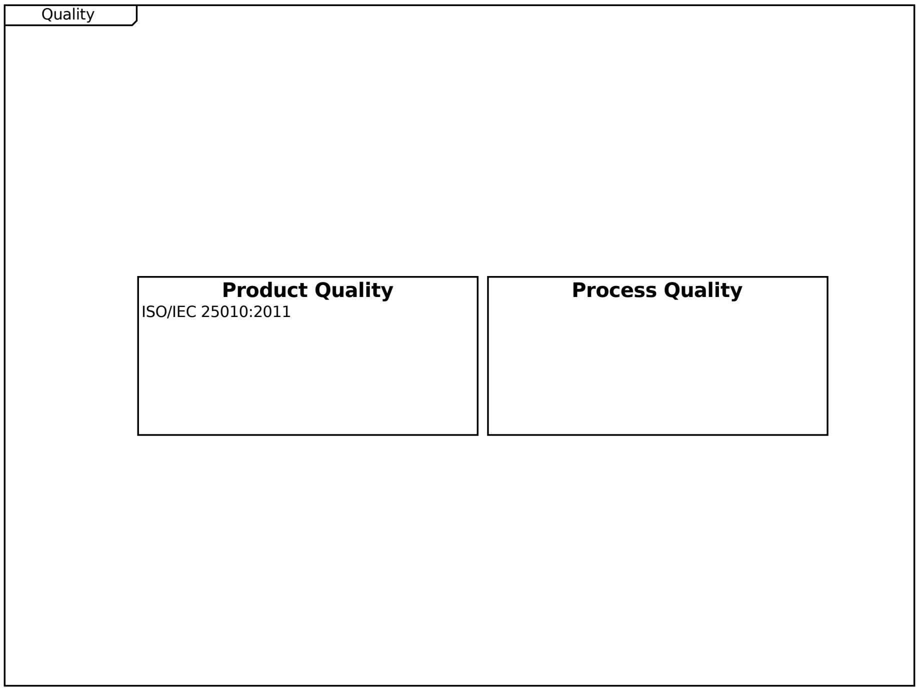
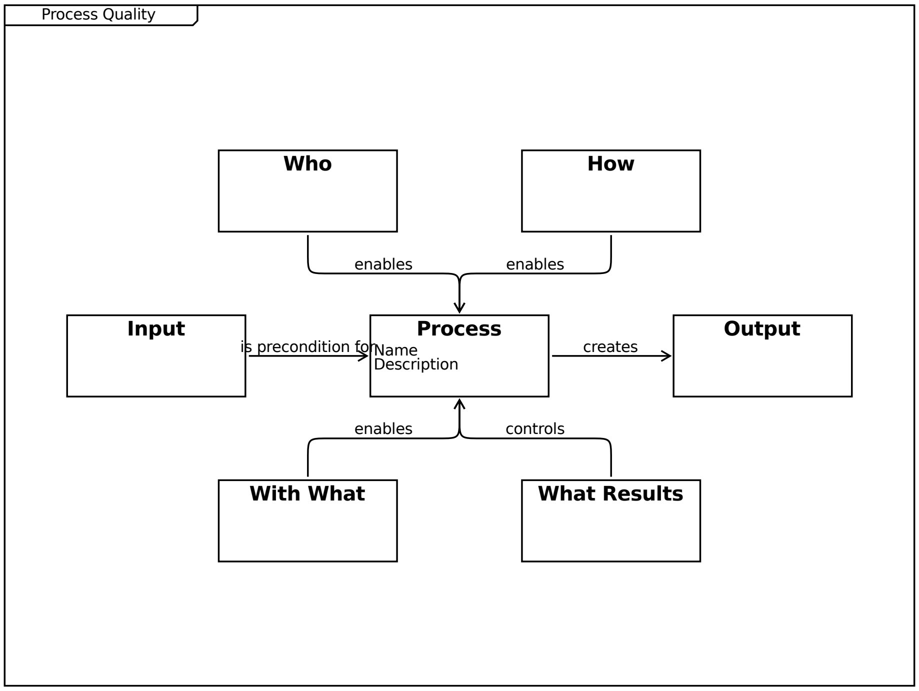
1 Quality Example



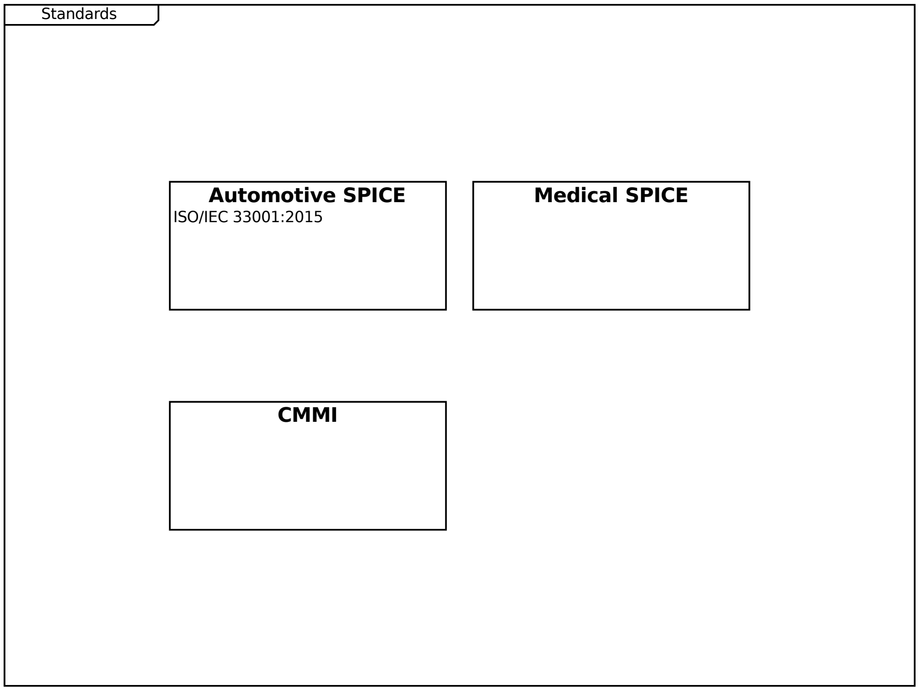
```
Quality


Product Quality
  ISO/IEC 25010:2011


Process Quality
```

2 Process Quality



```
Process Quality
| The turtle diagram shows the elements of a process.


Who
| Roles,
| Skills, Knowledge,
| Trainings
  enables --> Process


How
| Guidelines, Checklists,
| Templates
  enables --> Process


Input
  is precondition for --> Process
```

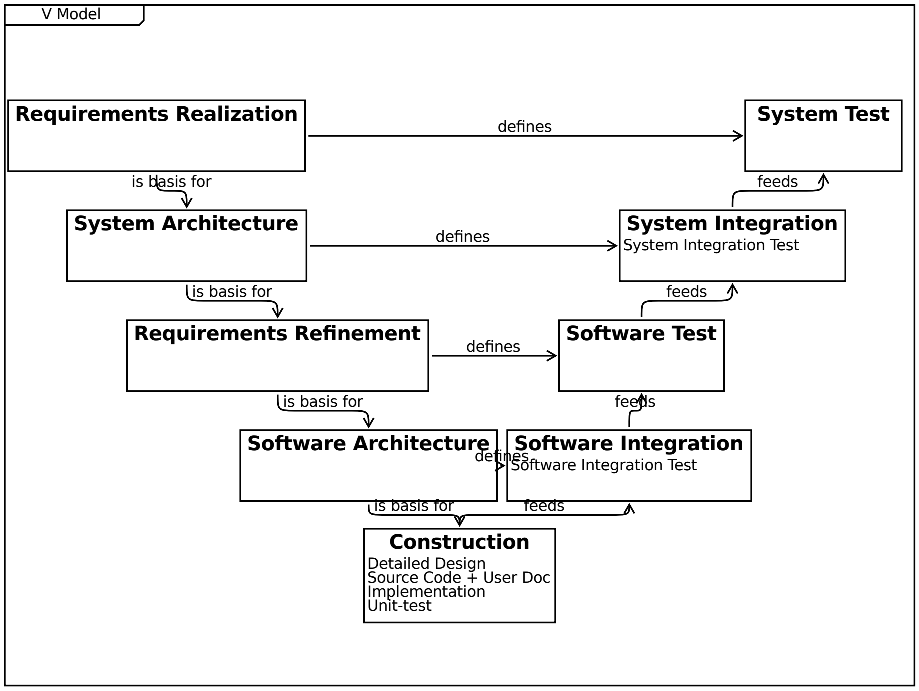
```
Process
  Name
  Description
  creates --> Output


Output
| Process output,
| Evidence on performed process


With What
  enables --> Process


What Results
  controls --> Process
```



```
Standards


Automotive SPICE
  ISO/IEC 33001:2015
```

Medical SPICE

CMMI



V Model

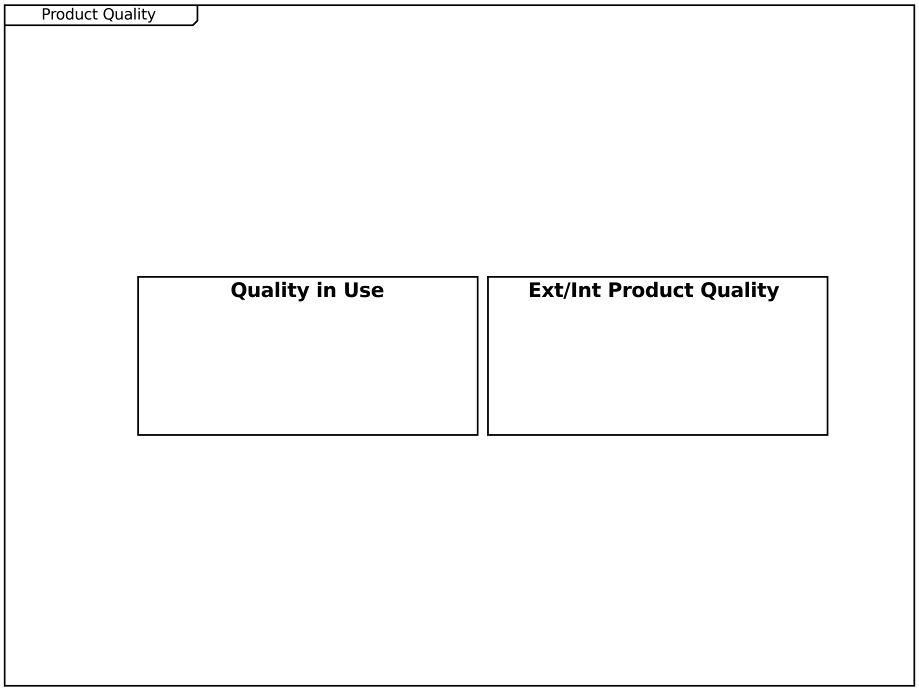
```
Requirements Realization
  is basis for --> System Architecture
  defines --> System Test
```

System Test

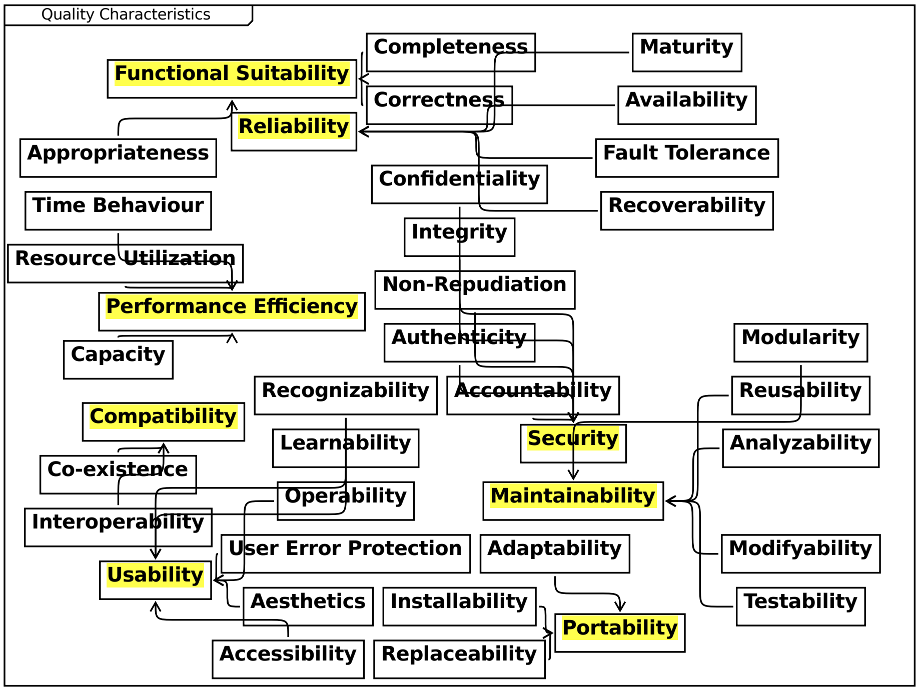
```
System Architecture
  is basis for --> Requirements Refinement
  defines --> System Integration
```

```
System Integration
  System Integration Test
  feeds --> System Test


Requirements Refinement
  is basis for --> Software Architecture
  defines --> Software Test


Software Test
  feeds --> System Integration


Software Architecture
  defines --> Software Integration
  is basis for --> Construction


Software Integration
  Software Integration Test
  feeds --> Software Test


Construction
  Detailed Design
  Source Code + User Doc
  Implementation
  Unit-test
  feeds --> Software Integration
```

3 Product Quality



```
Product Quality


Quality in Use
| Quality in use can be measured when the product is already in use,
| e.g. the percentage of satisfied customers can be determined.


Ext/Int Product Quality
| Product quality are internal and externally visible qualities,
| such as memory consumption or startup timings.
```



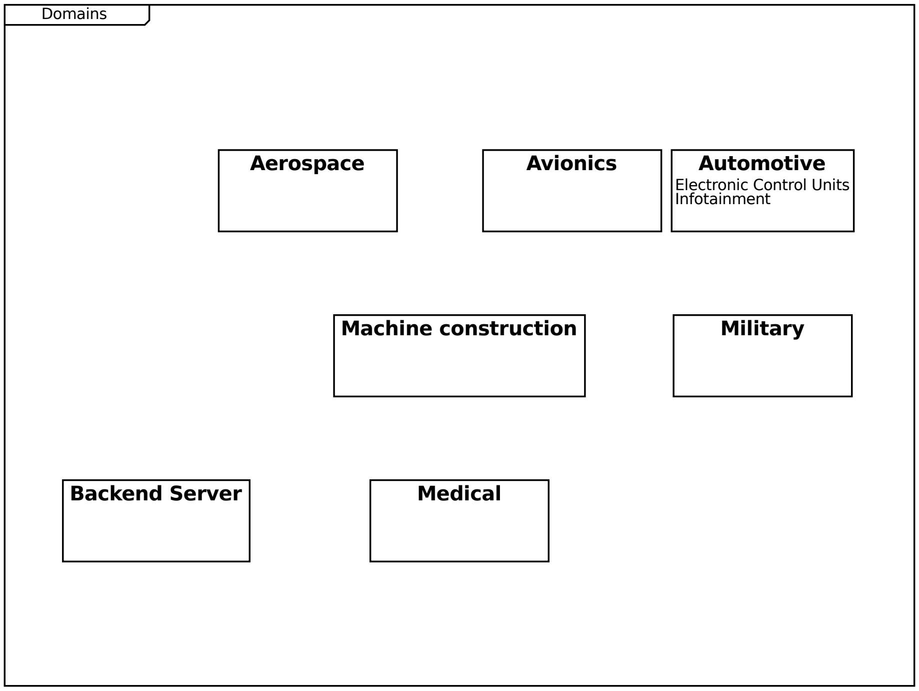
```
Quality Characteristics
| according to ISO 25010


Completeness
   --> Functional Suitability


Maturity
   --> Reliability


Functional Suitability


Correctness
   --> Functional Suitability


Availability
   --> Reliability
```

```
Reliability


Appropriateness
   --> Functional Suitability


Fault Tolerance
   --> Reliability


Confidentiality
   --> Security


Time Behaviour
   --> Performance Efficiency


Recoverability
   --> Reliability


Integrity
   --> Security


Resource Utilization
   --> Performance Efficiency


Performance Efficiency


Non-Repudiation
   --> Security


Capacity
   --> Performance Efficiency


Authenticity
   --> Security
```

```
Modularity
   --> Maintainability


Security


Recognizability
   --> Usability


Accountability
   --> Security


Reusability
   --> Maintainability


Compatibility


Learnability
   --> Usability


Analyzability
   --> Maintainability


Co-existence
   --> Compatibility


Operability
   --> Usability


Maintainability


Interoperability
   --> Compatibility


User Error Protection
```

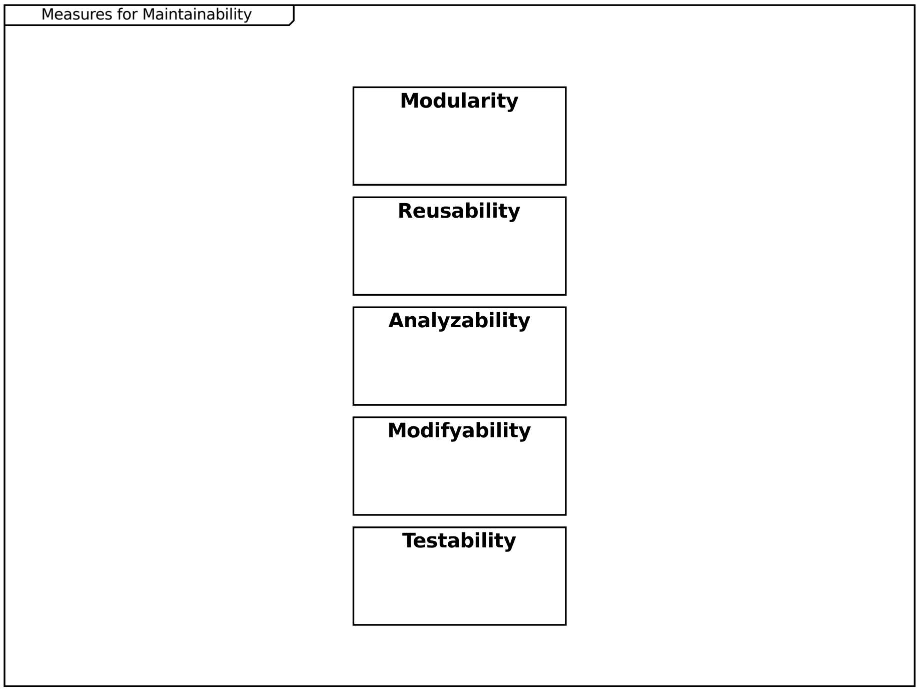
```
   --> Usability


Adaptability
   --> Portability


Modifyability
   --> Maintainability


Usability


Aesthetics
   --> Usability


Installability
   --> Portability


Testability
   --> Maintainability


Portability


Accessibility
   --> Usability


Replaceability
   --> Portability
```

3.1 Product Quality Measures



```
Domains


Aerospace


Avionics


Automotive
  Electronic Control Units
  Infotainment


Machine construction


Military


Backend Server
```

Medical



Measures for Maintainability

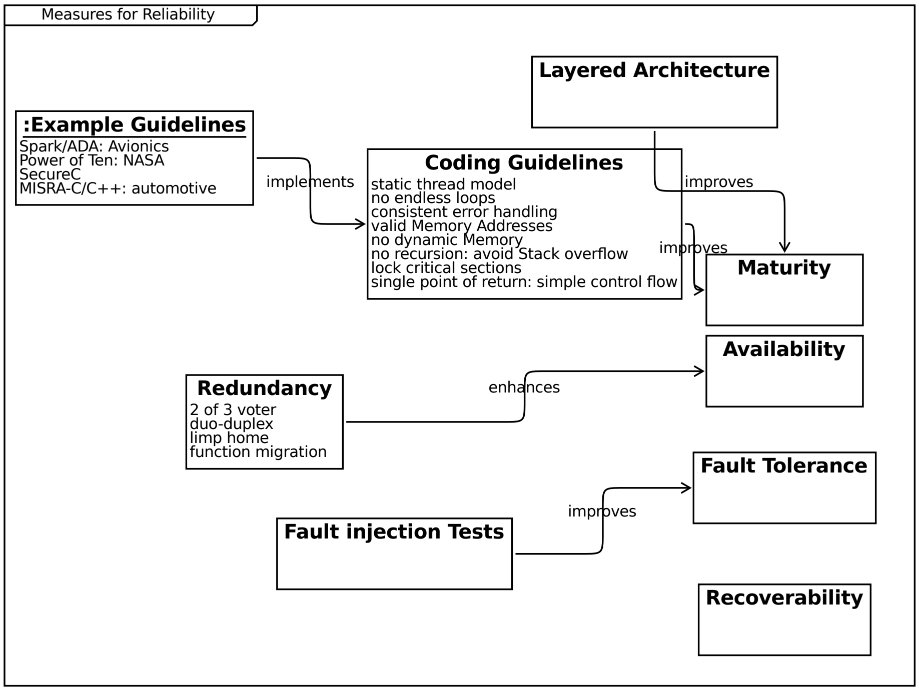
Modularity

Reusability

Analyzability

Modifyability

Testability



```
Measures for Reliability


Layered Architecture
  improves --> Maturity


Example Guidelines
  Spark/ADA: Avionics
  Power of Ten: NASA
  SecureC
  MISRA-C/C++: automotive
  implements --> Coding Guidelines


Coding Guidelines
  static thread model
  | Execution threads shall not be started/stopped dynamically
  no endless loops
  | Every loop shall have a counter to ensures that
  | after a predefined maximum value the loop is definitely quit
  consistent error handling
  | Inconsistencies in error handling make
  | bugs in error handling more likely
  valid Memory Addresses
  | Only valid memory addresses may be read/written.
  | E.g. Java solves this by prohibiting pointers,
  | In C/C++, check pointers and array indices before usage
  no dynamic Memory
  | When the program is running,
  | - it must not fail due to
  |   - memory fragmentation (virtual addresses/physical pages)
  |   - out of memory situations
  | - it shall have a defined timing (which new/malloc cannot provide)
  no recursion: avoid Stack overflow
  lock critical sections
  | Always lock critical sections.
  | Exceptions to locking are a nightmare.
  single point of return: simple control flow
  | Simple control flow is key to understandable code
  improves --> Maturity


Maturity


Availability


Redundancy
  2 of 3 voter
  duo-duplex
  limp home
  function migration
  enhances --> Availability


Fault Tolerance


Fault injection Tests
  improves --> Fault Tolerance


Recoverability
```