Public

A software modularity metric

Joost Zonneveld
Bram Schoenmakers

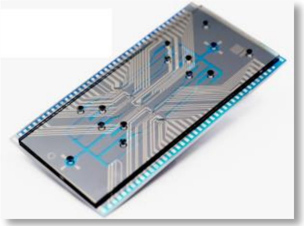
2016-02-02

ASML In 40 Seconds

New devices, new applications

Wearable sensors (Holst Centre)

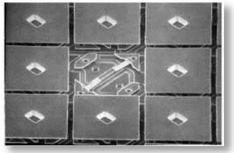
Imaging drone to monitor crop growth and yield (imec)



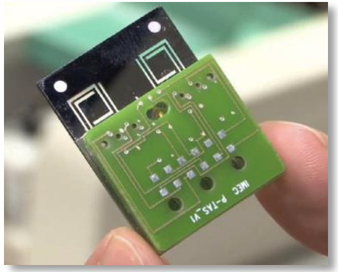
Cell sorter to detect metastization (imec)

Textile integrated health patch

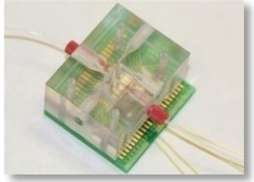
Simband with health monitoring (Samsung)



Micro mirrors for beamers (TI)

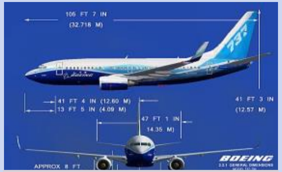


On-Chip DNA amplification and detection (imec/Panasonic)



Lab on a Chip (LOC) for counting red blood cells

Characteristics driving Twinscan SW architecture



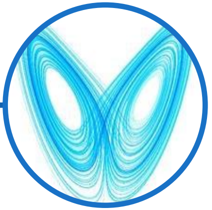
NXT weight ~ Boeing 737

NXE weight ~ Dreamliner

Product Family

Continuous enhancement after release

Concurrent development by +1000 SW engineers



Increasing coupling at nm level

ASML Twinscan software facts and figures

Architecture

- Specified (not derived) using ASML Architecture Description Language
  - Different perspectives (software layers, litho functions, product variants)
  - Build time enforced: not according to ADL → can not be built
- Explicit interfaces, specified with ASML IDL
- Focus on macro modularity and micro modularity

Patterns and tools for
Data, Control and Algorithms

Implementation

- 50 MLoc, mostly C, C++ ↑ , Python ↑ and Matlab↑
- 2200 components, 11000 interfaces
- About 8 DSLs with code generation.

Objectives for modular software

What

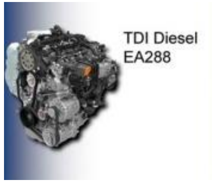
- Scalable software, support growing product and growing company
- Reuse functionality across releases
- Support outsourcing/OEM development

How: System of Systems approach:

- Develop modules like developing a system.
  - Maximum ownership / empowerment
  - Local optimization possibilities (process, tools, branching)
  - Focused on module’s core business
  - Local technology phase in/out

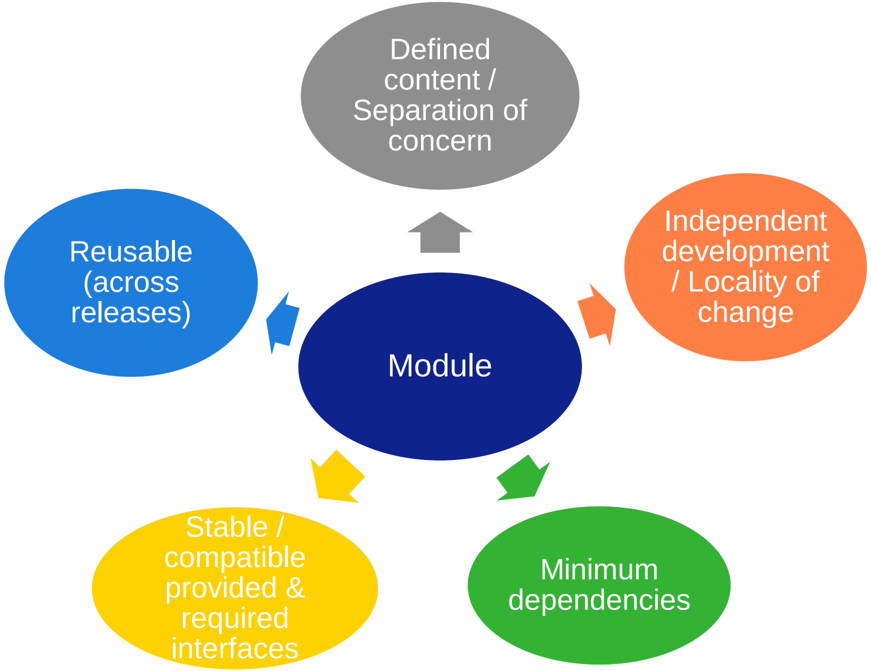
- Develop Twinscan by integrating / reusing modules
  - Decentral what can be, central what must be (efficiency, consistency)

System of Systems
in automotive domain

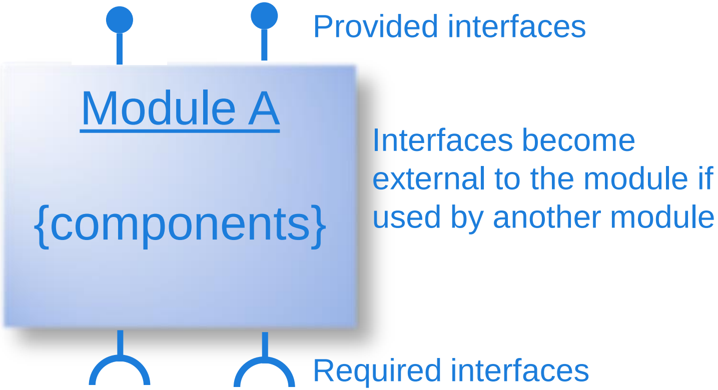


Intro: Module

Module properties



Module definition



A module is a (virtual) collection of ASML SW components.
A module is considered a black box.

Target: 25 - 50 macro modules

Intro: Interfaces

Interface is a contract between two or more modules
The contract is stored in one of the modules

Interface

Syntax

Semantics

Files

Functions

Structs

Enums

#defines

2500 inter module interfaces
(of 10000 total)

Protocol State Machines (POC)

#changes
can be counted

#symbols
can be counted

An interface change is backward compatible if
client sources/binaries need no adaption

Depending on source integration or
binary integration

Intro: Reuse module across releases

Since 2015

6.2

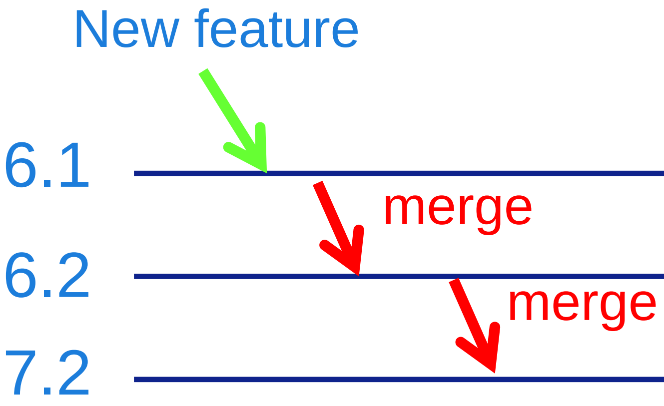
7.2

6.1

Now: reuse sources, build the whole system
Plan: reuse binaries

Before 2015

Share functionality by merging between monolithic system archives



Measure modularity

A modularity metric was developed to estimate modularity.

Suitable external metric was searched but not found

- **Steer** towards:

Reuse of modules across releases

Independent module evolution

Comprehensibility, minimize complexity

- **Assess** whether a prospect module is ready for an independent archive

! metric is not a goal, but a means to show modularity improvement.

Ref: You Are What You Measure (Hauser, Katz)

Modularity metric design guidelines

- Discourage small modules
  (lesson learned from industry partner)
- Prevent modules to become smaller and smaller
  (lesson learned from interface metric)
- Applicable for multiple abstraction levels
- Minimize biases that cause wrong conclusions / allow gaming
- Insensitive to relative position of module in hierarchy
- Measurable with reasonable cost/overhead
- Prefer snapshot measurements over measurement over time
- Prepared for binary integration (availability of source files not required)

Calculation of the metric: weighted sum of 7 submetrics

Value of submetric *m*, range: [0,1]

Modularity metric of module *A*

Weight of metric *m*

$$\text{Modularity metric}(A) = \left(\sum_{m \in Metrics} m(A) * w(m) \right) * C(A_{size})$$

Correction factor based on size of module *A*.
Lower for smaller modules

Mix of submetrics reduces vulnerability for gaming

Submetric 1: change frequency provided interfaces

Property	Metric	Weight
Interface stability	**Change frequency provided interfaces**	**15%**
	Change frequency required interfaces	15%
Coupling	# provided + required symbols	15%
	# direct cyclic dependencies	15%
Testability	Configuration space (prov. + req. VPs)	10%
Shareability	# missing symbols in other releases	10%
Locality of Change	% single module streams	20%

Measures: sum of number of changes to provided interfaces in the past year.

Ref: Open Closed Principle (OCP)

Rationale: minimize client impact when upgrading.

Range: 0 – 10 changes per year

Lower is better.

Biases: favors large interfaces; favors overly abstract interfaces; discourages interface refactoring. Does not cover semantics. Compatible and incompatible changes are treated equally.

Submetric 2: change frequency required interfaces

Property	Metric	Weight
Interface stability	Change frequency provided interfaces	15%
	Change frequency required interfaces	**15%**
Coupling	# provided + required symbols	15%
	# direct cyclic dependencies	15%
Testability	Configuration space (prov. + req. VPs)	10%
Shareability	# missing symbols in other releases	10%
Locality of Change	% single module streams	20%

Measures: sum of number of changes to required interfaces in the past year.

Ref: Stable Dependencies Principle (SDP)

Rationale: stability contributes to binary integration. Both sides are participants in the interface contract.

Range: 0 – 20 changes per year

Lower is better.

Biases: See previous slide + Could lead to duplicate functionality (reducing coupling)

Submetric 3: provided + required interface symbols

Property	Metric	Weight
Interface stability	Change frequency provided interfaces	15%
	Change frequency required interfaces	15%
Coupling	**# provided + required symbols**	**15%**
	# direct cyclic dependencies	15%
Testability	Configuration space (prov. + req. VPs)	10%
Shareability	# missing symbols in other releases	10%
Locality of Change	% single module streams	20%

Measures: number of provided + required symbols

Rationale: minimize coupling with other modules

Range: 0 – 10000 symbols

Lower is better

Biases: Could lead to duplicate functionality (reducing coupling); No distinction essential/accidental dependencies; Hidden dependencies not counted.

Submetric 4: cyclic dependencies

Property	Metric	Weight
Interface stability	Change frequency provided interfaces	15%
	Change frequency required interfaces	15%
Coupling	# provided + required symbols	15%
	# direct cyclic dependencies	**15%**
Testability	Configuration space (prov. + req. VPs)	10%
Shareability	# missing symbols in other releases	10%
Locality of Change	% single module streams	20%

Measures: number of interfaces causing a cycle between two modules.

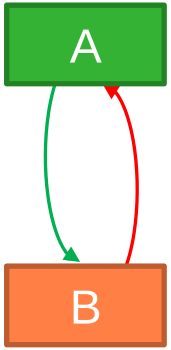
Ref: Acyclic Dependencies Principle (ADP)

Rationale: Minimize coupling, prevent upgrade dependencies and contributes to binary integration.

Range: 0-100 interfaces

Lower is better.

Bias: Accountability issue (account to A or B?)
Could lead to duplicate functionality to reduce coupling; Direct cycles only;
Could lead to smaller modules.



“Undesired” direction can be configured.

Submetric 5: configuration space

Property	Metric	Weight
Interface stability	Change frequency provided interfaces	15%
	Change frequency required interfaces	15%
Coupling	# provided + required symbols	15%
	# direct cyclic dependencies	15%
Testability	**Configuration space (prov. + req. VPs)**	**10%**
Shareability	# missing symbols in other releases	10%
Locality of Change	% single module streams	20%

Measures: multiplication of number of values of module's variation points.

Ref: Open Closed Principle (OCP)

Rationale: lower scores indicates that it's easier to test all possible configurations of a module.

Range: 0 - 110

Lower is better, this can be a *huge* number, therefore its 10-base log is used as metric.

Biases: Assumes all variants are orthogonal. Discourages adding more (configurable) functionality.

Submetric 6: missing symbols for shareability

Property	Metric	Weight
Interface stability	Change frequency provided interfaces	15%
	Change frequency required interfaces	15%
Coupling	# provided + required symbols	15%
	# direct cyclic dependencies	15%
Testability	Configuration space (prov. + req. VPs)	10%
Shareability	**# missing symbols in other releases**	**10%**
Locality of Change	% single module streams	20%

Measures: average number of symbols required by the mainline version of the module, but missing in selected releases

Ref: Release Reuse Equivalence Principle (REP)

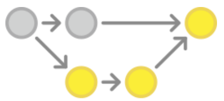
Rationale: module can more easily be “plugged” into other releases.

Range: 0 – 2000 symbols

Lower is better.

Biases: Semantics not covered.

Submetric 7: % single module streams



Property	Metric	Weight
Interface stability	Change frequency provided interfaces	15%
	Change frequency required interfaces	15%
Coupling	# provided + required symbols	15%
	# direct cyclic dependencies	15%
Testability	Configuration space (prov. + req. VPs)	10%
Shareability	# missing symbols in other releases	10%
Locality of Change	**% single module streams**	**20%**

Measures: Locality of Change for module *A*:

$$Index(A) = \frac{\text{streams } \mathit{only} \text{ affecting } A}{\text{all streams affecting } A}$$

Ref: Common Closure Principle

Rationale: a high score indicates that the module can evolve independently.

Range: 0 – 100%

Higher is better

Bias: captures process-oriented aspects, does not cover multiple single-module streams for the same function.

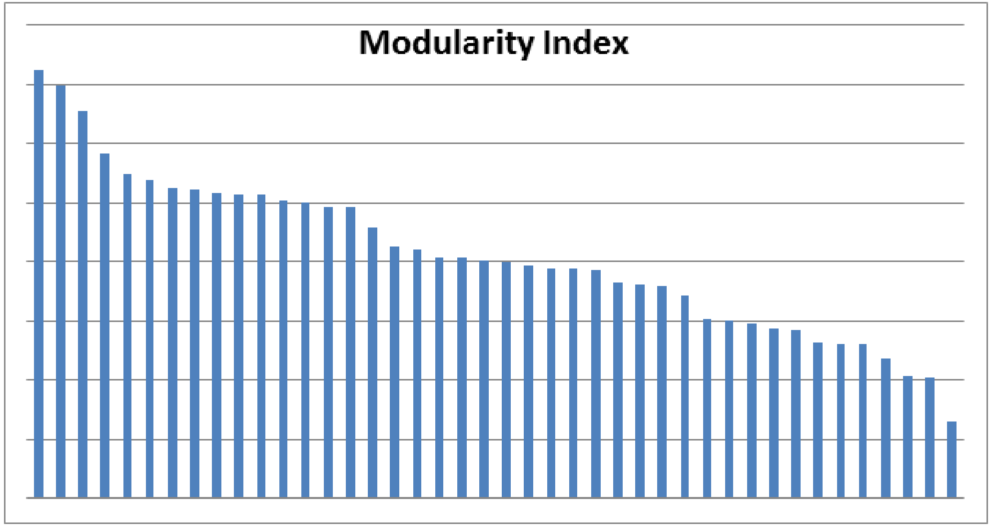
Deployment of modularity improvement

ASML has a roadmap to transform the monolithic archive in modular software

Now 6 independent macro modules, covering ~25% of the software.

Modularity metric used to steer the remaining 75% to be come sufficient modular.

The owners of candidate modules define their target for modularization



ASML