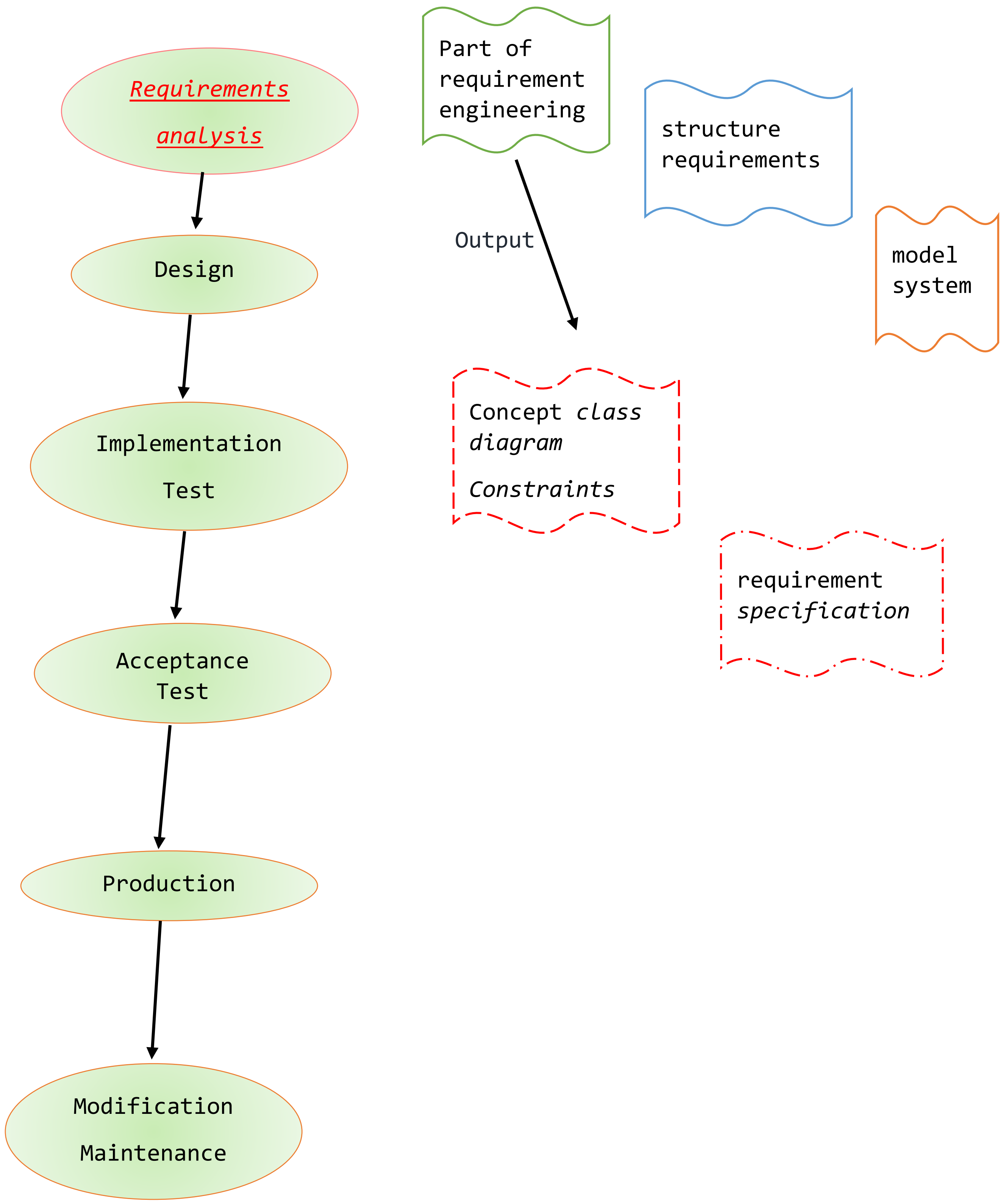
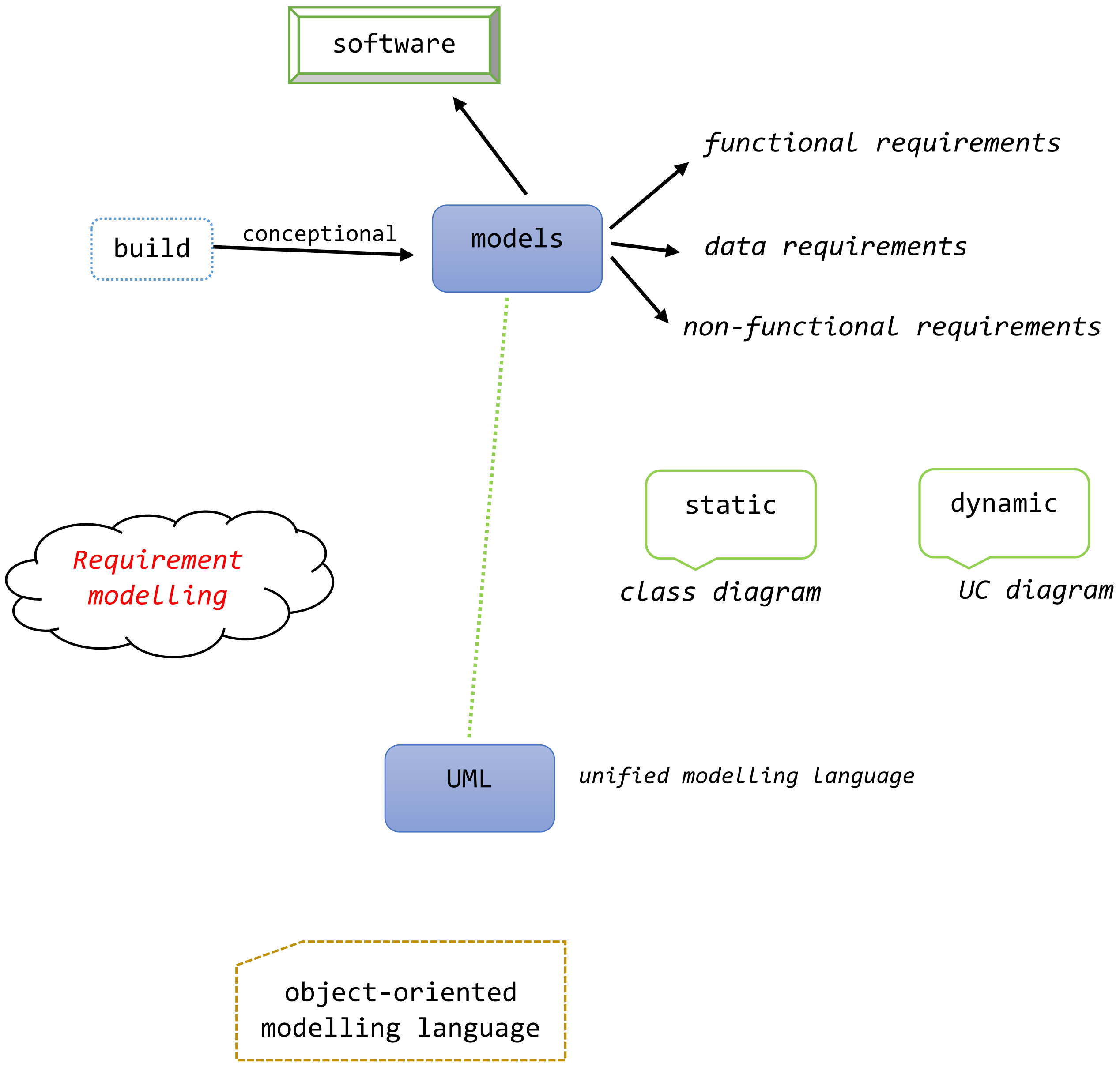
Development Process




software
functional requirements
build
conceptional
models
data requirements
non-functional requirements
static
dynamic
Requirement modelling
class diagram
UC diagram
UML
unified modelling language
object-oriented modelling language

Class diagram

model *classes* + *associations*

develop → analysis

refine → design

Analysis class diagram $\xrightarrow{\text{model}}$ domain entities: query, match, keyword

Design *class diagram model:*

- operations + (more) attributes: entities in fine detail
- additional software entities

Unified modelling language (UML)	*Entity relationship diagram (ERD)*
class: ✓ attributes ✓ operations (methods)	entity
association: ✓ cardinality	relationship
association class	associative entity
constraint	Domain Constraint,…

Construct:

1. **Map** *entities* → *domain classes*
2. *relationships* → *associations*
   cardinality constraints → class cardinalities
3. *associative entities* → *association classes*
4. **Write** *constraint statements*

KEngine entities

Document: title, url, body

Word: label

Keyword

NonKeyword

Query

Match: document, sum-freq

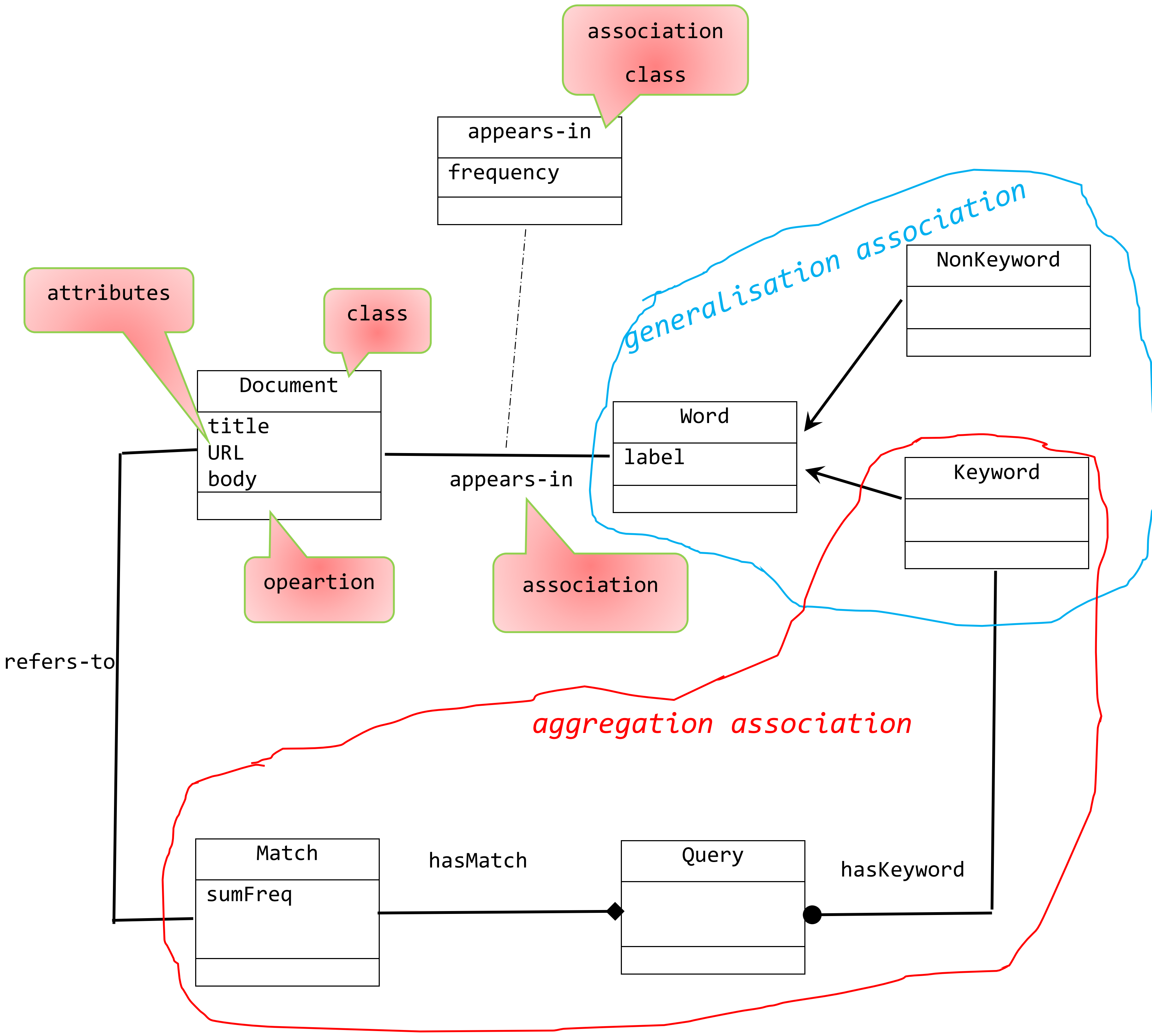
KEngine relationships

appears-in(Keyword, Document): frequency

hasKeyword(Query, Keyword)

hasMatch(Query, Match)

refers-to(Match, Document)

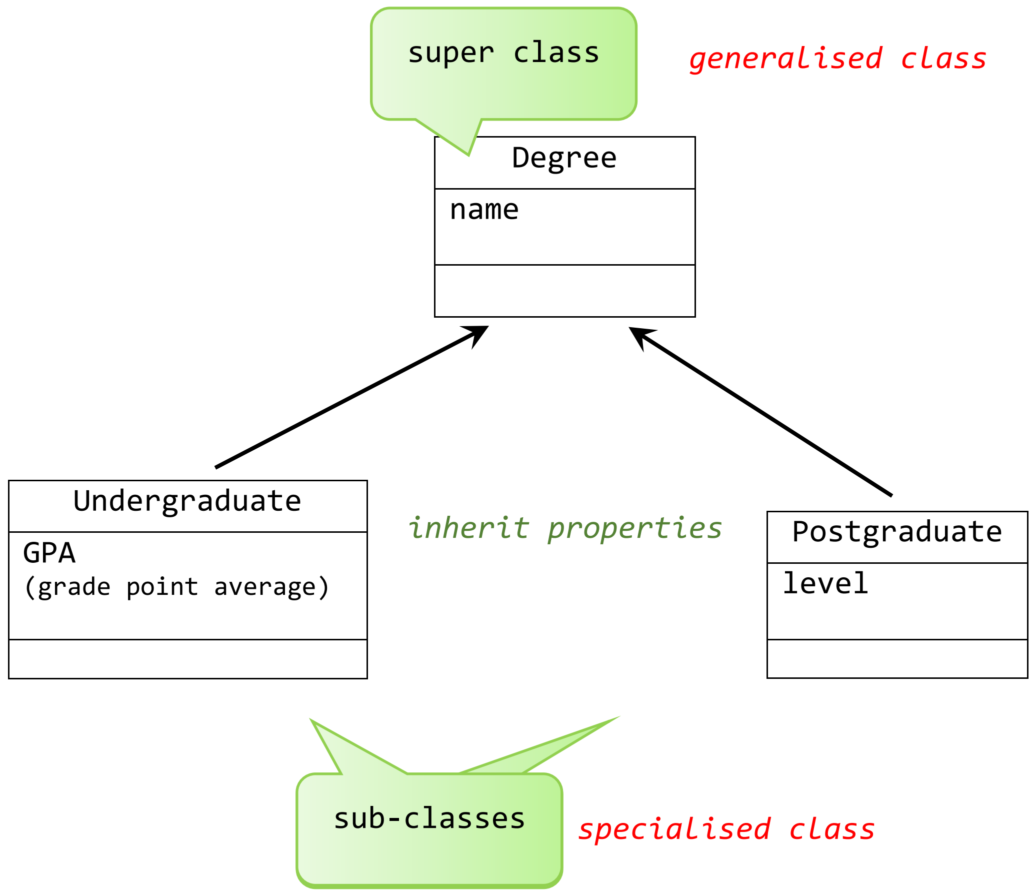

association class
appears-in
frequency
attributes
class
Document
title
URL
body
generalisation association
NonKeyword
Word
label
Keyword
appears-in
opeartion
association
refers-to
aggregation association
Match
sumFreq
hasMatch
Query
hasKeyword

Enhanced association

- ✓ Generalisation

  model type hierarchy

  gr classes (common characteristics) $\xrightarrow{\text{form}}$ more general



- ✓ Aggregation

  model a composition relationship

  eg: query, match, keyword

Constraint language

formal + informal

UML model

✓ natural language description (required)

appears-in: frequency *is the count of occurrences of a word in a given document*

✓ logic statement: constraint → concerned model elements

```
for all d: Document, w: Word  [
        appears-in(w, d) => appears-in(w, d):
                               frequency = |{k| k in d.body, k = w}|
]
```

Attribute constraints

```
appears-in: frequency
Match     : sumFreq
```

appears-in.frequency constraint

appears-in: frequency is the count of occurrences of a word in a given document

```
for all d: Document, w: Word   [
     appears-in(w, d)  => appears-in(w,d):
                          frequency = |{k| k in d.body, k=w}|
]
```

Match.sumFreq constraint

Match: sumFreq is the total count of occurrences of all keywords in that document

```
for all q: Query, m: Match, d: Document   [
     hasMatch(q, m)   /\   refers-to(m, d)
     => m.sumFreq = sum(appears-in(w, d): frequency),
     for all w in q
]
```

Association constraints

Document match *Query*

A document matches a query if it contains all query keywords

```
for all q: Query, m: Match, d: Document  [
      hasMatch(q, m)   /\    refers-to(m, d)
      => for all w in q (w in d.body)
]
```

Matches' ordering

Matches are ordered by sum of keyword counts

```
for all q: Query, m1, m2: Match  [
      hasMatch(q, m1)   /\   hasMatch(q, m2)   /\
      m1.sumFreq >= m2.sumFreq      // result in desc. sort
      => hasMatch(q, m1).index < hasMatch(q, m2).index
            // mean word before first
]
```

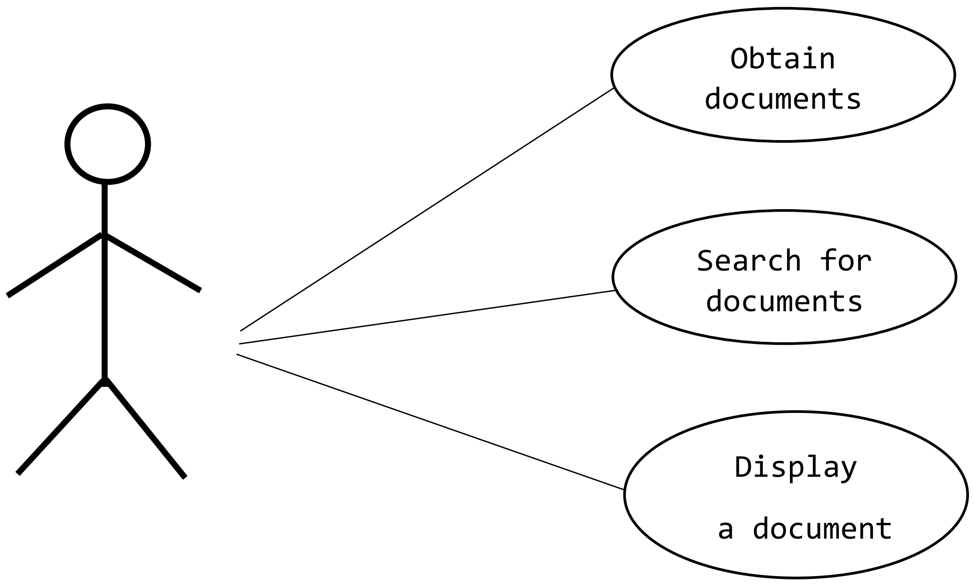
Use Case diagram

show actor interactions

$$\begin{tikzcd}
 & \text{actor} \arrow[r, "interact"] & \text{many UCs} \\
\text{many-to-many interactions} \arrow[ur, no head] \arrow[dr, no head] & & \\
 & \text{UC} \arrow[r, "involve"] & \text{many actors}
\end{tikzcd}$$

high-level abstraction (system): only functionality description

Graphical notation (KEngine System)



Requirement specification

System: high-level specification *(high-level abstraction)*

data + function models

what system provide

generate *design specification*

language

- design specification language (simplified form)
- model elements
- @checks (@~~requires~~): input + model constraints

  ~~@modifies~~: operation *always* modifies state (system)

  @effects

system specification

- system – *abstraction*
- UCs – *operations* (system)

procedural specification

- ~~return~~
- ~~exceptions~~
- total
- preserve model constraints

Engine

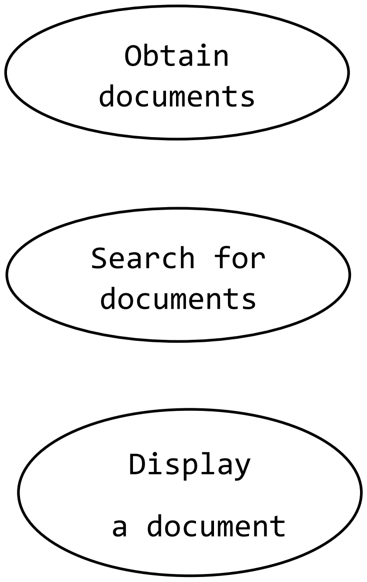
ξ startEngine

ξ addDocuments

ξ query

ξ queryMore

ξ findDoc



Engine specification

```
/**
  @overview
      represents keyword search engines
      A engine holds a mutable collection of documents – obtained from
      some given URLs
      The engine is able to pocess a keyword query to search for
      documents – contain keywords

      The matching documents are ranked based on frequencies of
      keywords found in them
      The engine has a private file – contains list of uninteresting
      words
*/
class KEngine   {
}
```

```
startEngine
/**
@overview …(omitted)…
*/
class Engine {
      /**
       @effects
        Starts the engine running with NonKeyWord containing the words
        in private file
        All other sets are empty.
      */
      static startEngine()



addDocuments
/**
 @checks u does not name a site in URL && u names a site – provide
documents

 @effects
 adds u to URL
 adds documents at site u – new titles to Document
 If keyword – non-empty
      adds any documents – match keywords to Match
*/
addDocuments(String u)
```

```
query
/**
 @checks: w is not in NonKeyword

 @effects
  Sets Keyword = {w}
  makes Match contain documents – match w, ordered as required
*/
query(String w)
```

```
queryMore
/**
 @checks Keyword != {}
         w not in NonKeyword
         w not in Keyword

 @effects
  Adds w to Keyword
  makes Match – documents already in Match – additionally match w
  Orders Match properly
*/
queryMore(String w)
```

```
finDoc
/**
 @checks   t is in titles

 @effects
      return d in Document s.t
      d’s title = t
*/
findDoc(String t)
}  // end Engine
```