teams.

At approximately the same time, we began an ambitious project to convert a single-user desktop application into a multi-user enterprise solution, dubbed "Harmony Enterprise." This project required unprecedented technical coordination between the five Agile teams and necessitated a major architectural change in the software.

We did not have a resident enterprise architecture expert, nor a strong history of an Architect role in our department. We did, however, have a small number of individuals with detailed knowledge of the existing platform. In the months leading up to project kick-off, this small group performed preliminary investigative work to examine high level requirements and existing architectural limitations. At project kick-off, we did not define an explicit "Architect" role; our initial view was that an Architect was at odds with our new-found Agile mindset: "the best architectures, requirements, and designs emerge from self-organizing teams". (Beck, et al)

Chris Edwards, Suite 200, 1331 Macleod Trail SE, Calgary, AB, T2G 0K3; email: chris.edwards@ihs.com

Sean Dunn, Suite 200, 1331 Macleod Trail SE, Calgary, AB, T2G 0K3; email: sean.dunn@ihs.com

3. ARCHITECTURE APPROACH 1 - LET THE TEAM DECIDE

Our initial approach to architecture was that of absolute egalitarianism: the entire responsibility for architectural design was entrusted to the teams. Historically, when teams had been working on largely independent features, this approach had been very successful; team members collaboratively developed designs and capably delivered business value. However, as we embarked on the Harmony Enterprise project, we observed different outcomes:

- Progress was much slower than anticipated;
- Significant rework was needed as developers struggled to find an effective design approach; and
- Morale was low; developers were becoming frustrated and dreaded working on the project.

As we inspected closer we found that developers were struggling with how to get started. Having been recently re-organized from component-based to feature-based teams, many of the developers were not yet acquainted with the breadth of the existing codebase. Additionally, as our pedigree was in stand-alone desktop applications, we did not have experience with software design patterns applicable to server-backed systems. We came to understand that we had not prepared our teams for success; our developers were desperate for the knowledge and initial patterns so that they could get started with confidence.

Our intention had been to empower teams, but it appeared as if we had swung too far to the other end of the spectrum. This was a challenging project, and we considered whether this was an appropriate case for someone knowledgeable about the existing architecture to provide some technical direction.

4. ARCHITECTURE APPROACH 2 - THE ARCHITECTURE SCOUT

We hypothesized that the challenges we were experiencing with the egalitarian approach could be overcome if developers had a clearer picture of the work and had fewer decisions to make. We considered whether we could perform "just-in-time architecture", feeding teams architectural information as they needed it, while avoiding extensive documentation typically associated with upfront design.

In considering our approach to just-in-time architecture, we performed simple value-stream analysis to determine where architecture fits in. Each team was iterating through the entire value stream, shown in Figure 1, in short cycles. In each iteration through this value stream, teams needed a minimum amount of information on architectural patterns and consistency to confidently perform tactical software design.

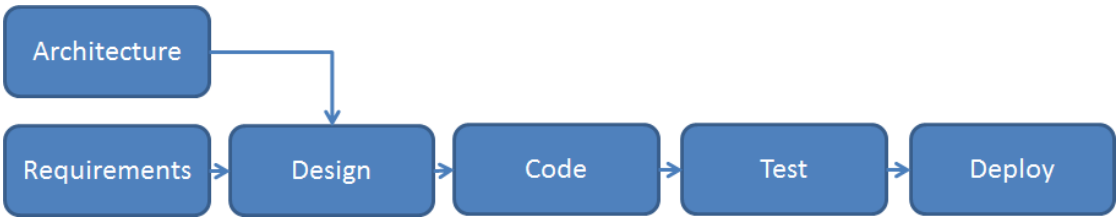

Figure 1. **Value Stream Mapping**

To accomplish just-in-time architecture, we invented the role of the "Architecture Scout". We chose an individual who had detailed knowledge of the existing system and had been involved in the preliminary investigations. The Scout looked about a month ahead in the product backlog for stories that would be novel or challenging, and was responsible for doing the legwork to research or prototype as needed. The Scout would then feed minimally sufficient architectural information to the teams as they were ready to begin new stories. To this end, the Scout performed activities such as:

- Research industry best practices and alternatives to inform upcoming architectural questions;
- Research specific topics to fill in knowledge and experience gaps, for example, nuances in database locking and concurrency;
- Define layers and their responsibilities;
- Create scaffolding for upcoming work, such as new code libraries for each layer, and interfaces for tying the layers together;

- Prototype examples to demonstrate the patterns being used;
- Provide training sessions to get developers up to speed; and
- Act as a conduit of information between teams by keeping apprised of each team's solutions and conveying to other teams who may be impacted.

As teams were ready to begin each new story, the Architecture Scout would hand-off prepared code scaffolding, as well as some prototypical patterns and samples for the team to consume. This was to provide just enough infrastructure so that developers had a foundation on which to quickly gain traction.

These responsibilities of the Architecture Scout were similar to those of the traditional role of architect, with two key differences: the Architecture Scout performed these activities continuously throughout development, and did so with a considerable amount of hands-on programming. Rather than producing detailed designs prior to starting, architecture was being developed incrementally with the goal of inspecting and adapting as we progressed. We were essentially performing "Architecture by Example".

In the weeks and months after creating the Architecture Scout role, we observed an improvement with both morale and productivity. With examples to work from and a better understanding of the problems to solve, the teams were able to make progress much more rapidly.

A few months after creating the Scout role, we were surprised to observe some new undesirable effects:

- **Communication Breakdowns**. Information would be misunderstood or missed. Developers wanted more and more information to be communicated, but the Architecture Scout became overwhelmed with the volume of information teams were requesting;
- **Inter-team Friction**. Friction formed between teams as they started consuming each other's work. Different teams had different design values, and had different expectations of the designs being produced. Teams would look to the Architecture Scout to resolve these differences by producing more rules and enforcing consistency;
- **Frustration over Changes.** A few months into the project, we identified challenges with our initial architectural approach. As teams began to perceive the Architecture Scout as more of an architectural authority, they felt frustrated when failures in initial designs led to significant changes and rework; and
- **Problems Go Undetected.** When developers would encounter awkwardness with an accepted technical approach, they did not consider that the accepted approach should be re-examined. Rather, they assumed there was good rationale (that they were unaware of) for why things were done a certain way. In reality, the approach was selected with certain assumptions that were being proven invalid once developers delved into the details. Without being involved in the decision-making process, developers could not detect when new information might invalidate a technical approach and should trigger re-evaluation.

These observations suggested teams did not feel ownership over their work, all symptomatic of an authoritative leadership approach, which we had been so desperate to avoid. This was surprising to us, as we had repeatedly communicated that the role of the Architecture Scout was simply that of an information provider and that teams owned their own designs.

In hindsight, it was obvious as to why the teams began to view the Scout as an authority figure: when developers encountered an unknown situation, they would come to the Architecture Scout seeking answers. With good intentions and eager to be helpful, the Scout would enthusiastically provide solutions. However, this behaviour clearly (albeit inadvertently) sent the message: the responsibility of designing the system belonged to the Architecture Scout.

From this experience, we became acutely aware of the leadership dimension of the architectural role; the behaviours, decisions and actions of the Architecture Scout had a direct impact on the behaviours, interactions, growth, and morale of the developers. Once we viewed the Architecture Scout through the lens of leadership, we understood that no matter how many times we verbally disclaimed the Scout as a decision authority, actions spoke louder than words.

5. ARCHITECTURE APPROACH 3 - THE ARCHITECTURE COACH

Our first approach was intended to empower the teams and avoid the pitfalls of autocratic technical direction, but failed to give people the information they needed to do their work effectively. Our second approach was

intended to provide this information without heavy upfront work, but unintentionally fell back into an authoritative leadership style. We discovered we needed an approach that provided all of the following:

- Help developers have a clear picture of how to solve their problems;
- Detect that we are wrong quickly; developers should feel encouraged to question the status quo;
- Detect and resolve misalignment between teams' understanding of the design approach and goals;
- Give developers a feeling of ownership over their designs;
- Encourage people to embrace failure as a learning opportunity; and
- Improve cross-team relationships.

Around this time we were exposed to a video from David Marquet about leadership. In this video, Marquet recounts his own journey in learning to give up control: "This idea of giving control, these are the two pillars that need to be in place: the technical competence …. and organization clarity. And you put those things in place and then you can give control." (Marquet)

Figure 2. **Pillars of Giving Control (Inno-Versity)**

This video had a profound impact on how we viewed leadership and helped diagnose the negative effects we were observing with the Architectural Scout role. With our newfound insight, we began to approach architecture from the perspective of transformational leadership (Burns): the Architect should be responsible for developing the technical competence and providing the organizational clarity so that control over the architecture can be divested. Following from Marquet's advice, we transitioned from an "Architecture Scout" to an "Architecture Coach". This had the following impacts on our approach:

- New design patterns were now investigated and prototyped by a team rather than the Architecture Coach;
- The Coach used each interaction with a team member to involve them in the decision making process, communicate the intention behind the architectural approach, develop their design skills, and discuss the impacts to high level design considerations such as maintainability, scalability, conceptual clarity, etc; and
- We instituted a daily half-hour design meeting. Attendance included a technical lead from each team and was facilitated by the Architecture Coach.

The format for a typical design meeting was as follows:

- A representative from one of the teams brings forward a design issue their team is currently experiencing or anticipates they will experience soon;
- The attendees discuss until there is collective clarity on the problem they are trying to solve;
- The group discusses various solutions that could be used to solve this problem;
- The Architecture Coach asks questions to guide the discussion, and suggests solutions they may not have thought of;
- The Coach encourages the team to discuss why they would choose one solution over another. The Coach asks provocative questions to get the group thinking about technical impacts they may not have considered; and
- The representative takes this information back to their team, which makes a decision on the approach to take.

In this meeting format, the Architecture Coach acted both as a facilitator and coach. Rather than make decisions, the responsibility of the Architecture Coach was to develop the collaborative decision-making skills of the group, while gently guiding them to appreciate the many engineering considerations and tradeoffs of those decisions. Over time, the various technical leads became more proficient in:

- Engaging in objective discussions with peers around technical decisions;
- Evaluating design alternatives and understanding technical trade-offs;

- Becoming confident in their own decisions; and
- Guiding others through the same collaborative, decision-making process.

This approach tended to more rapidly identify areas where we did not have organizational alignment on architectural values. In one case, the design group evaluated a problem in which the two main alternatives were either to place business logic in the database, or in the domain layer. The design group was overwhelmingly in favour of placing the logic in the domain layer, but when the representative brought the design back to their team, the team chose to put the logic in the database. We were interested to explore how the decision-making process of that team arrived at a vastly different decision than the design group.

Upon further investigation, we discovered that this team's design values were not aligned with the rest of the organization. A key piece of the intent behind the architecture had not been communicated effectively, and this was an opportunity for the whole team to get a deeper understanding of the project design goals. This was something that had gone undetected when the teams were not engaged in the problem solving process.

The Architecture Coach used the daily meetings to explore this disconnect further. Over the next two weeks, the group revisited the reasoning behind some of the key architectural decisions. Ultimately the existing approach was upheld, but this activity resulted in a team that was more aligned on the architecture, more confident in questioning the status quo and more skilled in navigating their disagreements.

Through each interaction, the Architecture Coach was able to help develop the design skills of the developers and reinforce the principles behind the architectural approach. As the teams researched new patterns and techniques, new expertise was grown, decentralizing the knowledge and decision making. After several months of the Architecture Coach facilitating the daily design meetings, we began to observe the following effects:

- Problems with design approach were now brought up more quickly by the teams;
- Teams were now very comfortable with cross-team collaboration;
- People grew in communication skills to identify and resolve inconsistencies in design values;
- The number of people with a deep understanding of the architecture approach increased; and
- Architectural changes were now discussed and agreed upon as a team.

6. FINAL ARCHITECTURE APPROACH - ARCHITECTING AS A TEAM

In time, teams began to arrange ad-hoc meetings to discuss particularly challenging or impactful design decisions. A culture of collaboration had formed among the teams, eliminating the Architecture Coach as a single conduit of information. In increasing frequency, the teams were collaborating and arriving upon solutions before the next daily design meeting. Eventually, we realized that the daily design discussions were serving only to keep the Architecture Coach apprised of decisions that had already been made by the teams.

The traditional responsibility of an Architect was now being shared among the technical leaders on each team, thus negating the need for any distinct architecture role. As such, we disbanded the role of the Architecture Coach and cancelled the daily design meetings. The responsibility of developing the architecture was now shared among all the teams.

We are careful to note that the approach we arrived at was different than "design by committee." In our model, there is always someone acting as a "Design Quarterback", responsible for investigating, gathering information, and facilitating a design discussion. The key here is that the skills to do this have been spread across many people, and the ownership of the system is felt by all teams.

7. LOOKING FORWARD

If in the future, if we were to encounter another project with equal or greater uncertainty, we would evaluate whether to start directly with the Architecture Coach, or first apply the Architecture Scout role and then transition towards Coach. The Architecture Scout solved a short-term need to get the project moving, but eventually had negative impacts on engagement. We did not reach any definitive conclusion whether the role of the Architecture Scout was a necessary evolution in a project's lifecycle, or simply an inferior approach in comparison to the Architecture Coach.

We hypothesize that a hybrid approach between the Scout and Coach would be more effective. An Architecture Coach may initially work closely with a single "pioneer" team to perform initial investigations and build early scaffolding. This would meet the short term needs of gaining traction. Over time, more people could be gradually on-boarded and control incrementally handed over. Initially the Architecture Coach would have

much more direct influence on the designs, but would be working closer to build conceptual integrity from the start.

Future investigation is warranted as to whether the Scout and Coach approach would be valid when applied to system architectures more complicated than a database-backed thick-client application. From our experience, it remains unknown how these roles may scale, for example, to a complex system of many diverse web-based services. This scenario may be further complicated if these diverse services are maintained by component, rather than feature-based teams.

8. CONCLUSION

Through our experience, we discovered that the Architect can play a pivotal role in Agile development, without sacrificing Agile principles. To accomplish this, the Architect must view themselves as a servant-leader, responsible for guiding, coaching and developing the teams to a self-organizing state. By developing the technical competence and organizational clarity around the design approach, the Architect can incrementally hand over control to those doing the work.

To this end, the Architect's approach may evolve over time; the Architecture Scout and Architecture Coach were two such variants in our approach. Rather than permanently institutionalizing these positions, we recognized when their utility had expired. This was not viewed as a failure, but rather as a success. We are cognizant that in the future, these roles may again be required, or perhaps a new, yet undiscovered alternative may be appropriate in the context at that time.

9. ACKNOWLEDGEMENTS

We would like to thank both IHS and our Senior Director, Chris Janostin, for providing us the trust and freedom to experiment and reach our own discoveries; through his support, we were able to view our mistakes as learning opportunities rather than as failures. Thank you to all the teams who endured our missteps as we learned along with you. Lastly, we would like to thank Michael Keeling for shepherding us through writing this paper. We couldn't have done it without you!

REFERENCES

Beck, Kent, et al "Manifesto for Agile Software Development", http://www.agilemanifesto.org
Marquet, David "Inno-Versity Presents: Greatness", https://www.youtube.com/watch?v=OqmdLcyES_Q
Inno-Versity "Inno-Versity Presents: Greatness", https://www.youtube.com/watch?v=OqmdLcyES_Q
Burns, James MacGregor "Leadership" Harper Collins, 1978