‘Problems and Programmers’ Game Report

This report is addressed to the developers of the Problems and Programmers learning game. Opinions expressed are of graduate Software Engineering students at the University of Waterloo, currently enrolled in CS 846, Software Project Management. A group of these students played P&P and then submitted a summary of their experiences and ideas about the game.

Good things about the game:

1. The true point of the game becomes apparent as you play it. Problems kept hitting the teams that hired cheaper programmers with lower personality ratings. It was clear that fewer higher quality programmers would have saved the team a lot of grief.
2. The game makes the players think about the actions they take. The player always has a variety of choices to make. From the timing of moving from one phase to the next, to the way resources are allocated, to deciding on what cards to keep at hand and which ones to discard, the game kept players thinking at all times.
3. The game forces the players to remember the stages of the waterfall model, and think about how neglecting certain stages affects the outcome of the project.
4. It became apparent that if the players rushed, they would get held up in later stages, and lose even more time, which demonstrated the main issues about the Software Development Cycle.
5. The game emphasizes how writing rush code results in more bugs.
6. Another beneficial learning aspect of the game is the fact that “nasty bugs”, which are hard to fix and result in a loss of a lot of man-hours, rarely appear if proper design methodology is followed.
7. The game felt very real. The teams that played the game actually weren’t sure if they were winning or losing until the end. Good and bad surprises made the game interesting.
8. The game can be played by either individual players or teams.

Negative aspects of the game:

1. The learning curve is steep due to lack of clarity in the instructions.
2. At times, it is ambiguous whether a card can be played or has to be held until it is used. Color coding cards to explicitly convey this message would be helpful.
3. A player cannot cast a problem on his opponent if the opponent has not started the coding phase. The game would have been more interesting if problems could be introduced during the design phase. This would also be penalizing those who went into design too quickly without spending enough time on the requirements phase.

4. ***The problems that increased the duration of the project did not result in any increase in the duration of the project, which is not how real projects behave.***

Anne’s Comment: What does the above mean?

5. Figuring out how much work a programmer can do is confusing at times. For example, the programmer is at skill level 4 and the player has a concept card that doubles the skill of the programmer. Then there are a large number of different things that programmer can do. Each one of these things had a cost that had to be deducted from the 8 time units the programmer had. On many occasions, we had to spend extra time to make sure the math was correct.

Anne’s Comment: The above is vaguely confusing. How can it be reworded?

6. Some instructions on the *concept* cards in the Main deck were ambiguous. The difficulty was usually the wording, as some specifically stated that they must be in play, while others alluded to it, but didn't state it. The main reason this mattered is that one player assumed that the card should remain in their hand, which prevented them from picking up an extra card during their next turn, while the other player put these cards into play and got an extra card.
7. The number of players that the game is intended for is still unclear. The game certainly works for two players or teams, but the instructions often make it sound that multiple players can play. This, however, seems to be a problem as there are a limited number of *document* cards (36), and in order to progress through the game as suggested, at least 12 clear *document* cards, of which there are only 24, are needed per player.
8. The fact that the Waterfall model is stressed puts importance on this model, whereas other SE process models are neglected. Obtaining certain concept cards allows you to use prototypes, but it isn't a path that can be chosen by each player.
9. The problem cards have requirements that are too easy to surpass (i.e. requirements < 2) and do not allow players to be penalized. In real life, generally it is not possible to obtain this many clear requirements.
10. To win the game, most players need to fulfill the basic requirements. It would add more realism if sometimes it is not possible to obtain certain goals. Many players only learn when they encounter certain issues and thus realize how to avoid them in the future.

Suggestions from the students:

1. Produce a Quick Rules Reference that the players can use during the game.
2. Provide a mechanism to quickly figure out how many time units are remaining for each programmer during the coding phase. Maybe a small Abacus-type device could be used.
3. More emphasis should be placed on diagrammatic and tabular representations than on narrative in both instructions and on the cards. This could help speed up the pace of the game and make it more enjoyable.
4. Perhaps a player should be allowed to go back to the previous phase if needed. For example, a player who notices that he or she is being hit by a lot of *problems* while in the coding phase because he or she did a poor job during the design phase may want to go back and fix the design and then continue coding. There could be a penalty such as cost increase for going back in addition to the cost that this action incurs. There also could be a limit on the number of times a player may go back.
5. The number of requirements and design cards the player has to pile up to be safe from the *problems* should be correlated with the complexity level of the project. This can be easily done by color-coding the projects and the criteria section of the problem cards so that it would be easier for the player to identify the criteria that apply to the project at hand.
6. There could be multiple versions of the game that adopt other approaches. For example, the **Prototyping Version** may force the player to circle through the different phases spending no more than two turns on each phase. The *problems* the user can be hit with could depend on the iteration he or she is in. For example, more severe problems could be only used against the player if he or she is in the fourth iteration or more and still has not covered the requirements phase properly. A *User* card deck could be introduced and, at the end of each iteration, a card could be drawn from this deck. This card could penalize the player if he or she had done a bad job so far to reflect the dissatisfaction of the user. The players would of course have to all use the same approach.
7. The instructions should recommend the number of players that the game is intended for.
8. The *Document* deck is referred to as *analysis and design* cards, which caused confusion. At first, it seemed as if a deck of cards was missing. The instructions should be more consistent.

9. Producing colored cards is impractical and expensive for a few hours of playing. One suggestion is that a black-and-white version of the card images be produced, in order to make playing the game more reasonable to the average student or one-time players.
10. It would be beneficial if the instructions did not suggest the number of cards a player should have before moving to the next stage. Since the instructions state that players should have at least six cards, which prevents any problems from occurring, most of the time the students played safe and did not experience the problems. If this number was not stated, the lesson of having a larger amount of *analysis and design* cards before progressing to the next stage would have been more obvious, and one of the game's main ideas would have been driven across much more when someone failed miserably.
11. One of the problem areas is that each player can accumulate an infinite amount of *concept* cards (most have no associated cost) and can seriously imbalance a player. Thus with a finite amount of cards in a deck, the goal would be to gain a well-balanced hand. If a player obtains too many *problems* to hand to opponents, then he or she will have less *concept* cards. If a player has too many *concept* cards, he or she will not be able to slow down the opposition with problems. This balance is important in keeping with competition. As a solution, rather than playing only the *programmers* and *concept* cards, each player should be dealt a finite number of cards from the main deck that consists of *programmers*, *concepts* and *problems*, which are never shown to the opposition until played. During the requirements and design stages these cards are not used much, but during the implementation stage, the player has the option of trading in some cards, or using them.
12. A maximum time allowed for complete project production could be imposed.

Overall, the game is interesting and presents a number of real-world factors that could impact a software project. Many aspects of the development process are incorporated into the game, including requirements and design documentation, budget restrictions, personality, skill, problems of all sorts, code bugs, and project complexity, length and quality. It is a painless lesson for players who may be tempted to skip requirements documentation or who rush through the coding phase. The game does a really good job of highlighting some of the realities of the software development process.

Student players were Becky Hodge, Yasser Ebrahim, Michael Kaastra, Philip Liew, and Usman Shakil.