The Japanese Tea Garden: Lessons for Long-Running Development Projects

Phillip Johnston

INTRODUCTION

Since September 2017, I have been volunteering at the San Francisco Japanese Tea Garden in Golden Gate Park (JTGSF 2020). I work with the head gardener, seasoned bonsai devotees, and a master gardener to prune trees, mend fences, beat back bamboo, pull weeds, catch koi fish, and clean ponds. These tasks may not seem relevant to engineering teams, but my time in the Japanese Tea Garden has allowed me to observe how the garden is managed, maintained, and expanded. I am a business owner and engineer first, so I am continually on the hunt for competitive advantages and valuable lessons.

There is more momentum and legacy wrapped up in the Japanese Tea Garden than even the most storied and long-running software projects. The San Francisco Japanese Tea Garden was constructed in 1894 for the Midwinter Fair. It has been a continual fixture of Golden Gate Park, remaining in operation for 125 years and counting (Brown 1998). Visitors feel a visceral sense of history and time as they walk through the garden, touch the trees, and hear the stories.

The Japanese Tea Garden is a continually operating business. It does not close for maintenance or holidays. It is not a trivial business either; as of 2019, the Japanese Tea Garden receives around 600,000 visitors yearly. Work must happen while the garden is open, which sets constraints on execution. These constraints have led the garden to adopt a working and planning style similar to the agile and extreme programming movements.

I believe programmers and engineers benefit greatly from learning how other businesses and industries manage their problems. The software field is young,

Image: ClimbOn via iStock.com

but inevitably we will maintain and improve software infrastructure that is a century old. There is no nobility in struggling and recreating the lessons learned by other industries and by other people in other times. What I've learned from the Japanese Tea Garden can help software professionals manage their own soon-to-be-long-running projects.

Beyond the Standard Gardening Analogy for Software

One of my favorite analogies for software development is that it is akin to gardening. I cannot deny that it is my favorite analogy because of my love for gardening, plants, and the time I've spent observing nature.

Software projects have always felt similar to my gardening experiences. If one does not continually prune, weed, and maintain a garden, complexity and entropy will take over and become overwhelming. Incremental and continual efforts to abate the chaos and complexity of the garden (or software system) is more efficient than the alternative: ignoring the garden (or software system) until it is weed-ridden and unusable and then finally deciding to clean it up in one singular, exhausting effort. Continual extraction of weeds while they are young is much more efficient than mass removal of well-established and deeply rooted plants. Likewise, it is more efficient to fix software issues as they arise rather than defer them until the organization is ready to pay for the corrections in a mass effort.

This is not to say gardening is the only viable analogy for software development. Software development can simultaneously be like gardening (I say as a gardener), like engineering (I say as an electrical engineer), and like construction (I say as the son of a home builder).

In this article, I am moving beyond the typical form of "software is like gardening" or "software is like construction" analogies. Here, I consider that "maintaining a software product after launch (for years and years) is like maintaining and improving the San Francisco Japanese Tea Garden."

A MENTAL MODEL OF THE JAPANESE TEA GARDEN PROCESS

Before diving into the lessons learned, I want to spend some time explaining how the operation of the Japanese Tea Garden is sufficiently complex and is useful for learning lessons relevant to engineering and software projects.

The Japanese Tea Garden is alive and continually evolving. This is true at a basic level, because it is composed of plants and fish, which have their own lives, behaviors, and tendencies. But it is also a system that is shaped, maintained, and enjoyed by humans, each with his or her own ideals, behaviors, and tendencies. It's a business with an income, and, accordingly, the Golden Gate Park management and the customers have their own expectations for the garden.

The Japanese Tea Garden has a legacy and a context that must be kept in mind. Japanese gardens in general have a certain aesthetic, and the San Francisco Japanese Tea Garden in particular has its own aesthetics and history. These aesthetics constrain the choices gardeners can make.

The garden is old. When people work in the garden, they are never starting from a blank slate. Something with a history already exists in this space. Every plant has a story attached to it. The garden has evolved over time. Entire sections have been added, removed, or changed at different times by different individuals with different visions. Dozens of gardeners have taken part in its care and development, each with a different understanding of the garden and how to care for it. These decisions continue to ripple through the garden's design.

Currently, the head gardener's ideal reigns: he or she must improve the garden while honoring and preserving the essence of what it is. In the future, one can expect the ideal to shift in a new direction. But even then, knowingly or unknowingly, the future maintainers will still be bound by the momentum of the garden's history. There will still be the inevitable complaints about the poor decisions of the people who came before them, just as there are today.

Because of its continual operation, customer expectations, and the existing structure, the gardeners cannot make dramatic changes to the garden. Some work can be done "behind the scenes" (before people arrive), such as watering; however, most of the work is performed while people are visiting the garden. Complex tasks are split into smaller chunks to reduce the impact on the customer's experience.

Some of the work must be done, even if it will disrupt the customer experience. For example, cleaning the ponds disappoints people, as they are a significant part of the garden's makeup. However, not maintaining the ponds is disastrous: algae builds up, the water turns green and nasty, and the beautiful koi fish die. Such work is structured to minimize the impact on the garden experience, so even if a pond is being cleaned, 90 percent of the garden is still there to enjoy.

Most of the day-to-day work in the garden is analogous to software development. New features (trees, plants, and rocks) are added. Old features (topiary-style pruning and cluttered scenes) are removed when they no longer serve a purpose. There are "bugs," such as pipe leaks, pond leaks, rotting fence posts, dying plants, rodents, and the ceaseless bamboo takeover attempts.

Some changes happen quickly, such as pruning a tree. Other changes happen slowly, such as changing the overall

shape and structure of a tree. Some changes in the garden are made on such a long time frame that the gardener might not be alive to observe their final development.

The Japanese Tea Garden deals with standard project management issues. Necessary work cannot begin because there is no budget or leadership interest. This can be seen in the current state of disrepair of the pagoda, which has been unmaintained for decades and is now visibly rotting. The head gardener has finally created political will and a budget to repair the pagoda. Even though a budget is approved and the contractors are ready to begin work, city bureaucracy is adding unanticipated delays.

As always, there are never enough people to support the amount of work that needs to be done, both within the Japanese Tea Garden and within Golden Gate Park. Last-minute fire drills happen when Golden Gate Park managers suddenly demand help on other park projects. These fire drills cause schedule delays for previously planned garden work.

Because of these factors, work scheduling within the garden resembles an agile or extreme programming approach. The park has to stay open and not disappoint customers, and the solution is the continual improvement of the garden in daily cycles. The team kicks off their day with a meeting that resembles a standup: the team discusses the work to be completed that day, including any unplanned changes to the schedule or scheduled help from apprentice gardeners. There is a focus on individual tasks that can be performed entirely within one working day, from start to finish, including tidying up. Regular chores have a checklist. Work is divided up across the team, with different gardeners focusing on different areas of the garden.

META-LESSONS LEARNED

The Japanese Tea Garden is a complex system with continual pressure to add new features, maintain the existing feeling and structure, and produce revenue for the city of San Francisco. The garden is continually being used by customers, and it is continually being maintained and improved by the staff.

Software developers are in a similar situation when they are building and maintaining software and electronic products. Many have to deal with legacy systems that were created by other people. Often, the developers dislike these designs or the code. They obsess over improving the system or throwing it out completely and starting over. They find themselves constrained in the same way as the gardeners at the Japanese Tea Garden. They are understaffed, underfunded, and have paying customers to support. There are valuable lessons to draw from the art of improving and maintaining an actively used garden over such a long time frame.

The following project management meta-lessons I have learned provide valuable insights into maintaining legacy systems, how software developers and engineers should think about their projects, and the way they plan their work. In the next section, these lessons are described in the context of lessons that developers can apply directly.

1. Know the history.
2. Have a clear vision but make changes incrementally over time.
3. Maintenance dominates.
4. You can't make everything a feature.
5. The best improvements are imperceptible to the end users.

Know the History

One of the most important lessons I've learned from working at the Japanese Tea Garden is the importance of knowing the history and context for the work. People cannot make effective changes within a system if they don't know where they have come from and where the ship was previously headed. They cannot make effective changes if they do not understand the goals and ideals of the framework in which they are operating.

The head gardener at the Japanese Tea Garden models this daily through his research into the garden's history and by studying Japanese gardens around the world. After the internment of Japanese Americans during WWII, many of the gardeners responsible for maintaining the Japanese Tea Garden knew nothing about Japanese gardens or Japanese gardening techniques. British topiary forms crept into the garden, plants were roughly trimmed with power tools, scenes were crowded with new plants, and the style began to shift. By understanding this historical context and improving his knowledge of Japanese gardening, he has slowly corrected course and reversed many of these changes, once again regaining the former Japanese garden aesthetic.

How many software developers and engineers spend time understanding the history of their projects? They often just complain about how terrible the code is and how the previous developers hacked together a shantytown instead of taking the responsible path. Rarely is there an effort to understand the historical context behind the system and the decisions that lead to the current moment. They rarely consider how many previous developers were involved and the impact the circumstances had on their work. Rarely are they patient enough to work within the existing context to make improvements over a long time frame.

If developers and engineers don't know where they came from, how can they decide where to head next? How do they know they won't recreate the same

situations and pains that were already experienced in the past? How do they know they aren't contributing to the eventual downfall of the system? By studying the context and history of the system, the developers and engineers are better armed with information that can improve the project.

Have a Clear Vision, but Make Changes Incrementally Over Time

One difficulty with maintaining and improving a garden is the different time scales that must be considered. A gardener might want to make drastic changes to a tree, but it will take 10 to 20 years of new growth until the gardener's vision can be fulfilled. Gardeners must learn to hold a clear vision in mind for the end goal. They must become comfortable working toward that vision with small, incremental changes made continually over a long time frame. They are making massive transformations over months and years, although in the present moment it looks and feels like they are making no progress.

I have never been a part of a software project that took this slow, incremental approach to improvement, even on supposedly agile teams. For most of the industry, speed is king. Software developers focus on how quickly they can make changes and build a new system. Another industry tendency is to throw out a frustrating or poorly designed system and build a new one from scratch. This new system will "fix all the problems that the old system exhibits." Inevitably, this second system is no better than the first system and exhibits many of the same problems. These efforts also take much longer than the team originally envisioned, often dragging on years after their original release schedules.

Continual, incremental changes are less of an imposition on an organization's customers on internal teams. A complete teardown and rebuild not only causes a delay, but it requires everyone to relearn these systems from the ground up. Small, incremental changes, on the other hand, can sometimes go unnoticed by customers altogether.

Next time, consider an alternate scenario before considering a massive rewrite: what if the software developers just steadily worked to improve the existing code base over the same time frame? Identify the problem areas or qualities that need improvement, define and document a vision for the final outcome, and create quantifiable goals that can be used to evaluate the progress. Then, figure out how to slowly morph the code piece by piece, one day at a time. While the immediate payoff is low, the improvements will quickly compound over a time frame of months and years.

Rarely does one find that a drastic sweeping change is useful. Sometimes these changes are called for, but it's much rarer than one might think.

Maintenance Dominates

After more than two and a half years at the Japanese Tea Garden, I have seen two new trees and a few stones added to the garden. Most of the work we perform is maintenance, moving plants, or removing plants completely.

Engineers and software developers often have rose-colored glasses when taking on a new project. They always think about the joy of creation and building something new. Rarely do they look to the future, where they must maintain and support their project for years and years.

Engineers commonly look at this "maintenance" work with disdain. New employees handle the maintenance tasks, while the experienced engineers move on to a new project.

The reality of building things is that on a long enough time frame, maintenance work dominates, not novel creation. Software developers must actively maintain their projects, or the increasing complexity of the world will render their projects obsolete (Johnston 2019). This maintenance obligation never ends. Inevitably, many projects die because their maintainers become tired of the work.

Remember, too, that adding more features (or plants to a garden) means even more work to maintain the program/garden. And adding too many things too quickly might become overwhelming due to the sheer volume of unanticipated maintenance work.

Frequently, I see gardeners make decisions because it will reduce maintenance requirements while still preserving the garden aesthetic. I rarely see these decisions made on software projects; instead, the focus remains entirely on adding to the software with no regard to future maintenance requirements.

Here's a thought exercise: what work or decisions would reduce the time developers spend on the maintenance efforts for their projects? These improvements pay dividends in time savings and free up time to work on something new.

You Can't Make Everything a Feature

When gardeners prune a tree, their focus anchors solely to the tree they are working on. They discuss changes within that focused context. Eventually they will step back and realize that the tree is just one component of a much larger scene. Often, they may see that they spent too much time debating about detailed changes for a supporting character.

The gardeners at the Japanese Team Garden commonly repeat the mantra of "not everything can be a feature" while they are working. This is an attempt to force their minds into maintaining the larger context. A scene that is exclusively composed of features lacks cohesiveness. Nothing stands out. Supporting plants serve important roles, such as framing the scene, providing color, or drawing the eye toward the primary feature

because of the "flow" of the different shapes. Ideally, the gardeners want to work rapidly on the supporting plants so they can focus their detailed efforts on the features.

There is a tendency for feature creep on software projects too. Marketing teams emphasize the new features that have been added in the latest release. A product that starts out with a focused set of features eventually grows into an unwieldy mess. Users might wait years for fixes and improvements to the core functionality, while the development team continues to churn out new features for those marketing bullet points. Rarely are the new features the reason people wanted to use the product in the first place.

Step back and remember: not everything should be a feature. Keep the whole scene in mind and focus the efforts on what really matters.

The Best Improvements Are Imperceptible to the End Users

It is often the case within the Japanese Tea Garden that those improvements that are considered the "best" are ultimately indiscernible to customers outside of producing a general good feeling. The gardeners might prune a tree or remove a plant, but they are the only ones who really notice the details of the change. The composition of the scene remains largely the same, but it just "feels better" and "works better" than it did before.

This differs from many software products, where teams are constantly changing software designs, interfaces, and behaviors. Consider how users feel when Twitter and Facebook change their designs. Many users are angry and frustrated in these situations and are not left with a "general good feeling."

In my experience, internal team desires are the driving force behind new designs, not customer requests. As a customer, I am looking for fixes to the existing problems. Instead, companies regularly deliver a new design or new features but continue to ignore major flaws and bugs for years on end. I feel a pervasive feeling of frustration in these situations. I bet that is not what the team thinks their customers are experiencing.

Small, incremental changes feel better, because these changes are less of an imposition to customers. They are not being asked to relearn the system over and over again. This is true for gardens, software, and many other aspects of human life. Tear downs, rewrites, and redesigns might bring us enjoyment and internal glory. But the small, incremental, unnoticed changes bring customers more joy than those massive changes.

PRACTICAL LESSONS LEARNED

I would be remiss to leave out the actionable lessons that can be used in day-to-day software development and engineering work. Here are five lessons I have taken from the Japanese Tea Garden:

1. When refactoring, begin with significant changes.
2. When you don't know what else to do, start by tidying.
3. The job isn't done until you've cleaned up.
4. Many important jobs are tedious.
5. End the day with something usable.

When Refactoring, Begin with Significant Changes

When pruning a tree, the general principle is to "remove coarse-to-fine." Gardeners want to identify and make large changes before they focus on smaller details such as thinning out the foliage.

This is the lesson that has been hardest for me to internalize. There have been countless occasions where I have spent 30 minutes pruning a branch, only to have the head gardener come by with a saw and completely remove the branch.

It is not good to waste effort in this way. Throwing away work never feels good, especially when it could have been avoided it altogether.

It is important for developers and engineers to take a moment and elevate their perspective when beginning a new task. They need to see the entire scene and understand it before identifying the biggest changes to make or components to build. They should make these changes before working on any of the smaller tasks or refinements. It is important to not waste time and energy by making small changes that could be eliminated with one a single stroke.

When You Don't Know What Else to Do, Start by Tidying

Sometimes I am unsure of how to approach pruning a tree. I know, intellectually, that it needs to be "thinned out" so light hits the interior and the trunk, hidden branches become visible, and the tree itself feels lighter. But when I look at the tree, or even zoom into individual parts, I don't see an obvious path that achieves those goals.

In these moments, I fall back on a simple activity: tidy up the tree. I remove dead leaves, dead or dying branches, and detritus that has fallen from taller trees. This act serves two purposes:

1. Even if I do nothing else to the tree, I've improved the scene by cleaning it up.
2. As I work on cleaning up the tree, I learn its structure and see the interesting and unnecessary parts.

The same approach will work for software developers whenever they feel stuck on a programming or engineering problem. If they are unsure of where to begin, they can start by tidying the code, design, or documentation.

They can then fix the formatting, add comments, remove unnecessary statements or components. As they work, they will notice other ways to make improvements.

The Job Isn't Done Until You've Cleaned Up

When making estimates or preparing a weekly plan, most people budget their time based on how they execute the task itself. For example, when planning trees to work on, gardeners are tempted to only budget the time it takes to prune the tree.

Working in the garden emphasizes a stark fact: the job of pruning a tree is more than just the act of pruning. The job isn't done until clippings are removed from the ground, the paths are swept to look neat, trampled grass is raked and made to look untouched, the clippings are taken to the compost pile, the tools are cleaned, and everything is put away. Sometimes cleaning up can take longer than the primary task.

I observe that developers (including myself) often estimate the time to write the code or solve the problem. They forget about the time they will spend testing and debugging their design, cleaning up the code for publication, reorganizing the branches and commit messages, working through code reviews, addressing feedback, merging into the master branch, and working with QA to confirm the problem is fixed.

Developers need to keep this in mind: the job isn't done until the primary work is completed and everything has been cleaned up.

Many Important Jobs Are Tedious

Some important jobs are fun, like pruning trees or catching koi fish. However, many important jobs are tedious. They still must be done.

Gardeners sometimes have to weed, pick diseased pine needles off a tree, or dig a hole for a new fence post. Picking off individual needles is tedious, careful work, but doing it makes the plant look beautiful and helps fight the infection. Without fence posts, the garden would be trampled with foot traffic in places people should not be. Rotting posts will topple when someone leans against them, creating a liability risk.

Many engineers think they are above tedious and boring work, especially if it involves maintenance. Many push such work down the totem pole. Understand: the importance of the work is never directly related to one's enjoyment. Important work must be done, even if it is not fun. The whole team— not just those at the bottom of the totem pole — should share in the burden of the tedious-yet-important jobs, .

People can make tedious jobs bearable by taking a moment to reflect on their value. They can gamify the situation, racing to see who can accomplish the task the quickest. They can use tedium as a stand-in for meditation, allowing themselves to become completely wrapped up in the task at hand, performing it to the best of their ability. Such strategies are effective, whether one is gardening, washing dishes, filling out paperwork, refactoring a program, writing tests, or manually counting resistors for a hardware assembly kit.

End the Day With Something Usable

When working in the garden, the goal is to end a work session with something the public (that is, the customers) can use. Gardeners don't want to leave plants in a state that is obviously half complete. They remove equipment and the evidence of their presence at the end of each working session. The result is that the garden is in a continually viewable and pleasant state.

Sometimes this approach isn't possible. Cleaning the ponds is one example. The gardener needs to drain the ponds halfway, catch the koi fish, complete the draining, clean out the organic matter from the pond floor, prune plants that overhang the pond, refill the ponds, and replace the fish. That work cannot be completed within a single day, especially because it takes an entire day just to drain or fill a pond.

The gardeners do their best to minimize the impact of pond cleaning on the customer experience. The Japanese Team Garden contains four separate ponds. In the past, all the ponds would be drained at once, and they would remain empty for two to three months while cleaning proceeded. These ponds are a major feature of the garden. Keeping them empty for such a long period is unattractive and disappointing for customers. Now, the cleaning is scheduled so only one pond is out of commission at a time. Each pond is out of commission for only one week. As a result, they shorten the overall timeline for pond cleaning. Some customers still express disappointment when there is pond work, but at least they can enjoy the rest of the garden.

A common tendency for software developers is to work on a feature or branch for a long time, making infrequent merges to the master branch once everything is working. This is akin to keeping the ponds empty for three months at a time. When working in this style, developers often find that the master branch received changes that force them to update their branch or their changes conflict with another developer's pending changes. Instead of working on your own branch for months at a time, larger efforts should be split up, reviewed, and integrated in pieces. We don't want to wait until completing the entire overarching effort before we integrate our changes.

This is not a new idea within the software development world. Ending the day with something usable is akin to the extreme programming ideal of continuous integration, which is now taking the form of continuous delivery.

PUTTING IT ALL TOGETHER

At some point, almost all software developers and engineers work in maintenance mode on projects that already exist. To maximize their effectiveness, they must learn how to explore and improve these systems with the proper mindsets. Using the lessons learned from the San Francisco Japanese Tea Garden can help them deal with legacy software and systems.

I find it fascinating that two different operating environments can converge on similar difficulties, lessons, and practices. It is important to notice that solutions like incremental improvement and continuous integration and delivery are generally applicable to multiple fields.

According to the head gardener, processes similar to those proposed by agile and extreme programming adherents are not only adopted in the maintenance of the garden but also in the creation of a Japanese garden. "A difference between the process of how Japanese and Western gardens are built is that while Western gardens are generally built from a master plan down, Japanese gardens are constructed from the details up. A stone is placed, a tree is placed in relation to that stone, and then the path is constructed to take advantage of the best view of that scene and so on. It isn't that a conceptual plan doesn't precede the construction. It's just that in the work of putting the pieces together, the materials dictate the flow in a much more improvisational nature. Rigidly adhering to a master plan would miss the opportunities found during the process."

The value of these ideas continues to be rediscovered in different fields, and that is an indication of their importance.

REFERENCES

Brown, K. H. 1998. Rashômo: The multiple histories of the Japanese Tea Garden at Golden Gate Park. Studies in the History of Gardens & Designed Landscapes 18, no. 2:93-119.

Johnson, P. 2019. Hypotheses on systems and complexity. Embedded Artistry (February 11). Available at: https://embeddedartistry.com/blog/2019/02/11/hypotheses-on-systems-and-complexity.

JTGSF. 2020. Japanese Tea Garden San Francisco. Available at: https://www.japaneseteagardensf.com.

BIOGRAPHY

Phillip Johnston is a principal at Embedded Artistry, an embedded systems consulting firm focused on improving early-stage hardware product development. The firm builds a solid foundation for new products with systems architecture, modular firmware development, automated software quality processes, and organization-specific product development strategies. Johnston's experience spans consumer electronics, defense, automotive, robotics, cameras, drones, publishing, packaging, product design, industrial design, product life-cycle management, materials and assembly, supply chain, and manufacturing. Johnston can be reached by email at **phillip@embeddedartistry.com**.