**Zusammenfassung zu EMF Refactor:**

**EMF Refactor** ist eine Erweiterung für Eclipse, welche in diesem Fortgeschrittenen-Praktikum erweitert werden soll. Dabei bietet EMF Refactor verschiedene Funktionen an, um UML-Modelle auf schlechte Struktur zu überprüfen und entsprechend zu bearbeiten, also zu refactorn.

**UML-Modelle** sind Diagramme, die zur besseren Verständlichkeit eines Projekts beitragen können. Häufig werden UML-Diagramme während der Planungsphase des Projekts erstellt, bevor die eigentliche Implementierungsphase beginnt. Das bekannteste Beispiel für ein UML-Diagramm ist das Klassendiagramm, welches die im Projekt verwendeten Klassen zusammen mit ihren Attributen modelliert und Beziehungen unter den Klassen wie beispielsweise Vererbung darstellt. Es zählt daher zu den Strukturdiagrammen. Allerdings gibt es auch viele andere UML-Modelle wie beispielsweise Aktivitätsdiagramme oder Anwendungsfalldiagramme, die das Verhalten eines Programms und nicht dessen Struktur modellieren.

**Refactoring** bezeichnet die manuelle oder automatische Verbesserung von bestehendem Code oder von bestehenden Modellen, wobei die eigentlichen Funktionalitäten dieses Codes oder dieser Modelle beibehalten werden. Dabei werden insbesondere die Verständlichkeit und Wartbarkeit der Modelle, die bearbeitet wurden, verbessert. In einigen Fällen kann man jedoch durch Refactoring auch eine Leistungssteigerung erreichen, weshalb sich Refactoring für alle größeren Projekte lohnen kann. Man benutzt Smells und Metriken, um festzustellen, ob ein Projekt durch Refactoring verbessert werden kann.

**Smells** sind eigentlich richtige Codestellen oder Strukturen in UML-Modellen, die ihre angedachte Funktion wie geplant erfüllen, aber nicht optimal entwickelt wurden und somit möglicherweise für andere Entwickler schwer verständlich oder umständlich zu ändern sind. Ein häufig auftretender Smell ist z.B. redundanter Code, also Programmcode, der an mehreren Stellen in einem Programm vorkommt. Ein Beispiel hierfür wäre Code, der eine Temperaturangabe von Celsius nach Fahrenheit umrechnet und umgekehrt. Dieser Smell kann verbessert werden, indem dieser Code in eine Funktion ausgelagert wird, welche an den entsprechenden Stellen im Programm aufgerufen wird.

Bei UML-Modellen, besonders Klassendiagrammen, gibt es häufig redundante Attribute von einzelnen Klassen wie z.B. das Attribut Geschwindigkeit bei den Klassen "Auto", "LKW" und "Motorrad". Durch diese Redundanz muss jede Klasse einzeln geändert werden, wenn etwas an dem Attribut Geschwindigkeit verändert werden soll. Eine geschicktere Lösung dafür wäre es, eine Oberklasse "Fahrzeug" einzuführen, die das Attribut Geschwindigkeit enthält. Von dieser Oberklasse Fahrzeug erben die Klassen "Auto", "LKW" und "Motorrad" ihre Attribute, wodurch im Falle einer nötigen Veränderung das Modell nur an einer Stelle, nämlich in der Oberklasse "Fahrzeug", angepasst werden muss.

**Metriken**, auch Softwaremetriken, sind (mathematische) Funktionen, die Maßzahlen berechnen, mit denen festgestellt werden kann, ob sich ein Refactoring für ein gegebenes Projekt lohnt. Eine sehr bekannte Metrik ist LoC, Lines of Code, die die Gesamtanzahl der Codezeilen in einem Projekt beschreibt. Üblicherweise kann mit 10 - 50 Codezeilen pro Programmierer pro Tag gerechnet werden. Dabei wird unter anderem auch zwischen LoC (alle Zeilen) und SLoC (nur Programmcode ohne Leerzeilen oder Kommentare) unterschieden. Liegt die berechnete Maßzahl für LoC weit über diesem Mittelwert, ist es wahrscheinlich, dass entweder redundanter Code benutzt wurde oder dieser zu ausführlich mit Kommentaren dokumentiert wurde. Falls die berechnete Maßzahl für LoC deutlich unter diesem Mittelwert liegt, wurde der Code wahrscheinlich nicht ausreichend dokumentiert. In beiden Fällen wäre es gut, über ein Refactoring zur Verbesserung der Lesbarkeit des Codes nachzudenken, um den Einstieg in das Projekt für neue Programmierer zu erleichtern.

Metriken können auch für UML-Modelle berechnet werden. Betrachtet man z.B. das eben genannte Fahrzeug-Beispiel kann man die Metrik NATP, also die Anzahl der Attribute der Klassen in einem Paket, berechnen. So kann man einen Überblick erhalten, ob redundante Attribute wahrscheinlich sind. Diese Zahl kann man senken, indem man wie in dem eben genannten Fahrzeug-Beispiel eine Oberklasse erstellt, welche die gemeinsamen Attribute der Klassen enthält und an die entsprechenden Klassen vererbt.

Bei UML-Modellen gibt es aber nicht nur Klassendiagramme zu beachten, für die bereits sehr viele Smells und Metriken in EMF Refactor implementiert sind. Es gibt auch andere wichtige Diagrammarten wie Anwendungsfalldiagramme, Aktivitätsdiagramme oder Zustandsdiagramme, bei denen jeweils eigene Smells und Metriken beachtet oder zumindest anders behandelt werden müssen, als bei einem Klassendiagramm. Diese Diagrammarten sind noch nicht vollständig in EMF Refactor enthalten und müssen daher noch implementiert werden.

Ist eine Diagrammart zusammen mit den jeweiligen Metriken und Smells fertig in EMF Refactor implementiert, können diese Metriken berechnet und Smells gefunden werden, die dann durch automatisches Refactoring verbessert werden können.