Software Verification Games: Designing Xylem, The Code of Plants

Heather Logas, Jim Whitehead, Michael Mateas, Richard Vallejos, Lauren Scott, Dan Shapiro, John Murray, Kate Compton, Joseph Osborn, Orlando Salvatore, Zhongpeng Lin, Huascar Sanchez, Michael Shavlovsky, Daniel Cetina, Shayne Clementi, Chris Lewis

Center for Games and Playable Media
University of California, Santa Cruz
Santa Cruz, CA 95060
{hlogas, ejw, michaelm}@soe.ucsc.edu

ABSTRACT

Formal software verification is a software engineering technique for modeling a software system's source code, and then proving properties about it, such as freedom from security vulnerabilities. Though proofs are largely automated, formal source code modeling is time consuming and requires substantial human attention. Xylem: The Code of Plants is an iPad game where players make observations about unusual plants, and thereby model the behavior of software loops. With large numbers of players, the goal of Xylem is to have players model the behavior of large software systems faster, and at lower cost, than is currently achievable with formal verification experts.

We present an overview of major design challenges and approaches encountered in the design of a game for crowd-sourced formal software verification. A core challenge is the need to accurately reflect the structure of real-world software source code in a game setting, without showing the source code text to players. This goal created a tension with the need to design a game that is a fun, engaging experience, which often requires simplification of the core activities of the player. Another challenge is not knowing what is the solution to a particular problem, and the resulting difficulty in providing scoring feedback to the player. These design considerations are of interest for crowd-sourcing games in general, and software engineering games in particular.

Categories and Subject Descriptors

D.2.7 [Software Engineering]: Software/Program Verification
K.8.0 [Personal Computing] General – Games.

General Terms

Design, Human Factors, Verification

Keywords

Software verification game, game design, games with a purpose, crowd-sourcing games, human computation, software engineering

1. INTRODUCTION

Formal software verification is a general approach for proving whether a software system satisfies certain properties. Software systems that are safety-critical, for example, want to prove that the software will never enter an unsafe state. Software systems exposed to the internet would like to prove they are free from security vulnerabilities. The general approach used in formal verification is to create a mathematical model of a software system, and then use this model in conjunction with automated reasoning tools. If the mathematical model is an accurate enough representation of the functionality of the software system, the outputs of the automated reasoning tools are then accurate statements about the software system itself. At present, the process of creating these mathematical models of software system behavior is time consuming, and requires a high degree of mathematical and computer science training. Worldwide, there are insufficient people with this training to formally verify every software system that would benefit from this approach. Due to the sophistication of the people involved, formally verifying a software system is expensive.

The goal of the Crowd-sourced Help with Emergent Knowledge for Optimized Formal Verification (CHEKOFV) project, a collaboration between SRI International, the University of California Santa Cruz, and CEA (Commissariat à l'énergie atomique et aux énergies alternatives, the French Alternative Energies and Atomic Energy Commission), is to create a game where a player's in-game actions contribute positively towards an ongoing formal software verification. Our game, Xylem: The Code of Plants, is an iPad game where players make mathematical observations about synthetic plants, and thereby contribute to the formal modeling of a software system. Specifically, flowers found on the plants represent the value of variables found inside a source code loop (for, while, do), and players are asked to find relationships that describe the number of flowers. In so doing, players contribute to modeling the behavior of the loop, what is known as a loop invariant. The goal of Xylem is hence the crowd-sourcing of loop invariant expressions from non-expert game players.

Designing the core gameplay mechanic of Xylem was an exercise in taking a complicated task that requires a great deal of domain knowledge (and likely a PhD) and converting it into a much simpler job that requires none of this knowledge. In general, we wanted to soften the emphasis on math and create an experience that would be inviting to a large number of people. At the same time, we wanted players to be able to find a wide range of loop invariants and have the ability to express them, which indicated maintaining at least some of the "mathy" aspects of the game. The trick then was to find a balance between these two design impulses: abstract away as much of the "mathiness" as possible while maintaining a core gameplay mechanic of finding mathematical patterns and expressing them as math equations.

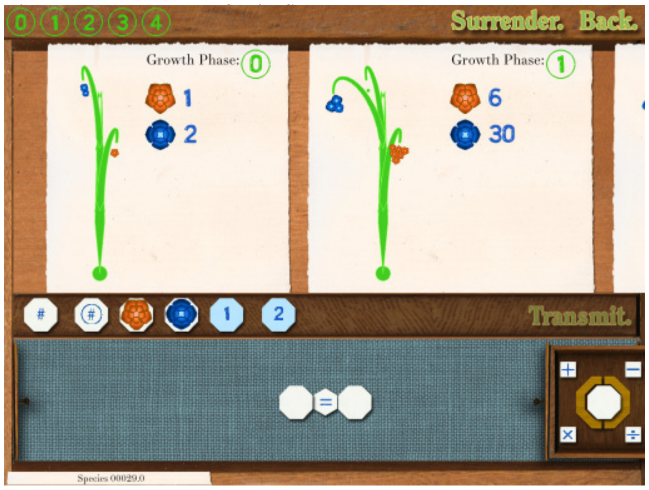


Figure 1. The main gameplay screen (FloraPhase Comparator), showing a plant with two flowers, in two successive growth phases. This represents a software loop with two variables, with values (1, 2) in the first loop iteration, and (6, 30) in the second.

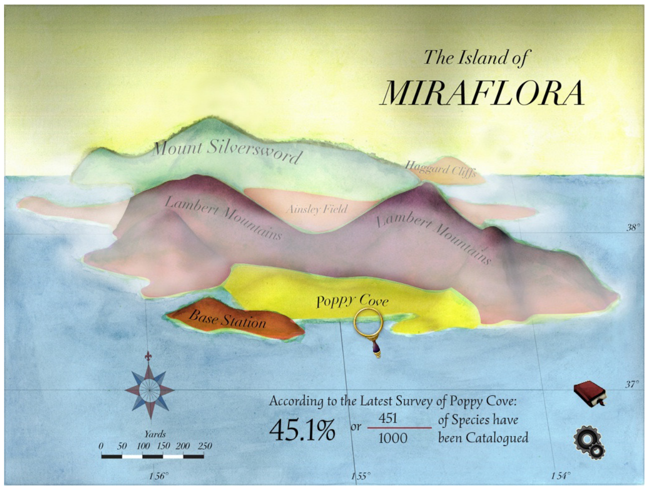


Figure 2. A map of the island of Miraflora. Tutorials are presented in the Base Camp region, and Poppy Cove is the first region of open gameplay. The figure shows that 451 of the 1,000 total problems in Poppy Cove have been solved by all players.

This paper presents an overview of the key design challenges encountered in creating Xylem. A common thread running through these issues is the tension between the need to accurately reflect both the software system and the formal modeling task, versus the need to create a fun and playable game. Another tension is not knowing whether a given player response is correct, and the related challenge of not knowing the difficulty of a given challenge. These factors—domain complexity, correctness determination, and difficulty modeling—all distinguish Xylem from classical crowdsourcing games (or, games with a purpose), where these issues are much less pronounced [10][11].

The paper begins with an overview of the game, and of the formal verification process. It then presents a series of design issues common to all software verification games, followed by an exploration of the most important issues encountered in the design of Xylem. Related work and conclusions complete the discussion.

2. XYLEM: THE CODE OF PLANTS

The year is 1921, the beginning of a decade of adventure. You are an esteemed botanist working at The University when IT happens. A mysterious island has appeared out of nowhere in the Pacific Ocean, an island reportedly filled with species of plants that have never before been seen by humankind—some with special medicinal properties. Scientists and adventurers from all over the world descend on the island, which has been dubbed Miraflora. All are called by the promise of making their mark on the history of botany and perhaps even striking it rich with the discovery of new valuable species. Your employers instruct you to pack your bags and catch the next boat out. The Green Rush is on.

Xylem: The Code of Plants is a logical induction puzzle game where the player plays a botanist exploring and discovering new forms of plant life on a mysterious island. Players observe patterns in the way a plant grows, and then construct mathematical equations to express the observations they make.

In order to find a valid observation, players must first examine the different growth phases (loop iterations) of a given plant species by comparing samples (growth phases) of that species side by side (see Figure 1). The quantities of each color flower can change from growth phase to growth phase. The player's job is to discover one rule (the loop invariant) that describes the plant's behavior in every growth phase. This one rule must hold true for each growth phase shown. Some sample rules include:

The number of blue flowers is always equal to the number of red flowers +1.

The number of blue flowers + the number of red flowers is always equal to 9.

The number of blue flowers is always equal to the growth phase number + 5.

Once the player has discovered a solution that holds true for each growth phase, their task is to construct a mathematical equation that describes the pattern. The player is provided with a set of tool tiles that can be dragged into a central workspace to form these equations. Tools include flowers which represent variables, mathematical operators, a tool which describes the growth phase (or iteration count of the loop) and "bonus" tiles (numbers that describe the loop at time zero). Additionally, parentheses are provided so that players can construct more complicated equations.

While solving puzzles, players also explore the strange island of Miraflora and begin to expose her secrets (see Figure 2). Pushing deeper into the island is a parallel collaborative experience. Each region of the island contains harder and harder problems. In order for a player to access interior regions the entire player base must collectively solve a certain number of problems in earlier areas. The number of problems solved in a particular region, as well as the number left to solve before the next region is unlocked, is shown to players via the island map screen.

As players complete puzzles and push towards the next island region, they occasionally discover collectible "secrets" of the island. These secrets are given to players as an occasional surprise when they solve problems. The secrets come in the form of text passages that offer a window into the history of the island. There are common, uncommon and rare unlockable secrets, and the intention is that there are enough of them that in order to put together all the pieces of the fictional puzzle, players will need to collaborate on the game's web-site and share what they have learned.

3. FORMAL SOFTWARE VERIFICATION

The central goal of Xylem is to ask human game players to help solve open problems in formal software verification. Specifically, Xylem players work in concert with the game's mechanics to perform a task called *loop invariant synthesis*. We chose not to burden players with knowledge of the underlying scientific task, but understanding the game's design constraints requires a working knowledge of software verification, loop invariant synthesis, and the particular ways in which Xylem's approach differs from traditional automated techniques for this task.

Formal software verification is the use of tools such as proof assistants and model checkers to automatically or semi-automatically verify properties of a computer program under consideration. Developers may want to prove safety properties (whether the program behaves badly when given certain inputs) as well as correctness properties (whether the program performs its intended function). These properties are described by experts using some formalism (e.g. statements in first-order logic), and program verification tools attempt to prove these statements using the semantics of the programming language.

The basic technique of formal software verification is, following Hoare [7], to work towards a proof of the property by using each statement of the program as a step towards proving the property. If we want to prove that the variable X is always less than ten (an *invariant* which the program should ensure), we must start from the beginning of the program and prove that no individual statement increases the value of X beyond ten. If, during this proof, we happen upon a conditional statement where either branch could apply given what we know of the program's possible inputs, we must prove that no matter which branch is taken, X remains less than ten. Since the branches may have different outcomes, we must fork our proof to handle both possible worlds. Language features like general recursion or while loops complicate this analysis: it cannot be known in advance how many times an arbitrary loop might execute due to the undecidability of the Halting Problem, and since each iteration may require a forking proof, we quickly come upon a potentially infinite (and thus invalid) proof structure.

Program verifiers work around this pitfall by using annotations called loop invariants. Loop invariants are statements which must hold every time a loop is entered, either on the initial run or on subsequent iterations up to the loop's termination. If we have some property which is always true whenever the loop is entered, then it no longer matters (for the purposes of a proof) how many times the loop is run: the property—and more importantly, its logical consequences—will always hold by induction [6]. To put it another way, a loop invariant collapses a loop into a single statement, like finding a closed-form solution to a differential equation.

As an example, consider a loop L which, on each iteration, decrements the variable X from before. We know that X will always leave L with a value smaller than (or equal to) the one it entered with; that is one invariant (written $X <= X_0$, if we use X_0 to indicate the initial value of X on first entering L). Now, if we can prove that X is less than ten upon entering L, then it will remain less than ten no matter how many times L executes.

Loop invariants are extremely useful, but they have to be invented individually for each loop and, often, for each property to be proved. Programmers or analysts generally provide these annotations, but discovering these invariants automatically has long been an area of active research (an excellent overview of early work here can be found in [1]). There are two dimensions by which we can classify automated approaches to generating (or *synthesizing*) loop invariants. The first criterion is how the technique chooses which invariants to synthesize: does it construct invariants top-down by working from a property to be proved and testing hypotheses that might justify the property against the loop's behavior, or does it build sets of invariants from the bottom up by summarizing the loop's activity? The second criterion is whether the approach works with the program code and the language's semantics, or whether it works with concrete data from real program executions.

Top-down techniques are most applicable when the set of properties to be proved is known in advance, and bottom-up techniques produce invariants that can be used to prove many different properties. The technique described by Chadha and Plaisted is top-down and semantics-based [1]: it starts from a property to be proved and justifies it by working backwards through the loop. This is the most common quadrant for invariant synthesis techniques to inhabit. The classic example of a bottom-up, data-driven approach is Daikon [5], which uses analyst-provided templates to synthesize possible invariants among the variables of interest, discarding the propositions that are not attested by concrete test cases.

One particular use case where bottom-up approaches excel—and the case for which Xylem is intended—is in annotating large legacy systems with loop invariants to help prove broad categories of safety and security properties. Inductive program verification as described above is an all-or-nothing proposition: every line and loop must be provable, or the whole edifice collapses. Invariant synthesis makes formal verification feasible for existing code: annotating a large and complex system with invariants from scratch is a daunting proposition.

4. DESIGN ISSUES FOR FORMAL SOFTWARE VERIFICATION GAMES

Though Xylem focuses on supporting players in developing loop invariants, other software verification games are also possible. For example, the game Flow Jam focuses on "taint" analysis, such as proving that all strings have been checked for security vulnerabilities before being sent to a back-end database. Another game, CircuitBot, focuses on pointer analysis (both games available at verigames.com). Though these games all have different focus areas, they all have shared design concerns. Any attempt to create a game for some aspect of formal verification needs to address the following design issues, which are specific to this domain, though they touch upon design concerns common to any crowd-sourced game [10][11][2][9].

Representing Software. Players of a software verification game must be able to understand some facet of the behavior of a piece of source code. Since game worlds are visual, there must be some visual representation of the source code within the game. That is, any software verification game must have a visualization of some part of the behavior of software source code. With this visualization, a player can develop an understanding about the behavior of the software, and then take action within the game world. In a software verification game, these game world actions contribute to verification goals, such as describing loop invariants.

Defining What the Player Can Say About the Software. Through their gameplay, players of software verification games make observations about the behavior of a piece of software. In Xylem, these observations concern the behavior of a particular software loop. In traditional formal verification, software engineers write statements in a logical language, such as first order logic, to describe the software. However, writing logical statements is not usually considered to be fun. A key design issue is then how to give the player the tools to state a wide range of interesting observations about the software, while avoiding (or hiding) the full complexity of logical languages.

Rewarding the Player. Selection of a reward system is a complex topic, involving both traditional motivation [11] and quality control [9]. Ideally, a reward system provides positive reinforcement for high quality responses, and this keeps players motivated. Traditional computer games have well defined good and bad player behaviors, and it is easy to build reward systems around them. Software verification games tend to have the property that it requires substantial computation, outside the game, to determine whether any given player response is beneficial to the overall software verification goals. In short, it takes a long time to know whether a player submitted output is good and should be rewarded.

There are two components to determining if a player provided response is beneficial. First, there may need to be a determination of whether the player response is correct. In the case of loop invariants, is it the case that a player correctly described the behavior of the loop? The second issue is whether the player response helps push forward the overall software verification goals. An example using loop invariants highlights this second point.

In the case of loop invariants, a good loop invariant will allow an automated analysis of the values of variables to provide more precise ranges on the value a particular program variable can have at a given point. For example, with a given integer variable, *z*, before the loop invariant from a player, its value might be analyzed as being between 0 and the maximum interger value. After receiving a loop invariant, this range might now be refined to lie between 0 and 128. A more precise variable might then permit a proof to proceed. If variable *z* is an array index, *a[z]*, and the array *a[]* is 256 elements long, then it is now possible to prove that *z* cannot cause an array bounds exception. However, it is also possible that a player-provided loop invariant might not help refine the value of *z* at all; some other part of the code might need to be described for this to happen.

5. XYLEM DESIGN ISSUES

Xylem has an unusual set of design goals because it is both a game and a research effort in formal software verification. As a game, it has to provide entertainment, in the form of enjoyable, interesting, and playable experiences. However, as a science tool, it must enable, and encourage users to assist program verification through game play. These goals are often in tension, and our efforts to navigate the resulting constraints have shaped the structure of the game.

5.1 Audience

The idea of building a game for finding loop invariants is itself a bit odd, as the problem is highly technical; it is commonly pursued by people with deep expertise in formal logic and theorem proving. It would be natural to design a logic-oriented puzzle game for players with these skills, but we have chosen, instead, to cast Xylem as a crowdsourced game, following the premise that group wisdom can address seemingly intractable scientific problems (like folding proteins [3]). This creates a challenge to represent the software verification task in an accessible form. We raised the bar even further by attempting to attract the largest possible player audience. At the same time, we knew that the audience wasn't the market definition of "casual", because the design we had in mind was going to require more math than we thought the "casual" audience would be comfortable with; we knew that we were not going to be able to design a casual game break-out hit in the vein of Angry Birds or Candy Crush Saga.

Instead of thinking in terms of casual and hard-core, we talked about the game in terms of its "niche". What we were producing was a niche game, but our goal was to create what we called a "low niche" experience. That is, a game that was not exactly casual, but did appeal to a casual-type audience that is comfortable with math. The direction we chose was one of a relaxed puzzle solving game, and hence our challenge was to create a safe, comfortable space for math puzzle play that would be accessible to as wide an audience as possible. Considering activities such as crossword puzzles or sudoku, we imagined a non-stressful but nonetheless brain exercising experience done in the evenings while relaxing in a comfortable chair. An unwinding activity for someone who likes to stimulate their brain.

This direction set the stage for nearly all of our design decisions. The game should be untimed, easily picked up and put down again, portable (for playing while in the easy chair), and the overall aesthetic should support an intriguing but relaxing atmosphere. Choosing the visual metaphor of flowers gave the core gameplay a sense of scientific discovery in the field and further helped us develop our design direction. We built off this feeling of exploration and discovery to construct the game fiction and supporting mechanics.

5.2 Representing the Software

Xylem is intended to assist the verification of software obtained from multiple sources (public, private and governmental) where source code privacy issues come into play. A commercial software vendor would not want snippets of their code being shown to game players, since a team of players might be able to reconstruct the software system. In order to address those issues, we adopted the rule that players cannot see the source code for the software under analysis. This created a significant design challenge, as the player must make meaningful observations about the behavior of loops in a software system without ever seeing the source code!

This decision shaped the design around the goal of finding invariants from simulated run-time data. That is, in the game, we present the values that variables have inside a loop, across several iterations of the loop. This approach stands in contrast to the predominant approach for deriving loop invariant expressions in which a person examines the program source code, which is viewed as a series of logical statements. The net result is that Xylem employs a relatively unexplored solution path to the underlying technical problem, in parallel with being a crowd sourced, quasi-casual game. Our approach can be viewed as the human computation dual to the machine driven approach used by Daikon [5], which employs machine learning to determine invariants within a software system from variable values derived from a run time execution (we note that our loop variable values are computed statically, however).

The key insight that came out of early design meetings was that the task of finding invariants could be cast as an inductive reasoning task. That is, given some information (data produced by the loop in question), the player can attempt to state what the rule is (the loop invariant) that connects all the data. Other examples of games that utilize inductive logic are Zendo and the card game Mao [8]. The core mechanic that arose from this approach was one of comparing different iterations of a loop and then expressing a mathematical equation that describes a pattern that holds for every iteration.

To represent the software loops and their data, we wanted to find a visual metaphor that mapped well to the domain, was appealing (or at least not off-putting) to a large number of people and was also flexible enough to represent the range of programmatic data structures that we knew we eventually would want the game to support. Our research into the code visualization literature did not yield any existing metaphor that can represent a wide range of data structures; most visualizations focus on just a single data structure, and do not have any kind of narrative theming, instead just showing boxes and arrows. After a bit of experimentation (we initially considered a space theme with a tabular representation of the variable values), we decided on plants as our visual metaphor.

The kingdom of plants fit all of our criteria. Plants are part of most people's everyday existence, and as living things have a certain universal appeal. Plants grow and change over time, as does the data produced by loops. Using features of a plant (number of flowers, number of petals on each flower, number of leaves, etc.) as variables allowed players to observe a pronounced visual change as they moved from iteration to iteration in the loop. The plant kingdom is also startlingly diverse—a little bit of research quickly turned up plant structures that would map quite easily to different code data structures. Integers can be represented by the number of flowers on the plant, roots with nodules can be used to represent arrays, and trees map directly onto a plant's structure. Plants also provide the opportunity to build a story and gameworld around them in a richer way than boxes and arrows.

5.3 (Not-Quite) Casual Math - Describing Loop Invariants

Stormbound (see verigames.com) also uses loop invariants as the foundation of its gameplay but abstracts out all math and numbers, asking players instead to choose from a pre-defined palette of common patterns to describe the behavior of spatially-oriented symbols. Although we could have taken a similar approach, it was important to us to allow players to describe the patterns they discovered in Xylem as mathematical equations. We knew this would eliminate part of our potential audience (we would never be able to call Xylem a true "casual" game, as the term is used in the game industry), however we felt this approach was nonetheless productive for several key reasons.

Perhaps most importantly, direct equation building allows for a wider range of invariants to be created and submitted. Each loop may have more than one possible invariant, and each invariant collected is useful to the backend processes that annotate the originating software. Allowing players to construct their own equations opens up the flexibility to receive multiple solutions for each puzzle while simultaneously taking advantage of the strengths of different player skill levels and play styles. This creates a customized challenge for each player depending on their sophistication while taking best advantage of the strengths of crowdsourcing.

In addition to direct gameplay ramifications, focusing the main game activity on building equations has some practical aspects as well for possible future iterations. The framework is easily expandable in the future to include more tools if needed/desired. Several tools have appeared in earlier versions of the game that were later removed to avoid confusing players. However, allowing more sophisticated players to unlock specialized tools (such as the mathematical logical construct implies) is a possibility for future updates. The direct equation building approach also supports implementation of different data structures as game levels without having to completely redesign parts of the game to deal with them.

A core tension of the game design is the desire to simultaneously have a large range of players while also having a large expressive range for their observations about loops. At times it seemed almost a one-for-one trade off between appealing to our desired audience and allowing for greater expressibility of the tool set. Early versions of the game, for example, included tools such as the mathematical symbol for the logical concept of "implies". While this would have allowed for a greater range of possible invariants that could be constructed, taking the time to explain the concept of "implies" to someone unfamiliar with it seemed like it would bog down the game flow and create a mental stumbling block. Ultimately we decided that losing this bit of expressivity was less important than supporting a wider player base. Even so, Xylem still is sufficiently mathy that it triggered math anxiety in some of our playtesters.

5.4 No Person is an Island: Cooperative Play on Miraflora

We knew from the beginning that we wanted players to play this game together, but sorting out exactly how was a long and involved iterative process. To complicate matters, we had to be very careful about not allowing players access to each others' personal information, due to privacy constraints from our research funding.

Given the audience we were after, and the emotional feel we wanted for the game, we opted for a collaborative scenario, but a very light one. We wanted to create player investment by encouraging the feeling that all players were working together towards some greater goal. It was also important to us that players knew other people were playing the same game at the same time as they were. We wanted to support that feeling of being "alone together" which we felt would be more enticing to our target audience than the feeling of playing a single player game.

We also imagined an audience that would be drawn in further to the fiction of the game, and we wanted to create an impetus for players to interact on the forums (in addition to helping each other with hard problems). We therefore created a complex backstory to the island which is revealed in a semi-random manner one clue at a time. Players can collect clues, compare them on the forums and discuss theories. The intended effect is a meta-layer of collaboration that occurs outside the game itself to reward and draw in the core player base.

5.5 Rewarding the Player

Xylem posed an unusual problem when it came to offering players feedback. Quite simply, the game has no way to gauge the utility or strength of a player-provided invariant. There are no established techniques for ranking the difficulty of an invariant-finding problem, assessing the quality of a solution, measuring incremental progress, or knowing when a problem is done. In this situation, the task of designing game mechanics merges with basic research on invariant discovery/analysis.

The only way to know for a certainty the usefulness of a given invariant is to test it out on the original loop, in the context of a larger verification problem being explored via the use of out-of-game software verification tools. This suggests that the only way to provide feedback to players is to have them submit their answers, wait some indeterminable amount of time for a remote expert to test out their solution, and then provide scoring based on this feedback. Games don't tend to work that way. Players don't complete a level in Candy Crush Saga and then wait a day for their score.

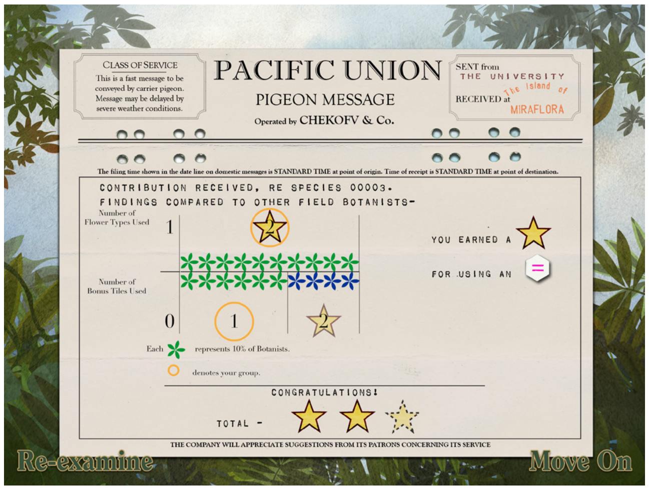
Ultimately, we can only use what we know. We know that using as many of the variables in the loop as possible is better than using fewer. We also know that utilizing the data from the time zero iteration from the loop generally resulted in better invariants (these became represented as the blue "bonus" tiles in the game). We know that—usually—stating that one thing in the invariant is equivalent to another thing is stronger than saying that one thing is less than or greater than the other thing. And we know that we want to encourage players to come up with a variety of invariants. With a lack of solid answers for how to give players feedback, we built our scoring system on these four considerations.

Since we also didn't know whether a solution with certain numbers of variables or bonus tiles was even possible in a given problem, we couldn't simply give stars based on the number of variables or bonus tiles used. Instead we used a model from SpaceChem. At the completion of a level, SpaceChem shows the player a visualization of how they have performed compared to other players of the same level. We adapted this to Xylem by plotting out on a chart the number of variables and bonus tiles used by a player as compared to other players of the same puzzle (see Figure 3). If a player has used the same or greater number of variables (and, separately, bonus tiles) than the highest number used so far by any player, then they receive a star. In the same way, players are scored for bonus tiles. Using an equals sign in a solution instead of an inequality gives the player another star. In this way, a player can receive a score of up to three stars per puzzle. Additionally, a special stamp is granted to the player if they have created a "novel solution". That is, a solution that hasn't yet been recorded for this particular puzzle.

Ulimately, we learned that this scoring approach is unsatisfactory for players. Bonus tile stars were often felt to be artificial, rewarding players for unusual equations, not for truly useful ones. Some useful equations only use a single variable, but these are scored lower that multi-variable equations. Scoring did not reward submission of unique solutions, even though these are very useful for the goal of developing invariants. To address these issues, future versions of Xylem will adopt an approach used in other crowdsourcing games of having players rate the equations developed by other players, thereby bringing human expertise to bear. This will allow interesting and novel solutions to be rewarded, but at the cost of some delay in the player receiving feedback on their submissions (players will still be able to compare their equations to others right away).

5.6 Problem Difficulty and Player Experience

Just as it is not possible to know the utility of an invariant, it is also not possible to know the difficulty of a puzzle. This creates a major


CLASS OF SERVICE
This is a fast message to be conveyed by carrier pigeon. Message may be delayed by severe weather conditions.

PACIFIC UNION
PIGEON MESSAGE
Operated by CHEKOFV & Co.

SENT from THE UNIVERSITY
RECEIVED at The Island of MIRAFLORA

The filing time shown in the date line on domestic messages is STANDARD TIME at point of origin. Time of receipt is STANDARD TIME at point of destination.

CONTRIBUTION RECEIVED, RE SPECIES 00003.
FINDINGS COMPARED TO OTHER FIELD BOTANISTS-

YOU EARNED A
FOR USING AN

Each represents 10% of Botanists.
denotes your group.

CONGRATULATIONS!
TOTAL -

THE COMPANY WILL APPRECIATE SUGGESTIONS FROM ITS PATRONS CONCERNING ITS SERVICE

Figure 3. A representative scoring screen for one problem, showing that the player matched all other players in number of flower types used, but used fewer bonus tiles than top players.

design challenge: how to craft the difficulty curve for each player.

We examined several approaches to estimate puzzle difficulty. One was to link the structure of a loop (visible to the front end) to difficulty. More "guards" (conditionals in the loop) suggest more paths and potential for a more logically complex invariant (`if A then I1, if B then I2`). However, it is unclear if this is true in practice. A second approach was to employ social ranking analogous to webpage ranking in some search engines. If skilled players found a problem hard, it must be hard. If novice players solved a problem, it must be easy. This would be useful for serving up solved problems to new players in a sensible order, but the issue of ranking new problems remains open. Additionally, there is no firm basis for applying similarity metrics (associating loop features with loop difficulty) to leverage social analyses. A third option was to employ machine learning to classify problem difficulty from player ratings, but again, this is stymied by a lack of understanding of the underlying feature base. All of these approaches represent research into invariant-finding tasks; from the perspective of game design, it was easier, and sufficient to employ heuristic measures.

The heuristic measure of difficulty was based on the experience of team members playing pre-release versions of the game. We assume that integer based problems are less difficult than array based problems. We assume that working with more variables is harder than working with fewer variables. We assume that working with larger numbers is more difficult than working with smaller numbers. Based on these assumptions, difficulty profiles are assigned to each problem and problems are grouped by difficulty into different regions on the island. Unfortunately, offering players a smooth difficulty curve isn't possible. Other factors—which are hard to test for, especially on a large scale—can influence how difficult a given problem is. Therefore, a player could breeze through the first three problems in the "easy" region, then encounter a very hard problem followed up by another easy one. This does not support handcrafted difficulty curves, and occasionally causes player confusion and frustration.

5.7 Teaching the Game

The tutorial of Xylem began after the theming and general concept of the game was pinned down. The task of onboarding new players to the game proved to be a massive challenge: presenting a problem this involved as an approachable, fun experience that is easy to learn required much research, design, and iteration. The first versions of the tutorial existed as paper prototypes, with which the designers were able to test different ideas quickly and efficiently with many testers. Only through extensive testing and iteration were we able to settle on a tutorial that effectively leads the player into the game while teaching them the basic skills they need to know in order to succeed. The tutorial design and polish took place in parallel with the core game design, requiring as much design time and effort as the rest of the game combined. Early feedback from external game players indicate that the tutorial does successfully train players to play the game, but is too long, a direct consequence of the many game elements that need to be taught to the player.

5.8 Fictional Setting

From the start of game design activities, attracting and retaining players was a key concern. One primary approach for retaining these players is the inclusion of a rich narrative backstory in the game. This narrative framing is designed to attract players who might not otherwise give the game a chance while evoking a connection to the experience offered them. The narrative needed broad appeal without too many complicated narrative trappings that would make the fiction harder to penetrate. An undercurrent of mystery and intrigue was woven into the game to create an sense of compulsion to the game while supporting the core gameplay.

Using plants as a gameplay metaphor had been decided on long before final decisions were made about theming for the game. Given the nature of the gameplay (examining and stating observations of plants) it made sense that the character would be a botanist of some kind. Given that the game's procedurally generated flowers did not specifically map to existing plant species and that discovering new things is more interesting than examining existing things, it made sense that the player would be discovering new plant species that had never been seen before. But still unanswered was the setting of the game. Did it take place in space on a newly-discovered planet? An alternative Victorian steampunk world? On Mars?

All of these options were examined (and more) but were found lacking for various reasons. Finally we hit on the idea of setting the game in the early 1920's, on a mysterious island that appears out of no where. This time period was an important one for exploration in the world, and at the same time the public's imagination was full of pulp adventure and lost lands. The game's story of botanists flocking to a newly discovered island fit with the themes of the time period well. At the same time, this fictional framing did not require too much backstory for players to potentially get mired in and since the theme was not overtly science fiction or fantasy would appeal to a much broader audience.

5.9 Aesthetic Experience

The game's aesthetics were developed with the dual goals of creating a pleasant place for the player to spend their time and creating consistency with the 1920's theme. Because Xylem is meant to be a slow, contemplative game (but with a hint of adventure), it was important that the visuals and music fostered this atmosphere. A great deal of research into map styles in the appropriate time period was done by our artist before the specific watercolor look and map orientation was decided upon. Likewise, our sound designer researched silent adventure movies and period jazz and classical music to develop the ambient music pieces that play on the map screen and puzzle screens.

The photorealistic look of the game interface came about almost as an accident. While developing the look for the opening screen (when the player is about to disembark for the Island), the artist arranged some items found around his house on a desk and took a photo to show the direction he was considering. The team loved the actual photo so much that it was further developed and used as an in-game asset. At that point, we knew that we wanted a similar look for the actual in-game UI. This direction adds to the first person immersion of the game and, because of our artist's skillful rendition of the interface combined with the satisfying sound effects created by our sound designer, helps create a very tactile feeling to the game which encourages players to manipulate the playing pieces.

5.10 One for the Road

The decision to adopt the iPad as Xylem's target platform again goes back to considerations of chosen audience and how the game would most likely be played. The iPad as a platform appeals to a wide variety of people. According to one study, the majority of tablet owners are between the ages of 35-44, and are spread equally amongst the genders, whereas smartphones skew a bit younger. Given the nature of the gameplay—contemplative, like working a sudoku or crossword puzzle—we imagined players playing the game while sitting somewhere comfortable for at least thirty minutes. More than either a smartphone or PC, the iPad has a form factor that encourages this sort of behavior.

6. RELATED WORK

The game Xylem can be viewed through multiple lenses. First and foremost, Xylem is a game that supports the crowd-sourced elicitation of loop invariant statements. Design issues for crowd-sourcing in general then apply to Xylem (see [11] for a survey). Since Xylem is a crowd-sourcing game, it can viewed as a *game with a purpose (GWAP)*, and the discussion of design issues about such games is relevant [10]. Since Xylem involves people performing work that computers cannot, it can be viewed as a form of *human computation* (see [2] for design issues concerning motivation and evaluation in this context). Since Xylem uses a game reward system to motivate players, it is also a form of *gamification* [4]. We view Xylem as involving a deeper use of game design that typical gamification efforts, and as disguising the core activity more than typical human computation tasks. However, all of these lenses provide insight.

Four other software verification games were created as part of the DARPA Crowd-Sourced Formal Verification program: Stormbound, Flow Jam, Ghost Map, CircuitBot, and Ghost Map. These web-based games are accessible via the Verigames web portal at verigames.com. Stormbound shares similar goals to Xylem, in that it also uses invariant finding of loops as a way to make progress on formal software verification. However, instead of asking players to identify patterns with mathematical equations, Stormbound presents them with a spatial interpretation of the loop data which is represented as icons ("sigils") on a grid. The players' job is to identify and choose from a pre-defined selection common patterns among the sigils. Players can also combine smaller statements into larger ones by selecting a particular pattern and then selecting another.

Both Flow Jam and Ghost Map provide graphical representations of data flow and control flow within software, and broadly focus on ensuring a particular condition holds across a particular path. Circuitbot focuses on pointer analysis.

Three games were particularly impactful on the design of Xylem. FoldIt, developed by the University of Washington, is a game with the aim of crowdsourcing protein folding [3]. While FoldIt's concept of "crowdsourcing for science" directly inspired the creation of the gaming portions of the CSFV project, our goal was to better maximize the "crowd" of our game by creating a more accessible player experience. Dragon Box is a game that teaches algebra by drawing players in with puzzles featuring cute dragon characters instead of numbers. As players continue to play, the dragon characters are slowly replaced with actual numbers and mathematical symbols. The design of the game successfully makes algebra accessible and fun. Dragon Box served as a constant inspiration to us in our goal of making Xylem more casual player friendly. The final game, Zendo, designed by Kory Heath, is an inductive reasoning puzzle game, which demonstrates that inductive reasoning can be the core of a fun game.

In the realm of techniques for finding loop invariants from variable values, Daikon is an automated technique for bottom-up loop invariant synthesis given only program data as input [5]. Daikon's invariant search uses a handful of a priori patterns, creating a tradeoff for Daikon: it gains computational tractability but sacrifices completeness. Daikon also assumes that the test cases given by analysts completely describe the program's behavior. Moreover, the tool cannot know in advance how useful or relevant the invariants it finds might be (though it employs a variety of heuristics to cull trivial invariants).

7. CONCLUSION

The design of Xylem presented a series of substantial challenges. The problem domain is complex, and math-focused. It wasn't possible to know, a priori, the difficulty of problems, or whether a player's solution was useful. Though players need to make useful statements about a software system, they could not see its source code. This paper presents the range of design issues encountered in creating Xylem, and our approach to resolving them. These issues are relevant to the design of software verification games specifically, but are also of interest to designers of crowd-sourcing games and software engineering games in general.

In the 3 months since its public release on December 4, 2013, Xylem has been downloaded by over 1,400 people, and game players have solved over 3,700 problems (created over 3,700 loop invariant descriptions). After an initial peak of 240 unique active players per week, Xylem settled down to a steady-state of 10-25 unique players per week. We feel this is a modest start towards our goal of a robust community of game players working together to model loop behavior in a large software system. It also highlights the challenge of creating crowdsourcing games for complex mathematical domains.

ACKNOWLEDGEMENTS

We gratefully acknowledge the contributions of our collaborators at SRI and CEA, especially John Murray, Martin Schäf, Min Yin, Natarajan Shankar, Sam Owre, Florent Kirchner, and Julien Signoles.

This material is based on research sponsored by DARPA under agreement number FA8750-12-C-0225. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA or the U.S. Government.

GAMES REFERENCED

Angry Birds, Rovio Entertainment, 2009
Candy Crush Saga, King, 2012
Circuitbot, Left Brain Games, Texas Tech University, Kestrel, 2013
Dragon Box, WeWantToKnowAS, 2012
Flow Jam, Univ. of Washington, 2013
FoldIt, Univ. of Washington, 2008
Ghost Map, Raytheon BBN, 2013
SpaceChem, Zachatronics Industries, 2011
Stormbound, voidAlpha/Galois, 2013
Zendo, Kory Heath, 2001

REFERENCES

[1] Ritu Chadha, David A. Plaisted, On the Mechanical Derivation of Loop Invariants, *J. Symbolic Computation*, 15(5-6), May–June 1993, pp. 705-744.

[2] Methods for Engaging and Evaluating Users of Human Computation Systems, in Jon Chamberlain, Udo Kruschwitz, Massimo Poesio, P. Michelucci (ed.), *Handbook of Human Computation*, Springer Science+Business Media New York 2013.

[3] Seth Cooper, Adrien Treuille, Janos Barbero, Andrew Leaver-Fay, Kathleen Tuite, Firas Khatib, Alex Cho Snyder, Michael Beenen, David Salesin, David Baker, Zoran Popović and Foldit players. The challenge of Designing Scientific Discovery Games. *Proc. Foundations of Digital Games*, 2010.

[4] Sebastian Deterding, Miguel Sicart, Lennart Nacke, Kenton O'Hara, and Dan Dixon. Gamification. Using Game-design Elements in Non-gaming Contexts. In *CHI '11 Extended Abs. on Human Factors in Computing Systems (CHI EA '11)*.

[5] Michael D. Ernst, Jeff H. Perkins, Philip J. Guo, Stephen McCamant, Carlos Pacheco, Matthew S. Tschantz, Chen Xiao. The Daikon System for Dynamic Detection of Likely Invariants. *Science of Computer Programming* 69 (2007) 35–45.

[6] Floyd, Robert W., Assigning Meanings to Programs. *Mathematical Aspects of Computer Science* 19, pp. 19-32 , 1967.

[7] C. A. R. Hoare. 1969. An axiomatic basis for computer programming. *Commun. ACM* 12, 10 (October 1969), 576-580.

[8] Mao card game, Wikipedia, http://en.wikipedia.org/wiki/Mao_(card_game)

[9] Alexander J. Quinn, Benjamin B. Bederson, Human Computation: A Survey and Taxonomy of a Growing Field, *CHI 2011*, May 7–12, 2011, Vancouver, BC, Canada, 1403-1412.

[10] Luis von Ahn, Laura Dabbish, Designing Games with a Purpose, *Commun. ACM*, August, 2008, Vol. 51, No. 8, pp. 58-67.

[11] Xu Yin, Wenjie Liu, Yafang Wang, Chenglei Yang and Lin Lu, What? How? Where? A Survey of Crowdsourcing, *Frontier and Future Development of Information Technology in Medicine and Education*, Lecture Notes in Electrical Engineering 269, 2014.