Felipe Nunes Gaia

Proceedings of 25th International Conference on Software Engineering (2003).

Problems and Programmers: An Educational Software Engineering Card Game

Alex Baker

Emily Oh Navarro

André van der Hoek

Summary

1. Introduction
2. Objectives
3. Overall Design
4. Game Play
5. Experiment
6. Conclusions

1. Introduction

- Problem is simple: class projects must be completed within the length of their course.
  - little room is left to **illustrate the many facets** of the **software process**.

- **Industry has recognized this problem**.
- Over time has requested academia to **better prepare** its students for their future in the **workplace**.

1. Introduction

- Problems and Programers is built as a physical card game.
  - has the advantage of being **easy** and open to use,
  - a **competitive** game,
  - interactive games ensue in which students **learn** from **each other**.

2. Objectives

- **Enrich a student's understanding of the software process.**

- The choices were guided by high level objectives:
  - *"Should advocate proper use of software engineering techniques."*
  - *"Should illustrate both general and specific lessons concerning the software process."*
  - *"Should provide a student with clear feedback concerning their decisions."*
  - *"Game play should be easy and comparatively quick."*
  - *"Should encourage interaction among students."*

3. Overall Design

- A **multiplayer** game;
- Two or more players each attempt to be the **first to complete** a hypothetical software project.

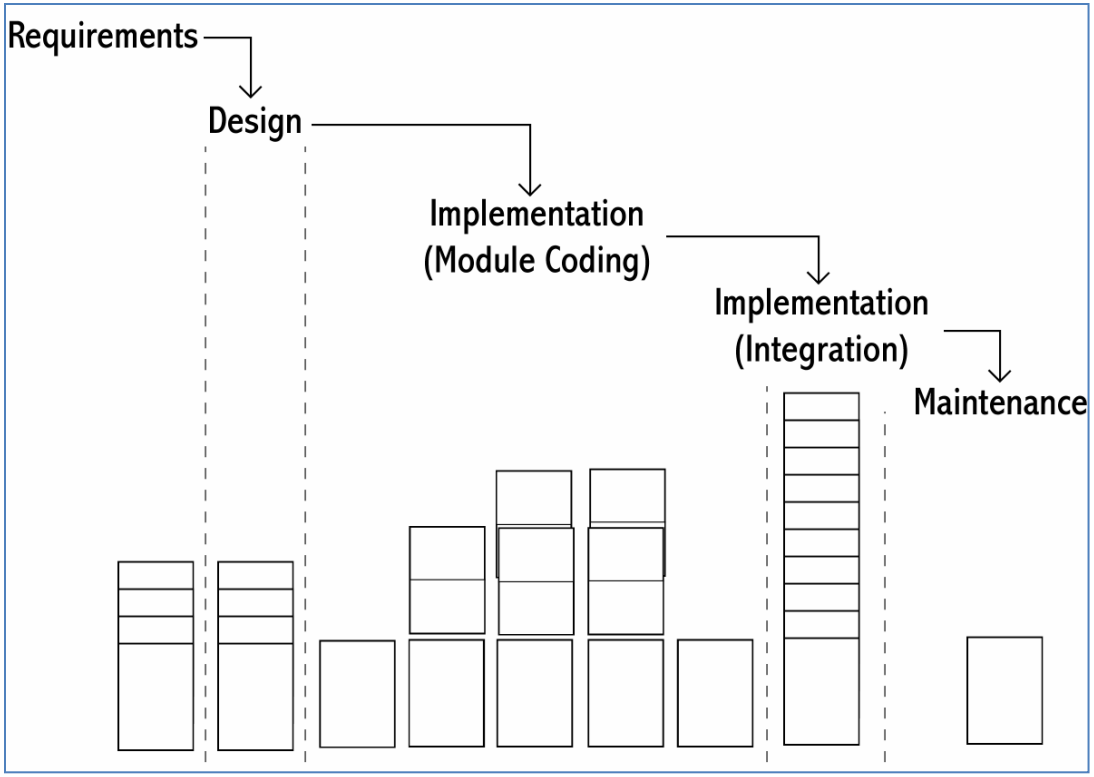
- Competitive nature encourages interaction.
  - Different players follow different strategies.
  - More than one strategy is exposed per game.

3. Overall Design

Follows the steps of the waterfall model.

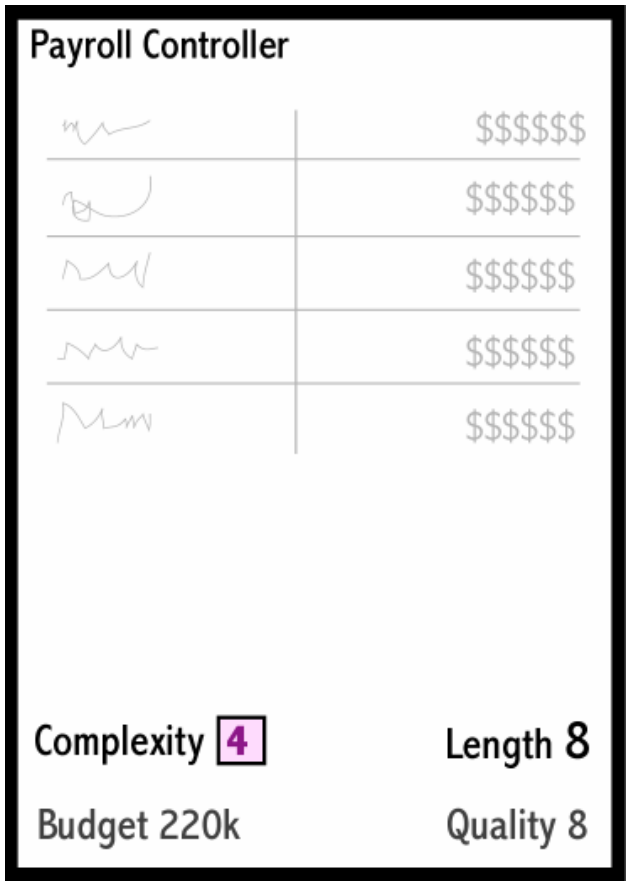
They considered incorporating other life cycle models, but:

- Physically visualizing: creates **non-uniform card layouts**.
- Establish different rules for different life cycles: violates the objective **play being easy**.



3. Overall Design

- The project is determined at the beginning of the game.
- A single project card is drawn and placed in the middle amongst all players.



- Complexity influences the **progress** of each player during the **implementation phase**.

- Length defines the size of the application (number of code cards that **must be integrated**).

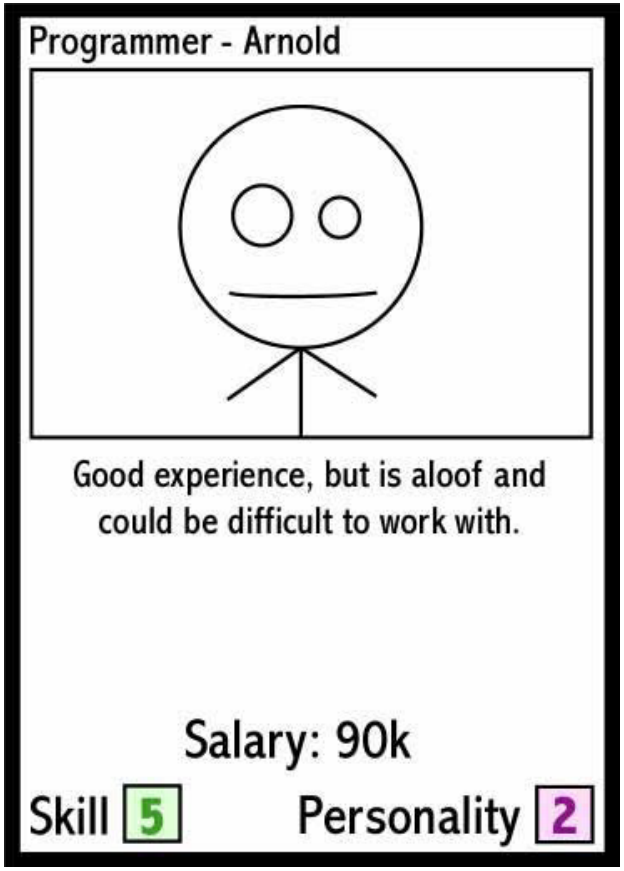
- Quality defines the **number of code cards** that will be **inspected for bugs**.

3. Overall Design

- Used during the **implementation phase**.
- Each programmer is defined by a salary, skill, personality, and brief description.

- The salary defines the cost to the overall project:

  The sum of all programmer salaries may not exceed the project budget.

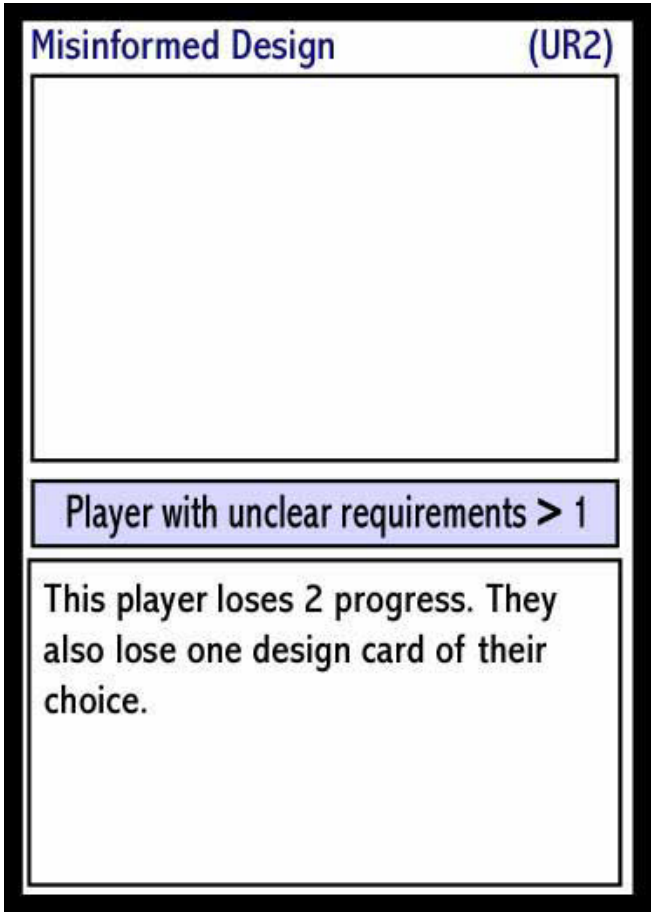


3. Overall Design

- **Problem cards** are at the **heart of game play**.
- Create many of the **uncertainties** that make winning the game difficult.

- Are drawn by one player and played upon another player.

- If other player **matches the condition** on the card, he is **penalized**.



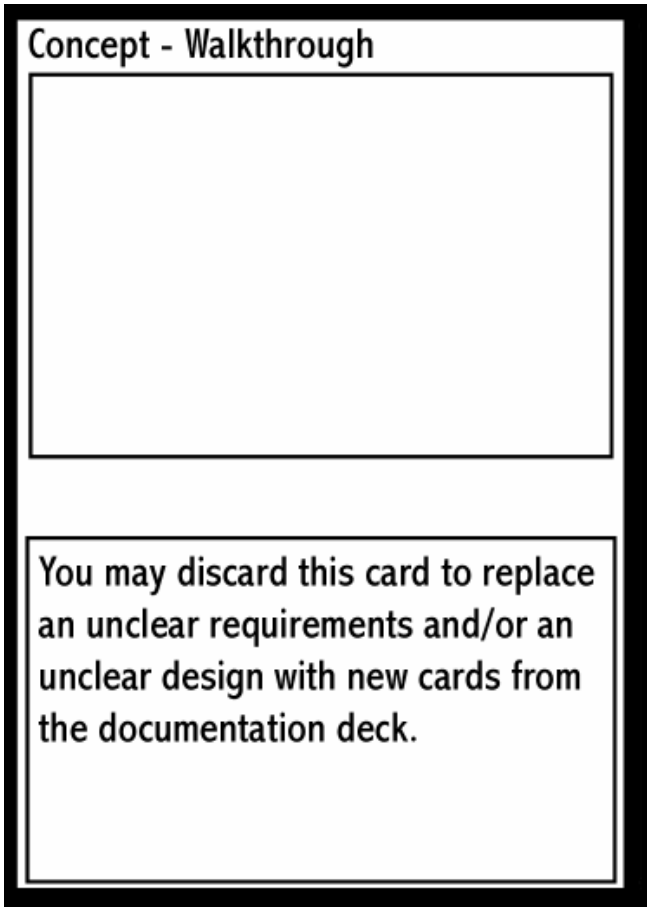
3. Overall Design

- **Concept cards** play the **opposite** role of **problem cards**: represent benefits to a project.

- This example helps to **reduce unclear requirements or design**.

- A few concepts will incur a **cost**.

Card that increases the motivation of each programmer with a bonus, resulting in higher productivity.



4. Game Play

1. Decide to move to the next phase of the life cycle, the previous phase, or remain in the current phase.
2. Allow each opponent the chance to play one problem card on them.
3. Draw cards.
4. Take actions, as allowed in the respective phase.
5. Play any programmer and concept cards.
6. Discard any extra cards.

I. Requirements

- Over a number of turns, a player must create a single column of requirements cards.
  - the length **simulates** the **size of the requirements** document.
  - decide when is complete **based on the pr**oject.

- Requirements cards are either clear or unclear.

II. Design

- Players create a separate stack for the design document.
- And at each turn can choose
  - add new cards,
  - rework existing cards,
  - or mix the two activities.

III. Implementation

- Per turn, each programmer hired can take four different actions:
  - program good code
  (cost equal to project complexity)
  - program rush code
  (cost equal to project complexity divided by two)
  - inspect code (cost one)
  - fix a bug (cost one or more).

IV. Integration testing

- Take all of the code developed by a particular programmer and move it into the integrated code column.

- Buggy code after inspection and code not yet inspected can be integrated, but:
  - Exposes the player to the risks described in maintenance.

- Code is integrated programmer by programmer:
  - Tradeoffs between having more programmers **(less time to build)**, and fewer programmers **(quicker integration)** .

V. Maintenance / acceptance testing

- Decide whether to continue (a) integrating code or (b) attempt to win the game.

- In (b) case, all integrated code is placed face down, shuffled, and a number of cards equaling the project quality are pulled from the top of the pile.
  - If **no bugs appear**, the player has **won the game**.
  - If **any bugs appear**, however, the **penalty is severe**: all code cards must be placed above a single programmer.

- Play continues: the bugs must be fixed, additional code can be inspected, and the player is once again vulnerable to problem cards.

5. Experiment

- **28** undergraduate students played the game.
- Teamed in pairs, each student was:
  1. Introduced to the game, its objectives, and its mechanics of game play **(30 min.)**
  2. Played twice against the other person

     (45 min. / game)
  1. Asked to fill out a questionnaire regarding their particular experiences **(10 min.)**

5. Experiment - Scores

• All students had passed the **same software engineering course** as background material (ICS 52, **Introduction to Software Engineering**).

• So in **different quarters** with **different instructors**.

Question	***1***	***2***	***3***	***4***	***5***
1. How enjoyable is it to play?			6	13	9
2. How difficult/easy is it to play?		3	10	12	3
3. How well does it reinforce knowledge?		6	9	6	6
4. How well does it teach new knowledge?	7	8	6	3	4
5. How well does it teach the software process?	1	4	8	12	3
6. Incorporate it as standard part of SE course?	1	6	3	12	6
7. As an optional part?	1	5	4	10	8
8. As a mandatory part?	1	6	8	11	2

Overall, the results are very encouraging.

5. Experiment - Opened Questions

- *"I learn that even though doing requirements and design take a lot of time, but if we don't do well on these parts of the project might actually take longer to finish!"*

- *"Requirements and design are boring."*
- *"It was unclear the amount of time should be spent on requirements and design."*

6. Conclusions

- The game represents a first attempt at using a physical card game for educating students.

- The card-based simulation has the advantage of:
  - being highly visual,
  - being easy and fun to play,
  - engaging students in collaborative learning,
  - providing almost instantaneous feedback on the actions.

6. Conclusions

- The results of our experiments confirm our intuitions:
  - Students **largely believe** that use of the game in their **software engineering class** would have **helped them** in their studies.

- Planned **collaborations** with **3 other institutions**:
  - Provide a broad sample size in this process.

- Second, we will develop an **automated version** of the game.

Questions?