VINOD SANKARANARAYANAN

# SOFTWARE OWNERSHIP TRANSFER

EVOLVING KNOWLEDGE TRANSFER FOR THE AGILE WORLD

## Praise for *Software Ownership Transfer*

"There are many shifts occurring today as companies implement their digital business strategies. One aspect of this shift—companies are adjusting their approach to outsourcing in part by reinsourcing systems critical to their new technology-driven strategies. The problem is that there is little research or information available on transferring application systems knowledge and ownership from one organization to another—until now. In *Software Ownership Transfer,* Vinod draws on his experiences, large and small, in making such critical transfers. I particularly like his distinction between knowledge transfer and ownership—not ownership in the legal sense, but ownership in the sense of a team taking 'ownership' of their new responsibility. Knowledge transfer is only the first step in ownership transfer. If you are contemplating the transfer of an application system—from a vendor to in-house or from one internal location to another (as happens often today)—then you need to use this book as a model for making your transfer a success."

—*Jim Highsmith, author of* Agile Project Management,
*coauthor of* The Agile Manifesto

"Software is becoming increasingly central to most modern organizations. Indeed, it has become trite to observe that one should no longer have a 'digital strategy' or a 'technology strategy': These strategies now form the core of the business strategy itself. This shift is seeing many organizations reconsider their software-sourcing strategy, such as, for example, bringing back in-house resources that they previously outsourced. The process of transferring the 'ownership' of core digital products is complex and can frequently be the source of disappointment, to say the least. In the worst case, significant investments can be lost in the transfer process. In this book, Vinod Sankaranarayanan draws upon his significant experience in digital product ownership and transfer, backed by copious real-life examples, and draws some important recommendations for any leader or organization overseeing a transition between teams, whether internal or external. A highly recommended read for any leader who is contemplating or is in the process of a digital product transfer."

—*Chris Murphy, group managing director, Europe,*
*Middle East, and South Asia, ThoughtWorks*

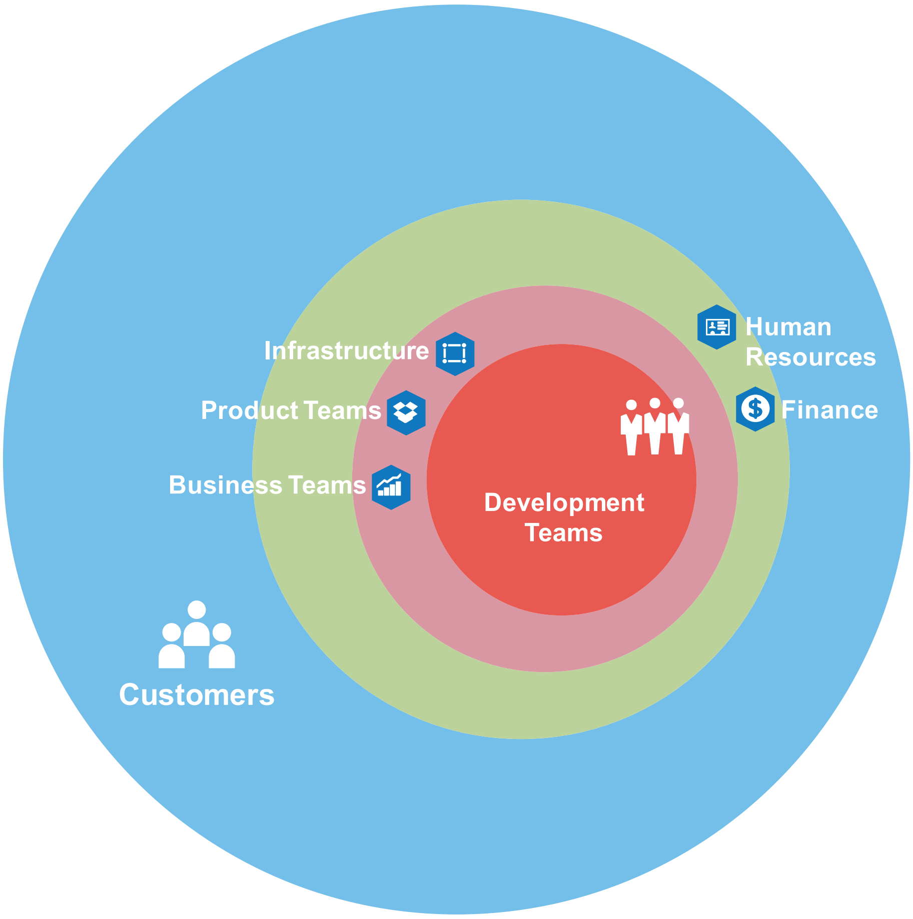

**Figure 4.1** *Orbits of influence*

teams would be considered the nucleus. The closest orbit would be the IS and support teams. The development teams interact with the IS teams on a daily basis, whether it be for releases, production support, or even deployment-related challenges.

The next group contains all the business teams with which the team directly liaises for their project delivery. These could range from project managers to product owners. Depending on how an organization is structured, product owners could very well be in the center of the orbit as well.

Enabling teams may be considered key stakeholders, too. They are the members of the organization who provide support: human resources, finance, accounting, and administration. Ownership transfer involves people and responsibilities. Needless to say, some members are going to be personally impacted at the end of this project. It's important to keep morale high through the entire process. This is where the human resources department will need to play a major role.

Finance and accounting is impacted in two ways. Ownership transfer itself involves a cost. This needs to be accounted for and budgeted. The other angle is the cost of projects during the transfer and post-transfer. Many things will change during the program, starting with the platform, processes, and the people working on the application. It is to be expected that the cost of projects will also change. A reset of baselines will be required to define the new status.

## Cost of the Program

We had initially identified the cost of this EuroT transformation at a million euros. Now, how does one estimate the cost of a transformation program? After all, this is unlike any other project. There is no historic precedent to refer back to. Using industry standards like function points or even story points (as defined for functional streams) will not work. One simply does not have a basis to assess the effort. Our biggest achievement was that we accepted this point almost immediately. We also accepted that any estimate may not be accurate. In this case, we broke down the program into what we termed KT (knowledge transfer) units. This was based on the current lay of the system.

But we also realized that there may be specific units focusing around process and tasks, activities such as production support, release activities, or functional streams that are relevant for the business analysts. We had automated the entire system. At its peak, we had about 6,000 automated tests running every day on the EuroT program. We optimized the tests over time to about 1000 tests running every day. Transfer would also require us to hand over quality assurance for the components.

Our estimate essentially relied upon asking senior developers and architects on how much time it would take to transfer over discrete KT units. We had to rely on the instincts of the Bangalore developers because the London teams opined that they had no way of knowing what they don't know. This is a matter of great sensitivity. The owner has to rely on the incumbent for how much effort she needs to put in. Trust and maturity are paramount in this exercise. The developers estimated in time. A unit would take *x* number of weeks or *y* number of months. This is in stark contrast to how regular transfers occur. The new team, a team that has not worked on the application at all, is asked for estimates. Whichever way a number is thrown, the client (or the management in cases of internal transfers) will negotiate on those numbers. It's a classic example of contract negotiation over customer collaboration. It's another matter entirely that both parties have little understanding of the complexity involved!

During the entire process, the running assumption was that someone from Bangalore would pair with someone from London. We had decided on remote pairing based on logistical as well as cost reasons. Processing visas as well as having the right set of individuals willing to travel is challenging. Some of these elements must be identified in advance. Remote pairing brings in its own level of inefficiencies. These need to be identified ahead of time, giving an opportunity for the developers to factor them in when estimating. As with all Agile concepts, we need to get individuals to sign up for such exercises. An ownership transfer exercise calls for team members with a high level of experience and maturity.

The next important factor was that pairing would not occur only to teach. Instead, pairing occurred to deliver functional projects. By functional projects, I mean projects whose purpose was to either increase revenues or bring down costs for the business. As the incumbent team, we were clear that learning happens best by doing, and not just by teaching. We created pairs to work on functional stories and ensured that these stories covered areas primarily around those that accounted for the KT unit's transfer. The question then became how to factor for costs. Our senior developers gave a sense of the "cost of transfer" while pairing on each story, not in monetary terms, but in the loss of productivity when playing the functional story. This loss of productivity was included in the transfer program budget.

Getting buy-in on this concept itself is a herculean task. This is when the stakeholders we talked to earlier come into play, especially the ones in finance. The managers whose projects become "guinea pigs" for the ownership transfer exercise become worried that work on the project may suffer. Stakeholder management is key during this transition. Because commercially significant projects are to be delivered, we need to keep the business groups in the loop. The traditional concept of project pricing takes a deviation when doing this transition. Part of the project cost goes to the business group sponsoring it, whereas another chunk goes towards the transition budget. It's easy for stakeholders to get into protracted discussions regarding costs. This is wasted energy. The best way to move forward is for senior management to keep all stakeholders informed at every major milestone and, more importantly, to convey what is expected in the ensuing months and how each group needs to participate in the transition.

## The Scope

In the case of a functional program, it's easy to define scope. You know what functionality has to be realized and can work through the business value of a certain feature. It's not very different with ownership transfer. Before you start, a large project should be broken down into smaller manageable chunks. This is neither easy nor straightforward. The approach to dividing up the work on the application needs to suit the objective. We did not take the approach of breaking the code based on functionality. Working code is paramount. We broke down the code into discrete elements, keeping in mind how the system architecture has been defined. This step determines the entire direction of the ownership transfer. It has consequences on how teams work with the transferred code. Our end state of ownership had different teams owning components, but not functionality. Hence, breaking the code into components for transfer made sense. A different program may have an end state derived out of functionality. In that case, classification of the units could be different. Apart from the actual code, we also identified elements around processes and roles (release process for example) that needed to be transferred. Overall, we identified seventeen units to transfer.

## Timeline

In most projects, timelines are determined by external factors. Perhaps there is a huge convention coming up that you need to be ready for. Maybe a regulatory change comes with mandated timelines, or you need to catch the sales cycle for a certain year. But for a program such as ownership transfer, you don't have an external factor determining the timeline. However, it's good to look at the timing of the transfer. It's useful to ensure that the transfer goes through key seasons during the process. This provides the new team an opportunity to experience the stress periods with the safety net of the incumbent's expertise. The timeline should also allow a few releases into production. Releases are always stressful periods and, in a sense, will reveal the maturity of the teams.

Most travel companies will see ticket sales peaking through the year-end holiday cycle. In the case of EuroT, we commenced the transfer activities in the early part of the year. Both teams witnessed five releases between them. By the time we reached the year-end, a good amount of the transfer was complete. However, we did have a small Bangalore presence during the year-end as well and both teams finished off the peak season together. The EuroT program also had intermediate milestones. Each of the KT units had to be transferred over in a given time. After they were transferred over, the London team took responsibility for handling those units. In parallel, some members from Bangalore left the project team, but only after one release with the relevant features went into production.

## Program Structure and Governance

Being Agile is never an excuse for being unstructured. All Agile programs require a structure, a conscious mapping of stakeholders, and a governance mechanism to steer the program. Although we take great pride in saying the teams ought to do what they think is best and the scrum master (or iteration manager, or project manager, as the case may be) must remove impediments, we do not think enough about how the scrum master is enabled. If the scrum master is not empowered to talk to the right set of people and influence decisions, the project is doomed. That is where an appropriate structure and steering group is required. *Agile IT Organization Design* by Sriram Narayan provides a detailed illustration of how to set up organizations to embrace the Agile mindset.[1]

The question any CIO will need to answer is, "Who will run this program?" Because code is the essence of what is being transferred, the obvious choice would be someone within the development team. The problem with that is everyone in the development team also owns some element of the KT unit. Also, someone deeply embedded within the nucleus can be blinded by the details. The chances of missing the larger picture increases. In the case of the EuroT transition, we identified a program manager who was not intrinsically part of the development team, but was part of the business solutions team. She had experience working with the development teams and had the title of product owner for many critical features within the system, so she had a deep understanding of the application. She had also worked with most teams and interacted with many of the team members in her previous role. Hence we found that happy medium of having a level of familiarity, but not so close as to be blinded to the larger program.

## Risks

Risk is often defined as all things uncertain. In the early stages of the program, we held a separate meeting with the EuroT CIO and discussed what could possibly go wrong. In that exercise, our intention was not to get through all of the risks and identify how to mitigate them. The EuroT CIO ran this meeting as a workshop with the sole purpose of identifying as many risks as possible. The consequences of these risks could be dealt with at a later time, if the need arose. As can be expected, many risks were identified. We came up with as many as 60 different areas of risk that touched upon all facets of the transformation.

We categorized these into seven areas. I am certain these risks will resonate with most transition exercises. Nomenclatures may change, but the themes should be consistent.

1. www.agileorgdesign.com/p/table-of-contents-preface.html

## An Imperfect World

This was our way of stating that we are not handing over a perfect product. There will always be issues that will not be to the satisfaction of the new team. No application has zero faults. These faults often come to the forefront when a new team takes over. Differences of opinion arise on the reason for these faults. It is a challenge to identify the context on things that were not done right, but were done for a reason.

In the same vein, it is equally difficult for the incumbent to remember all of the minute technical details in the system and pass it on to the new team. In one of our earlier engagements, we had encountered several instances where the incumbent team was scared to modify an existing code because they did not know enough about it. This becomes all the more frustrating when addressing production issues.

**An Imperfect System**

Atozhotels.com is run by a hotel chain in the United States. The Web site provides revenues to the company of more than 5 billion dollars. Multiple hotel brands are run from the same platform. The IT team of this company, based in the US, used to put out releases every two weeks. Because they had a dual-site approach, part of their release strategy involved switching traffic to one of the data centers while upgrading the application version on another site. After testing the first site and ensuring everything was fine, they would redirect traffic to the first site and then upgrade the second site. The IT group traditionally had not followed a platform deploy for their application. Their releases essentially were glorified patch updates.

After several years, this team was required to hand the application over to their offshore center in Ukraine. There were several snippets of code and modules that confused the Ukraine team. They did not see a reason for the existence of those modules. The US team agreed with their Ukraine counterparts, but neither party had the confidence to delete the code. They just did not know what would break if the code gets deleted. In the maddening rush of releases every other week, the US team never had the chance to sit back and review these pieces of code. More than 50 percent of the code seemed unnecessary, but had to stay in the system because no one felt confident enough to make any changes.

The team taking over needs to have high levels of maturity to handle an imperfect world. More on this in Chapter 6 in the section, "'Not Invented Here' Syndrome."