A Constructivist Approach to Teaching Software Process

Jayakanth Srinivasan, Kristina Lundqvist
Massachusetts Institute of Technology
May 23rd, 2007
Overview

- Why did we create the game?
  - Needed a more effective way of teaching software process models to undergraduate aerospace engineering students
  - Reflects our evolving research and teaching philosophy

- How we played the game
  - Context, Roles, Structure and Execution

- Lessons Learned
Teaching Software Process

- Lecture-based teaching of software processes is “dry”
  - “I have also found students glaze over on these topics” – Anonymous Reviewer

- Learning often **NOT** anchored in long term memory
  - Performance in concept questions compared to the final exam
Constructivism-based Game Design

- Seven Values
  - Collaboration
  - Active Engagement
  - Personal Relevance
  - Pluralism
- Multiple Stakeholders
- Personal Autonomy
- Generativity
- Reflexivity

Software Development as Problem Solving
Game Design Philosophy

- Leveraged the 8 instructional principles created by Savery and Duffy

- Today’s discussions focused on
  - Design an authentic task
  - Give learner ownership of the process used to develop a solution
  - Encourage the testing of ideas against alternative views and alternative contexts
Congratulations! Your team has been selected to upgrade the software for the glass cockpit of the Next_Generation-7 helicopter. The avionics system architecture for the helicopter exactly mirrors that of the S-70, as seen in the architecture diagram. A preliminary set of requirements have been generated by the technology feasibility study team, and will act as the starting point for your work. Given that our flight testing schedule has been moved forward, your delivery dates for software components have been moved up as well. We have provided additional funding to the program to be dispensed at the discretion of the program manager.
Game Roles

- **Facilitator**
  - Set up game
  - Active listening

- **Implementation**
  - Eliminate all incorrect requirements

- **Requirements**
  - Create a work package for downstream sub-teams

- **Design**
  - Refine work package

- **Integration**
  - Eliminate all ambiguous requirements

- **Reserve**
  - Manage reserve funds
Game Set up

- Provide each team with instructions, stickers, set of timers and dice

- Create preliminary requirements set
  - Three incorrect requirements
  - Six ambiguous requirements
  - Six correct requirements

### Requirement Sheet

<table>
<thead>
<tr>
<th>Category</th>
<th>Type</th>
<th>Weight</th>
<th>Requirement</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Incorrect</td>
<td>A</td>
<td>-10</td>
<td>1. The displays interface directly to the Arinc 629</td>
<td></td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>1</td>
<td>2.1. FLIR does not connect to the HUD</td>
<td></td>
</tr>
</tbody>
</table>

- Place Red/Yellow/Blue does in the box for the phase
  - Design
  - Implementation
  - Integration
Exemplar Instruction

- Integration
  - Ensure there are no ambiguous requirements that have been implemented.

<table>
<thead>
<tr>
<th>Type</th>
<th>Requirement</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>FLIR connects a mission bus</td>
<td>7</td>
</tr>
</tbody>
</table>

If Total_Blue = 0, PROJECT SUCCESSFUL
Set Dice_Total to zero

Loop Total_Blue Times,
1. Throw a pair of dice
2. Add the dice value to Dice_Total

If Dice_Total < 6, Set Total_Blue to zero, and restart

If 6 ≤ Dice_Total ≤ 12, you have two options
Spend 10 units, Reduce Total_Blue by 1, restart

OR
Start the black timer, when it expires,
Reduce Total_Blue by 1 restart

If 12 < Dice_Total, Spend 10 units and Start the black timer, restart
Do people just sit around drinking free coffee and donuts, and doing nothing while they wait?

- All the teams that failed blamed the upstream process as a key reason for failure.
- Stories
  - Requirements team did nothing
  - Integration team could not integrate anything
Second Round: Integrated Teaming

- Project Team Brainstorm
  Time: 10 min

- Setup
  Time: 5 Min

- Reserve
  Budget: 30

- Retrospective
  Time: 15 Min

- Requirements and Design
  Time: 8 Min
  Budget: 30

- Implementation and Integration
  Time: 6 Min
  Budget: 40

- Questions asked by the students
  - Can we all work together on both sub-teams so that we can maximize resource usage?
  - Can we start development and integrating activities earlier than the current handoff process?
Third Round: Value Stream

Project Team Brainstorm
Time: 10 min

Setup
Time: 5 Min

Single Team
Time: 15 Min
Budget: 70

Retrospective
Time: 15 Min

- “We now know how to work together, now we want to see if we can do it as a complete group”

- Extremely flexible round based on progress of students
  - Seeded wrong requirements
  - Change the customer between rounds
Conclusions

● The Game
  ● Created around constructivist values
  ● Enables students to discover the strengths and weaknesses of software process models for themselves
  ● Enables instructors to anchor learning in long term memory through concrete examples

● Future Work
  ● Need to develop more detailed evidence on the effectiveness of the game
  ● Develop agent based models that encapsulate decision rules captured through case studies
  ● Support hypothesis testing and scenario analysis for process improvement
Acknowledgements

• Thanks to …
  • 16.35 Real-Time Systems and Software
  • 16.01 Unified Engineering
  • Funding support from LAI

• Game available via email
  • jksrini@mit.edu  Jayakanth “JK” Srinivasan
  • kristina@mit.edu  Kristina Lundqvist
BACKUP
### Example Rules

<table>
<thead>
<tr>
<th>Role</th>
<th>Responsibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements</td>
<td>Create a work package where the total weight of requirements is $\geq 50$</td>
</tr>
<tr>
<td>Design</td>
<td>Create a design that is in the worst case based on one incorrect requirement, and two ambiguous requirements</td>
</tr>
<tr>
<td>Implementation</td>
<td>Create an implementation that is in the worst case based on three ambiguous requirements</td>
</tr>
<tr>
<td>Integration</td>
<td>Successfully integrate the system, ensuring that the implementation has no ambiguous requirements</td>
</tr>
<tr>
<td>Revenue</td>
<td>Manage the funding for the project</td>
</tr>
<tr>
<td>Coordinator</td>
<td>Ensure that the game is being played by the rules, and observe team dynamics during the game, may also act as customer in later iterations</td>
</tr>
</tbody>
</table>

### Rule for Integration Phase

If Total_Blue = 0, PROJECT SUCCESSFUL  
Set Dice_Total to zero

Loop Total_Blue Times,  
1. Throw a pair of dice  
2. Add the dice value to Dice_Total

If Dice_Total < 6, Set **Total_Blue to zero**, and **restart**

If 6$\leq$Dice_Total$\leq$ 12, you have two options  
Spend 10 units, Reduce Total_Blue by 1, **restart**  
OR  
Start the black timer, when it expires, Reduce Total_Blue by 1 **restart**

If 12 < Dice_Total, **Spend 10 units** and **Start** the black timer, **restart**