UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

Colegio de Ciencias e Ingenierías

Evaluación de la Resistencia a la Fractura en Recubrimientos de WC-12%Co por Termorociado (HVOF) en Pruebas de Tracción mediante Análisis de Elementos Finitos

Proyecto de investigación

Diego Santiago Morales Arcos

Ingeniería Mecánica

Trabajo de titulación presentado como requisito
para la obtención del título de
Ingeniero Mecánico

Quito, 21 de mayo de 2018

UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

COLEGIO DE CIENCIAS E INGENIERÍA

HOJA DE CALIFICACIÓN
DE TRABAJO DE TITULACIÓN

Evaluación de la Resistencia a la Fractura en Recubrimientos de WC-12%Co por Termorociado (HVOF) en Pruebas de Tracción mediante Análisis de Elementos Finitos

Diego Santiago Morales Arcos

Calificación:

Nombre del profesor, Título académico Alfredo Valarezo PhD.

Firma del profesor: _____

Quito, 21 de mayo de 2018

Derechos de Autor

Por medio del presente documento certifico que he leído todas las Políticas y Manuales de la Universidad San Francisco de Quito USFQ, incluyendo la Política de Propiedad Intelectual USFQ, y estoy de acuerdo con su contenido, por lo que los derechos de propiedad intelectual del presente trabajo quedan sujetos a lo dispuesto en esas Políticas.

Asimismo, autorizo a la USFQ para que realice la digitalización y publicación de este trabajo en el repositorio virtual, de conformidad a lo dispuesto en el Art. 144 de la Ley Orgánica de Educación Superior.

Firma del estudiante: _____

Nombres y apellidos: Diego Santiago Morales Arcos

Código: 00116598

Cédula de Identidad: 2100614392

Lugar y fecha: Quito, 21 de mayo de 2018

Dedicatoria

A mi madre Juanita y a mi padre Henry.

Por haberme dado la vida, por quererme demasiado, por haberme dado todo ese cariño indispensable, por sus consejos, por su preocupación diaria, por la motivación constante a lo largo de mi carrea y porque gracias a ellos he crecido y se me ha permitido ser la persona que soy ahora. Por ser mis ejemplos a seguir, por creer en mí, por su perseverancia y constancia, por haberme dado todo lo necesario y más. Este logro es gracias a ellos.

A mi hermana Kimberly.

Por ser mi confidente, mi acompañante siempre, mi compañera de travesuras y por ser la persona que más quiero en este mundo.

A mis mejores amigos Danny, Miguel, Raul, DanielaP y DanielaL por compartir conmigo los buenos y malos momentos.

RESUMEN

El presente estudio se enfoca en el cálculo computacional de la resistencia a la fractura (K_{IC}) de un recubrimiento de WC-12%Co fabricado por el proceso de HVOF, en un ensayo a tracción, mediante un análisis de elementos finitos (FEA). Empleando el software ABAQUS e "input files" generados por medio de un código escrito en Java, se procede a realizar un modelo de probetas de tipo sanduche con dimensiones: 64 [mm] de largo, 12.5 [mm] de ancho y 3.2 [mm] de espesor de sustrato, variando el espaciado entre iniciadores de fractura en el recubrimiento en un rango de 360 a 650 [µm] entre iniciador y con un espesor de recubrimiento de 200 [μm]. El sustrato se modela con propiedades elasto-plásticas, y al recubrimiento como lineal-puramente elástico. Los resultados obtenidos muestran que la resistencia a la fractura para modo I experimental, se encuentra en un rango de 11.17-19.33 MPa $m^{0.5}$, dependiendo del tamaño del iniciador: 1-3 [μm]. El modelo utiliza como datos los resultados experimentales de esfuerzo-deformación, obtenidos de ensayos a tracción en probetas similares a las usadas en el modelo, para obtener K_{IC}.

Palabras clave: Recubrimiento cermets, Resistencia a la fractura, HVOF, Análisis de elementos finitos (FEA), lenguaje de programación Java, ABAQUS.

ABSTRACT

This research is focused on the experimental calculation of the fracture toughness (K_{IC}) of a WC-12%Co coating made by HVOF processing using finite element analysis(FEA). This research uses the FE software ABAQUS and input files generated by a code written in Java. The model considers a sandwich type specimen with dimensions of: 64 [mm] in length, 12.5 [mm] in width and 3.2 [mm] in thickness of the base material. The model varies the spacing between fracture initiators in the coating in a range of 360 to 650 [μm] between initiators, with a coating thickness of 200 [μm]. The base material is modeled as an elastic-plastic material, whereas the coating is linear elastic. The results showed that the fracture toughness is in a range of 11.17-19.33 MPa $m^{0.5}$, depending upon de initiator size: 1-3 [μm]. The model uses experimental load-displacement results as input from specimens loaded under uniaxial tension to calculate K_{IC}.

Key words: Cermets, Fracture Toughness, HVOF, Finite element analysis (FEA), Java programming language, ABAQUS.

SIMBOLOGÍA

E	Modulo de Young	GPa
K_{IC}	Resistencia a la fractura (modo I)	MPa $m^{0.5}$
K_{IIC}	Resistencia a la fractura (modo II)	MPa $m^{0.5}$
G_{IC}	Tasa de liberación de energía en tensión (modo I)	MPa m
G_{IIC}	Tasa de liberación de energía en tensión (modo II)	MPa m

TABLA DE CONTENIDO

Introducción **12**

Materiales Y Métodos **15**
1. Modelo: Probeta Larga 22
2. Modelo: Probeta Corta 28
3. Modelo: Análisis de espaciamiento de grietas 29

Resultados **31**
1. Modelo: Probeta Larga 31
2. Modelo: Probeta Corta 47
3. Modelo: Análisis de espaciamiento de grietas 50

Discusiones Y Recomendaciones **52**

Conclusiones **54**

Referencias bibliográficas **55**

Anexo A: Instalación de JDK (Java Development Kit) para uso de los códigos adjuntos **57**

Anexo B: Instalación IDE IntelliJ IDEA para ejecución de códigos Java **58**

Anexo C: Revisión del path de ABAQUS en Windows **59**

Anexo D: Explicación de las propiedades y dimensiones descritos en los códigos Java para los modelos **60**

Anexo E: Código Java para modelos de fractura desde la interface **65**

Anexo F: Código Java para modelos de la primera fractura **77**

Anexo G: Código Java para modelos de la segunda fractura **88**

Anexo H: Código Java para modelos de análisis del espaciado **101**

ÍNDICE DE TABLAS

Tabla 1. Propiedades mecánicas de la probeta.....15
Tabla 2. Datos del espaciado de iniciadores de fractura vs deformación en un recubrimiento WC-12%Co de 200 µm.....17
Tabla 3. Análisis para escoger el tamaño del elemento en la dirección vertical 26
Tabla 4. Datos del espaciado ajustado de iniciadores de fractura.....30
Tabla 5. Resistencia a la fractura para modelos con tamaño de elemento en la dirección horizontal de 30 [μm].....32
Tabla 6. Datos del tamaño del iniciador con su respectiva resistencia a la fractura para modelos con tamaño de elemento en la dirección horizontal de 30 [μm].....32
Tabla 7. Esfuerzos y deformaciones para modelos de tamaño de elemento de 30 [μm].....34
Tabla 8. Resistencia a la fractura para modelos con tamaño de elemento en el eje x de 100 [μm].....37
Tabla 9. Datos del tamaño del iniciador con su respectiva resistencia a la fractura para modelos con tamaño de elemento en la dirección horizontal de 100 [μm].....37
Tabla 10. Esfuerzos y deformaciones para modelos de tamaño de elemento de 100 [μm].....38
Tabla 11. Resultados para esfuerzos y deformaciones en el análisis de calibración en el instante antes de la fractura 41
Tabla 12. Esfuerzos críticos en zonas afectadas tras la fractura 44
Tabla 13. Resultados de la deformación a la fractura acorde al tamaño del espesor del recubrimiento 46
Tabla 14. Esfuerzos y deformaciones para modelos en la primera fractura de tamaño de elemento de 30 [μm] variando el tamano del espesor.....47
Tabla 15. Datos de la deformación para la segunda fractura para modelos con tamaño de elemento en la dirección horizontal de 30 [μm].....48
Tabla 16. Esfuerzos y deformaciones para modelos en la segunda fractura de tamaño de elemento de 30 [μm] 48
Tabla 17. Resultados del espaciado con su respectiva fractura 50
Tabla 18. Definición de las variables de entrada del programa en Java. 60

ÍNDICE DE FIGURAS

Figura 1. Huella de la indentación de Vickers 14
Figura 2. Problema experimental 15
Figura 3. Fisuras en el recubrimiento 16
Figura 4. Gráfica experimental que muestra el espaciado de iniciadores de fractura vs deformación en un recubrimiento WC-12%Co de 200 µm. 17
Figura 5. Criterio VCCT 18
Figura 6. Micrografía de una fractura en recubrimientos en un ensayo de tracción 19
Figura 7. Modelo de ruptura desde la interface de los dos materiales 19
Figura 8. Modelo de ruptura desde la interface de los dos materiales a un 25% de deformación 20
Figura 9. Modelo global de ruptura desde la superficie externa del recubrimiento para la primera fractura 20
Figura 10. Modelo de ruptura desde la superficie externa del recubrimiento para la primera fractura 21
Figura 11. Modelos de delaminación tras la primera ruptura desde la superficie externa del recubrimiento 21
Figura 12. Modelo de ruptura desde la superficie externa del recubrimiento para la segunda fractura 22
Figura 13. Esquema de la probeta para la primera fractura. 23
Figura 14. Resultado del depósito de partículas por HVOF 23
Figura 15. Efecto de borde en probeta 24
Figura 16. Tamaño del elemento en el eje x 24
Figura 17. Grafica del análisis del tamaño de elemento en la dirección horizontal para una probeta de 0.03 [m] de largo, iniciador de 1 [μm] y K_{IC} de 6.27 [MPa $m^{0.5}$] 25
Figura 18. Diferencia de la distribución de esfuerzos en un mallado fino y en un mallado grueso tras la ruptura del recubrimiento 25
Figura 19. Tamaño del elemento en la dirección vertical 26
Figura 20. Modelo 1 grieta vs modelo 7 grietas 27
Figura 21. Distribución de esfuerzos tras la ruptura 27
Figura 22. Esquema de la probeta para la segunda fractura. 28
Figura 23. Modelo de ruptura para análisis de espaciado entre fracturas 29
Figura 24. Deformación a la fractura vs la resistencia a la fractura para un tamaño de elemento de 30 [μm], con iniciadores de varios tamaños 31
Figura 25. Instante antes de la fractura 33
Figura 26. Instante después de la fractura 33
Figura 27. Deformaciones elásticas para conjunto recubrimiento-sustrato 33
Figura 28. Deformaciones elásticas para sustrato 34
Figura 29. Deformaciones elásticas para recubrimiento 34
Figura 30. Esfuerzos en el recubrimiento antes y después de la fractura para elementos de 30 [μm] 35
Figura 31. Deformaciones elásticas en el recubrimiento antes y después de la fractura para elementos de 30 [µm] 35
Figura 32. Esfuerzos en el sustrato antes y después de la fractura para elementos de 30 [µm] 35
Figura 33. Deformaciones elásticas en el sustrato antes y después de la fractura para elementos de 30 [µm] 35

Figura 34. Deformaciones plásticas en el sustrato antes y después de la fractura para elementos de 30 [μm] 36
Figura 35. Grafica de la deformación a la fractura vs resistencia a la fractura para un tamaño de elemento de 100 [μm] 36
Figura 36. Esfuerzos en el recubrimiento antes y después de la fractura para elementos de 100 [μm] 38
Figura 37. Deformaciones elásticas en el recubrimiento antes y después de la fractura para elementos de 100 [μm] 39
Figura 38. Esfuerzos en el sustrato antes y después de la fractura para elementos de 100 [μm] 39
Figura 39. Deformaciones elásticas en el sustrato antes y después de la fractura para elementos de 100 [μm] 39
Figura 40. Deformaciones plásticas en el sustrato antes y después de la fractura para elementos de 100 [μm] 40
Figura 41. Modelo con 1 grieta superior e inferior 40
Figura 42. Modelo con 3 grietas superiores e inferiores 40
Figura 43. Modelo con 5 grietas superiores e inferiores 41
Figura 44. Modelo con 7 grietas superiores e inferiores 41
Figura 45. Zonas críticas sustrato 42
Figura 46. Zonas críticas recubrimiento 42
Figura 47. Zona para una nueva fractura 42
Figura 48. Zonas críticas para una posible tercera fractura 43
Figura 49. Ruptura al 0.6% en la primera fractura 43
Figura 50. Ruptura al 0.675% en la segunda fractura 43
Figura 51. Ruptura al 0.775% en la tercera fractura 43
Figura 52. Esfuerzos críticos dentro de la zona de fractura 44
Figura 53. Iniciador de 1 [μm] y K_{IC} de 3 [MPa $m^{0.5}$] para fractura en zonas cercanas 45
Figura 54. Tamaño del espesor del recubrimiento de 50 [μm] y 400 [μm] 45
Figura 55. Grafica del tamaño del espesor del recubrimiento vs la deformación a la fractura 46
Figura 56. Dimensiones probeta para la segunda fractura. 47
Figura 57. Instante antes de la fractura 49
Figura 58. Instante después de la fractura 49
Figura 59. Modelos de ruptura para análisis de espaciado de fractura antes y después de la fractura 50
Figura 60. Espaciado entre grietas vs la deformación a la fractura 51
Figura 61. Verificación de la instalación del JDK 57
Figura 62. Interfaz IntelliJ IDEA 58
Figura 63. Verificación de Abaqus Command 59
Figura 64. Definición de dimensiones para el modelo. 60
Figura 65. Líneas de código que contienen el path donde se guardaran los archivos de los análisis 61
Figura 66. Definición de módulos de Young, coeficientes de Poisson para el modelo y propiedades plásticas 62
Figura 67. Archivos creados por los códigos de Java. 62
Figura 68. Líneas de código para generar superficies de fractura por medio del nombre del elemento 64
Figura 69. Nombre del elemento que se debe escribir en el código para generar la fractura 64

INTRODUCCIÓN

El desgaste, la corrosión y la fatiga pueden llevar a que piezas y componentes de una máquina pierdan su funcionalidad y su resistencia con el tiempo. Esta posibilidad de falla, implica la necesidad de utilizar técnicas de recuperación del material como la del termorociado.

El uso de la técnica de depósito de material por termorociado comenzó en 1909 cuando el Dr. Max Ulrick Schoop observó que partículas fundidas proyectadas sobre sí mismas creaban capas una tras otra (Pawlowski, 1995). Junto con la ayuda de sus colaboradores iniciaron el proceso de proyección térmica o también llamado termorociado. Este proceso ha incursionado en la industria mundial en la cual se mueve más de dos mil millones de dólares por año (Papyrin, Kosarev, Klinkov, Alkhimov, & Fomin, 2007). El termorociado es un procedimiento de manufactura aditiva de bajo costo para crear recubrimientos superficiales funcionales. El termorociado consiste en depositar partículas de material a altas velocidades para recuperar el sustrato, logrando así llevar el componente a sus dimensiones originales o proteger al material contra corrosión o desgaste. Este procedimiento logra recuperar herramientas desgastadas o piezas vitales de maquinaria logrando propiedades que superan los estándares originales de la pieza original. (Jeandin, Boller, & Pauchet, 2013)

Dado que este proceso implica el depósito de un nuevo material sobre otro material diferente, es recomendable tener un modelo que simule la interacción de varios materiales (ej. Recubrimiento-substrato) y que sirva como una base para ampliar la comprensión de estos sistemas, además de funcionar como un posible predictor de fallas.

El propósito del proyecto presentado en este documento es realizar un modelo para determinar la resistencia a la fractura K_{IC} de un recubrimiento de WC-12%Co por medio de

un análisis de elementos finitos (FEA) comparando los resultados con datos experimentales para una vez calculada la resistencia a la fractura, poder usar el modelo para predecir el esfuerzo y la deformación que se necesitan para fracturar el recubrimiento.

La necesidad de tener un modelo valido para predecir fractura WC-12%Co, nace en las características que tiene este material ante el desgaste lo que lo hace un material adecuado en recubrimientos de piezas que necesitan recuperación, y alta resistencia al desgaste por fricción. En este tipo de aplicación, conocer la resistencia a la fractura del recubrimiento es indispensable para mejorar el diseño de productos, procesos de fabricación y procesos de recuperación.

Entre los métodos de medición de la resistencia a la fractura, el método experimental que más se utiliza es el método por indentación Vickers. Este procedimiento consiste en medir las longitudes de grietas mostradas en la Figura 1 tras una prueba de dureza. Para el cálculo de la resistencia a la fractura se han desarrollado un sinnúmero de ecuaciones en las cuales se requiere el valor del módulo de Young y Poisson, además de los resultados de la prueba de dureza. Las ecuaciones se dividen en dos grupos: empíricas y experimentales. La ecuación empírica más utilizada es la propuesta por Evans (1), y la ecuación experimental más utilizada es la propuesta por Niihara (2), las ecuaciones se las describe a continuación. (Rocha & Díaz, 2008, p. 54)

$$K_{IC} = 0.16\left(\frac{c}{a}\right)^{-1.5}(Ha^{1/2}) \qquad (1)$$

$$K_{IC} = 0.0298H\sqrt{a}\left(\frac{E}{H}\right)^{\frac{1}{2}}\left(\frac{c}{a}\right)^{-1.38} \qquad (2)$$

Donde:

$H = \frac{1.8P}{a^2}$

K_{IC} = Resistencia a la fractura (MPa $m^{0.5}$)

H = Dureza Vickers (MPa)

E = Modulo de Young (MPa)

P = Carga de prueba en durómetro Vickers (MPa)

c = Longitud media de las grietas obtenidas en las puntas de la huella Vickers (µm)

a = Longitud media de la diagonal de la huella Vickers (µm)

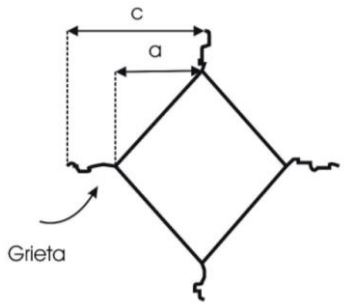


Figura 1. Huella de la indentación de Vickers

Los valores de la resistencia a la fractura en la publicación de Usmani para el WC-12%Co por HVOF, por medio de este método oscilan entre 13 y 15.2 [MPa $m^{0.5}$] y varían acorde al tamaño de las partículas depositadas. (Usmani & Sampath, 1996)

Otros métodos que pueden ser utilizados incluyen: ensayos de doblado en 3 puntos, o 4 puntos, ensayos de torsión con grietas pre-fabricadas, etc. La característica de fragilidad del recubrimiento de WC-12%CO limita la fabricación de este tipo de probetas, y por tanto la medición de K_{IC} ha estado limitada al uso de métodos de indentación. El uso de ensayos de tracción viene a constituir en una alternativa para medir esta propiedad, con el uso complementario del modelado computacional.

En este proyecto, se espera realizar un modelo computacional capaz de reproducir resultados experimentales semejantes a los que presenta en la publicación: Mechanical Behavior of Spray-Coated Metallic Laminates (Vackel, Nakamura & Sampath, 2015) y que a su vez pueda realizar el cálculo de la resistencia a la fractura del recubrimiento. La propiedad de resistencia a la fractura K_{IC} se alimenta al modelo para reproducir el ensayo a tracción de probetas recubiertas.

MATERIALES Y MÉTODOS

Se requiere modelar probetas de acero dúctil recubiertas de Carburo de Tungsteno en matriz de cobalto (WC-12%Co); el modelo de la probeta se lo muestra en la Figura 2, en estas probetas se procederá a introducir propiedades mecánicas, iniciadores de fractura, y el coeficiente de resistencia a la fractura del recubrimiento. Luego se procederá a hacer la simulación por medio de elementos finitos de un ensayo a tracción.

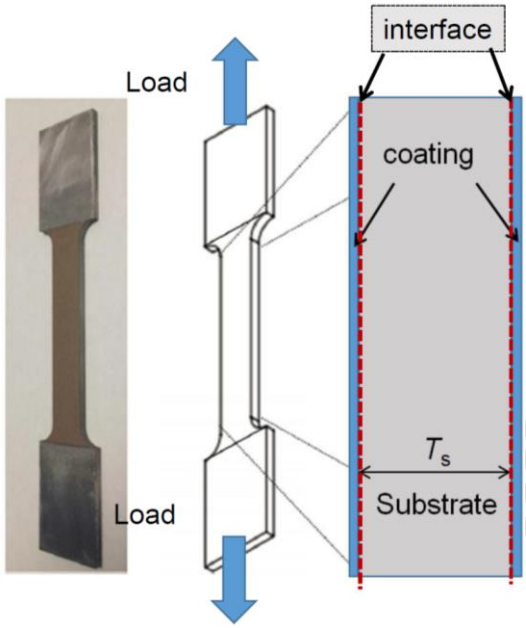


Figura 2. Problema experimental

Las propiedades mecánicas a usar se detallan en la Tabla 1 que se muestra a continuación.

Tabla 1. Propiedades mecánicas de la probeta

	WC-12%Co	Acero 1018
Módulo de Young [GPa]	560	207
Coeficiente de Poisson	0.24	0.3
Esfuerzo de fluencia [MPa]	-	370
Esfuerzo último [MPa]	-	440
Resistencia transversal[MPa]	3100	-

En este documento, los modelos de elementos finitos son ejecutados y resueltos por medio del software de elementos finitos ABAQUS, para esto se generan modelos de ABAQUS mediante “input files” o archivos “inp”; estos modelos son leídos por ABAQUS sin necesidad de utilizar el GUI o interfaz gráfica. Los archivos “inp” pueden ser escritos por cualquier editor de texto y a su vez pueden ser modificados por lenguajes de programación de escritura de archivos como Python, C++ y Java. En este documento todos los modelos utilizados, son generados por medio de “input files” generados por medio de códigos escritos en lenguaje Java y compilados desde Abaqus Command.

Los modelos generados buscan simular lo que ocurre en la experimentación, al momento de realizar el ensayo a tracción de la probeta aparecen fisuras transversales a la dirección de la fuerza a lo largo del recubrimiento como muestra la Figura 3.

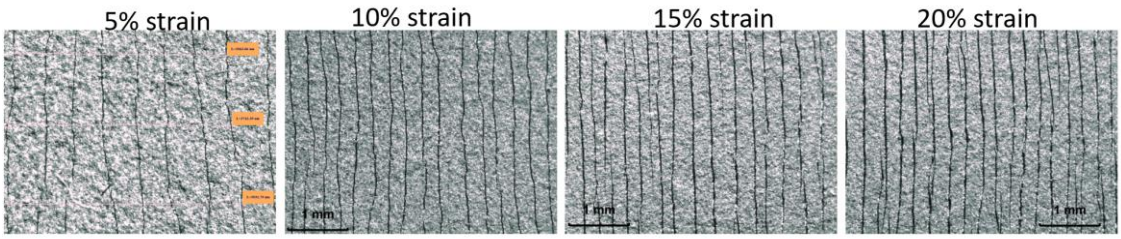


Figura 3. Fisuras en el recubrimiento

De la misma manera, los modelos intentan recrear los datos experimentales presentados en la Figura 4. En esta Figura se muestra tanto el espaciado entre fractura como la deformación a la que se presentan estas fracturas transversales.

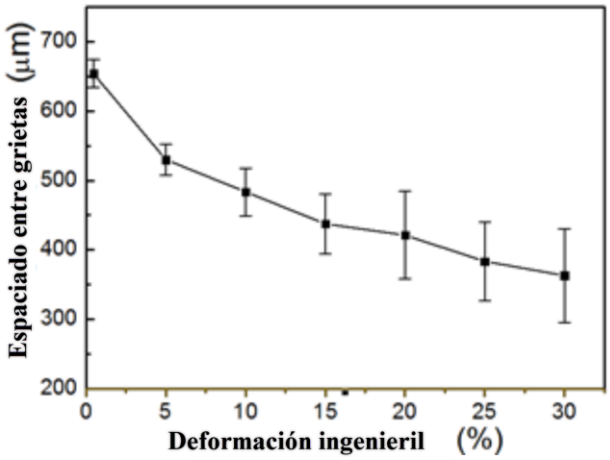


Figura 4. Gráfica experimental que muestra el espaciado de iniciadores de fractura vs deformación en un recubrimiento WC-12%Co de 200 µm.

Los datos experimentales fueron obtenidos del Centro de Investigación de Thermal Spray de la Universidad de Stony Brook en los cuales se realizó el ensayo a tracción de una probeta de acero dúctil recubierta por un recubrimiento WC-12%Co de 200 [μm] de espesor. (The Thermal Spray Laboratory Stony Brook, 2017).

La Tabla 2 muestra los datos de espaciado entre fracturas que se trata de reproducir en el modelo, junto con la deformación necesaria para realizar la fractura a ese punto.

Tabla 2. Datos del espaciado de iniciadores de fractura vs deformación en un recubrimiento WC-12%Co de 200 µm.

Deformación [%]	Espacio entre fracturas [µm]
0.6	650
4.8	540
10	490
15	440
20	425
25	380
30	360

Con respecto a la simulación y los modos de fractura, el modelo intenta simular fractura y sufre deformación en una sola dirección, para esto se necesita del valor de G_{IC} que corresponde al valor de la Tasa de liberación de energía en tensión (Strain energy release

rate) para esto se calcula este valor a partir del K_{IC} con la ecuación de Griffith (1920, pp 163-198):

$$G_{IC} = \frac{K_{IC}^2}{E'} \qquad (4)$$

Donde E' es el módulo de Young para esfuerzo plano.

El modelo a su vez necesita de los coeficientes de G_{IIC} y G_{IIIC} necesarios para que los modelos converjan, estos coeficientes se calculan a partir del K_{IIC} y K_{IIIC} con la ecuación de Griffith respectivamente.

Para la simulación en el software ABAQUS, se utiliza el criterio de Técnica de Cierre Virtual de Grietas o The Virtual Crack Closure Technique (VCCT), criterio que utiliza los principios de la mecánica elástica lineal (LEFM), y es apropiado para la propagación de grietas frágiles a lo largo de superficies predefinidas. Este criterio a su vez se basa en la suposición de que la energía para cerrar una grieta es la misma energía necesaria para realizar la ruptura de la misma. Como se observa en la Figura 5, este criterio une dos o más nodos del modelo de elementos finitos por medio del coeficiente de G_{IC} que representa a la energía necesaria para abrir nuevamente estos dos nodos.

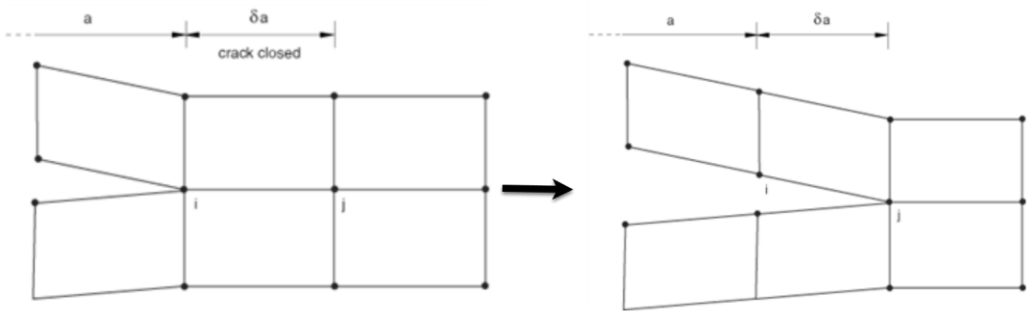


Figura 5. Criterio VCCT

En la experimentación, las fracturas empiezan a aparecer transversalmente a la dirección de la fuerza aplicada en el recubrimiento, una vez que aparece la grieta y se ruptura el recubrimiento, empieza la delaminación del recubrimiento antes de empezar la fractura en el sustrato. Esto se debe a que el recubrimiento no está perfectamente adherido al sustrato y

este último se comporta como un material dúctil; todo esto permite la deformación plástica antes de que ocurra una fractura frágil

La fractura del recubrimiento puede iniciar desde la interface de los dos materiales (recubrimiento-sustrato) o desde la superficie exterior del recubrimiento. En el presenta trabajo se intenta recrear estos dos tipos de fractura. En la Figura 6, se puede apreciar gráficamente los dos tipos fractura.

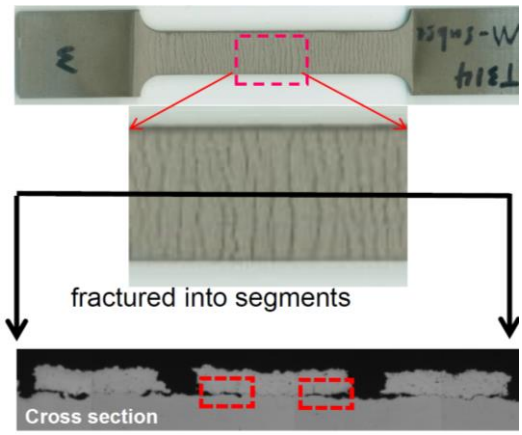
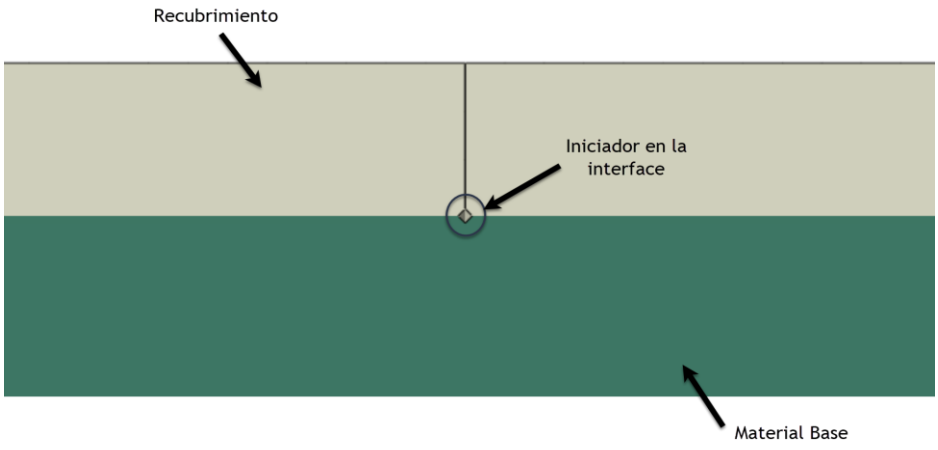


Figura 6. Micrografía de una fractura en recubrimientos en un ensayo de tracción

Para el primer modelo, la ruptura inicia desde la interface recubrimiento – sustrato, para esto se establece un iniciador o grieta en la interface de los dos materiales. Este último junto con las propiedades para fractura en el recubrimiento se especifican en un código Java que se encarga de la creación de los inputs files y se adjunta en el Anexo E.



.

Figura 7. Modelo de ruptura desde la interface de los dos materiales

Cuando el iniciador es colocado cerca de la interfaz recubrimiento – sustrato, no se obtiene ruptura del recubrimiento, aunque se utilicen grietas grandes, valores de resistencia a la fractura pequeños y grandes deformaciones (de hasta 25%). Como se muestra en la Figura 8, para este modelo con un iniciador de 10 [μm] y K_{IC} de 3 [MPa $m^{0.5}$] los esfuerzos en el sustrato llegan a aproximadamente 270 [MPa] y en el recubrimiento a 25 [GPa] sin ocasionar fractura;

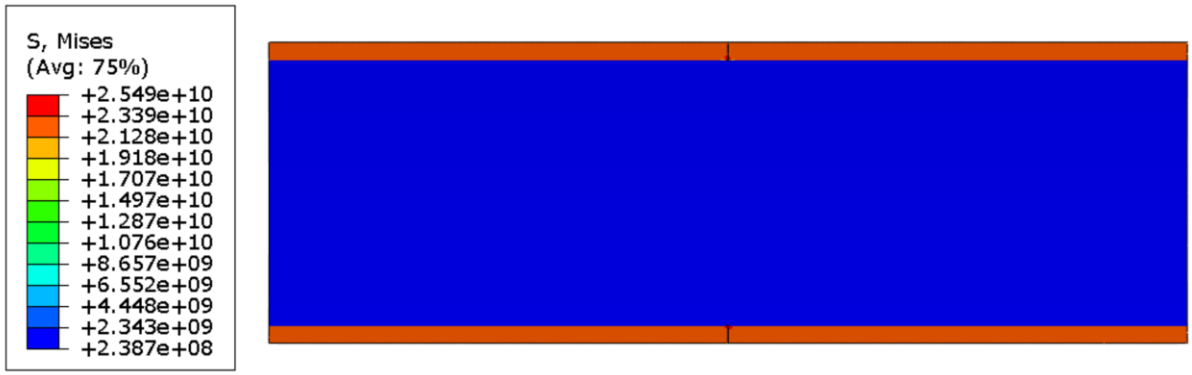


Figura 8. Modelo de ruptura desde la interface de los dos materiales a un 25% de deformación

Un segundo modelo implica la ruptura desde la superficie del recubrimiento; para esto se modelan probetas con las características de la mostrada en la Figura 9, el código en Java utilizado para crear los modelos es anexados en el Anexo F.

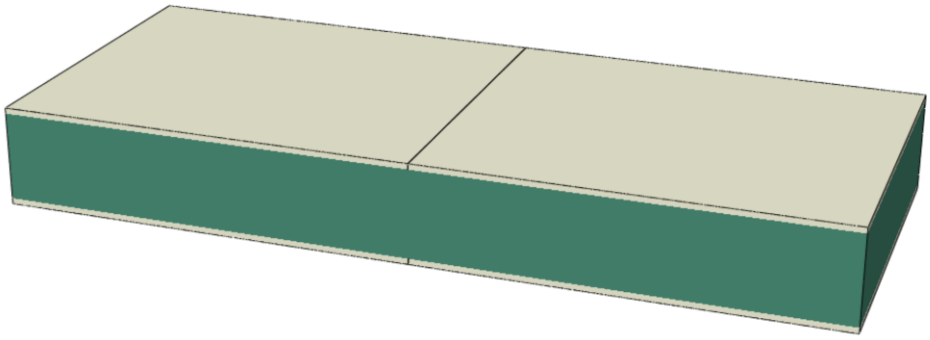


Figura 9. Modelo global de ruptura desde la superficie externa del recubrimiento para la primera fractura

Este segundo modelo se consigue colocando un iniciador en la superficie exterior del recubrimiento, Figura 10.

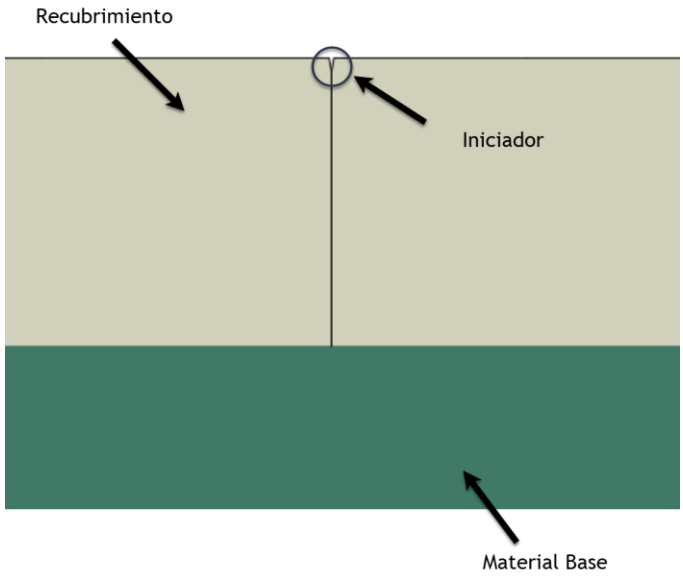


Figura 10. Modelo de ruptura desde la superficie externa del recubrimiento para la primera fractura

El recubrimiento empieza a fracturarse a una deformación global específica, la ruptura se produce en una dirección perpendicular al sustrato y una vez que alcanza la base del recubrimiento, empieza a aparecer la delaminación de este (Figura 6). Para poder simular este comportamiento se ingresan propiedades de fractura en los nodos en la interfaz de los materiales.

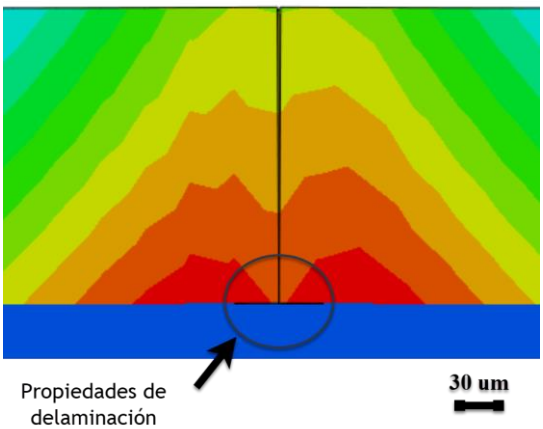


Figura 11. Modelos de delaminación tras la primera ruptura desde la superficie externa del recubrimiento

El modelo anterior no presenta convergencia ni resultados al querer simular una segunda fractura una vez ocurrida una fractura inicial junto con delaminación. Para superar este inconveniente, se introduce un modelo con sustrato (sustrato) sin recubrimiento en las zonas laterales de la probeta tal y como se muestra en la Figura 12. El código Java para este modelo se presenta en el Anexo G.

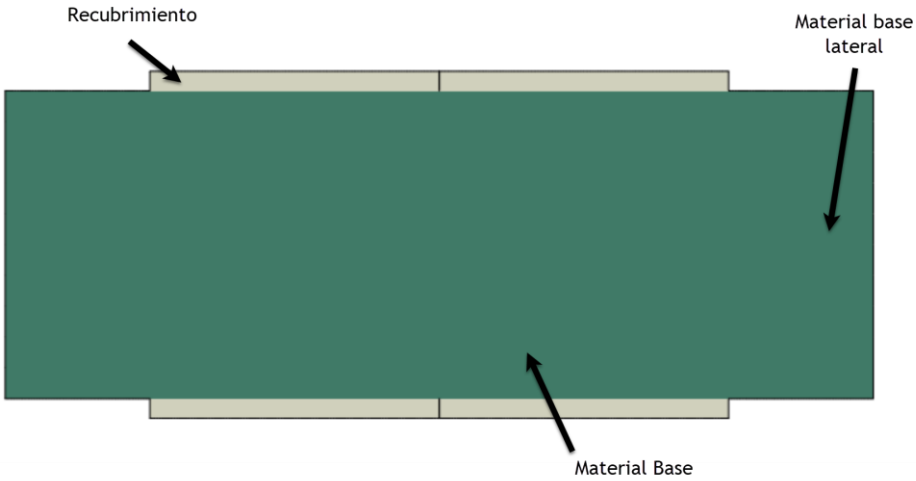


Figura 12. Modelo de ruptura desde la superficie externa del recubrimiento para la segunda fractura

Una vez especificados los modelos que tienen convergencia y los modelos que se pueden utilizar, se proceden a hacer análisis más a fondo de las dimensiones y parámetros a utilizar. En este caso los modelos que cumplen con estas condiciones son: el que tienen un iniciador en la superficie y el que presenta sustrato sin recubrimiento en los laterales de la probeta.

1. Modelo: Probeta Larga

Para este modelo, los parámetros fijos son: el espesor del sustrato (3.2 mm), el tamaño del ancho de la probeta (12.5 mm) y el espesor del recubrimiento (200 μm); por otro lado los parametros que se pueden modificar son: el tamaño del largo de la probeta (este valor puede cambiar para evitar el tiempo de análisis), el tamaño del iniciador de fractura (desde 1 μm hasta

10 μm), el tipo de mallado y el coeficiente de la resistencia a la fractura (desde 3 MPa $m^{0.5}$ hasta 20 MPa $m^{0.5}$).

El modelo de la probeta se muestra en la Figura 13.

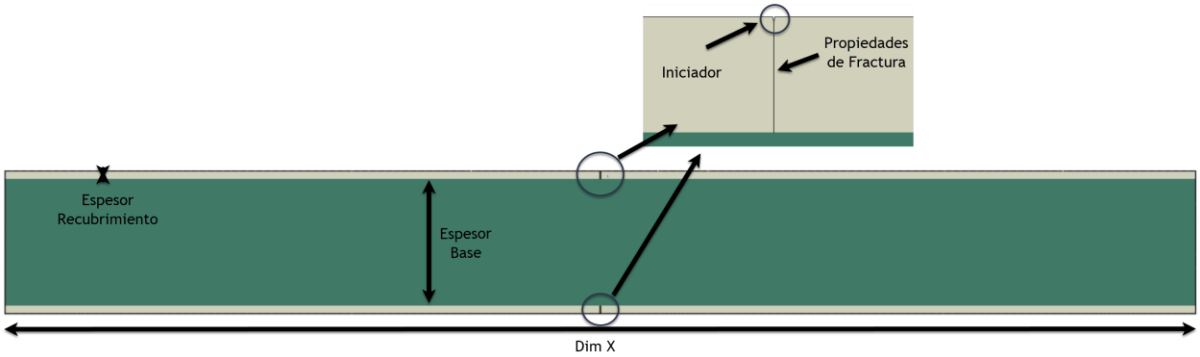


Figura 13. Esquema de la probeta para la primera fractura.

El tamaño del iniciador varia de 1 μm hasta 10 μm, este iniciador hace referencia a una grieta ubicada en la parte superior del recubrimiento que se crea como resultado del proceso de HVOF, este, al ser un proceso de deposición de partículas a alta velocidad, da como resultado un recubrimiento con una superficie no uniforme como se puede observar en la Figura 14, las imperfecciones en dicha superficie actúan como iniciadores o grietas en las que se pueden originar fractura.

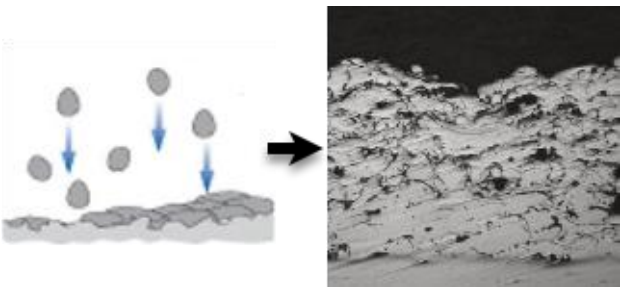


Figura 14. Resultado del depósito de partículas por HVOF

Originalmente, la probeta tiene una dimensión horizontal dim x, ver Figura 13, de 0.064 [m]. Los parámetros dimensionales de la probeta están fuertemente ligados al tiempo de cómputo y los efectos de borde. En especial los efectos de borde pueden tener una afectación al modo de fractura como se indica en la Figura 15. Es por esta razón que se debe seleccionar un tamaño de probeta que sea representativo y que no afecte a los dos factores antes mencionados. Para esto, se selección dimx = 0.03 [m] y los resultados con esta dimensión se

presentan para la primera fractura en la sección de resultados: este valor demuestra ser lo suficientemente representativo para evitar los efectos de borde y un excesivo tiempo de simulación.

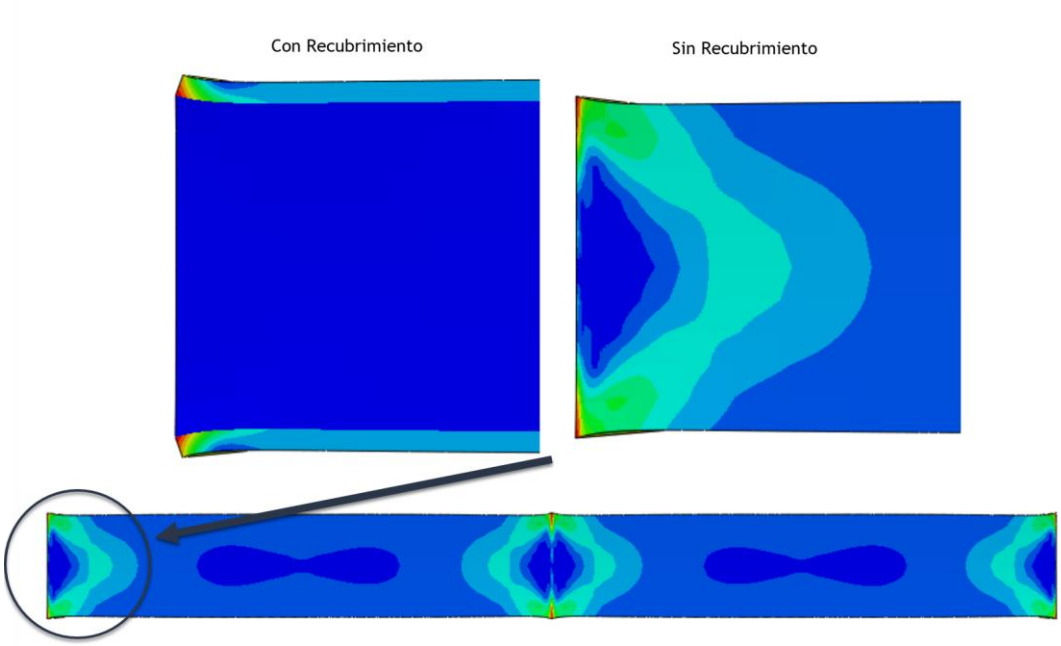


Figura 15. Efecto de borde en probeta

Uno de los factores a considerar es el tipo de mallado utilizado al momento de la simulación. Por este motivo, una vez seleccionado el tamaño de la probeta dim x, se procedió a hacer un análisis de la afectación del tamaño del elemento en la dirección horizontal en las deformaciones y la distribución de esfuerzos. Se encontró que conforme disminuye el tamaño de los elementos en la dirección horizontal, la deformación para la ruptura disminuye (Figura 17) y la distribución de esfuerzos es más uniforme (Figura 18)

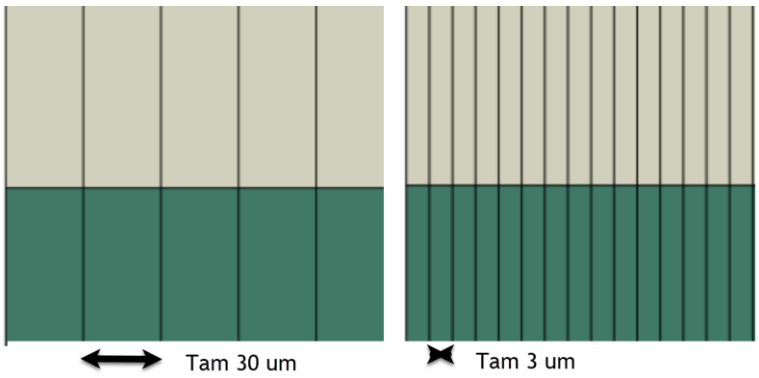


Figura 16. Tamaño del elemento en el eje x

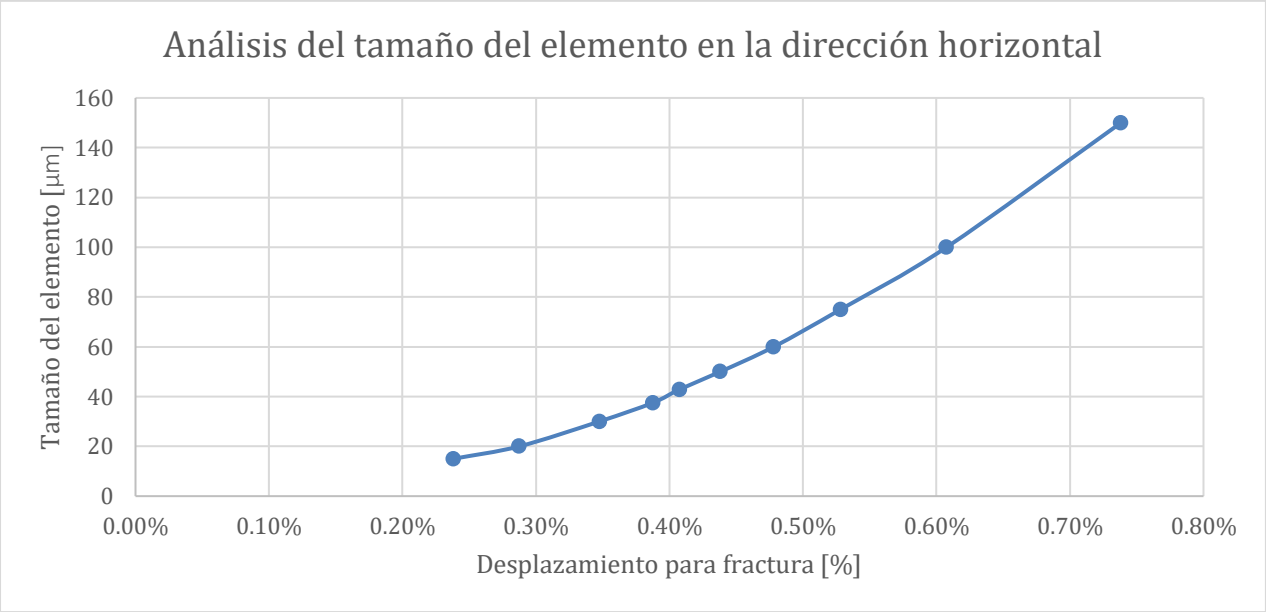


Figura 17. Grafica del análisis del tamaño de elemento en la dirección horizontal para una probeta de 0.03 [m] de largo, iniciador de 1 [μm] y K_{IC} de 6.27 [MPa $m^{0.5}$]

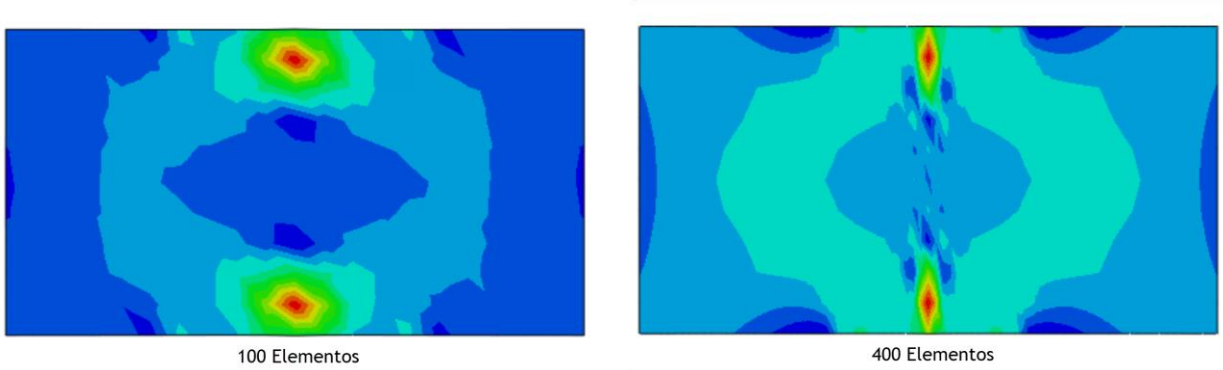


Figura 18. Diferencia de la distribución de esfuerzos en un mallado fino y en un mallado grueso tras la ruptura del recubrimiento

Luego se procede a hacer un análisis de la afectación, si hubiere alguna, del tamaño del elemento en la dirección vertical a la deformación, distribución de esfuerzos y tiempo de computo. Al variar el tamaño del elemento se obtuvo los datos presentados en la Tabla 3 y se puede notar que el cambio de tamaño del elemento en y solo tiene afectación para el tiempo de

computo. De estos datos, se escoge un tamaño de elemento en la dirección vertical de 320 [μm].

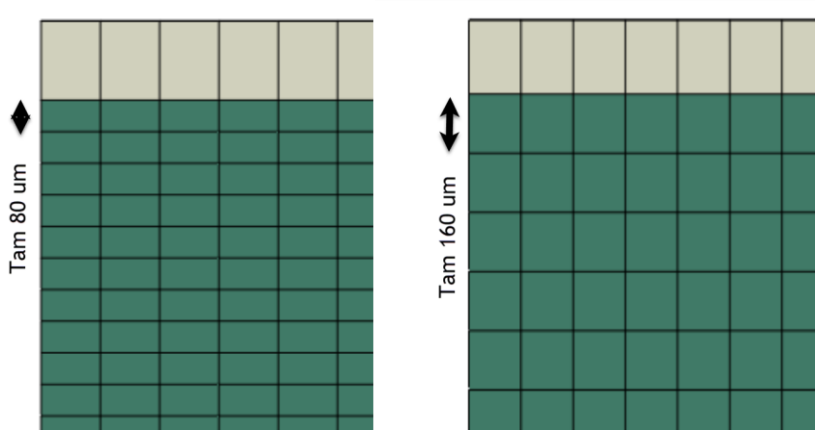


Figura 19. Tamaño del elemento en la dirección vertical

Tabla 3. Análisis para escoger el tamaño del elemento en la dirección vertical

Iniciador [μm]	K_{IC} [MPa $m^{0.5}$]	dimx [m]	Tamaño elemento en x [μm]	Tamaño elemento en y [μm]	% de deformación a la ruptura	Tiempo de computación
1	6.27	0.03	150	320	0.74%	2 min 14 seg
1	6.27	0.03	150	160	0.75%	3 min 33 seg
1	6.27	0.03	150	80	0.76%	4 min 24 seg
1	6.27	0.03	100	320	0.61%	2 min 15 seg
1	6.27	0.03	100	160	0.61%	4 min 20 seg
1	6.27	0.03	60	320	0.48%	4 min 16 seg
1	6.27	0.03	60	32	0.48%	41 min

Para realizar la calibración de la primera fractura, el análisis a realizar es para observar si la primera fractura depende del número de grietas que hay en el recubrimiento, la Figura 20 muestra dos modelos uno con 1 grieta superior e inferior y uno con 7 grietas superiores e

inferiores, en estos dos modelos se procede a realizar el análisis de elementos finitos para observar si los resultados son semejantes o si el número de grietas afecta a la fractura.

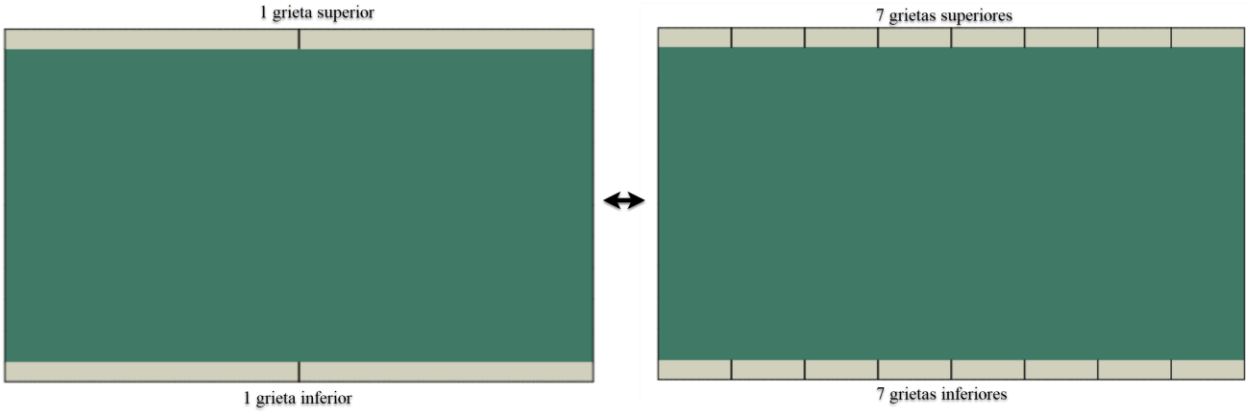


Figura 20. Modelo 1 grieta vs modelo 7 grietas

Para este modelo, un criterio a considerar es la probabilidad de que una nueva grieta produzca ruptura en el modelo, Para hacer estas pruebas de probabilidad, se debe observar cómo se distribuyen los esfuerzos en la probeta una vez que la misma ha fracturado, como se observa en la Figura 21, se puede observar que la fractura se origina en lugares donde el esfuerzo se acumula hasta llegar a un valor crítico.

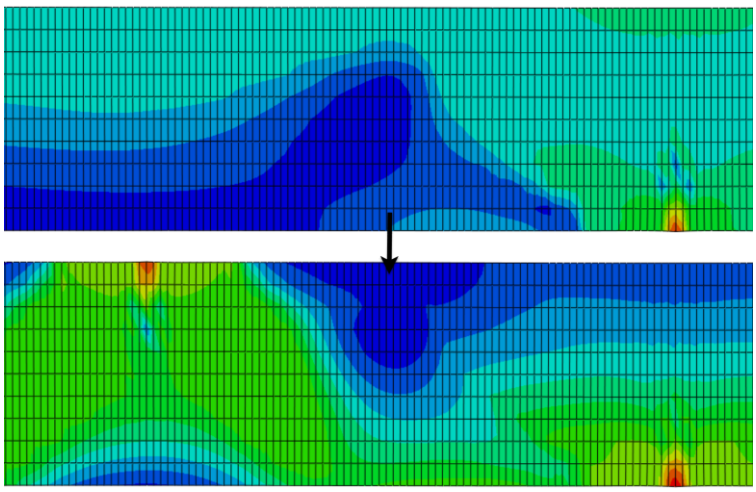


Figura 21. Distribución de esfuerzos tras la ruptura

Por último, para este modelo, se debe observar cual es el efecto del tamaño del recubrimiento en la deformación necesaria para realizar la fractura, los datos experimentales que se tienen

son para recubrimientos de 200 [μm], pero se hará un barrido para valores desde 50 [μm] hasta 400 [μm] con los valores de resistencia a la fractura obtenidos en los análisis anteriores.

2. Modelo: Probeta Corta

Para este modelo, los parámetros que se deben ajustar son el tipo del mallado que se utilizará, es decir, las dimensiones de los elementos en dirección vertical y dirección horizontal; y la dimensión del sustrato lateral. El esquema del modelo se lo presenta en la Figura 22.

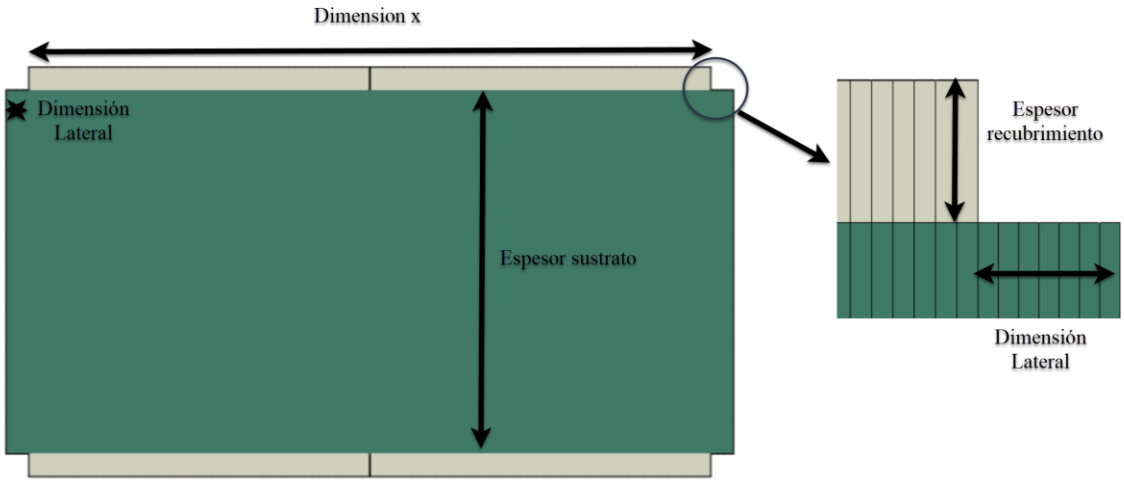


Figura 22. Esquema de la probeta para la segunda fractura.

Para el tipo de mallado, se utiliza un tamaño de elemento en la dimensión vertical de 320 [μm] que fue justificado en el análisis del primer tipo de fractura. Para el tamaño de elemento en la dirección horizontal, se utiliza el valor de 30 [μm], siendo este el tamaño promedio de una splat de WC-12%Co.

Para la dimensión de la probeta y la dimensión lateral del sustrato sin recubrimiento, se realiza un barrido de las posibles medidas para la dimensión hasta obtener una convergencia.

La dimensión de la probeta con recubrimiento que tiene más convergencia es de 0.006 [m] o 6 [mm]. Para dimensiones más pequeñas o grandes, los modelos no tienen fractura o fracturan solo a especificas condiciones lo que dificulta el análisis, por lo que todos los análisis se los realiza con un tamaño para la dimensión de la probeta con recubrimiento de 0.006 [m]. Para la dimensión del tamaño del material lateral se realizó un barrido de 1[mm] hasta 0.1

[mm] para esta dimensión y se obtuvo una convergencia para tamaños menores a 0.2[mm]. Los análisis se los realiza con un tamaño para la dimensión del material lateral de 0.1[mm].

3. Modelo: Análisis de espaciamiento de grietas

Según los datos experimentales, el espaciado entre grieta afecta a la deformación para fracturar el recubrimiento. Sin embargo, ABAQUS no tiene convergencia cuando se ubica grietas demasiado cercanas una a otra, por lo que para poder estudiar el efecto del espaciado entre grietas se plantea el modelo mostrado en la Figura 24. Se varia la dimensión del espaciado acorde a los datos de la Tabla 2; sin embargo, se realiza un ajuste a los valores del espaciado para que se ajusten al tamaño de 30 [μm] que es el tamaño de cada elemento en la dimensión x que se utilizara en los cálculos, los valores del espaciado ajustado se los adjunta en la Tabla 4, se realiza este ajuste para que el valor del espaciado entre fractura sea divisible para el tamaño del elemento y se puedan incluir propiedades de fractura en las superficies. El código para este modelo se lo adjunta en el Anexo H.

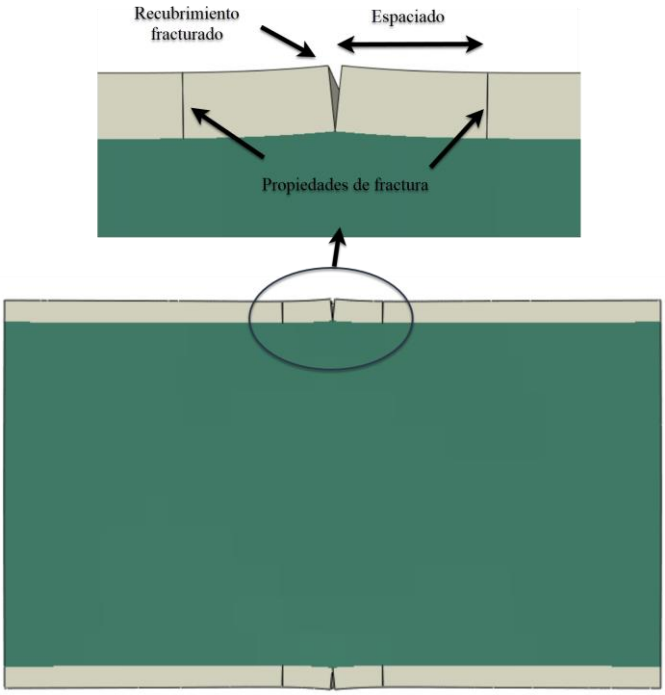


Figura 23. Modelo de ruptura para análisis de espaciado entre fracturas

Tabla 4. Datos del espaciado ajustado de iniciadores de fractura.

Espaciado entre fracturas [µm]	Espaciado entre fracturas ajustado [µm]
650	660
540	540
490	480
440	450
425	420
380	390
360	360

RESULTADOS

1. *Modelo: Probeta Larga*

Para este modelo, se utilizan dos valores del tamaño del elemento en la dimensión horizontal que se mostraba en la Figura 16: uno con un tamaño de 100 micras y otro de 30 micras.

Aquí extraemos el primer par de valores de la Tabla 2: (Deformación 0.6% a un espaciado de 650[μm]). Este valor también se lo adjunta a las gráficas como una línea horizontal; cada punto en las grafica muestra el conjunto deformación-iniciador-K_{IC} que tuvo esa simulación en donde la deformación graficada es la deformación global de la probeta a la primera ruptura.

Primera Fractura

Para esto utilizando el código adjunto en el Anexo D, variando los iniciadores de 1 a 10 [μm] y la resistencia a la fractura de 3 a 20 [MPa $m^{0.5}$] se presentan las siguientes graficas:

Para un tamaño de elemento de 30 [μm]:

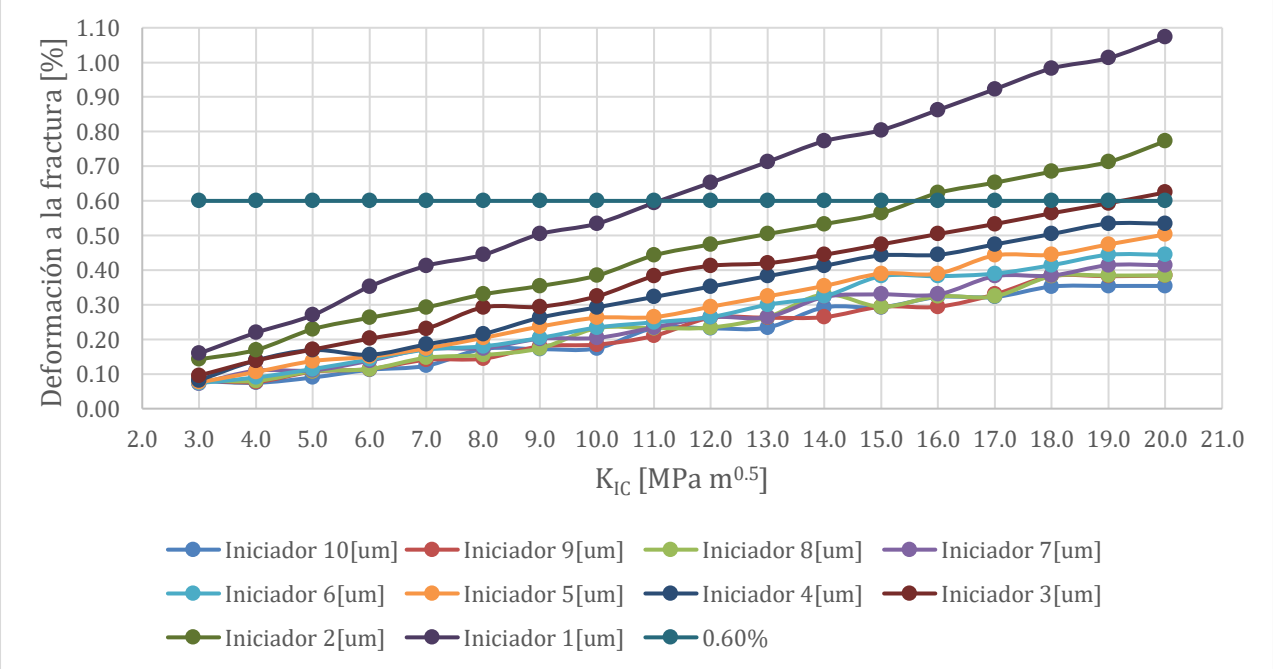


Figura 24. Deformación a la fractura vs la resistencia a la fractura para un tamaño de elemento de 30 [μm], con iniciadores de varios tamaños

De esta gráfica, se puede interpolar los valores de K_{IC} para cada iniciador al tomar los valores tanto de deformación como de K_{IC} al acercarse al valor del 0.6% que representa el primer punto de fractura. Con estos datos, se presenta la Tabla 5 mostrada a continuación:

Tabla 5. Resistencia a la fractura para modelos con tamaño de elemento en la dirección horizontal de 30 [μm]

Iniciador [μm]	K_{IC} [MPa $m^{0.5}$]			
	Primera Fractura			
1	Deformación	0.59%	0.60%	0.65%
	K_{IC}	11	**11.17**	12
2	Deformación	0.56%	0.60%	0.62%
	K_{IC}	15	**15.67**	16
3	Deformación	0.59%	0.60%	0.62%
	K_{IC}	19	**19.33**	20

De esta tabla podemos extraer los valores de K_{IC} para iniciador de 1, 2 y 3[μm] como se presenta en la Tabla 6.

Tabla 6. Datos del tamaño del iniciador con su respectiva resistencia a la fractura para modelos con tamaño de elemento en la dirección horizontal de 30 [μm]

Tamaño Iniciador [μm]	K_{IC} [MPa $m^{0.5}$]
1	11.17
2	15.67
3	19.33

Con estos tres sets de valores obtenidos se procede a hacer un análisis de esfuerzos y deformaciones en la zona critica de la fractura.

Este análisis se lo realiza antes de la fractura del recubrimiento ya que en este instante se pueden obtener los esfuerzos y deformaciones que ocasionan la fractura, el instante antes y después de la fractura se lo puede observar en las Figuras 26 y 27.

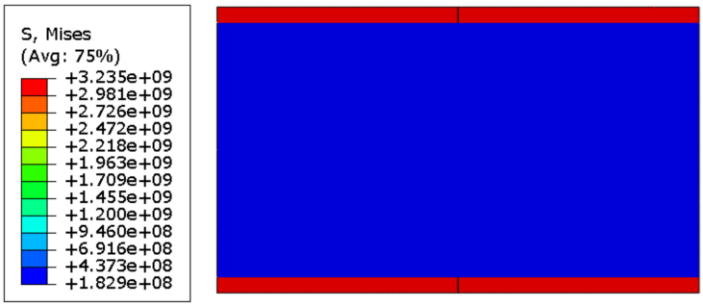


Figura 25. Instante antes de la fractura

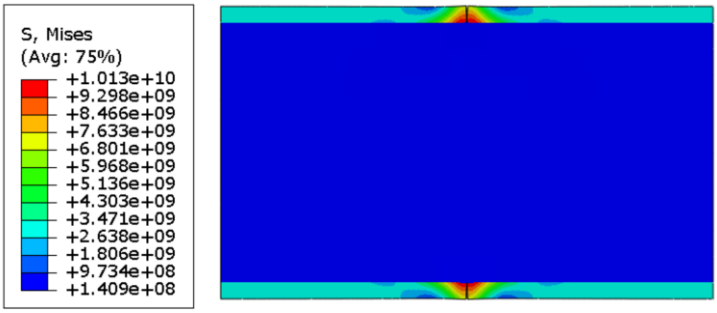


Figura 26. Instante después de la fractura

De las simulaciones, se puede obtener el rango de esfuerzos, deformaciones elásticas y deformaciones plásticas tanto para el recubrimiento como para el sustrato individualmente como de forma conjunta como se muestra a continuación:

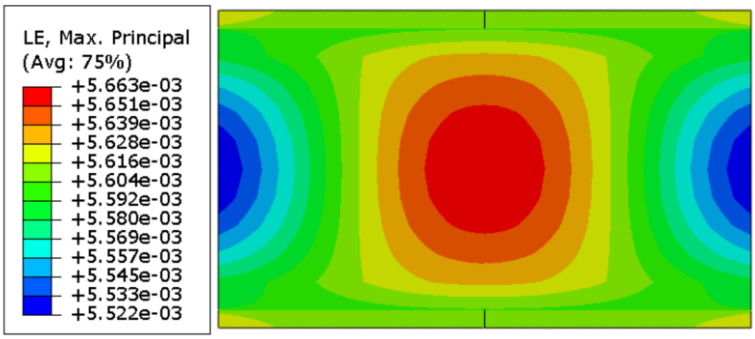


Figura 27. Deformaciones elásticas para conjunto recubrimiento-sustrato

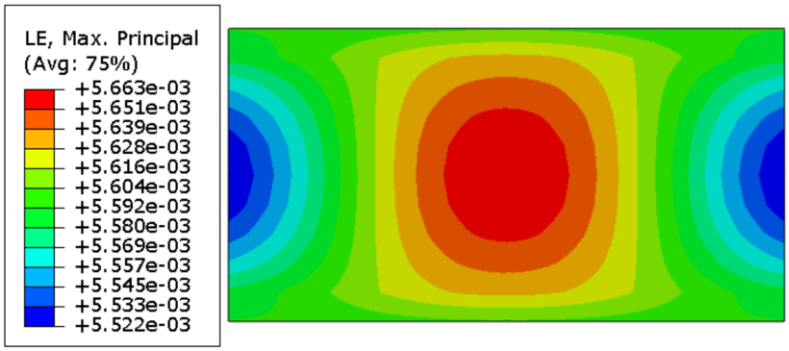


Figura 28. Deformaciones elásticas para sustrato

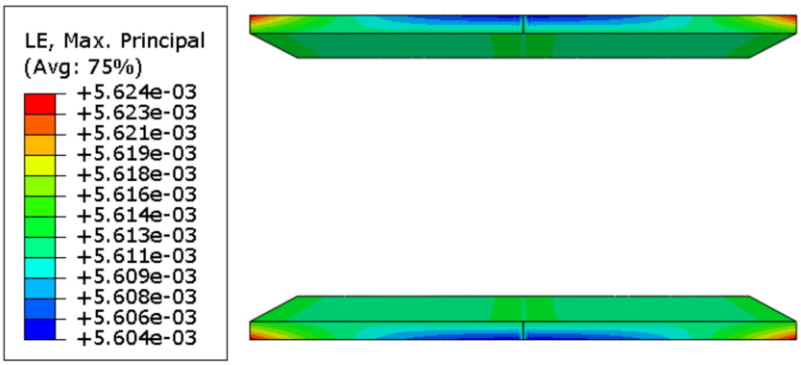


Figura 29. Deformaciones elásticas para recubrimiento

Con esta forma de extraer datos, se extrae el rango de esfuerzos y deformaciones tanto para el sustrato como para el recubrimiento, esto para cada uno de los pares iniciador-K_{IC}, los resultados de esfuerzos y deformaciones tanto en el sustrato como en el recubrimiento antes de que se produzca fractura se los presenta en la tabla adjunta a continuación.

Tabla 7. Esfuerzos y deformaciones para modelos de tamaño de elemento de 30 [μm]

Tamaño Iniciador [μm]	K_{IC} [MPa $m^{0.5}$]	Deformación [%]		Esfuerzo	
		Sustrato	Recubrimiento	Sustrato [MPa]	Recubrimiento [GPa]
1	11.17	0.47	0.562	186.5	3.24
2	15.67	0.45	0.54	185.9	3.09
3	19.33	0.45	0.54	185.9	3.12

Como se puede observar, los valores tanto de esfuerzo como de deformación críticos antes de la fractura; son valores similares en todos los sets de iniciador-resistencia a la fractura.

Los graficas de los resultados se presentan a continuación; se presentan del modelo con tamaño de iniciador de 3 [μm] con resistencia a la fractura de 19.33 [MPa $m^{0.5}$].

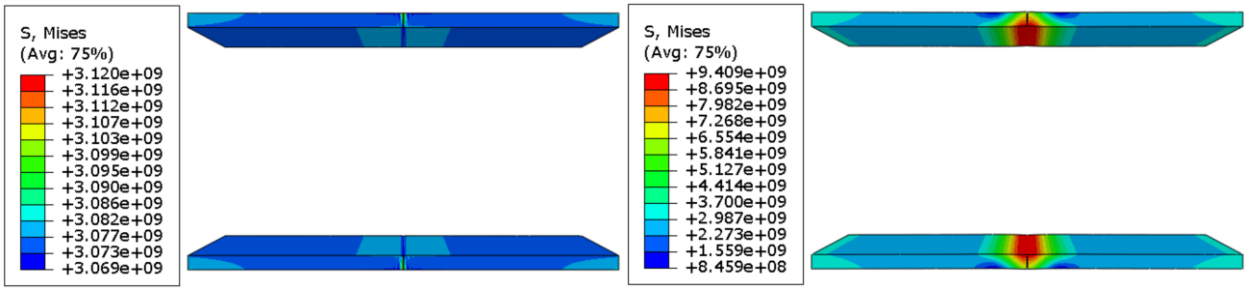


Figura 30. Esfuerzos en el recubrimiento antes y después de la fractura para elementos de 30 [μm]

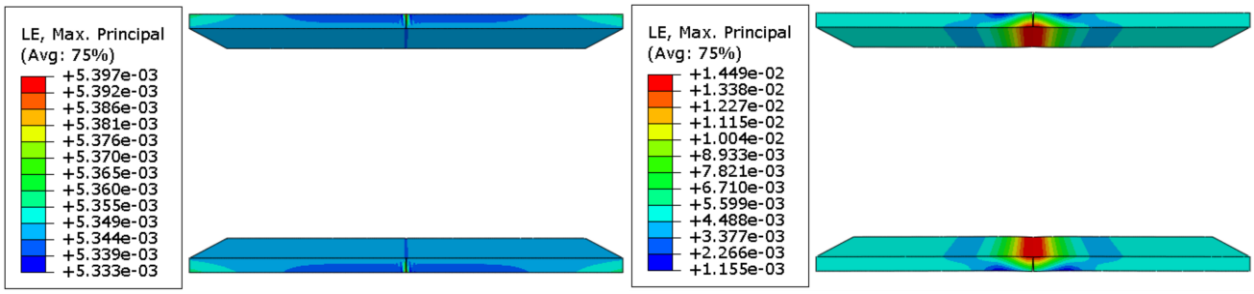


Figura 31. Deformaciones elásticas en el recubrimiento antes y después de la fractura para elementos de 30 [µm]

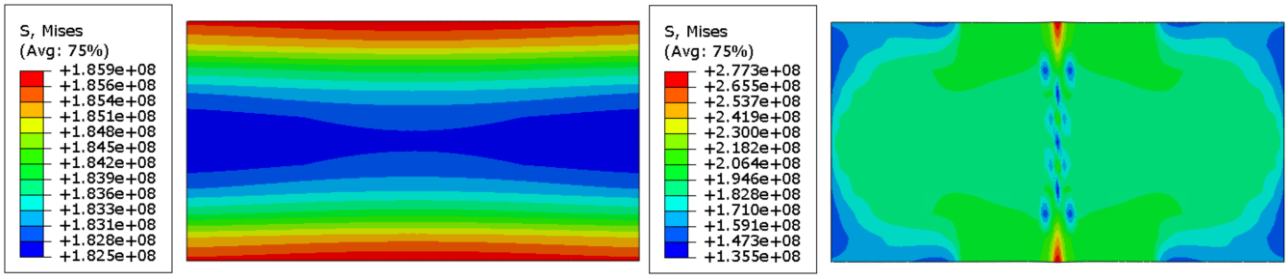


Figura 32. Esfuerzos en el sustrato antes y después de la fractura para elementos de 30 [µm]

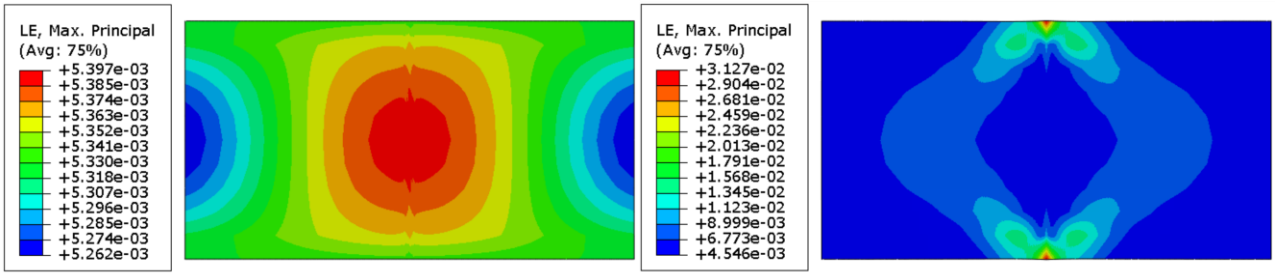


Figura 33. Deformaciones elásticas en el sustrato antes y después de la fractura para elementos de 30 [µm]

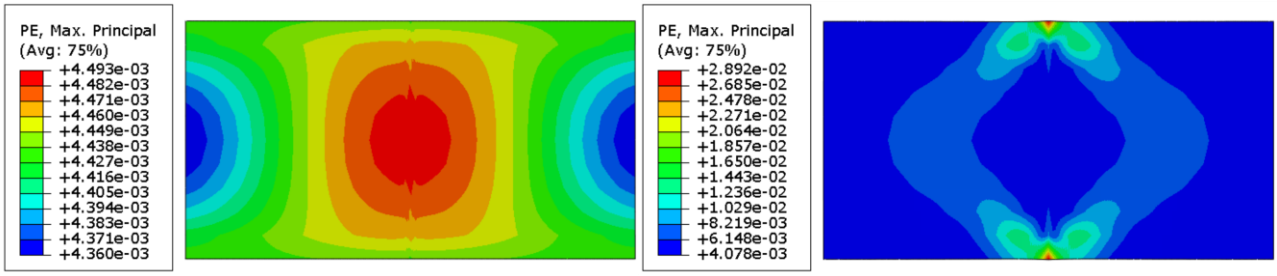


Figura 34. Deformaciones plásticas en el sustrato antes y después de la fractura para elementos de 30 [µm]

Para un tamaño de elemento de 100 [µm]:

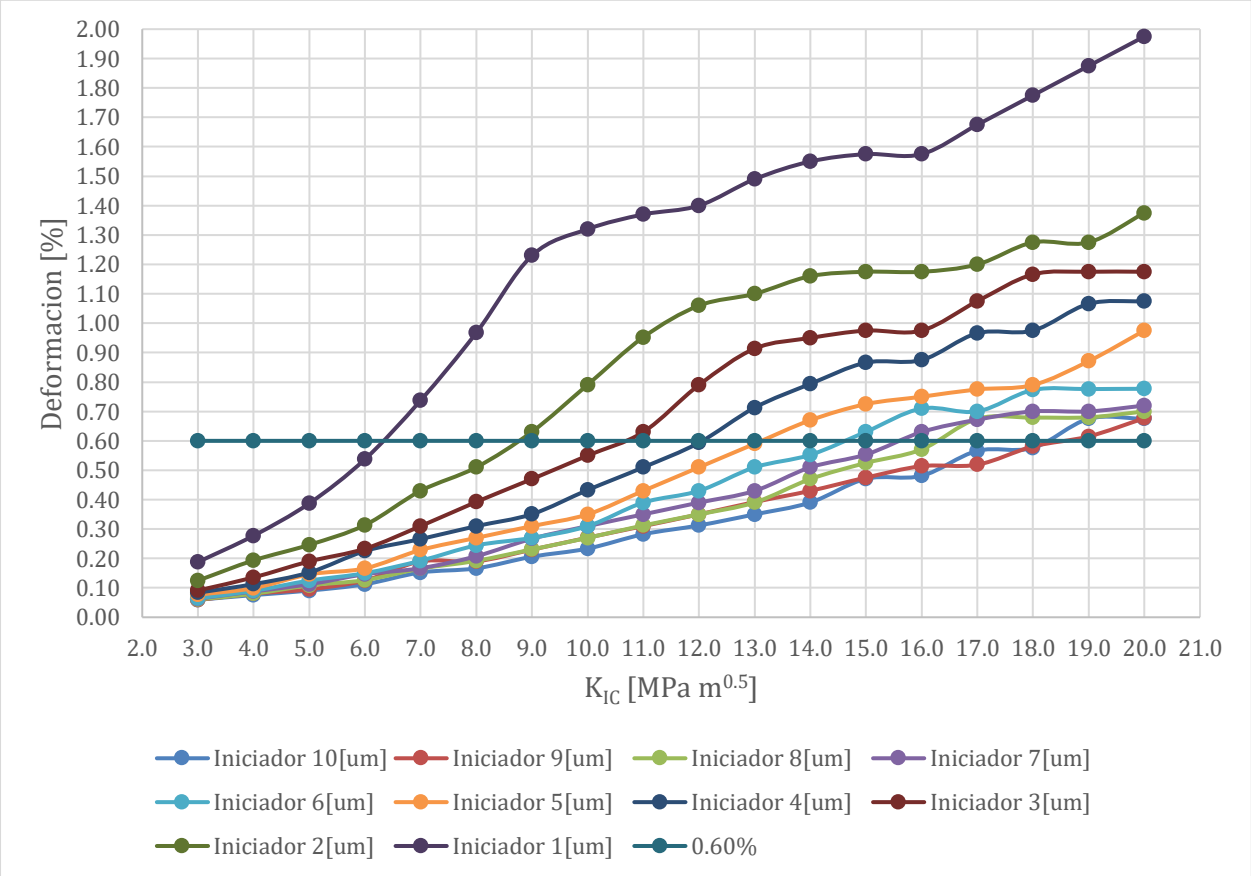


Figura 35. Grafica de la deformación a la fractura vs resistencia a la fractura para un tamaño de elemento de 100 [μm]

De la misma manera de esta gráfica, se puede interpolar los valores de K_{IC} para cada iniciador al tomar los valores tanto de la deformación como de K_{IC} al acercarse al valor del

0.6% que representa el primer punto de fractura. Con estos datos, se presenta la Tabla 8 mostrada a continuación:

Tabla 8. Resistencia a la fractura para modelos con tamaño de elemento en el eje x de 100 [μm]

Iniciador [μm]	K_{IC} [MPa $m^{0.5}$]			
	Primera Fractura			
1	Deformación	0.54%	0.60%	0.74%
	K_{IC}	6	6.3	7
2	Deformación	0.51%	0.60%	0.63%
	K_{IC}	8	8.75	9
3	Deformación	0.55%	0.60%	0.63%
	K_{IC}	10	10.63	11
4	Deformación	0.59%	0.60%	0.71%
	K_{IC}	12	12.08	13
5	Deformación	0.59%	0.60%	0.67%
	K_{IC}	13	13.13	14
6	Deformación	0.55%	0.60%	0.63%
	K_{IC}	14	14.63	15
7	Deformación	0.55%	0.60%	0.63%
	K_{IC}	15	15.63	16
8	Deformación	0.57%	0.60%	0.68%
	K_{IC}	16	16.27	17
9	Deformación	0.59%	0.60%	0.68%
	K_{IC}	18	18.11	19
10	Deformación	0.58%	0.60%	0.68%
	K_{IC}	18	18.20	19

De esta tabla podemos extraer los valores de K_{IC} para los iniciadores que se presenta en la Tabla 9.

Tabla 9. Datos del tamaño del iniciador con su respectiva resistencia a la fractura para modelos con tamaño de elemento en la dirección horizontal de 100 [μm]

Tamaño Iniciador [μm]	K_{IC} [MPa $m^{0.5}$]
1	6.30
2	8.75
3	10.63
4	12.08
5	13.13
6	14.63
7	15.63
8	16.27
9	18.11
10	18.20

Se extrae el rango de esfuerzos y deformaciones en el instante antes de producir fractura tanto para el sustrato como para el recubrimiento, esto para cada uno de los pares iniciador-K_{IC}, los resultados se los presenta en la tabla a continuación.

Tabla 10. Esfuerzos y deformaciones para modelos de tamaño de elemento de 100 [μm]

Tamaño Iniciador [μm]	K_{IC} [MPa $m^{0.5}$]	Deformación [%]		Esfuerzo	
		Sustrato	Recubrimiento	Sustrato [MPa]	Recubrimiento [GPa]
1	6.3	0.22-0.13	0.32-0.30	179-177	1.85-1.76
2	8.75	0.2-0.12	0.3-0.28	178.8-177	1.72-1.63
3	10.63	0.2-0.12	0.3-0.28	178.8-177	1.72-1.63
4	12.08	0.2-0.12	0.3-0.28	178.9-177	1.72-1.63
5	13.13	0.18-0.11	0.28-0.27	178.6-176.9	1.66-1.54
6	14.63	0.2-0.12	0.3-0.28	178.8-177	1.79-1.61
7	15.63	0.2-0.12	0.31-0.28	178.8-177	1.82-1.6
8	16.27	0.18-0.11	0.3-0.26	178.6-176.9	1.76-1.51
9	18.11	0.22-0.13	0.34-0.3	179.3-177.2	2-1.7
10	18.20	0.22-0.13	0.35-0.3	179.3-177.2	2.1-1.7

Como se puede observar nuevamente, los valores tanto de esfuerzo como de deformación críticos antes de la fractura; son valores similares en todos los sets de iniciador-resistencia a la fractura.

Los graficas de los resultados se presentan a continuación; se presentan del modelo con tamaño de iniciador de 4 [μm] con resistencia a la fractura de 12.08 [MPa $m^{0.5}$].

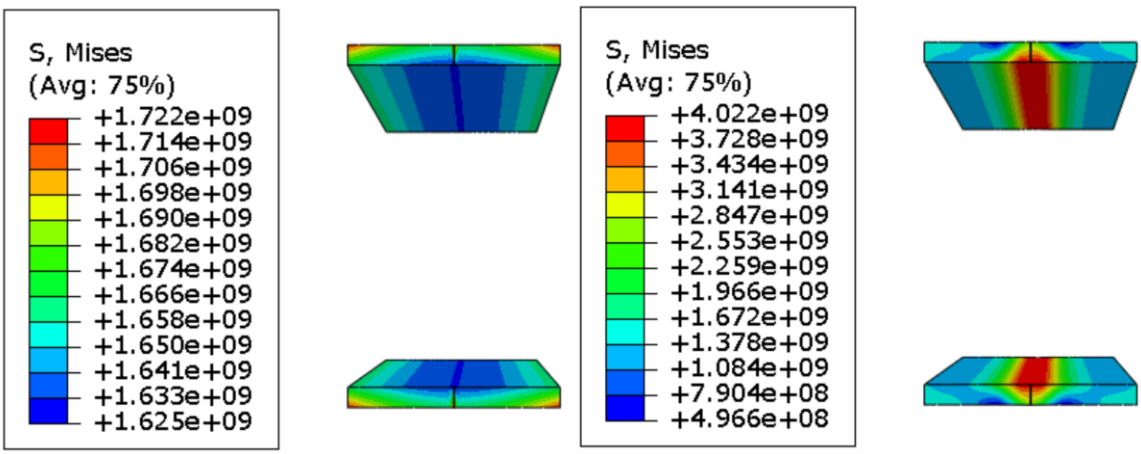


Figura 36. Esfuerzos en el recubrimiento antes y después de la fractura para elementos de 100 [μm]

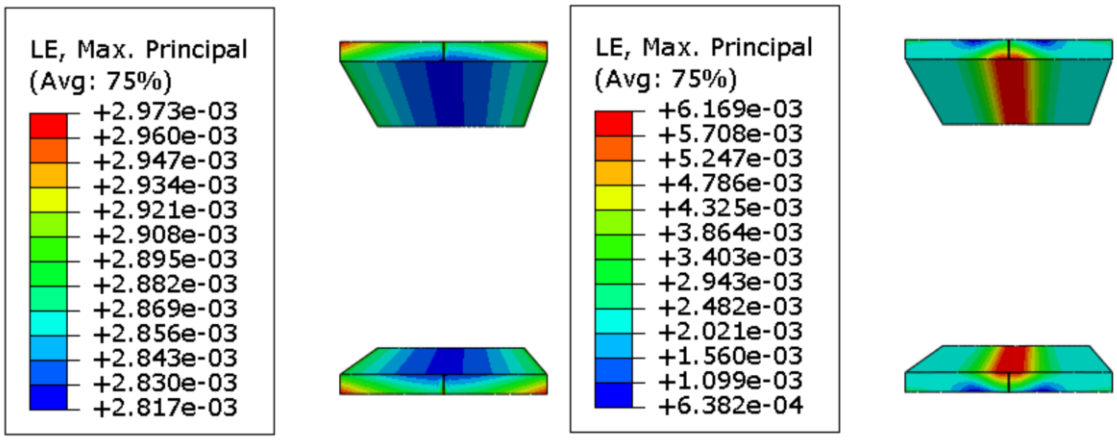


Figura 37. Deformaciones elásticas en el recubrimiento antes y después de la fractura para elementos de 100 [µm]

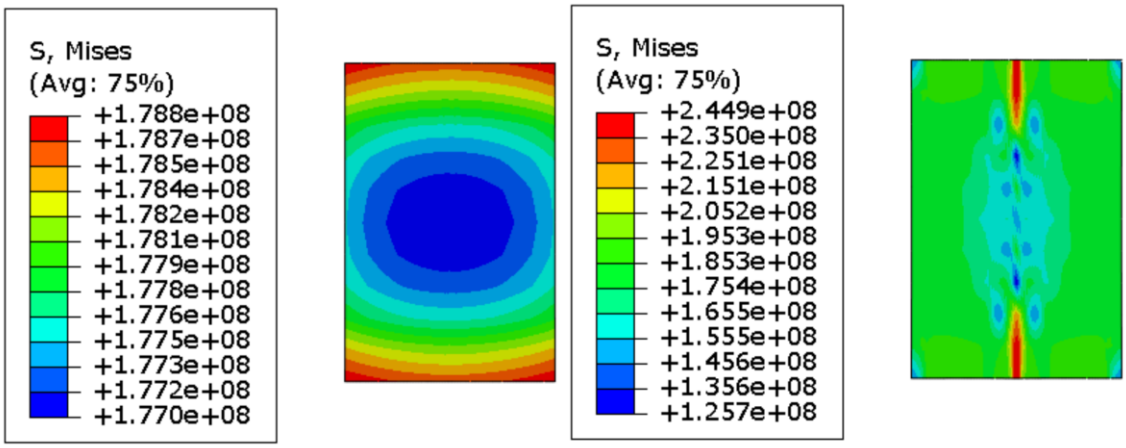


Figura 38. Esfuerzos en el sustrato antes y después de la fractura para elementos de 100 [µm]

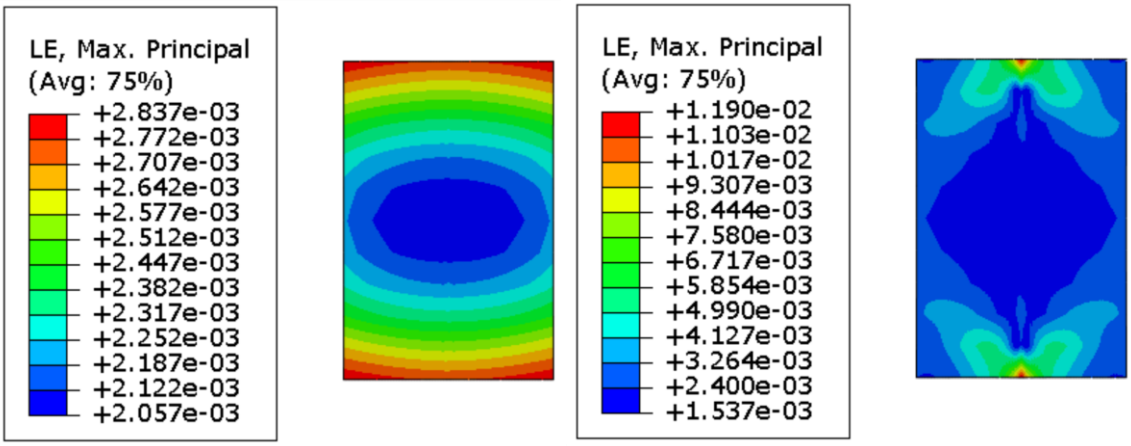


Figura 39. Deformaciones elásticas en el sustrato antes y después de la fractura para elementos de 100 [µm]

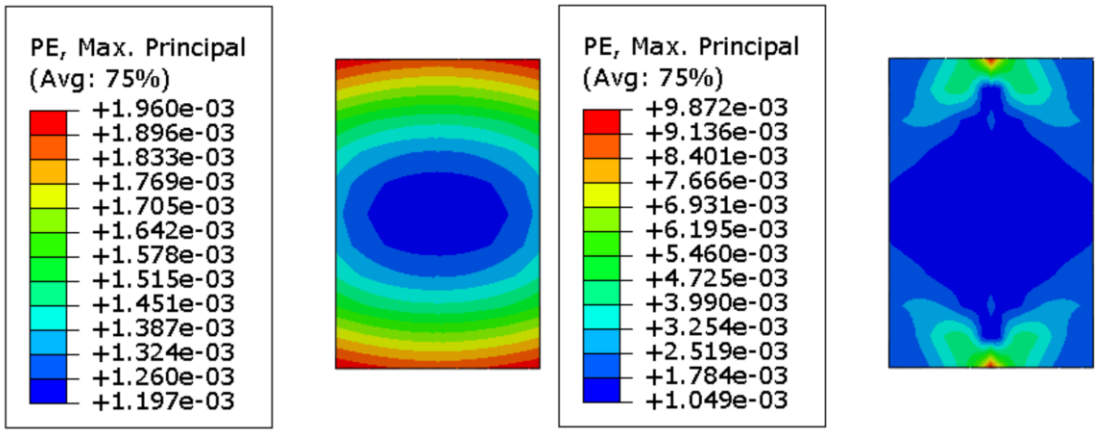


Figura 40. Deformaciones plásticas en el sustrato antes y después de la fractura para elementos de 100 [µm]

Calibración mediante espaciado entre grietas

Los resultados para este análisis muestran que no hay diferencia entre ingresar 1 grieta superior e inferior que ingresar 7 grietas superiores e inferiores lo que corrobora al análisis de la primera fractura. Como se puede observar en las Figuras 41, 42, 43 y 44, la fractura ocurre a una deformación aproximada del 0.6% y todas las rupturas ocurren de manera instantánea, en las Figuras se incluye el instante antes y después de la fractura.

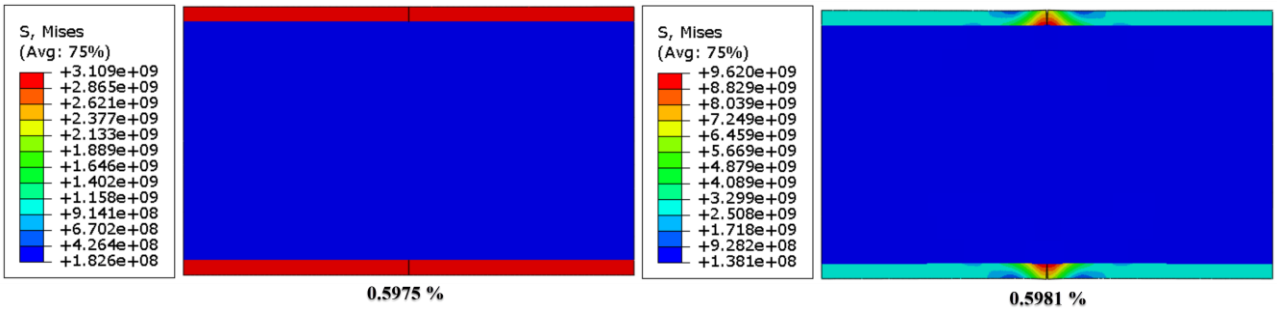


Figura 41. Modelo con 1 grieta superior e inferior

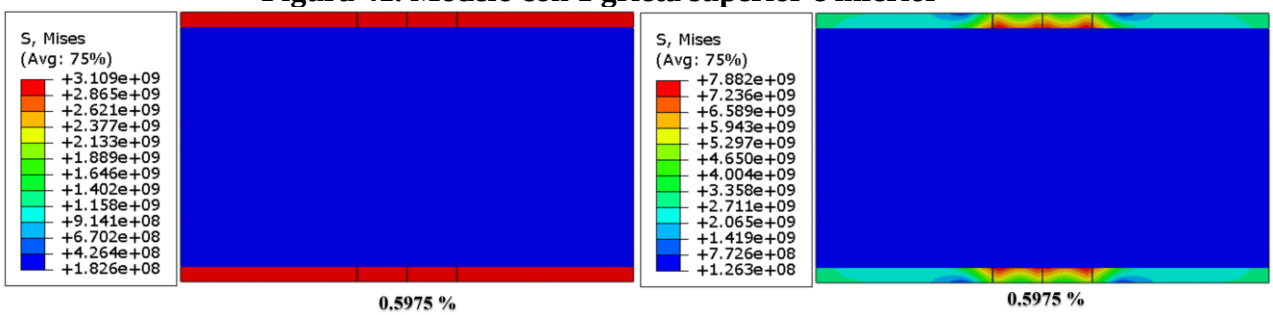


Figura 42. Modelo con 3 grietas superiores e inferiores

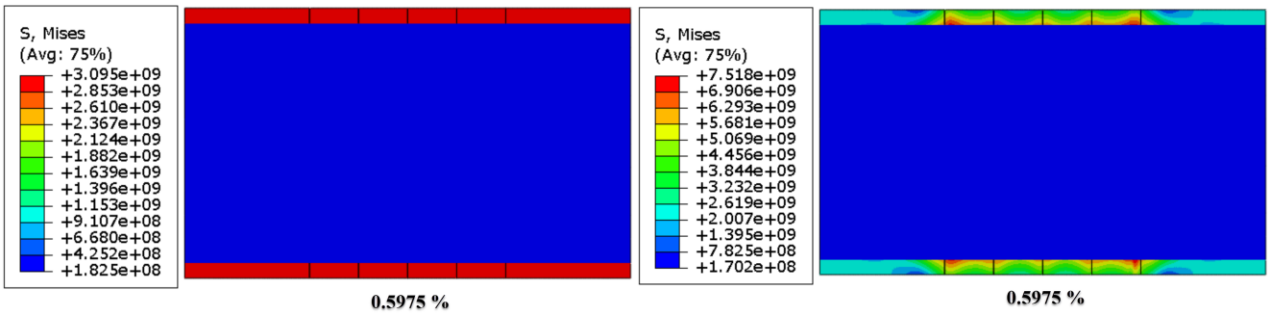


Figura 43. Modelo con 5 grietas superiores e inferiores

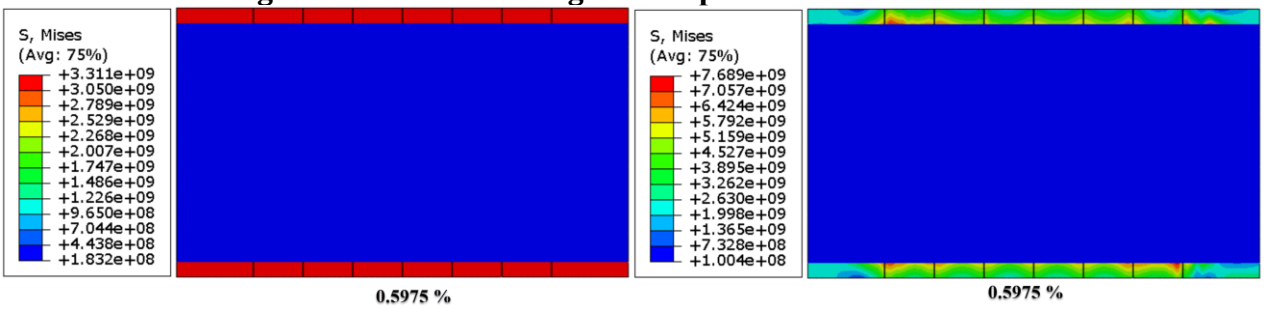


Figura 44. Modelo con 7 grietas superiores e inferiores

Los valores para estos 4 modelos tanto de esfuerzos como deformaciones en el instante antes de la fractura tanto en el sustrato como en el recubrimiento no tienen una diferencia notoria como se muestra en la Tabla a continuación.

Tabla 11. Resultados para esfuerzos y deformaciones en el análisis de calibración en el instante antes de la fractura

Modelo	Deformación [%]		Esfuerzo	
	Sustrato	Recubrimiento	Sustrato [MPa]	Recubrimiento [GPa]
1 grieta	0.4684	0.559	186.1	3.195
3 grietas	0.4683	0.5559	186.1	3.195
5 grietas	0.4683	0.5559	186.1	3.196
7 grietas	0.4682	0.5562	186.1	3.198

Probabilidad de fractura en zonas críticas

Para realizar el análisis de la probabilidad de una nueva fractura, se observan las distribuciones de esfuerzos tras la primera fractura, para esto como ya se mencionó tenemos el criterio de que la fractura se origina en lugares donde el esfuerzo se acumula hasta llegar a un valor crítico, estos valores críticos se los tiene en la Tabla 7, para los análisis, se toma en cuenta

el rango de esfuerzos tanto en el recubrimiento (3.07-3.24 [GPa]) como en el sustrato (182.9-185.9 [MPa]) de la primera fractura para un tamaño de elemento de 30 [μm].

Se analizan como se distribuyen los esfuerzos en la probeta a medida que esta se somete a tracción, en la Figura 45 y Figura 46 podemos observar que hay 3 zonas en el sustrato como en el recubrimiento donde se acumula el esfuerzo critico que se menciona en la Tabla 7. Por medio del código adjunto en el Anexo H se puede realizar probetas reconociendo los elementos donde se quiere ubicar propiedades de fractura.

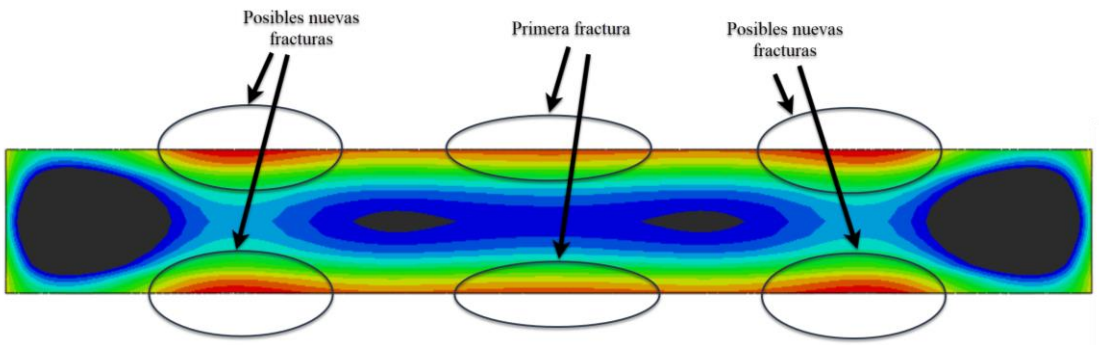


Figura 45. Zonas críticas sustrato

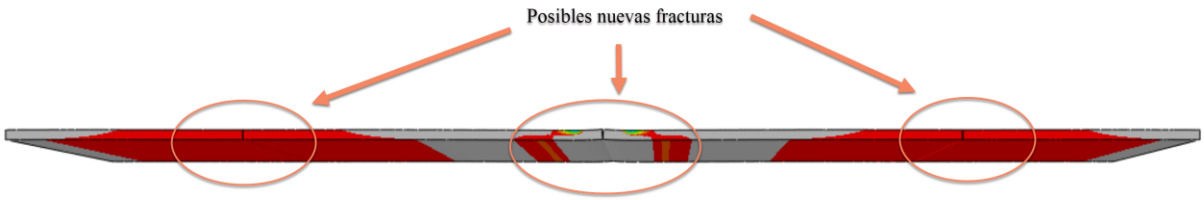


Figura 46. Zonas críticas recubrimiento

Tras la fractura en el centro de la probeta, se ubican propiedades de fractura en las dos zonas críticas que se reconocieron anteriormente, estas zonas se ubican a 7.23 [mm], desde la primera fractura como se puede apreciar en la Figura 47.

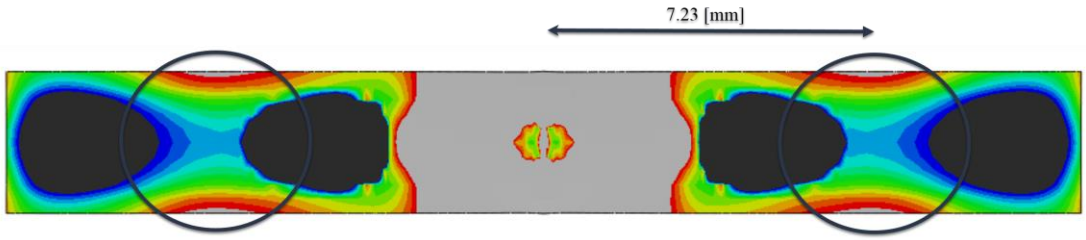


Figura 47. Zona para una nueva fractura

Se tracciona la probeta nuevamente, pero esta probeta ya tiene la fractura en el centro, se observa nuevamente zonas que acumulan el esfuerzo necesario para la ruptura, estas zonas se las puede apreciar en la Figura 48.

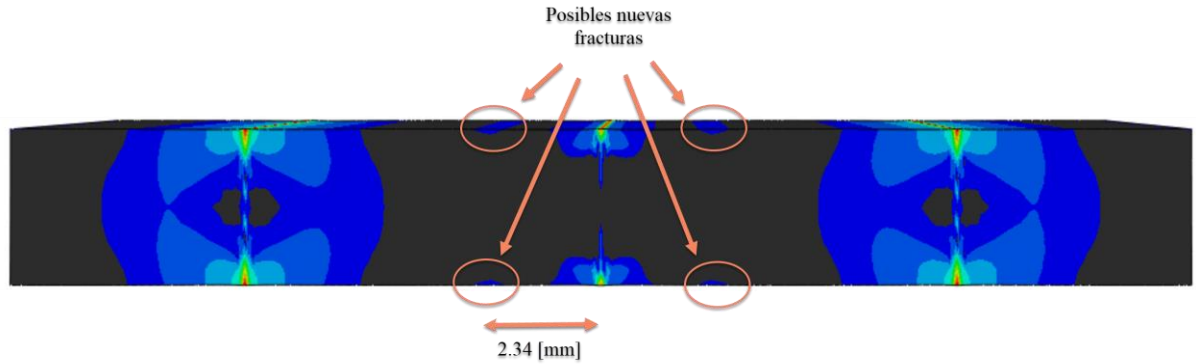


Figura 48. Zonas críticas para una posible tercera fractura

Tras ingresar propiedades de fractura en estas nuevas zonas, se tracciona la probeta y se puede apreciar nuevamente la fractura en estos puntos. Los resultados de la tracción se pueden observar en las Figuras 49, 50 y 51.

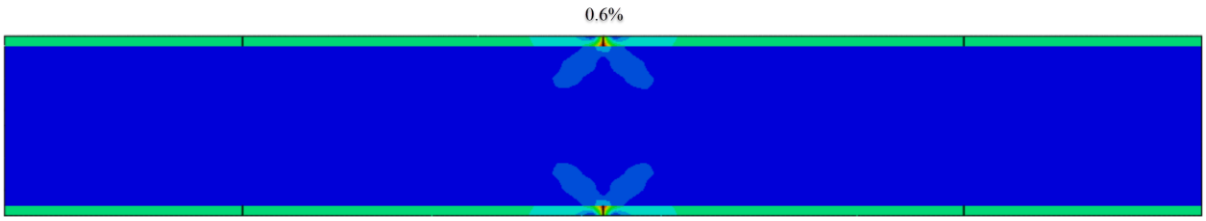


Figura 49. Ruptura al 0.6% en la primera fractura

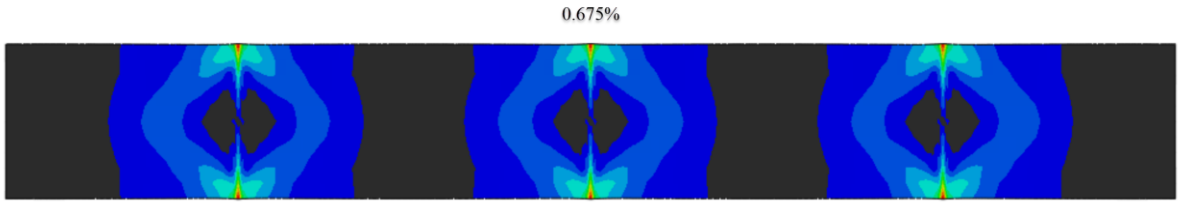


Figura 50. Ruptura al 0.675% en la segunda fractura

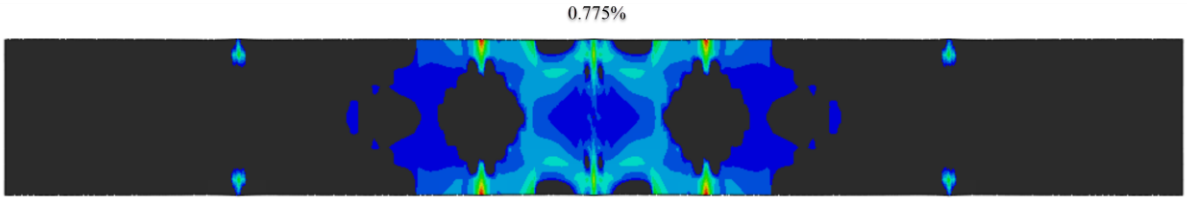


Figura 51. Ruptura al 0.775% en la tercera fractura

Las zonas más críticas para ocasionar nuevas fracturas, se ubican dentro de la zona de ruptura, como se puede observar en la Figura 52, tras la fractura del recubrimiento, en el sustrato se acumulan esfuerzos lo suficientemente críticos para provocar nuevas fracturas. Los esfuerzos se los detalla en la Tabla 12.

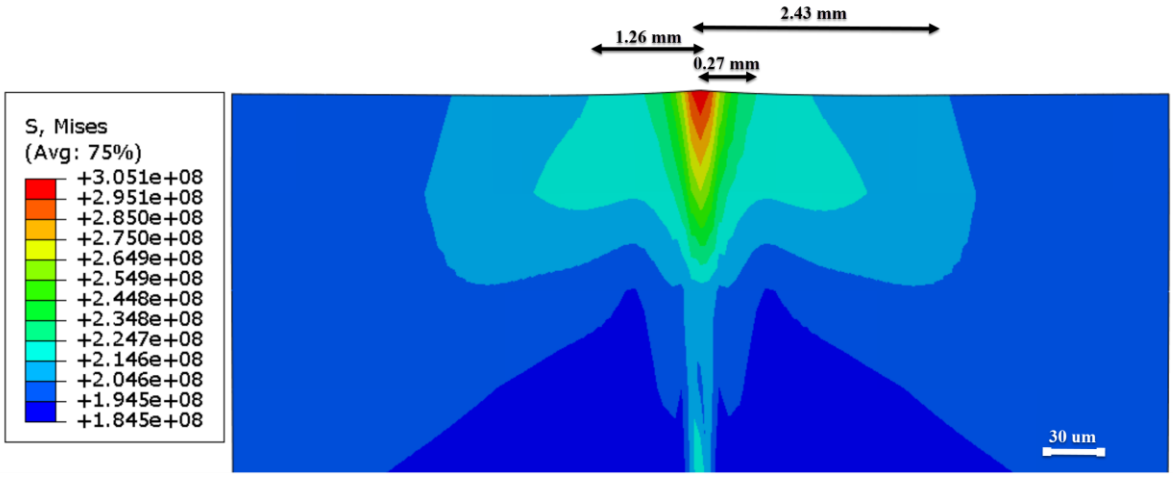


Figura 52. Esfuerzos críticos dentro de la zona de fractura

Tabla 12. Esfuerzos críticos en zonas afectadas tras la fractura

Zona afectada [mm]	Esfuerzo [MPa]
0.27	244.8
1.26	214.6
2.43	184.5

Como se puede observar y según los criterios antes mencionados de que se necesita de un esfuerzo crítico para provocar la fractura, podemos concluir que la zona más propensa a seguir fracturando tras la ruptura del recubrimiento, son zonas a 0.27 [mm] de la fractura, es una zona que llega a tener un esfuerzo de hasta 244.8 [MPa], cuando el esfuerzo necesario es apenas 185.9 [MPa]. Las zonas a 1.26 [mm] también son propensas a fracturarse, pero con una menor probabilidad de ocurrir ya que los esfuerzos llegan a 214.6 [MPa]. Por último, las zonas más alejadas desde 1.26 [mm] a 2.43 [mm] no tienen la posibilidad de fracturar si mantenemos el criterio del esfuerzo crítico.

No se puede realizar este análisis, de ubicar grietas cercanas al punto de fractura, porque no se tiene una convergencia del modelo para los sets iniciadores-K_{IC} de la Tabla 6. Sin

embargo, se puede realizar este tipo de análisis en modelos con iniciadores pequeños y coeficientes de resistencia a la fractura bajos (Iniciador de 1 [μm] y K_{IC} de 3 [MPa $m^{0.5}$], el modelo se lo muestra en la Figura 53), sin embargo, no se puede tener una estimación de la deformación necesaria para realizar la fractura, pero si se puede apreciar la fractura gracias al esfuerzo crítico del sustrato y del recubrimiento en el análisis de probabilidad.

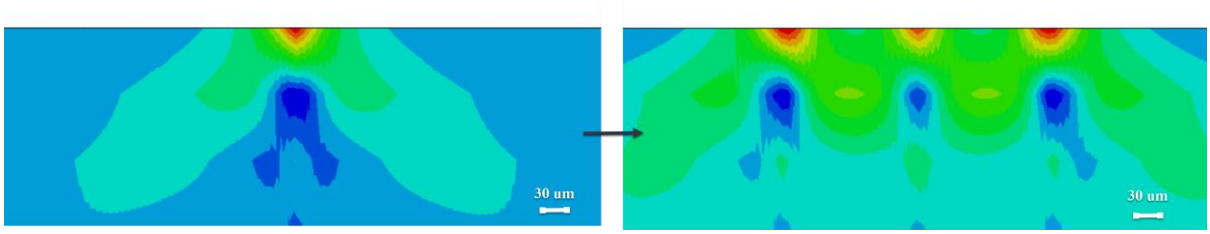


Figura 53. Iniciador de 1 [μm] y K_{IC} de 3 [MPa $m^{0.5}$] para fractura en zonas cercanas

Tamaño del espesor del recubrimiento

El último análisis que se realiza se lo realiza para un iniciador de 1 [μm] con K_{IC} de 11.17 [MPa $m^{0.5}$] calculados en el análisis de la primera fractura y descritos en la Tabla 6. La diferencia del tamaño del espesor del recubrimiento se la puede apreciar en la Figura 54. Los resultados se los adjunta en la Tabla 13 y se muestra la relación en la Figura 55.

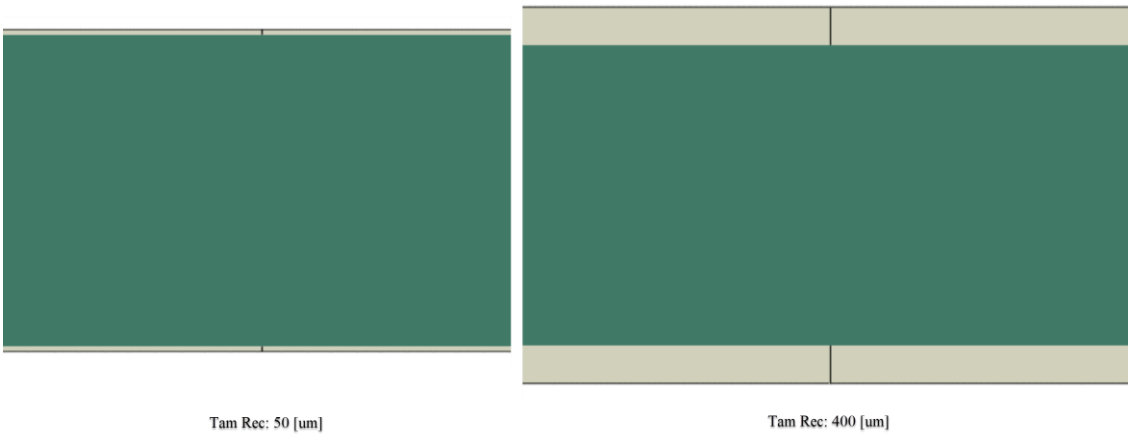


Figura 54. Tamaño del espesor del recubrimiento de 50 [μm] y 400 [μm]

Tabla 13. Resultados de la deformación a la fractura acorde al tamaño del espesor del recubrimiento

Tamaño del espesor del recubrimiento [μm]	% Deformación
50	1.095
100	0.7962
150	0.6762
200	0.5962
250	0.555
300	0.515
350	0.4962
400	0.4762

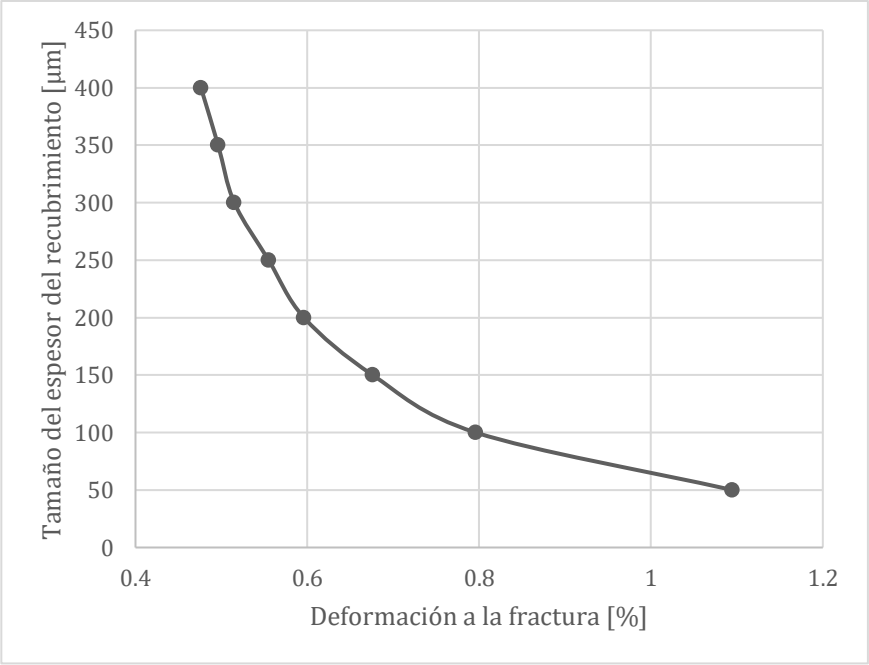


Figura 55. Grafica del tamaño del espesor del recubrimiento vs la deformación a la fractura

Para el tamaño del espesor, se puede observar que a medida que el espesor del recubrimiento es menor, la probeta se puede deformar más antes de la fractura. A su vez, como se sabe que la resistencia a la fractura es independiente del espesor del recubrimiento se obtiene de los modelos, los esfuerzos críticos para cada caso. Los valores se los muestra en la Tabla mostrada a continuación:

Tabla 14. Esfuerzos y deformaciones para modelos en la primera fractura de tamaño de elemento de 30 [μm] variando el tamano del espesor

Tamaño del espesor del recubrimiento [μm]	Deformación [%]		Esfuerzo	
	Sustrato	Recubrimiento	Sustrato [MPa]	Recubrimiento [GPa]
50	0.9765	1.072	193.80	6.30
100	0.67	0.76	189.20	4.41
150	0.54	0.63	187.40	3.62
200	0.45	0.54	186.00	3.09
250	0.40	0.49	185.30	2.80
300	0.36	0.40	184.50	2.53
350	0.33	0.41	184.10	2.37
400	0.30	0.39	183.60	2.21

2. Modelo: Probeta Corta

Para este modelo, el tamaño del elemento en el eje x es de 30 [μm], el tamaño de la dimensión de la probeta es de 0.006 [m] y el tamaño de la dimensión del material lateral es de 0.0002 [m].

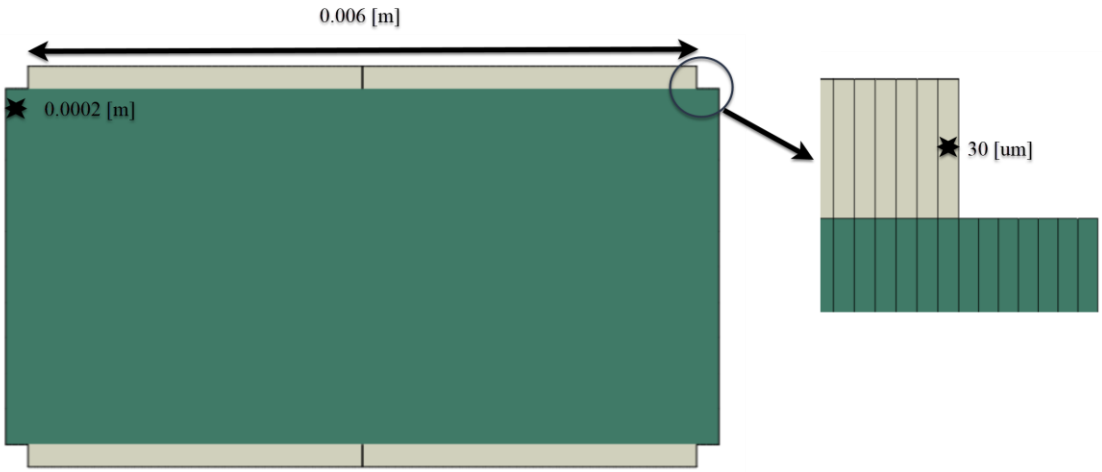


Figura 56. Dimensiones probeta para la segunda fractura.

Las propiedades que se ingresan en el modelo, son las que se obtuvieron y fueron presentadas en la Tabla 6, tanto el tamaño del iniciador de fractura como la resistencia a la

fractura obtenidos del análisis de la primera fractura para un tamaño de elemento en la dirección horizontal de 30 [μm].

Segunda Fractura

Las deformaciones a la fractura que se obtienen para los sets de datos se presentan en la Tabla adjunta a continuación:

Tabla 15. Datos de la deformación para la segunda fractura para modelos con tamaño de elemento en la dirección horizontal de 30 [μm]

Tamaño Iniciador [μm]	K_{IC} [MPa $m^{0.5}$]	% deformación a la fractura
1	11.17	7.02
2	15.67	7.47
3	19.33	7.76

Se extrae el rango de esfuerzos y deformaciones tanto para el sustrato como para el recubrimiento, esto para cada uno de los pares iniciador-K_{IC}, los resultados se los presenta en la tabla a continuación.

Tabla 16. Esfuerzos y deformaciones para modelos en la segunda fractura de tamaño de elemento de 30 [μm]

Tamaño Iniciador [μm]	K_{IC} [MPa $m^{0.5}$]	Deformación [%]		Esfuerzo	
		Sustrato	Recubrimiento	Sustrato [MPa]	Recubrimiento [GPa]
1	11.17	0.2581-1.1	0.3415-0.5791	211.9-230.8	2.015-3.459
2	15.67	0.2373-1.139	0.338-0.5876	213.9-232.7	1.995-3.508
3	19.33	0.2557-1.186	0.3352-0.599	216.5-235.1	1.973-3.575

El instante antes y después de la fractura se lo puede observar en las Figuras 57 y 58.

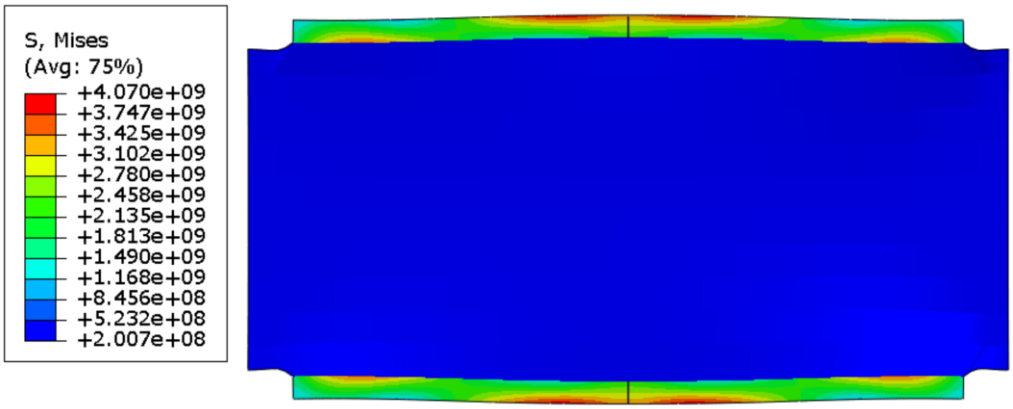


Figura 57. Instante antes de la fractura

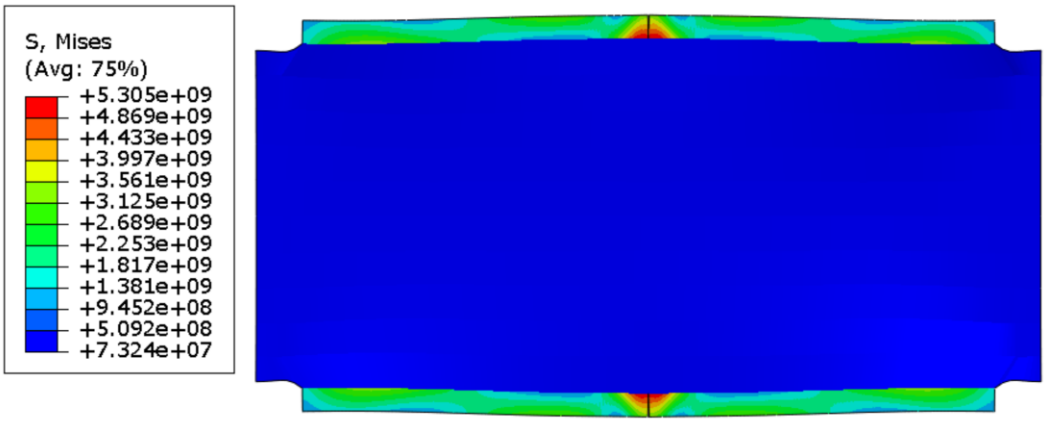


Figura 58. Instante después de la fractura

Los valores que se obtienen de esfuerzo en el sustrato son diferentes a los obtenidos en la primera fractura; sin embargo, los valores del esfuerzo en el recubrimiento entran en el rango

que se obtuvo para la primera fractura, por lo que podemos concluir que se necesita un valor crítico en el recubrimiento más que en el sustrato para ocasionar la fractura.

3. Modelo: Análisis de espaciamiento de grietas

Con los resultados de la Tabla 15 se proceden a hacer los análisis del espaciado. Los modelos se ven como se muestra en la Figura 59.

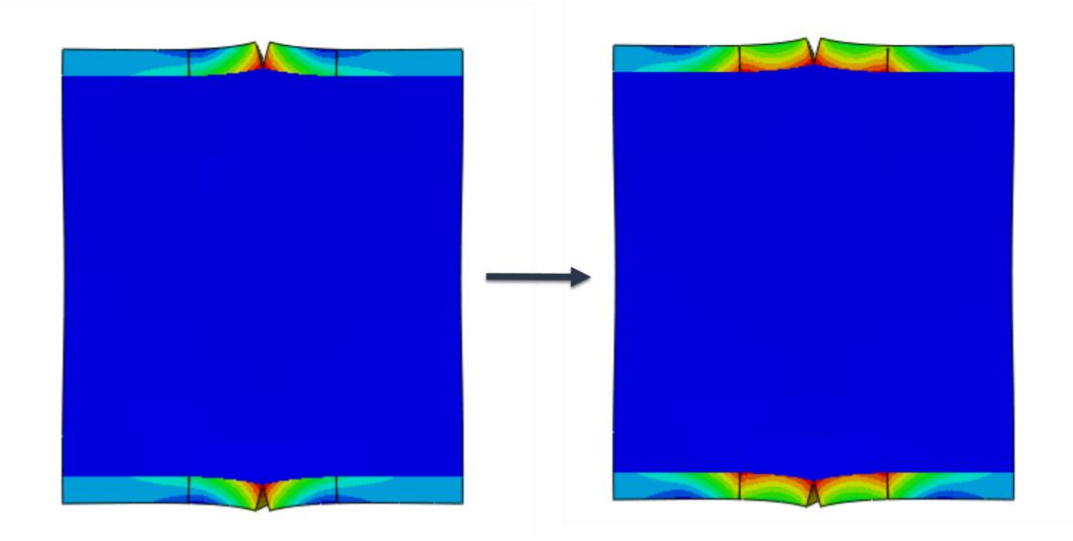


Figura 59. Modelos de ruptura para análisis de espaciado de fractura antes y después de la fractura

El espaciado se lo varia acorde al espaciado ajustado descrito en la Tabla 4. Los resultados que se muestran en la Tabla 17 mostrada a continuación con un iniciador de 1 [μm] y K_{IC} de 11.17 [MPa $m^{0.5}$].

Tabla 17. Resultados del espaciado con su respectiva fractura

Espaciado [μm]	% Deformación a la fractura
660	2.1975
540	3.4515
480	4.3125
450	4.845
420	5.4795
390	6.2475
360	7.194

De la misma forma, se grafican estos puntos para asemejar a los datos experimentales mostrados en la Figura 4.

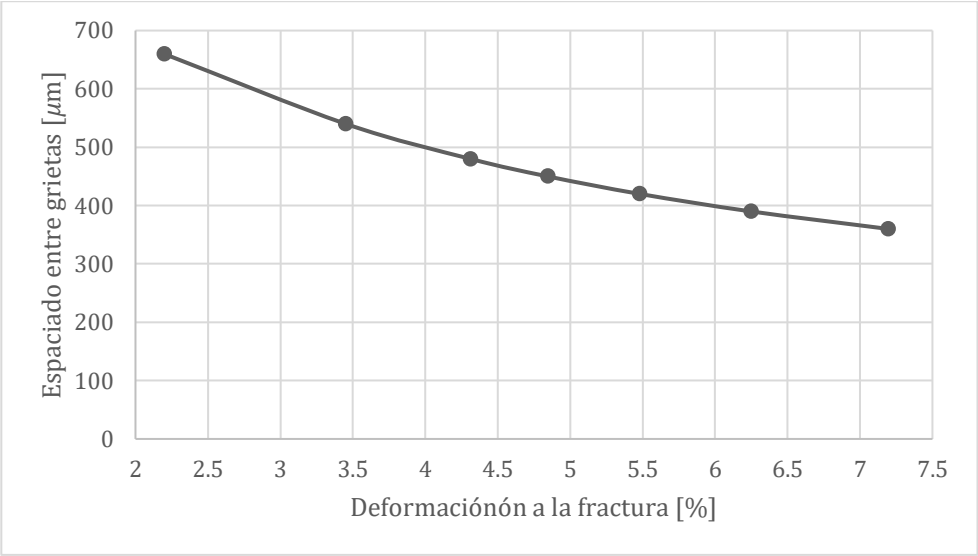


Figura 60. Espaciado entre grietas vs la deformación a la fractura

DISCUSIONES Y RECOMENDACIONES

La determinación de la resistencia a la fractura en un recubrimiento no es trivial. Esto es debido a que hay disponibles muy pocas técnicas (por ejemplo. indentación Vickers) cuya reproducibilidad es discutible. Además, no todos los recubrimientos pueden llegar a ser ensayados. En este caso, tener un modelo válido para el cálculo de KIC viene a ser una ventaja y un ahorro para los interesados.

Durante el desarrollo de este modelo se encontraron diferentes problemas y limitaciones, que se exponen a continuación:

- Para todos los modelos, la fractura en el recubrimiento solo puede ocurrir si el recubrimiento tiene solo un nodo para fracturar, esto hace que no sea posible hacer un mallado más fino en el material a fracturar.
- Para realizar una segunda fractura en el modelo de la probeta larga, a medida que se ubican grietas cercanas entre sí, el modelo tiende a no converger y no se pueden obtener resultados.
- El espaciado entre grietas no afecta a los resultados para la primera fractura; sin embargo, afecta a las siguientes fracturas en el recubrimiento lo que no se puede recrear por problemas de convergencia. Para esto se plantea el modelo de análisis de espaciamiento entre grietas, en el cual los resultados demuestran la tendencia de la curva experimental pero no corresponden a los valores numéricos del experimento.
- Para el análisis del tamaño de espesor del recubrimiento, se observa que el esfuerzo en el sustrato antes de la fractura para los diferentes espesores se mantiene en un rango entre 183.6 y 193.8 [MPa], lo cual no muestra una diferencia notable y es el factor que se toma en cuenta para la predicción de la fractura. Sin embargo, el esfuerzo en el recubrimiento y las deformaciones tanto para sustrato como recubrimiento no son

similares. Esto último se replanteará y se buscará una explicación en futuras investigaciones.

- Para los modelos de probeta corta, la segunda fractura es más difícil de modelar por problemas de convergencia del software ABAQUS. Se encuentra que el esfuerzo en el sustrato no se encuentra en el rango de la primera fractura. A su vez, se observa que la segunda fractura depende de la dimensión de la probeta y de la dimensión del material lateral sin recubrimiento, estos son criterios a considerar de la misma forma en futuros análisis.
- Para los modelos de probabilidad, se observó que se puede volver a ocasionar una fractura cuando el esfuerzo tanto en el recubrimiento como en el sustrato llega a un valor crítico. La reproducibilidad es más compleja ya que se tiene que hacer un seguimiento a medida que se carga la probeta para observar las zonas donde se acumula el esfuerzo hasta llegar a estos valores críticos.

Finalmente, el presente método tiene la limitación de ser largo y complicado, debido a los tiempos de simulación que se relacionan directamente con el equipo que se utilice, así como el alto número de pruebas que se deben realizar hasta obtener un valor aceptable, errores de convergencia provenientes de la falta de capacidad del software ABAQUS; y la necesidad de un valor de referencia obtenido en pruebas experimentales. Se propone en el futuro realizar nuevos tipos de análisis considerando que en este documento se logró representar de manera satisfactoria la primera fractura, pero no se logró representar todos los datos experimentales. Se debe también buscar que parámetros pueden hacer que todos los modelos planteados en este documento converjan, y por último incluir nuevos factores que pudieran afectar la fractura como porosidad, esfuerzos residuales y nuevos materiales.

CONCLUSIONES

El fin de este proyecto de investigación fue realizar un modelo de elementos finitos para simular fractura en un recubrimiento WC-Co12% de 200 µm, y poder obtener valores de la resistencia a la fractura del recubrimiento que permitan predecir futuros modelos con diferentes espesores de recubrimiento.

Para el primer modelo los valores de la resistencia a la fractura del recubrimiento van desde 11.17 hasta 19.33 [MPa $m^{0.5}$], para el tamaño del iniciador o grieta ubicada en la parte exterior del recubrimiento, este valor puede ir desde 1 μm hasta 3 μm. A medida que el tamaño de este iniciador es más grande el valor de la resistencia a la fractura obtenido de los modelos aumenta y se encuentra por fuera de los límites especificados en la publicación de Usmani y Sampath que exponen valores para la resistencia a la fractura entre 13 y 15.2 [MPa $m^{0.5}$] (1996).Para el análisis del tamaño del espesor del recubrimiento, se utilizan los coeficientes de resistencia a la fractura obtenidos en el primer modelo ya que se adaptan de mejor manera a la publicación antes mencionada, se muestra que a medida que el espesor se vuelve más delgado la probeta puede deformarse más hasta fracturarse, criterio que se puede tomar en cuenta al momento de realizar la deposición del recubrimiento en el termorociado.

Por último, obtener valores de resistencia a la fractura por medio de un modelo de elementos finitos viene a ser un proceso más barato, accesible y reproducible para cualquier tipo de combinación sustrato-recubrimiento; esto lo convierte en un avance significante dentro de la búsqueda de un método más general y llamativo para los interesados. Este método permitirá realizar nuevos modelos de probetas donde se usen combinaciones de nuevos materiales y tamaños de espesores para el recubrimiento, esto para obtener una combinación de materiales más resistente a la fractura y predecir su comportamiento antes de la fabricación permitiendo disminuir costos.

REFERENCIAS BIBLIOGRÁFICAS

[1] Jeandin, M., Boller, E., & Pauchet, F. (2013). Best Paper in Journal of Thermal Spray Technology: 3D Analysis in Microstructure of Thermal Spray Coatings. *Metallography, Microstructure, and Analysis*, *2*(3), 196–201. https://doi.org/10.1007/s13632-013-0077-5

[2] Khennane, A. (2013). *Finite Element Analysis Using MATLAB and Abaqus.* Boca Raton: CRC Press.

[3] Usmani, S. & Sampath, S. (1996). The Thermal Spray Laboratory Stony Brook: Effect of Carbide Grain Size on the Sliding and Abrasive Wear Behavior of Thermally Sprayed WC-Co Coatings.

[4] Vackel, A., Nakamura, T. & Sampath, S. (2015). Journal of Thermal Spray Technology: Mechanical Behavior of Spray-Coated Metallic Laminates.

[5] Griffith, A. (1920). The Phenomena of Rupture and Flow in Solids. Philosophical Transactions. Vol 221

[6] Rocha, E. & Díaz, S. (2008). Determinación de la tenacidad a la fractura mediante indentación Vickers. *Ingenierías, Vol. XI No.39,* pp. 52-58. https://www.researchgate.net/publication/28211170_Determinacion_de_la_tenacidad_a_la_fractura_mediante_indentacion_Vickers.

[7] Cuadrado, N., Casellas, D., Caro, J. & Llanes, L. (2009). Caracterización mecánica mediante la técnica de nanoindentación de partículas duras. *Anales de Mecánica de la Fractura, Vol. 1,* pp. 566-571. https://upcommons.upc.edu/bitstream/handle/2117/10489/AnalesCaractNanomecaLlanes.pdf

[8] The Thermal Spray Laboratory Stony Brook. (2017). Resultados del ensayo a tracción de una probeta con recubrimiento WC-12%Co.

[9] Pawlowski, L. (1995). *The Science and Engineering of Thermal Spray Coatings*. New York: Wiley.

[10] Papyrin, A., Kosarev, V., Klinkov, S., Alkhimov, A., & Fomin, V. (2007). *Cold Spray Technology*. Elsevier: Oxford.

ANEXO A: INSTALACIÓN DE JDK (JAVA DEVELOPMENT KIT) PARA USO DE LOS CÓDIGOS ADJUNTOS

El JDK es el ambiente para crear, compilar y ejecutar aplicaciones por medio del lenguaje Java, para correr todos los códigos adjuntos en este documento, se debe utilizar una maquina con un JDK instalado. Para la escritura de los códigos adjuntos se utilizó la versión 8 del JDK que se la puede encontrar en el link a continuación:

http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

Una vez instalado el JDK se puede realizar la verificación del mismo abriendo el CMD en Windows y ejecutando el comando “java -version”.

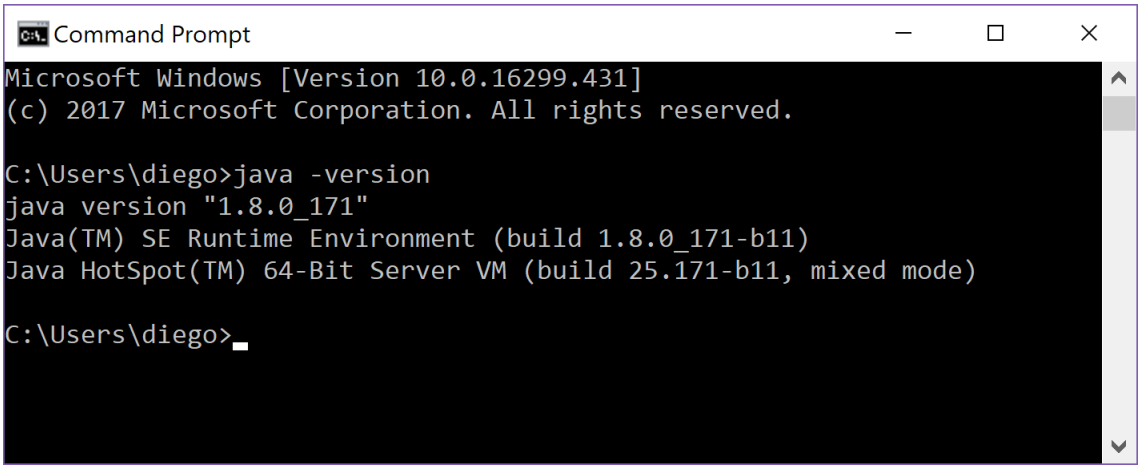


Figura 61. Verificación de la instalación del JDK

Si el CMD muestra un mensaje como el anterior, esto significa que Java está instalando en el equipo y que se puede ejecutar los códigos adjuntos en el documento.

ANEXO B: INSTALACIÓN IDE INTELLIJ IDEA PARA EJECUCIÓN DE CÓDIGOS JAVA

El IDE es una herramienta que no es indispensable para desarrollar códigos Java pero que es de bastante ayuda al momento del control de errores en el código, además de resaltar las líneas de código y usar colores para todo tipo de sentencias. Para el desarrollo de los códigos de este documento, se utilizó IntelliJ IDEA como IDE. Este es un IDE que tiene una versión libre que se puede encontrar en el link adjunto:

https://www.jetbrains.com/idea/

Una vez realizada la instalación, el uso del IDE es bastante intuitivo y fácil.

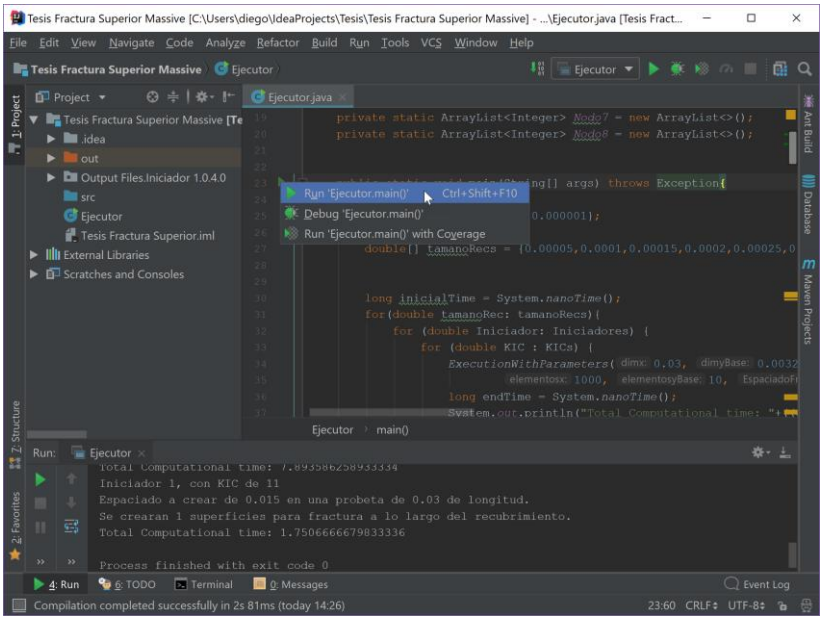


Figura 62. Interfaz IntelliJ IDEA

Los códigos que se adjuntan en el documento pueden ser copiados dentro de un archivo Java en un proyecto creado dentro de IntelliJ IDEA, cualquiera de los archivos de Java adjuntos debe copiarse dentro de una clase/archivo llamado Ejecutor que es el nombre de la clase de los códigos adjuntos. Java no entiende y lanzara un error si se copia uno de los códigos dentro de un archivo que no se llame Ejecutor.

ANEXO C: REVISIÓN DEL PATH DE ABAQUS EN WINDOWS

Todos los códigos de Java adjuntos tienen una sentencia para llamar al análisis de ABAQUS una vez que se ejecute el código. El análisis se lo realiza por medio de una extensión de ABAQUS llamada Abaqus Command que para saber si está instalada y conectada con el path de variables de Windows, se abre CMD y se ejecuta el comando “abaqus verify -std”.

Si se obtiene PASS en la verificación, el path ya está vinculado con Abaqus Command.

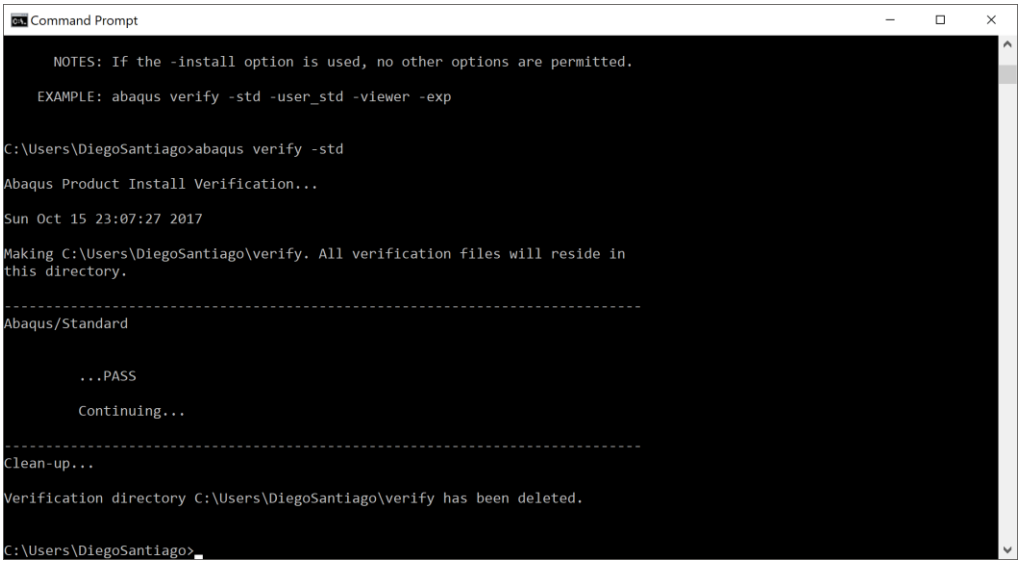


Figura 63. Verificación de Abaqus Command

ANEXO D: EXPLICACIÓN DE LAS PROPIEDADES Y DIMENSIONES DESCRITOS EN LOS CÓDIGOS JAVA PARA LOS MODELOS

Se describen 4 códigos en este documento; en la mayoría de los códigos se utilizan los mismos nombres para las variables, en la Tabla 18 se describen los nombres de las variables que se pueden utilizar.

Tabla 18. Definición de las variables de entrada del programa en Java.

Variable	Descripción	Unidades
dimx	Largo de la probeta	[m]
dimyBase	Espesor del sustrato	[m]
dimz	Ancho de la probeta	[m]
dimyRec	Espesor del recubrimiento	[m]
elementosx, elementosyBase	Número de elementos que el programa hace por cada uno de los dimx y dimyBase	
K_{IC}	Resistencia a la fractura en modo I	[MPa $m^{0.5}$]
Iniciador	Tamaño del iniciador de la fractura	[m]
EspaciadoFractura	Número de elementos en los cuales el programa ingresa propiedades de fractura y una grieta	
PorcentajeDesplazamiento	Porcentaje del total de dimx que se desplazara a la probeta	[%]

Los códigos adjuntos en los Anexos F, G y H utilizan una función llamada ExecutionWithParameters que toma como parámetros todas las variables descritas en la Tabla 19. El código adjunto en el Anexo E no tiene esta función, pero se describen todas las variables de entrada de la misma forma.

```
public static void main(String[] args) throws Exception{

    double[] Iniciadores = {00.000001,0.000002};
    double[] KICs = {3,4,5,10};

    for (double Iniciador: Iniciadores) {
        for (double KIC : KICs) {
            ExecutionWithParameters( dimx: 0.03,  dimyBase: 0.0032,  dimz: 0.0125,  dimyRec: 0.0015, Iniciador,
                    elementosx: 2000,  elementosyBase: 10,  EspaciadoFractura: 1000,  KIC: KIC * 1E6,  PorcentajeDesplazamiento: 5);
        }
    }
}
```

Figura 64. Definición de dimensiones para el modelo.

Como se muestra en la Figura 64, se puede ver el uso de dos arreglos el arreglo Iniciadores y el arreglo KICs, en estos arreglos se puede ver que se pueden ingresar más de una variable a la vez, esto de forma que el programa ejecute varios análisis. Por ejemplo: Si se ingresan dos iniciadores y cinco valores de la resistencia a la fractura, el programa ejecutara 10 análisis, de manera que probara cada uno de los K_{IC} para cada uno de los iniciadores y guardaría el análisis en carpetas separadas.

Para cambiar la ruta de las carpetas donde se ejecutarán los análisis se debe hacer un cambio en las líneas de código descritas en la Figura 65.

```
String CarpetaRoot = "C:\\Users\\diego\\Desktop\\Tesis No Drive\\Output Files";
File RootFolder = new File(CarpetaRoot);
if(!RootFolder.isDirectory())
    RootFolder.mkdir();

String CarpetaRoot1 = CarpetaRoot+"\\Dimx "+dimx+" DimyBase "+dimyBase+" Dimz "+dimz+" DimyRec "+dimyRec+" Desp "+PorcentajeDesplazamiento;
File RootFolder1 = new File(CarpetaRoot1);
if(!RootFolder1.isDirectory())
    RootFolder1.mkdir();

String CarpetaInic = CarpetaRoot1+"\\"+"Iniciador " + Integer.toString((int)(dimyInic*1E6));
File InicFolder = new File(CarpetaInic);
if(!InicFolder.isDirectory())
    InicFolder.mkdir();

String Carpeta = CarpetaInic+"\\"+"KIC "+Integer.toString((int)(KIC*1E-6));
File Folder = new File(Carpeta);
if(!Folder.isDirectory())
    Folder.mkdir();
```

Figura 65. Líneas de código que contienen el path donde se guardaran los archivos de los análisis

El programa automáticamente crea el nombre de las carpetas acorde a la dimensión x, a la dimensión y del sustrato, a la dimensión z, al tamaño del espesor del recubrimiento y al porcentaje de desplazamiento todos estos parámetros como nombre de una carpeta dentro de la ruta que se especifique en la primera línea llamada Carpeta Root. Esta dirección se debe cambiar ya que si no existe el path: "C:\\Users\\diego\\Desktop\\Tesis No Drive\\Output Files" la ejecución del programa fallara. A su vez creara carpetas para cada uno de las combinaciones iniciador-resistencia a la fractura. Los modelos para la segunda fractura a su vez adjuntan en su nombre la dimensión del material lateral sin recubrimiento. Nota: Se debe utilizar doble \\ para hacer la referencia de una nueva carpeta.

Como se mencionó al principio del documento, las propiedades mecánicas del sustrato, son propiedades elástico-plásticas mientras que las propiedades mecánicas del recubrimiento son propiedades frágiles, las propiedades como el módulo de Young y los coeficientes de poisson se pueden modificar en las líneas de código mostradas en la Figura 68.

```
double YoungModBase = 207E9;
double YoungModRecubrimiento = 560E9;
double PoissonBase = 0.3;
double PoissonRecubrimiento = 0.24;

ArrayList<KVpair<Double, Double>> propiedadesplasticas = new ArrayList<>();
propiedadesplasticas.add(new KVpair<>( kval: 175E6, eval: 0.0));
propiedadesplasticas.add(new KVpair<>( kval: 260E6, eval: 0.05));
propiedadesplasticas.add(new KVpair<>( kval: 310E6, eval: 0.1));
propiedadesplasticas.add(new KVpair<>( kval: 340E6, eval: 0.15));
propiedadesplasticas.add(new KVpair<>( kval: 375E6, eval: 0.2));
propiedadesplasticas.add(new KVpair<>( kval: 395E6, eval: 0.25));
propiedadesplasticas.add(new KVpair<>( kval: 420E6, eval: 0.3));
propiedadesplasticas.add(new KVpair<>( kval: 425E6, eval: 0.35));
```

Figura 66. Definición de módulos de Young, coeficientes de Poisson para el modelo y propiedades plásticas

El código de Java una vez compilado y sin errores de compilación crea automáticamente 6 archivos para cada análisis, se usan estos 6 archivos para distribuir mejor toda la información necesaria para el modelo.

Figura 67. Archivos creados por los códigos de Java.

Cada uno de los archivos contiene información importante para el modelo de la probeta y deben ser conservados en el mismo directorio donde se va a ejecutar el análisis de ABAQUS, el contenido de cada archivo se describe a continuación:

Fracture_properties.inp: Contiene las propiedades mecánicas tanto elásticas como plásticas de ambos materiales.

Fracture_nodset.inp: Contiene los nodos que forman parte de las superficies que se van a desplazar y dar condiciones de borde.

Fracture_nodes.inp: Contiene la declaración de las coordenadas de todos los nodos del modelo.

Fracture_elset.inp: Contiene la declaración de la relación de que elementos pertenecen a que material del modelo.

Fracture_elems.inp: Contiene la declaración de los nodos que conforman a cada elemento del modelo.

Fracture.inp: Contiene una declaración que junta a los 5 archivos anteriores, los coeficientes de resistencia a la fractura, las condiciones de borde del modelo, las propiedades para provocar la fractura, y el step que se necesita para correr el modelo. Este es el archivo que se alimenta al Abaqus Command.

Para los modelos de análisis del espaciado se utiliza el código adjunto en el anexo H, la diferencia de este código, es que se generan las superficies de fractura ingresando los números de los elementos en los que se quiere ingresar la grieta, el valor de value es utilizado para crear fracturas en el recubrimiento sin las propiedades de fractura. Por ejemplo, si el valor de value

es 2, los elementos 13500 y 500 tendrán ruptura en el recubrimiento incluso antes de iniciar el ensayo.

```
ArrayList<Integer> ElementosFracturaSup = new ArrayList<>();
ArrayList<Integer> ElementosFracturaInf = new ArrayList<>();
int value = 0;

ElementosFracturaSup.add(13500);
ElementosFracturaSup.add(13478);
ElementosFracturaSup.add(13522);
ElementosFracturaSup.add(13456);
ElementosFracturaSup.add(13544);
ElementosFracturaSup.add(13434);
ElementosFracturaSup.add(13566);

ElementosFracturaInf.add(500);
ElementosFracturaInf.add(478);
ElementosFracturaInf.add(522);
ElementosFracturaInf.add(456);
ElementosFracturaInf.add(544);
ElementosFracturaInf.add(434);
ElementosFracturaInf.add(566);
```

Figura 68. Líneas de código para generar superficies de fractura por medio del nombre del elemento

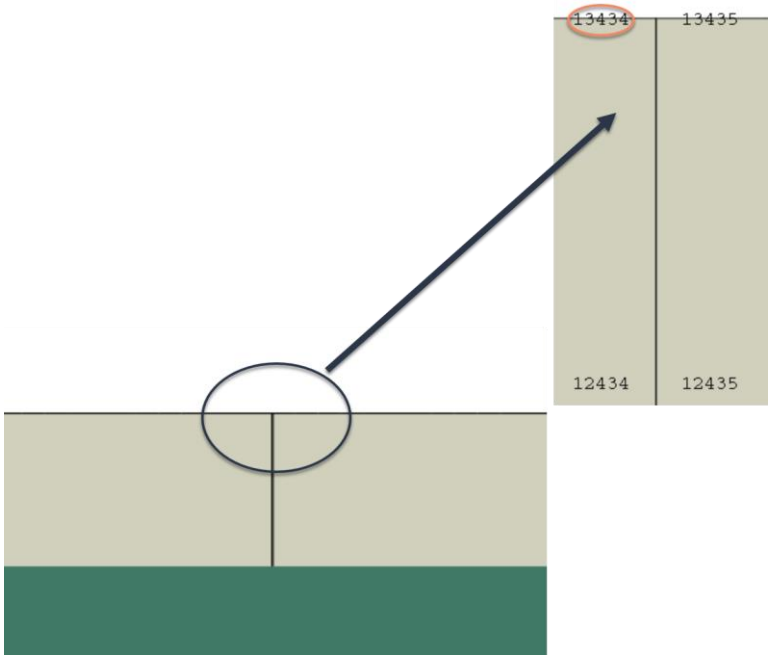


Figura 69. Nombre del elemento que se debe escribir en el código para generar la fractura

ANEXO E: CÓDIGO JAVA PARA MODELOS DE FRACTURA DESDE LA INTERFACE

Ejecutor.java

```
import java.util.ArrayList;
import java.io.*;

public class Ejecutor{
    private static ArrayList<Integer> Nodos = new ArrayList<>();
    private static ArrayList<Double> NodosX = new ArrayList<>();
    private static ArrayList<Double> NodosY = new ArrayList<>();
    private static ArrayList<Double> NodosZ = new ArrayList<>();
    private static ArrayList<Integer> Elementos = new ArrayList<>();
    private static ArrayList<Integer> ElementosBase = new ArrayList<>();
    private static ArrayList<Integer> ElementosRecubrimiento = new ArrayList<>();
    private static ArrayList<Integer> ElementosIniciadores = new ArrayList<>();
    private static ArrayList<Integer> ElementosEdgeEffect = new ArrayList<>();
    private static ArrayList<Integer> Nodo1 = new ArrayList<>();
    private static ArrayList<Integer> Nodo2 = new ArrayList<>();
    private static ArrayList<Integer> Nodo3 = new ArrayList<>();
    private static ArrayList<Integer> Nodo4 = new ArrayList<>();
    private static ArrayList<Integer> Nodo5 = new ArrayList<>();
    private static ArrayList<Integer> Nodo6 = new ArrayList<>();
    private static ArrayList<Integer> Nodo7 = new ArrayList<>();
    private static ArrayList<Integer> Nodo8 = new ArrayList<>();

    public static void main(String[] args) throws Exception{
        //Dimensiones
        //Base
        double dimx = 0.03;    //metros
        double dimyBase = 0.0032;  //metros
        double dimz = 0.0125;      //metros

        //Recubrimiento
        double dimyRec =  0.0002;   //metros
        double dimyInic = 0.00001; //metros

        //Elementos
        //Base
        int elementosx = 1000;//Funciona para mas de 20 elementos
        int elementosyBase = 10;
        int elementosz = 1; //Siempre un elemento

        //Recubrimiento
        int elementosyRec = 3;
        int elementosyInic = 1; //Siempre un elemento

        int elementosedge = 4*elementosx/10;
        int EspaciadoFractura = 500; //Por elementos

        int nodosboundaryeliminar =
(elementosyRec+elementosyInic);//(elementosyRec+2*elementosyInic);

        double YoungModBase = 207E9;
        double YoungModRecubrimiento = 560E9;
        double PoissonBase = 0.3;
        double PoissonRecubrimiento = 0.24;

        double KIC = 3E6;
        double KIIC = 4E6;
        double KIIIC = 4E6;
        double eta = 1.75;
```

```
int maximoIncremento = 200;
double TimeAnalysis = 1;
double InitialTime = 0.005;
double MaxTime = 0.01;
double MinTime = 0.00000000015;

double PorcentajeDesplazamiento = 25;

ArrayList<KVpair<Double, Double>> propiedadesplasticas = new ArrayList<>();
propiedadesplasticas.add(new KVpair<>(175E6,0.0));
propiedadesplasticas.add(new KVpair<>(260E6,0.05));
propiedadesplasticas.add(new KVpair<>(310E6,0.1));
propiedadesplasticas.add(new KVpair<>(340E6,0.15));
propiedadesplasticas.add(new KVpair<>(375E6,0.2));
propiedadesplasticas.add(new KVpair<>(395E6,0.25));
propiedadesplasticas.add(new KVpair<>(420E6,0.3));
propiedadesplasticas.add(new KVpair<>(425E6,0.35));

int elementosyTotal = (elementosyBase+2*(elementosyRec+2*elementosyInic));
int auxelemsx = elementosx-elementosedge;
int contadorelementos = elementosx*elementosyTotal*elementosz;
double AumentoElementosRec = dimyRec/elementosyRec;
double AumentoElementosBase = dimyBase/elementosyBase;
double AumentoHorizonal = dimx/elementosx;
double AumentoProfundidad = dimz/elementosz;

double GIC = KIC*KIC/(YoungModRecubrimiento);
double GIIC = KIIC*KIIC/(YoungModRecubrimiento);
double GIIIC = KIIIC*KIIIC/(YoungModRecubrimiento);

double desplazarProbeta = (PorcentajeDesplazamiento/100)*dimx/2;

//Nodos
int nodosCont = 1;
for(int z=0; z<=elementosz; z++){
    for(int y=0; y<=elementosyRec; y++){
        for(int x=0; x<=elementosx; x++){
            Nodos.add(nodosCont);
            NodosX.add(x*AumentoHorizonal);
            NodosY.add(y*AumentoElementosRec);
            NodosZ.add(z*AumentoProfundidad);
            nodosCont++;
        }
    }
    for(int y=0; y<elementosyInic; y++) {
        for (int x = 0; x <= elementosx; x++) {
            Nodos.add(nodosCont);
            NodosX.add(x * AumentoHorizonal);
            NodosY.add(dimyInic + dimyRec);
            NodosZ.add(z * AumentoProfundidad);
            nodosCont++;
        }
    }
    for(int y=0; y<elementosyInic; y++) {
        for (int x = 0; x <= elementosx; x++) {
            Nodos.add(nodosCont);
            NodosX.add(x * AumentoHorizonal);
            NodosY.add(2*dimyInic + dimyRec);
            NodosZ.add(z * AumentoProfundidad);
            nodosCont++;
        }
    }
    for(int y=0; y<elementosyBase; y++){
        for(int x=0; x<=elementosx; x++){
            Nodos.add(nodosCont);
            NodosX.add(x*AumentoHorizonal);

NodosY.add(2*dimyInic+dimyRec+AumentoElementosBase+y*AumentoElementosBase);
```

```
                    NodosZ.add(z*AumentoProfundidad);
                    nodosCont++;
                }
             }
            for(int y=0; y<elementosyInic; y++){
                for(int x=0; x<=elementosx; x++){
                    Nodos.add(nodosCont);
                    NodosY.add(3*dimyInic+dimyRec+dimyBase);
                    NodosX.add(x*AumentoHorizonal);
                    NodosZ.add(z*AumentoProfundidad);
                    nodosCont++;
                }
             }
             for(int y=0; y<elementosyInic; y++){
                for(int x=0; x<=elementosx; x++){
                    Nodos.add(nodosCont);
                    NodosX.add(x*AumentoHorizonal);
                    NodosY.add(4*dimyInic+dimyRec+dimyBase);
                    NodosZ.add(z*AumentoProfundidad);
                    nodosCont++;
                }
             }
              for(int y=0; y<elementosyRec; y++){
                for(int x=0; x<=elementosx; x++){
                    Nodos.add(nodosCont);
                    NodosX.add(x*AumentoHorizonal);

NodosY.add(4*dimyInic+dimyRec+dimyBase+AumentoElementosRec+y*AumentoElementosRec);
                    NodosZ.add(z*AumentoProfundidad);
                    nodosCont++;
                }
             }
        }

        //Elementos
        int elementoscontar = 1;
        for(int z=0; z<elementosz; z++){
            for(int y=0; y<elementosyRec; y++){
                for(int x=0; x<elementosx; x++){
                    ElementosRecubrimiento.add(elementoscontar);
                    Elementos.add(elementoscontar);
                    elementoscontar++;
                }
            }
            for(int y=0; y<elementosyInic; y++){
                for(int x=0; x<elementosx; x++){
                    ElementosRecubrimiento.add(elementoscontar);
                    ElementosIniciadores.add(elementoscontar);
                    Elementos.add(elementoscontar);
                    elementoscontar++;
                }
            }
            for(int y=0; y<elementosyInic; y++){
                for(int x=0; x<elementosx; x++){
                    Elementos.add(elementoscontar);
                    ElementosIniciadores.add(elementoscontar);
                    ElementosBase.add(elementoscontar);
                    elementoscontar++;
                }
            }
            for(int y=0; y<elementosyBase; y++){
                for(int x=0; x<elementosx; x++){
                    Elementos.add(elementoscontar);
                    ElementosBase.add(elementoscontar);
                    elementoscontar++;
                }
            }
            for(int y=0; y<elementosyInic; y++){
```

```
        for(int x=0; x<elementosx; x++){
            ElementosIniciadores.add(elementoscontar);
            Elementos.add(elementoscontar);
            ElementosBase.add(elementoscontar);
            elementoscontar++;
        }
    }
    for(int y=0; y<elementosyInic; y++){
        for(int x=0; x<elementosx; x++){
            ElementosIniciadores.add(elementoscontar);
            ElementosRecubrimiento.add(elementoscontar);
            Elementos.add(elementoscontar);
            elementoscontar++;
        }
    }
    for(int y=0; y<elementosyRec; y++){
        for(int x=0; x<elementosx; x++){
            Elementos.add(elementoscontar);
            ElementosRecubrimiento.add(elementoscontar);
            elementoscontar++;
        }
    }
}
for(int i=0; i<Elementos.size();i++){
    if(i<elementosedge) {
        ElementosEdgeEffect.add(Elementos.get(i));
    }
    if(i>=auxelemsx){
        ElementosEdgeEffect.add(Elementos.get(i));
    }
    if((i+1)%elementosx==0) {
        auxelemsx= auxelemsx + elementosx;
        elementosedge = elementosedge + elementosx;
    }
}

int nodosporcara = (elementosx+1)*(elementosyTotal+1);
nodosCont=1;
for(int z=0;z<elementosz;z++){
    for(int y=0; y<elementosyTotal; y++){
        for(int x=0; x<elementosx; x++){
            Nodo1.add(nodosCont);
            Nodo2.add(nodosCont+1);
            Nodo3.add(nodosCont+elementosx+2);
            Nodo4.add(nodosCont+elementosx+1);
            Nodo5.add(nodosCont+nodosporcara);
            Nodo6.add(nodosCont+1+nodosporcara);
            Nodo7.add(nodosCont+elementosx+2+nodosporcara);
            Nodo8.add(nodosCont+elementosx+1+nodosporcara);
            nodosCont++;
        }
        nodosCont++;
    }
}

ArrayList<Integer> BoundaryNodosIzquierda = new ArrayList<>();
ArrayList<Integer> BoundaryNodosDerecha = new ArrayList<>();

for(int i=0; i<Nodos.size();i++) {
    if (NodosX.get(i) == 0) {
        BoundaryNodosIzquierda.add(Nodos.get(i));
    }
    if (NodosX.get(i) == dimx) {
        BoundaryNodosDerecha.add(Nodos.get(i));
    }
}

for(int i=0; i<nodosboundaryeliminar;i++){
```

```
            int sizetemp = BoundaryNodosDerecha.size()/2;
            BoundaryNodosIzquierda.remove(sizetemp-1);
            BoundaryNodosDerecha.remove(sizetemp-1);
            BoundaryNodosIzquierda.remove(sizetemp-1);
            BoundaryNodosDerecha.remove(sizetemp-1);
            BoundaryNodosIzquierda.remove(0);
            BoundaryNodosDerecha.remove(0);
            BoundaryNodosIzquierda.remove(BoundaryNodosIzquierda.size()-1);
            BoundaryNodosDerecha.remove(BoundaryNodosDerecha.size()-1);
        }

        //Generacion de superficies para fractura
        int numeroDeFracturas = (elementosx/EspaciadoFractura)-1;

        System.out.println("Espaciado a crear de
"+EspaciadoFractura*AumentoHorizonal+" en una probeta de "+dimx+" de longitud.");
        System.out.println("Se crearan "+numeroDeFracturas+" superficies para
fractura a lo largo del recubrimiento.");


if((double)elementosx/(double)EspaciadoFractura%(double)(elementosx/EspaciadoFractu
ra)>0)
            System.out.println("Advertencia: No hay simetria en el modelo");

        //Generacion del crack de fractura
        for(int i=0; i<ElementosIniciadores.size()/4;i++) {
            int division = ElementosIniciadores.size()/4;
            if((i+1)%EspaciadoFractura==0&&(i+1)%division!=0){
                int indexinf = Elementos.indexOf(ElementosIniciadores.get(i));
                int indexsup = indexinf+elementosx;
                int indexinf2 =
Elementos.indexOf(ElementosIniciadores.get(i+2*elementosx));
                int indexsup2 = indexinf2+elementosx;
                int nuevoindicedenodo = DuplicarNodo(Nodo3.get(indexinf));
                int nuevoindicedenodo2 = DuplicarNodo(Nodo3.get(indexinf2));

                Nodo3.remove(indexinf);
                Nodo3.add(indexinf,nuevoindicedenodo);
                Nodo2.remove(indexsup);
                Nodo2.add(indexsup,nuevoindicedenodo);

                Nodo3.remove(indexinf2);
                Nodo3.add(indexinf2,nuevoindicedenodo2);
                Nodo2.remove(indexsup2);
                Nodo2.add(indexsup2,nuevoindicedenodo2);

                nuevoindicedenodo = DuplicarNodo(Nodo7.get(indexinf));
                nuevoindicedenodo2 = DuplicarNodo(Nodo7.get(indexinf2));

                Nodo7.remove(indexinf);
                Nodo7.add(indexinf,nuevoindicedenodo);
                Nodo6.remove(indexsup);
                Nodo6.add(indexsup,nuevoindicedenodo);

                Nodo7.remove(indexinf2);
                Nodo7.add(indexinf2,nuevoindicedenodo2);
                Nodo6.remove(indexsup2);
                Nodo6.add(indexsup2,nuevoindicedenodo2);
            }
        }

        ArrayList<ArrayList<Integer>> elementostop = new ArrayList<>();
        ArrayList<ArrayList<Integer>> elementosbottom = new ArrayList<>();
        ArrayList<ArrayList<Integer>> bnodesfractura = new ArrayList<>();

        int contador = 0;

        //Elementos inferiores
```

```
        for(int i=0; i<ElementosRecubrimiento.size()/2;i++){
            int division =
ElementosRecubrimiento.size()/(2*(elementosyRec+elementosyInic));

            if((i+1)%division==0)
                contador=0;

            if((i+1)%EspaciadoFractura==0&&i<(ElementosRecubrimiento.size()/2-
elementosx)&&(i+1)%division!=0){
                if((contador+1)>elementostop.size())
                    elementostop.add(new ArrayList<>());
                if((contador+1)>elementosbottom.size())
                    elementosbottom.add(new ArrayList<>());
                if((contador+1)>bnodesfractura.size())
                    bnodesfractura.add(new ArrayList<>());


                elementostop.get(contador).add(ElementosRecubrimiento.get(i));
                elementosbottom.get(contador).add(ElementosRecubrimiento.get(i+1));

                int index = Elementos.indexOf(ElementosRecubrimiento.get(i));

                bnodesfractura.get(contador).add(Nodo2.get(index));
                bnodesfractura.get(contador).add(Nodo6.get(index));

                int nuevoindicedenodo = DuplicarNodo(Nodo2.get(index));
                int nuevoindicedenodo2 = DuplicarNodo(Nodo6.get(index));

                if(i>elementosx){
                    Nodo3.remove(index-elementosx);
                    Nodo3.add(index-elementosx,nuevoindicedenodo);
                    Nodo7.remove(index-elementosx);
                    Nodo7.add(index-elementosx,nuevoindicedenodo2);
                }

                Nodo2.remove(index);
                Nodo2.add(index,nuevoindicedenodo);
                Nodo6.remove(index);
                Nodo6.add(index,nuevoindicedenodo2);

                bnodesfractura.get(contador).add(Nodo2.get(index));
                bnodesfractura.get(contador).add(Nodo6.get(index));

                contador++;
            }

            else if((i+1)%EspaciadoFractura==0&&(i+1)%division!=0){
                if((contador+1)>bnodesfractura.size())
                    bnodesfractura.add(new ArrayList<>());

                int index = Elementos.indexOf(ElementosRecubrimiento.get(i));

                bnodesfractura.get(contador).add(Nodo2.get(index));
                bnodesfractura.get(contador).add(Nodo6.get(index));

                int nuevoindicedenodo = DuplicarNodo(Nodo2.get(index));
                int nuevoindicedenodo2 = DuplicarNodo(Nodo6.get(index));

                Nodo3.remove(index-elementosx);
                Nodo3.add(index-elementosx,nuevoindicedenodo);
                Nodo7.remove(index-elementosx);
                Nodo7.add(index-elementosx,nuevoindicedenodo2);

                Nodo2.remove(index);
                Nodo2.add(index,nuevoindicedenodo);
                Nodo6.remove(index);
                Nodo6.add(index,nuevoindicedenodo2);
```

```
                bnodesfractura.get(contador).add(Nodo2.get(index));
                bnodesfractura.get(contador).add(Nodo6.get(index));

                contador++;
            }
        }

        int contadoraux = bnodesfractura.size();
        contador = contadoraux;

        //Elementos superiores
        for(int i=ElementosRecubrimiento.size()/2;
i<ElementosRecubrimiento.size();i++) {
            int division = ElementosRecubrimiento.size() / (2 * (elementosyRec +
elementosyInic));

            if ((i + 1) % division == 0)
                contador = contadoraux;

            if ((i + 1) % EspaciadoFractura == 0 && (i + 1) % division != 0 && (i +
1) < (ElementosRecubrimiento.size() / 2 + elementosx)) {
                if ((contador + 1) > bnodesfractura.size())
                    bnodesfractura.add(new ArrayList<>());

                int index = Elementos.indexOf(ElementosRecubrimiento.get(i));

                bnodesfractura.get(contador).add(Nodo3.get(index));
                bnodesfractura.get(contador).add(Nodo7.get(index));

                int nuevoindicedenodo = DuplicarNodo(Nodo3.get(index));
                int nuevoindicedenodo2 = DuplicarNodo(Nodo7.get(index));

                Nodo3.remove(index);
                Nodo3.add(index, nuevoindicedenodo);
                Nodo7.remove(index);
                Nodo7.add(index, nuevoindicedenodo2);

                bnodesfractura.get(contador).add(Nodo3.get(index));
                bnodesfractura.get(contador).add(Nodo7.get(index));

                contador++;
            }
            else if ((i + 1) % EspaciadoFractura == 0 && (i + 1) % division != 0) {
                if ((contador + 1) > elementostop.size())
                    elementostop.add(new ArrayList<>());
                if ((contador + 1) > elementosbottom.size())
                    elementosbottom.add(new ArrayList<>());
                if ((contador + 1) > bnodesfractura.size())
                    bnodesfractura.add(new ArrayList<>());

                elementostop.get(contador).add(ElementosRecubrimiento.get(i));
                elementosbottom.get(contador).add(ElementosRecubrimiento.get(i +
1));

                int index = Elementos.indexOf(ElementosRecubrimiento.get(i));

                bnodesfractura.get(contador).add(Nodo3.get(index));
                bnodesfractura.get(contador).add(Nodo7.get(index));

                int nuevoindicedenodo = DuplicarNodo(Nodo3.get(index));
                int nuevoindicedenodo2 = DuplicarNodo(Nodo7.get(index));

                Nodo2.remove(index);
                Nodo2.add(index, Nodo3.get(index-elementosx));
                Nodo6.remove(index);
                Nodo6.add(index, Nodo7.get(index-elementosx));

                Nodo3.remove(index);
```

```
                Nodo3.add(index, nuevoindicedenodo);
                Nodo7.remove(index);
                Nodo7.add(index, nuevoindicedenodo2);

                bnodesfractura.get(contador).add(Nodo3.get(index));
                bnodesfractura.get(contador).add(Nodo7.get(index));

                contador++;
            }
        }

        String CarpetaRoot = "C:\\Users\\diego\\Desktop\\Tesis No Drive\\Output
Files\\Inferior";
        File RootFolder = new File(CarpetaRoot);
        if(!RootFolder.isDirectory())
            RootFolder.mkdir();

        String CarpetaRoot1 = CarpetaRoot+"\\Dimx "+dimx+" DimyBase "+dimyBase+"
Dimz "+dimz+" DimyRec "+dimyRec+" Desp "+PorcentajeDesplazamiento;
        File RootFolder1 = new File(CarpetaRoot1);
        if(!RootFolder1.isDirectory())
            RootFolder1.mkdir();

        String CarpetaInic = CarpetaRoot1+"\\"+"Iniciador " +
Integer.toString((int)(dimyInic*1E6));
        File InicFolder = new File(CarpetaInic);
        if(!InicFolder.isDirectory())
            InicFolder.mkdir();

        String Carpeta = CarpetaInic+"\\"+"KIC "+Integer.toString((int)(KIC*1E-6));
        File Folder = new File(Carpeta);
        if(!Folder.isDirectory())
            Folder.mkdir();

        cleanDirectoryBefore(Folder);

        //Imprimiendo los archivos

        File nodosFile = new File(Carpeta+"\\"+"Fracture_nodes.inp");
        PrintWriter nodosPrintWriter = new PrintWriter(nodosFile);
        PrintHeading(nodosPrintWriter);
        nodosPrintWriter.println("*Node");
        for(int i=0; i<Nodos.size();i++){
            nodosPrintWriter.println(Nodos.get(i)+", "+NodosX.get(i)+",
"+NodosY.get(i)+", "+NodosZ.get(i));
        }
        PrintFooting(nodosPrintWriter);

        File elemsFile = new File(Carpeta+"\\"+"Fracture_elems.inp");
        PrintWriter elemsPrintWriter = new PrintWriter(elemsFile);
        PrintHeading(elemsPrintWriter);
        elemsPrintWriter.println("*Element, type=C3D8");
        for(int i=0; i<Elementos.size();i++){
            elemsPrintWriter.println(Elementos.get(i)+", "+Nodo1.get(i)+",
"+Nodo2.get(i)+", "+Nodo3.get(i)+", "+Nodo4.get(i)+", "+Nodo5.get(i)+",
"+Nodo6.get(i)+", "+Nodo7.get(i)+", "+Nodo8.get(i));
        }
        PrintFooting(elemsPrintWriter);

        File nodesetsFile = new File(Carpeta+"\\"+"Fracture_nodesets.inp");
        PrintWriter nodesetsPrintWriter = new PrintWriter(nodesetsFile);
        PrintHeading(nodesetsPrintWriter);

        nodesetsPrintWriter.println("*NSET, NSET=NodosIzq");
        for(int i=0; i<BoundaryNodosIzquierda.size();i++){
            nodesetsPrintWriter.print(BoundaryNodosIzquierda.get(i)+", ");
            if((i+1)%8==0)
                nodesetsPrintWriter.println();
```

```
        }
        if(BoundaryNodosIzquierda.size()%8!=0)
            nodesetsPrintWriter.println();

        nodesetsPrintWriter.println("*NSET, NSET=NodosDer");
        for(int i=0; i<BoundaryNodosDerecha.size();i++){
            nodesetsPrintWriter.print(BoundaryNodosDerecha.get(i)+", ");
            if((i+1)%8==0)
                nodesetsPrintWriter.println();
        }
        if(BoundaryNodosDerecha.size()%8!=0)
            nodesetsPrintWriter.println();

        PrintFooting(nodesetsPrintWriter);

        File elsetFile = new File(Carpeta+"\\"+"Fracture_elset.inp");
        PrintWriter elsetPrintWriter = new PrintWriter(elsetFile);
        PrintHeading(elsetPrintWriter);
        elsetPrintWriter.println("*Elset, elset=cube, generate\n1, "+
contadorelementos +", 1");

        elsetPrintWriter.println("*Elset, elset=Base_set");
        for(int i=0; i<ElementosBase.size();i++){
            elsetPrintWriter.print(ElementosBase.get(i)+", ");
            if((i+1)%8==0)
                elsetPrintWriter.println();
        }
        if(ElementosBase.size()%8!=0)
            elsetPrintWriter.println();

        elsetPrintWriter.println("*Elset, elset=Recubrimiento_set");
        for(int i=0; i<ElementosRecubrimiento.size();i++){
            elsetPrintWriter.print(ElementosRecubrimiento.get(i)+", ");
            if((i+1)%8==0)
                elsetPrintWriter.println();
        }
        if(ElementosRecubrimiento.size()%8!=0)
            elsetPrintWriter.println();

        elsetPrintWriter.println("*Elset, elset=Iniciadores_set");
        for(int i=0; i<ElementosIniciadores.size();i++){
            elsetPrintWriter.print(ElementosIniciadores.get(i)+", ");
            if((i+1)%8==0)
                elsetPrintWriter.println();
        }
        if(ElementosIniciadores.size()%8!=0)
            elsetPrintWriter.println();

        elsetPrintWriter.println("*Elset, elset=EdgeEffect_set");
        for(int i=0; i<ElementosEdgeEffect.size();i++){
            elsetPrintWriter.print(ElementosEdgeEffect.get(i)+", ");
            if((i+1)%8==0)
                elsetPrintWriter.println();
        }
        if(ElementosEdgeEffect.size()%8!=0)
            elsetPrintWriter.println();

        PrintFooting(elsetPrintWriter);

        File propertiesFile = new File(Carpeta+"\\"+"Fracture_properties.inp");
        PrintWriter propertiesPrintWriter = new PrintWriter(propertiesFile);
        PrintHeading(propertiesPrintWriter);
        propertiesPrintWriter.println("*Solid Section, elset=Base_set,
material=Base\n1.\n*Solid Section, elset=Recubrimiento_set,
material=Recubrimiento\n1.");
        propertiesPrintWriter.println("*Material, name=Base\n" + "*ELASTIC\n" +
YoungModBase+", "+PoissonBase+"\n" + "*PLASTIC");
```

```
        for (KVpair<Double, Double> propiedadesplastica : propiedadesplasticas) {
            propertiesPrintWriter.println(propiedadesplastica.key() + ", " +
propiedadesplastica.value());
        }

        propertiesPrintWriter.println("*Material, name=Recubrimiento\n" +
"*ELASTIC\n" + YoungModRecubrimiento+", "+PoissonRecubrimiento);
        PrintFooting(propertiesPrintWriter);

        File fractureFile = new File(Carpeta+"\\"+"Fracture.inp");
        PrintWriter fracturePrintWriter = new PrintWriter(fractureFile);
        PrintHeading(fracturePrintWriter);
        fracturePrintWriter.println("*Heading\nFracture");
        fracturePrintWriter.println("** ----------------------------------------
----------------------");
        fracturePrintWriter.println("*parameter\n" +
                "** Fracture toughness:\n" +
                " GIc   = "+GIC+"\n" +
                " GIIc  = "+GIIC+"\n" +
                " GIIIc = "+GIIIC+"\n" +
                "** B-K parameter:\n" +
                " eta="+eta+"\n" +
                "***\n" +
                "*Preprint, echo = NO, model = NO, history = NO, contact = NO");

        fracturePrintWriter.println(
                "*Include, Input =Fracture_nodes.inp\n" +
                "*Include, Input =Fracture_elems.inp\n" +
                "*Include, Input =Fracture_elset.inp\n" +
                "*Include, Input =Fracture_properties.inp\n"+
                "*Include, Input =Fracture_nodesets.inp");

        fracturePrintWriter.println("** BNODES DECLARATION\n**");

        for(int i=0; i<bnodesfractura.size();i++){
            fracturePrintWriter.println("*NSET,NSET=BNODES"+(i+1));
            for(int j=0; j<bnodesfractura.get(i).size();j++){
                fracturePrintWriter.print(bnodesfractura.get(i).get(j)+", ");
                if((j+1)%8==0)
                    fracturePrintWriter.println();
            }
            if(bnodesfractura.get(i).size()%8!=0)
                fracturePrintWriter.println();
        }

        fracturePrintWriter.println("**ELSET DECLARATION\n**");

        for(int i=0; i<elementosbottom.size();i++){
            fracturePrintWriter.println("*Elset, elset=BOT"+(i+1));
            for(int j=0; j<elementosbottom.get(i).size();j++){
                fracturePrintWriter.print(elementosbottom.get(i).get(j)+", ");
                if((j+1)%8==0)
                    fracturePrintWriter.println();
            }
            if(elementosbottom.get(i).size()%8!=0)
                fracturePrintWriter.println();
        }

        for(int i=0; i<elementostop.size();i++){
            fracturePrintWriter.println("*Elset, elset=TOP"+(i+1));
            for(int j=0; j<elementostop.get(i).size();j++){
                fracturePrintWriter.print(elementostop.get(i).get(j)+", ");
                if((j+1)%8==0)
                    fracturePrintWriter.println();
            }
            if(elementostop.get(i).size()%8!=0)
                fracturePrintWriter.println();
        }
```

```
        for(int i=0; i<elementosbottom.size();i++){
            fracturePrintWriter.println("*SURFACE, NAME=BOT"+(i+1));
            fracturePrintWriter.println("BOT"+(i+1)+", S6");
        }

        for(int i=0; i<elementostop.size();i++){
            fracturePrintWriter.println("*SURFACE, NAME=TOP"+(i+1));
            fracturePrintWriter.println("TOP"+(i+1)+", S4");
        }

        for(int i=0; i<bnodesfractura.size();i++){
            fracturePrintWriter.println("*CONTACT PAIR, INTERACTION=FRACT"+(i+1)+",
ADJUST=BNODES"+(i+1)+"\n"+"BOT"+(i+1)+", TOP"+(i+1));
            fracturePrintWriter.println("*SURFACE INTERACTION,
NAME=FRACT"+(i+1)+"\n"+"1.0");
        }

        fracturePrintWriter.println("*INITIAL CONDITIONS,TYPE =CONTACT");

        for(int i=0; i<bnodesfractura.size();i++){
            fracturePrintWriter.println("BOT"+(i+1)+", TOP"+(i+1)+",
BNODES"+(i+1));
        }

        fracturePrintWriter.println("**\n" +
                "*STEP,NLGEOM, INC="+maximoIncremento+",convert sdi=no\n" +
                "*STATIC\n" +
                InitialTime+", "+TimeAnalysis+", "+MinTime+", "+MaxTime+"\n"+
                "*CONTROLS,PARAMETERS=TIME INCREMENTATION\n" +
                " , , , , , , 10000");

        for(int i=0; i<bnodesfractura.size();i++){
            fracturePrintWriter.println("*CONTACT
PRINT\n"+"*DEBOND,SLAVE=BOT"+(i+1)+",MASTER=TOP"+(i+1)+",FREQ=1\n"
                    +"*FRACTURE CRITERION,TYPE=VCCT,MIXED MODE BEHAVIOR=BK\n"+
                    "<GIc>,<GIIc>,<GIIIc>,<eta>");
        }

        fracturePrintWriter.println("*BOUNDARY\n" +
                "NodosDer, 1,1,"+desplazarProbeta+"\n" +
                "NodosIzq, 1,1,-"+desplazarProbeta+"\n" +
                "*NODE PRINT\n" +
                "RF,\n" +
                "*EL FILE\n" +
                "S, E\n" +
                "*NODE FILE\n" +
                "U, RF\n" +
                "*END STEP");

        PrintFooting(fracturePrintWriter);

        Process process = Runtime.getRuntime().exec("cmd /c start /wait cmd.exe /K
\"cd " + Carpeta + " && abaqus analysis job=Fracture int && exit");

        process.waitFor();
        process.destroy();
        cleanDirectoryAfter(Folder);
    }

    private static void cleanDirectoryAfter(File dir){
        for (File file : dir.listFiles()) {
            if (!file.getName().equals("Fracture.odb")){
                file.delete();
            }
        }
    }
```

```
    private static void cleanDirectoryBefore(File dir) {
        for (File : dir.listFiles()) {
            file.delete();
        }
    }
    private static void PrintHeading(PrintWriter write){
        write.print("** Generated by : Diego S. Morales Arcos\n** ----------------
-----------------------------------------------\n**\n");
    }
    private static void PrintFooting(PrintWriter write){
        write.print("**\n** ------------------------------------------------------
---------\n**");
        write.close();
    }

    private static Integer DuplicarNodo(Integer NodoADuplicar){
        int index = Nodos.indexOf(NodoADuplicar);
        Nodos.add(Nodos.get(Nodos.size()-1)+1);
        NodosX.add(NodosX.get(index));
        NodosY.add(NodosY.get(index));
        NodosZ.add(NodosZ.get(index));
        return Nodos.get(Nodos.size()-1);
    }

    static class KVpair<Key, E> {
         private Key k;
         private E e;

        KVpair()
        { k = null; e = null; }
        KVpair(Key kval, E eval)
        { k = kval; e = eval; }

        public Key key() { return k; }
        public E value() { return e; }
    }

}
```

ANEXO F: CÓDIGO JAVA PARA MODELOS DE LA PRIMERA FRACTURA

Ejecutor.java

```
import java.util.ArrayList;
import java.io.*;
public class Ejecutor {
    private static ArrayList<Integer> Nodos = new ArrayList<>();
    private static ArrayList<Double> NodosX = new ArrayList<>();
    private static ArrayList<Double> NodosY = new ArrayList<>();
    private static ArrayList<Double> NodosZ = new ArrayList<>();
    private static ArrayList<Integer> Elementos = new ArrayList<>();
    private static ArrayList<Integer> ElementosBase = new ArrayList<>();
    private static ArrayList<Integer> ElementosRecubrimiento = new ArrayList<>();
    private static ArrayList<Integer> ElementosIniciadores = new ArrayList<>();
    private static ArrayList<Integer> ElementosEdgeEffect = new ArrayList<>();
    private static ArrayList<Integer> Nodo1 = new ArrayList<>();
    private static ArrayList<Integer> Nodo2 = new ArrayList<>();
    private static ArrayList<Integer> Nodo3 = new ArrayList<>();
    private static ArrayList<Integer> Nodo4 = new ArrayList<>();
    private static ArrayList<Integer> Nodo5 = new ArrayList<>();
    private static ArrayList<Integer> Nodo6 = new ArrayList<>();
    private static ArrayList<Integer> Nodo7 = new ArrayList<>();
    private static ArrayList<Integer> Nodo8 = new ArrayList<>();


    public static void main(String[] args) throws Exception{

        double[] Iniciadores =
{0.000001,0.000002,0.000003,0.000004,0.000005,0.000006,0.000007,0.000008,0.000009,0
.000010};
        double[] KICs = {3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20};

        for (double Iniciador: Iniciadores) {
            for (double KIC : KICs) {
                ExecutionWithParameters(0.03, 0.0032, 0.0125, 0.0015, Iniciador,
                        2000, 10, 1000, KIC * 1E6, 5);
            }
        }
    }

    private static void ExecutionWithParameters (double dimx, double dimyBase,
double dimz, double dimyRec, double dimyInic, int elementosx,
                                        int elementosyBase, int EspaciadoFractura,
double KIC, double PorcentajeDesplazamiento)throws Exception{
        //Elementos
        //Base
        int elementosz = 1; //Siempre un elemento

        //Recubrimiento
        int elementosyRec = 1;//Siempre un elemento
        int elementosyInic = 1; //Siempre un elemento

        int elementosedge = elementosx*4/10;

        int nodosboundaryeliminar =
elementosyRec+elementosyInic;//(elementosyRec+2*elementosyInic);(elementosyRec+elem
entosyInic);

        double YoungModBase = 207E9;
        double YoungModRecubrimiento = 560E9;
        double PoissonBase = 0.3;
        double PoissonRecubrimiento = 0.24;
```

```
double KIIC = 4E6;
double KIIIC = 4E6;
double eta = 1.75;

int maximoIncremento = 200;
double TimeAnalysis = 1;
double InitialTime = 0.005;
double MaxTime = 0.01;
double MinTime = 0.00000000015;

ArrayList<KVpair<Double, Double>> propiedadesplasticas = new ArrayList<>();
propiedadesplasticas.add(new KVpair<>(175E6,0.0));
propiedadesplasticas.add(new KVpair<>(260E6,0.05));
propiedadesplasticas.add(new KVpair<>(310E6,0.1));
propiedadesplasticas.add(new KVpair<>(340E6,0.15));
propiedadesplasticas.add(new KVpair<>(375E6,0.2));
propiedadesplasticas.add(new KVpair<>(395E6,0.25));
propiedadesplasticas.add(new KVpair<>(420E6,0.3));
propiedadesplasticas.add(new KVpair<>(425E6,0.35));

int elementosyTotal = (elementosyBase+2*(elementosyRec+elementosyInic));
int auxelemsx = elementosx-elementosedge;
int contadorelementos = elementosx*elementosyTotal*elementosz;
double AumentoElementosRec = dimyRec/elementosyRec;
double AumentoElementosBase = dimyBase/elementosyBase;
double AumentoElementosInic = dimyInic/elementosyInic;
double AumentoHorizonal = dimx/elementosx;
double AumentoProfundidad = dimz/elementosz;

double GIC = KIC*KIC/(YoungModRecubrimiento);
double GIIC = KIIC*KIIC/(YoungModRecubrimiento);
double GIIIC = KIIIC*KIIIC/(YoungModRecubrimiento);

double desplazarProbeta = (PorcentajeDesplazamiento/100)*dimx/2;

Nodos.clear();
NodosX.clear();
NodosY.clear();
NodosZ.clear();
Elementos.clear();
ElementosBase.clear();
ElementosRecubrimiento.clear();
ElementosIniciadores.clear();
ElementosEdgeEffect.clear();
Nodo1.clear();
Nodo2.clear();
Nodo3.clear();
Nodo4.clear();
Nodo5.clear();
Nodo6.clear();
Nodo7.clear();
Nodo8.clear();

//Nodos
int nodosCont = 1;
for(int z=0; z<=elementosz; z++){
    for(int y=0; y<=elementosyInic; y++){
        for(int x=0; x<=elementosx; x++){
            Nodos.add(nodosCont);
            NodosX.add(x*AumentoHorizonal);
            NodosY.add(y*AumentoElementosInic);
            NodosZ.add(z*AumentoProfundidad);
            nodosCont++;
        }
    }
    for(int y=0; y<elementosyRec; y++) {
        for (int x = 0; x <= elementosx; x++) {
```

```
                    Nodos.add(nodosCont);
                    NodosX.add(x * AumentoHorizonal);
                    NodosY.add(dimyInic + (y+1)*AumentoElementosRec);
                    NodosZ.add(z * AumentoProfundidad);
                    nodosCont++;
                }
            }
            for(int y=0; y<elementosyBase; y++){
                for(int x=0; x<=elementosx; x++){
                    Nodos.add(nodosCont);
                    NodosY.add(dimyInic+dimyRec+(y+1)*AumentoElementosBase);
                    NodosX.add(x*AumentoHorizonal);
                    NodosZ.add(z*AumentoProfundidad);
                    nodosCont++;
                }
            }
            for(int y=0; y<elementosyRec; y++){
                for(int x=0; x<=elementosx; x++){
                    Nodos.add(nodosCont);
                    NodosZ.add(z*AumentoProfundidad);
                    NodosX.add(x*AumentoHorizonal);

NodosY.add(dimyInic+dimyRec+dimyBase+(y+1)*AumentoElementosRec);
                    nodosCont++;
                }
            }
            for(int y=0; y<elementosyInic; y++){
                for(int x=0; x<=elementosx; x++){
                    Nodos.add(nodosCont);
                    NodosX.add(x*AumentoHorizonal);

NodosY.add(dimyInic+2*dimyRec+dimyBase+(y+1)*AumentoElementosInic);
                    NodosZ.add(z*AumentoProfundidad);
                    nodosCont++;
                }
            }
        }

        //Elementos
        int elementoscontar = 1;
        for(int z=0; z<elementosz; z++){
            for(int y=0; y<elementosyInic; y++){
                for(int x=0; x<elementosx; x++){
                    ElementosRecubrimiento.add(elementoscontar);
                    ElementosIniciadores.add(elementoscontar);
                    Elementos.add(elementoscontar);
                    elementoscontar++;
                }
            }
            for(int y=0; y<elementosyRec; y++){
                for(int x=0; x<elementosx; x++){
                    ElementosRecubrimiento.add(elementoscontar);
                    Elementos.add(elementoscontar);
                    elementoscontar++;
                }
            }
            for(int y=0; y<elementosyBase; y++){
                for(int x=0; x<elementosx; x++){
                    Elementos.add(elementoscontar);
                    ElementosBase.add(elementoscontar);
                    elementoscontar++;
                }
            }
            for(int y=0; y<elementosyRec; y++){
                for(int x=0; x<elementosx; x++){
                    Elementos.add(elementoscontar);
                    ElementosRecubrimiento.add(elementoscontar);
                    elementoscontar++;
```

```
        }
    }
    for(int y=0; y<elementosyInic; y++){
        for(int x=0; x<elementosx; x++){
            ElementosIniciadores.add(elementoscontar);
            ElementosRecubrimiento.add(elementoscontar);
            Elementos.add(elementoscontar);
            elementoscontar++;
        }
    }
}
for(int i=0; i<Elementos.size();i++){
    if(i<elementosedge) {
        ElementosEdgeEffect.add(Elementos.get(i));
    }
    if(i>=auxelemsx){
        ElementosEdgeEffect.add(Elementos.get(i));
    }
    if((i+1)%elementosx==0) {
        auxelemsx= auxelemsx + elementosx;
        elementosedge = elementosedge + elementosx;
    }
}

int nodosporcara = (elementosx+1)*(elementosyTotal+1);
nodosCont=1;
for(int z=0;z<elementosz;z++){
    for(int y=0; y<elementosyTotal; y++){
        for(int x=0; x<elementosx; x++){
            Nodo1.add(nodosCont);
            Nodo2.add(nodosCont+1);
            Nodo3.add(nodosCont+elementosx+2);
            Nodo4.add(nodosCont+elementosx+1);
            Nodo5.add(nodosCont+nodosporcara);
            Nodo6.add(nodosCont+1+nodosporcara);
            Nodo7.add(nodosCont+elementosx+2+nodosporcara);
            Nodo8.add(nodosCont+elementosx+1+nodosporcara);
            nodosCont++;
        }
        nodosCont++;
    }
}

ArrayList<Integer> BoundaryNodosIzquierda = new ArrayList<>();
ArrayList<Integer> BoundaryNodosDerecha = new ArrayList<>();

for(int i=0; i<Nodos.size();i++) {
    if (NodosX.get(i) == 0) {
        BoundaryNodosIzquierda.add(Nodos.get(i));
    }
    if (NodosX.get(i) == dimx) {
        BoundaryNodosDerecha.add(Nodos.get(i));
    }
}

for(int i=0; i<nodosboundaryeliminar;i++){
    int sizetemp = BoundaryNodosDerecha.size()/2;
    BoundaryNodosIzquierda.remove(sizetemp-1);
    BoundaryNodosDerecha.remove(sizetemp-1);
    BoundaryNodosIzquierda.remove(sizetemp-1);
    BoundaryNodosDerecha.remove(sizetemp-1);
    BoundaryNodosIzquierda.remove(0);
    BoundaryNodosDerecha.remove(0);
    BoundaryNodosIzquierda.remove(BoundaryNodosIzquierda.size()-1);
    BoundaryNodosDerecha.remove(BoundaryNodosDerecha.size()-1);
}

//Generacion de superficies para fractura
```

```
        int numeroDeFracturas = (elementosx/EspaciadoFractura)-1;
        System.out.println("Iniciador "+(int)(dimyInic*1E6)+", con KIC de
"+(int)(KIC*1E-6));
        System.out.println("Espaciado a crear de
"+EspaciadoFractura*AumentoHorizonal+" en una probeta de "+dimx+" de longitud.");
        System.out.println("Se crearan "+numeroDeFracturas+" superficies para
fractura a lo largo del recubrimiento.");


if((double)elementosx/(double)EspaciadoFractura%(double)(elementosx/EspaciadoFractu
ra)>0)
            System.out.println("Advertencia: No hay simetria en el modelo");

        //Generacion del crack de fractura
        for(int i=0; i<ElementosIniciadores.size()/2;i++) {
            int division = ElementosIniciadores.size()/2;
            if((i+1)%EspaciadoFractura==0&&(i+1)%division!=0){
                int index = Elementos.indexOf(ElementosIniciadores.get(i));
                int index2 =
Elementos.indexOf(ElementosIniciadores.get(i+elementosx));
                int nuevoindicedenodo = DuplicarNodo(Nodo2.get(index));
                int nuevoindicedenodo2 = DuplicarNodo(Nodo3.get(index2));

                Nodo2.remove(index);
                Nodo2.add(index,nuevoindicedenodo);
                Nodo3.remove(index2);
                Nodo3.add(index2,nuevoindicedenodo2);

                nuevoindicedenodo = DuplicarNodo(Nodo6.get(index));
                nuevoindicedenodo2 = DuplicarNodo(Nodo7.get(index2));

                Nodo6.remove(index);
                Nodo6.add(index,nuevoindicedenodo);
                Nodo7.remove(index2);
                Nodo7.add(index2,nuevoindicedenodo2);
            }
        }

        ArrayList<ArrayList<Integer>> elementostop = new ArrayList<>();
        ArrayList<ArrayList<Integer>> elementosbottom = new ArrayList<>();
        ArrayList<ArrayList<Integer>> bnodesfractura = new ArrayList<>();

        int contador = 0;

        //Elementos inferiores
        for(int i=0; i<ElementosRecubrimiento.size()/2;i++){
            int division =
ElementosRecubrimiento.size()/(2*(elementosyRec+elementosyInic));

            if((i+1)%EspaciadoFractura==0&&i<(ElementosRecubrimiento.size()/2-
elementosx)&&(i+1)%division!=0){
                if((contador+1)>elementostop.size())
                    elementostop.add(new ArrayList<>());
                if((contador+1)>elementosbottom.size())
                    elementosbottom.add(new ArrayList<>());
                if((contador+1)>bnodesfractura.size())
                    bnodesfractura.add(new ArrayList<>());

                elementostop.get(contador).add(ElementosRecubrimiento.get(i));
                elementosbottom.get(contador).add(ElementosRecubrimiento.get(i+1));

                int index = Elementos.indexOf(ElementosRecubrimiento.get(i));

                bnodesfractura.get(contador).add(Nodo3.get(index));
                bnodesfractura.get(contador).add(Nodo7.get(index));

                int nuevoindicedenodo = DuplicarNodo(Nodo3.get(index));
                int nuevoindicedenodo2 = DuplicarNodo(Nodo7.get(index));
```

```
                Nodo2.remove(index+elementosx);
                Nodo2.add(index+elementosx,nuevoindicedenodo);
                Nodo6.remove(index+elementosx);
                Nodo6.add(index+elementosx,nuevoindicedenodo2);

                Nodo3.remove(index);
                Nodo3.add(index,nuevoindicedenodo);
                Nodo7.remove(index);
                Nodo7.add(index,nuevoindicedenodo2);

                bnodesfractura.get(contador).add(Nodo3.get(index));
                bnodesfractura.get(contador).add(Nodo7.get(index));
                contador++;
            }
        }

        //Elementos superiores
        for(int i=ElementosRecubrimiento.size()/2;
i<ElementosRecubrimiento.size();i++) {
            int division = ElementosRecubrimiento.size() / (2 * (elementosyRec +
elementosyInic));

            if ((i + 1) % EspaciadoFractura == 0 && (i + 1) % division != 0 && (i +
1) < (ElementosRecubrimiento.size() / 2 + elementosx)) {
                if((contador+1)>elementostop.size())
                    elementostop.add(new ArrayList<>());
                if((contador+1)>elementosbottom.size())
                    elementosbottom.add(new ArrayList<>());
                if((contador+1)>bnodesfractura.size())
                    bnodesfractura.add(new ArrayList<>());


elementostop.get(contador).add(ElementosRecubrimiento.get(i+elementosx));

elementosbottom.get(contador).add(ElementosRecubrimiento.get(i+1+elementosx));

                int index = Elementos.indexOf(ElementosRecubrimiento.get(i));

                bnodesfractura.get(contador).add(Nodo3.get(index));
                bnodesfractura.get(contador).add(Nodo7.get(index));

                int nuevoindicedenodo = DuplicarNodo(Nodo3.get(index));
                int nuevoindicedenodo2 = DuplicarNodo(Nodo7.get(index));

                Nodo2.remove(index+elementosx);
                Nodo2.add(index+elementosx,nuevoindicedenodo);
                Nodo6.remove(index+elementosx);
                Nodo6.add(index+elementosx,nuevoindicedenodo2);

                Nodo3.remove(index);
                Nodo3.add(index,nuevoindicedenodo);
                Nodo7.remove(index);
                Nodo7.add(index,nuevoindicedenodo2);

                bnodesfractura.get(contador).add(Nodo3.get(index));
                bnodesfractura.get(contador).add(Nodo7.get(index));
                contador++;
            }
        }

        String CarpetaRoot = "C:\\Users\\diego\\Desktop\\Tesis No Drive\\Output
Files";
        File RootFolder = new File(CarpetaRoot);
        if(!RootFolder.isDirectory())
            RootFolder.mkdir();

        String CarpetaRoot1 = CarpetaRoot+"\\Dimx "+dimx+" DimyBase "+dimyBase+"
```

```
Dimz "+dimz+" DimyRec "+dimyRec+" Desp "+PorcentajeDesplazamiento;
        File RootFolder1 = new File(CarpetaRoot1);
        if(!RootFolder1.isDirectory())
            RootFolder1.mkdir();

        String CarpetaInic = CarpetaRoot1+"\\"+"Iniciador " +
Integer.toString((int)(dimyInic*1E6));
        File InicFolder = new File(CarpetaInic);
        if(!InicFolder.isDirectory())
            InicFolder.mkdir();

        String Carpeta = CarpetaInic+"\\"+"KIC "+Integer.toString((int)(KIC*1E-6));
        File Folder = new File(Carpeta);
        if(!Folder.isDirectory())
            Folder.mkdir();

        cleanDirectoryBefore(Folder);

        //Imprimiendo los archivos

        File nodosFile = new File(Carpeta+"\\"+"Fracture_nodes.inp");
        PrintWriter nodosPrintWriter = new PrintWriter(nodosFile);
        PrintHeading(nodosPrintWriter);
        nodosPrintWriter.println("*Node");
        for(int i=0; i<Nodos.size();i++){
            nodosPrintWriter.println(Nodos.get(i)+", "+NodosX.get(i)+",
"+NodosY.get(i)+", "+NodosZ.get(i));
        }
        PrintFooting(nodosPrintWriter);

        File elemsFile = new File(Carpeta+"\\"+"Fracture_elems.inp");
        PrintWriter elemsPrintWriter = new PrintWriter(elemsFile);
        PrintHeading(elemsPrintWriter);
        elemsPrintWriter.println("*Element, type=C3D8");
        for(int i=0; i<Elementos.size();i++){
            elemsPrintWriter.println(Elementos.get(i)+", "+Nodo1.get(i)+",
"+Nodo2.get(i)+", "+Nodo3.get(i)+", "+Nodo4.get(i)+", "+Nodo5.get(i)+",
"+Nodo6.get(i)+", "+Nodo7.get(i)+", "+Nodo8.get(i));
        }
        PrintFooting(elemsPrintWriter);

        File nodesetsFile = new File(Carpeta+"\\"+"Fracture_nodesets.inp");
        PrintWriter nodesetsPrintWriter = new PrintWriter(nodesetsFile);
        PrintHeading(nodesetsPrintWriter);

        nodesetsPrintWriter.println("*NSET, NSET=NodosIzq");
        for(int i=0; i<BoundaryNodosIzquierda.size();i++){
            nodesetsPrintWriter.print(BoundaryNodosIzquierda.get(i)+", ");
            if((i+1)%8==0)
                nodesetsPrintWriter.println();
        }
        if(BoundaryNodosIzquierda.size()%8!=0)
            nodesetsPrintWriter.println();

        nodesetsPrintWriter.println("*NSET, NSET=NodosDer");
        for(int i=0; i<BoundaryNodosDerecha.size();i++){
            nodesetsPrintWriter.print(BoundaryNodosDerecha.get(i)+", ");
            if((i+1)%8==0)
                nodesetsPrintWriter.println();
        }
        if(BoundaryNodosDerecha.size()%8!=0)
            nodesetsPrintWriter.println();

        PrintFooting(nodesetsPrintWriter);

        File elsetFile = new File(Carpeta+"\\"+"Fracture_elset.inp");
        PrintWriter elsetPrintWriter = new PrintWriter(elsetFile);
        PrintHeading(elsetPrintWriter);
```

```
        elsetPrintWriter.println("*Elset, elset=cube, generate\n1, "+
contadorelementos +", 1");

        elsetPrintWriter.println("*Elset, elset=Base_set");
        for(int i=0; i<ElementosBase.size();i++){
            elsetPrintWriter.print(ElementosBase.get(i)+", ");
            if((i+1)%8==0)
                elsetPrintWriter.println();
        }
        if(ElementosBase.size()%8!=0)
            elsetPrintWriter.println();

        elsetPrintWriter.println("*Elset, elset=Recubrimiento_set");
        for(int i=0; i<ElementosRecubrimiento.size();i++){
            elsetPrintWriter.print(ElementosRecubrimiento.get(i)+", ");
            if((i+1)%8==0)
                elsetPrintWriter.println();
        }
        if(ElementosRecubrimiento.size()%8!=0)
            elsetPrintWriter.println();

        elsetPrintWriter.println("*Elset, elset=Iniciadores_set");
        for(int i=0; i<ElementosIniciadores.size();i++){
            elsetPrintWriter.print(ElementosIniciadores.get(i)+", ");
            if((i+1)%8==0)
                elsetPrintWriter.println();
        }
        if(ElementosIniciadores.size()%8!=0)
            elsetPrintWriter.println();

        elsetPrintWriter.println("*Elset, elset=EdgeEffect_set");
        for(int i=0; i<ElementosEdgeEffect.size();i++){
            elsetPrintWriter.print(ElementosEdgeEffect.get(i)+", ");
            if((i+1)%8==0)
                elsetPrintWriter.println();
        }
        if(ElementosEdgeEffect.size()%8!=0)
            elsetPrintWriter.println();

        PrintFooting(elsetPrintWriter);

        File propertiesFile = new File(Carpeta+"\\"+"Fracture_properties.inp");
        PrintWriter propertiesPrintWriter = new PrintWriter(propertiesFile);
        PrintHeading(propertiesPrintWriter);
        propertiesPrintWriter.println("*Solid Section, elset=Base_set,
material=Base\n1.\n*Solid Section, elset=Recubrimiento_set,
material=Recubrimiento\n1.");
        propertiesPrintWriter.println("*Material, name=Base\n" + "*ELASTIC\n" +
YoungModBase+", "+PoissonBase+"\n" + "*PLASTIC");

        for (KVpair<Double, Double> propiedadesplastica : propiedadesplasticas) {
            propertiesPrintWriter.println(propiedadesplastica.key() + ", " +
propiedadesplastica.value());
        }

        propertiesPrintWriter.println("*Material, name=Recubrimiento\n" +
"*ELASTIC\n" + YoungModRecubrimiento+", "+PoissonRecubrimiento);
        PrintFooting(propertiesPrintWriter);

        File fractureFile = new File(Carpeta+"\\"+"Fracture.inp");
        PrintWriter fracturePrintWriter = new PrintWriter(fractureFile);
        PrintHeading(fracturePrintWriter);
        fracturePrintWriter.println("*Heading\nFracture");
        fracturePrintWriter.println("** ----------------------------------------
----------------------");
        fracturePrintWriter.println("*parameter\n" +
                "** Fracture toughness:\n" +
                " GIc   = "+GIC+"\n" +
```

```
                " GIIc  = "+GIIC+"\n" +
                " GIIIc = "+GIIIC+"\n" +
                "** B-K parameter:\n" +
                " eta="+eta+"\n" +
                "***\n" +
                "*Preprint, echo = NO, model = NO, history = NO, contact = NO");

        fracturePrintWriter.println(
                "*Include, Input =Fracture_nodes.inp\n" +
                        "*Include, Input =Fracture_elems.inp\n" +
                        "*Include, Input =Fracture_elset.inp\n" +
                        "*Include, Input =Fracture_properties.inp\n"+
                        "*Include, Input =Fracture_nodesets.inp");

        fracturePrintWriter.println("** BNODES DECLARATION\n**");

        for(int i=0; i<bnodesfractura.size();i++){
            fracturePrintWriter.println("*NSET,NSET=BNODES"+(i+1));
            for(int j=0; j<bnodesfractura.get(i).size();j++){
                fracturePrintWriter.print(bnodesfractura.get(i).get(j)+", ");
                if((j+1)%8==0)
                    fracturePrintWriter.println();
            }
            if(bnodesfractura.get(i).size()%8!=0)
                fracturePrintWriter.println();
        }

        fracturePrintWriter.println("**ELSET DECLARATION\n**");

        for(int i=0; i<elementosbottom.size();i++){
            fracturePrintWriter.println("*Elset, elset=BOT"+(i+1));
            for(int j=0; j<elementosbottom.get(i).size();j++){
                fracturePrintWriter.print(elementosbottom.get(i).get(j)+", ");
                if((j+1)%8==0)
                    fracturePrintWriter.println();
            }
            if(elementosbottom.get(i).size()%8!=0)
                fracturePrintWriter.println();
        }

        for(int i=0; i<elementostop.size();i++){
            fracturePrintWriter.println("*Elset, elset=TOP"+(i+1));
            for(int j=0; j<elementostop.get(i).size();j++){
                fracturePrintWriter.print(elementostop.get(i).get(j)+", ");
                if((j+1)%8==0)
                    fracturePrintWriter.println();
            }
            if(elementostop.get(i).size()%8!=0)
                fracturePrintWriter.println();
        }

        for(int i=0; i<elementosbottom.size();i++){
            fracturePrintWriter.println("*SURFACE, NAME=BOT"+(i+1));
            fracturePrintWriter.println("BOT"+(i+1)+", S6");
        }

        for(int i=0; i<elementostop.size();i++){
            fracturePrintWriter.println("*SURFACE, NAME=TOP"+(i+1));
            fracturePrintWriter.println("TOP"+(i+1)+", S4");
        }

        for(int i=0; i<bnodesfractura.size();i++){
            fracturePrintWriter.println("*CONTACT PAIR, INTERACTION=FRACT"+(i+1)+",
ADJUST=BNODES"+(i+1)+"\n"+"BOT"+(i+1)+", TOP"+(i+1));
            fracturePrintWriter.println("*SURFACE INTERACTION,
NAME=FRACT"+(i+1)+"\n"+"1.0");
        }
```

```
        fracturePrintWriter.println("*INITIAL CONDITIONS,TYPE =CONTACT");

        for(int i=0; i<bnodesfractura.size();i++){
            fracturePrintWriter.println("BOT"+(i+1)+", TOP"+(i+1)+",
BNODES"+(i+1));
        }

        fracturePrintWriter.println("**\n" +
                "*STEP,NLGEOM, INC="+maximoIncremento+",convert sdi=no\n" +
                "*STATIC\n" +
                InitialTime+", "+TimeAnalysis+", "+MinTime+", "+MaxTime+"\n"+
                "*CONTROLS,PARAMETERS=TIME INCREMENTATION\n" +
                " , , , , , , 10000");

        for(int i=0; i<bnodesfractura.size();i++){
            fracturePrintWriter.println("*CONTACT
PRINT\n"+"*DEBOND,SLAVE=BOT"+(i+1)+",MASTER=TOP"+(i+1)+",FREQ=1\n"
                    +"*FRACTURE CRITERION,TYPE=VCCT,MIXED MODE BEHAVIOR=BK\n"+
                    "<GIc>,<GIIc>,<GIIIc>,<eta>");
        }

        fracturePrintWriter.println("*BOUNDARY\n" +
                "NodosDer, 1,1,"+desplazarProbeta+"\n" +
                "NodosIzq, 1,1,-"+desplazarProbeta+"\n" +
                "*NODE PRINT\n" +
                "RF,\n" +
                "*EL FILE\n" +
                "S, E\n" +
                "*NODE FILE\n" +
                "U, RF\n" +
                "*END STEP");

        PrintFooting(fracturePrintWriter);

        Process process = Runtime.getRuntime().exec("cmd /c start /wait cmd.exe /K
\"cd " + Carpeta + " && abaqus analysis job=Fracture int && exit");

        process.waitFor();
        process.destroy();
        cleanDirectoryAfter(Folder);
    }

    private static void cleanDirectoryAfter(File dir){
        for (File file : dir.listFiles()) {
            if (!file.getName().equals("Fracture.odb")){
                file.delete();
            }
        }
    }

    private static void cleanDirectoryBefore(File dir){
        for (File file : dir.listFiles()) {
                file.delete();
        }
    }

    private static void PrintHeading(PrintWriter write){
        write.print("** Generated by : Diego S. Morales Arcos\n** ----------------
------------------------------------------------\n**\n");
    }
    private static void PrintFooting(PrintWriter write){
        write.print("**\n** ----------------------------------------------------------
---------\n**");
        write.close();
    }

    private static Integer DuplicarNodo(Integer NodoADuplicar){
        int index = Nodos.indexOf(NodoADuplicar);
```

```
            Nodos.add(Nodos.get(Nodos.size()-1)+1);
            NodosX.add(NodosX.get(index));
            NodosY.add(NodosY.get(index));
            NodosZ.add(NodosZ.get(index));
            return Nodos.get(Nodos.size()-1);
        }

        static class KVpair<Key, E> {
             private Key k;
             private E e;

            KVpair()
            { k = null; e = null; }
            KVpair(Key kval, E eval)
            { k = kval; e = eval; }

            public Key key() { return k; }
            public E value() { return e; }
        }

    }
```

ANEXO G: CÓDIGO JAVA PARA MODELOS DE LA SEGUNDA FRACTURA

Ejecutor.java

```
import java.util.ArrayList;
import java.io.*;
public class Ejecutor {
    private static ArrayList<Integer> Nodos = new ArrayList<>();
    private static ArrayList<Double> NodosX = new ArrayList<>();
    private static ArrayList<Double> NodosY = new ArrayList<>();
    private static ArrayList<Double> NodosZ = new ArrayList<>();
    private static ArrayList<Integer> Elementos = new ArrayList<>();
    private static ArrayList<Integer> ElementosBase = new ArrayList<>();
    private static ArrayList<Integer> ElementosRecubrimiento = new ArrayList<>();
    private static ArrayList<Integer> ElementosIniciadores = new ArrayList<>();
    private static ArrayList<Integer> ElementosEdgeEffect = new ArrayList<>();
    private static ArrayList<Integer> ElementosLaterales = new ArrayList<>();
    private static ArrayList<Integer> Nodo1 = new ArrayList<>();
    private static ArrayList<Integer> Nodo2 = new ArrayList<>();
    private static ArrayList<Integer> Nodo3 = new ArrayList<>();
    private static ArrayList<Integer> Nodo4 = new ArrayList<>();
    private static ArrayList<Integer> Nodo5 = new ArrayList<>();
    private static ArrayList<Integer> Nodo6 = new ArrayList<>();
    private static ArrayList<Integer> Nodo7 = new ArrayList<>();
    private static ArrayList<Integer> Nodo8 = new ArrayList<>();

    public static void main(String[] args) throws Exception{

        double[] Iniciadores =
{0.000001,0.000002,0.000003,0.000004,0.000005,0.000006,0.000007,0.000008,0.000009,0
.000010};
        double[] KICs = {3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20};
        int[] EspaciadosFractura = {100};
        double[] dimyRecs = {0.0002};

        for (double Iniciador: Iniciadores) {
            for (double KIC : KICs) {
                for(int EspaciadoFractura : EspaciadosFractura) {
                    for(double dimyRec : dimyRecs) {
                        ExecutionWithParameters(0.006, 0.0032, 0.0125, dimyRec,
Iniciador,
                                200, 10, EspaciadoFractura, KIC * 1E6, 3,
                                50, 0.0015);
                    }
                }
            }
        }
    }

    private static void ExecutionWithParameters (double dimx, double dimyBase,
double dimz, double dimyRec, double dimyInic, int elementosx,
                                                 int elementosyBase, int
EspaciadoFractura, double KIC, double PorcentajeDesplazamiento,
                                                 int elementosxlaterales, double
dimxlateral)throws Exception{
        //Elementos
        //Base
        int elementosz = 1; //Siempre un elemento

        //Recubrimiento
        int elementosyRec = 1;//Siempre un elemento
        int elementosyInic = 1; //Siempre un elemento
```

```
int elementosedge = elementosx*4/10;

double YoungModBase = 207E9;
double YoungModRecubrimiento = 560E9;
double PoissonBase = 0.3;
double PoissonRecubrimiento = 0.24;

double KIIC = 4E6;
double KIIIC = 4E6;
double eta = 1.75;

int maximoIncremento = 200;
double TimeAnalysis = 1;
double InitialTime = 0.005;
double MaxTime = 0.01;
double MinTime = 0.00000000015;

ArrayList<KVpair<Double, Double>> propiedadesplasticas = new ArrayList<>();
propiedadesplasticas.add(new KVpair<>(175E6,0.0));
propiedadesplasticas.add(new KVpair<>(260E6,0.05));
propiedadesplasticas.add(new KVpair<>(310E6,0.1));
propiedadesplasticas.add(new KVpair<>(340E6,0.15));
propiedadesplasticas.add(new KVpair<>(375E6,0.2));
propiedadesplasticas.add(new KVpair<>(395E6,0.25));
propiedadesplasticas.add(new KVpair<>(420E6,0.3));
propiedadesplasticas.add(new KVpair<>(425E6,0.35));

int elementosyTotal = (elementosyBase+2*(elementosyRec+elementosyInic));
int auxelemsx = elementosx-elementosedge;
int contadorelementos = elementosx*elementosyTotal*elementosz;
double AumentoElementosRec = dimyRec/elementosyRec;
double AumentoElementosBase = dimyBase/elementosyBase;
double AumentoElementosInic = dimyInic/elementosyInic;
double AumentoHorizonal = dimx/elementosx;
double AumentoProfundidad = dimz/elementosz;

double GIC = KIC*KIC/(YoungModRecubrimiento);
double GIIC = KIIC*KIIC/(YoungModRecubrimiento);
double GIIIC = KIIIC*KIIIC/(YoungModRecubrimiento);

double desplazarProbeta = (PorcentajeDesplazamiento/100)*dimx/2;

Nodos.clear();
NodosX.clear();
NodosY.clear();
NodosZ.clear();
Elementos.clear();
ElementosBase.clear();
ElementosRecubrimiento.clear();
ElementosIniciadores.clear();
ElementosEdgeEffect.clear();
Nodo1.clear();
Nodo2.clear();
Nodo3.clear();
Nodo4.clear();
Nodo5.clear();
Nodo6.clear();
Nodo7.clear();
Nodo8.clear();
ElementosLaterales.clear();

//Nodos
int nodosCont = 1;
for(int z=0; z<=elementosz; z++){
    for(int y=0; y<=elementosyInic; y++){
        for(int x=0; x<=elementosx; x++){
            Nodos.add(nodosCont);
            NodosX.add(x*AumentoHorizonal);
```

```
                    NodosY.add(y*AumentoElementosInic);
                    NodosZ.add(z*AumentoProfundidad);
                    nodosCont++;
                }
            }
            for(int y=0; y<elementosyRec; y++) {
                for (int x = 0; x <= elementosx; x++) {
                    Nodos.add(nodosCont);
                    NodosX.add(x * AumentoHorizonal);
                    NodosY.add(dimyInic + (y+1)*AumentoElementosRec);
                    NodosZ.add(z * AumentoProfundidad);
                    nodosCont++;
                }
            }
            for(int y=0; y<elementosyBase; y++){
                for(int x=0; x<=elementosx; x++){
                    Nodos.add(nodosCont);
                    NodosY.add(dimyInic+dimyRec+(y+1)*AumentoElementosBase);
                    NodosX.add(x*AumentoHorizonal);
                    NodosZ.add(z*AumentoProfundidad);
                    nodosCont++;
                }
            }
            for(int y=0; y<elementosyRec; y++){
                for(int x=0; x<=elementosx; x++){
                    Nodos.add(nodosCont);
                    NodosZ.add(z*AumentoProfundidad);
                    NodosX.add(x*AumentoHorizonal);

NodosY.add(dimyInic+dimyRec+dimyBase+(y+1)*AumentoElementosRec);
                    nodosCont++;
                }
            }
            for(int y=0; y<elementosyInic; y++){
                for(int x=0; x<=elementosx; x++){
                    Nodos.add(nodosCont);
                    NodosX.add(x*AumentoHorizonal);

NodosY.add(dimyInic+2*dimyRec+dimyBase+(y+1)*AumentoElementosInic);
                    NodosZ.add(z*AumentoProfundidad);
                    nodosCont++;
                }
            }
        }

        //Elementos
        int elementoscontar = 1;
        for(int z=0; z<elementosz; z++){
            for(int y=0; y<elementosyInic; y++){
                for(int x=0; x<elementosx; x++){
                    ElementosRecubrimiento.add(elementoscontar);
                    ElementosIniciadores.add(elementoscontar);
                    Elementos.add(elementoscontar);
                    elementoscontar++;
                }
            }
            for(int y=0; y<elementosyRec; y++){
                for(int x=0; x<elementosx; x++){
                    ElementosRecubrimiento.add(elementoscontar);
                    Elementos.add(elementoscontar);
                    elementoscontar++;
                }
            }
            for(int y=0; y<elementosyBase; y++){
                for(int x=0; x<elementosx; x++){
                    Elementos.add(elementoscontar);
                    ElementosBase.add(elementoscontar);
                    elementoscontar++;
```

```
                }
            }
            for(int y=0; y<elementosyRec; y++){
                for(int x=0; x<elementosx; x++){
                    Elementos.add(elementoscontar);
                    ElementosRecubrimiento.add(elementoscontar);
                    elementoscontar++;
                }
            }
            for(int y=0; y<elementosyInic; y++){
                for(int x=0; x<elementosx; x++){
                    ElementosIniciadores.add(elementoscontar);
                    ElementosRecubrimiento.add(elementoscontar);
                    Elementos.add(elementoscontar);
                    elementoscontar++;
                }
            }
        }
        for(int i=0; i<Elementos.size();i++){
            if(i<elementosedge) {
                ElementosEdgeEffect.add(Elementos.get(i));
            }
            if(i>=auxelemsx){
                ElementosEdgeEffect.add(Elementos.get(i));
            }
            if((i+1)%elementosx==0) {
                auxelemsx= auxelemsx + elementosx;
                elementosedge = elementosedge + elementosx;
            }
        }

        int nodosporcara = (elementosx+1)*(elementosyTotal+1);
        nodosCont=1;
        for(int z=0;z<elementosz;z++){
            for(int y=0; y<elementosyTotal; y++){
                for(int x=0; x<elementosx; x++){
                    Nodo1.add(nodosCont);
                    Nodo2.add(nodosCont+1);
                    Nodo3.add(nodosCont+elementosx+2);
                    Nodo4.add(nodosCont+elementosx+1);
                    Nodo5.add(nodosCont+nodosporcara);
                    Nodo6.add(nodosCont+1+nodosporcara);
                    Nodo7.add(nodosCont+elementosx+2+nodosporcara);
                    Nodo8.add(nodosCont+elementosx+1+nodosporcara);
                    nodosCont++;
                }
                nodosCont++;
            }
        }

        //Generacion de superficies para fractura
        int numeroDeFracturas = (elementosx/EspaciadoFractura)-1;
        System.out.println("Iniciador "+(int)(dimyInic*1E6)+", con KIC de
"+(int)(KIC*1E-6));
        System.out.println("Espaciado a crear de
"+EspaciadoFractura*AumentoHorizonal+" en una probeta de "+dimx+" de longitud.");
        System.out.println("Se crearan "+numeroDeFracturas+" superficies para
fractura a lo largo del recubrimiento.");


if((double)elementosx/(double)EspaciadoFractura%(double)(elementosx/EspaciadoFractu
ra)>0)
            System.out.println("Advertencia: No hay simetria en el modelo");

        //Generacion del crack de fractura
        for(int i=0; i<ElementosIniciadores.size()/2;i++) {
            int division = ElementosIniciadores.size()/2;
            if((i+1)%EspaciadoFractura==0&&(i+1)%division!=0){
```

```
                int index = Elementos.indexOf(ElementosIniciadores.get(i));
                int index2 =
Elementos.indexOf(ElementosIniciadores.get(i+elementosx));
                int nuevoindicedenodo = DuplicarNodo(Nodo2.get(index));
                int nuevoindicedenodo2 = DuplicarNodo(Nodo3.get(index2));

                Nodo2.remove(index);
                Nodo2.add(index,nuevoindicedenodo);
                Nodo3.remove(index2);
                Nodo3.add(index2,nuevoindicedenodo2);

                nuevoindicedenodo = DuplicarNodo(Nodo6.get(index));
                nuevoindicedenodo2 = DuplicarNodo(Nodo7.get(index2));

                Nodo6.remove(index);
                Nodo6.add(index,nuevoindicedenodo);
                Nodo7.remove(index2);
                Nodo7.add(index2,nuevoindicedenodo2);
            }
        }

        ArrayList<ArrayList<Integer>> elementostop = new ArrayList<>();
        ArrayList<ArrayList<Integer>> elementosbottom = new ArrayList<>();
        ArrayList<ArrayList<Integer>> bnodesfractura = new ArrayList<>();

        int contador = 0;

        //Elementos inferiores
        for(int i=0; i<ElementosRecubrimiento.size()/2;i++){
            int division =
ElementosRecubrimiento.size()/(2*(elementosyRec+elementosyInic));

            if((i+1)%EspaciadoFractura==0&&i<(ElementosRecubrimiento.size()/2-
elementosx)&&(i+1)%division!=0){
                if((contador+1)>elementostop.size())
                    elementostop.add(new ArrayList<>());
                if((contador+1)>elementosbottom.size())
                    elementosbottom.add(new ArrayList<>());
                if((contador+1)>bnodesfractura.size())
                    bnodesfractura.add(new ArrayList<>());

                elementostop.get(contador).add(ElementosRecubrimiento.get(i));
                elementosbottom.get(contador).add(ElementosRecubrimiento.get(i+1));

                int index = Elementos.indexOf(ElementosRecubrimiento.get(i));

                bnodesfractura.get(contador).add(Nodo3.get(index));
                bnodesfractura.get(contador).add(Nodo7.get(index));

                int nuevoindicedenodo = DuplicarNodo(Nodo3.get(index));
                int nuevoindicedenodo2 = DuplicarNodo(Nodo7.get(index));

                Nodo2.remove(index+elementosx);
                Nodo2.add(index+elementosx,nuevoindicedenodo);
                Nodo6.remove(index+elementosx);
                Nodo6.add(index+elementosx,nuevoindicedenodo2);

                Nodo3.remove(index);
                Nodo3.add(index,nuevoindicedenodo);
                Nodo7.remove(index);
                Nodo7.add(index,nuevoindicedenodo2);

                bnodesfractura.get(contador).add(Nodo3.get(index));
                bnodesfractura.get(contador).add(Nodo7.get(index));
                contador++;
            }
        }
```

```
        //Elementos superiores
        for(int i=ElementosRecubrimiento.size()/2;
i<ElementosRecubrimiento.size();i++) {
            int division = ElementosRecubrimiento.size() / (2 * (elementosyRec +
elementosyInic));

            if ((i + 1) % EspaciadoFractura == 0 && (i + 1) % division != 0 && (i +
1) < (ElementosRecubrimiento.size() / 2 + elementosx)) {
                if((contador+1)>elementostop.size())
                    elementostop.add(new ArrayList<>());
                if((contador+1)>elementosbottom.size())
                    elementosbottom.add(new ArrayList<>());
                if((contador+1)>bnodesfractura.size())
                    bnodesfractura.add(new ArrayList<>());


elementostop.get(contador).add(ElementosRecubrimiento.get(i+elementosx));

elementosbottom.get(contador).add(ElementosRecubrimiento.get(i+1+elementosx));

                int index = Elementos.indexOf(ElementosRecubrimiento.get(i));

                bnodesfractura.get(contador).add(Nodo3.get(index));
                bnodesfractura.get(contador).add(Nodo7.get(index));

                int nuevoindicedenodo = DuplicarNodo(Nodo3.get(index));
                int nuevoindicedenodo2 = DuplicarNodo(Nodo7.get(index));

                Nodo2.remove(index+elementosx);
                Nodo2.add(index+elementosx,nuevoindicedenodo);
                Nodo6.remove(index+elementosx);
                Nodo6.add(index+elementosx,nuevoindicedenodo2);

                Nodo3.remove(index);
                Nodo3.add(index,nuevoindicedenodo);
                Nodo7.remove(index);
                Nodo7.add(index,nuevoindicedenodo2);

                bnodesfractura.get(contador).add(Nodo3.get(index));
                bnodesfractura.get(contador).add(Nodo7.get(index));
                contador++;
            }
        }

        //Agregando elementos laterales
        double AumentoLateral = dimxlateral/elementosxlaterales;
        elementoscontar = Elementos.size()+1;
        for(int z=0; z<elementosz; z++){
            for(int y=0; y<elementosyBase; y++){
                for(int x=0; x<elementosxlaterales; x++){
                    ElementosBase.add(elementoscontar);
                    ElementosLaterales.add(elementoscontar);
                    Elementos.add(elementoscontar);
                    elementoscontar++;
                }
            }
        }

        int aux = Nodos.size()+1;
        int nodocaralateral = (elementosxlaterales+1)*(elementosyBase+1);
        nodosCont = Nodos.size()+1;
        for(int z=0; z<=elementosz; z++){
            for(int y=0; y<=elementosyBase; y++){
                for(int x=0; x<=elementosxlaterales; x++){
                    Nodos.add(nodosCont);
                    NodosX.add(dimx+x*AumentoLateral);
                    NodosY.add(dimyInic+dimyRec+y*AumentoElementosBase);
                    NodosZ.add(z*AumentoProfundidad);
```

```
                    nodosCont++;
                }
            }
        }

        nodosCont = aux;
        for(int z=0;z<elementosz;z++){
            for(int y=0; y<elementosyBase; y++){
                for(int x=0; x<elementosxlaterales; x++){
                    Nodo5.add(nodosCont+nodocaralateral);
                    Nodo6.add(nodosCont+1+nodocaralateral);
                    Nodo7.add(nodosCont+elementosxlaterales+2+nodocaralateral);
                    Nodo8.add(nodosCont+elementosxlaterales+1+nodocaralateral);
                    Nodo1.add(nodosCont);
                    Nodo2.add(nodosCont+1);
                    Nodo3.add(nodosCont+elementosxlaterales+2);
                    Nodo4.add(nodosCont+elementosxlaterales+1);
                    nodosCont++;
                }
                nodosCont++;
            }
        }

        for(int i=0; i<ElementosLaterales.size();i++){
            //Elementos derecha
                int indexelem = Elementos.indexOf(ElementosLaterales.get(i));
                int indexnode1 = Nodos.indexOf(Nodo1.get(indexelem));
                int indexnode4 = Nodos.indexOf(Nodo4.get(indexelem));
                int indexnode5 = Nodos.indexOf(Nodo5.get(indexelem));
                int indexnode8 = Nodos.indexOf(Nodo8.get(indexelem));

                for(int j=0;j<(aux-1);j++){

if(NodosX.get(indexnode1).equals(NodosX.get(j))&&NodosY.get(indexnode1).equals(Nodo
sY.get(j))
                            &&NodosZ.get(indexnode1).equals(NodosZ.get(j))){
                        Nodo1.remove(indexelem);
                        Nodo1.add(indexelem, Nodos.get(j));
                    }

if(NodosX.get(indexnode4).equals(NodosX.get(j))&&NodosY.get(indexnode4).equals(Nodo
sY.get(j))
                            &&NodosZ.get(indexnode4).equals(NodosZ.get(j))){
                        Nodo4.remove(indexelem);
                        Nodo4.add(indexelem, Nodos.get(j));
                    }

if(NodosX.get(indexnode5).equals(NodosX.get(j))&&NodosY.get(indexnode5).equals(Nodo
sY.get(j))
                            &&NodosZ.get(indexnode5).equals(NodosZ.get(j))){
                        Nodo5.remove(indexelem);
                        Nodo5.add(indexelem, Nodos.get(j));
                    }

if(NodosX.get(indexnode8).equals(NodosX.get(j))&&NodosY.get(indexnode8).equals(Nodo
sY.get(j))
                            &&NodosZ.get(indexnode8).equals(NodosZ.get(j))){
                        Nodo8.remove(indexelem);
                        Nodo8.add(indexelem, Nodos.get(j));
                    }
            }
        }

        elementoscontar = Elementos.size()+1;
        for(int z=0; z<elementosz; z++){
            for(int y=0; y<elementosyBase; y++){
                for(int x=0; x<elementosxlaterales; x++){
                    ElementosLaterales.add(elementoscontar);
```

```
                    ElementosBase.add(elementoscontar);
                    Elementos.add(elementoscontar);
                    elementoscontar++;
                }
            }
        }

        aux = Nodos.size()+1;
        nodosCont = Nodos.size()+1;
        for(int z=0; z<=elementosz; z++){
            for(int y=0; y<=elementosyBase; y++){
                for(int x=0; x<=elementosxlaterales; x++){
                    Nodos.add(nodosCont);
                    NodosX.add(0-x*AumentoLateral);
                    NodosY.add(dimyInic+dimyRec+y*AumentoElementosBase);
                    NodosZ.add(z*AumentoProfundidad);
                    nodosCont++;
                }
            }
        }

        nodosCont = aux;
        for(int z=0;z<elementosz;z++){
            for(int y=0; y<elementosyBase; y++){
                for(int x=0; x<elementosxlaterales; x++){
                    Nodo2.add(nodosCont);
                    Nodo1.add(nodosCont+1);
                    Nodo4.add(nodosCont+elementosxlaterales+2);
                    Nodo3.add(nodosCont+elementosxlaterales+1);
                    Nodo6.add(nodosCont+nodocaralateral);
                    Nodo5.add(nodosCont+1+nodocaralateral);
                    Nodo7.add(nodosCont+elementosxlaterales+1+nodocaralateral);
                    Nodo8.add(nodosCont+elementosxlaterales+2+nodocaralateral);
                    nodosCont++;
                }
                nodosCont++;
            }
        }

        for(int i=ElementosLaterales.size()/2; i<ElementosLaterales.size();i++){
            //Elementos izquierda
            int indexelem = Elementos.indexOf(ElementosLaterales.get(i));
            int indexnode2 = Nodos.indexOf(Nodo2.get(indexelem));
            int indexnode3 = Nodos.indexOf(Nodo3.get(indexelem));
            int indexnode6 = Nodos.indexOf(Nodo6.get(indexelem));
            int indexnode7 = Nodos.indexOf(Nodo7.get(indexelem));

            for(int j=0;j<(aux-1);j++){

if(NodosX.get(indexnode2).equals(NodosX.get(j))&&NodosY.get(indexnode2).equals(Nodo
sY.get(j))
                        &&NodosZ.get(indexnode2).equals(NodosZ.get(j))){
                    Nodo2.remove(indexelem);
                    Nodo2.add(indexelem, Nodos.get(j));
                }

if(NodosX.get(indexnode3).equals(NodosX.get(j))&&NodosY.get(indexnode3).equals(Nodo
sY.get(j))
                        &&NodosZ.get(indexnode3).equals(NodosZ.get(j))){
                    Nodo3.remove(indexelem);
                    Nodo3.add(indexelem, Nodos.get(j));
                }

if(NodosX.get(indexnode6).equals(NodosX.get(j))&&NodosY.get(indexnode6).equals(Nodo
sY.get(j))
                        &&NodosZ.get(indexnode6).equals(NodosZ.get(j))){
                    Nodo6.remove(indexelem);
                    Nodo6.add(indexelem, Nodos.get(j));
```

```
                }

if(NodosX.get(indexnode7).equals(NodosX.get(j))&&NodosY.get(indexnode7).equals(Nodo
sY.get(j))
                        &&NodosZ.get(indexnode7).equals(NodosZ.get(j))){
                    Nodo7.remove(indexelem);
                    Nodo7.add(indexelem, Nodos.get(j));
                }
            }
        }

        ArrayList<Integer> BoundaryNodosIzquierda = new ArrayList<>();
        ArrayList<Integer> BoundaryNodosDerecha = new ArrayList<>();

        for(int i=0; i<Nodos.size();i++) {
            if (NodosX.get(i) == (-dimxlateral)) {
                BoundaryNodosIzquierda.add(Nodos.get(i));
            }
            if (NodosX.get(i) == (dimx+dimxlateral)) {
                BoundaryNodosDerecha.add(Nodos.get(i));
            }
        }

        String CarpetaRoot = "C:\\Users\\diego\\Desktop\\Tesis No Drive\\Output
Files";
        File RootFolder = new File(CarpetaRoot);
        if(!RootFolder.isDirectory())
            RootFolder.mkdir();

        String CarpetaRoot1 = CarpetaRoot+"\\Dimx "+dimx+" DimyBase "+dimyBase+"
Dimz "+dimz+" DimyRec "+dimyRec+" DimxLat "+dimxlateral+" Desp
"+PorcentajeDesplazamiento;
        File RootFolder1 = new File(CarpetaRoot1);
        if(!RootFolder1.isDirectory())
            RootFolder1.mkdir();

        String CarpetaInic = CarpetaRoot1+"\\"+"Iniciador " +
Integer.toString((int)(dimyInic*1E6));
        File InicFolder = new File(CarpetaInic);
        if(!InicFolder.isDirectory())
            InicFolder.mkdir();

        String Carpeta = CarpetaInic+"\\"+"KIC "+Integer.toString((int)(KIC*1E-6));
        File Folder = new File(Carpeta);
        if(!Folder.isDirectory())
            Folder.mkdir();

        cleanDirectoryBefore(Folder);

        //Imprimiendo los archivos

        File nodosFile = new File(Carpeta+"\\"+"Fracture_nodes.inp");
        PrintWriter nodosPrintWriter = new PrintWriter(nodosFile);
        PrintHeading(nodosPrintWriter);
        nodosPrintWriter.println("*Node");
        for(int i=0; i<Nodos.size();i++){
            nodosPrintWriter.println(Nodos.get(i)+", "+NodosX.get(i)+",
"+NodosY.get(i)+", "+NodosZ.get(i));
        }
        PrintFooting(nodosPrintWriter);

        File elemsFile = new File(Carpeta+"\\"+"Fracture_elems.inp");
        PrintWriter elemsPrintWriter = new PrintWriter(elemsFile);
        PrintHeading(elemsPrintWriter);
        elemsPrintWriter.println("*Element, type=C3D8");
        for(int i=0; i<Elementos.size();i++){
            elemsPrintWriter.println(Elementos.get(i)+", "+Nodo1.get(i)+",
"+Nodo2.get(i)+", "+Nodo3.get(i)+", "+Nodo4.get(i)+", "+Nodo5.get(i)+",
```

```
"+Nodo6.get(i)+", "+Nodo7.get(i)+", "+Nodo8.get(i));
        }
        PrintFooting(elemsPrintWriter);

        File nodesetsFile = new File(Carpeta+"\\"+"Fracture_nodesets.inp");
        PrintWriter nodesetsPrintWriter = new PrintWriter(nodesetsFile);
        PrintHeading(nodesetsPrintWriter);

        nodesetsPrintWriter.println("*NSET, NSET=NodosIzq");
        for(int i=0; i<BoundaryNodosIzquierda.size();i++){
            nodesetsPrintWriter.print(BoundaryNodosIzquierda.get(i)+", ");
            if((i+1)%8==0)
                nodesetsPrintWriter.println();
        }
        if(BoundaryNodosIzquierda.size()%8!=0)
            nodesetsPrintWriter.println();

        nodesetsPrintWriter.println("*NSET, NSET=NodosDer");
        for(int i=0; i<BoundaryNodosDerecha.size();i++){
            nodesetsPrintWriter.print(BoundaryNodosDerecha.get(i)+", ");
            if((i+1)%8==0)
                nodesetsPrintWriter.println();
        }
        if(BoundaryNodosDerecha.size()%8!=0)
            nodesetsPrintWriter.println();

        PrintFooting(nodesetsPrintWriter);

        File elsetFile = new File(Carpeta+"\\"+"Fracture_elset.inp");
        PrintWriter elsetPrintWriter = new PrintWriter(elsetFile);
        PrintHeading(elsetPrintWriter);
        elsetPrintWriter.println("*Elset, elset=cube, generate\n1, "+
contadorelementos +", 1");

        elsetPrintWriter.println("*Elset, elset=Base_set");
        for(int i=0; i<ElementosBase.size();i++){
            elsetPrintWriter.print(ElementosBase.get(i)+", ");
            if((i+1)%8==0)
                elsetPrintWriter.println();
        }
        if(ElementosBase.size()%8!=0)
            elsetPrintWriter.println();

        elsetPrintWriter.println("*Elset, elset=Recubrimiento_set");
        for(int i=0; i<ElementosRecubrimiento.size();i++){
            elsetPrintWriter.print(ElementosRecubrimiento.get(i)+", ");
            if((i+1)%8==0)
                elsetPrintWriter.println();
        }
        if(ElementosRecubrimiento.size()%8!=0)
            elsetPrintWriter.println();

        elsetPrintWriter.println("*Elset, elset=Iniciadores_set");
        for(int i=0; i<ElementosIniciadores.size();i++){
            elsetPrintWriter.print(ElementosIniciadores.get(i)+", ");
            if((i+1)%8==0)
                elsetPrintWriter.println();
        }
        if(ElementosIniciadores.size()%8!=0)
            elsetPrintWriter.println();

        elsetPrintWriter.println("*Elset, elset=EdgeEffect_set");
        for(int i=0; i<ElementosEdgeEffect.size();i++){
            elsetPrintWriter.print(ElementosEdgeEffect.get(i)+", ");
            if((i+1)%8==0)
                elsetPrintWriter.println();
        }
        if(ElementosEdgeEffect.size()%8!=0)
```

```
            elsetPrintWriter.println();

        elsetPrintWriter.println("*Elset, elset=ElementosLaterales_set");
        for(int i=0; i<ElementosLaterales.size();i++){
            elsetPrintWriter.print(ElementosLaterales.get(i)+", ");
            if((i+1)%8==0)
                elsetPrintWriter.println();
        }
        if(ElementosLaterales.size()%8!=0)
            elsetPrintWriter.println();

        PrintFooting(elsetPrintWriter);

        File propertiesFile = new File(Carpeta+"\\"+"Fracture_properties.inp");
        PrintWriter propertiesPrintWriter = new PrintWriter(propertiesFile);
        PrintHeading(propertiesPrintWriter);
        propertiesPrintWriter.println("*Solid Section, elset=Base_set,
material=Base\n1.\n*Solid Section, elset=Recubrimiento_set,
material=Recubrimiento\n1.");
        propertiesPrintWriter.println("*Material, name=Base\n" + "*ELASTIC\n" +
YoungModBase+", "+PoissonBase+"\n" + "*PLASTIC");

        for (KVpair<Double, Double> propiedadesplastica : propiedadesplasticas) {
            propertiesPrintWriter.println(propiedadesplastica.key() + ", " +
propiedadesplastica.value());
        }

        propertiesPrintWriter.println("*Material, name=Recubrimiento\n" +
"*ELASTIC\n" + YoungModRecubrimiento+", "+PoissonRecubrimiento);
        PrintFooting(propertiesPrintWriter);

        File fractureFile = new File(Carpeta+"\\"+"Fracture.inp");
        PrintWriter fracturePrintWriter = new PrintWriter(fractureFile);
        PrintHeading(fracturePrintWriter);
        fracturePrintWriter.println("*Heading\nFracture");
        fracturePrintWriter.println("** ----------------------------------------
----------------------");
        fracturePrintWriter.println("*parameter\n" +
                "** Fracture toughness:\n" +
                " GIc   = "+GIC+"\n" +
                " GIIc  = "+GIIC+"\n" +
                " GIIIc = "+GIIIC+"\n" +
                "** B-K parameter:\n" +
                " eta="+eta+"\n" +
                "***\n" +
                "*Preprint, echo = NO, model = NO, history = NO, contact = NO");

        fracturePrintWriter.println(
                "*Include, Input =Fracture_nodes.inp\n" +
                        "*Include, Input =Fracture_elems.inp\n" +
                        "*Include, Input =Fracture_elset.inp\n" +
                        "*Include, Input =Fracture_properties.inp\n"+
                        "*Include, Input =Fracture_nodesets.inp");

        fracturePrintWriter.println("** BNODES DECLARATION\n**");

        for(int i=0; i<bnodesfractura.size();i++){
            fracturePrintWriter.println("*NSET,NSET=BNODES"+(i+1));
            for(int j=0; j<bnodesfractura.get(i).size();j++){
                fracturePrintWriter.print(bnodesfractura.get(i).get(j)+", ");
                if((j+1)%8==0)
                    fracturePrintWriter.println();
            }
            if(bnodesfractura.get(i).size()%8!=0)
                fracturePrintWriter.println();
        }

        fracturePrintWriter.println("**ELSET DECLARATION\n**");
```

```
        for(int i=0; i<elementosbottom.size();i++){
            fracturePrintWriter.println("*Elset, elset=BOT"+(i+1));
            for(int j=0; j<elementosbottom.get(i).size();j++){
                fracturePrintWriter.print(elementosbottom.get(i).get(j)+", ");
                if((j+1)%8==0)
                    fracturePrintWriter.println();
            }
            if(elementosbottom.get(i).size()%8!=0)
                fracturePrintWriter.println();
        }

        for(int i=0; i<elementostop.size();i++){
            fracturePrintWriter.println("*Elset, elset=TOP"+(i+1));
            for(int j=0; j<elementostop.get(i).size();j++){
                fracturePrintWriter.print(elementostop.get(i).get(j)+", ");
                if((j+1)%8==0)
                    fracturePrintWriter.println();
            }
            if(elementostop.get(i).size()%8!=0)
                fracturePrintWriter.println();
        }

        for(int i=0; i<elementosbottom.size();i++){
            fracturePrintWriter.println("*SURFACE, NAME=BOT"+(i+1));
            fracturePrintWriter.println("BOT"+(i+1)+", S6");
        }

        for(int i=0; i<elementostop.size();i++){
            fracturePrintWriter.println("*SURFACE, NAME=TOP"+(i+1));
            fracturePrintWriter.println("TOP"+(i+1)+", S4");
        }

        for(int i=0; i<bnodesfractura.size();i++){
            fracturePrintWriter.println("*CONTACT PAIR, INTERACTION=FRACT"+(i+1)+",
ADJUST=BNODES"+(i+1)+"\n"+"BOT"+(i+1)+", TOP"+(i+1));
            fracturePrintWriter.println("*SURFACE INTERACTION,
NAME=FRACT"+(i+1)+"\n"+"1.0");
        }

        fracturePrintWriter.println("*INITIAL CONDITIONS,TYPE =CONTACT");

        for(int i=0; i<bnodesfractura.size();i++){
            fracturePrintWriter.println("BOT"+(i+1)+", TOP"+(i+1)+",
BNODES"+(i+1));
        }

        fracturePrintWriter.println("**\n" +
                "*STEP,NLGEOM, INC="+maximoIncremento+",convert sdi=no\n" +
                "*STATIC\n" +
                InitialTime+", "+TimeAnalysis+", "+MinTime+", "+MaxTime+"\n"+
                "*CONTROLS,PARAMETERS=TIME INCREMENTATION\n" +
                " , , , , , , 10000");

        for(int i=0; i<bnodesfractura.size();i++){
            fracturePrintWriter.println("*CONTACT
PRINT\n"+"*DEBOND,SLAVE=BOT"+(i+1)+",MASTER=TOP"+(i+1)+",FREQ=1\n"
                    +"*FRACTURE CRITERION,TYPE=VCCT,MIXED MODE BEHAVIOR=BK\n"+
                    "<GIc>,<GIIc>,<GIIIc>,<eta>");
        }

        fracturePrintWriter.println("*BOUNDARY\n" +
                "NodosDer, 1,1,"+desplazarProbeta+"\n" +
                "NodosIzq, 1,1,-"+desplazarProbeta+"\n" +
                "*NODE PRINT\n" +
                "RF,\n" +
                "*EL FILE\n" +
                "S, E\n" +
```

```
                "*NODE FILE\n" +
                "U, RF\n" +
                "*END STEP");

        PrintFooting(fracturePrintWriter);

        Process process = Runtime.getRuntime().exec("cmd /c start /wait cmd.exe /K
\"cd " + Carpeta + " && abaqus analysis job=Fracture int && exit");

        process.waitFor();
        process.destroy();
        cleanDirectoryAfter(Folder);
    }

    private static void cleanDirectoryAfter(File dir){
        for (File file : dir.listFiles()) {
            if (!file.getName().equals("Fracture.odb")){
                file.delete();
            }
        }
    }

    private static void cleanDirectoryBefore(File dir) {
        for (File file : dir.listFiles()) {
            file.delete();
        }
    }

    private static void PrintHeading(PrintWriter write){
        write.print("** Generated by : Diego S. Morales Arcos\n** -----------------
-----------------------------------------------\n**\n");
    }
    private static void PrintFooting(PrintWriter write){
        write.print("**\n** ------------------------------------------------------------
---------\n**");
        write.close();
    }

    private static Integer DuplicarNodo(Integer NodoADuplicar){
        int index = Nodos.indexOf(NodoADuplicar);
        Nodos.add(Nodos.get(Nodos.size()-1)+1);
        NodosX.add(NodosX.get(index));
        NodosY.add(NodosY.get(index));
        NodosZ.add(NodosZ.get(index));
        return Nodos.get(Nodos.size()-1);
    }

    static class KVpair<Key, E> {
         private Key k;
         private E e;

        KVpair()
        { k = null; e = null; }
        KVpair(Key kval, E eval)
        { k = kval; e = eval; }

        public Key key() { return k; }
        public E value() { return e; }
    }

}
```

ANEXO H: CÓDIGO JAVA PARA MODELOS DE ANÁLISIS DEL ESPACIADO

Ejecutor.java

```
import java.util.ArrayList;
import java.io.*;
public class Ejecutor {
    private static ArrayList<Integer> Nodos = new ArrayList<>();
    private static ArrayList<Double> NodosX = new ArrayList<>();
    private static ArrayList<Double> NodosY = new ArrayList<>();
    private static ArrayList<Double> NodosZ = new ArrayList<>();
    private static ArrayList<Integer> Elementos = new ArrayList<>();
    private static ArrayList<Integer> ElementosBase = new ArrayList<>();
    private static ArrayList<Integer> ElementosRecubrimiento = new ArrayList<>();
    private static ArrayList<Integer> ElementosIniciadores = new ArrayList<>();
    private static ArrayList<Integer> ElementosEdgeEffect = new ArrayList<>();
    private static ArrayList<Integer> Nodo1 = new ArrayList<>();
    private static ArrayList<Integer> Nodo2 = new ArrayList<>();
    private static ArrayList<Integer> Nodo3 = new ArrayList<>();
    private static ArrayList<Integer> Nodo4 = new ArrayList<>();
    private static ArrayList<Integer> Nodo5 = new ArrayList<>();
    private static ArrayList<Integer> Nodo6 = new ArrayList<>();
    private static ArrayList<Integer> Nodo7 = new ArrayList<>();
    private static ArrayList<Integer> Nodo8 = new ArrayList<>();


    public static void main(String[] args) throws Exception{

        double[] Iniciadores = {0.000001};
        double[] KICs = {11.17};
        int[] elementosyBases = {10};

        long inicialTime = System.nanoTime();
        for(int elementoyBase: elementosyBases){
            for (double Iniciador: Iniciadores) {
                for (double KIC : KICs) {
                    ExecutionWithParameters(0.03, 0.0032, 0.0125, 0.0002,
Iniciador,
                            1000, elementoyBase, KIC * 1E6, 5);
                    long endTime = System.nanoTime();
                    System.out.println("Total Computational time: "+((endTime-
inicialTime)*1e-9)/60);
                    inicialTime = System.nanoTime();
                }
            }
        }
    }

    private static void ExecutionWithParameters (double dimx, double dimyBase,
double dimz, double dimyRec, double dimyInic, int elementosx,
                                         int elementosyBase, double KIC, double
PorcentajeDesplazamiento)throws Exception{

        ArrayList<Integer> ElementosFracturaSup = new ArrayList<>();
        ArrayList<Integer> ElementosFracturaInf = new ArrayList<>();
        int value = 0;

        ElementosFracturaSup.add(13500);
        ElementosFracturaSup.add(13478);
        ElementosFracturaSup.add(13522);
        ElementosFracturaSup.add(13456);
        ElementosFracturaSup.add(13544);
        ElementosFracturaSup.add(13434);
```

```
        ElementosFracturaSup.add(13566);

        ElementosFracturaInf.add(500);
        ElementosFracturaInf.add(478);
        ElementosFracturaInf.add(522);
        ElementosFracturaInf.add(456);
        ElementosFracturaInf.add(544);
        ElementosFracturaInf.add(434);
        ElementosFracturaInf.add(566);

        //Elementos
        //Base
        int elementosz = 1; //Siempre un elemento

        //Recubrimiento
        int elementosyRec = 1;//Siempre un elemento
        int elementosyInic = 1; //Siempre un elemento

        int elementosedge = 400;

        int nodosboundaryeliminar =
elementosyRec+elementosyInic;//(elementosyRec+2*elementosyInic);(elementosyRec+elem
entosyInic);

        double YoungModBase = 207E9;
        double YoungModRecubrimiento = 560E9;
        double PoissonBase = 0.3;
        double PoissonRecubrimiento = 0.24;

        double KIIC = 4E6;
        double KIIIC = 4E6;
        double eta = 1.75;

        int maximoIncremento = 200;
        double TimeAnalysis = 1;
        double InitialTime = 0.005;
        double MaxTime = 0.01;
        double MinTime = 0.00000000015;

        ArrayList<KVpair<Double, Double>> propiedadesplasticas = new ArrayList<>();
        propiedadesplasticas.add(new KVpair<>(175E6,0.0));
        propiedadesplasticas.add(new KVpair<>(260E6,0.05));
        propiedadesplasticas.add(new KVpair<>(310E6,0.1));
        propiedadesplasticas.add(new KVpair<>(340E6,0.15));
        propiedadesplasticas.add(new KVpair<>(375E6,0.2));
        propiedadesplasticas.add(new KVpair<>(395E6,0.25));
        propiedadesplasticas.add(new KVpair<>(420E6,0.3));
        propiedadesplasticas.add(new KVpair<>(425E6,0.35));

        int elementosyTotal = (elementosyBase+2*(elementosyRec+elementosyInic));
        int auxelemsx = elementosx-elementosedge;
        int contadorelementos = elementosx*elementosyTotal*elementosz;
        double AumentoElementosRec = dimyRec/elementosyRec;
        double AumentoElementosBase = dimyBase/elementosyBase;
        double AumentoElementosInic = dimyInic/elementosyInic;
        double AumentoHorizonal = dimx/elementosx;
        double AumentoProfundidad = dimz/elementosz;

        double GIC = KIC*KIC/(YoungModRecubrimiento);
        double GIIC = KIIC*KIIC/(YoungModRecubrimiento);
        double GIIIC = KIIIC*KIIIC/(YoungModRecubrimiento);

        double desplazarProbeta = (PorcentajeDesplazamiento/100)*dimx/2;

        Nodos.clear();
        NodosX.clear();
        NodosY.clear();
        NodosZ.clear();
```

```
        Elementos.clear();
        ElementosBase.clear();
        ElementosRecubrimiento.clear();
        ElementosIniciadores.clear();
        ElementosEdgeEffect.clear();
        Nodo1.clear();
        Nodo2.clear();
        Nodo3.clear();
        Nodo4.clear();
        Nodo5.clear();
        Nodo6.clear();
        Nodo7.clear();
        Nodo8.clear();

        //Nodos
        int nodosCont = 1;
        for(int z=0; z<=elementosz; z++){
            for(int y=0; y<=elementosyInic; y++){
                for(int x=0; x<=elementosx; x++){
                    Nodos.add(nodosCont);
                    NodosX.add(x*AumentoHorizonal);
                    NodosY.add(y*AumentoElementosInic);
                    NodosZ.add(z*AumentoProfundidad);
                    nodosCont++;
                }
            }
            for(int y=0; y<elementosyRec; y++) {
                for (int x = 0; x <= elementosx; x++) {
                    Nodos.add(nodosCont);
                    NodosX.add(x * AumentoHorizonal);
                    NodosY.add(dimyInic + (y+1)*AumentoElementosRec);
                    NodosZ.add(z * AumentoProfundidad);
                    nodosCont++;
                }
            }
            for(int y=0; y<elementosyBase; y++){
                for(int x=0; x<=elementosx; x++){
                    Nodos.add(nodosCont);
                    NodosY.add(dimyInic+dimyRec+(y+1)*AumentoElementosBase);
                    NodosX.add(x*AumentoHorizonal);
                    NodosZ.add(z*AumentoProfundidad);
                    nodosCont++;
                }
            }
            for(int y=0; y<elementosyRec; y++){
                for(int x=0; x<=elementosx; x++){
                    Nodos.add(nodosCont);
                    NodosZ.add(z*AumentoProfundidad);
                    NodosX.add(x*AumentoHorizonal);

NodosY.add(dimyInic+dimyRec+dimyBase+(y+1)*AumentoElementosRec);
                    nodosCont++;
                }
            }
            for(int y=0; y<elementosyInic; y++){
                for(int x=0; x<=elementosx; x++){
                    Nodos.add(nodosCont);
                    NodosX.add(x*AumentoHorizonal);

NodosY.add(dimyInic+2*dimyRec+dimyBase+(y+1)*AumentoElementosInic);
                    NodosZ.add(z*AumentoProfundidad);
                    nodosCont++;
                }
            }
        }

        //Elementos
        int elementoscontar = 1;
```

```
for(int z=0; z<elementosz; z++){
    for(int y=0; y<elementosyInic; y++){
        for(int x=0; x<elementosx; x++){
            ElementosRecubrimiento.add(elementoscontar);
            ElementosIniciadores.add(elementoscontar);
            Elementos.add(elementoscontar);
            elementoscontar++;
        }
    }
    for(int y=0; y<elementosyRec; y++){
        for(int x=0; x<elementosx; x++){
            ElementosRecubrimiento.add(elementoscontar);
            Elementos.add(elementoscontar);
            elementoscontar++;
        }
    }
    for(int y=0; y<elementosyBase; y++){
        for(int x=0; x<elementosx; x++){
            Elementos.add(elementoscontar);
            ElementosBase.add(elementoscontar);
            elementoscontar++;
        }
    }
    for(int y=0; y<elementosyRec; y++){
        for(int x=0; x<elementosx; x++){
            Elementos.add(elementoscontar);
            ElementosRecubrimiento.add(elementoscontar);
            elementoscontar++;
        }
    }
    for(int y=0; y<elementosyInic; y++){
        for(int x=0; x<elementosx; x++){
            ElementosIniciadores.add(elementoscontar);
            ElementosRecubrimiento.add(elementoscontar);
            Elementos.add(elementoscontar);
            elementoscontar++;
        }
    }
}
for(int i=0; i<Elementos.size();i++){
    if(i<elementosedge) {
        ElementosEdgeEffect.add(Elementos.get(i));
    }
    if(i>=auxelemsx){
        ElementosEdgeEffect.add(Elementos.get(i));
    }
    if((i+1)%elementosx==0) {
        auxelemsx= auxelemsx + elementosx;
        elementosedge = elementosedge + elementosx;
    }
}

int nodosporcara = (elementosx+1)*(elementosyTotal+1);
nodosCont=1;
for(int z=0;z<elementosz;z++){
    for(int y=0; y<elementosyTotal; y++){
        for(int x=0; x<elementosx; x++){
            Nodo1.add(nodosCont);
            Nodo2.add(nodosCont+1);
            Nodo3.add(nodosCont+elementosx+2);
            Nodo4.add(nodosCont+elementosx+1);
            Nodo5.add(nodosCont+nodosporcara);
            Nodo6.add(nodosCont+1+nodosporcara);
            Nodo7.add(nodosCont+elementosx+2+nodosporcara);
            Nodo8.add(nodosCont+elementosx+1+nodosporcara);
            nodosCont++;
        }
        nodosCont++;
```

```
    }
}

ArrayList<Integer> BoundaryNodosIzquierda = new ArrayList<>();
ArrayList<Integer> BoundaryNodosDerecha = new ArrayList<>();

for(int i=0; i<Nodos.size();i++) {
    if (NodosX.get(i) == 0) {
        BoundaryNodosIzquierda.add(Nodos.get(i));
    }
    if (NodosX.get(i) == dimx) {
        BoundaryNodosDerecha.add(Nodos.get(i));
    }
}

for(int i=0; i<nodosboundaryeliminar;i++){
    int sizetemp = BoundaryNodosDerecha.size()/2;
    BoundaryNodosIzquierda.remove(sizetemp-1);
    BoundaryNodosDerecha.remove(sizetemp-1);
    BoundaryNodosIzquierda.remove(sizetemp-1);
    BoundaryNodosDerecha.remove(sizetemp-1);
    BoundaryNodosIzquierda.remove(0);
    BoundaryNodosDerecha.remove(0);
    BoundaryNodosIzquierda.remove(BoundaryNodosIzquierda.size()-1);
    BoundaryNodosDerecha.remove(BoundaryNodosDerecha.size()-1);
}

//Generacion del crack de fractura Inferior
for(int i=0; i<ElementosFracturaInf.size();i++){
    int index = Elementos.indexOf(ElementosFracturaInf.get(i));
    int nuevoindicedenodo = DuplicarNodo(Nodo2.get(index));
    Nodo2.remove(index);
    Nodo2.add(index,nuevoindicedenodo);
    nuevoindicedenodo = DuplicarNodo(Nodo6.get(index));
    Nodo6.remove(index);
    Nodo6.add(index,nuevoindicedenodo);

}

//Generacion del crack de fractura Superior
for(int i=0; i<ElementosFracturaSup.size();i++){
    int index2 = Elementos.indexOf(ElementosFracturaSup.get(i));
    int nuevoindicedenodo2 = DuplicarNodo(Nodo3.get(index2));
    Nodo3.remove(index2);
    Nodo3.add(index2,nuevoindicedenodo2);
    nuevoindicedenodo2 = DuplicarNodo(Nodo7.get(index2));
    Nodo7.remove(index2);
    Nodo7.add(index2,nuevoindicedenodo2);
}

ArrayList<ArrayList<Integer>> elementostop = new ArrayList<>();
ArrayList<ArrayList<Integer>> elementosbottom = new ArrayList<>();
ArrayList<ArrayList<Integer>> bnodesfractura = new ArrayList<>();

int contador = 0;

//Elementos inferiores
for(int i=0; i<ElementosFracturaInf.size();i++){
        if((contador+1)>elementostop.size())
            elementostop.add(new ArrayList<>());
        if((contador+1)>elementosbottom.size())
            elementosbottom.add(new ArrayList<>());
        if((contador+1)>bnodesfractura.size())
            bnodesfractura.add(new ArrayList<>());

        elementostop.get(contador).add(ElementosFracturaInf.get(i));
        elementosbottom.get(contador).add(ElementosFracturaInf.get(i)+1);
```

```
                int index = Elementos.indexOf(ElementosFracturaInf.get(i));

                bnodesfractura.get(contador).add(Nodo3.get(index));
                bnodesfractura.get(contador).add(Nodo7.get(index));

                int nuevoindicedenodo = DuplicarNodo(Nodo3.get(index));
                int nuevoindicedenodo2 = DuplicarNodo(Nodo7.get(index));

                Nodo2.remove(index+elementosx);
                Nodo2.add(index+elementosx,nuevoindicedenodo);
                Nodo6.remove(index+elementosx);
                Nodo6.add(index+elementosx,nuevoindicedenodo2);

                Nodo3.remove(index);
                Nodo3.add(index,nuevoindicedenodo);
                Nodo7.remove(index);
                Nodo7.add(index,nuevoindicedenodo2);

                bnodesfractura.get(contador).add(Nodo3.get(index));
                bnodesfractura.get(contador).add(Nodo7.get(index));
                contador++;
        }

        //Elementos superiores
        for(int i=0;i<ElementosFracturaSup.size();i++) {
                if((contador+1)>elementostop.size())
                    elementostop.add(new ArrayList<>());
                if((contador+1)>elementosbottom.size())
                    elementosbottom.add(new ArrayList<>());
                if((contador+1)>bnodesfractura.size())
                    bnodesfractura.add(new ArrayList<>());

                elementostop.get(contador).add(ElementosFracturaSup.get(i));
                elementosbottom.get(contador).add(ElementosFracturaSup.get(i)+1);

                int index = Elementos.indexOf(ElementosFracturaSup.get(i)-
elementosx);

                bnodesfractura.get(contador).add(Nodo3.get(index));
                bnodesfractura.get(contador).add(Nodo7.get(index));

                int nuevoindicedenodo = DuplicarNodo(Nodo3.get(index));
                int nuevoindicedenodo2 = DuplicarNodo(Nodo7.get(index));

                Nodo2.remove(index+elementosx);
                Nodo2.add(index+elementosx,nuevoindicedenodo);
                Nodo6.remove(index+elementosx);
                Nodo6.add(index+elementosx,nuevoindicedenodo2);

                Nodo3.remove(index);
                Nodo3.add(index,nuevoindicedenodo);
                Nodo7.remove(index);
                Nodo7.add(index,nuevoindicedenodo2);

                bnodesfractura.get(contador).add(Nodo3.get(index));
                bnodesfractura.get(contador).add(Nodo7.get(index));
                contador++;
        }

        //Fracture to avoid
        for(int i=0; i<value;i++) {
            bnodesfractura.remove(bnodesfractura.size()/2);
            elementosbottom.remove(elementosbottom.size()/2);
            elementostop.remove(elementostop.size()/2);

            bnodesfractura.remove(0);
            elementosbottom.remove(0);
            elementostop.remove(0);
```

```
        }


        String CarpetaRoot = "C:\\Users\\diego\\Desktop\\Tesis No Drive\\Output
Files";
        File RootFolder = new File(CarpetaRoot);
        if(!RootFolder.isDirectory())
            RootFolder.mkdir();

        String CarpetaRoot1 = CarpetaRoot+"\\Dimx "+dimx+" DimyBase "+dimyBase+"
Dimz "+dimz+" DimyRec "+dimyRec+" Desp "+PorcentajeDesplazamiento;
        File RootFolder1 = new File(CarpetaRoot1);
        if(!RootFolder1.isDirectory())
            RootFolder1.mkdir();

        String CarpetaInic = CarpetaRoot1+"\\"+"Iniciador " +
Integer.toString((int)(dimyInic*1E6));
        File InicFolder = new File(CarpetaInic);
        if(!InicFolder.isDirectory())
            InicFolder.mkdir();

        String Carpeta = CarpetaInic+"\\"+"KIC "+Integer.toString((int)(KIC*1E-6));
        File Folder = new File(Carpeta);
        if(!Folder.isDirectory())
            Folder.mkdir();

        cleanDirectoryBefore(Folder);

        //Imprimiendo los archivos

        File nodosFile = new File(Carpeta+"\\"+"Fracture_nodes.inp");
        PrintWriter nodosPrintWriter = new PrintWriter(nodosFile);
        PrintHeading(nodosPrintWriter);
        nodosPrintWriter.println("*Node");
        for(int i=0; i<Nodos.size();i++){
            nodosPrintWriter.println(Nodos.get(i)+", "+NodosX.get(i)+",
"+NodosY.get(i)+", "+NodosZ.get(i));
        }
        PrintFooting(nodosPrintWriter);

        File elemsFile = new File(Carpeta+"\\"+"Fracture_elems.inp");
        PrintWriter elemsPrintWriter = new PrintWriter(elemsFile);
        PrintHeading(elemsPrintWriter);
        elemsPrintWriter.println("*Element, type=C3D8");
        for(int i=0; i<Elementos.size();i++){
            elemsPrintWriter.println(Elementos.get(i)+", "+Nodo1.get(i)+",
"+Nodo2.get(i)+", "+Nodo3.get(i)+", "+Nodo4.get(i)+", "+Nodo5.get(i)+",
"+Nodo6.get(i)+", "+Nodo7.get(i)+", "+Nodo8.get(i));
        }
        PrintFooting(elemsPrintWriter);

        File nodesetsFile = new File(Carpeta+"\\"+"Fracture_nodesets.inp");
        PrintWriter nodesetsPrintWriter = new PrintWriter(nodesetsFile);
        PrintHeading(nodesetsPrintWriter);

        nodesetsPrintWriter.println("*NSET, NSET=NodosIzq");
        for(int i=0; i<BoundaryNodosIzquierda.size();i++){
            nodesetsPrintWriter.print(BoundaryNodosIzquierda.get(i)+", ");
            if((i+1)%8==0)
                nodesetsPrintWriter.println();
        }
        if(BoundaryNodosIzquierda.size()%8!=0)
            nodesetsPrintWriter.println();

        nodesetsPrintWriter.println("*NSET, NSET=NodosDer");
        for(int i=0; i<BoundaryNodosDerecha.size();i++){
            nodesetsPrintWriter.print(BoundaryNodosDerecha.get(i)+", ");
            if((i+1)%8==0)
```

```
                nodesetsPrintWriter.println();
        }
        if(BoundaryNodosDerecha.size()%8!=0)
            nodesetsPrintWriter.println();

        PrintFooting(nodesetsPrintWriter);

        File elsetFile = new File(Carpeta+"\\"+"Fracture_elset.inp");
        PrintWriter elsetPrintWriter = new PrintWriter(elsetFile);
        PrintHeading(elsetPrintWriter);
        elsetPrintWriter.println("*Elset, elset=cube, generate\n1, "+
contadorelementos +", 1");

        elsetPrintWriter.println("*Elset, elset=Base_set");
        for(int i=0; i<ElementosBase.size();i++){
            elsetPrintWriter.print(ElementosBase.get(i)+", ");
            if((i+1)%8==0)
                elsetPrintWriter.println();
        }
        if(ElementosBase.size()%8!=0)
            elsetPrintWriter.println();

        elsetPrintWriter.println("*Elset, elset=Recubrimiento_set");
        for(int i=0; i<ElementosRecubrimiento.size();i++){
            elsetPrintWriter.print(ElementosRecubrimiento.get(i)+", ");
            if((i+1)%8==0)
                elsetPrintWriter.println();
        }
        if(ElementosRecubrimiento.size()%8!=0)
            elsetPrintWriter.println();

        elsetPrintWriter.println("*Elset, elset=Iniciadores_set");
        for(int i=0; i<ElementosIniciadores.size();i++){
            elsetPrintWriter.print(ElementosIniciadores.get(i)+", ");
            if((i+1)%8==0)
                elsetPrintWriter.println();
        }
        if(ElementosIniciadores.size()%8!=0)
            elsetPrintWriter.println();

        elsetPrintWriter.println("*Elset, elset=EdgeEffect_set");
        for(int i=0; i<ElementosEdgeEffect.size();i++){
            elsetPrintWriter.print(ElementosEdgeEffect.get(i)+", ");
            if((i+1)%8==0)
                elsetPrintWriter.println();
        }
        if(ElementosEdgeEffect.size()%8!=0)
            elsetPrintWriter.println();

        PrintFooting(elsetPrintWriter);

        File propertiesFile = new File(Carpeta+"\\"+"Fracture_properties.inp");
        PrintWriter propertiesPrintWriter = new PrintWriter(propertiesFile);
        PrintHeading(propertiesPrintWriter);
        propertiesPrintWriter.println("*Solid Section, elset=Base_set,
material=Base\n1.\n*Solid Section, elset=Recubrimiento_set,
material=Recubrimiento\n1.");
        propertiesPrintWriter.println("*Material, name=Base\n" + "*ELASTIC\n" +
YoungModBase+", "+PoissonBase+"\n" + "*PLASTIC");

        for (KVpair<Double, Double> propiedadesplastica : propiedadesplasticas) {
            propertiesPrintWriter.println(propiedadesplastica.key() + ", " +
propiedadesplastica.value());
        }

        propertiesPrintWriter.println("*Material, name=Recubrimiento\n" +
"*ELASTIC\n" + YoungModRecubrimiento+", "+PoissonRecubrimiento);
        PrintFooting(propertiesPrintWriter);
```

```
        File fractureFile = new File(Carpeta+"\\"+"Fracture.inp");
        PrintWriter fracturePrintWriter = new PrintWriter(fractureFile);
        PrintHeading(fracturePrintWriter);
        fracturePrintWriter.println("*Heading\nFracture");
        fracturePrintWriter.println("** ----------------------------------------
----------------------");
        fracturePrintWriter.println("*parameter\n" +
                "** Fracture toughness:\n" +
                " GIc   = "+GIC+"\n" +
                " GIIc  = "+GIIC+"\n" +
                " GIIIc = "+GIIIC+"\n" +
                "** B-K parameter:\n" +
                " eta="+eta+"\n" +
                "***\n" +
                "*Preprint, echo = NO, model = NO, history = NO, contact = NO");

        fracturePrintWriter.println(
                "*Include, Input =Fracture_nodes.inp\n" +
                        "*Include, Input =Fracture_elems.inp\n" +
                        "*Include, Input =Fracture_elset.inp\n" +
                        "*Include, Input =Fracture_properties.inp\n"+
                        "*Include, Input =Fracture_nodesets.inp");

        fracturePrintWriter.println("** BNODES DECLARATION\n**");

        for(int i=0; i<bnodesfractura.size();i++){
            fracturePrintWriter.println("*NSET,NSET=BNODES"+(i+1));
            for(int j=0; j<bnodesfractura.get(i).size();j++){
                fracturePrintWriter.print(bnodesfractura.get(i).get(j)+", ");
                if((j+1)%8==0)
                    fracturePrintWriter.println();
            }
            if(bnodesfractura.get(i).size()%8!=0)
                fracturePrintWriter.println();
        }

        fracturePrintWriter.println("**ELSET DECLARATION\n**");

        for(int i=0; i<elementosbottom.size();i++){
            fracturePrintWriter.println("*Elset, elset=BOT"+(i+1));
            for(int j=0; j<elementosbottom.get(i).size();j++){
                fracturePrintWriter.print(elementosbottom.get(i).get(j)+", ");
                if((j+1)%8==0)
                    fracturePrintWriter.println();
            }
            if(elementosbottom.get(i).size()%8!=0)
                fracturePrintWriter.println();
        }

        for(int i=0; i<elementostop.size();i++){
            fracturePrintWriter.println("*Elset, elset=TOP"+(i+1));
            for(int j=0; j<elementostop.get(i).size();j++){
                fracturePrintWriter.print(elementostop.get(i).get(j)+", ");
                if((j+1)%8==0)
                    fracturePrintWriter.println();
            }
            if(elementostop.get(i).size()%8!=0)
                fracturePrintWriter.println();
        }

        for(int i=0; i<elementosbottom.size();i++){
            fracturePrintWriter.println("*SURFACE, NAME=BOT"+(i+1));
            fracturePrintWriter.println("BOT"+(i+1)+", S6");
        }

        for(int i=0; i<elementostop.size();i++){
            fracturePrintWriter.println("*SURFACE, NAME=TOP"+(i+1));
```

```
            fracturePrintWriter.println("TOP"+(i+1)+", S4");
        }

        for(int i=0; i<bnodesfractura.size();i++){
            fracturePrintWriter.println("*CONTACT PAIR, INTERACTION=FRACT"+(i+1)+",
ADJUST=BNODES"+(i+1)+"\n"+"BOT"+(i+1)+", TOP"+(i+1));
            fracturePrintWriter.println("*SURFACE INTERACTION,
NAME=FRACT"+(i+1)+"\n"+"1.0");
        }

        fracturePrintWriter.println("*INITIAL CONDITIONS,TYPE =CONTACT");

        for(int i=0; i<bnodesfractura.size();i++){
            fracturePrintWriter.println("BOT"+(i+1)+", TOP"+(i+1)+",
BNODES"+(i+1));
        }

        fracturePrintWriter.println("**\n" +
                "*STEP,NLGEOM, INC="+maximoIncremento+",convert sdi=no\n" +
                "*STATIC\n" +
                InitialTime+", "+TimeAnalysis+", "+MinTime+", "+MaxTime+"\n"+
                "*CONTROLS,PARAMETERS=TIME INCREMENTATION\n" +
                " , , , , , , 10000");

        for(int i=0; i<bnodesfractura.size();i++){
            fracturePrintWriter.println("*CONTACT
PRINT\n"+"*DEBOND,SLAVE=BOT"+(i+1)+",MASTER=TOP"+(i+1)+",FREQ=1\n"
                    +"*FRACTURE CRITERION,TYPE=VCCT,MIXED MODE BEHAVIOR=BK\n"+
                    "<GIc>,<GIIc>,<GIIIc>,<eta>");
        }

        fracturePrintWriter.println("*BOUNDARY\n" +
                "NodosDer, 1,1,"+desplazarProbeta+"\n" +
                "NodosIzq, 1,1,-"+desplazarProbeta+"\n" +
                "*NODE PRINT\n" +
                "RF,\n" +
                "*EL FILE\n" +
                "S, E\n" +
                "*NODE FILE\n" +
                "U, RF\n" +
                "*END STEP");

        PrintFooting(fracturePrintWriter);

        Process process = Runtime.getRuntime().exec("cmd /c start /wait cmd.exe /K
\"cd " + Carpeta + " && abaqus analysis job=Fracture int && exit");

        process.waitFor();
        process.destroy();
        cleanDirectoryAfter(Folder);
    }

    private static void cleanDirectoryAfter(File dir){
        for (File file : dir.listFiles()) {
            if (!file.getName().equals("Fracture.odb")){
                file.delete();
            }
        }
    }

    private static void cleanDirectoryBefore(File dir){
        for (File file : dir.listFiles()) {
                file.delete();
        }
    }

    private static void PrintHeading(PrintWriter write){
        write.print("** Generated by : Diego S. Morales Arcos\n** -----------------
```

```
-----------------------------------------------\n**\n");
    }
    private static void PrintFooting(PrintWriter write){
        write.print("**\n** ------------------------------------------------------------
---------\n**");
        write.close();
    }

    private static Integer DuplicarNodo(Integer NodoADuplicar){
        int index = Nodos.indexOf(NodoADuplicar);
        Nodos.add(Nodos.get(Nodos.size()-1)+1);
        NodosX.add(NodosX.get(index));
        NodosY.add(NodosY.get(index));
        NodosZ.add(NodosZ.get(index));
        return Nodos.get(Nodos.size()-1);
    }

    static class KVpair<Key, E> {
        private Key k;
        private E e;

        /** Constructors */
        KVpair()
        { k = null; e = null; }
        KVpair(Key kval, E eval)
        { k = kval; e = eval; }

        /** Data member access functions */
        public Key key() { return k; }
        public E value() { return e; }
    }

}
```