LOS JUEGOS DE CLASE NO TECNOLÓGICOS COMO UNA ESTRATEGIA DIDÁCTICA PARA LA ENSEÑANZA DE LA INGENIERÍA DE *SOFTWARE*

Por:

CARLOS MARIO ZAPATA JARAMILLO
M.Sc. en Ingeniería de Sistemas
Ph.D. en Ingeniería-Sistemas

FACULTAD DE MINAS
UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE MEDELLÍN
2007

LOS JUEGOS DE CLASE NO TECNOLÓGICOS COMO UNA ESTRATEGIA DIDÁCTICA PARA LA ENSEÑANZA DE LA INGENIERÍA DE *SOFTWARE*

Por:

CARLOS MARIO ZAPATA JARAMILLO
M.Sc. en Ingeniería de Sistemas
Ph.D. en Ingeniería-Sistemas

Trabajo de promoción presentado como requisito parcial
para la promoción a la Categoría de Profesor Asociado

FACULTAD DE MINAS
UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE MEDELLÍN
2007

DEDICATORIA

A Vicky, Sebas y Pipe, mis más grandes maestros
en la tarea más dura de mi vida:
tratar de ser una mejor persona cada día.

AGRADECIMIENTOS

Un trabajo como el que ahora presento posee un sinnúmero de personas que contribuyen con su apoyo implícito o explícito en su realización. Desde los participantes en los diferentes juegos, que con paciencia me han permitido experimentar con ellos esta forma de enseñanza, pasando por los profesores que me han acompañado en algunas etapas de este camino facilitándome sesiones en sus clases, revisando con antelación los juegos o simplemente discutiendo en los pasillos o en sus oficinas conmigo los pormenores de algunos de estos juegos, hasta las personas que me han permitido llevar esta experiencia a una realidad concreta. A todos les agradezco infinitamente y creo que no tendría espacio suficiente para mencionarlos a todos.

Un agradecimiento especial para los profesores María Teresa Berdugo, Fernando Arango y mi gran amigo Santiago Montoya (Q. E. P. D.) de la Escuela de Sistemas, quienes siempre han creído que este camino puede conducir a algún destino; a la profesora Amparo Saldarriaga, por suministrarme un espacio en “el Jardín de la Ciencia” para demostrar que los niños pueden aprender con este tipo de experiencias; a los profesores Gabriel Awad y Elkin Rodríguez, de la Escuela de Ingeniería de la organización, por compartir sus experiencias conmigo; a los profesores Alfonso Medina y Juan Carreón de la Universidad Nacional Autónoma de México, por permitirme compartir esta experiencia con algunos de los profesores y alumnos de ese importante centro académico; y muy especialmente a mis “niños” Mary, Natalia, Yudi y Hernán, por diseñar conmigo algunos de los juegos.

TABLA DE CONTENIDO

DESCRIPCIÓN	PÁG.
Índice de figuras	v
Índice de tablas	vi
Resumen	viii
Abstract	ix
Capítulo 1: Introducción	1
Capítulo 2: Una revisión crítica de la enseñanza de la Ingeniería de *Software*	3
Capítulo 3: Planteamiento del problema	9
Capítulo 4: Marco teórico	13
4.1. Algunas definiciones importantes	13
4.2. Los juegos como estrategia de aprendizaje	13
4.3. Algunas corrientes pedagógicas y su relación con los juegos planteados	17
4.4. Algunos aspectos teóricos de la Ingeniería de *Software* y la gestión de proyectos	20
Capítulo 5: Juegos desarrollados para la enseñanza de la Ingeniería de *Software*	26
5.1. El juego del desarrollo de *software*	26
5.1.1. Objetivo del juego	26
5.1.2. Materiales	26
5.1.3. Reglas de juego	29
5.2. El juego de UN-Método	30
5.2.1. Objetivo del juego	30
5.2.2. Materiales	31
5.2.3. Reglas de juego	32
5.3. El juego de la consistencia	35
5.3.1. Objetivo del juego	35

DESCRIPCIÓN PÁG.

5.3.2. Materiales 35
5.3.3. Reglas de juego 38
5.4. El Juego de la planeación de proyectos de *software* 41
5.4.1. Objetivo del juego 41
5.4.2. Materiales 41
5.4.3. Reglas de juego 43
5.5. El juego de la trazabilidad y la comunicación 45
5.5.1. Objetivo del juego 45
5.5.2. Materiales 46
5.5.3. Reglas de juego 47

Capítulo 6: Resultados de la aplicación de los juegos 51

6.1. El juego del desarrollo de *software* 51
6.2. El juego de UN-Método 56
6.3. El juego de la consistencia 62
6.4. El juego de la planeación de proyectos de *software* 70
6.5. El juego de la trazabilidad y la comunicación 74

Capítulo 7: Conclusiones y trabajo futuro 76

Referencias 82

Anexo: La visión particular de un participante en los juegos 89

ÍNDICE DE FIGURAS

No.	DESCRIPCION	PÁG.
1	Modelo en cascada. Tomado de Zapata y Gómez (2006)	21
2	Hoja de especificaciones del juego del desarrollo de *software*	27
3	Hoja de materia prima para los módulos del juego del desarrollo de *software*	28
4	Tablero del juego de UN-Método	31
5	Programa de simulación de dados del juego de UN-Método	32
6	Una posición específica en el juego de UN-Método	34
7	Plantilla de esquema preconceptual del juego de la consistencia	36
8	Plantilla del diagrama de clases del juego de la consistencia	36
9	Plantilla del diagrama de casos de uso del juego de la consistencia	37
10	Plantilla del diagrama de secuencias del juego de la consistencia	37
11	Lista de palabras del juego de la consistencia	38
12	Soluciones a la elaboración de diagramas del juego de la consistencia	40
13	Plantilla para la elaboración de la planeación temporal	42
14	Cuadro de seguimiento a las planeaciones temporales	42
15	Programa de simulación de dados para el juego de la planeación	43
16	Una posición específica en el juego de la planeación de proyectos de *software*	45
17	Tablero del juego de la trazabilidad y la comunicación	46
18	Artefacto caso de uso general para el juego de la trazabilidad y la comunicación	47
19	Artefacto caso de uso *organizar* para el juego de la trazabilidad y la comunicación	47
20	Artefacto caso de uso *atacar* para el juego de la trazabilidad y la comunicación	48
21	Artefacto diagrama de clases para el juego de la trazabilidad y la comunicación	48
22	Artefacto interfaz gráfica de usuario para el juego de la trazabilidad y la comunicación	49
23	Distribución de la zona de juego	50

ÍNDICE DE TABLAS

No.	DESCRIPCION	PÁG.
1	Hoja de control del juego del desarrollo de *software*	28
2	Asignación de puntajes por la elaboración de diagramas del juego de la consistencia	39
3	Estimación de la duración en horas-hombre de las actividades de tres proyectos de *software*	41
4	Características de los grupos de personas en los cuales se ha aplicado el juego del desarrollo de *software*	51
5	Resumen de respuestas a la pregunta “¿Qué aprendió del juego?”	53
6	Resumen de respuestas a la pregunta “¿Qué cree que se necesitaba para ganar el juego?”	55
7	Resumen de respuestas a la pregunta “¿Qué modificaciones le haría al juego?”	56
8	Características de los grupos de personas en los cuales se ha aplicado el juego de UN-Método	57
9	Resumen de respuestas a la pregunta “¿Qué aprendió del juego?”	58
10	Resumen de respuestas a la pregunta “¿Qué cree que se necesitaba para ganar el juego?”	60
11	Resumen de respuestas a la pregunta “¿Qué modificaciones le haría al juego?”	60
12	Características de los grupos de personas en los cuales se ha aplicado el juego de la consistencia	62
13	Cambios en los porcentajes de opinión acerca de las preguntas fundamentales de la consistencia	65
14	Resumen de opiniones de los participantes en relación con la dificultad de manejo de la consistencia entre diagramas antes y después del juego	67
15	Resumen de justificaciones de los participantes a la pregunta “¿Fue este juego una experiencia de aprendizaje que valió la pena?”	68
16	Resumen de respuestas a la pregunta “¿Qué aprendió del juego?”	69
17	Resumen de respuestas a la pregunta “¿Qué modificaciones le haría al juego?”	69
18	Características de los grupos de personas en los cuales se ha aplicado el juego de la planeación de proyectos de *software*	70
19	Resumen de respuestas a la pregunta “¿Qué aprendió del juego?”	71

No.	DESCRIPCION	PÁG.
20	Resumen de respuestas a la pregunta “¿Qué cree que se necesitaba para ganar el juego?”	73
21	Resumen de respuestas a la pregunta “¿Qué modificaciones le haría al juego?”	73
22	Características de los grupos de personas en los cuales se ha aplicado el juego de la trazabilidad y la comunicación	74
23	Resumen de respuestas a la solicitud “Mencione los tres aspectos que considere fundamentales en la elicitación de requisitos”	75

RESUMEN

La Ingeniería de *Software* es una profesión especial, pues sus practicantes requieren conocimientos y habilidades ligadas simultáneamente con muchas áreas del conocimiento; en especial, se requieren habilidades en comunicación y administración de proyectos. La enseñanza de la Ingeniería de *Software* se ha realizado tradicionalmente con una combinación de clases expositivas y pequeños proyectos prácticos; también, se han usado estrategias como los estudios de casos y los juegos de computador o video con fines de instrucción. Sin embargo, las necesidades de la industria del *software* plantean nuevos retos educativos que requieren también una combinación de nuevas estrategias complementarias a la educación tradicional. En este trabajo se presentan cinco juegos no tecnológicos que se pueden aplicar en el aula de clase para la enseñanza de diferentes aspectos de la Ingeniería de *Software*. Se presentan, además, los resultados de la aplicación de estos juegos a varios grupos de estudiantes y profesores de diferentes niveles.

Palabras Clave: Ingeniería de *Software*, estrategias educativas, aprendizaje basado en juegos, juegos no tecnológicos.

ABSTRACT

Software Engineering is a special profession, and its practitioners need knowledge and capabilities simultaneously linked to many knowledge areas (for example, they require communication and project management skills). Software Engineering teaching has been traditionally made with a mix of lectures and “toy” practical projects; they also have been used strategies like Case Study and instructional video/computer games. However, the needs of software industry pose new educational challenges; these challenges require, again, a mix of new complementary strategies to traditional education. We present in this work five non-technological in-class games for teaching various aspects of Software Engineering. Also, we present the results of the games’ application to several groups of students and teachers from different levels.

Keywords: Software Engineering, educational strategies, game-based learning, non-technological games.

CAPÍTULO 1: INTRODUCCIÓN

Desde sus primeros pasos a finales de los años sesenta, la Ingeniería de *Software* ha venido despertando un creciente interés en la industria del *software*, dado que los problemas ligados con el desarrollo de estos productos intangibles se han vuelto cada vez más complejos; esta complejidad se debe a la gran demanda de este tipo de productos y al alto impacto que pueden alcanzar en el medio. En efecto, ya es casi impensable la vida cotidiana sin la participación de las diferentes aplicaciones de *software* empleadas a diario, desde aquéllas basadas en Internet hasta *software* bancario o de atención hospitalaria, pasando por *software* comercial y de oficina.

Si bien la Ingeniería de *Software* ha evolucionado, su enseñanza ha presentado pocas modificaciones desde sus inicios: se ha empleado para la formación de los ingenieros de *software* una combinación de clases expositivas y pequeños proyectos de "juguete", con los cuales se ha pretendido que los futuros ingenieros de *software* adquieran las destrezas necesarias para su ejercicio profesional. Este tipo de enseñanza, ampliamente centrada en el profesor como fuente del conocimiento para las clases expositivas y como asesor o guía para los proyectos prácticos, viene siendo revaluado debido a los problemas de participación e iniciativa que puede generar en los estudiantes, y estas dos características son ampliamente deseables para los futuros egresados de esta profesión.

Los cambios han sido sutiles a través de los años. Por ejemplo, la manera en que se desarrollan los proyectos prácticos es lo que más ha variado en la formación en Ingeniería de *Software*, tratando de buscar nuevas formas de motivación para los estudiantes o nuevos retos del ejercicio profesional que sirvan como simulación en dichos proyectos. Para ello, se han comenzado a emplear juegos de video o computador, pues, se reconoce la poderosa influencia que ejerce este tipo de elementos en los estudiantes de áreas tecnológicas. También, se han empleado de forma aún incipiente estrategias provenientes de varias disciplinas (la medicina, el derecho o la administración), como los denominados estudios de casos, que han probado ser exitosos para simular los comportamientos de los futuros profesionales de un área discutiendo hechos pasados o ficticios, preparados específicamente para ello.

Sin embargo, estas estrategias aún poseen problemas: en el caso particular de la Universidad Nacional de Colombia, no se cuenta con recursos de *hardware* y *software*, además de recursos humanos (número de docentes) para emplear los juegos de video o computador como estrategia

regular en las aulas. Además, hay una creciente demanda por formación en Ingeniería de *Software* (en particular) y en Ingeniería de Sistemas e Informática (en general), que agrava el problema de los recursos y dificulta el uso de nuevas estrategias como las que se mencionan.

Como una forma de solución a estos problemas, en este trabajo de promoción se presentan cinco juegos no tecnológicos (no son juegos de video o computador, sino juegos que se pueden realizar en un aula regular) que posibilitan la enseñanza de ciertos aspectos de la Ingeniería de *Software*. Estos juegos son: el juego del desarrollo de *software*, el juego de UN-Método, el juego de la consistencia, el juego de la planeación de proyectos de *software* y el juego de la trazabilidad y la comunicación.

La estructura de este trabajo de promoción es la siguiente: en el capítulo 2, se realiza una revisión crítica de los métodos que se han empleado en la enseñanza de la Ingeniería de *Software*; en el capítulo 3, se realiza el planteamiento detallado del problema que es objeto de este trabajo; en el capítulo 4, se discute la importancia de los juegos como una estrategia de enseñanza-aprendizaje; en el capítulo 5, se presentan los cinco juegos con sus principales características; en el capítulo 6, se discuten los principales resultados de la aplicación de estos cinco juegos a diferentes grupos de estudiantes y profesores. Finalmente, en el capítulo 7, se presentan las conclusiones y trabajo futuro que se pueden derivar de los juegos no tecnológicos como estrategia de enseñanza-aprendizaje.

A modo de anexo, se presenta una reflexión del ingeniero Andrés Felipe Muñetón en torno a algunos de los juegos que se presentan en este trabajo, y que se realizó como trabajo propuesto en el curso *tópicos avanzados en Ingeniería de Software*. De este anexo, se puede extraer la visión de un participante en los juegos, sus recomendaciones de mejoramiento y las enseñanzas que él pudo recopilar de esta experiencia.

CAPÍTULO 2: UNA REVISIÓN CRÍTICA DE LA ENSEÑANZA DE LA INGENIERÍA DE *SOFTWARE*

Fairley (1978 y 1986) señala una de las características que hacen de la Ingeniería de *Software* una profesión especial: la extraña mezcla que se presenta en su cuerpo de conocimientos entre ingeniería (a nivel general) y ciencias de la computación (de manera más específica), con metodología del *software* y habilidades de comunicación y administración.

El ingeniero de *software* debe poseer también una mezcla de habilidades y conocimientos difíciles de conseguir, como una amalgama de diferentes disciplinas que puede hacer la diferencia entre el éxito o el fracaso de un proyecto de *software* que tenga bajo su mando. Efectivamente, el ingeniero de *software*, en ocasiones, deberá ser muy gerencial, para utilizar adecuadamente los recursos a su cargo; en otras ocasiones, deberá ser muy técnico y tener amplios conocimientos de métodos de desarrollo, para permitir que las aplicaciones de *software* que se desarrollen puedan tener las características de calidad requeridas (tales como *usability*, la capacidad de uso, *maintainibility*, la capacidad para ser mantenidas, y *portability*, la capacidad para adaptarse a cambios en los requisitos o el entorno); puede que, en otras ocasiones, deba actuar como una especie de médico, para diagnosticar los problemas que aquejan a una organización, y, posteriormente, sugerir la mejor manera de solucionarlos mediante una aplicación informática; es posible que, también, deba actuar como "sabelotodo" para poder capturar la información que la aplicación informática requiere y poderla analizar y transformar en los modelos necesarios para su comprensión. Bagert *et al.* (1999) señalan como habilidades principales del ingeniero de *software* la capacidad de elaborar programas, la habilidad para adecuar los requisitos del *software* con las necesidades planteadas por los interesados y la capacidad de manejar proyectos de *software*. Nuseibeh y Easterbrook (2000) reconocen la relación estrecha entre la Ingeniería de Requisitos (una parte de la Ingeniería de *Software*) y otras ciencias como la psicología cognitiva, la antropología, la sociología y la lingüística; con estas ciencias, se procura comprender las actividades humanas desde diferentes puntos de vista, y, con la Ingeniería de *Software*, se procura aplicar esa comprensión en el desarrollo de *software*.

Estas características requeridas por los ingenieros de *software* no están aisladas del contexto en que se desenvuelven. La industria, que es quien los contrata, exige cada vez más que estas características estén latentes en cada nuevo egresado de los centros académicos; de esta forma, es la

industria la que condiciona el tipo de formación que deberían recibir los estudiantes de Ingeniería de *Software*. En este contexto, las industrias, según Jaccheri y Morasca (2006), pueden desempeñar alguno de los siguientes roles en relación con la academia:

- Estudiantes: los ingenieros vinculados con la industria requieren cursos de actualización y otros tipos de entrenamiento.
- Fuentes de cursos: las necesidades de conocimiento de la industria condicionan los cursos de extensión que se pueden ofrecer en las academias.
- Profesores: los ingenieros vinculados que son invitados a la academia para compartir su experiencia profesional con los estudiantes.
- Investigadores: los resultados de las investigaciones realizadas en la academia pueden ser potencialmente útiles para la industria, que, por lo general, participa en su financiación.
- Clientes: los proyectos prácticos de las clases se pueden enfocar a solucionar problemas de la industria.

Carrington *et al.* (2005) presentan uno de los casos en que la industria y la academia trabajan mancomunadamente en la definición de sus necesidades de capacitación, pero éste es tan sólo un caso aislado de cooperación que poco se presenta. Busenberg y Tam (1979) presentan otra manera en que la industria colabora con la academia, mediante el suministro de casos prácticos en los cuales los futuros ingenieros de *software* pueden practicar sus conocimientos; esta situación también se repite poco en la Universidad Nacional de Colombia, puesto que, además, existen problemas de propiedad intelectual en relación con los productos de *software* elaborados por los estudiantes en sus prácticas.

Pero, no sólo la industria condiciona el conocimiento que se debería transmitir a los ingenieros de *software*. También, la academia ha realizado esfuerzos para definir cuál debería ser el cuerpo de conocimientos necesario para los ingenieros de *software*. En principio, la Ingeniería de *Software* se consideraba una de las disciplinas incluidas en el currículo de las Ciencias de la Computación, como consta en el currículo conjunto emitido por la ACM y la IEEE (2001) para esta profesión. Sin embargo, en algunos trabajos se aducen las razones por las cuales el currículo de la Ingeniería de *Software* debería estar desligado de las Ciencias de la Computación y de la Tecnología de Información (véase Mitchell, 2004).

Desde los trabajos iniciales de Fairley (1978), donde se discuten los aspectos curriculares de la Ingeniería de *Software*, incluyendo la manera como se deberían dictar los cursos en esta profesión, se han venido realizando propuestas que procuran definir los contenidos que requiere un ingeniero de *software* desde el punto de vista de la industria y la academia. Se resaltan, en este aspecto, los esfuerzos del *Software Engineering Institute* de *Carnegie Mellon University*, donde se exploró el currículo que debe hacer parte de la enseñanza de la Ingeniería de *Software*, particularizando en las habilidades que deben primar en el ingeniero de *software* (Bagert *et al.*, 1999). Este estudio lo retoman Budgen y Tomayko (2005) para evaluarlo en relación con la forma como se imparte la Ingeniería de *Software* en la actualidad. Kitchenham *et al.* (2005) realizan un trabajo similar al elaborar una encuesta sobre la forma como el currículo de Ingeniería de *Software* se adapta a las necesidades de la industria; a manera de conclusión, encontraron que la Ingeniería de *Software* es mucho más que matemática y que se requiere dar un mayor énfasis a aspectos de negocios y comunicación, aunque estos aspectos deberían ser comunes para todas las ingenierías. Cowling (2005) incluye también el modelamiento como un área fundamental dentro del currículo de la Ingeniería de *Software*.

En relación con el currículo y la forma como se imparte la educación en la Ingeniería de *Software*, Boehm (2006) presenta una interesante visión del pasado, presente y futuro de la Ingeniería de *Software*, e incluye una serie de recomendaciones para mantener actualizados los métodos de enseñanza de esta disciplina. Si se toma en cuenta este aspecto, poco ha cambiado la enseñanza de la Ingeniería de *Software* desde los trabajos iniciales planteados por Fairley (1978). Como lo señalan Baker *et al.* (2005), el curso típico de Ingeniería de *Software* consta, por lo general, de un conjunto de clases expositivas combinadas con un pequeño proyecto práctico (y de "juguete" como ellos mismos lo señalan); esta última estrategia ha sido también muy defendida por Stiller y LeBlanc (2002), lo cual no permite que se desarrollen muchas de las habilidades que se requieren para un buen ingeniero de *software* tal y como se presentaron anteriormente. Estas habilidades sólo se pueden enmarcar en algunas de las inteligencias múltiples presentadas por Gardner (1983), tales como la inteligencia lingüística y la lógico-matemática, pero no se pueden ubicar en otros tipos de inteligencia requeridos por los ingenieros de *software*, tales como la inteligencia interpersonal y la intrapersonal.

Wankat y Oreovicz (1993) propusieron un conjunto de estrategias didácticas para enseñanza de la Ingeniería en general, entre las que se incluyen las clases expositivas y los proyectos prácticos. Sin embargo, las demás estrategias por ellos propuestas poco se han aplicado en la enseñanza de la

Ingeniería de *Software*. Estos autores, además, afirman que la clase expositiva como estrategia didáctica requiere complementarse con otras estrategias para alcanzar objetivos cognitivos de más alto nivel, y en esto coinciden con Rugarcia *et al.* (2000), quienes sugieren la complementación de la enseñanza tradicional de la Ingeniería con nuevos métodos alternativos que ofrezcan buenas posibilidades de éxito. Según Wankat y Oreovicz (1993), los métodos alternativos suelen ser de cuatro tipos:

- Tecnológicos, como es el caso de los juegos de computador existentes o los talleres de elaboración de juegos de video o computador como proyectos de curso.
- No tecnológicos[1], entre los que se incluyen el estudio de casos, la realización de actividades lúdicas no ligadas con computadores y la modificación a métodos no tecnológicos existentes.
- Laboratorios, los cuales están sugeridos por la ACM y la IEEE (2001) para las Ciencias de la Computación, como los que se emplean en ciencias como la Química y la Física.
- Educación personalizada, como en el caso de la participación en investigación—sugerida por Boehm (2006) para los estudiantes de Ingeniería de *Software*—, la respuesta a preguntas específicas de los estudiantes y las tutorías.

De estos métodos, únicamente los no tecnológicos y la educación personalizada están dirigidos a desarrollar la inteligencia intrapersonal e interpersonal propuestas por Gardner (1983). Los métodos tecnológicos son los que más se han empleado para complementar la enseñanza tradicional en la Ingeniería de *Software*, en casos como los siguientes:

- Uso de juegos de video o computador existentes para enseñar diferentes aspectos de la Ingeniería de *Software* y las Ciencias de la Computación, tales como:
    - Toma de decisiones (incluyendo resolución de conflictos) y planeación estratégica (De Aguilera y Méndiz, 2003).
    - Patrones de diseño, empleando el denominado *juego de la vida* (Wick, 2004 y 2005); patrones de diseño y polimorfismo empleando el *juego de los conjuntos* (Hansen, 2004).

[1] En el contexto de este trabajo, se asumirá esta denominación para los métodos que se emplean en el aula de clase y que no requieren el uso masivo de computadores. Si bien, esta denominación no es del todo adecuada, ya que, si se mira que, rigurosamente, la tecnología se relaciona con todo aquello que el hombre ha creado para modificar su entorno natural, es la denominación más cercana del trabajo de Wankat y Oreovicz para los juegos que se plantean en este trabajo, y posibilita su comparación con estrategias similares de las que ellos plantearon en su trabajo.

- Fases de la Ingeniería de *Software* (Claypool y Claypool, 2005) y procesos de Ingeniería de *Software* empleando *SimSE* (Navarro y Van der Hoek, 2004).
- Algoritmos, gráficos, inteligencia artificial y procesamiento paralelo empleando *Reversi* (Valentine, 2005).
- Lenguajes de programación y programación orientada a objetos (Sanders, 2005; Kafai, 1996; Jiménez *et al.*, 2005; Külling y Henriksen, 2005; Goschnick y Balbo, 2005; Coleman *et al.*, 2005; Leutenegger, 2006).

- Elaboración de juegos de computador o video como proyectos prácticos en cursos de Ciencias de la Computación e Ingeniería de *Software* (Becker, 2001; Swartout y Van Lent, 2003; Schaefer y Warren, 2004).
- Entrenamiento en diferentes aspectos de la actividad profesional empleando juegos existentes de realidad virtual y animación 3D (Faust, 2002; Hailey, 2004; Parisi, 2002; Jiménez *et al.*, 2005; Bierre y Phelps, 2004).
- Uso de entornos de desarrollo para la generación de juegos, tales como:

  - Diseño centrado en el aprendiz (*Learner-centered Design*), que es una de las tendencias en *Human-Computer Interaction* (HCI) y en el cual el maestro se convierte en un guía o un entrenador que, simplemente, ayuda en la solución de dudas en relación con el artefacto construido (Soloway *et al.*, 1994).
  - Desarrollo de robots virtuales empleando *Java*, que se utiliza para enseñar Inteligencia Artificial (Hartness, 2004).
  - Desarrollo de *software* empleando Ingeniería de *Software* (Morell y Middleton, 2001).
  - Programación orientada a objetos (Külling y Henriksen, 2005; Coleman *et al.*, 2005; Leutenegger, 2006; Kelleher y Pausch, 2005; Bierre y Phelps, 2004).

Entre los métodos no tecnológicos empleados para complementar la enseñanza de la Ingeniería de *Software* se cuentan:

- Estudio de casos, una técnica empleada inicialmente en la formación de estudiantes de Negocios, Leyes y Medicina, principalmente en la Universidad de Harvard, y que, en el contexto de la Ingeniería de *Software* y las Ciencias de la Computación, se ha empleado en HCI, especialmente en capacidad de uso de las aplicaciones de *software* (Carroll y Rosson, 2005) y en diseño centrado en

el usuario o HCD (Seffah, 2003). También, se ha empleado este tipo de métodos en cursos de Ingeniería de *Software* a nivel general (Busenberg y Tam, 1979).

- Uso de actividades lúdicas no ligadas con elementos tecnológicos, como por ejemplo en el juego *problemas y programadores* (Baker *et al.*, 2005) y *el juego de los requisitos* (Zapata y Awad, 2005).
- Modificación a los proyectos prácticos con diferentes estrategias:

    - Revisión de los proyectos por parte de pares del mismo nivel (en este caso, estudiantes) para enseñar la importancia de las pruebas en el desarrollo de *software* por parte de personas independientes al desarrollo (Clark, 2004).
    - Concursos de programación y revisión automática de los proyectos para fomentar actividades de mejoramiento (Pardo y Delgado, 2005).
    - Uso de *Extreme Programming*, un método de desarrollo de *software*, para la realización de los proyectos (LeJeune, 2005; Hedin *et al.*, 2005).
    - Simulación del entorno real de una compañía de desarrollo de *software* empleando, además, proyectos de dos semestres académicos de duración (Way, 2005).
    - Planteamiento de un único proyecto colaborativo para todos los grupos de estudiantes, haciendo que el proyecto tenga características de gran escala y sin que el profesor invierta más tiempo en su revisión (Coppit y Haddox, 2005).
    - Planteamiento de proyectos basados en juegos de computador y de video, con el fin de mejorar la participación de los estudiantes incrementando su motivación (Sweedyk y Keller, 2005; Ladd, 2006; Holliday, 1995; Hill *et al.*, 2003).

CAPÍTULO 3: PLANTEAMIENTO DEL PROBLEMA

Desde los inicios de la Ingeniería de *Software*, su enseñanza se ha acometido por lo general con dos estrategias principales: las clases expositivas y la elaboración de proyectos prácticos en el área, generalmente en pequeña escala, y que sirven como pequeños laboratorios en ambientes excesivamente simulados para enseñar el desarrollo metodológico de *software*. Según la revisión del estado del arte, realizada en el capítulo anterior, tanto la industria como la academia son conscientes de las necesidades de modificación a la enseñanza tradicional de la Ingeniería de *Software*, ya que en el momento no se está promoviendo la formación en ciertas habilidades necesarias para los ingenieros de *software*. A este respecto, Felder *et al.* (2000) señalan la importancia de promover del aprendizaje activo en la clase y el uso del aprendizaje cooperativo, con el fin de lograr que los estudiantes aprendan habilidades tan complejas y necesarias como la resolución de conflictos, la comunicación y el manejo de los recursos durante la ejecución de un proyecto de *software*.

Algunas estrategias que se han propuesto en la literatura como alternativas a la enseñanza tradicional de la Ingeniería de *Software* se compendiaron en el capítulo anterior. Son de especial importancia:

- Los esfuerzos realizados con el uso de juegos de computador o video para la enseñanza de ciertos temas o como ambientes de desarrollo de aplicaciones prácticas.
- Las modificaciones planteadas a los proyectos prácticos para potencializar su uso, complementándolo con el fomento de otras habilidades.

Sin embargo, y pese al uso de estas estrategias, subsisten varios problemas, tales como:

- Los métodos tecnológicos requieren muchos recursos y, consecuentemente, poseen altos costos que, a veces, universidades como la Nacional de Colombia no pueden pagar. En general, cuando se habla de juegos de computador o video y entornos para su desarrollo, se habla de ciertas necesidades en *software* y *hardware* que son difíciles de atender en este tipo de universidades; por ejemplo, se requieren aulas muy bien dotadas en las cuales un computador se asigne a cada estudiante para la realización de la clase y las prácticas extraclase que se derivan de este tipo de métodos. En la actualidad, en la Universidad Nacional, la clase de Ingeniería de *Software* se dicta parcialmente en aulas regulares, debido a la carencia de una infraestructura que pueda garantizar la

totalidad de las clases en aulas de computadores. En cuanto al *software*, es necesario adquirir licencias de los entornos y lenguajes de desarrollo especiales que se deban utilizar, con el agravante que este tipo de licencias comúnmente tendrían éste como único uso académico. Esta realidad riñe con el hecho que los cursos de este tipo tienden a tener más estudiantes con el paso del tiempo, lo que representa otro obstáculo para la dotación del *software* y *hardware* necesario para el uso de juegos de video o computador como estrategia didáctica. En la Universidad Nacional, por ejemplo, el curso de Ingeniería de *Software* ha pasado de 20 estudiantes como máximo durante los dos semestres de 2004 a tener 60 estudiantes durante el semestre 01/2007. Este también parece ser un problema generalizado en las universidades públicas de América Latina, según el estudio realizado por Arocena (2004).

- Una de las tendencias en métodos tecnológicos para complementar la enseñanza tradicional de la Ingeniería de *Software* se ha centrado casi completamente en la enseñanza de programación orientada a objetos y lenguajes de programación. Si bien estos temas son de gran importancia para los futuros ingenieros de *software*, se están descuidando otros temas que también pueden ser de importancia para este tipo de ingenieros. La Ingeniería de *Software* es mucho más que sólo programación y se relaciona con la adopción de un enfoque ingenieril para el desarrollo de *software*, y por ello requiere de otras habilidades como la planeación y ejecución de proyectos, el manejo adecuado de los recursos, la coherencia en el manejo de la información, los métodos de desarrollo de *software* y otros. En general, estos temas han sido poco o nada abordados desde las diferentes estrategias alternativas a la enseñanza tradicional en Ingeniería de *Software*.
- La estrategia no tecnológica más empleada es la modificación de los proyectos prácticos con un conjunto de estrategias para remarcar algunos temas especiales de la Ingeniería de *Software.* Sin embargo, este tipo de proyectos siguen siendo sólo simulaciones de lo que se pueden encontrar los futuros ingenieros de *software* en el desempeño de su actividad profesional. Es difícil para los estudiantes, por ejemplo, estimar el tamaño o el esfuerzo necesario en la elaboración de un proyecto de *software* real cuando sólo se ha practicado con proyectos de "juguete" con un tamaño diez o veinte veces menor. También, es difícil estimar los riesgos que podría correr un proyecto real con grandes cantidades de personal, cuando sólo se han manejado pequeños proyectos en los cuales los estudiantes deben desempeñar, por efectos de tamaño, todos los roles que se encuentran presentes en un desarrollo normal. Por ello, se requiere fomentar en los estudiantes de Ingeniería de *Software* la adquisición de otras habilidades que los proyectos prácticos no están en capacidad de impartir. Este panorama se hace paulatinamente más difícil puesto que, además de las dificultades para conseguir los contactos en la industria que permitan la elaboración de proyectos prácticos de "juguete" con temas reales, en ocasiones, cuando esos contactos se consiguen, se encuentran

problemas adicionales, como, por ejemplo, lo relativo a la propiedad intelectual sobre los productos elaborados en desarrollo de una práctica de este tipo. Esta situación reduce las posibilidades de cooperación con la industria para hacer proyectos prácticos más reales.

- En la revisión de la literatura, se presentaron algunos ejemplos de dos estrategias que fomentan el denominado aprendizaje colaborativo: el estudio de casos y las actividades lúdicas no ligadas con tecnología. La responsabilidad de este tipo de aprendizaje se comparte entre el docente y los estudiantes (Johnson *et al.*, 1993), a diferencia del aprendizaje tradicional, en el cual la responsabilidad se asigna completamente al docente. Se puede considerar, entonces, que, en el aprendizaje colaborativo, se puede propiciar la reflexión acerca de la labor de aprendizaje, que plantean Ausubel *et al.* (1978), posibilitando el denominado *aprendizaje significativo*. Sin embargo, son pocos los ejemplos que se pueden citar del uso de estas estrategias en la enseñanza de la Ingeniería de *Software*.
- En el caso de los otros dos tipos de estrategias alternativas a la enseñanza tradicional de la Ingeniería de *Software*, los laboratorios y la enseñanza personalizada, también las dificultades para su adopción se relacionan principalmente con los recursos necesarios para ello. En el caso de los laboratorios, se requeriría una gran cantidad de computadores para atender las necesidades de experimentación en desarrollo de *software* por parte de los estudiantes de Ingeniería de *Software*, además de las licencias requeridas para la elaboración de los mismos. En el caso de la enseñanza personalizada, el número de docentes tendría que crecer sustancialmente para atender con un seguimiento más cuidadoso el avance de los estudiantes a su cargo. Estas condiciones, nuevamente, no tendrían mucha aplicación en la Universidad Nacional de Colombia, dadas las limitaciones de recursos que afronta.

Una solución a los problemas esbozados, deberá tomar en consideración las siguientes restricciones:

- Es difícil la consecución de recursos tecnológicos de *hardware* y *software*. A este respecto, Mejía (2006) realiza una reflexión interesante en relación con los retos de la globalización en relación con la educación, en el cual se muestra la inversión como uno de los problemas importantes en este aspecto. Arocena (2004) presenta una conclusión similar para el ámbito latinoamericano.
- El número de estudiantes de esta profesión tiende a ser creciente, mientras que el número de profesores vinculados en la Universidad Nacional de Colombia tiende a conservarse. Por ello, los cursos masivos tienden a ser cada vez más comunes.

- La cooperación entre la academia y la industria tiende a ser cada vez más difícil. Existen problemas en esta cooperación asociados con aspectos como la privacidad de la información y la propiedad intelectual de los productos desarrollados en los proyectos prácticos.

Para efectos de este trabajo de promoción, se adoptará como solución la realización de juegos de clase con pocas necesidades de tecnología para su realización (*software* y *hardware*) y que se enfoquen en impartir algunos temas poco abordados en los métodos alternativos a la enseñanza tradicional de la Ingeniería de *Software*, tales como:

- El manejo de proyectos de *software*.
- El trabajo en equipo.
- La distribución de funciones al interior de un proyecto.
- La planeación de proyectos de *software*.
- Los métodos de desarrollo de *software*.
- La consistencia en el desarrollo de *software*.
- La trazabilidad y la comunicación en el desarrollo de *software*.

El grupo de investigación en Ingeniería de *Software* inició desde hace tres años una estrategia basada en juegos de clase no tecnológicos o con poca participación tecnológica. Producto de este trabajo surgió *El juego de los requisitos* (Zapata y Awad, 2005), un juego que se realiza en un aula de clase convencional apoyada por un número reducido de computadores (pues, los alumnos se organizan en grupos variables de 5 a 10 personas y cada grupo requiere un computador), y que repasa algunos de los conceptos fundamentales del desarrollo de *software* y la gestión de proyectos. A esta primera experiencia se suman otros cinco juegos que se presentan en este trabajo de promoción, denominados: *el juego del desarrollo de software*, *el juego de UN-Método*, *el juego de la consistencia*, *el juego de la planeación de proyectos de software* y *el juego de la comunicación y la trazabilidad*. Estos juegos puede coordinarlos el docente en aulas convencionales y permiten el refuerzo de los conocimientos que se imparten en el curso de una manera lúdica, y, por ello, exenta de las presiones que implica el proceso de evaluación. Además, se enfocan en áreas consideradas importantes en el currículo de la Ingeniería de *Software* que discute Fairley (1978).

CAPÍTULO 4: MARCO TEÓRICO

4.1. Algunas definiciones importantes

Huizinga (1972 e. o. 1938), establece el *juego* como una actividad libre, con unos límites temporales y espaciales definidos y entre los cuales se desarrolla, y con unas reglas obligatorias que todos los jugadores deben conocer, pero que voluntariamente pueden aceptar o no. Los juegos, en sí mismos, van acompañados de una profunda alegría, pero, paradójicamente, también de una sensación de tensión. Quien participa en un juego, lo hace con el ánimo de ganar, y espera de los demás una disposición igual pero sin recurrir a las trampas. Huizinga (1972 e. o. 1938) establece también que el juego tiene una finalidad, que puede ser la descarga del exceso de energía vital, la imitación o el ejercicio de actividades serias que se exigirán más adelante al jugador en su vida.

Daré y Barreteau (2003) establecen que los *roles* son conjuntos de funciones simuladas que desempeñan los jugadores y que se encuentran limitados por las reglas del juego. En otras palabras, los roles están circunscritos a las reglas formales que rigen el juego, las cuales controlan las actividades y comportamientos de los jugadores al interior del juego.

Juul (2003) considera que las *reglas* son órdenes que se imparten en el juego o conjuntos de preceptos que restringen las acciones que se pueden realizar al interior del juego. En este sentido, todo aquello que se debe cumplir por parte de los jugadores y que limita sus decisiones puede ser considerado una regla. Juul (2003) también define como elementos integrantes del juego las *variables*, que son resultados esperados del juego, a las cuales se les pueden asignar determinados *valores* que pueden ser positivos (la adición de puntos, por ejemplo) o negativos (algún tipo de penalización en términos de los elementos del juego), de forma tal que, con base en esas variables, se puede determinar el(los) ganador(es) del juego.

4.2. Los juegos como estrategia de aprendizaje

Boehm (2006) sugiere, para la enseñanza futura de la Ingeniería de *Software*, estrategias de autoaprendizaje para los estudiantes, y refiere específicamente entre estas estrategias la participación en investigación desde el pregrado y la realización de juegos instructivos en clase. Esta sugerencia coincide con la apreciación de Felder *et al.* (2000) para la Ingeniería en general,

cuando afirman que se requiere que el aprendizaje en clase sea activo y cooperativo, de forma que los estudiantes tengan una actitud más activa en relación con los conocimientos que se imparten, a diferencia de las tradicionales clases expositivas que se suelen impartir. Estrategias como los juegos en clase no han sido comúnmente empleadas en la enseñanza tradicional de la Ingeniería (o, más específicamente, en la Ingeniería de *Software*), pero ha sido estudiada como una estrategia viable de aprendizaje por varios autores, debido a razones como las siguientes:

- Motivación: Se reconoce que los juegos generan entretenimiento en sus practicantes, quienes optan por jugarlos movidos por el deseo de divertirse. La clase expositiva no es, en sí misma, divertida, pero la sola invitación a un juego suele tomarse como una invitación a la diversión. Los juegos como un forma de enseñanza motivadora están discutidos por Jensen (2006), Lee *et al.* (2004) y Dibona (2004).
- Representatividad: Es posible simular una parte de la realidad mediante juegos (Kasvi, 2000). En el Capítulo 2 se analizaron muchas de las experiencias con juegos de video o de computador que se empleaban para simular partes de la realidad (presente o pasado) o para simular comportamientos que se podrían dar en el futuro. Gee (2003) presenta una discusión en relación con juegos como *Age of Mythology* donde muestra cómo un niño puede captar muchos conocimientos a partir del juego como una simulación de los eventos del pasado.
- Interactividad y dinamismo: Además de representar una parte de la realidad, es posible interactuar con ella (Kasvi, 2000). Es posible recrear batallas enteras o simular comportamientos deseables para experimentarlos una y otra vez; ya sea que se trate de juegos de video o computador, o de juegos de mesa como *Clue* o *Monopoly*, el juego se puede repetir incontables veces, y tomar ventaja de los comportamientos que se practican en ellos. En este caso, los juegos de rol son especialmente útiles, pues, a diferencia de otros juegos de computador, permiten una interacción casi directa del jugador con el entorno que se está simulando (Gee, 2003).
- Competencia: La rivalidad que se presenta entre los participantes de un juego conserva el interés en el juego hasta el final (Pivec *et al.,* 2003). Mientras existan posibilidades de que los jugadores triunfen en el juego, el interés del juego se mantendrá. Huizinga (1972 e. o. 1938) realiza un análisis de la competencia en el juego, develando algunas de sus consideraciones éticas como la rivalidad excesiva, el codicioso deseo de ser ganador o la tendencia a las trampas.
- Seguridad: Es posible recrear una parte de la realidad, pero sin ningún peligro de perjuicios físicos a la salud o la integridad (Kasvi, 2000). Cuando se simulan las batallas en *Age of Mythology*, o cuando se va a la cárcel en *Monopoly*, esas acciones están ocurriendo en un ambiente completamente simulado que no causa daños físicos a quien lo juega. Esto es válido también para

juegos de simulación de vuelo, por ejemplo, donde se pueden practicar los efectos de ciertas acciones sin el temor de perder aviones llenos de pasajeros.

Klassen y Willoughby (2003) presentan otras razones para emplear los juegos como estrategias educativas:

- El conocimiento se devela al participar en el juego. Los participantes experimentan una sensación de conocimiento "obvio" una vez han participado en el juego, y esa experiencia permite un aprendizaje significativo mucho mayor.
- Entre mayor participación se da en el juego, mayor es el conocimiento adquirido. Sin importar si los participantes ganan o pierden el juego, el nivel de intensidad de su participación se refleja en los conocimientos que se logran en el mismo.
- Reduce los niveles de tensión de los participantes. En el caso de las clases expositivas, el temor de la evaluación actúa como una fuente de tensión para los estudiantes. Cuando se trata de participar en un juego, lo que menos importa en un momento dado es el resultado y la participación se logra de manera mucho más desinteresada que en una clase expositiva.
- Se pueden emplear materiales sencillos. No siempre es necesario un material tecnológicamente avanzado para realizar una clase mediante juegos. Con un tablero convencional y unos dados o unos cartones de juego se puede realizar este tipo de actividades. No es necesario disponer de *hardware* o *software* especializado como en el caso de los juegos de simulación que se presentaron en el capítulo 2.

Existe en la literatura especializada una gran cantidad de ejemplos de juegos de computador o video que se han empleado para la enseñanza de diferentes habilidades o valores. Además de los casos reportados en el capítulo 2, se pueden mencionar las experiencias positivas que se han realizado con el juego de baile (Höysniemi, 2006), los valores positivos de los juegos de cómputo (Barr *et al.*, 2005), la capacidad de uso de los juegos gráficos para niños (Egloff, 2004) y el proyecto de juegos para enseñar que se está desarrollando en MIT (Jenkins *et al.*, 2003). Sin embargo, por las razones aducidas en el capítulo 3 en relación con las limitaciones de recursos de la Universidad Nacional de Colombia, este tipo de juegos no podría utilizarse ampliamente en la enseñanza de la Ingeniería de *Software* en este medio.

Por otra parte, en ciencias como la administración y la investigación de operaciones se han tenido experiencias positivas con la utilización de juegos no tecnológicos en el aula de clase, como por ejemplo:

- *El juego de la cerveza* (Senge, 1994), que se ha jugado durante años para explicar la dinámica del manejo de inventarios a diferentes audiencias.
- *El juego de la negociación* (Christopher y Smith, 1991).

En Ingeniería de *Software* se pueden mencionar sólo dos experiencias de este tipo:

- *Problemas y programadores* (Baker *et al.*, 2005), un juego de cartas que sirve para ejemplificar el proceso completo de desarrollo de *software*, pero haciendo un énfasis especial en el proceso final de implementación y los problemas que pueden aquejar a los programadores dependiendo de diferentes factores (humanos, relativos al proyecto, imprevistos, etc.).
- *El juego de los requisitos* (Zapata y Awad, 2005), un juego enfocado en el desarrollo de una pequeña aplicación de *software* en la cual se pueden simular aspectos como adecuación con los requisitos, documentación, trabajo en equipo, competitividad, etc.

En el siguiente capítulo, se presentan cinco juegos no tecnológicos que se pueden aplicar en el aula de clase para enseñar varios aspectos de la Ingeniería de *Software*, aún no cubiertos por los juegos mencionados. Estos juegos hacen parte de las experiencias en el desarrollo de este tipo de estrategias durante los tres últimos años en el grupo de Ingeniería de *Software* de la Escuela de Sistemas de la Universidad Nacional de Colombia, Sede Medellín, y se proponen superar algunas de las limitaciones (especialmente de recursos) que se pueden presentar en la Universidad Nacional de Colombia para la enseñanza de la Ingeniería de *Software*. Se discuten en el capítulo 6 los resultados de la aplicación de estos juegos a varios grupos de estudiantes de la Universidad Nacional de Colombia en las áreas de Ingeniería de *Software* y la Línea de Profundización en Ingeniería de *Software*, así como a otros públicos como los estudiantes de primer semestre de Ingeniería de Sistemas y los estudiantes de la maestría en Ingeniería de Sistemas de la misma universidad. Además, se incluye un grupo de estudiantes y profesores de Ingeniería de *Software* e Ingeniería Lingüística de la Universidad Nacional Autónoma de México y un grupo de estudiantes de diferentes universidades latinoamericanas, reunidos en Lima (Perú) para las VI jornadas de Ingeniería del *Software* e Ingeniería del Conocimiento, y con los cuales se realizó *el juego de la consistencia* como parte de un tutorial impartido por el autor de este trabajo (Zapata, 2007).

El origen de los juegos ha sido diverso: *el juego de la consistencia* se diseñó, implementó y aplicó como un trabajo de grado. *El juego de la trazabilidad y la comunicación* se desarrolló como un proyecto especial de trabajo académico. Los demás juegos surgieron de la necesidad de diversificar la forma de enseñanza tradicional de la Ingeniería de *Software* en la Escuela de Sistemas de la Universidad Nacional de Colombia.

4.3. Algunas corrientes pedagógicas y su relación con los juegos planteados

A mediados del siglo XX, Skinner (1958) propuso una "máquina de enseñar", que seguía los lineamientos de la denominada ciencia del comportamiento, conocida como conductismo. En esta corriente, se estudia la interacción que se presenta entre el individuo y los diferentes contextos que lo rodean, tales como el social, el biológico y el físico. Para Skinner la conducta posee regularidades que la hacen predecible y controlable mediante estímulos que la pueden reforzar positiva o negativamente. Esta teoría del refuerzo fue controvertida por White (1959), cuando propuso la teoría de la competencia como un elemento crucial en la motivación y en la manera de aprender de los individuos. White es reconocido como el primer investigador que "introdujo el término *competencia* para describir aquellas características de la personalidad asociadas con rendimiento superior y alta motivación" (Delamare y Winterton, 2005). Desde entonces, tanto el conductivismo como la formación por competencias se han alternado en los diferentes escenarios pedagógicos para tratar de representar la forma en que los individuos logran su aprendizaje.

El constructivismo es otra de las tendencias que se han empleado para comprender el aprendizaje de los individuos. Siguiendo esta tendencia, Piaget (1952) propuso una serie de etapas que describen la manera en que se desarrolla el conocimiento en los niños. Según Piaget, los mecanismos de asimilación y acomodación son los que caracterizan la evolución y adaptación del hombre. En el primer mecanismo, se produce una interiorización de los conceptos. En el segundo mecanismo, se modifican los conceptos para acoger nuevos conceptos y eventos. En el constructivismo, el conocimiento es una construcción activa del individuo y, por ello, no existe un conocimiento realmente objetivo. De esta manera, uno de los principales riesgos del enfoque constructivista es la tendencia a desembocar en el relativismo, un sistema de pensamiento que sostiene que las verdades universalmente válidas no existen, pues existe una alta dependencia sobre condiciones o contextos de la persona o grupo que la afirma.

Estas corrientes pedagógicas han recibido algunas críticas:

- Miranda (2000) afirma que el constructivismo es una corriente más psicológica que pedagógica y que, de forma errónea, se ha pretendido emplearla como una explicación a los problemas educativos. El constructivismo, visto desde esta óptica, coloca al estudiante como "constructor de su propio conocimiento", pero ello hace que se desvirtúe el papel del docente en relación con este proceso.
- Arcila y Gaona (2006) realizan un examen minucioso de lo que ha representado el conductismo en la formación por competencias de los alumnos. A través del análisis que realizan, muestran una idea que ha sido el resultado de muchas tergiversaciones de la teoría conductista, en la cual las competencias se adquieren como "recetarios" o manuales, en los cuales prácticamente no se requiere la intervención del docente, cumpliendo de alguna manera con el sueño de Skinner (1958) en relación con la "máquina de enseñar". Este enfoque, que podría ser válido para la formación de habilidades mecánicas en oficios que así lo requieran, no es igualmente válido cuando se trata de formar competencias mucho más creativas o, si se quiere, menos mecánicas. Estos autores afirman que la construcción de competencias es un juego comunicativo de saber y poder: el profesor reconoce en el alumno a un interlocutor competente en potencia, en tanto que el alumno reconoce que no ha desarrollado las competencias necesarias, pero que puede hacerlo. En este sentido, conviene rescatar del enfoque psicológico constructivista de Piaget el hecho de que existe en los alumnos una "materia prima" inicial para la construcción de las competencias, pero, a diferencia del enfoque constructivista, esa construcción de competencias se obtiene con la intervención docente-estudiante, con la participación de múltiples estrategias.
- Mejía (2006) plantea algunas críticas severas a la educación por competencias que se está realizando en Colombia. Al respecto, afirma que se ha pasado de un enfoque social de las competencias a un enfoque de competencias individualizadas que buscan hacer que los individuos desarrollen únicamente las habilidades que les permitan pugnar por las escasas plazas disponibles en el mercado laboral.
- Ocampo (2003) realiza una crítica severa a la aplicación del constructivismo en la educación en Colombia, especialmente en dos de las principales críticas que hacen los constructivistas a la enseñanza tradicional: el transmisionismo repetitivista y la negociación del conocimiento entre estudiantes y docentes. En cuanto a la primera crítica, afirma Ocampo que es prácticamente imposible construir el conocimiento desde cero, sin que haya algún tipo de transmisión básica del conocimiento del profesor al alumno. En cuanto a la segunda crítica, respecto de la cual los

constructivistas afirman que tanto los estudiantes como los profesores son pares y, por ello, tienen igual validez las ideas de unos y otros, Ocampo afirma que en muchas ocasiones las ideas de los estudiantes carecen de validez científica, lo que las descalifica para imponerse sobre las ideas de los profesores.

En relación con las corrientes pedagógicas descritas, los juegos que se proponen en el capítulo siguiente corresponden más a una mezcla de la enseñanza tradicional con algunas de las corrientes anotadas. Esta afirmación se sustenta en los siguientes hechos:

- Los juegos son un complemento de las clases expositivas. Se requiere que los alumnos hayan recibido algo del "transmisionismo" que afirma Ocampo (2003) para que los conceptos se puedan internalizar a través de los juegos.
- Se intentan desarrollar algunas competencias necesarias para los ingenieros de *software*. En este sentido, la formación por competencias, con las limitaciones anotadas, constituye la manera de que los estudiantes adquieran ciertos conocimientos que a partir de las clases expositivas es difícil transmitir, como es el caso del comportamiento de la competencia o la incidencia del azar en el desarrollo de los proyectos en Ingeniería de *Software*. Además, se pretende que los estudiantes adquieran ciertas competencias sociales: la dirección de grupos de trabajo, la participación en el trabajo colectivo, la capacitación requerida por los integrantes de un mismo grupo, entre otras. En este sentido, se trata de adquirir las habilidades que posibiliten, no sólo las habilidades y destrezas necesarias para luchar individualmente por las plazas disponibles de trabajo, sino también adquirir el sentido social de la participación en grupos de trabajo.
- Los juegos son en parte constructivistas, porque el docente no transmite completamente todo el conocimiento en los alumnos, sino que le da los insumos necesarios para que, mediante el ejercicio del juego, se construyan e internalicen ciertas conclusiones e ideas.
- En sentido estricto, los juegos que se presentan no emplean estrategias conductistas, puesto que el apoyo del docente no está orientado específicamente a modificar un esquema mental en los alumnos, sino a movilizar procesos de pensamiento, cuyo resultado no siempre será determinado por el estímulo presentado por el docente con el juego. Los juegos son más una construcción colectiva, en la cual el resultado muchas veces es impredecible y le muestra al docente conocimientos alternativos que ni siquiera había intuido en la planeación inicial del juego.

En todo caso, lo que se pretende a través de los juegos es una mezcla de educación y formación, que es completamente centrada en el individuo mismo, tal como se presenta en la antropología filosófica (Gevaert, 1976). Más que la construcción de conocimientos o el desarrollo de competencias, se busca que los alumnos tengan una educación integral, en la cual interactúen con situaciones que, como profesionales, enfrentarán en el futuro.

4.4. Aspectos teóricos de la Ingeniería de *Software* y la gestión de proyectos

Gibbs (1994) analiza el surgimiento de la Ingeniería de *Software* a partir de la situación caótica que vivía el *software* a finales de los años sesenta. En ese tiempo, las aplicaciones se desarrollaban como esfuerzos colectivos sin mucha planeación y realizados de manera artística, casi dependiente de las capacidades de los programadores de entonces. En ese instante particular de la historia, el desarrollo de *software* se encontraba al borde del colapso: los plazos rara vez se cumplían y las estimaciones en cuanto a los costos solían desbordarse, de modo que las aplicaciones de *software*, por lo general, resultaban ser mucho más costosas de lo esperado y se solían entregar con un retraso considerable. A raíz de esta situación, la OTAN decidió intervenir y sentó las bases de lo que podría ser una solución: emplear los principios de otras ingenierías en el desarrollo de *software*. De ahí surgió el término *Ingeniería de Software*, como una manera de expresar que la disciplina y los métodos ingenieriles podrían tener aplicación en el desarrollo de una aplicación de *software*. En otras profesiones, como por ejemplo la Ingeniería Civil, la construcción de los productos no se inicia hasta tanto se haya agotado una serie de etapas consideradas *preliminares* y en las cuales se trata de conceptualizar el producto y de definirlo de manera completa, con el fin de que los encargados de realizar la construcción posean planos detallados del mismo. De manera sintética, exactamente esto mismo se pretende en la Ingeniería de *Software*: construir los planos de una aplicación mucho antes de escribir la primera línea de código, de tal forma que se pueda comprender el problema y conceptualizar su solución, para identificar de manera rápida los principales obstáculos que se pueden presentar en la construcción de la aplicación de *software*.

Después de casi cuarenta años de existencia, la Ingeniería de *Software* ha logrado incorporar una serie de modelos de procesos, métodos y artefactos al interior de su cuerpo de conocimientos, de forma que:

- Los modelos de procesos representan una serie de etapas o fases que se deben realizar para obtener las especificaciones completas de la aplicación de *software* y el *software* mismo. El modelo de procesos más común es el modelo en cascada propuesto inicialmente por Royce (1970), y que posee actualmente las siguientes fases: definición, análisis de requisitos, diseño, construcción, transición y producción. El modelo en cascada se puede apreciar en la figura 1, tomada de Zapata y Gómez (2006).

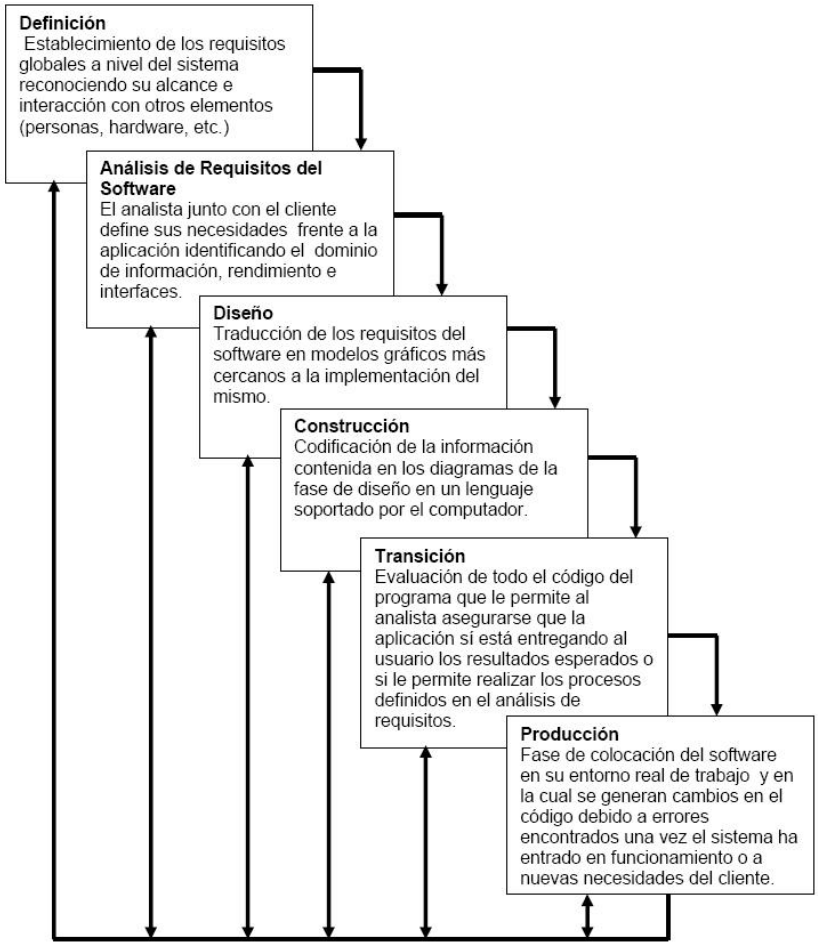

Figura 1. Modelo en cascada. Tomado de Zapata y Gómez (2006).

- Los métodos de desarrollo detallan la aplicación de los modelos de procesos. Existen dos tendencias en lo relativo a métodos de desarrollo: la línea tradicional, integrada por los denominados métodos monumentales, en los cuales se trata de desarrollar una documentación muy completa que permita especificar detalladamente la aplicación de *software*, y la línea moderna, integrada por los denominados métodos ágiles, caracterizados por la comprensión rápida del problema para documentar poco y pasar rápidamente al desarrollo del *software*, con la revisión permanente del cliente. Los dos tipos de métodos presentan diferencias apreciables y públicos

diferentes. Sin embargo, y sin entrar a discutir cuáles de ellos son mejores que los otros, la Ingeniería de *Software* pretende que se utilice un método de desarrollo, sin importar cuál de ellos se aplique. Dentro de los métodos monumentales se destacan el *Rational Unified Process* (RUP) (Kruchten, 1999) y el *Custom Development Method* (CDM) (ORACLE, 2000); estos dos métodos tienen aceptación mundial y se han utilizado repetidamente en el desarrollo de grandes aplicaciones de *software*. Dentro de los métodos monumentales, se cuenta también el que se encuentra actualmente en desarrollo por parte del grupo de Ingeniería de *Software* de la Escuela de Sistemas de la Universidad Nacional de Colombia, denominado UN-Método (Arango y Zapata, 2006). Entre los métodos ágiles más importantes se encuentran el *Feature-Driven Development* (FDD) (Coad *et al.*, 1999) y la *Extreme Programming* (XP) (Beck, 2000). También, estos dos métodos se han usado repetidamente a nivel mundial para el desarrollo de aplicaciones.

- El término *artefacto* se refiere a los diferentes elementos con los cuales los métodos de desarrollo cumplen con las diferentes fases del modelo de procesos al que pertenecen. En último término, los artefactos son los diagramas, tablas y elementos que permiten especificar de manera completa la aplicación de *software* que se desea construir. Entre los artefactos de más común uso en el desarrollo de *software* se encuentran el *Unified Modeling Language* (UML) (OMG, 2007), que es un conjunto de diagramas semiformales que permite la especificación de cualquier aplicación de *software*, y el *Entity-Relationship diagram* (ERD) (Chen, 1976), uno de los diagramas más famosos para la representación de los datos en un problema particular. UN-Método utiliza, además, otros diagramas y artefactos para la especificación del *software*, como los denominados *esquemas preconceptuales* (Zapata *et al.*, 2006a), los diagramas de objetivos, los diagramas causa-efecto, los diagramas de procesos y algunos de los diagramas de UML. Ahora, cuando los artefactos se utilizan dentro de cualquier método de desarrollo para especificar la solución informática a un problema del mundo, esos artefactos deben guardar una coherencia entre ellos, que se suele denominar *consistencia* y que se traduce en que los diferentes conceptos y relaciones del mundo deberían estar representados en los diferentes artefactos que hacen parte del método seleccionado, pero respetando unas reglas de formación tanto al interior de los diagramas como entre diferentes diagramas. Además, debe ser posible hacer un seguimiento de un concepto o relación a través de los diferentes diagramas en que se puede representar, lo que se suele denominar *trazabilidad*.

Entre los diferentes artefactos de UN-Método, cabe destacar la utilidad de los siguientes artefactos:

- Esquema preconceptual: es una especie de grafo conceptual que se emplea para la definición de los diferentes conceptos del dominio de un problema, las relaciones que los ligan (ya sean

estructurales o dinámicas) y las condiciones para la realización de las acciones en el mundo (Zapata *et al.*, 2006a)

- Modelo del dominio: es un modelo que permite representar la parte estructural del dominio del problema (únicamente los conceptos, representados como clases y atributos) y las relaciones estructurales.
- Diagrama de objetivos: es un diagrama que representa de manera jerárquica los objetivos de alto nivel de la organización y los va desglosando hasta llegar a los requisitos del *software* y los actores responsables de su cumplimiento. Se suele emplear como diagrama inicial de la metodología KAOS (Dardenne *et al.*, 1993), que consiste en la elicitación de los requisitos y expectativas del interesado a partir de los objetivos de la organización.
- Diagrama causa-efecto: es un diagrama con forma de espina de pescado en la cual un problema se va desagregando en las causas que lo generan (Ishikawa, 1986).
- Diagrama de procesos: es un diagrama que representa los principales actores del problema y las diferentes funciones que realizan en el dominio, tomando en consideración los almacenes de datos que se generan en dicho dominio. Para el caso de UN-Método, este diagrama se ha modificado según las necesidades del método y se presenta de manera un poco diferente al diagrama original, que se emplea en el método CDM (ORACLE, 2000).
- Diagrama de casos de uso: es un diagrama que presenta los actores de la organización y las funciones que comenzarán a desempeñar en el *software*, cuando éste entre en funcionamiento (OMG, 2007). A diferencia de los otros artefactos de UN-Método descritos hasta ahora, este diagrama se enfoca en la solución y no en la comprensión del problema.
- Interfaces gráficas de usuario: correspondientes a las pantallas de la aplicación informática que se desarrolla luego que los requisitos han sido analizados y se presenta una solución al problema. Están íntimamente ligadas con el diagrama de casos de uso.
- Diagrama de clases: es un diagrama que representa los objetos del mundo agrupados en categorías especiales denominadas *clases*, sus características (atributos), las funciones que se pueden realizar con los grupos de objetos (operaciones) y las diferentes formas de conexión entre clases (OMG, 2007). En UN-Método se pueden construir diagramas de clases en cualquier momento que la aplicación lo requiera, pero, en el contexto del juego en que se van a utilizar, los diagramas de clases pertenecerán a la fase de diseño.

Ahora, las empresas de *software* suelen acometer el desarrollo de aplicaciones mediante proyectos; un proyecto es un conjunto ordenado de actividades que se deben cumplir para lograr un objetivo, que en este caso es la culminación de una aplicación de *software*. Para ello, las empresas de

software suelen dividir las personas en especialidades, denominadas *roles*, para que cada rol se encargue de realizar unas funciones específicas dentro del desarrollo. En el caso de las empresas convencionales, existen roles como: analista, diseñador, programador y jefe de proyectos. Un rol que no hace parte de las empresas de *software*, pero que es de gran importancia dentro del desarrollo es el rol de *interesado* o *participante* (tomado del vocablo inglés *stakeholder*), que es toda aquella persona que posee un interés en que la aplicación de *software* se desarrolle. El desarrollo de *software* se suele acometer conjuntamente con los interesados, pues, estos son los que definen las necesidades y expectativas que rodean esa aplicación de *software*, y también son los llamados a emplear dicho *software* una vez se haya desarrollado. Tanto al interior del grupo de trabajo como entre el grupo de trabajo y los interesados debe existir una comunicación permanente, que posibilite la comprensión de las necesidades y expectativas para su conversión en los requisitos del *software* y su conservación en la aplicación hasta el final de su desarrollo. Cada proyecto debería planearse meticulosamente, de forma que se pueda hacer un seguimiento cuidadoso del desarrollo de las diferentes actividades, pero ello pocas veces ocurre en la práctica. Una de las herramientas generalmente empleadas para la planeación de tiempos y recursos son los diagramas de Gantt, que son diagramas de barras en los cuales las actividades se distribuyen en el tiempo tomando en cuenta los recursos que son necesarios para desarrollarlas.

Con este marco teórico de la Ingeniería de *Software* en mente, es posible ya definir el ámbito de los cinco juegos que se proponen para complementar la enseñanza tradicional de la Ingeniería de *Software*. Estos ámbitos son los siguientes:

- El juego del desarrollo de *software* es una simulación de la fase de construcción de una aplicación de *software*, en la cual se procura que los jugadores conviertan un diseño predefinido en un conjunto de módulos y aplicacioness de *software*. Además, en este juego se practican los principios de especialización de las tareas del proyecto y comunicación permanente al interior del grupo de trabajo y con los interesados.
- El juego de UN-Método es una simulación de las fases de definición, análisis de requisitos y diseño de una aplicación de *software* empleando UN-Método. La idea es identificar la secuencia correcta de los artefactos que hacen parte de este método, con el fin de completar la especificación genérica de una aplicación. En este juego se tratan de representar adicionalmente algunos de los riesgos que posee el desarrollo de *software* y los procesos de toma de decisiones que deben realizar los diferentes integrantes de los grupos de desarrollo para sacar adelante los proyectos.

- El juego de la consistencia es un ejercicio de correlación entre cuatro artefactos comúnmente empleados en UN-Método para el desarrollo de *software*: los esquemas preconceptuales y los siguientes diagramas de UML: clases, secuencias y casos de uso. Se pretende determinar las relaciones de similitud que existen entre estos diagramas y las reglas que se deben verificar para evaluar la consistencia entre ellos.
- El juego de la planeación es una simulación de la planificación temporal que se debe hacer en un proyecto de *software* para realizar el desarrollo, pero, al igual que el juego de UN-Método, es una representación de la gran cantidad de riesgos que hacen parte del desarrollo de *software*. Aquí también los procesos de toma de decisiones cumplen un papel preponderante.
- El juego de la trazabilidad y la comunicación es, al igual que el juego de la consistencia, una manera de ligar los diferentes artefactos de UN-Método, en este caso el diagrama de clases de UML, el diagrama de casos de uso de UML y las interfaces gráficas de usuario, con el fin de rastrear los elementos comunes a los diagramas y la manera como subsisten hasta llegar a la interfaz. Además, en este juego, se procura representar la comunicación interna del grupo de desarrollo para poder culminar la tarea que se encomienda.

CAPÍTULO 5: JUEGOS DESARROLLADOS PARA LA ENSEÑANZA DE LA INGENIERÍA DE *SOFTWARE*

5.1. El juego del desarrollo de *software*

5.1.1. Objetivo del juego

Obtener el mayor número posible de módulos de *software* (en este caso, cajas elaboradas mediante *origami* con cuatro grupos de letras diferentes SO, FT, WA y RE), que conformen el mayor número posible de aplicaciones completas de *software* (en este caso, cada cuatro módulos que contengan un grupo de letras para formar la palabra *SOFTWARE* completa), en una competencia abierta con otros equipos de desarrollo. Además, se procura lograr las mayores utilidades para la compañía que representa cada uno de los equipos de desarrollo.

5.1.2. Materiales

- Hoja de especificaciones: Es una hoja que contiene las instrucciones en inglés para elaborar una caja de papel con una técnica de corte y doblaje denominada *origami*. El diseño corresponde a Glynn (1999) y se puede apreciar en la figura 2. Se entrega una hoja de especificaciones por equipo.
- Hoja de materia prima para los módulos: Es una hoja que contiene los cuatro grupos de letras (SO, FT, WA y RE) distribuidos de manera especial, tal y como se muestra en la figura 3. Se entrega un número variable de hojas de materia prima por equipo.
- Lápiz: se entrega un número variable de lápices por equipo.
- Hoja de control de juego: Es una hoja en la cual se registra el número de hojas de especificaciones, hojas de materia prima, lápices, empleados administrativos, empleados operativos, módulos comprados y palabras *software* completas. Los valores de cada elemento ya están predefinidos y se pueden asimilar a unidades monetarias. Con esta hoja, se determina la utilidad o pérdida de cada una de las empresas para determinar el equipo ganador del juego. La hoja de control, se puede apreciar en la tabla 1.

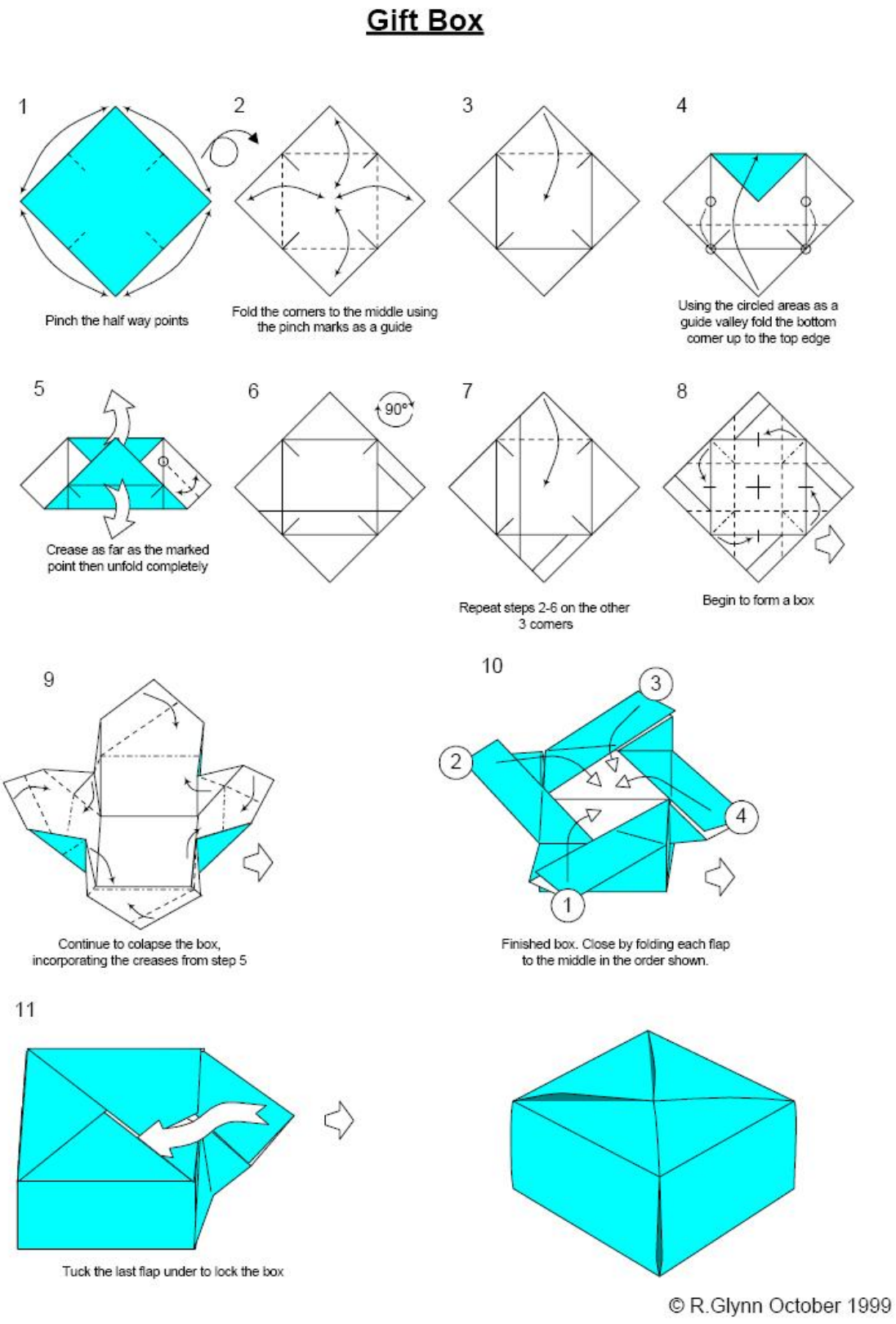

Figura 2. Hoja de especificaciones del juego del desarrollo de *software*.

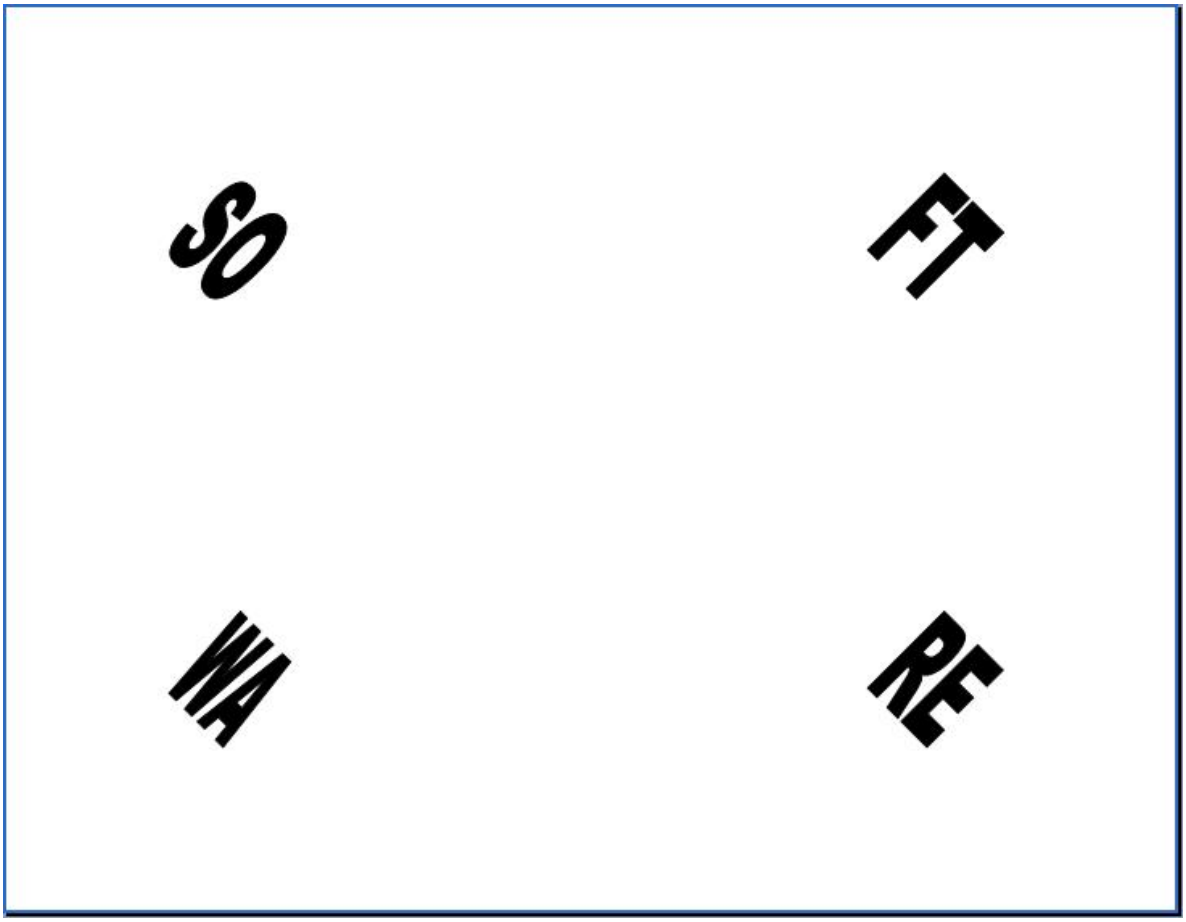

Figura 3. Hoja de materia prima para los módulos del juego del desarrollo de *software*.

Grupo No. ____

Descripción	Cantidad	Vr. Unit	Vr. Total
Hoja de especificaciones		500	
Hoja de materia prima		80	
Lápices		30	
Empleado administrativo por ciclo		300	
Operario por ciclo		150	
Total Costos			
Módulos Recibidos		150	
Palabra *software* completa		200	
Total Ingresos			
Utilidad o pérdida bruta			

Tabla 1. Hoja de control del juego del desarrollo de *software*.

5.1.3. Reglas de juego

- Los jugadores se distribuyen en equipos en los cuales se nombra un empleado administrativo y un número variable de empleados operativos.
- Se explica el juego en una sesión de 10 minutos de duración.
- El director del juego (que, por lo general, puede ser el profesor de la asignatura) entrega a cada equipo la hoja de especificaciones y la primera hoja de materia prima para los módulos. Estas hojas tienen el costo registrado en la hoja de control del juego y constituyen los primeros costos en los que deben incurrir los diferentes equipos de desarrollo. Se espera que los equipos de desarrollo saquen cuatro módulos, correspondientes a una palabra *SOFTWARE* de cada hoja. Como las especificaciones muestran un módulo que parte de una hoja cuadrada y la página tamaño carta es rectangular, los equipos deberán diseñar una estrategia para que se elaboren los módulos tal cual deben quedar. Para ello, el director del juego puede tener un conjunto de módulos de muestra, los cuales poseen en esas caras cuadrados de 3,8 cm. aproximadamente.
- Se da un plazo de 5 minutos para practicar la elaboración de los módulos. El equipo debe deliberar para determinar cuántas hojas de materia prima deberá comprar para elaborar módulos durante un tiempo de 50 minutos. Durante ese tiempo, el director del juego deberá dejar visibles los requisitos de los módulos, que son los siguientes:

    - No deben estar rotos.
    - No deben presentar arrugas.
    - No deben estar rayados.
    - Los rasgados no deben ser visibles.
    - Las letras no deben sobrepasar los límites del cuadro central.
    - No deben estar rasgados.
    - Deben estar limpios.
    - No deben presentar dobleces adicionales a los que se presentan en la hoja de especificaciones.
    - Deben poseer un número consecutivo en la cara lateral derecha.

- Se da un plazo de 5 minutos para que los empleados administrativos de cada equipo soliciten el número de hojas que emplearán durante los 50 minutos de juego. Sólo se compra una vez y no se

devuelve el material sobrante. Desde el momento de la entrega, las hojas son propiedad de cada uno de los equipos.

- El director del juego registra en las hojas de control los datos de cada uno de los equipos: número de hojas de especificaciones, número de hojas de materia prima, número de lápices, número de empleados administrativos y número de empleados operativos. Registra, además, en un lugar visible, la hora de inicio de los 50 minutos de juego.
- Se da un plazo de 50 minutos para la elaboración de los módulos. Durante ese tiempo sólo el empleado administrativo podrá hacer preguntas de cualquier tipo al director del juego y será además el único de los integrantes del grupo que no podrá realizar labores de corte y/o doblado. Con ello, se pretende que el empleado administrativo asuma las labores correspondientes a su cargo (que posee un costo diferente al de los empleados operativos), especialmente en lo concerniente a la comunicación con el cliente y la coordinación de las labores operativas. El director responderá las preguntas y verificará que las normas del juego se cumplan.
- El director del juego anuncia el final de los 50 minutos y procede a la recepción de los módulos elaborados que cumplan con los requisitos establecidos. Si, para un módulo particular, no se cumple alguno de los requisitos, el módulo no se recibe y se procede a la revisión del siguiente. Los módulos recibidos junto con la cantidad de las aplicaciones de *software* completas se registran en la hoja de control.
- Se totalizan los resultados y se determina el ganador.
- El director del juego modera una discusión de 20 minutos en relación con los principales hallazgos del juego. Es importante la participación activa de los jugadores en la discusión, la cual de todos modos será conducida por el director del juego, si bien es él quien conoce de antemano algunos de los conocimientos que se deben reafirmar con el juego.
- El director del juego brinda una charla de 5 minutos para entregar el resumen de las opiniones y las conclusiones definitivas del juego.

5.2. El juego de UN-Método

5.2.1. Objetivo del juego

Determinar el orden de elaboración de los diagramas en el método de desarrollo UN-Método, realizando un recorrido aleatorio a un tablero de casillas que contienen pistas para ello. Adicionalmente, se pretende reconocer algunos de los riesgos que pueden hacer parte del desarrollo de una aplicación de *software* empleando UN-Método.

5.2.2. Materiales

- Tablero de juego: Es un archivo en procesador de texto que puede también dibujarse en una pizarra o en cartones de juego, con forma similar al popular juego Monopoly®. El tablero posee las siguientes zonas, como se muestra en la figura 4:

<table>
<tr><td>INICIO</td><td>Pierde un turno</td><td>Escoja Casilla Diagrama</td><td>Clases Propio</td><td>Libera Diagrama</td><td>Clases Ajeno</td><td>Pierde un turno</td><td>Grafo Concep. Propio</td><td>Grafo Concep. Ajeno</td><td>Pierde Diagrama</td><td>Interfaz Ajena</td><td>Interfaz Propia</td></tr>
<tr><td>Escoja Casilla Diagrama</td><td colspan="3">EQUIPO AZUL</td><td colspan="3">EQUIPO VERDE</td><td colspan="3">EQUIPO ROJO</td><td colspan="2">EQUIPO NEGRO</td><td>Pierde un turno</td></tr>
<tr><td>Casos de Uso Propio</td><td colspan="3" rowspan="3"></td><td colspan="3" rowspan="3"></td><td colspan="3" rowspan="3"></td><td colspan="2" rowspan="3"></td><td>Modelo Dominio Propio</td></tr>
<tr><td rowspan="2">Casos de Uso Ajeno</td><td rowspan="2">Modelo Dominio Ajeno</td></tr>
<tr><td colspan="3">CAPTURADOS</td><td colspan="3">CAPTURADOS</td><td colspan="3">CAPTURADOS</td><td colspan="2">CAPTURADOS</td></tr>
<tr><td>Pierde un turno</td><td colspan="3"></td><td colspan="3"></td><td colspan="3"></td><td colspan="2"></td><td>Libera Diagrama</td></tr>
<tr><td>Pierde Diagrama</td><td colspan="10">DIAGRAMAS DISPONIBLES</td><td>Escoja Casilla Diagrama</td></tr>
<tr><td>Diagrama Objetivos Propio</td><td>Libera Diagrama</td><td>Diagrama Objetivos Ajeno</td><td>Pierde un turno</td><td>Causa Efecto Ajeno</td><td>Escoja Casilla Diagrama</td><td>Causa Efecto Propio</td><td>Libera Diagrama</td><td>Pierde un turno</td><td>Diagrama Procesos Propio</td><td>Pierde Diagrama</td><td>Diagrama Procesos Ajeno</td></tr>
</table>

Figura 4. Tablero del juego de UN-Método

  - Casillas de recorrido: Son casillas perimetrales que permiten el desplazamiento de las fichas de juego.
  - Diagramas disponibles: Es la zona donde se encuentran los diagramas que se pueden solicitar o capturar por parte de los diferentes equipos.
  - Diagramas por equipo (azul, verde, rojo, negro): Es la zona donde se encuentran los diagramas que ya hacen parte del desarrollo específico de *software* que está realizando cada equipo.
  - Capturados por equipo (azul, verde, rojo, negro): Es la zona donde se encuentran los diagramas de los otros equipos que un determinado equipo ha logrado capturar.

- Fichas de movimiento: Fichas de colores que representan los diferentes equipos de juego. Se ubican sobre casillas de recorrido del tablero.

- Fichas de diagramas: Fichas correspondientes a los diagramas de los diferentes equipos. Cada equipo posee grafo conceptual (esquema preconceptual), modelo del dominio, diagrama de objetivos, diagrama causa-efecto, diagrama de procesos, diagrama de casos de uso, interfaz y diagrama de clases. Estas fichas se deben organizar de manera aleatoria en la zona de diagramas disponibles, con el fin de que los diferentes equipos no intuyan la secuencia en la cual deberán solicitarlos.
- Dados: Se debe disponer de dos dados convencionales para realizar los movimientos sobre las casillas de recorrido. El archivo en procesador de texto viene acompañado por un programa de simulación de dados elaborado en Visual Basic® y cuya interfaz se muestra en la figura 5.

Figura 5. Programa de simulación de dados del juego de UN-Método.

5.2.3. Reglas de juego

- Se ubican las fichas de movimiento correspondientes a cada uno de los equipos en juego en la casilla INICIO.
- Los jugadores se distribuyen en equipos y se les asigna un color (azul, verde, rojo, negro).
- Se rifa cuál de los equipos va a iniciar, lo que se puede hacer con los dados, buscando el número máximo.
- El equipo que inicia lanza los dados y cuenta el número de casillas indicado por los dados en sentido de las agujas del reloj. Existen las siguientes posibilidades de casillas:

- Casillas de diagramas propios: Pueden ser clases, grafo conceptual, interfaz, modelo del dominio, diagrama de procesos, causa-efecto, diagrama de objetivos o casos de uso. En este caso, el equipo debe verificar que el diagrama se encuentre en la zona de diagramas disponibles y, en ese caso, debe decidir si solicita o no ese diagrama al director del juego. Si decide solicitarlo y el director del juego se lo niega, el equipo pierde un turno de tiro. Si decide no solicitarlo, continúa el siguiente equipo y podrá esperar su próximo turno. El director del juego decide si le aprueba o le niega el diagrama basado en el siguiente orden (desconocido por los participantes): primero se deben conseguir el grafo conceptual y el modelo del dominio en cualquier orden; una vez se han conseguido los dos anteriores, se pueden conseguir el diagrama de procesos, el diagrama de objetivos y el diagrama causa-efecto en cualquier orden; cuando se tengan los cinco diagramas descritos se pueden tener la interfaz y el diagrama de casos de uso en cualquier orden; finalmente, cuando ya se tengan los siete diagramas anteriores, se puede obtener el diagrama de clases. Cuando niega un diagrama, el director del juego no debe entregar ninguna explicación en relación con la negación. Los diferentes equipos deberán ir descubriendo la secuencia con base en la información que logren recolectar de los turnos sucesivos propios y ajenos.
- Casilla de diagramas ajenos: Son los mismos diagramas del caso anterior, pero con un color diferente al equipo del que cae en la casilla. En este caso, el equipo que cae en la casilla puede capturar el diagrama en el que cae, siempre y cuando se encuentre en la zona de diagramas disponibles. Si hay varios equipos en contienda y los diagramas correspondientes están disponibles, el equipo puede seleccionar cuál de ellos captura.
- Pierde un turno: El equipo que caiga en esta casilla pierde el próximo turno de tiro.
- Escoja casilla de diagrama: En este caso, el equipo beneficiado podrá elegir una casilla de diagrama propio o ajeno y actuar en consecuencia como si hubiera caído en una casilla de diagrama propio o ajeno.
- Libera diagrama: El equipo que caiga en esta casilla puede elegir entre los diferentes diagramas que se encuentren en la zona de capturados de otro equipo para pasarlos a la casilla de diagramas disponibles. Cabe anotar que el diagrama liberado sólo podrá pasar a la zona de diagramas del equipo cuando en un turno vuelva a caer en la casilla respectiva.

- Se continúan asignando turnos, regidos por la información de las casillas del numeral anterior, hasta tanto uno de los equipos consiga tener en la zona correspondiente la totalidad de los diagramas de su color. De esta forma, será declarado vencedor del juego.
- El juego no tiene una duración definida; por ello, se podría pactar la finalización en cualquier momento por parte del director del juego, ya que el azar puede hacer que muchos los diagramas se capturen repetidamente e incluso pueden existir situaciones en las cuales todos los equipos se encuentren bloqueados porque sus diagramas claves se encuentren capturados. Si el director del juego pacta con los equipos la terminación anticipada del mismo, el ganador se determinará asignándole dos puntos (2) a cada diagrama propio que posea, un punto (1) por cada diagrama ajeno que tenga capturado y un punto negativo (-1) por cada diagrama que se encuentre en la zona de capturados de otro equipo. En la figura 6 se ejemplifica una posición específica del juego. El equipo azul no tiene diagramas propios y tiene capturado el grafo conceptual azul; el equipo verde no tiene diagramas propios ni capturados. El equipo rojo no tiene diagramas propios y tiene capturado el diagrama de clases azul y el equipo negro posee un diagrama propio y no tiene diagramas capturados. Si el juego terminara en esta posición específica, el ganador sería el equipo negro con 2 puntos, con segundo lugar del equipo rojo con 1 punto, tercer lugar del equipo azul con 0 puntos y cuarto lugar del equipo verde con -1 punto.

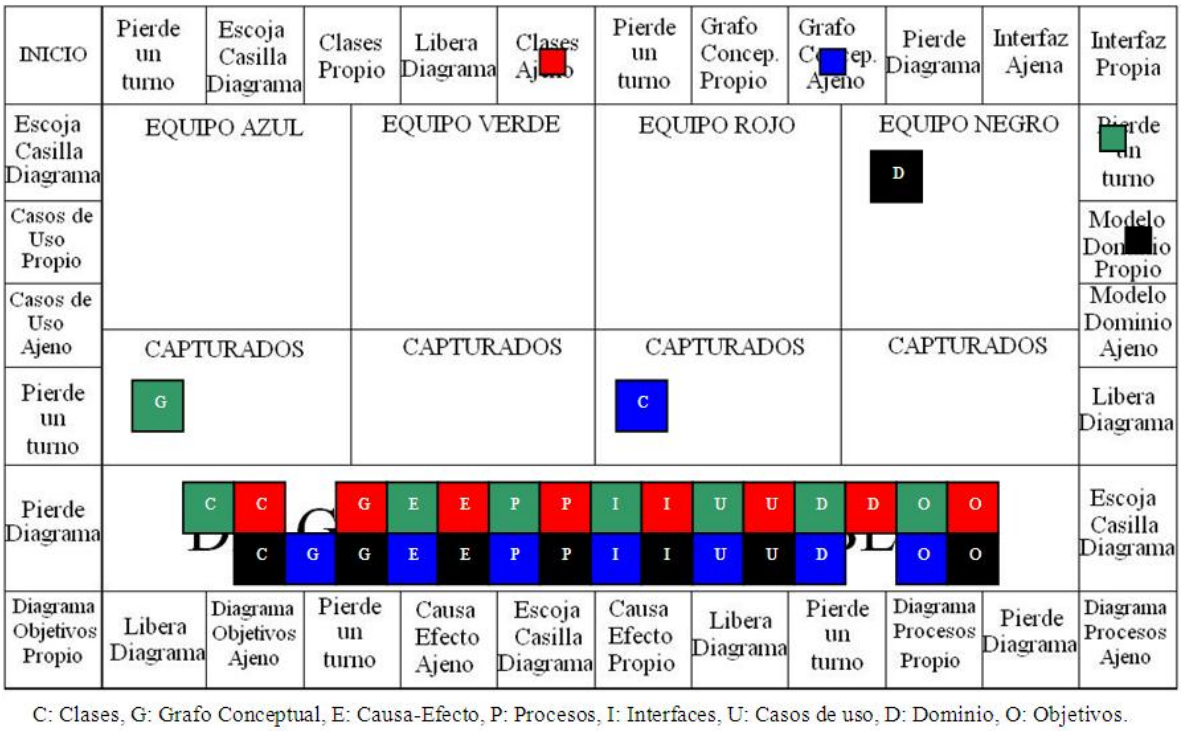

Figura 6. Una posición específica en el juego de UN-Método

5.3. El juego de la consistencia

5.3.1. Objetivo del juego

Llenar correctamente las plantillas predefinidas de cuatro diagramas (esquema preconceptual, diagrama de clases, diagrama de casos de uso y diagrama de secuencias), correspondientes al modelo verbal de un problema específico. Se debe usar una cantidad también predefinida de palabras para los diagramas correspondientes. Como una observación importante, se debe notar que las plantillas corresponden a una de las muchas maneras en las cuales se podría traducir el modelo verbal entregado. Sin embargo, por efectos de simplicidad en las reglas del juego, se limita a este único juego de plantillas para facilitar el proceso de conteo de puntos y determinación del ganador.

5.3.2. Materiales

- Modelo verbal: Es un discurso en un lenguaje controlado que representa el manejo de una máquina que fabrica y vende café. El discurso es el siguiente:

> *El cliente entrega dinero que se recibe en la máquina.*
> *El cliente escoge el azúcar y escoge el producto, el cual tiene tres tipos que son: tinto, café y capuchino.*
> *Cada producto tiene un precio.*
> *Cuando el cliente haya escogido el producto y el azúcar y cuando haya entregado el dinero, la máquina prepara el producto.*
> *La máquina entrega el cambio, imprime el recibo y realiza el reporte que incluye la cantidad (clasificada en dinero y producto).*
> *El operario solicita el reporte y realiza el mantenimiento que se aplica a la máquina.*

- Plantilla de esquema preconceptual: Es uno de los esquemas preconceptuales que, presumiblemente, se obtendrá a partir del modelo verbal, y en el cual sólo uno de los conceptos se etiqueta (mantenimiento); los demás elementos (conceptos y relaciones) se dejan en blanco. En la figura 7, se muestra la plantilla del esquema preconceptual.
- Plantilla de diagrama de clases: Al igual que en el caso anterior, esta plantilla posee uno de los diagrama de clases que se espera resulte del modelo verbal. En este caso, también se ubican algunos elementos a modo de pistas para la construcción del diagrama (cliente, máquina y operario). En la figura 8, se muestra la plantilla del diagrama de clases.

- Plantilla de diagrama de casos de uso: Esta plantilla posee los elementos de uno de los presuntos diagramas de casos de uso que surge del modelo verbal y se puede apreciar en la figura 9.
- Plantilla de diagrama de secuencias: Se puede apreciar en la figura 10.

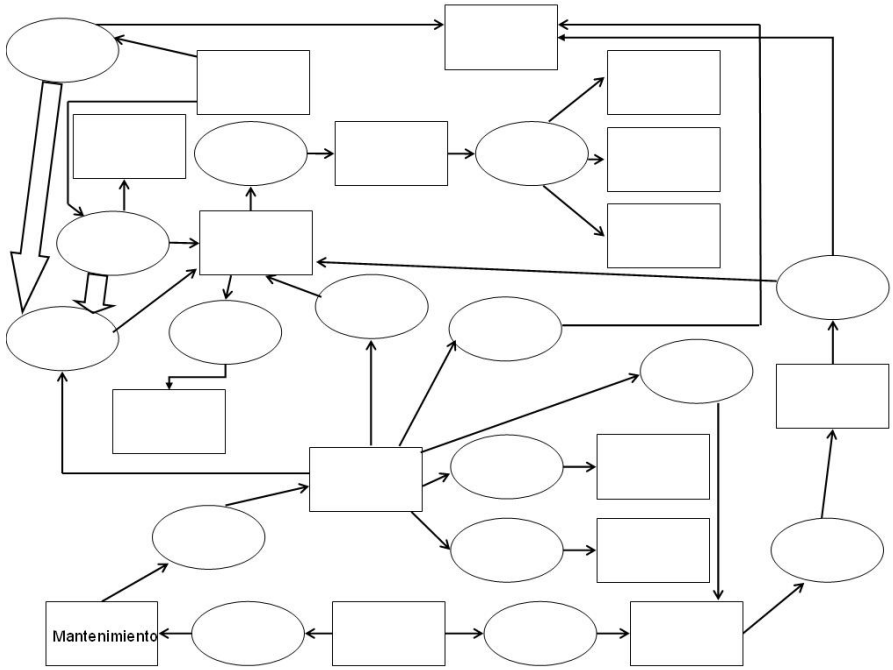

Figura 7. Plantilla de esquema preconceptual del juego de la consistencia.

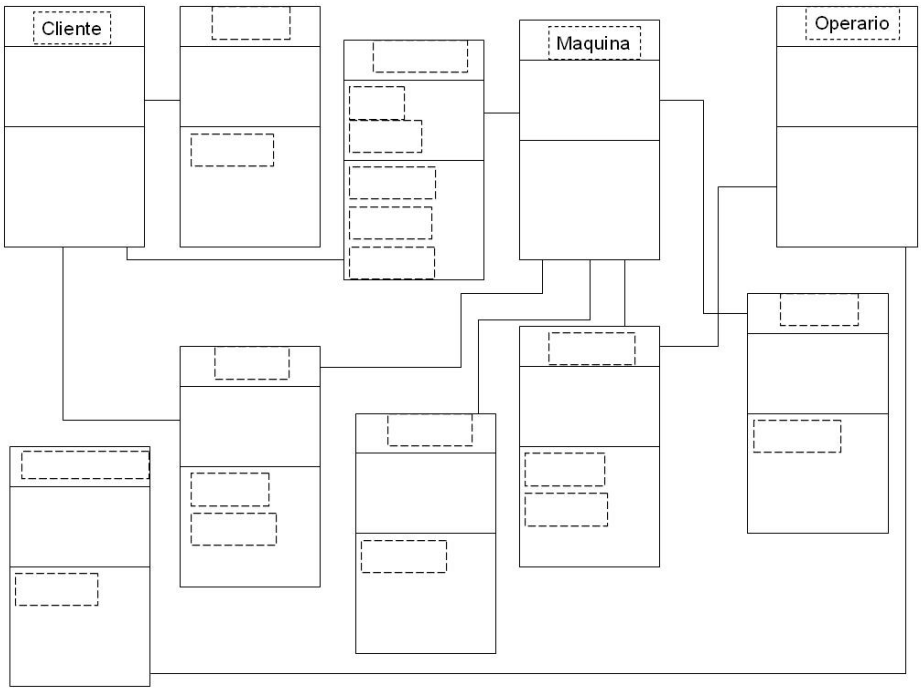

Figura 8. Plantilla del diagrama de clases del juego de la consistencia.

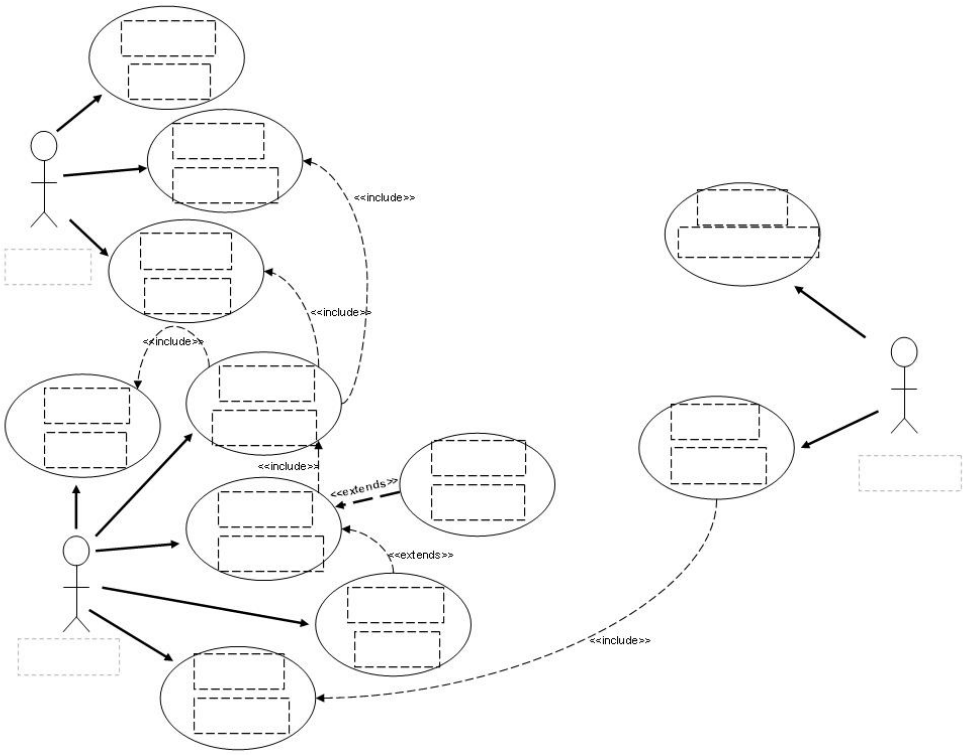

Figura 9. Plantilla del diagrama de casos de uso del juego de la consistencia.

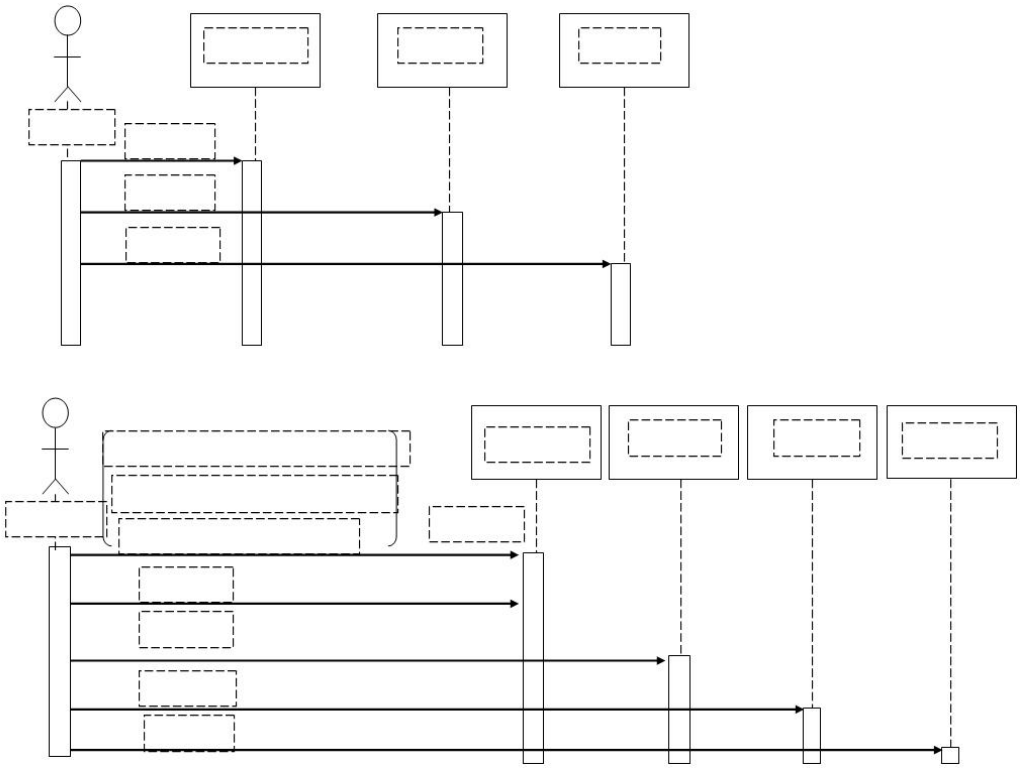

Figura 10. Plantilla del diagrama de secuencias del juego de la consistencia.

- Lista de palabras: es el conjunto de palabras que aparecen en el modelo verbal incluyendo una casilla por cada aparición en alguna de las plantillas de diagramas. Constituye una hoja de control para el desarrollo del juego y se puede apreciar en la figura 11.

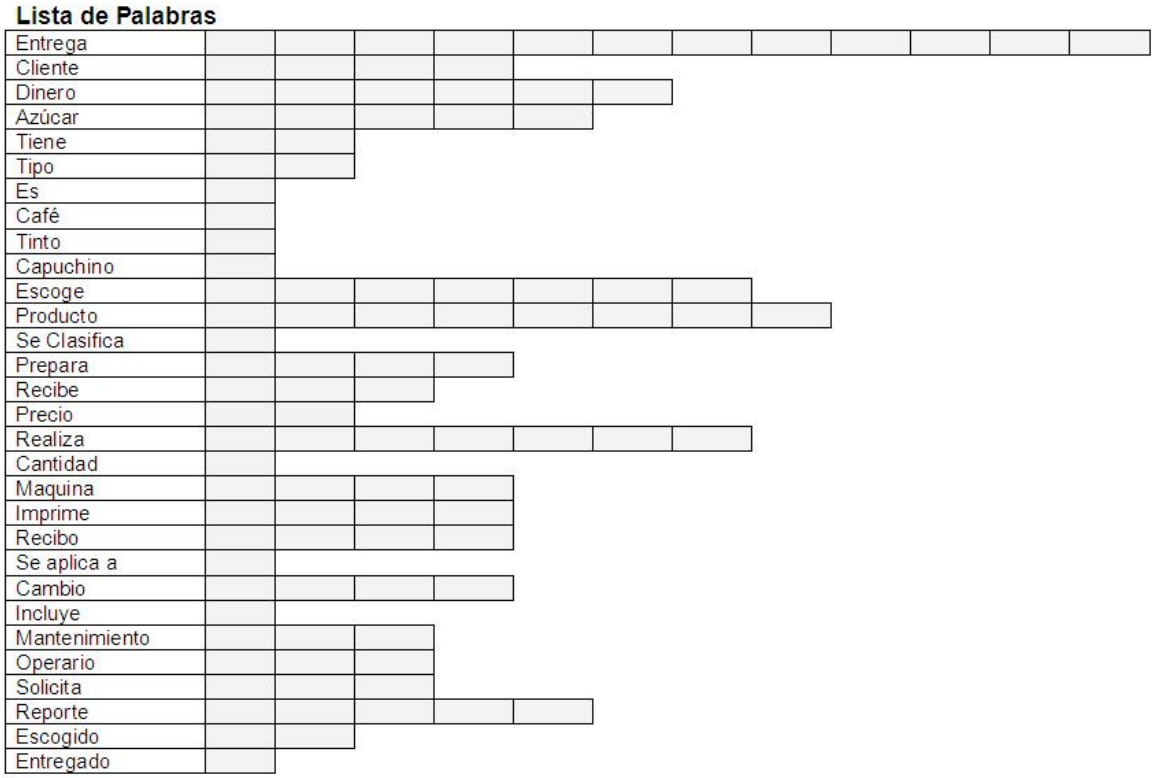

Figura 11. Lista de palabras del juego de la consistencia.

5.3.3. Reglas de juego

- Los jugadores se distribuyen en equipos de cinco a ocho personas.
- A cada grupo, se le hace entrega de un modelo verbal, cuatro plantillas de diagramas como se muestra en las figuras 7–10 y un listado de palabras como el correspondiente a la figura 11.
- El director del juego anuncia el inicio de un tiempo de 75 minutos, en los cuales los equipos se deben dedicar a completar los diagramas en el orden que deseen, pero llenando la lista de palabras con las posiciones de las diferentes palabras en cada diagrama. No se pueden usar más casillas de las que aparecen en la lista de palabras. En cualquier momento antes de finalizar el tiempo correspondiente, los equipos pueden entregar sus resultados.
- Una vez finalizado el tiempo, el director del juego recoge los diagramas y determina el ganador del juego, asignando los puntajes que se consignan en la tabla 2. En caso de existir empate en los puntajes, el ganador será quien primero haya entregado los diagramas correspondientes.

Diagrama	**Puntos**
Esquema preconceptual	+1 punto por cada elemento correcto. -1 punto por cada elemento incorrecto.
Caso de uso	+1 punto por cada elemento correcto. -1 punto por cada elemento incorrecto.
Diagrama de clases	+3 puntos por cada clase correcta. -3 puntos por cada clase incorrecta. +2 puntos por cada operación correcta. -2 puntos por cada operación incorrecta. +1 punto por cada atributo correcto. -1 punto por cada atributo incorrecto.
Diagrama de secuencia	+1 punto por cada elemento correcto. -1 punto por cada elemento incorrecto.

Tabla 2. Asignación de puntajes por la elaboración de diagramas del juego de la consistencia.

En la figura 12, se muestran las respuestas correspondientes a los cuatro diagramas.

Cabe anotar que existen en los diagramas algunas posiciones intercambiables y que no afectan la puntuación. Esas posiciones son las siguientes:

- Esquema preconceptual:
    - El orden en las casillas café, tinto y capuchino es indiferente.
    - El conjunto conformado por la relación *imprime* y el concepto *recibo* es intercambiable con el conjunto conformado por la relación *entrega* y el concepto *cambio*.
- Diagrama de clases:
    - La clase *cambio* se puede intercambiar con la clase *recibo*.
    - Cuando hay más de una operación en una clase, el orden de esas operaciones no es importante.
    - Igual ocurre con los atributos.
- Diagrama de casos de uso:
    - El caso de uso *escoge producto* es intercambiable con el caso de uso *escoge azúcar*.
    - El caso de uso *entrega cambio* es intercambiable con el caso de uso *imprime recibo*.

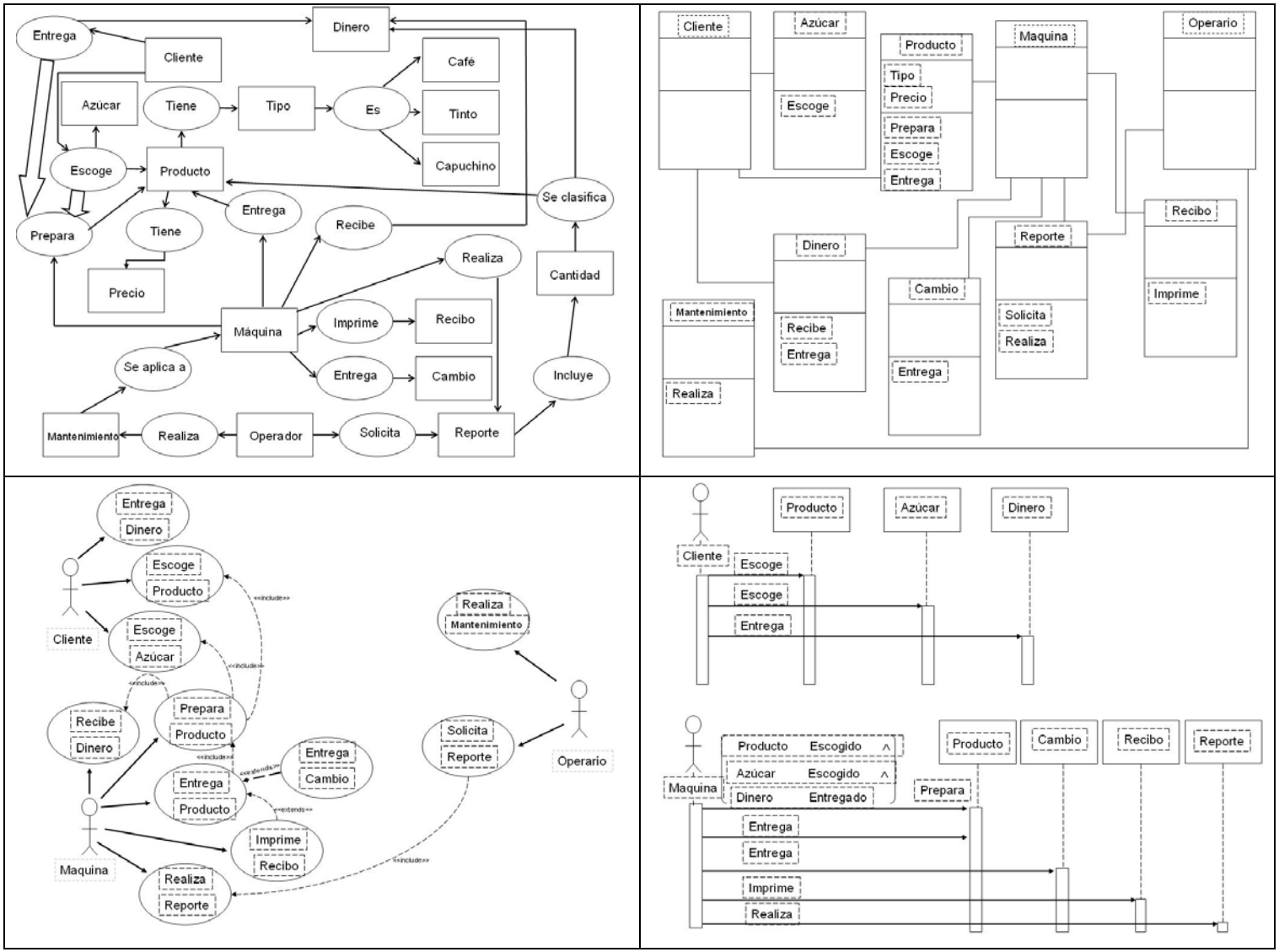

Figura 12. Soluciones a la elaboración de diagramas del juego de la consistencia.

- Diagrama de secuencias:
    - Las posiciones de las condiciones de guarda *producto escogido*, *azúcar escogido* y *dinero entregado* son intercambiables.
    - En el diagrama de secuencias del cliente, los tres objetos tienen posiciones intercambiables.
    - En el diagrama de secuencias de la máquina, los objetos *producto* y *cambio* son intercambiables.

5.4. El juego de la planeación de proyectos de *software*

5.4.1. Objetivo del juego

Obtener la mejor planeación temporal posible (expresada mediante un diagrama de Gantt) para tres proyectos cuyas actividades ya se han estimado y tienen tiempos fijos de duración. Con base en la planeación obtenida, realizar un seguimiento a la ejecución de esa planeación hasta la culminación de las tareas, en un ambiente simulado que toma en consideración el azar y la competencia.

5.4.2. Materiales

- Datos de los proyectos: Se tienen tres proyectos de *software*, P1, P2 y P3, los cuales poseen ya estimaciones del tiempo (en horas-hombre) que se gastarán en las fases de análisis, diseño y construcción. Dicha información se consigna en la tabla 3.

PROYECTO	ANÁLISIS	DISEÑO	CONSTRUCCIÓN
P1	14	3	9
P2	5	12	6
P3	2	10	7

Tabla 3. Estimación de la duración en horas-hombre de las actividades de tres proyectos de *software*.

- Plantilla para la elaboración de la planeación temporal: Es una cuadrícula que se entrega a los diferentes equipos para la elaboración de los diagramas de Gantt correspondientes, tal como se muestra en la figura 13.

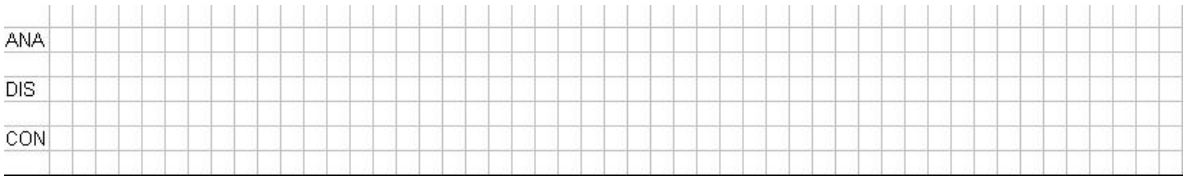

Figura 13. Plantilla para la elaboración de la planeación temporal.

- Cuadro de seguimiento a las planeaciones temporales: Es una hoja en Excel® que puede reemplazarse con el dibujo de las planeaciones en el tablero o con hojas de seguimiento de las diferentes planeaciones. Este cuadro se puede visualizar en la figura 14. La ejecución de la planeación se registra debajo de la barra correspondiente a una actividad particular; cada cuadro en blanco que queda por debajo de dichas barras es una hora-hombre de ejecución de la planeación y en el desarrollo del juego se van llenando con el procedimiento abajo indicado hasta que no quede ningún cuadro vacío por debajo de una actividad.

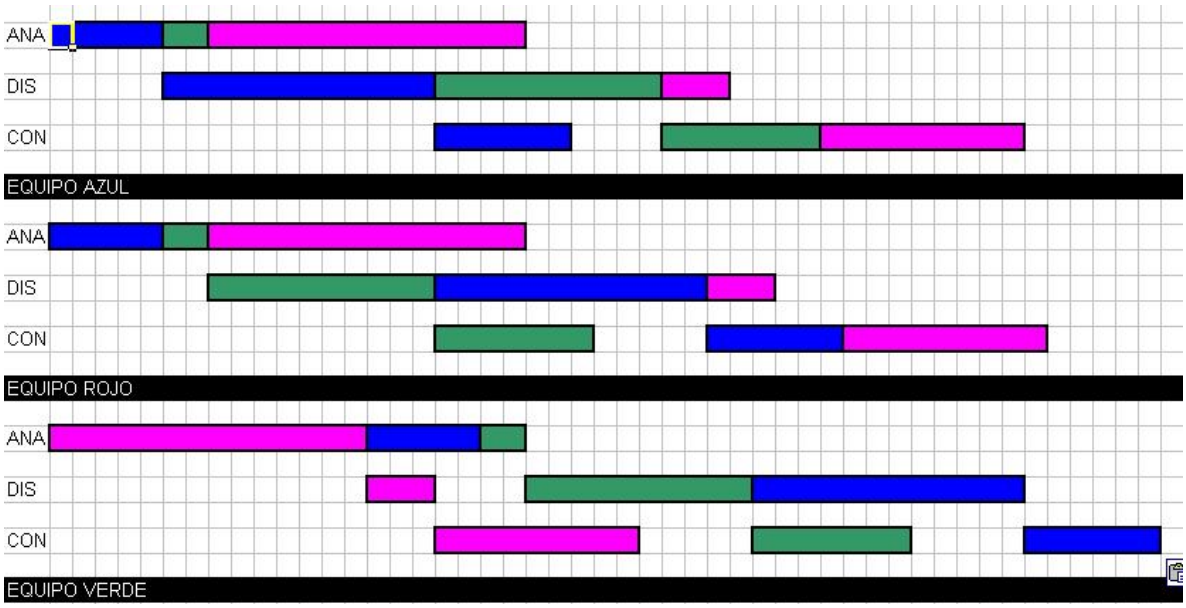

Figura 14. Cuadro de seguimiento a las planeaciones temporales.

- Programa de simulación de dados: En la figura 15, se puede apreciar la interfaz del programa de simulación de dados que se emplea para el juego. Este programa puede sustituirse con dados convencionales en los que se fijen las opciones de acuerdo con los valores que arrojen los dados.

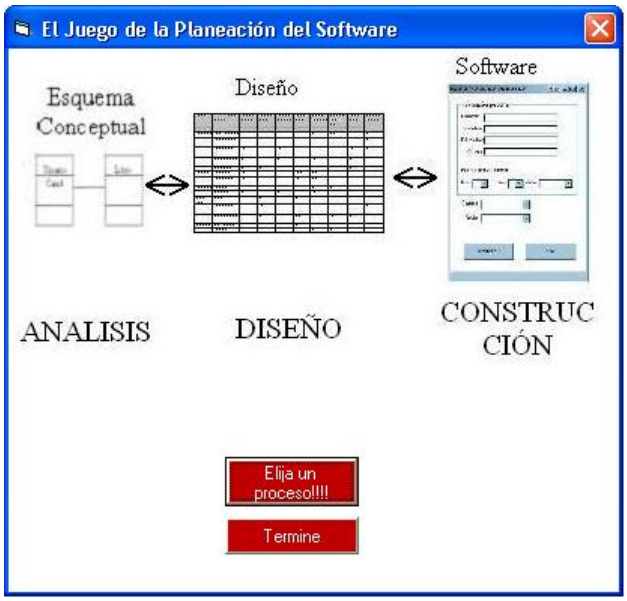

Figura 15. Programa de simulación de dados para el juego de la planeación.

5.4.3. Reglas de juego

- Se dividen los jugadores por equipos de 4 a 8 jugadores.
- Se entrega a cada equipo una plantilla para la elaboración de la planificación temporal, mostrándoles los datos de los proyectos que se consignan en la tabla 3. Se debe tomar en consideración que, para efectos de planeación, para los tres proyectos se restringe el personal a un analista, un diseñador y un programador; además, las fases se deben ejecutar en orden (análisis, diseño y construcción) y para iniciar una fase en algún proyecto, es necesario haber terminado la anterior. La plantilla diligenciada se debe entregar en los 10 minutos siguientes.
- Al finalizar los 10 minutos, el director del juego recoge las planeaciones temporales resultantes para cada equipo. En caso de existir errores en alguna planeación, ésta se sustituirá con la peor planeación posible, que el director trae ya predefinida en el cuadro de seguimiento a las planeaciones temporales.
- Se determina cuáles fueron la mejor y la peor planeación, calculando la cantidad de horas-hombre totales que se requieren para completar los dos proyectos. A la planeación con el menor

tiempo total se le entrega como premio el llenado de horas-hombre en ejecución de su proyecto, en una cantidad que se calcula con la diferencia de horas-hombre entre la peor y la mejor planeación.

- El equipo con la mejor planeación lanza los dados inicialmente. Las diferentes opciones de resultado son:

    - Análisis M horas-hombre: En este caso, se incrementan en el seguimiento M casillas, siempre y cuando no se exceda el máximo de horas-hombre que tienen los tres proyectos.
    - Diseño N horas-hombre: En este caso, se pueden incrementar en el seguimiento N casillas, siempre y cuando no se exceda el máximo de horas-hombre que tienen los tres proyectos y se haya terminado el análisis de cada uno de los proyectos. Si alguno de los proyectos no ha terminado la fase de análisis, el seguimiento correspondiente al diseño no se puede incrementar.
    - Construcción W horas-hombre: La mecánica del juego es la misma, sólo que, en este caso, además del análisis, el diseño de un determinado proyecto debe estar concluido antes de iniciar la construcción.
    - Devuelve X horas-hombre: Como su nombre lo indica, consiste en devolver X casillas en el seguimiento de la planeación. La decisión se deberá tomar acorde con las reglas enunciadas, es decir, no se podrán devolver casillas en análisis si ya, en un determinado proyecto, se había iniciado el diseño. La decisión de cuáles casillas devolver corre a cargo de uno de los grupos competidores, seleccionado por el director del juego.
    - Contratación adicional sobre un proceso Y horas-hombre: En este caso, únicamente sobre un proceso, se pueden incrementar Y casillas sobre el seguimiento, sin exceder el total de ese proceso.
    - Contratación adicional sobre dos procesos Z horas-hombre: En este caso, se pueden repartir las Z horas en seguimiento sobre dos procesos. El grupo debe decidir la forma de repartición entre los procesos.
    - Pierde un turno: Al equipo que le corresponda esta opción, se le quitará un turno de lanzamiento de los dados.

- Continúan los demás equipos lanzando los dados y ejecutando las opciones mencionadas.
- El juego termina cuando alguno de los equipos participantes logre terminar el seguimiento a la planeación que realizó. Si por efectos del azar la culminación del juego se está retrasando, es

posible pactar una terminación anticipada, por ejemplo, limitando en tiempo la extensión del juego. En ese caso, el ganador del juego será aquel equipo que tenga la actividad más adelantada en su planeación temporal; si se presenta un empate, se premiará a la planeación más corta y si aún subsiste el empate, se discernirá por medio de los dados. En la figura 16, se ejemplifica una posición cualquiera en medio del juego, tomando en cuenta que las casillas de ejecución se ubican por debajo de cada una de las barras. En dicha figura, el equipo azul está avanzando sobre el diseño de uno de los proyectos, mientras que ya terminó el análisis de dos proyectos y el equipo rojo está en el diseño el segundo de sus proyectos con el análisis de los dos primeros proyectos ya concluido. Finalmente, el equipo verde aún no ha concluido el análisis del primero de sus proyectos, lo que no le permite avanzar en diseño y construcción. En este caso, el equipo ganador será el rojo, por tener la actividad más adelantada en diseño.

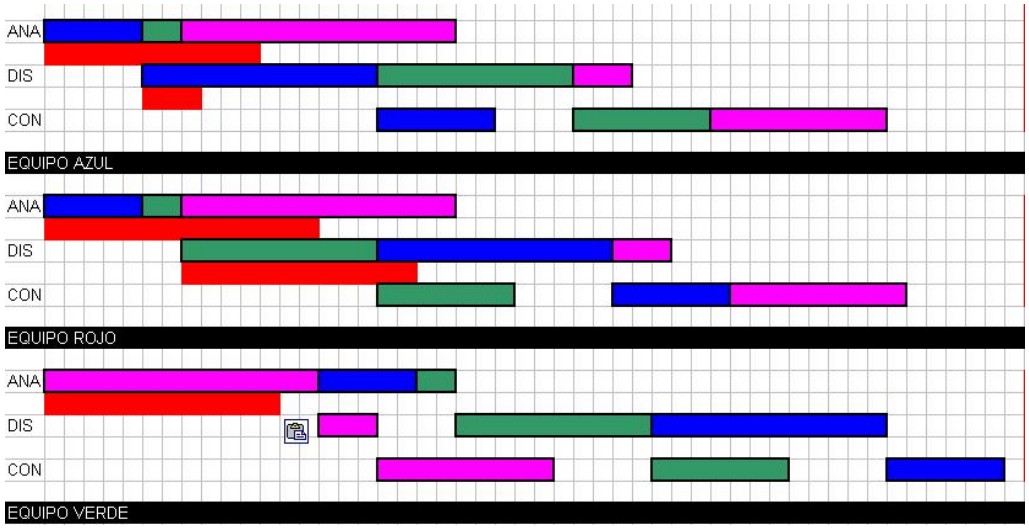

Figura 16. Una posición específica en el juego de la planeación de proyectos de *software*.

5.5. El juego de la trazabilidad y la comunicación.

5.5.1. Objetivo del juego

Culminar correctamente, y en el menor tiempo posible, un conjunto de artefactos para el desarrollo de una aplicación de *software* (diagramas de casos de uso y clases y una interfaz gráfica de usuario) relativos al popular juego juvenil de cartas Yu-gi-oh!®. En este juego, al igual que en el juego de la consistencia, los diagramas que se presentan son tan sólo un conjunto de los posibles diagramas con que se podría modelar el problema.

5.5.2. Materiales

- Tablero de juego: Es una cuadrícula de filas y columnas etiquetadas como se muestra en la figura 17. El material de fabricación puede ser papel, cartón, madera, etc.

	A	B	C	D	E	F	G
1							
2							
3							
4							
5							
6							
7							
8							
9							

Figura 17. Tablero del juego de la trazabilidad y la comunicación.

- Fichas de juego: Son los diferentes elementos de los diagramas, que se recortan por separado para ubicarse encima del tablero de juego. Las fichas de juego son de varios tipos, dependiendo del artefacto a que se refiera:

    - Diagramas de casos de uso: actores, casos de uso, líneas de comunicación, líneas divisorias de sistema.
    - Diagrama de clases: nombre de clase, zona de atributos, zona de operaciones, líneas de asociación.
    - Interfaz gráfica de usuario: línea de encabezado, cartas de zona de monstruos, cartas de zona de magia, cartas de mano, cartas de mazo, cartas de cementerio.

- Especificaciones de los artefactos: Son las imágenes de los artefactos ya construidos y adaptados a una cuadrícula similar al tablero de juego. En las figuras 18 a 22, se muestran los cinco artefactos disponibles para realizar el juego de la trazabilidad y la comunicación.

5.5.1. Reglas de juego

- Los jugadores se dividen en equipos, en los cuales se debe nombrar un capitán y un líder.

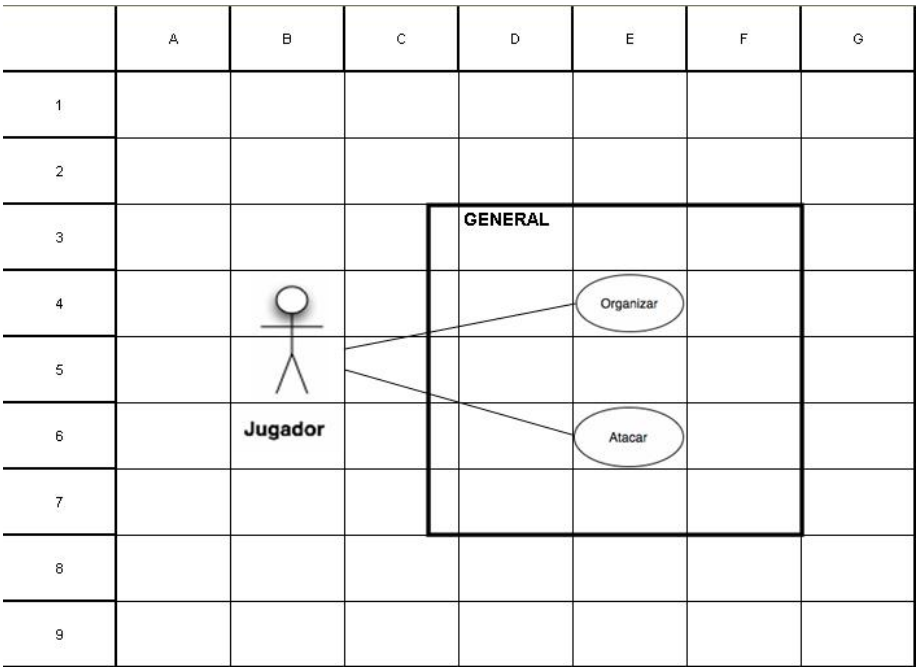

Figura 18. Artefacto caso de uso general para el juego de la trazabilidad y la comunicación.

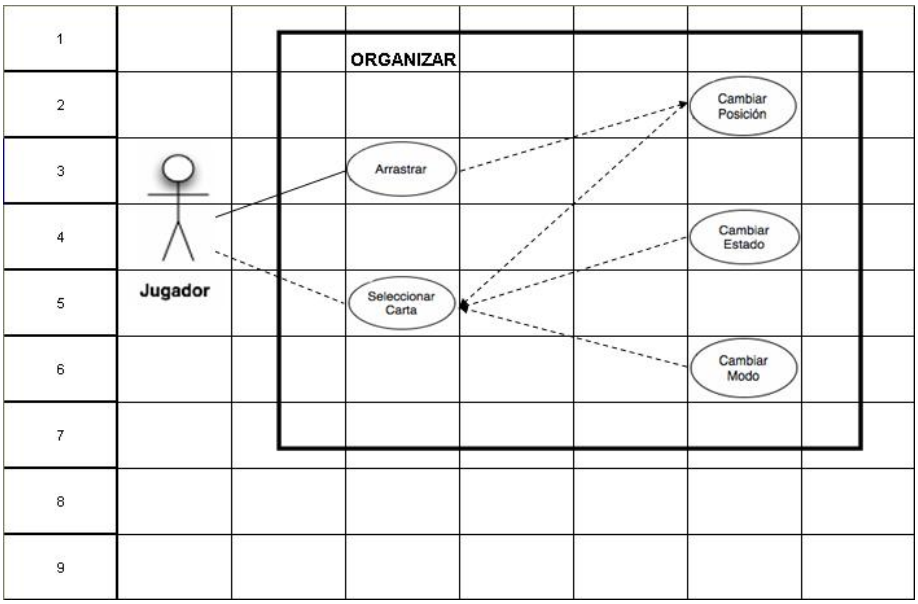

Figura 19. Artefacto caso de uso *organizar* para el juego de la trazabilidad y la comunicación.

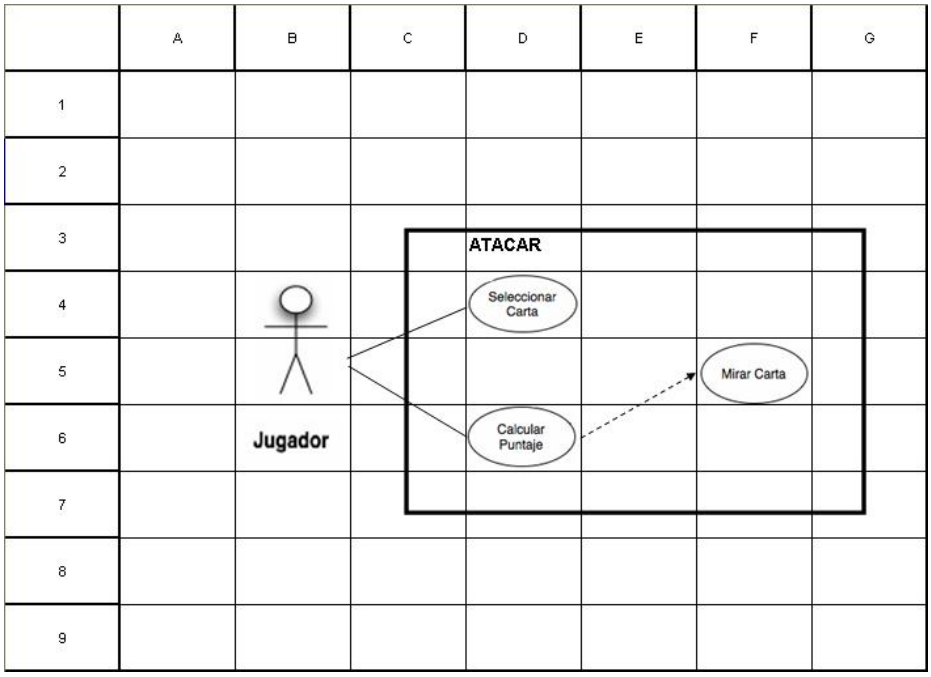

Figura 20. Artefacto caso de uso *atacar* para el juego de la trazabilidad y la comunicación.

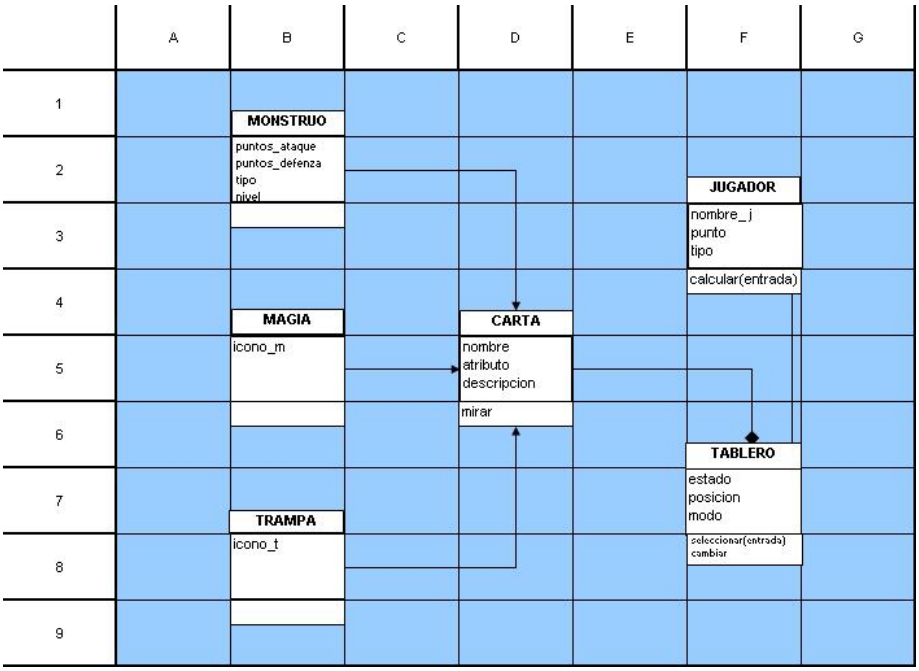

Figura 21. Artefacto diagrama de clases para el juego de la trazabilidad y la comunicación.

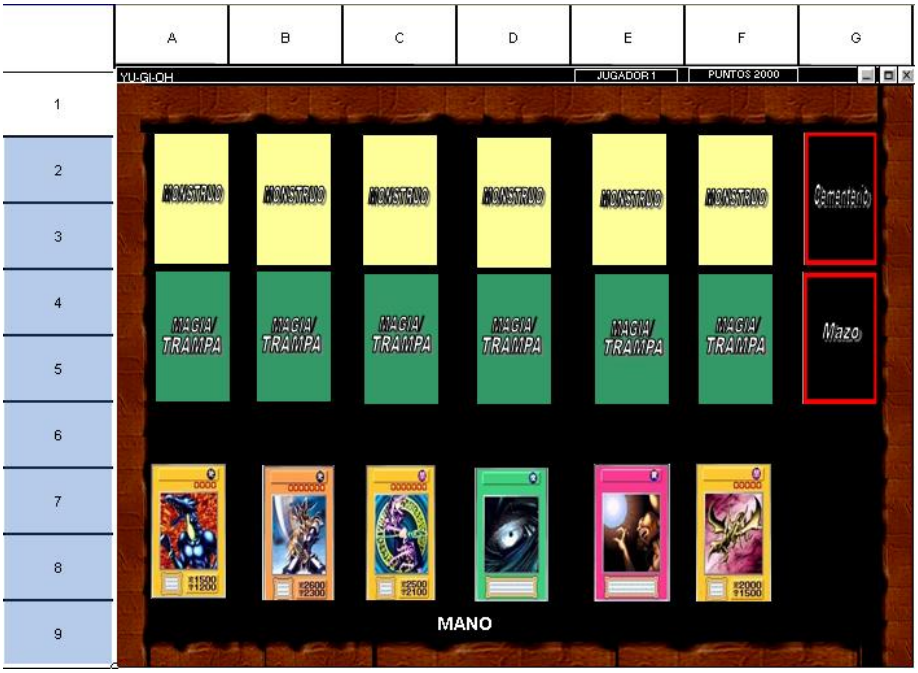

Figura 22. Artefacto interfaz gráfica de usuario para el juego de la trazabilidad y la comunicación

- El sitio donde se va a realizar el juego debe estar ordenado como se muestra en la figura 23. En la zona de diagramas, el director del juego debe ubicar la especificación de uno de los artefactos por cada iteración del juego. En la tienda, se ubican todas las fichas de juego (incluyendo las de todos los diagramas), sin ningún orden en particular. El tablero de juego se debe ubicar en una mesa localizada entre el capitán y el líder de cada equipo.
- La primera iteración del juego se inicia con la ubicación de la especificación del caso de uso general (véase figura 18) en la zona de diagramas por parte del director del juego.
- El director del juego autoriza el inicio del mismo e, inmediatamente, los capitanes se dirigen a la zona de diagramas para verificar la especificación. Los capitanes regresan y le comunican a sus respectivos líderes las fichas de juego que se requieren para completar el diagrama. Nótese que, por la posición de los jugadores en la zona de juego, el capitán no tiene contacto visual con el líder del grupo y todas sus instrucciones se transmiten oralmente. Este proceso se puede realizar las veces que consideren necesarias el capitán y el líder de cada grupo.
- Paralelamente, los líderes se van comunicando también oralmente con sus respectivos grupos, solicitando las diferentes fichas requeridas.

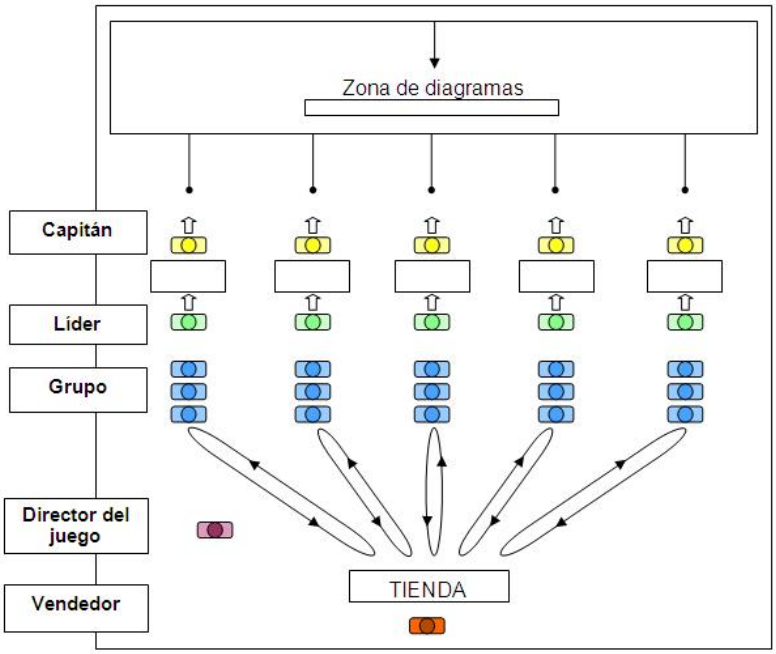

Figura 23. Distribución de la zona de juego

- Los integrantes del grupo se dirigen de a uno cada vez a la tienda, y retiran de allí una sola ficha de juego, que deben entregar al líder. Este proceso se repite hasta que todas las fichas se completen.
- Simultáneamente, el capitán y el líder de cada grupo van discutiendo las posiciones de las diferentes fichas en el tablero, por medio de las coordenadas en las cuales se encuentran ubicadas. Cuando el líder considera que el artefacto está listo en el tablero de juego de manera idéntica a como se encuentra en la especificación y, si ello se cumple, el equipo que primero termine habrá ganado la iteración. Es importante registrar el orden en que terminan los equipos para definir posteriormente al ganador general del juego.
- Se realizan iteraciones sucesivas del juego, ubicando las especificaciones de los demás artefactos (véanse las figuras 19 a 22). El equipo ganador será aquél que haya tenido mayor regularidad en el orden de terminación de los artefactos.

CAPÍTULO 6: RESULTADOS DE LA APLICACIÓN DE LOS JUEGOS

6.1. El juego del desarrollo de *software*

El juego se ha aplicado hasta ahora en 9 grupos diferentes de personas, con niveles de escolaridad muy diversos. El semestre de aplicación, el nombre del curso, la cantidad de participantes en cada caso y el nivel de escolaridad correspondiente a cada grupo se pueden consultar en la tabla 4.

código	Semestre	Asignatura	No. Parti-cipantes	Nivel de escolaridad	Preguntas respondidas
1	2005_01	Introducción a la Ingeniería de Sistemas	59	Estudiantes de primer semestre de Ingeniería de Sistemas e informática de la UNAL	1, 2 y 3
2	2005_01	Ingeniería de Requisitos	12	Estudiantes de sexto a noveno semestre de Ingeniería de Sistemas e Informática de la UNAL	1 y 2
3	2005_02	Introducción a la Ingeniería de Sistemas	43	Estudiantes de primer semestre de Ingeniería de Sistemas e informática de la UNAL	1 y 2
4	2005_02	Ingeniería de Requisitos	5	Estudiantes de sexto a noveno semestre de Ingeniería de Sistemas e Informática de la UNAL	1 y 2
5	2005_02	Inglés IV	10	Grupo heterogéneo de estudiantes, profesores y empleados de varias universidades.	1 y 2
6	2006_01	Introducción a la Ingeniería de Sistemas	90	Estudiantes de primer semestre de Ingeniería de Sistemas e informática de la UNAL	Ninguna
7	2006_01	Ingeniería de *Software*, Ingeniería Lingüística UNAM	14	Grupo combinado de estudiantes y profesores de Ingeniería de *Software* e Ingeniería Lingüística	1 y 2
8	2006_01	Tópicos Avanzados Ingeniería de *Software*	26	Estudiantes de maestría en Sistemas de primer semestre.	1 y 3
9	2006_03	Introducción a la Ingeniería de Sistemas	60	Estudiantes de primer semestre de Ingeniería de Sistemas e informática de la UNAL	Ninguna

Tabla 4. Características de los grupos de personas en los cuales se ha aplicado el juego del desarrollo de *software*.

Además de los grupos anotados, este juego se ha venido practicando de manera sistemática con estudiantes de primaria y bachillerato, en un proyecto de la Facultad de Minas denominado *El Jardín de la Ciencia*, que pretende la realización de actividades lúdicas en la Facultad que vayan promoviendo las competencias científicas de los estudiantes de primaria y bachillerato. Dado que el tiempo de desarrollo del juego es más corto para estos grupos, ellos no realizan actividades de retroalimentación. Para los demás grupos, el proceso de retroalimentación de los resultados del juego consistió en una encuesta en la cual los participantes respondieron algunas de las siguientes tres preguntas abiertas, que buscan centralizar los elementos reafirmados por el juego:

(1) ¿Qué aprendió del juego?
(2) ¿Qué cree que se necesitaba para ganar el juego?
(3) ¿Qué modificaciones le haría al juego?

En la tabla 4, se muestra también cuáles de las preguntas resolvieron los diferentes grupos. En el caso de los grupos 6 y 9, este proceso no se pudo llevar a efecto. Las preguntas son abiertas porque se requería conocer cuáles eran específicamente los principales aportes del juego. Respondieron la encuesta 169 participantes, de los cuales sólo los grupos 4 y 8 respondieron la tercera pregunta y únicamente el grupo 8 dejó de responder la segunda, puesto que se trataba con ellos de realizar un ensayo de modificación del juego, dada su condición de estudiantes de postgrado.

Las respuestas, posteriormente, se tabularon para determinar en cada caso los hallazgos en relación con el aprendizaje, los requisitos para ganar el juego o las respectivas modificaciones. Las respuestas se agruparon por su similitud y se determinó el porcentaje de participantes que las habían expresado, en relación con el número total de participantes. Por ello, el porcentaje de cada respuesta se encuentra entre 0 y 100%. En las tablas 5, 6 y 7 se pueden consultar respectivamente las respuestas a la primera, segunda y tercera preguntas.

Siendo éste un juego que se relaciona con la gestión de proyectos de *software*, para el análisis de los resultados se recurrió al modelo de las cuatro P's planteado por Pressman (2002), el cual establece que se deben tomar cuatro aspectos en consideración cuando se trata de este tema: *Personal, Producto, Proceso* y *Proyecto*. Desde esta óptica, y coincidiendo con Pressman en su apreciación de la gestión eficaz de un proyecto de *software*, el juego fue visto por los participantes como un juego relacionado con el aspecto *Personal*, ya que de las cinco primeras respuestas en relación con el aprendizaje que deja el juego, tres son relativas a este aspecto (véase la tabla 5). En efecto, el 48%

de los participantes consideró que la necesidad de dividir las tareas por especialidades de los integrantes de una empresa es la principal enseñanza que deja el juego. Si bien la simulación se realiza con un producto tangible (las cajas de *origami*) para representar un producto intangible como es el *software*, esta experiencia permite recrear mediante el juego el comportamiento de una empresa de *software* real, que suele poseer diferentes especialidades: analistas, diseñadores, programadores, directores de proyectos, gerentes, etc. El 40% de los participantes señaló la importancia del trabajo en equipo como otro de los aprendizajes del juego y el 28% señaló además que es vital que los equipos de trabajo posean una buena comunicación interna, otro de los aspectos básicos del desarrollo de *software* de calidad, y uno de los elementos que más se trata de impartir en la enseñanza de la Ingeniería de *Software*.

APRENDIZAJES	%	Aspecto
División de tareas por especialidades	48	Personal
Planeación previa de costos y tiempos	45	Proyecto
Trabajo en equipo	40	Personal
Comunicación entre los integrantes del equipo	28	Personal
Comunicación permanente con el cliente	21	Proceso
El producto final debe cumplir con los requisitos	20	Producto
Comprensión del problema desde el principio	17	Proceso
Revisión constante del proceso y la calidad	14	Proceso
Producir poco y bien es preferible a mucho y mal	11	Producto
Trabajo bajo presión	8	Proyecto
Algo de *origami*	8	Personal
Se deben conocer las diferentes etapas del proceso de construcción	7	Proceso
La capacitación se puede lograr con la práctica en el trabajo	6	Personal
La experiencia (ejecución de trabajos previos) es importante al construir un *software*	5	Proyecto
Se debe contar con una buena estrategia	3	Proyecto
Es importante contar con asesoría externa	1	Proyecto

Tabla 5. Resumen de respuestas a la pregunta “¿Qué aprendió del juego?”.

Ahora, el 45% de los participantes aprendió del juego la importancia de una buena planeación de costos y tiempos antes de iniciar cualquier labor de desarrollo. Ese aspecto, perteneciente al aspecto *Proyecto* de las cuatro P’s está, sin embargo, altamente correlacionado con el aspecto *Personal*,

puesto que para realizar este tipo de estimaciones, según lo señalaron los participantes, es necesario conocer muy bien las capacidades de los integrantes del equipo de trabajo, para poder establecer la capacidad real de producción de la empresa.

El último elemento de los cinco primeros, mencionado por el 21% de los participantes, es relativo al aspecto *Proceso* y es la comunicación permanente que debe mantenerse con los interesados del proceso de desarrollo. Dentro del currículo de la Ingeniería de Sistemas e Informática de la Escuela de Sistemas de la Universidad Nacional de Colombia, se hace énfasis de este elemento en el curso de Ingeniería de Requisitos, pero debe conocerse y practicarse por parte de todo buen Ingeniero de *Software*, si bien es fundamental en la determinación de las necesidades y expectativas de los interesados.

La sexta enseñanza propuesta por el 20% de los participantes se relaciona con el cumplimiento de los requisitos por parte del producto mismo, un asunto que se considera de vital importancia para que el *software* sea bien recibido por parte de los interesados. Este es uno de los objetivos últimos de la Ingeniería de *Software*, puesto que, si la aplicación de *software* desarrollada no se adapta a las necesidades y expectativas de los interesados, comienzan a surgir problemas de mantenimiento que finalmente harán fracasar el producto.

Con una menor incidencia, los participantes mencionaron los siguientes temas:

- Personal: La capacitación de quienes están realizando el *software* es también importante para la obtención de un buen resultado y puede lograrse con la realización de diferentes aplicaciones de *software*. Los participantes también mencionaron en este punto la necesidad de captar rápidamente el proceso de elaboración del producto, lo que desde el punto de vista de la Ingeniería de *Software* plantea un vacío en el conocimiento, puesto que la formación del personal en esta área debería ser muy intensiva en métodos de desarrollo y habilidades para adaptarse a diferentes métodos.
- Producto: Los participantes resaltaron también la necesidad de que la calidad se respete, así no sea a la velocidad requerida. La celeridad próxima a la entrega de una aplicación de *software* puede hacer que se ignoren los requisitos de los interesados, lo que, finalmente, desemboca en problemas posteriores a la entrega.
- Proceso: Los participantes señalaron en orden de importancia otros tres asuntos. El primero se relaciona con la comprensión del problema desde el principio; en este asunto, muchos de las opiniones se centraron en la necesidad de dedicar una buena parte del tiempo inicial del juego a

entender cómo era el producto que se debía entregar, y éste es uno de los principios fundamentales de la Ingeniería de Requisitos. El segundo asunto es la revisión constante del proceso por parte de los empleados administrativos. El tercer asunto es el conocimiento de los procesos de construcción. Los dos últimos asuntos se suelen relacionar con las fases de implementación y mantenimiento, tan necesarias en el desarrollo de una buena aplicación de *software*.

- Proyecto: En este caso, los participantes expresaron el trabajo bajo presión, la experiencia previa, la estrategia y la asesoría externa como temas suplementarios a la planeación. La experiencia del juego, en este sentido, ayudó a visualizar algunas de las variables que se suelen presentar en el desarrollo de proyectos reales de *software*.

En relación con lo que se necesitaba para ganar el juego, resumidos los aspectos correspondientes en la tabla 6, los participantes expresaron respuestas consistentes con los aprendizajes.

NECESIDADES PARA GANAR EL JUEGO	%	Aspecto
Dividir las tareas por habilidades	57	Personal
Planear adecuadamente tiempos y costos	48	Proyecto
Conocer y respetar los requisitos del cliente	34	Proceso
Mantener una buena comunicación interna	27	Personal
Capacitación y aprendizaje del *origami*	25	Personal
Tener un buen director	22	Personal
Conocer el proceso de desarrollo	21	Proceso
Hacer seguimiento	15	Proceso
Calma	10	Proyecto
Motivación	8	Personal
Tiempo adicional	3	Proceso
Herramientas más precisas (reglas, tijeras, etc.)	3	Proyecto
Confianza en las propias capacidades	1	Personal

Tabla 6. Resumen de respuestas a la pregunta "¿Qué cree que se necesitaba para ganar el juego?".

Mayoritariamente, los participantes votaron por la división de las tareas según las habilidades del personal a cargo (57%) y por la planeación de los tiempos del juego y los costos en que debía incurrir la empresa ficticia para la elaboración del proyecto (48%). Otras respuestas que no habían sido mencionadas en los aprendizajes fueron la necesidad de contar con un buen director (22% de

los participantes lo expresaron, aunque éste podría ser el resumen de algunas de las respuestas a esta pregunta, pues son labores del director la planeación y la selección del personal) y la capacitación y el aprendizaje del *origami* (25%), que se podría asimilar con las habilidades requeridas por el personal operativo para la realización del *software*, es decir, en el caso de la Ingeniería de *Software*, las habilidades de análisis, diseño y programación por parte de los involucrados en un proceso de desarrollo.

Ahora, en cuanto a las modificaciones, tan sólo la mitad de los participantes respondieron a esta pregunta, y, de estos, el 55% manifestó que no era necesario realizar modificaciones al juego. Ninguna de las modificaciones planteadas fue mayoritaria y expresaban visiones muy particulares de los participantes. Sólo un mayor conocimiento del *origami* y el incremento del tiempo de juego (ambos sugeridos por el 11% de los encuestados) podrían ser modificaciones con algún porcentaje de incidencia. Las demás opiniones se pueden consultar en la tabla 7.

MODIFICACIONES A LOS JUEGOS	%
Ninguna	55
Conocimiento previo del *origami*	11
Más tiempo de juego	11
Libertad para seleccionar el grupo de trabajo	9
Rotación de roles	6
Menos trabajo manual	5
Incrementar tamaño de los módulos	4
Permitir herramientas mejores (regla, tijeras)	2
Rebajar el costo de los materiales	2
Tiempo variable de análisis del problema	2
Permitir que el director haga trabajo operativo	2

Tabla 7. Resumen de respuestas a la pregunta “¿Qué modificaciones le haría al juego?”.

6.2. El juego de UN-Método

El juego se ha practicado hasta ahora con 4 grupos de participantes, cuyas características se resumen en la tabla 8. Como en el caso del juego anterior, la retroalimentación se consiguió mediante una encuesta a los participantes que incluía las siguientes preguntas:

(1) ¿Qué aprendió del juego?

(2) ¿Qué cree que se necesitaba para ganar el juego?

(3) ¿Qué modificaciones le haría al juego?

código	Semestre	Asignatura	No. Parti-cipantes	Nivel de escolaridad	Preguntas Respondidas
1	2005_01	Ingeniería de Requisitos	7	Estudiantes de sexto a noveno semestre de Ingeniería de Sistemas e Informática de la UNAL	1 y 2
2	2006_01	Ingeniería de *Software*, Ingeniería Lingüística UNAM	8	Grupo combinado de estudiantes y profesores de Ingeniería de *Software* y Lingüística Computacional	1 y 2
3	2006_01	Tópicos Avanzados Ingeniería de *Software*	29	Estudiantes de maestría en Sistemas de primer semestre.	1 y 3
4	2006_03	Ingeniería de Requisitos	18	Estudiantes de sexto a noveno semestre de Ingeniería de Sistemas e Informática de la UNAL	1 y 2

Tabla 8. Características de los grupos de personas en los cuales se ha aplicado el juego de UN-Método.

En este caso, únicamente el tercer grupo cambió la segunda pregunta por la tercera, pues este grupo debía realizar un análisis intensivo de las modificaciones requeridas por el juego.

Debido a las características del juego, nuevamente se recurrió al modelo de las 4 P's (Pressman, 2002) para realizar el análisis de las respuestas. En las tablas 9, 10 y 11, se compendian las respuestas a las tres preguntas por parte de los participantes. De la tabla 9, se desprende que, a diferencia del juego del desarrollo de *software*, los participantes percibieron el juego de UN-Método como un juego relativo al aspecto *Proceso*, pues, mayoritariamente, afirmaron que los principales aprendizajes del juego fueron las etapas del desarrollo de *software* (y, en consecuencia, la manera en que se elaboran secuencialmente los diferentes artefactos de un método de desarrollo), con un 60% de los participantes, y la utilidad de los artefactos que se elaboran en UN-Método, con un 52%. Este resultado es completamente previsible, puesto que el juego se diseñó como una experiencia

dedicada principalmente a reconocer la importancia y la secuencialidad de los artefactos que hacen parte de la elicitación de requisitos mediante UN-Método. En este sentido, la retroalimentación constante que suministra el juego en relación con las diferentes casillas de diagramas, aún sin hacer explícita la secuencia en que debían conseguirse los diagramas, entrega una herramienta metodológica que guía el aprendizaje.

APRENDIZAJES	%	Aspecto
Etapas del desarrollo de *software* (secuencia de elaboración de diagramas)	60	Proceso
Artefactos usados por UN-Método	52	Proceso
Comportamiento de la competencia	18	Proyecto
Toma de decisiones y estrategia	15	Personal
Comunicación entre los integrantes del equipo	15	Personal
Propósito de la Ingeniería de *Software*	13	Proceso
La suerte hace parte del desarrollo de *software*	8	Proceso
Planeación de un proyecto de *software*	5	Proyecto
Trabajo bajo presión	3	Proyecto

Tabla 9. Resumen de respuestas a la pregunta "¿Qué aprendió del juego?".

Ahora, un aspecto que se esperaría tuviera una mayor participación dentro de los aprendizajes del juego, como es la incidencia de la suerte en el resultado, paradójicamente, tiene un porcentaje muy bajo de opiniones por parte de los participantes (8%). Aparentemente, los participantes se preocuparon mucho más por tratar de establecer el orden en que debían obtener los diagramas e hicieron caso omiso al hecho de que el mecanismo controlador del juego eran los dados, símbolo de la aleatoriedad del mismo.

Un tercer nivel de importancia en el aprendizaje, aunque con muy bajo nivel de opiniones (18%) se relaciona con el comportamiento de la competencia. En el juego del desarrollo de *software*, el aprendizaje se relacionó mucho más con las capacidades internas de la organización, en tanto que, en el juego de UN-Método, los factores exógenos comenzaron a ser preponderantes. Los participantes que mencionaron este elemento, relativo al aspecto *Proyecto* del modelo de las 4 P's, señalaron el hecho de que, en la realidad, la competencia debe ser un factor a tomar en consideración, en especial porque los retrasos que se pueden tener en un proyecto y las dificultades

que puede afrontar una empresa de desarrollo de *software* pueden tener como raíz las prácticas desleales de los competidores.

Dentro de los factores del aspecto *Personal*, los participantes mencionaron como aprendizajes, con el mismo nivel de participación (15%), la toma de decisiones y la estrategia y la comunicación entre los integrantes del grupo. Sí era notorio, de la experiencia, que los equipos que más dialogaban y trataban de poner en común los conocimientos que podían recabar de las jugadas que hacían los contrincantes, eran los que más pronto se percataban de que existía una secuencia en la consecución de los diagramas. Además, se notó especialmente en la experiencia del grupo 4, un marcado apasionamiento por tratar de "perjudicar" a los otros grupos en lugar de conseguir los diagramas necesarios para el triunfo. Este aprendizaje, tristemente experimentado en el juego por este grupo de participantes, es, sin embargo, un refuerzo a una de las razones expuestas por Kasvi (2000) para el uso de los juegos como simulaciones de la vida real, como es la seguridad. En la realidad, este es un aprendizaje costoso, en especial para quienes son reclutados por pequeñas empresas, en las cuales se vive un ambiente de competencia con otras compañías, que puede ser incluso nocivo. Es importante que los futuros ingenieros reconozcan que este tipo de factores existe y que pueden perjudicar notoriamente las organizaciones de las que pueden hacer parte en el futuro.

De forma minoritaria, los participantes también expresaron otras opiniones en relación con los aprendizajes que se pueden extraer del juego, como son:

- Proceso: La Ingeniería de *Software* tiene un propósito y se relaciona con desarrollar el *software* de manera metodológica y disciplinada. Este aprendizaje se mencionó por parte del 13% de los participantes.
- Proyecto: Se encuentra en este juego, con opinión minoritaria de los participantes, la importancia de la planeación (aunque no fue definitiva en el resultado del juego), con un 5% de los participantes, y el reconocimiento de que la presión es uno de los factores que influyen en el desarrollo de *software* (3%).

Ahora, en lo relativo a los elementos que se requerían para ganar el juego (véase tabla 10), los participantes opinaron mayoritariamente (88%) que si hubieran conocido con antelación el orden en que los diagramas debían adquirirse por parte de los equipos, el resultado hubiera sido diferente. Esta apreciación, sin embargo, es contradictoria con la experiencia vivida por el grupo 4, quienes

llevaban casi un semestre realizando las fases de definición y análisis de diferentes aplicaciones de *software* con UN-Método y aún así no lograron captar desde el principio la esencia del juego.

Los demás factores, con una participación en opinión relativamente pequeña (27%, 21% y 21%, respectivamente), fueron la suerte, el conocimiento de las intenciones de la competencia y la buena toma de decisiones. El solo reconocimiento de estos factores es una justificación importante para el juego, puesto que son estos algunos de los elementos que podrían determinar el éxito de un proyecto en el futuro profesional de cualquiera de los participantes en esta experiencia.

NECESIDADES PARA GANAR EL JUEGO	%	Aspecto
Conocer el orden de los diagramas	88	Proceso
Suerte	27	Proceso
Conocer las intenciones de la competencia	21	Proyecto
Buena toma de decisiones	21	Personal

Tabla 10. Resumen de respuestas a la pregunta "¿Qué cree que se necesitaba para ganar el juego?".

Finalmente, en lo relativo a las modificaciones planteadas para el juego (véase tabla 11), que se respondieron únicamente por parte del grupo 3, sólo cuatro modificaciones alcanzaron a tener más del 30% de las opiniones de los participantes. Sin embargo, en este juego todos los encuestados entregaron sugerencias de posible mejoramiento del juego.

MODIFICACIONES A LOS JUEGOS	%
Breve definición de los artefactos de UN-Método	45
Mejoramiento del tablero	38
Mayor profundidad en el tema	31
Cambios en las reglas de juego	31
Más tiempo de juego	17
Incluir penalizaciones por fallas	14
Grupos más pequeños	7
Charla intermedia para socializar la información que se tenga en relación con el orden de los diagramas	3

Tabla 11. Resumen de respuestas a la pregunta "¿Qué modificaciones le haría al juego?".

La modificación que más sugirieron los participantes fue la necesidad de presentar una charla introductoria sobre los artefactos de UN-Método antes del inicio del juego. Esto es especialmente válido cuando el grupo no conoce de manera detallada el método o cuando el grupo es heterogéneo en cuanto a la formación previa, como ocurrió con el grupo 3, cuya formación en pregrado era bastante disímil y lo único que tenían en común era que estaban cursando la misma maestría. Sin embargo, una sugerencia tal puede incrementar sustancialmente el tiempo de juego. Para futuras partidas, se requeriría encontrar una manera didáctica de impartir el contenido mínimo de los artefactos de UN-Método, pero sin atentar contra el tiempo de juego.

La aplicación que se desarrolló para estos grupos, un archivo en Winword® con una interfaz sencilla de dados en Visual Basic®, fue la fuente del segundo nivel de importancia de las sugerencias de modificación, con el 38% de los participantes. A este respecto, las sugerencias se orientaron a una interfaz que pudiera garantizar de manera más autónoma el reglamento, especialmente en el manejo de la casilla "pierde un turno". Si bien intencionalmente el director del juego procuraba que el manejo de esa casilla fuera grupal, con el fin de que cada uno de los participantes del grupo de forma lo más ética posible fuera quien recordara qué grupo había perdido un turno, en la mayoría de las sesiones se generaron discusiones por el respeto del turno, tergiversando el sentido de lo que se pretendía. También, algunas de las opiniones se centraron en la posibilidad de tener una interfaz que pudiera jugarse simultáneamente en varios países por medio de la internet, pero esto atentaría contra la idea de que la tecnología se involucrara lo menos posible en el desarrollo del juego. A futuro, podría ser importante desarrollar un tablero plástico muy grande, como el que actualmente se utiliza en las versiones del juego de la cerveza (Senge, 1994), para despejar completamente la idea de que es necesaria la tecnología para practicar este juego.

La sugerencia de mayor profundidad en el tema, en la que coincidió el 31% de los participantes, podría atenderse simultáneamente con juegos complementarios, que muestren la forma de articular los diferentes artefactos o que, simplemente, muestren el comportamiento de los diferentes diagramas en un entorno de elicitación de requisitos. Otra sugerencia emitida por el 31% de los participantes fue el cambio en las reglas de juego para hacer más interesante el juego. Algunas de las propuestas se centraron en cambios para agilizar el juego y uno de los principales elementos del mismo que causa los mayores retrasos, como es el de la captura de los diagramas. En este sentido, se propusieron cambios como hacer que la casilla de diagramas propios y ajenos fuera genérica y que fuera el equipo que cae en ella el que decidiera cuál diagrama solicitaría o capturaría; además,

se sugirió que existieran comodines para la liberación de diagramas. Las demás sugerencias no tuvieron un porcentaje de opinión representativo y se pueden consultar directamente en la tabla 11. En ese grupo, se destaca la necesidad de mayor tiempo de juego, una sugerencia que debería estudiarse con cuidado, puesto que incrementar ese tiempo podría significar una disminución en la atención de los participantes.

6.3. El juego de la consistencia

Si bien este juego fue el resultado de un trabajo de grado presentado en el semestre 2006-03, se ha seguido practicando en grupos adicionales a los que ya se analizaron en dicho trabajo. Esos grupos adicionales se listan en la tabla 12. La forma de retroalimentación a partir del juego tuvo dos experiencias diferentes. Por un lado, los grupos 1 y 3 tuvieron dos tipos de encuestas: una antes del juego y otra después del juego. Por otro lado, el grupo 2 continuó con la forma de retroalimentación de los juegos como en las secciones 6.1 y 6.2.

código	Semestre	Asignatura	No. Partici-pantes	Nivel de escolaridad
1	2006-01	Ingeniería de *Software*, Ingeniería Lingüística UNAM	6	Grupo combinado de estudiantes y profesores de Ingeniería de *Software* y Lingüística Computacional
2	2006-01	Tópicos Avanzados Ingeniería de *Software*	29	Estudiantes de maestría en Sistemas de primer semestre.
3	2006-03	Ingeniería de Requisitos	13	Estudiantes de sexto a noveno semestre de Ingeniería de Sistemas e Informática de la UNAL
4	2007-01	Tutorial: Uso de esquemas preconceptuales para la generación automática de diagramas de clases, comunicación y máquina de estados de UML	41	Estudiantes y profesores de distintas áreas asistentes a las VI Jornadas Iberoamericanas de Ingeniería del *Software* e Ingeniería del Conocimiento. Eran en su mayoría peruanos, por ser el país sede.

Tabla 12. Características de los grupos de personas en los cuales se ha aplicado el juego de la consistencia.

La primera forma de retroalimentación y las preguntas básicas de la encuesta se realizaron con base en una adaptación de la propuesta de Klassen y Willougby (2003), tal como se presentó en el trabajo de grado correspondiente a la estudiante Mary Inés Duarte, con quien se hizo el diseño y la aplicación inicial del juego a los primeros grupos. Dichas encuestas son las siguientes:

CUESTIONARIO ANTES DEL JUEGO

Este cuestionario debe resolverse antes de iniciar con el juego. El propósito de este cuestionario es evaluar cuánto aprenden los estudiantes del juego. Este cuestionario no se utilizará para los propósitos de calificar o afectar sus notas para esta clase.

1. Escriba las dos preguntas principales que usted considera se deben formular acerca de consistencia entre diagramas:

a) _____

b) _____

2. ¿Qué entiende por consistencia? _____

3. ¿Qué tan difícil cree usted que es manejar la consistencia entre diferentes diagramas?

Muy fácil Fácil Normal Difícil Muy difícil

La siguiente información no cuenta para ningún propósito de calificación en la asignatura:

Nombre: _____

¿Leyó usted libros sobre consistencia antes del juego? SI_____ NO_____

¿Leyó usted las notas de clase sobre consistencia, diagramas UML, ó grafos conceptuales? SI_____ NO_____

CUESTIONARIO DESPUÉS DEL JUEGO

Este cuestionario debe resolverse después de terminar con el juego. El propósito de este cuestionario es evaluar cuánto aprenden los estudiantes del juego. Este cuestionario no se utilizará para los propósitos de calificar o afectar sus notas para esta clase.

1. Escriba las dos preguntas principales que usted considera se deben formular acerca de consistencia entre diagramas:

a) _____

b) _____

2. ¿Qué entiende por consistencia? _____

3. ¿Qué tan difícil le fue a usted manejar la consistencia entre los diferentes diagramas propuestos?

Muy fácil Fácil Normal Difícil Muy difícil

4. ¿Fue este juego una experiencia de aprendizaje que valió la pena?

_____

_____

5. ¿Qué fue lo mas importante que usted aprendió de este juego?

_____

_____

La siguiente información no cuenta para ningún propósito de calificación en la asignatura:

Nombre: _____

El resumen de opiniones en relación con la primera pregunta se puede visualizar en la tabla 13. Allí, se puede apreciar el cambio que sufren las preguntas fundamentales sobre consistencia antes y después del juego, de lo cual se pueden extraer conclusiones como las siguientes:

- Se puntualizan los elementos relativos a la consistencia después del juego. Al iniciar, el concepto es muy abstracto, pero, después de la sesión de juego, se ejemplifican los conceptos. De preguntas tan generales como "¿Este elemento de este diagrama cómo se representa en este otro diagrama?" se pasa a preguntas tan específicas como "¿Están todas las operaciones que realiza cada actor durante todo el proceso?".
- Después del juego, se establecen relaciones con otros términos íntimamente ligados con la consistencia. Por ejemplo, aparece la consistencia como un proceso de mapeo (transformación) de elementos entre diagramas y surge la inquietud de las reglas que pueden posibilitar esa transformación. Un elemento que se encuentra antes y después del juego es la posibilidad de representar todos los elementos de un dominio específico a lo largo de los diferentes diagramas; esta representación está ligada con el concepto de *completitud*, es decir, la propiedad que tienen los

modelos de incluir la totalidad de los conceptos de un dominio, la cual es de gran importancia en la medición de la calidad de los modelos.

PREGUNTAS ANTES DEL JUEGO	%	PREGUNTAS DESPUÉS DEL JUEGO	%
¿Este elemento de este diagrama cómo se representa en este otro diagrama?	30	¿Cómo debemos mapear este elemento de este diagrama en este otro?	27
¿Se puede tener consistencia entre estos dos diagramas?	22	¿Puede existir consistencia entre este par de diagramas?	25
¿Qué es consistencia?	20	¿Es adecuada la consistencia entre diagramas?	17
¿Para qué se utiliza la consistencia?	18	¿Existen todos los componentes del dominio en cada diagrama?	13
¿Los componentes del dominio se pueden encontrar en los diferentes diagramas?	13	¿Cuál diagrama se elabora primero que los otros?	13
¿Cómo se maneja la consistencia en la metodología elegida?	12	¿Cuáles son las reglas para expresar consistencia?	12
¿Es adecuada la consistencia?	8	¿Cómo se maneja la consistencia en la metodología elegida?	10
¿Cuántos y cuáles elementos gráficos debe tener un diagrama para ser claro?	8	¿Están todas las operaciones que realiza cada actor durante todo el proceso?	5
¿En algunas oraciones puede ser normal la variación de relaciones, su nombre?	5	¿Cómo se maneja la consistencia entre el esquema preconceptual y los casos de uso?	5
¿Cómo se maneja la trazabilidad?	3	¿Existe consistencia entre las relaciones de los objetos?	5
¿Qué nivel de detalle tienen los diferentes diagramas?	3	¿Cómo se demuestra la consistencia?	5
¿Existe ambigüedad en el proceso de desarrollo de *software*?	3	¿No todo se incluye en cada diagrama?	3
¿Las clases en el diagrama de clases de alguna forma están involucradas en el preconceptual?	2	¿Los diferentes diagramas representan el mismo dominio?	3
¿Son iguales los nombres de las acciones y los objetos?	2	¿Existe consistencia entre los nombres?	3
		¿Puede ser tan rígido el modelamiento?	2
		¿Cómo se diferencian la trazabilidad y la consistencia?	2

Tabla 13. Cambios en los porcentajes de opinión acerca de las preguntas fundamentales de la consistencia.

- Igualmente, surgen interrogantes entre términos aparentemente similares, tales como consistencia y la trazabilidad (una propiedad que se asocia con poder seguir el rastro de un elemento a través de todos los diagramas en las diferentes fases del desarrollo). Este tipo de dudas pueden

conducir a los participantes a consultar ya sea con el profesor o en la literatura disponible sobre estos temas.

- La pregunta "¿Qué es consistencia?", que, antes del juego se mencionó como una de las preguntas principales por parte del 20% de los participantes, desaparece después del juego. Se espera que el juego haya contribuido de esta manera a clarificar el concepto.

En cuanto a la definición de consistencia solicitada por la pregunta 2 de la encuesta antes y después del juego, se asignó una calificación de 1 a 5 a cada una de las respuestas suministradas por los participantes. El grupo 2 no presentó esta encuesta y, por lo tanto, no se pudo evaluar el conjunto de sus definiciones de consistencia. El grupo 1 tuvo una calificación de 3.0 antes del juego y de 3.8 después del juego; la calificación inicial era previsible, pues el grupo era bastante heterogéneo, con personas de varias disciplinas, algunas poco ligadas con la Ingeniería de *Software*. El grupo 3 tuvo una calificación de 4.1 antes del juego y de 4.7 después del juego; este grupo era el que tenía más contacto con el concepto de consistencia, si bien lo venían trabajando durante todo el semestre en su curso de Ingeniería de Requisitos. El grupo 4 era el que menos conocía de consistencia, también por las características heterogéneas, pues, se trataba de público de una conferencia iberoamericana que, aunque especializada, reunió personas de diferentes disciplinas; este hecho se vio reflejado en la calificación de 1.5 antes del juego, que se incrementó a 2.7 después del juego. La media ponderada de la calificación antes del juego para los tres grupos fue 2.2 y presentó un incremento del 37% para ubicarse después del juego en 3.1. Si bien no es el ideal de calificación en lo relativo a un concepto como el que se evalúa, las definiciones mostraron una evolución significativa, especialmente tomando en consideración las características heterogéneas de los participantes.

Ahora, en lo que respecta a la dificultad para manejar la consistencia entre diagramas, las respuestas antes y después del juego no sufrieron variaciones notables, salvo un pequeño decremento en la opinión de que es muy difícil de manejar. En general, el promedio se encuentra para ambos casos entre normal y difícil, con mayor tendencia a ser difícil antes del juego y mayor tendencia a ser normal después del juego. Esto se refleja también en el hecho de que las percepciones acerca del grado de dificultad *normal* después del juego incrementaron (pasaron de 43% a 54%) en relación con el dato antes del juego y que las percepciones acerca del grado de dificultad *difícil* rebajaron (pasaron de 51% a 32%). Estos datos se consignan en la tabla 14.

DIFICULTAD ANTES	%	DIFICULTAD DESPUÉS	%

Muy fácil	0	Muy fácil	0
Fácil	0	Fácil	7
Normal	43	Normal	54
Difícil	51	Difícil	32
Muy difícil	6	Muy difícil	7

Tabla 14. Resumen de opiniones de los participantes en relación con la dificultad de manejo de la consistencia entre diagramas antes y después del juego.

Los participantes manifestaron de forma mayoritaria (92% de las opiniones) no haber leído libros sobre consistencia y los que habían leído materiales de clase sobre consistencia, UML o grafos conceptuales fueron una minoría (37%). Al respecto, es importante cuestionarse en relación con la disponibilidad de material bibliográfico en el entorno académico sobre estos temas. Si bien la consistencia representa un tópico supremamente avanzado en la enseñanza de la Ingeniería de *Software*, la carencia de conocimiento de su existencia puede acarrear un sinnúmero de fallas en el ámbito profesional. El esfuerzo de los profesores del área de Ingeniería de *Software* para enseñar estos temas debería complementarse con la adquisición de material bibliográfico que permita a los estudiantes interesados realizar consultas al respecto, lo cual introduce una inquietud para las bibliotecas universitarias, que, tradicionalmente, pocos ejemplares tienen de temas tan específicos.

En la encuesta posterior al juego, los participantes respondieron de forma unánime (100% de las opiniones) que el juego fue una experiencia de aprendizaje que valió la pena. Pocos de ellos justificaron su respuesta (lo cual implica bajos porcentajes de opinión), pero, en la tabla 15, se pueden apreciar las respuestas que se entregaron para esta justificación con sus respectivos porcentajes. Estas respuestas se relacionan con los nuevos conceptos a que se hacía alusión en el análisis de las preguntas fundamentales de la consistencia, tales como la secuencia que debe existir en la realización de los diagramas (que se traduce en la utilización de un método de desarrollo de *software*), las reglas de transformación entre los diferentes diagramas, y la importancia que puede tener la consistencia como concepto en el futuro desarrollo profesional de los participantes. La participación alcanzada por los esquemas preconceptuales en la consistencia (8%) se debió completamente al grupo 4, a cuyos integrantes se les estaba presentando un tutorial específicamente en el tema de creación y utilización de estos esquemas para la obtención automática de diagramas de UML.

RAZONES	%
Secuencia en la realización de diagramas	13
Los esquemas preconceptuales ayudan a manejar la consistencia	8
El juego fue una experiencia de aprendizaje	7
Transformación entre diagramas	5
Parece simplificar la consistencia	5
La consistencia a veces se pasa por alto en la realidad	2
En la realidad existe mucha presión para el modelamiento	2

Tabla 15. Resumen de justificaciones de los participantes a la pregunta "¿Fue este juego una experiencia de aprendizaje que valió la pena?".

Finalmente, la respuesta a la última pregunta de la encuesta posterior al juego está compartida con el análisis del grupo 2, puesto que es común a ambos métodos de retroalimentación.

La segunda forma de retroalimentación, realizada únicamente sobre el grupo 2, consistió en una encuesta posterior al juego donde se debían responder las siguientes preguntas:

(1) ¿Qué aprendió del juego?
(2) ¿Qué modificaciones le haría al juego?

El resumen de las respuestas a la primera pregunta, junto con el aspecto del modelo de las cuatro P's al que pertenece, se puede consultar en la tabla 16. En este grupo, se encuentran también tabuladas las respuestas del grupo 3. Los aprendizajes se pudieron correlacionar únicamente con los aspectos *Proceso* y *Personal*, porque éste es un juego que se desarrolla al interior de un proyecto y es una parte en la cual aún el producto definitivo no existe. Se destaca el aprendizaje del uso de la consistencia (aunque, por las preguntas fundamentales, se alcanza a intuir que no es clara la manera de extraer reglas de consistencia a partir del juego) y la utilidad de los diferentes diagramas. En un segundo renglón de importancia, se encuentran las fases de desarrollo de *software* y la comunicación en el grupo de trabajo. En un tercer renglón, se encuentran el trabajo en equipo y la importancia de la documentación y el modelamiento. Como ya se ha mostrado para los demás juegos, estos elementos hacen parte del saber específico que requiere un Ingeniero de *Software* y es posible su simulación mediante el juego.

APRENDIZAJES	%	Aspecto
Cómo y cuando se valida la consistencia entre diagramas	48	Proceso
Utilidad de los diferentes diagramas	45	Proceso
Fases del desarrollo de *software*	27	Proceso
Comunicación en el grupo de trabajo	13	Personal
Trabajo en equipo	7	Personal
Importancia de la documentación y el modelamiento	6	Proceso

Tabla 16. Resumen de respuestas a la pregunta "¿Qué aprendió del juego?".

Ahora, la segunda pregunta sólo se respondió por parte del grupo 2 y los resultados se pueden consultar en la tabla 17.

MODIFICACIONES AL JUEGO	%
Ampliación previa del tema de la consistencia	43
Permitir refinamiento mostrando reglas de mapeo	32
Debería existir más de una solución, como en la realidad.	18
Efectuar dos etapas en el juego	11
Promover la realización de una aplicación de *software* a partir de los diagramas	11
Más tiempo de juego	11
Ninguna	11
Dar más libertad al esquema preconceptual	7
Incrementar el número de diagramas	7
Agregar "pistas" a los diagramas	7
Cambiar el ejemplo del modelo verbal	4
Hacer una evaluación individual al final del juego	4

Tabla 17. Resumen de respuestas a la pregunta "¿Qué modificaciones le haría al juego?".

Nuevamente, aparece como una modificación ampliamente sugerida (43%) la necesidad de realizar una charla u otro tipo de actividad en relación con el concepto de consistencia antes del juego; esta modificación se complementa con la siguiente, en la cual los participantes opinan que es importante conocer las reglas de mapeo desde el principio (32% de los participantes), como una manera más sencilla de evaluar la consistencia entre los diagramas considerados en el juego. Las demás respuestas no tuvieron porcentajes importantes de opinión, aunque podrían resumirse en una

necesidad de simplificar el juego para poderlo completar a cabalidad y la posibilidad no explorada en el juego de que el modelamiento no sea único.

6.4. El juego de la planeación de proyectos de *software*

Este es uno de los juegos más recientes y, hasta ahora, se ha practicado con cuatro grupos, cuyas características se muestran en la tabla 18.

Código	Semestre	Asignatura	No. Partici-pantes	Nivel de escolaridad	Respuestas
1	2006-01	Ingeniería de *Software*, Ingeniería Lingüística UNAM	13	Grupo combinado de estudiantes y profesores de Ingeniería de *Software* y Lingüística Computacional	1 y 2
2	2006-01	Ingeniería de Producción	9	Estudiantes de sexto a noveno semestre de Ingenería de Producción UNAL	1 y 2
3	2006-01	Tópicos Avanzados Ingeniería de *Software*	26	Estudiantes de maestría en Sistemas de primer semestre.	1 y 3
4	2006-03	Ingeniería de Requisitos	14	Estudiantes de sexto a noveno semestre de Ingeniería de Sistemas e Informática de la UNAL	1 y 2

Tabla 18. Características de los grupos de personas en los cuales se ha aplicado el juego de la planeación de proyectos de *software*.

La retroalimentación del juego se realizó con una encuesta posterior al juego, en la que los participantes debían responder algunas de las siguientes preguntas:

(1) ¿Qué aprendió del juego?
(2) ¿Qué cree que se necesitaba para ganar el juego?
(3) ¿Qué modificaciones le haría al juego?

En la tabla 18, se consigna igualmente cuáles preguntas se respondieron por parte de cada uno de los grupos en los cuales se realizó el juego.

Para la primera pregunta, en la tabla 19, se muestra el resumen de las respuestas entregadas por los participantes. Nuevamente, se recurrió para el análisis de estas respuestas al modelo de las 4 P's y en la tabla se señala el aspecto de ese modelo al que se refiere cada una de las respuestas.

APRENDIZAJES	%	Aspecto
Siempre se debe realizar una planeación del proyecto	89	Proyecto
Existen riesgos en el desarrollo de *software*	60	Proceso
Toma de decisiones y estrategia	24	Personal
Trabajo en equipo	15	Personal
Las planeaciones no se pueden seguir al pie de la letra	8	Proyecto
Utilidad de los diagramas de Gantt	8	Proyecto
Existe la competencia	5	Proyecto

Tabla 19. Resumen de respuestas a la pregunta "¿Qué aprendió del juego?".

En este caso, el juego puede considerarse una mezcla de los aspectos *Proyecto* y *Proceso*, puesto que las dos primeras respuestas se contestaron, mayoritariamente, en estos aspectos. Se pone de manifiesto que el mayor aprendizaje del juego es la necesidad de realizar una buena planeación del proyecto (con el 89% de los participantes). Fue claro en el juego que quienes peor planearon sufrieron consecuencias como la pérdida de horas-hombre que les salían en los dados por no haber completado una de las fases previas, o las dificultades para repartir los beneficios generados por las horas-hombre de contratación adicional en dos procesos. Ahora, en cuanto a los riesgos que están presentes en el desarrollo de *software* (resaltados por el 60% de los participantes como aprendizaje del juego), fue mucho más definitiva en este caso la incidencia de la suerte en el resultado. En el juego de UN-Método, a pesar de tener características similares a éste, los participantes no señalaron la suerte con un papel tan preponderante en ese caso. En el juego de la planeación, se pudo apreciar, e incluso deducir en las sesiones, los efectos devastadores que pueden tener los riesgos del desarrollo de *software*, como, por ejemplo, los retrasos que pueden surgir por la mala captura de los requisitos del interesado o la mala elaboración de los diagramas correspondientes. Otros riesgos identificados fueron la pérdida de personal clave cuando el proyecto está en marcha y los problemas que pueden surgir por efecto de un cambio en las regulaciones de un país. Estas reflexiones se realizaron durante las sesiones por los mismos participantes, pensando en el significado de la devolución de horas-hombre que podía salir en los dados.

Las siguientes respuestas en orden de importancia correspondieron al aspecto *Personal* de las 4 P's y se relacionan con la buena toma de decisiones y la elección de una estrategia adecuada (24% de las opiniones) y el trabajo en equipo (15%). En relación con la primera respuesta, la simulación permitió que los participantes experimentaran los efectos de las decisiones que tomaban especialmente en el tema de contratación adicional, donde podían existir varias opciones de repartición de las horas-hombre, que generaban distintas posibilidades de avance. Además, se realizó un ejercicio en los grupos 3 y 4 para que la devolución de horas señalada por los dados estuviera determinada por un equipo competidor, lo que le añadió un ingrediente estratégico especial al juego. En relación con el trabajo en equipo, se notaba en los diferentes equipos la discusión profunda que requerían los ejercicios de toma de decisiones marcados por los dados; este fue uno de los juegos donde la estrategia se discutió con mayor intensidad al interior de los grupos.

Las últimas respuestas, con un bajo porcentaje de opiniones de los participantes, se relacionan con el aspecto *Proyecto*, más específicamente con la imposibilidad de seguir una planeación al pie de la letra, la utilidad de los diagramas de Gantt y la existencia de la competencia. A este respecto, en el juego de UN-Método, se notó un porcentaje de respuestas un poco mayor en relación con la competencia, debido a que en ese juego era mucho más claro que existían estrategias que procuraban desestabilizar a los demás equipos en contienda.

Ahora, en lo que respecta a los elementos necesarios para ganar el juego (véase tabla 20) los participantes tuvieron mayor consenso, pues, sólo entregaron cuatro tipos de opiniones, de las cuales las tres primeras obtuvieron el 50% o más de las opiniones de los participantes. Se resalta la importancia de repartir adecuadamente las horas adicionales por parte del equipo (56%) como el factor más importante para ganar el juego. Cuando las horas de contratación adicional se repartían de manera inadecuada, las consecuencias en el desarrollo del juego eran bastante notorias. Los siguientes aspectos, con el 50% de las opiniones, fueron la generación de una buena planeación adicional y el factor suerte. El único factor exógeno (con únicamente el 6% de las opiniones) se relaciona con la estrategia para perjudicar a la competencia.

Finalmente, en relación con las modificaciones planteadas al juego (véase tabla 21), las cuales se propusieron únicamente por parte del grupo 3, se destaca con el mayor porcentaje de opinión (46%) el cambio de la interfaz de juego. A este respecto, la interfaz en Excel® fue una de las principales causas de inconformidad de parte de los participantes, puesto que retrasaba considerablemente el juego. A este respecto, cabe el mismo comentario expresado para el juego de UN-Método:

desarrollar un tablero de juego en el que la interfaz no sea por computador y que se pueda jugar en cualquier tipo de aula, así no disponga de computador y equipos de proyección.

NECESIDADES PARA GANAR EL JUEGO	%	Aspecto
Buena toma de decisiones para repartir las horas adicionales	56	Personal
Tener una buena planeación inicial	50	Proyecto
Suerte	50	Proyecto
Buena estrategia para afectar a la competencia	6	Personal

Tabla 20. Resumen de respuestas a la pregunta "¿Qué cree que se necesitaba para ganar el juego?".

MODIFICACIONES AL JUEGO	%
Modificaciones a la interfaz y los materiales de juego	46
Mayor conocimiento de diagramas de Gantt y otras técnicas de planeación	19
Más tiempo de juego	19
Ninguna	15
Retroalimentación después de terminar cada fase	12
No depender completamente de los dados	8
Incluir costos y otros elementos en la planeación	8
Cambio en las reglas: poder utilizar horas-hombre sobrantes de un proceso en otro	8
Dar "comodines" para gestionar riesgos	4
Menos personas por grupo	4

Tabla 21. Resumen de respuestas a la pregunta "¿Qué modificaciones le haría al juego?".

Se reiteran modificaciones ya planteadas en los otros juegos, como el mayor conocimiento previo en el objeto de estudio del juego (19% de las opiniones), en este caso sobre el diagrama de Gantt, y la necesidad de más tiempo de juego (19% de las opiniones). A este respecto, cabe realizar el mismo comentario de juegos anteriores en el sentido que un mayor tiempo de juego podría fatigar a los participantes y restarles su disposición y motivación para el aprendizaje, por lo cual deberá analizarse con mucho cuidado. Tan sólo un 15% de los participantes expresó que el juego no necesitaba ninguna modificación. Las demás modificaciones planteadas tuvieron un bajo porcentaje de opinión y pueden consultarse en la tabla 21.

6.5. El juego de la trazabilidad y la comunicación.

Este es el juego de más reciente creación y por ello sólo se ha experimentado en un grupo de participantes, cuyas características se pueden consultar en la tabla 22. Es de notar que este juego se diseñó para reemplazar el juego del desarrollo de *software* al interior del proyecto *El Jardín de la Ciencia*, impulsado por la Facultad de Minas, y por ello el público objetivo que se espera para la práctica del juego en futuras partidas estará conformado por estudiantes de primaria y bachillerato.

Semestre	Asignatura	No. Estudiantes	Nivel de escolaridad
2006-03	Ingeniería de Requisitos	18	Estudiantes de sexto a noveno semestre de Ingeniería de Sistemas e Informática de la UNAL

Tabla 22. Características de los grupos de personas en los cuales se ha aplicado el juego de la trazabilidad y la comunicación.

La retroalimentación del juego se realizó con una encuesta que se debía responder al principio y al final del juego, con las siguientes solicitudes:

(1) Mencione los tres aspectos que considere fundamentales en la elicitación de requisitos.
(2) ¿Qué entiende por trazabilidad?

Las respuestas a la primera solicitud se resumen en la tabla 23. De esta tabla, se pueden apreciar los cambios en las opiniones de los participantes producto de la experiencia del juego. Por ejemplo, el aspecto *consistencia*, que, antes del juego, contaba con un 67% de las opiniones de los participantes, consiguió un 11% de las opiniones después del juego. Con esto no se puede concluir que la consistencia no sea importante en la elicitación de requisitos, sino que después del juego las prioridades de los participantes cambiaron, y que el aspecto consistencia no es tan fundamental si no se dan algunas cosas primero, tales como la comunicación, la claridad en la información y la comprensión del problema. El aspecto *comunicación*, que antes del juego contaba con un 8% de las opiniones de los participantes, pasó a un 56% después del juego, lo que muestra lo efectiva que fue la simulación realizada mediante el juego. Un factor decisivo en esa simulación fue la imposibilidad de comunicación “cara a cara”, que dificultó notoriamente la elaboración correcta de los diferentes diagramas. Otros de los aspectos que perdieron protagonismo, en opinión de los participantes, fueron los artefactos de representación, que pasaron del 50% al 17% de las opiniones, y la

trazabilidad, que pasó del 25% al 6% de las opiniones. Ello se debe a que estos aspectos se pueden garantizar después de que los diferentes elementos del dominio del problema estén lo suficientemente claros, que, nuevamente, es un asunto de comunicación.

ASPECTOS FUNDAMENTALES DE LA ELICITACIÓN DE REQUISITOS ANTES DEL JUEGO	%	ASPECTOS FUNDAMENTALES DE LA ELICITACIÓN DE REQUISITOS DESPUÉS DEL JUEGO	%
Consistencia	67	Comunicación	56
Artefactos de representación	50	Claridad en la información	39
Claridad en la información	50	Comprensión del problema	39
Comprensión del problema	42	Completitud en la información	33
Trazabilidad	25	Artefactos de representación	17
Completitud de la información	8	Consistencia	11
Comunicación	8	Método de desarrollo	11
Liderazgo	8	Observación	11
Responsabilidad	8	Liderazgo	11
		Organización	11
		Coordinación	11
		Trazabilidad	6
		Interpretación	6

Tabla 23. Resumen de respuestas a la solicitud “Mencione los tres aspectos que considere fundamentales en la elicitación de requisitos”.

De la tabla 23, también, se deduce que aparecen nuevos elementos que antes del juego no se consideraban fundamentales y, después del juego, ya sí lo son. Aspectos como los métodos de desarrollo, la observación, la organización, la coordinación y la interpretación aparecen con una participación minoritaria en opinión de los participantes.

Ahora, en relación con la segunda pregunta, se le asignó un puntaje de 1 a 5 a cada una de las definiciones de trazabilidad que entregaron los participantes tanto antes como después del juego. Se nota por las respuestas que los participantes conocían el término con anterioridad, pues, el puntaje promedio fue 3.8. Se nota igualmente que al final del juego el término se logró clarificar un poco, pues el puntaje promedio se incrementó a 4.4. Esto representa un incremento del 16% en la apreciación del profesor para juzgar las definiciones de trazabilidad suministradas por los participantes.

CAPÍTULO 7: CONCLUSIONES Y TRABAJO FUTURO

Los estudiantes de Ingeniería de *Software* requieren formación en algunas competencias que son difíciles de trasmitir con los métodos tradicionales de enseñanza de esa disciplina. Las típicas clases expositivas y los proyectos de "juguete" que se suelen realizar en las clases de Ingeniería de *Software* contribuyen sólo parcialmente a delimitar los conceptos y habilidades de modelamiento que requieren estos estudiantes para su futuro profesional. Sin embargo, existen otras habilidades que sólo se forman en el ejercicio profesional y que poco o nada se simulan en el aula de clase. Entre estas habilidades se cuentan: los riesgos a que se ven sometidos los proyectos de *software*, la importancia de la planeación cuidadosa de las actividades, la organización de las tareas al interior de un grupo de trabajo, o incluso el abordaje en mayor profundidad de temas propios del ejercicio profesional como la trazabilidad y la consistencia. Estos temas requieren un complemento a las técnicas tradicionales de enseñanza de la Ingeniería de *Software*, que, en el contexto de este trabajo de promoción, se han procurado solucionar empleando juegos no tecnológicos para realizar en clase.

Para llegar a esta solución, se plantearon las alternativas que se han empleado como complemento a la enseñanza tradicional y se identificaron los principales problemas que aún subsisten en la utilización de estas alternativas particularmente en la Universidad Nacional de Colombia: las alternativas tecnológicas (como los juegos de video o computador) son costosas y requieren una gran cantidad de recursos técnicos y humanos, que esta Universidad no está en capacidad de suministrar. Igual ocurre con otras alternativas como los laboratorios o la enseñanza personalizada. Estos problemas se agravan cuando se toma en consideración la alta demanda formación que tiene este tipo de disciplinas en la actualidad y la baja tasa de contratación de nuevos profesores en estas áreas.

En este trabajo de promoción, se propusieron cinco juegos no tecnológicos para aplicar en las clases de Ingeniería de *Software* para promover el aprendizaje de un conjunto de temas no cubiertos por las diferentes experiencias encontradas en la literatura. Estos cinco juegos se diseñaron y aplicaron a diferentes grupos de participantes durante cursos relacionados con la Ingeniería de *Software* para estudiantes de pregrado y postgrado, y para algunos grupos especiales donde se mezclaron estudiantes y profesores del área y de otras áreas. Igualmente, se presentó un análisis de la retroalimentación brindada por los practicantes de los juegos, donde se pudo compendiar lo que habían aprendido del juego y las oportunidades de mejoramiento para la mayoría de esos juegos.

Como resultado de la experiencia de aplicación de estos juegos durante los últimos dos años, algunas conclusiones que se pueden extraer son las siguientes:

- Los juegos realizados han trascendido las barreras culturales y de formación de sus participantes. Estos juegos se han practicado con grupos disímiles de participantes, con resultados muy similares en todos los casos. Por ejemplo, en la Universidad Nacional Autónoma de México, se dictó un curso de veinte (20) horas sobre Ingeniería de *Software*, únicamente con juegos y estudio de casos. El público de ese curso fue bastante heterogéneo, puesto que estaba conformado por estudiantes de diferentes niveles y de diferentes carreras, además de profesores de distintas áreas. En este caso, el grupo concluyó que, para aprender los conceptos fundamentales de la Ingeniería de *Software* utilizando este tipo de métodos alternativos, sólo se requería un mínimo de conceptos previos que se podrían adquirir con algunas clases expositivas. A una conclusión similar llegó el grupo de Tópicos Avanzados en Ingeniería de *Software*, conformado por un conjunto de ingenieros de muy diversas áreas y con los cuales se realizó un trabajo especial con los juegos, pues, durante veinte (20) horas de clase se realizó una competencia con ellos simulando empresas de *software* que competían en los diferentes juegos planteados. Una experiencia, también con un grupo heterogéneo, se realizó en las VI Jornadas Iberoamericanas de Ingeniería del *Software* e Ingeniería del Conocimiento, en Lima (Perú), con los cuales se realizó el juego de la consistencia durante un tutorial complementario a las ponencias realizadas en el congreso. Así pues, desde niños de primaria y bachillerato hasta profesionales con doctorado, pasando por estudiantes universitarios de varios niveles y carreras, han experimentado el trabajo con los juegos y han podido expresar sus aprendizajes a raíz de su participación en los mismos.
- Una conclusión se desprende de la anterior: los juegos de clase no reemplazan los métodos tradicionales de enseñanza, sino que los complementan. En las respuestas de retroalimentación de los diferentes juegos, una de las solicitudes constantes por parte de los participantes fue la profundización en los temas relativos a los diferentes juegos. Una de las sugerencias para lograr esa profundización, fue el inicio de cada uno de los juegos con una charla en relación con el tema que se cubriría con los juegos, y con una mejor explicación de lo que se pretende en cada juego como parte de esa charla inicial. Aunque las experiencias de aprendizaje son muy vívidas con los juegos, y la posibilidad de recordar ciertos conceptos es mucho mayor, con las clases expositivas se cubre con más profundidad el tema.
- Quizá uno de los elementos que ha facilitado la práctica de estos cinco juegos se relaciona con el hecho que siempre se han realizado con personas que tienen algún nivel—así sea mínimo—de conocimientos en el área o el conjunto de conceptos de que trata el juego. En estos casos, los juegos

han servido para afianzar los conocimientos que los participantes ya tienen y que, probablemente, tenían dificultades en recordar. Además de esos conocimientos, la participación en los juegos ha incentivado el descubrimiento de otros conocimientos, que en ocasiones ni el mismo director del juego hubiera esperado lograr. Desde ese punto de vista, la experiencia es siempre diferente, puesto que en cada grupo que juega aparecen nuevos elementos de discusión que contribuyen a modificar o perfeccionar las reglas de juego, y reelaborar la experiencia para los próximos grupos en los que se aplique.

- Algunos conceptos como los riesgos en el proceso de desarrollo y la suerte, sólo se comprenden cabalmente cuando se experimentan, cuando se vivencian. Es difícil preparar una clase expositiva en la que el tema sean las prácticas desleales de la competencia o la manera de "robar" al mejor programador de otra empresa de desarrollo. Desde el punto de vista ético, no se deberían enseñar estas cosas. Sin embargo, es necesario que los estudiantes se informen sobre lo que puede ocurrir en un ambiente de competencia desleal, y lo que sucedió particularmente en el juego de UN-Método y el juego de la planeación de proyectos de *software* fue especialmente importante como simulación de lo que potencialmente puede ocurrir en el desempeño profesional de las personas de esta área. Es mejor captar esta experiencia a partir de una simulación, en la que no ocurren daños potenciales en las personas, y no esperar a que el ambiente laboral real enseñe de manera más dolorosa (con el desbordamiento de los costos de un proyecto real, por ejemplo), que este tipo de cosas puede ocurrir.
- El conjunto de aprendizajes de los cinco juegos permitió un vistazo general a la Ingeniería de *Software*, tomando en consideración la visión de gestión de proyectos reclamada por Fairley (1978) para este tipo de profesionales. Si se resumen los aprendizajes captados en los juegos, se encuentra una gran mayoría de las lecciones que debería tener un curso de Ingeniería de *Software*. Con excepción de los aprendizajes específicos (cómo programar en un determinado lenguaje, cómo utilizar una determinada herramienta CASE, cómo definir una especificación en un lenguaje formal, etc.), para los cuales aún no se dispone de juegos, los conocimientos generales que se imparten mediante este conjunto de juegos son los deseables para un curso introductorio de Ingeniería de *Software*. Los aprendizajes específicos mencionados aún se pueden lograr con la práctica, pero sería interesante tratar de definir un juego con el que se pudieran transmitir dichos conocimientos.
- Una de las lecciones aprendidas a través de este proceso de diseño y aplicación de juegos para la enseñanza de la Ingeniería de *Software* se relaciona con el hecho de que, sin importar el tema de que se trate, siempre es posible diseñar algún tipo de juego que soporte su enseñanza. Lo importante es conocer de antemano cuál es el cuerpo de conocimiento que se desea impartir, delimitar claramente los conceptos que se piensa transmitir y definir si existe un juego que se pueda adaptar a la forma

requerida para la transmisión de esos conocimientos. La mayoría de los juegos que se presentan en este trabajo de promoción se basan en juegos ya conocidos: *Monopoly*®, rompecabezas, *origami*, batalla naval, etc. Otros, como el juego de la planeación de proyectos de *software*, no tienen equivalencias en juegos conocidos, pero se convierten en alternativas divertidas que se pueden obtener desde la práctica en el diseño de tales juegos.

La retroalimentación obtenida a partir de la aplicación de los juegos a diferentes grupos ha sido muy interesante y ha generado un conjunto de cuestionamientos que posibilitan la continuación de los trabajos en esta área. Como trabajo futuro de la generación de juegos para la enseñanza de la Ingeniería de *Software*, se pueden mencionar los siguientes temas:

- La necesidad de desarrollar nuevos juegos en otros temas que aún no están cubiertos. Por ejemplo, se está iniciando el desarrollo de un juego que ayude a reforzar el conocimiento de las primitivas conceptuales de UML y que, probablemente, se estará aplicando para el segundo semestre de 2007. Otros juegos similares podrían ser de utilidad en la enseñanza de lenguajes de programación o lenguajes formales. Ahora, en el proyecto especial de trabajo académico que dio origen al juego de la trazabilidad y la comunicación, se discutió ampliamente el caso de juegos como Yu-Gi-Oh!® y la forma como un niño puede memorizar un conjunto de reglas tan complejas como las de este juego de cartas. Una de las ideas que surgió al respecto es la generación de un juego de cartas que se distribuyan entre los jóvenes durante el primer semestre de la carrera y en cuyo reglamento se incluyan reglas complicadas, por ejemplo, de modelamiento o generación de código. Si bien se reconoce que los juegos del estilo de Yu-Gi-Oh!® van acompañados de una estrategia comercial bastante agresiva, este tipo de juegos de cartas podrían potencialmente ser asistentes de la labor docente mientras entretienen a los jóvenes que los practican. Habría que hacer un estudio de la apariencia que debería tener el juego, puesto que existen otras experiencias como la de Baker *et al.* (2005), en la cual las cartas no son lo suficientemente llamativas y podrían no generar la motivación adecuada en los estudiantes.
- Se debe trabajar en las modificaciones propuestas a los diferentes juegos, con el fin de que las reglas se vayan perfeccionando y los conocimientos que se espera de los juegos se afiancen.
- Es importante aplicar los juegos a empleados de una empresa real de desarrollo de *software* para establecer si el cuerpo de conocimientos que se está impartiendo con los juegos coincide con las prácticas profesionales que se encuentran comúnmente en el medio. Si se realiza la validación desde el punto de vista de la industria y se encuentra un resultado similar al que se ha conseguido en la

academia, se tendría potencialmente una herramienta de entrenamiento muy versátil, que podría servir para fines académicos e industriales.

- En el mundo, los juegos tecnológicos (de video o computador) han presentado muy buenos resultados desde el punto de vista pedagógico. Una opción que se ha explorado en este sentido es la elaboración de un juego de estrategia, del estilo de *Age of Empires*® por ejemplo, en el que los participantes interactúen con una simulación del entorno de las empresas de *software*. Se ha encontrado que un gran porcentaje de los estudiantes de las áreas tecnológicas, como Ingeniería de Sistemas e Informática de la Universidad Nacional de Colombia, participan habitualmente en juegos de simulación del tipo *Counter Strike*® y que existen torneos a nivel local y nacional de este tipo de juegos. El desafío en este aspecto es lograr un juego que simule adecuadamente los conocimientos que se piensa impartir y que, a la vez, tenga los elementos llamativos que poseen los juegos de simulación que emplean los jóvenes en la actualidad.
- Un trabajo adicional que se debe realizar es la divulgación de experiencias de este tipo para comenzar a generar credibilidad en sus posibilidades para mejorar la formación de los futuros ingenieros de *software* y de otras áreas tecnológicas. Uno de los obstáculos que se puede encontrar a este tipo de experiencias es la reticencia por parte de los profesores a aplicar nuevas formas de enseñanza como la que se plantea en este trabajo de promoción.
- Un asunto final por tratar es la exploración que debería hacerse de las diversas implicaciones éticas de los juegos en la enseñanza de la Ingeniería de *Software*, puesto que, por el hecho de generar la participación de los estudiantes bajo diferentes roles, se están induciendo algunos tipos de prácticas que pueden tener implicaciones positivas y negativas en la educación de los futuros ingenieros de *software*. Aspectos como la competencia desleal y el espionaje industrial, que se vislumbraron en algunos de los juegos (como en el del desarrollo de *software*, el de UN-Método y el de la planificación de proyectos de *software*), podrían tener implicaciones negativas desde el punto de vista ético, al demostrar en los participantes que esas prácticas existen y pueden parecer comunes en las industrias de *software*. A este respecto, la intención de esta estrategia pedagógica aplicada a la Ingeniería de *Software*, ha sido completamente basada en la dimensión antropológica de la educación, pensando mucho más en mostrar riesgos que se presentan en la realidad con el fin de no replicarlos. No sobra, de todos modos, hacer una exploración respecto de la dimensión ética de los juegos y su incidencia en la educación integral de los futuros ingenieros de *software.*

El desarrollo de juegos tanto tecnológicos como no tecnológicos es una herramienta potencialmente útil para la complementación de la enseñanza tradicional de la Ingeniería de *Software*. Desde el punto de vista pedagógico, existen aún muchos desafíos por enfrentar, que justifican el surgimiento

de grupos de investigación y desarrollo en estos temas. Se han iniciado algunas propuestas tomando como base trabajos dirigidos de grado y proyectos especiales de trabajo académico, y se han comenzado a enviar artículos a revistas y congresos nacionales e internacionales, buscando la revisión de pares académicos en el área para retroalimentar estas experiencias. *El juego de los requisitos* (Zapata y Awad, 2005), la primera experiencia del grupo de Ingeniería de *Software* en este tema, ya se presentó en el XII Congreso Iberoamericano de Educación Superior en Computación CIESC y *El juego de la consistencia* se encuentra en evaluación por pares académicos en la revista de Ingenierías de la Universidad de Zulia en Venezuela, donde pasó ya la segunda revisión. Otros artículos como *El juego del desarrollo de software* y *El juego de la planeación* se encuentran actualmente en preparación. Se está explorando también la posibilidad de publicación de estos temas en medios de carácter más amplio, como revistas dedicadas exclusivamente a divulgación.

REFERENCIAS

ACM y IEEE. (2001). Computing Curricula 2001 Computer Science. En: *ACM Journal of Educational Resources in Computing*, Vol. 1, No. 3, 1–240.

Arango, F. y Zapata, C. M. (2006). *UN-Método para la Elicitación de Requisitos de Software*. Carlos Mario Zapata (Ed.), Medellín.

Arcila, G. y Gaona, E. (2006). *Crear competencias para pensar las ciencias: Hacia una enseñanza universitaria sin aprendizaje*. Ediciones desde abajo, Bogotá.

Arocena, R. (2004). Las reformas de la educación superior y los problemas del desarrollo en América Latina. *Educação & Sociedade*, Vol. 25, No. 88, 915–936.

Ausubel, D., Novak, J. y Hanesian, H. (1978). *Educational Psychology: A Cognitive View (2nd Ed.)*. Holt, Rinehart & Winston, New York.

Bagert, D.Hilburn, Th., Hislop, G., Lutz, M., McCracken, M. y Mengel, S. (1999). Guidelines for Software Engineering Education Version 1.0. *Technical Report CMU/SEI-99-TR-032 ESC-TR-99-002*, Software Engineering Institute, Carnegie Mellon University.

Baker, A., Navarro, E. y van der Hoek, A. (2005). An experimental card game for teaching software engineering processes. En: *The Journal of Systems and Software*, Vol. 75, 3–16.

Barr, P., Marsen, S. y Noble, J. (2005). Oppositional play: gathering negative evidence for computer game values. En: *Proceedings of the second Australasian conference on Interactive entertainment*, Sydney, 3–10.

Beck, K. (2000). *Extreme Programming Explained: Embrace Change*. Addison-Wesley, Reading.

Becker, K. (2001). Teaching with games: the Minesweeper and Asteroids experience. En: *Journal of Computing Sciences in Colleges*, Vol. 17, No. 2, 23–33.

Bierre, K. y Phelps, A. (2004). The use of MUPPETS in an introductory java programming course. En: *Proceedings of the 5th conference on Information technology education*, Salt Lake City, 122–127.

Boehm, B. (2006). A View of 20th and 21st Century Software Engineering. En: *Proceeding of the 28th international conference on Software engineering*, Shanghai, 12–29.

Budgen, D. y Tomayko, J. (2005). The SEI Curriculum Modules and their influence: Norm Gibbs' legacy to Software Engineering Education. En: *Journal of Systems and Software*, Vol. 75, 55–62.

Busenberg, S. y Tam, W. (1979). An Academic Program Providing Realistic Training in Software Engineering. En: *Communications of the ACM*, Volumen 22, No. 6, 341–345.

Carrington, D., Strooper, P., Newby, Sh. y Stevenson, T. (2005). An industry/university collaboration to upgrade software engineering knowledge and skills in industry. En: *The Journal of Systems and Software*, Vol. 75, 29–39.

Carroll, J. y Rosson, M. B. (2005). A Case Library for Teaching Usability Engineering: Design Rationale, Development, and Classroom Experience. En: *ACM Journal on Educational Resources in Computing*, Vol. 5, No. 1, 1–22.

Chen, P. (1976). The Entity-Relationship Model-Toward a Unified View of Data. En: *ACM Transactions on Database Systems*, Vol. 1, No. 1, Pages 9–36.

Christopher, E. y Smith, L. (1991). *Negotiation Training through Gaming: strategies, tactics and manoeuvres*. Kogan, London.

Claypool, K. y Claypool, M. (2005). Teaching software engineering through game design. En: *Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education*, Caparica, 123–127.

Clark, N. (2004). Peer testing in Software Engineering Projects. En: *Proceedings of the sixth conference on Australasian computing education*—Volume 30, Dunedin, 41–48.

Coad, P., LeFebvre, E. y DeLuca, J. (1999). *Java Modeling in Color with UML*. Prentice Hall, Upper Saddle River.

Coleman, R., Roebke, S. y Grayson, L. (2005). Gedi: a game engine for teaching videogame design and programming. En: *Journal of Computing Sciences in Colleges*, Vol. 21, No. 2, 72–82.

Coppit, D. y Haddox, J. (2005). Large team projects in software engineering courses. En: *ACM SIGCSE Bulletin*, Vol. 37, No. 1, 137–141.

Cowling, A. J. (2005). The role of modeling in the software engineering curriculum. En: *The Journal of Systems and Software*, No. 75, 41–53.

Dardenne, A., Van Lamsweerde, A. y Fickas, S. (1993). Goal-Directed Requirements Acquisition. En: *Science of Computer Programming*, Vol. 20, 3–50.

Daré, W. y Barreteau, O. (2003). A role-playing game in irrigated system negotiation: between play and reality. En: *Journal of Artificial Societies and Social Simulation*, Vol. 6, No. 3.

De Aguilera, M. y Méndiz, A. (2003). Video Games and Education (Education in the Face of a "Parallel School"). En: *ACM Computers in Entertainment*, Vol. 1, No. 1, 1–14.

Delamare, F. y Winterton, J. (2005). What is competence? En: *Human Resource development international*, Vol. 8, No. 1, 27–46.

Dibona, Ch. (2004). A conversation with Will Harvey. En: *ACM Queue*, Febrero de 2004, 21–27.

Egloff, T. (2004). Edutainment: A Case Study of Interactive CD-ROM Playsets. En: *ACM Computers in Entertainment*, Volume 2, Number 1, 1–22.

Fairley, R. (1978). Educational Issues in Software Engineering. En: *Proceedings of the 1978 annual conference ACM/CSC-ER*, Washington, D.C., 58–62.

Fairley, R. (1986). The role of Academe in Software Engineering Education. En: *Proceedings of the 1986 ACM fourteenth annual conference on Computer science*, Cincinnati, 39–52.

Faust, M. (2003). Mixed reality gaming environment QuakeRunner. En: *Proceedings of the second international conference on Entertainment computing*, Pittsburgh, 1–4.

Felder, R., Woods, D., Stice, J. y Rugarcia, A. (2000). The future of Engineering Education II: Teaching methods that work. En: Chemical Engineering Education, Vol. 34, No. 1, 26–39.

Gardner, H. (1983). *Frames of mind: The theory of multiple intelligences*. Basic Books, New York.

Gee, J. (2003). What Video Games Have to Teach Us About Learning and Literacy. En: *ACM Computers in Entertainment*, Vol. 1, No. 1, 1–4.

Gevaert, Joseph. (1976). *El problema del hombre: Introducción a la Antropología Filosófica*. Sígueme, Salamanca.

Gibbs, W. (1994). Software's Chronic Crisis. En: *Scientific American*, Septiembre 1994, 72–81.

Glynn, R. (1999). *Origami Gift box*. Disponible en: http://dev.origami.com/images_pdf/giftbox.pdf

Goschnick, S. y Balbo, S. (2005). Game-first programming for information systems students. En: *Proceedings of the second Australasian conference on Interactive entertainment*, Sydney, 71–74.

Hailey, D. (2004). A next generation of digital genres: expanding eocumentation into animation and virtual reality. En: *Proceedings of the 22nd annual international conference on Design of communication: The engineering of quality documentation*, 19–26.

Hansen, S. (2004). The Game of Set®–An Ideal Example for Introducing Polymorphism and Design Patterns. En: *Proceedings of the 35th SIGCSE technical symposium on Computer science education*, Norfolk, 110–114.

Hartness, K. (2004). Robocode: using games to teach artificial intelligence. En: *Journal of Computing Sciences in Colleges*, Vol. 19, No. 4, 287–291.

Hedin, G., Bendix, L. y Magnusson, B. (2005). Teaching extreme programming to large groups of students. En: *The Journal of Systems and Software*, Vol. 74, 133–146.

Hill, J., Ray, C. Blair, J. y Carver, C. (2003). Puzzles and games: addressing different learning styles in teaching operating systems concepts. En: *Proceedings of the 34th SIGCSE technical symposium on Computer science education*, Reno, 182–186.

Hollyday, M. (1995). Incremental game development in an introductory programming course. En: *Proceedings of the 33rd annual on Southeast regional conference*, Clemson, 170–175.

Höysniemi, J. (2006). International Survey on the Dance Dance Revolution Game. En: *ACM Computers in Entertainment*, Vol. 4, No. 2, 1–31.

Huizinga, J. (1972, e.o. 1938). *Homo ludens.* Alianza Editorial, Madrid.

Ishikawa, K. (1986). *Guide to Quality Control*. Asian Productivity Organization, Tokio.

Jaccheri, L. y Morasca, S. (2006). On the Importance of Dialogue with Industry about Software Engineering Education. En: *Proceedings of the 2006 international workshop on Summit on software engineering education*, Shanghai, 5–8.

Jenkins, H., Klopfer, E., Squire, K. y Tan, Ph. (2003). Entering the Education Arcade. En: *ACM Computers in Entertainment*, Vol. 1, No. 1, 1–11.

Jensen, B. (2006). Responding to the enrollment crisis—Alternative strategies to increasing students interest in Computer Science. En: Journal of Computing Sciences in Colleges, Vol. 21, No. 4, 8–8.

Jiménez, G., Gómez, M., Gómez, M. A. y González, P. (2005). Software behaviour understanding Supported by dynamic visualization and role-play. En: *Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education*, Caparica, 54–58.

Johnson, D. W., Johnson, R. y Holubec, E. (1993). *Circles of learning (4th ed.)*. Interaction Book Company, Edina.

Juul, J. (2003). The game, the player, the world: Looking for a heart of gameness. En: Marinka Copier and Joost Raessens (eds.) *Level Up: Digital Games Research Conference Proceedings* (Utrecht University), 30-45.

Kafai, Y. (1996). Software by Kids for Kids. En: *Communications of the ACM*, Vol. 39, No. 4, 38–39.

Kasvi, J. (2000). Not Just Fun and Games—Internet Games as a Training Medium. In Kymäläinen, P. & Seppänen, L.C. —Learning With Computerised Simulation Games, 23–34.

Kelleher, C. y Pausch, R. (2005). Lowering the Barriers to Programming: A Taxonomy of Programming Environments and Languages for Novice Programmers. En: *ACM Computing Surveys*, Vol. 37, No. 2, 83–137.

Kitchenham, B., Budgen, D., Brereton, P. y Woodall, Ph. (2003). An investigation of software engineering curricula. En: *The Journal of Systems and Software*, Vol. 74, 325–335.

Klassen, K. y Willoughby, K. (2003). In-Class Simulation Games: Assessing Student Learning. En: *Journal of Information Technology Education*, Vol. 2, 1–13.

Külling, M. y Henriksen, P. (2005). Game programming in introductory courses with direct state manipulation. En: *Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education*, Caparica, 59–63.

Kruchten, Ph. (1999). *Rational Unified Process—An Introduction*. Addison-Wesley, Reading.

Ladd, C. (2006). The curse of Monkey Island: holding the attention of students weaned on computer games. En: *Journal of Computing Sciences in Colleges*, Vol. 21, No. 6, 162–174.

Lee, J., Luchini, K., Michael, B., Norris, C. y Soloway, E. (2004). More than just fun and games: assessing the value of educational video games in the classroom. En: *Proceedings of CHI '04 extended abstracts on Human factors in computing systems*, Vienna, 1375–1378.

LeJeune, N. (2006). Teaching software engineering practices with Extreme Programming. En: *Journal of Computing Sciences in Colleges*, Vol. 21, No. 3, 107–117.

Leutenegger, S. (2006). A CS1 to CS2 bridge class using 2D game programming. En: *Journal of Computing Sciences in Colleges*, Vol. 21, No. 5, 76–83.

Mejía, M. A. (2006). *Educación(es) en la(s) globalización(es) I.* Ediciones desde abajo, Bogotá.

Miranda, M. G. (2000). El constructivismo como principio explicativo en la educación: una pretensión y un riesgo. En: *Educere*, Vol. 4, No. 10, 7–15.

Mitchell, W. (2004). Is Software Engineering for everyone?. En: *Proceedings of the 2nd annual conference on Mid-south college computing*, Little Rock, 53–64.

Morell, L. y Middleton, D. (2001). The software engineering learning facility. En: *Journal of Computing Sciences in Colleges*, Vol. 16, No. 3, 299–307.

Navarro, E. y Van der Hoek, A. (2004). *SimSE: an educational simulation game for teaching the Software engineering process*. En: Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education, Leeds, 233–233.

Nuseibeh, B. A. y Easterbrook, S. M. (2000). Requirements Engineering: A Roadmap. En: A. C. W. Finkelstein (ed) *The Future of Software Engineering*. (Companion volume to the proceedings of the 22nd International Conference on Software Engineering, ICSE'00), IEEE Computer Society Press.

Ocampo, J. F. (2003). El constructivismo en decadencia: de cómo fracasa su práctica pedagógica. En: *Educere*, Vol. 6, No. 21, 119–123.

ORACLE. (2000). *Oracle® MethodSM CDM Quick Tour*. Oracle Corporation, Redwood City.

OMG. (2007). *Object Management Group—Unified Modeling Language Specification*. Disponible en: http://www.omg.org/UML/

Pardo, A. y Delgado, C. (2005). Deploying interactive e-labs for a course on operating systems. En: *Proceedings of the 6th conference on Information technology education*, Newark, 71–78.

Parisi, D. (2002). An interactive ethics game. En: *Journal of Computing Sciences in Colleges*, Vol. 17, No. 5, 46–50.

Piaget, J. (1952). *The origins of intelligence in children (2nd ed.)*, International Universities Press, New York.

Pivec, M., Dziabenko, O. y Schinner, I. (2003). Aspects of Game-Based Learning. En: *Proceedings of I-KNOW 03, the Third International Conference on Knowledge Management*, Graz.

Pressman, R. S. (2002). *Ingeniería del Software: Un enfoque práctico, Quinta Edición.* McGraw Hill, Madrid.

Royce, W. (1970). Managing the Development of Large Software Systems: Concepts and Techniques. En: Proceedings of IEEE WESCON, Los Angeles, 1–9.

Rugarcia, A., Felder, R., Woods, D. y Stice, J. (2000). The Future of Engineering Education I: The vision for a new century. En: *Chemical Engineering Education*, Vol. 34, No. 1, 16–25.

Sanders, D. (2005). Software tools to support an objects-first curriculum. En: *Journal of Computing Sciences in Colleges*, Vol. 20, No. 4, 37–38.

Schaefer, S. y Warren, J. (2004). Teaching computer game design and construction. En: Computer-Aided Design, Vol. 36, 1501–1510.

Seffah, A. (2003). Learning the ropes: Human-Centered Design Skills and Patterns for Software Engineers' Education. En: *Interactions*, Vol. 10, No. 5, 36–45.

Senge, P. (1994). *The Fifth Discipline: The Art and Practice of the Learning Organization*. Currency Doubleday, New York.

Skinner, B. F. (1958). Teaching machines. En: *Science*, Vol. 128, 969–977.

Soloway, E., Guzdial, M. y Hay, K. (1994). Learner-Centered Design: The Challenge For HCI In The 21st Century. En: *Interactions*, Vol. 1, No. 2, 36–48.

Stiller, E. y LeBlanc, C. (2002). Effective Software Engineering Pedagogy. En: *Journal of Computing Sciences in Colleges*, Vol. 17, No. 6, 124–134.

Swartout, W. y Van Lent, M. (2003). Making a Game of System Design. En: *Communications of the ACM*, Vol. 46, No. 7, 32–39.

Sweedyk, E. y Keller, R. (2005). Fun and games: a new software engineering course. En: *ACM SIGCSE Bulletin*, Vol. 37, No. 3, 138–142.

Valentine, D. (2005). Playing around in the CS curriculum: reversi as a teaching tool. En: *Journal of Computing Sciences in Colleges*, Vol. 20, No. 5, 214–222.

Wankat, P.C. y Oreovicz, F.S. (1993). *Teaching Engineering*, McGraw-Hill, Nueva York.

Way, Th. (2005). A company-based framework for a software engineering course. En: *Proceedings of the 36th SIGCSE technical symposium on Computer science education*, St. Louis, 132–136.

White, R. W. (1959). Motivation reconsidered: the concept of competence. En: *Psychological review*, Vol. 66, 297–333.

Wick, M. (2004). Using the game of life to introduce freshman students to the power and elegance of design patterns. En: *Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*, Vancouver, 103–105.

Wick, M. (2005). Teaching Design Patterns in CS1: a Closed Laboratory Sequence based on the Game of Life. En: *Proceedings of the 36th SIGCSE technical symposium on Computer science education*, St. Louis, 487–491.

Zapata, C. M. (2007). Tutorial: Uso de Esquemas Preconceptuales para la generación automática de Diagramas de Clases, Comunicación y Máquina de Estados. En: *Memorias de las VI Jornadas de Ingeniería del Software e Ingeniería del Conocimiento*, Lima, 469–471.

Zapata, C. M. y Awad, G. (2005). El Juego de los Requisitos: Enseñanza de la Gestión de Proyectos de Software. En: *Memorias del XIII Congreso Iberoamericano de Educación Superior en Computación CIESC2005*, 33–43.

Zapata, C. M. y Gómez, M. C. (2006). *Ingeniería de Software: una Disciplina de Modelamiento.* Carlos Mario Zapata (Ed.), Medellín.

Zapata, C. M., Gelbukh, A. y Arango, F. (2006a). Pre-conceptual Schema: A Conceptual-Graph-Like Knowledge Representation for Requirements Elicitation. En: *Lecture Notes in Computer Science*, Vol. 4293, 17–27.

ANEXO: LA VISIÓN PARTICULAR DE UN PARTICIPANTE EN LOS JUEGOS

Uno de los grupos que se ha señalado como partícipe en la experiencia de los juegos ha sido el grupo de "Tópicos Avanzados en Ingeniería de *Software*", del cual, repetidamente, en este trabajo, se ha hecho alusión por la labor especial que se les encomendó a los estudiantes de este curso: la crítica y mejoramiento de los juegos en que participaron. En este anexo, se presenta la visión de uno de los ingenieros participantes en el curso, y que, a la vez, había participado ya en experiencias previas de algunos de los juegos. Es importante conocer esta opinión para mostrar las reflexiones que se pueden generar a partir de los juegos, cuando se combina la enseñanza tradicional, de la cual el ingeniero Muñetón también fue partícipe por su experiencia como estudiante de la Universidad Nacional de Colombia, con las experiencias lúdicas que se pretende promover mediante este trabajo. En su reflexión, el ingeniero Muñetón alude a cuatro de los juegos que se presentan en este trabajo: el juego de UN-Método, el juego de la consistencia, el juego del desarrollo de *software* y el juego de la planeación de proyectos de *software*. El quinto juego es el juego de los requisitos (Zapata y Awad, 2005), del cual se puede encontrar una descripción completa en el artículo referenciado.

ENSEÑANZAS Y PROPUESTAS DE MEJORAMIENTO A LOS JUEGOS DE TÓPICOS AVANZADOS EN INGENIERÍA DE *SOFTWARE*

Por: Andrés Felipe Muñetón Lopera

I. Introducción

En el curso "Tópicos Avanzados de Ingeniería de *Software"* de la maestría en Sistemas de la Universidad Nacional de Colombia, Sede Medellín, se desarrollaron cinco juegos agrupados así: tres sesiones con el juego de UN-Método, el juego de la consistencia y el juego de los requisitos, y dos sesiones con el juego del desarrollo de *software* y el juego de la planeación. Este documento se divide en dos partes: en la parte II, se presentan diez aspectos valiosos de los juegos y propuestas de

modificación que permitan aumentar el contenido educativo del primer grupo de juegos y, en la parte III, se presentan las enseñanzas y propuestas de cambios para el segundo grupo de juegos.

II. Enseñanzas y propuestas: primer grupo de juegos

1. Contextualizando. Los juegos fueron un buen complemento para concretar los conceptos analizados en las exposiciones y los relacionados con UN-Método. Sin embargo, considerando la baja calidad de algunas exposiciones, debido en parte a la dificultad para entender los conceptos, es muy posible que las didácticas no hayan sido suficientes para refinar las ideas principales. Sería importante una fase de conceptualización mayor a la actual, donde se presenten situaciones reales que sirvan como iniciadoras en la construcción individual de los conceptos, para luego ponerlos a prueba en los juegos mediante socialización.

2. Pensamiento sistémico. Cada uno de los juegos, especialmente los de requisitos y consistencia, muestran la importancia de tener presente que en cada fase y en cada paso se modela el mismo sistema, pero con diferentes notaciones, puntos de vista y objetivos: entender, estructurar y dinamizar. Para ello, es fundamental tener presente en todo momento la importancia del refinamiento consistente de y entre modelos. Sería interesante que se hiciera un estudio de los modelos generados, promoviendo el refinamiento a otro modelo siguiendo ciertas reglas definidas por el director del juego. Por ejemplo, el juego de los requisitos podría modificarse, al menos en su primera parte, para que a partir de la descripción de un caso de uso se generen las interfaces gráficas de usuario. Esto, además, motivaría la participación de ingenieros, además de los de sistemas, a adoptar roles como el de programador.

3. Programar… al final. Una conclusión muy importante que surge con las didácticas, es la relevancia que tiene la aplicación de una metodología desde las fases iniciales del proyecto de *software*. Entender qué puede obtenerse con cada diagrama y en qué parte de la secuencia de la metodología debe aplicarse lleva a inferir cómo debe darse el proceso de análisis y construcción del producto: conocer, entender, abstraer e implementar. Enfatizando en la importancia de conocer el proceso, los juegos deberían estar enlazados por un objetivo común, que sería el desarrollo de un proyecto simple de *software*. En el primer juego se trataría de entender el sistema; en el segundo, se vería globalmente cómo continuaría el proyecto; y, finalmente, en el juego de los requisitos, se implementaría la documentación generada hasta el momento, que serviría como evaluación a lo hecho en los juegos anteriores.

4. La otra ingeniería. Los juegos permiten apreciar uno de los rasgos más característicos de la Ingeniería de *Software* y es la informalidad con la que aún se desarrollan los productos, lo cual marca una seria diferencia con las demás ingenierías, donde se trabaja con leyes consideradas como absolutas y procedimientos con pocos grados de libertad. Pero, los juegos dejan ver el esfuerzo que a diario se hace por la formalización de las metodologías y las herramientas, y que se está consiguiendo desde la ampliación de los dominios del área hacia la lógica y la investigación de operaciones, entre otras. Una mejora a la metodología consistiría en hacer una mejor revisión de los diagramas hechos en cada juego que, considerando el nivel promedio del grupo, serviría, más que como elemento calificador, para construir con la guía del profesor un modelo que pueda considerarse bueno. El profesor podría representar las reglas básicas que muestren que, en realidad, existen relaciones entre los diagramas y que puede ser posible formalizarlas.

5. Es un proyecto. La heterogeneidad de los grupos, debido a las diferentes áreas que participan en el curso, y las dinámicas de los juegos permiten entender el desarrollo de un producto de *software* como un proyecto, en el que, claramente, surgen diversas necesidades de tipo técnico y gerencial, y donde el tiempo es prioritario y parece ser inversamente proporcional a la satisfacción del cliente. En el desarrollo de cada juego, se notó la adopción de roles por los miembros de los equipos, algunas veces afines a sus áreas de trabajo, que, en algunos casos, permitieron conseguir la mayoría de objetivos planteados y, lo más importante, dar un paso más en el entendimiento de la importancia de las buenas prácticas de la Ingeniería de *Software*. En este caso, podría plantearse una mejora en el juego de los requisitos, que consistiría en dividirlo en intervalos de tiempo para cada fase del desarrollo (más que las del juego actual). En cada fase participarían roles diferentes del grupo y, para pasar a una siguiente etapa, debería contarse con la aprobación del cliente, que revisaría en detalle los modelos y pediría en algún momento cambios a lo establecido hasta el momento, con el fin de medir la calidad de los modelos.

6. Tiempo-calidad. Estos dos elementos son claves en el juego de los requisitos. A pesar de tener identificados los problemas y sus causas, ningún grupo se enfocó en lo relevante del producto, aquello que satisficiera las necesidades más prioritarias del cliente. Sin embargo, en la segunda parte del juego, sólo un grupo no cumplió con el objetivo. Es claro que, con los resultados de la primera fase, los grupos identificaron los elementos claves para organizarse y conseguir los objetivos planteados. La propuesta de mejoramiento consistiría en una introducción al grupo completo sobre la administración de recursos del proyecto, pues, aún con la experiencia en el juego,

las conclusiones tienden a orientarse al problema como tal, pero no a sus causas: tenemos problemas de organización, pero, entonces, ¿cómo puede organizarse un proyecto de *software*? Creo que esta pregunta, tan relevante, debería, al menos, discutirse por el grupo como lineadora del objetivo del juego de los requisitos.

7. La utopía. Los juegos, especialmente el de los requisitos, permiten tener cierto acercamiento al mundo real de los proyectos de *software*, que siempre está poblado por las ambigüedades de los clientes y donde el tiempo pasa más rápido de lo normal y el manejo de los demás recursos termina siendo más complejo que el *software* mismo. Las metodologías se han desarrollado para acabar con los dolores de cabeza del equipo de *software* y de los usuarios (lo que marca el punto de partida del problema). Pero, estos no han querido entenderlo aún: no hay usuarios perfectos. Sería interesante iniciar la participación de un cliente desde la obtención misma de los requisitos.

8. Se requiere ingeniero de sistemas. No es necesario que sepa programar. Los juegos desarrollados, al simular entornos reales, enseñan con la práctica las sinergias que se pueden dar entre los miembros de diferentes áreas. Una de las mayores ganancias es la posibilidad de encontrar el aporte que puedes dar gracias a tu visión del mundo, según tu formación académica. Desde la Ingeniería de Sistemas, la contribución es mayor gracias al enfoque holístico en lugar de la generación de un código, actividad en la cual siempre se ha encasillado a los ingenieros de sistemas. Como elemento motivador y educativo, una mejora a los juegos sería proponer proyectos más afines a las diferentes áreas de la maestría. Esto puede llevar a una concepción más clara de los roles en los juegos y una participación más activa en los mismos. Por ejemplo, se podría abordar un proyecto donde deban considerarse los aspectos administrativos, de procesos (afín a la Ingeniería Industrial) y que busque alguna solución simple en otra área.

9. Del modelo a la realidad. Una de las mayores ventajas obtenidas con los tres juegos es que, en poco tiempo, los participantes pudieron recorrer, aunque fuese de manera informal, todo el proceso de desarrollo de *software* hasta ver materializados en una aplicación de *software* las abstracciones hechas mediante los diagramas ofrecidos por la metodología. Este privilegio lo tienen pocos, especialmente, porque pueden captar la importancia de cada fase, la seriedad en el modelado y las diferencias existentes con lo visto en los cursos de modelado para las ingenierías en general. El juego de la consistencia es el representante de este numeral. Considero que el juego es bastante bueno, pero podría mejorar agregando una parte socializadora, donde se pueda aprovechar la experiencia de los ingenieros de sistemas orientados al área de *software*. Es común que las

conclusiones más relevantes queden en grupos reducidos, cuando debería ser un recurso para todo el grupo. Además, permitiría generar discusiones, muy importantes en la concreción de conceptos, pues las inquietudes e ideas de personas de otras áreas pueden chocar con lo que creemos tener muy claro, pero que no ha salido de un dominio de pensamiento muy similar.

10. Y… ¿qué es la Ingeniería de *Software*? El desarrollo del conocimiento se acelera cuando pensamos acerca del conocimiento mismo. Si no aplicamos nuestros saberes con sentido, no tendrá sentido cuanto pueda aprenderse sobre requisitos, diseño o programación, o sobre los pasos que hay que seguir en el desarrollo de un *software* porque así lo dictamina una metodología. La discusión conductora del aprendizaje de la Ingeniería de *Software* debe enfocarse en entender qué es la Ingeniería de *Software*, qué pretende y por qué a diario estamos velando por su desarrollo. Creo que los juegos desarrollados en el curso, fuera del aspecto didáctico, tuvieron un alto contenido educativo, que no debe medirse sólo en los ingenieros ajenos al área, sino también, y me arriesgo a proponer que, en mayor medida, en aquellos que hemos degustado (quisiera decir algún día conocedores), que gracias a las nuevas interacciones y nuevos puntos de vista pudimos avanzar un paso más en nuestra formación como ingenieros de *software*. Los juegos pueden aportar en este sentido teniendo dentro de sus objetivos la calidad, vista como la satisfacción de nuestro cliente.

III. Enseñanzas y propuestas: segundo grupo de juegos

A. Enseñanzas

1. Personal capacitado: de ambos juegos, especialmente del juego del desarrollo de *software*, se infiere la necesidad de contar con personal con las competencias necesarias para su asignación a los diferentes tipos de proyectos con los que se puede comprometer la compañía.

2. Factores externos: una empresa de *software* está muy expuesta al entorno. Es necesario estar preparado, estando informado de los factores externos, como la competencia, la dinámica del medio en el que se mueve la compañía y, además, otros factores que, aunque no se relacionen directamente con la empresa, pueden afectarla de una u otra forma. Sin embargo, es un hecho que no es posible tener en cuenta todo lo que pueda ocurrir, o que algunos eventos, simplemente, no se pueden predecir, ni evitar. Depende de los líderes de la empresa fomentar en todo el equipo de trabajo la capacidad de pensar en las posibles situaciones por improbables que parezcan en la actualidad.

3. Calidad: velar por la calidad es fundamental en un proyecto de cualquier tipo, y eso pudo verse claramente en el juego del desarrollo de *software*. El resultado de este juego demostró la preocupación por los equipos en producir la mayor cantidad de cajas (modelos, código, interfaces gráficas), y no tuvieron en cuenta que podría ser más importante disminuir el ritmo de producción hasta que se tuviera controlada la calidad del producto, lo cual al final, fue un gran error: tiempo y recursos perdidos.

4. Cumplir con los requisitos: los requisitos son los deseos del cliente. Y si el objetivo del proyecto de *software* es solucionar los problemas del cliente, ¿por qué no tenerlos en cuenta en todo momento? En el proceso de desarrollo, las especificaciones iniciales parece que se diluyeran en las diferentes abstracciones hechas sobre ellas mismas, hasta que el producto, el *software*, o una caja, no es la deseada por el cliente. Al respecto, es importante contar con un elemento dentro del grupo encargado de hacer verificaciones en todo momento, ya que los demás miembros tienden a concentrarse tanto en su labor tan específica que pierden el enfoque global del sistema.

5. Retroalimentación constante: Ambos juegos mostraron la importancia de tener ciclos de retroalimentación en el proyecto. En el juego del desarrollo porque significaba el mejoramiento del proceso de construcción de las cajas, y en el juego de la planificación porque los eventos provocaban la reconsideración de las estrategias definidas para obtener los objetivos.

6. El cliente: Tal vez la mejor enseñanza que deja el juego de la construcción es que el cliente es parte del proyecto de desarrollo, y muy importante, pues, tiene la imagen de lo que el grupo pretende hacer, aunque no sea muy clara. Al respecto, el grupo debe ayudar a definir exactamente qué es lo que quiere, y la única forma de hacerlo es estar en constante comunicación con el cliente, donde haya una comparación entre lo que él quiere y lo que nosotros pensamos, hasta que sean tan similares ambas ideas que se pueda considerar alcanzado el objetivo del proyecto.

7. El gerente: En estos dos juegos, como en los anteriores, se notó la importancia de un buen gerente, que, en estos casos, podría verse como coordinador de proyecto. El equipo azul contó en estos juegos con el gerente más activo y más responsable en sus tareas y, particularmente, fueron los equipos ganadores. En cambio, el equipo verde, del cual fui miembro, fue muy desordenado en estos dos juegos donde era vital la participación activa de su gerente, diferente a la mayoría de los anteriores donde el peso recaía sobre los diseñadores y programadores.

8. Estrategia: con el fin de cumplir los objetivos de un proyecto, es fundamental contar con una buena estrategia que considere cada una de las actividades con sus recursos y duración, el tiempo total del proyecto, los posibles retrasos, etc. Inicio diciendo "con el fin de.." porque, desgraciadamente, no es nada seguro, siempre existe algo que puede ocurrir, y que, al no estar dentro del cronograma de trabajo, trae consigo el caos. Algo curioso en el juego es que, aunque el equipo azul ganó, el equipo verde que fue el equipo con la peor estrategia, la supo atacar muy bien una vez iniciado el proyecto, lo cual les permitió ocupar el segundo lugar. En cambio, otros equipos tomaron malas decisiones que restringieron sus posibilidades de avance en su trabajo, y, por tanto, el azar parecía estar cada vez más en su contra.

9. ¿Qué hace nuestra empresa?: este punto tiene dos aspectos. Para el juego del desarrollo de *software*, aunque igual había que desarrollar las cajas, en un entorno real lo más sano habría sido considerar si realmente debíamos intentar el desarrollo de un producto tan diferente de lo que normalmente hemos desarrollado (y no quiero que se piense en la lógica diferencia entre una caja y un *software*, sino en *software* de diferentes dominios). Y para el juego de la planeación, dentro de la estrategia, debía desglosarse ese concepto de "desarrollamos cajas de cartón" en sus actividades (cortamos la hoja, doblamos la hoja según las instrucciones, etc.) de la empresa.

10. Manejo del tiempo: en estos y todos los juegos el manejo del tiempo es fundamental. Es necesario organizarse, tomar decisiones y actuar rápidamente. El problema es que en ocasiones se teme caer en ideas poco meditadas, que puedan llevar al fracaso del proyecto.

B. Propuesta de cambios

1. Que deba crearse una caja y prestarla al cliente para validación y para poder continuar con el trabajo (juego del desarrollo).

2. Posibilidad de compra de miembros clave (el que sepa armar las cajas, el buen gerente) de otros grupos, beneficiando al vendedor, económicamente (puntos), y esperando qué puede ocurrir en el comprador (ambos juegos).

3. Posibilidad de contratar capacitaciones (juego del desarrollo).

4. Agregar al juego de las especificaciones eventos más cercanos a la realidad: leyes, factores ambientales, etc., y sus consecuencias para el equipo que lo sufre, o para todos. Esto aumentaría la necesidad de analizar las decisiones.

5. Agregar diferentes ventajas en los grupos que dé indicios de los elementos más relevantes dentro del proyecto de *software*.

6. En ambos juegos, premiar los equipos en los cuales se utilice una metodología de trabajo con pasos bien definidos.

7. En el juego de la planeación, no sólo considerar las fases, sino las actividades, al menos las más relevantes.

8. En el juego de la planeación, el castigo no debería ser quitar puntos, sino que deberían agregarse retrasos en el proyecto y pérdida de personal.

9. En el juego de la planeación, al final de cada fase terminada, debería simularse la entrega de la documentación de esta fase y la retroalimentación con el cliente mediante preguntas hechas al grupo sobre los productos que debieron obtener y, dependiendo de la satisfacción del cliente ante las respuestas, permitir o no su continuación con la siguiente fase. Las preguntas pueden formularse por parte de los otros grupos.

10. Promover una mayor participación del gerente mediante un informe sobre las decisiones tomadas durante su gestión.