Una Propuesta de Análisis de Necesidades mediante las Gráficas de Dependencia en la fase de Estrategia de TSPi[SM(1)]

Hanna Oktaba y Jorge L. Ortega Arjona
Departamento de Matemáticas
Facultad de Ciencias, UNAM
{ho,jloa}@fciencias.unam.mx

Resumen

Se propone una técnica de análisis de las necesidades funcionales del cliente, basado en las Gráficas de Dependencia entre Necesidades, a fin de facilitar el desarrollo de la fase de Estrategia durante un curso de TSPi. Se contrasta este análisis con un desarrollo de tal fase por parte de estudiantes de licenciatura. La técnica de análisis de necesidades se propone para utilizarse a fin de mejorar el desempeño de grupos de estudiantes en un curso basado en TSPi.

1. Introducción.

La Introducción al Proceso de Software por Equipos (*Introduction to the Team Software Process*[SM], o simplemente TSPi[SM]) es propuesto por Watts Humphrey [2] como una guía de las fases a seguir para un curso de proyectos de software. La guía hace una definición de roles para los miembros de un equipo, y muestra cómo aplicar el conocimiento de la Ingeniería de Software en un ambiente de trabajo en equipo. En la actualidad, TSPi se ha propuesto y empezado a utilizar en varias universidades, como un marco de trabajo para cursos de Ingeniería de Software [1, 3, 4, 5, 6, 7].

Debido a que las actividades de TSPi se desarrollan en forma incremental e iterativa, en general, las decisiones tomadas durante las primeras fases y ciclos tienden a tener un considerable impacto sobre cómo se desarrollan las fases y ciclos subsecuentes. En particular, se ha notado en la práctica que al aplicar TSPi en cursos a nivel licenciatura, las decisiones hechas por los estudiantes durante la fase de Estrategia en el primer ciclo tienen una fuerte influencia sobre el desarrollo de un proyecto de software, y sobre el desempeño de los estudiantes durante el curso. Esto puede deberse a la manera como inicialmente los estudiantes conciben y entienden las necesidades planteadas. Al evaluar los productos generados durante el curso, aquellos grupos de estudiantes que parecen haber obtenido una comprensión más clara de la *Declaración de Necesidades* y de los *Criterios de la fase de Estrategia* [2], tuvieron un mejor desempeño final. Esto, en cierto modo, plantea una desventaja del uso de TSPi en cursos de Ingeniería de Software, ya que tal comprensión recae en la forma como los estudiantes analizan las necesidades y criterios. Este análisis actualmente se realiza con base a la intuición y escasa experiencia de los estudiantes.

Con el objetivo de sistematizar la forma de analizar la Declaración de Necesidades, a fin de satisfacer los Criterios de la fase de Estrategia, este artículo propone una técnica de análisis basado en *Gráficas de Dependencia entre Necesidades*. Tales gráficas se construyen para describir las relaciones de dependencia entre necesidades funcionales. Esto pretende servir como apoyo para el análisis de necesidades, y para el cumplimiento de los criterios.

El artículo se organiza de la siguiente forma: primero, se introducen conceptos básicos del desarrollo de TSPi, y en particular, una descripción de la fase de Estrategia. A continuación, se describe cómo se desarrolla un experimento que ha servido como base para observar y desarrollar la técnica de análisis. En seguida, se presenta el análisis de necesidades, introduciendo las Gráficas de Dependencia entre Necesidades, los pasos para su construcción, y su relación con los Criterios de la fase de Estrategia. Con base en tales gráficas, se presenta un análisis de las decisiones tomadas por los grupos de estudiantes

1 *Introduction to the Team Software Process* y TSPi son marcas de servicio (*service marks*) de Carnegie Mellon University

durante el experimento, revisando cómo las gráficas pueden describir la forma en que los estudiantes intuitivamente interpretan las necesidades y toman decisiones.

1.1. Desarrollo cíclico de TSPi.

TSPi se basa en una aproximación de desarrollo cíclico, que inicia con una versión básica del producto de software, y continúa en una secuencia de ciclos. Cada ciclo subsecuente genera una versión ampliada y mejorada del producto. Así, el producto final es consecuencia de los desarrollos parciales que se realizan como producto de cada ciclo.

Un ciclo de TSPi consiste en la siguiente secuencia de fases: Lanzamiento, Estrategia, Planeación, Requerimientos, Diseño, Implementación, Evaluación y Postmortem. El interés de este trabajo se enfoca en la fase de Estrategia, que se describe a continuación.

1.2. Descripción de la fase de Estrategia de TSPi.

El objetivo de la fase de Estrategia de TSPi es construir un diseño conceptual del producto de software, estimar su tamaño y tiempo necesario para su elaboración, basándose en la poca información incluida en la declaración de las necesidades para el sistema. El resultado de la elaboración de la Estrategia sirve de entrada a la fase de Planeación. En general, la elaboración de la fase de Estrategia parte de manera implícita de dos principios ampliamente aceptados para el desarrollo de software:

1. *Divide y vencerás para el producto.* Cada producto de software se ensambla a partir de una colección de componentes.
2. *Divide y vencerás para el proceso.* Un producto se desarrolla en varios ciclos de manera incremental.

1.3. Actividades del proceso de la fase de Estrategia.

La fase de Estrategia se desarrolla a partir de las siguientes actividades [2]:

1. Seleccionar los elementos de la declaración de necesidades para el producto que se considera factible de abarcar en el tiempo estipulado para el proyecto.
2. Tomando en cuenta el número de ciclos previstos para el proyecto, seleccionar las necesidades a cubrir en cada ciclo.
3. Realizar el diseño conceptual para el conjunto de necesidades seleccionado.
4. Estimar el tamaño y el número de horas necesarias para realizar cada uno de los componentes.
5. Analizar la carga de trabajo por ciclo y tratar de balancearla.

1.4. Criterios para evaluar la Estrategia.

Para evaluar el resultado de la fase de Estrategia se proponen los siguientes criterios [2]:

1. El producto del primer ciclo debe de proporcionar un subconjunto de funcionalidades mínimas del producto final.
2. El producto del primer ciclo debe de proporcionar una base fácilmente mejorable.
3. El diseño de producto tiene una estructura modular que permite trabajo independiente de los miembros del equipo.
4. Los productos de cada ciclo deben de ser de alta calidad y pueden ser fácilmente probados.

Es importante observar que los tres primeros criterios se refieren al modo en que se desarrolla el producto en cada ciclo, mientras que el cuarto criterio considera la calidad con la que deben ser dotados tales productos. La técnica que se propone a continuación, basada en el uso de Gráficas de Dependencias entre Necesidades, se enfoca en cómo se planea el desarrollo del producto, y por lo tanto, atiende los tres primeros criterios.

2. Las Gráficas de Dependencias entre Necesidades.

Para poder cumplir con los criterios de la fase de Estrategia es necesario analizar la declaración de necesidades para el problema dado. Sin embargo, TSPi no proporciona ninguna guía práctica para realizar tal análisis. Las Gráficas de Dependencias entre Necesidades, que se proponen a continuación, pretenden servir como una técnica para el análisis de las necesidades funcionales.

Una Gráfica de Dependencias entre Necesidades es una gráfica dirigida con dos tipos de vértices, que representan las relaciones de dependencia total o parcial entre las necesidades funcionales de un problema dado. Se construye a partir de los siguientes pasos:

1. Se identifican las necesidades independientes, colocándose como nodos terminales de la gráfica. Una necesidad ***a*** se considera independiente si su satisfacción no depende de la satisfacción de ninguna otra necesidad.
2. Se analizan, una por una, el resto de las necesidades, revisando si su satisfacción tiene dependencia total de la satisfacción de otra necesidad previamente colocada en la gráfica. Una necesidad ***a*** depende totalmente de otra necesidad ***b*** si la satisfacción de ***a*** requiere previamente de la satisfacción completa de la necesidad ***b***. Si este es el caso, la necesidad ***a*** se coloca en la gráfica como un nuevo nodo, unido por un vértice dirigido hacia el nodo de la necesidad ***b*** de la cual depende totalmente.
3. Se repite el paso anterior hasta colocar como nodos todas las necesidades.
4. La gráfica resultante se analiza, a fin de identificar los nodos cuyas necesidades se satisfagan parcialmente a partir de la satisfacción de otros nodos. Una necesidad ***a*** depende parcialmente de otra necesidad ***b***, si ***a*** puede satisfacerse de modo incompleto mediante la satisfacción completa de la necesidad ***b***. En estos casos, la necesidad ***a*** se conecta mediante un vértice dirigido (trazado como una línea discontinua) hacia el nodo de la necesidad ***b***, de la cual depende parcialmente.

Es importante hacer notar que el análisis se realiza a partir de una descripción de las necesidades funcionales en lenguaje natural, por lo que es susceptible a distintas interpretaciones. En consecuencia, es de esperar que este tipo de análisis pueda generar más de una gráfica representativa del problema.

A continuación se presenta una traducción de la Declaración de Necesidades para el problema de Contador de Cambios, definido por Humphrey [2]. Esta declaración se usa como ejemplo para generar una Gráfica de Dependencias entre Necesidades para este problema.

2.1. Declaración de Necesidades Funcionales para un Contador de Cambios.

"*Un Contador de Cambios es una herramienta de software para contar el tamaño y modificaciones de un programa. Cuando se modifican programas o se desarrollan programas en varios ciclos, es importante saber cuantas líneas de código (LOC) han sido añadidas, eliminadas o modificadas entre una versión y otra. Si no se cuenta con esta herramienta, es necesario el conteo manual de estos cambios*" [2].

En seguida, se presentan la declaración de las necesidades funcionales del Contador de Cambios [2]:

1. *Compara la versión modificada de un programa con la anterior.*
2. *Identifica las LOC añadidas y eliminadas en el programa modificado.*
3. *Cuenta las LOC añadidas y eliminadas en el programa modificado.*
4. *Cuenta el número total de LOC en el programa modificado.*
5. *Agrega una etiqueta de referencia en la línea para indicar el número de cambio.*
6. *Proporciona una etiqueta de cambio en el comentario de encabezado del programa indicando el número de cambio, la fecha en la cual se hizo el cambio, quién hizo el cambio, porque se hizo el cambio, y el número de LOC añadidas, eliminadas y el total.*
7. *Si el programa sufrió varios cambios mantener el registro de todos los cambios en el encabezado.*
8. *Cuando un programa previamente modificado es nuevamente modificado y contado, todos los registros de las modificaciones previas tienen que mantenerse.*

9. *Produce un nuevo archivo con el programa fuente de la versión modificada incluyendo etiquetas de cambio en el encabezado y etiquetas de línea modificada.*
10. *Diseña inicialmente el programa para trabajar sólo con uno de los lenguajes de programación (por ejemplo, Ada, C++, C, Pascal, Java).*
11. *Haz que la versión completa del programa sea capaz de contar cambios por lo menos para tres lenguajes.*
12. *Imprime, a solicitud, el listado del programa incluyendo ambos tipos de etiquetas.*
13. *Imprime, a solicitud, el listado del programa modificado con el número de cada línea al principio.*
14. *Imprime, a solicitud, el reporte de cambios del programa con las estadísticas de todos los cambios sufridos.*

2.2. La Gráfica de Dependencias entre Necesidades para el Contador de Cambios.

Al aplicar los pasos para generar la Gráfica de Dependencias entre Necesidades para la Declaración de Necesidades del ejemplo del Contador de Cambios, se obtiene la gráfica que se muestra en la Figura 1.

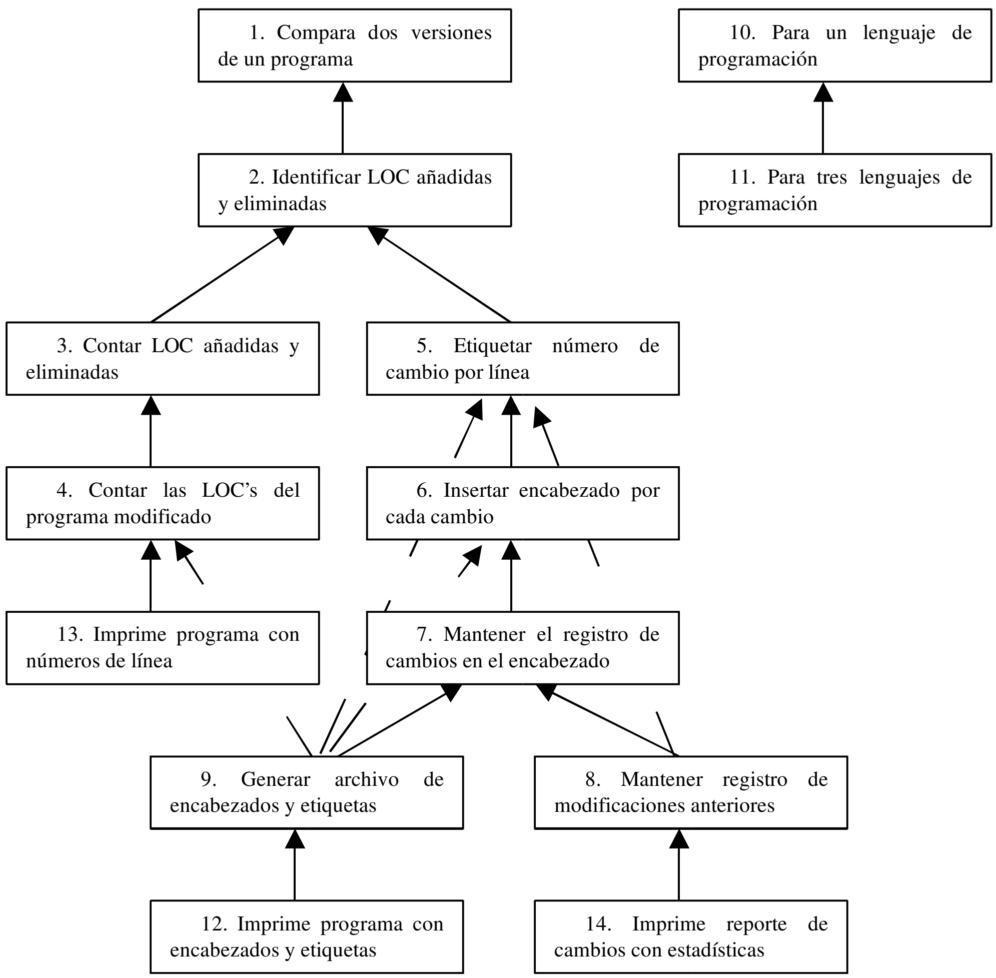

Figura 1. Gráfica de Dependencias entre Necesidades para el ejemplo del Contador de Cambios.

La utilidad de este gráfica tiene que ver propiamente con el cumplimiento de los tres primeros Criterios de Estrategia. A continuación se enuncian tales criterios, y se analiza cómo la Gráfica de Dependencias entre Necesidades se usa a fin de cumplirlos para el caso del Contador de Cambios.

1. *El producto del primer ciclo debe de proporcionar un subconjunto ejecutable de funcionalidades mínimas del producto final.* La gráfica permite revisar y seleccionar las funcionalidades mínimas del primer criterio de la fase de Estrategia, mediante seleccionar los nodos terminales (no necesariamente todos) y una o mas trayectorias, de poca longitud, que se dirijan a esos nodos. Los nodos incluidos dentro de las trayectorias representan necesidades a satisfacer en el primer ciclo. En el caso del Contador de Cambios, se proponen los nodos 1, 2 y 10 para definir las funcionalidades mínimas. Esto significa que el producto del primer ciclo en este caso tenga la capacidad de comparar dos versiones de un programa en un lenguaje de programación particular, identificando las líneas de código (LOC) añadidas o eliminadas. Los nodos 1 y 10 se toman como parte de la funcionalidad mínima dado que son nodos terminales. Sin embargo, el nodo 2 se hace necesario dado que el nodo 1 establece *qué* hacerse (comparar versiones de un programa), mas no *cómo* hacerlo, que es donde entra el nodo 2 (comparar mediante identificar las líneas añadidas o eliminadas). De hecho, es posible hacer una selección de más nodos para determinar una funcionalidad, pero debido a la dependencia entre las necesidades, en cualquier caso siempre tendrá que cumplirse primero con las necesidades de los nodos 1, 2 y 10 como funcionalidad mínima.

2. *El producto del primer ciclo debe de proporcionar una base fácilmente mejorable.* En este criterio, no resulta claro cómo determinar si el producto es "fácilmente mejorable". Esta frase es poco precisa para representarse en el contexto del primer ciclo, por lo que se propone para los fines de este artículo considerar más bien al producto del primer ciclo como "extendible". Así, partiendo de los nodos seleccionados en la Gráfica de Dependencias como parte de las funcionalidades mínimas, es posible ir añadiendo otros nodos dependientes durante los siguientes ciclos. De esta forma, el esfuerzo de desarrollo (costo y trabajo) por cada ciclo se aprovecha e incrementa de manera mas eficiente. Para el ejemplo del Contador de Cambios, la gráfica muestra a las necesidades 3, 5 y 11 como candidatas posibles para desarrollarse a continuación, extendiendo la funcionalidad mínima de identificar líneas añadidas o eliminadas, mediante contarlas o etiquetarlas, o bien, para otro(s) lenguaje(s). Si bien, es posible considerar las tres extensiones concurrentemente, ya que la propia gráfica muestra que no hay dependencia entre estas necesidades. El producto se va extendiendo a través de los diferentes ciclos, conforme se considera el cumplimiento de nuevos conjuntos de necesidades.

3. *El diseño de producto tiene una estructura modular que permite trabajo independiente de los miembros del equipo.* La gráfica sugiere una probable división de trabajo. Observando aquellos nodos que tienen más de un vértice de entrada (necesidades cuya satisfacción habilita la satisfacción de dos o más necesidades), sus nodos descendientes representan potencialidad de desarrollo independiente dentro de la estructura del producto. En el caso del Contador de Cambios, se hace claro que las necesidades 1, 2 y 10 deben trabajarse en conjunto, dado que definen la funcionalidad mínima. Sin embargo, ya que no hay dependencia entre las siguientes necesidades a cubrir, la labor puede subdividirse y realizarse concurrentemente, en forma modular.

3. Un Experimento Didáctico con TSPi.

Partiendo de la Gráfica de Dependencias entre Necesidades para el Contador de Cambios, es posible analizar las causas de las diferencias entre estrategias propuestas por grupos de estudiantes. Los estudiantes intuitivamente interpretaron las necesidades del problema, y a partir de ellas tomaron diferentes decisiones. El experimento consistió en usar TSPi como base para el curso de Ingeniería de Software para alumnos del sexto semestre de la Licenciatura en Ciencias de la Computación, en la Facultad de Ciencias, UNAM. Durante el curso, el trabajo de los alumnos fue desarrollar un programa que resuelva el problema del Contador de Cambios, siguiendo el proceso definido por TSPi. Al evaluar los sistemas finales presentados por los grupos, se observó que cada uno presentó una solución diferente, tanto en la funcionalidad expresa como en la documentación producida durante dos ciclos de TSPi.

Observando y analizando los sistemas finales y la documentación, surgen cuestionamientos y conjeturas acerca de las diferencias y las causas de tales diferencias, lo que ha motivado la propuesta de análisis de la fase de Estrategia mediante Gráficas de Dependencias entre Necesidades. A continuación se presentan tres casos (grupos A, B y C) tomados del experimento. La Tabla 1 muestra la selección de necesidades realizada por cada grupo durante la fase de Estrategia en los primeros dos ciclos.

Tabla 1. Selección de Necesidades por equipo durante la fase de Estrategia

Grupo A		***Grupo B***		***Grupo C***	
Ciclo 1	**Ciclo 2**	**Ciclo 1**	**Ciclo 2**	**Ciclo 1**	**Ciclo 2**
1	3	1	8	1	2
2	4	2	9	3	7
5	6	3	12	4	9
9	7	4	13	5	
10	8	5	14	8	
	13	6		10	
	14	7			
		10			

El análisis y comparación entre los resultados obtenidos por los grupos de estudiantes se hace con base en los tres primeros Criterios de Estrategia. A continuación, se presenta el análisis, como sigue:

1. *El producto del primer ciclo debe de proporcionar un subconjunto ejecutable de funcionalidades mínimas del producto final.* Para el Contador de Cambios, los nodos 1,2 y 10 se proponen como las necesidades a cubrir por un producto con funcionalidad mínima (véase sección 2.2). Al observar las selecciones de necesidades de los estudiantes para el primer ciclo, los grupos A y B identificaron los nodos 1, 2 y 10 como parte de la funcionalidad mínima. Sin embargo, la selección hecha por el grupo C hace pensar que los estudiantes no detectaron la dependencia del nodo 3 (contar las líneas añadidas y eliminadas) y del nodo 5 (etiquetar número de cambio por línea) con respecto al nodo 2 (identificar líneas añadidas y eliminadas) (véase Figura 1). De este modo, los grupos A y B satisficieron el criterio de la obtención de un primer producto con funcionalidad mínima, mientras que el grupo C no lo logró.

2. *El producto del primer ciclo debe de proporcionar una base fácilmente mejorable.* Como se discutió anteriormente, se considera que el producto es mejorable en tanto sea extendible (véase sección 2.2), lo que significa que la funcionalidad mejora al ir añadiendo y satisfaciendo los nodos dependientes durante los siguientes ciclos. Tomando como base la gráfica de la Figura 1, en general se debe procurar que los nodos seleccionados incluyan y extiendan la funcionalidad mínima a lo largo de las trayectorias de dependencia total o parcial propuestas en la gráfica. El grupo A extiende la funcionalidad mínima al añadir los nodos 5 y 9 en el primer ciclo, y los nodos 3, 4, 6, 7, 8, 13 y 14 durante el segundo ciclo. Nótese que tal selección de nodos sigue las trayectorias (de dependencia total o parcial entre nodos) planteadas en la gráfica de la Figura 1. En el caso del grupo B, la extensión a la funcionalidad mínima se realiza de manera similar, pero tomando en cuenta otros conjuntos de nodos. Para el primer ciclo, toman los nodos 3, 4, 5, 6 y 7, y para el segundo ciclo los complementan con los nodos 8, 9, 12, 13 y 14. Nótese que esta selección continúa siguiendo trayectorias que se apegan a dependencias totales de la gráfica. El grupo C, dado que considera equivocadamente posponer el cumplimiento del nodo 2 para el segundo ciclo, obtiene una extensión durante el primer ciclo añadiendo los nodos 3, 4, 5 y 8, que tienen una relación de dependencia total, exceptuando la relación entre los nodos 5 y 8, que tienen una dependencia parcial. En el segundo ciclo, consideran los nodos 2, 7 y 9. A pesar de la equivocación de considerar el nodo 2 hasta el segundo ciclo, los otros nodos seleccionados continúan siguiendo las trayectorias de la gráfica, con dependencias totales. Se puede concluir que los tres grupos (A, B y C) cumplen con el criterio de que los productos del primer ciclo son extendibles durante el segundo ciclo.

3. *El diseño de producto tiene una estructura modular que permite trabajo independiente de los miembros del equipo.* La gráfica para el Contador de Cambios sugiere una probable división de trabajo, particularmente en los nodos que tienen más de un vértice de entrada, como son 2, 4, 5, 6 y 7. Los nodos descendientes de cada uno de estos nodos presentan potencialidad de desarrollo concurrente dentro de la estructura del producto. Nótese que la división del trabajo también se afecta si la dependencia entre el vértice y los descendientes es total o parcial. Así se tienen los casos de los nodos 2 y 7, cuyas dependencias son todas totales; mientras que los nodos 4, 5 y 6 presentan ambos tipos de dependencias. Respecto a esto, el grupo A eligió los nodos 2 y 5 durante el primer ciclo, pero no eligió mas que un nodo descendiente de 2 (el nodo 5), lo que no les sirvió para la división concurrente del trabajo entre sus miembros. De haber seleccionado además el nodo 3, pudo haberse dado una división de trabajo entre sus miembros. Sin embargo, en el segundo ciclo, tuvieron la posibilidad de desarrollar simultáneamente dos trayectorias de la gráfica, partiendo de los nodos 2 y 5, trabajando 3, 4 y 13 en paralelo con 6, 7, 8 y 14. Por su parte, el grupo B incluyó los nodos 2, 3 y 5 desde el primer ciclo. Nótese que 3 y 5 son nodos descendientes de 2, por lo que potencialmente el grupo pudo desarrollar ambas trayectorias (3 y 4 en paralelo con 5, 6 y 7) . Más aún, para el segundo ciclo eligió los nodos 8 y 9, que son nodos descendientes de 7. Esto nuevamente le permitió al grupo distribuir mejor el trabajo entre sus miembros. Por otro lado, la propuesta del grupo C de nuevo presenta el problema de no haber elegido al nodo 2 dentro de la funcionalidad mínima. Sin embargo, el grupo elige para el primer ciclo los nodos 3 y 5, lo que le da la posibilidad de trabajar las trayectorias de los nodos 3 y 4 en forma simultánea con 5 y 8. Ya en el segundo ciclo, el grupo agregó los nodos 7 y 9, pero esto no le permitió hacer una considerable división de trabajo entre sus miembros. Como conclusión, el criterio de trabajo independiente entre los miembros de los grupos se logra hasta el segundo ciclo en el caso del grupo A, durante ambos ciclos para los grupos B, y mayoritariamente en el primer ciclo para el grupo C.

El uso de las Gráficas de Dependencia es una técnica que permite revisar y evaluar el cumplimiento de los Criterios de la fase de Estrategia. Para el ejemplo del Contador de Cambios, la gráfica permite establecer:

- *(a)* un conjunto de necesidades básicas que al satisfacerse generan un producto con funcionalidad mínima,
- *(b)* sugiere un orden para la extensión del producto aprovechando las trayectorias dentro de la gráfica, y
- *(c)* sugiere las posibilidades de división de trabajo entre los miembros de un equipo.

4. Conclusiones y Trabajo Futuro.

Durante la fase de Estrategia, es necesario que el estudiante entienda lo más claramente posible la Declaración de Necesidades, y particularmente las dependencias entre necesidades. Solo si la declaración de necesidades es entendida en forma completa (toda necesidad y sus dependencias son consideradas) y consistente (no existen contradicciones observables respecto a las dependencias) es posible continuar el desarrollo de una solución satisfactoria. Si la declaración de necesidades no se entiende claramente, esto se hará aparente una vez que la Especificación de Requerimientos se desarrolle durante la fase de Requerimientos [2]. Será necesario entonces revisar de nuevo la declaración del problema, a fin de remover todas las inconsistencias y asegurar que todas las necesidades están cubiertas.

La fase de Estrategia es un paso difícil de TSPi, ya que con poca información es necesario tomar decisiones que pueden tener un fuerte impacto en el proyecto. Durante el curso estudiantes e instructores deben analizar con detenimiento las necesidades para tener una interpretación común y acordada. Esto es difícil debido a que se trata de una interpretación de lenguaje natural. De esta interpretación depende el éxito o fracaso de la fase de Estrategia, y en consecuencia, del resultado final del proyecto.

Con el objetivo de sistematizar la forma de analizar la Declaración de Necesidades, este artículo propone una técnica de análisis basada en las Gráficas de Dependencia entre Necesidades, a fin de facilitar el desarrollo de la fase de Estrategia durante un curso de TSPi. Una Gráfica de Dependencias entre Necesidades es una gráfica dirigida con dos tipos de vértices, que representan las relaciones de dependencia total o parcial entre las necesidades funcionales de un problema dado. Tales gráficas se construyen para describir las relaciones de dependencia entre necesidades, lo que pretende servir como apoyo para el análisis y el cumplimiento de los Criterios de la fase de Estrategia.

En este artículo, se presenta la construcción de la gráfica para el ejemplo del Contador de Cambios, y se utiliza para el análisis y evaluación de las propuestas de selección de necesidades realizadas por grupos de estudiantes durante la fase de Estrategia de un curso basado en TSPi. En el caso presentado aquí, la gráfica fue sugerida y construida posteriormente de analizar el trabajo de los estudiantes. Sin embargo, se considera que esta técnica de análisis de necesidades puede enseñarse para ser utilizada por los estudiantes, a fin de facilitar el análisis de necesidades y definición de la estrategia.

Para el ejemplo del Contador de Cambios, la gráfica permitió establecer la funcionalidad mínima del producto para el primer ciclo, sugirió un orden para la extensión de tal producto en ciclos subsecuentes, y la posibilidad de diferentes divisiones para el trabajo independiente de los miembros de los grupos. Esto permitió evaluar las propuestas de selección de necesidades de los estudiantes con respecto a los tres primeros Criterios de la fase de Estrategia.

Sin embargo, aún se requiere realizar otros experimentos en los que los estudiantes efectivamente hagan uso de las Gráficas de Dependencia para analizar la Declaración de Necesidades y definir la estrategia a seguir. También, se hace necesario comprobar la facilidad de la construcción de las gráficas para otros ejemplos educativos. De hecho, el objetivo a mediano plazo es comprobar que estas gráficas pueden tener una aplicación más general para desarrollos comerciales.

Referencias.

[1] M. Dick, M. Postema and J. Miller, "Process Issues of PSP and TSPi for Undergraduate Students". *Workshop on Teaching PSP and TSPi in Universities*. The 14th Conference on Software Engineering Education and Training (CSEET, 2001). Charlotte, NC, USA. February 19–21, 2001.

[2] Humphey, W. *Introduction to the Team Software Process.* The SEI Series in Software Engineering, Addison–Wesley Longman, Inc., Reading, Massachusetts, 2000.

[3] G.W. Hislop, "Teaching the PSP and TSPi in a University". *Workshop on Teaching PSP and TSPi in Universities*. The 14th Conference on Software Engineering Education and Training (CSEET, 2001). Charlotte, NC, USA. February 19–21, 2001.

[4] A. Lattanze, M. Rosso–Llopart and J. Tomayko, "Teaching the Personal Software Process and the Team Software Process in Carnegie Mellon's Master of Software Engineering Program". *Workshop on Teaching PSP and TSPi in Universities*. The 14th Conference on Software Engineering Education and Training (CSEET, 2001). Charlotte, NC, USA. February 19–21, 2001.

[5] S.K. Lisack, "The Personal Software Process: Where Does it Belong in an Undergraduate Information Systems Curriculum?" *Workshop on Teaching PSP and TSPi in Universities*. The 14th Conference on Software Engineering Education and Training (CSEET, 2001). Charlotte, NC, USA. February 19–21, 2001.

[6] K.B. Olson, "Position Statement on Teaching of PSP and TSPi". *Workshop on Teaching PSP and TSPi in Universities*. The 14th Conference on Software Engineering Education and Training (CSEET, 2001). Charlotte, NC, USA. February 19–21, 2001.

[7] L. Williams, "An Alternative: The Collaborative Software Process (CSP)". *Workshop on Teaching PSP and TSPi in Universities*. The 14th Conference on Software Engineering Education and Training (CSEET, 2001). Charlotte, NC, USA. February 19–21, 2001.