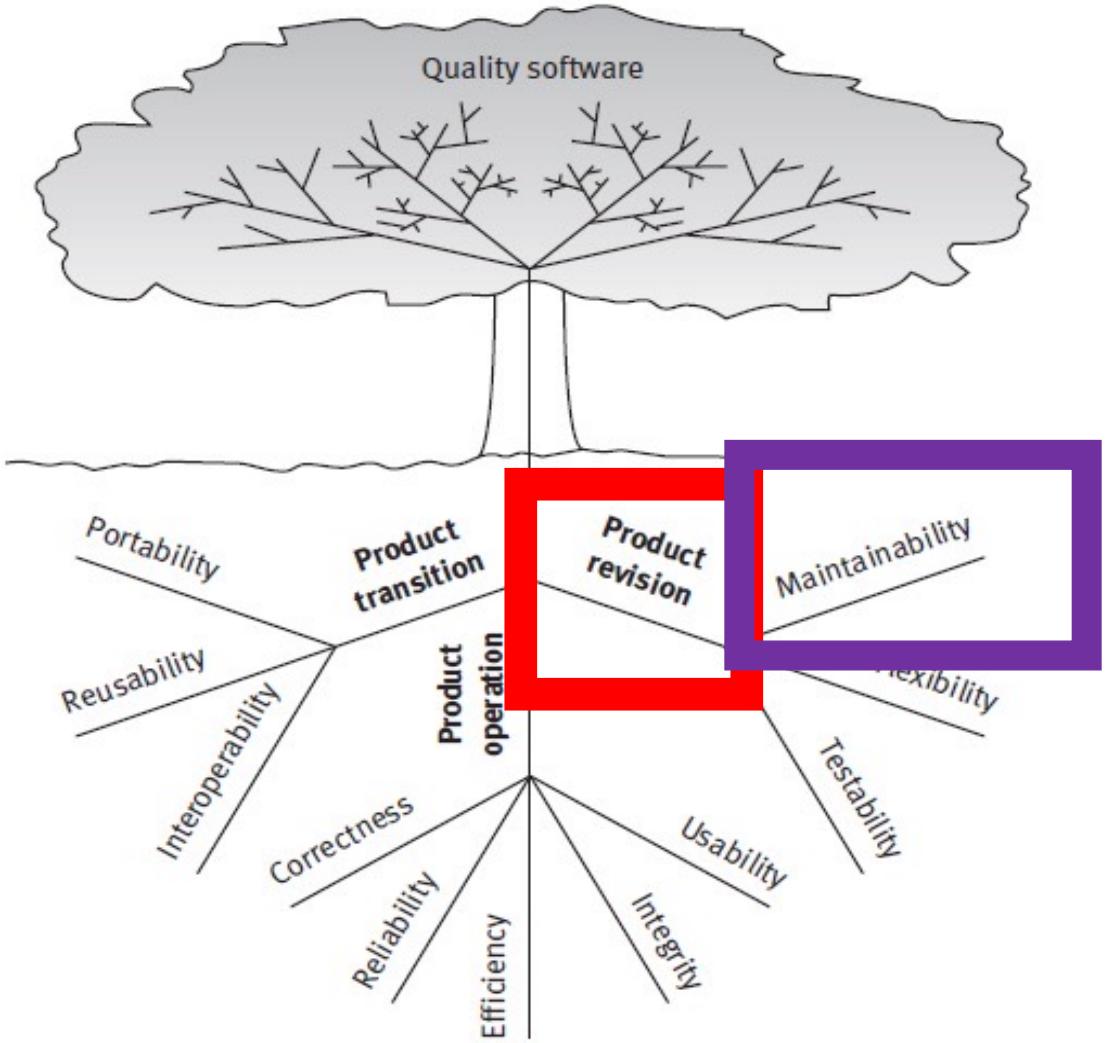
Course 6

Maintainability

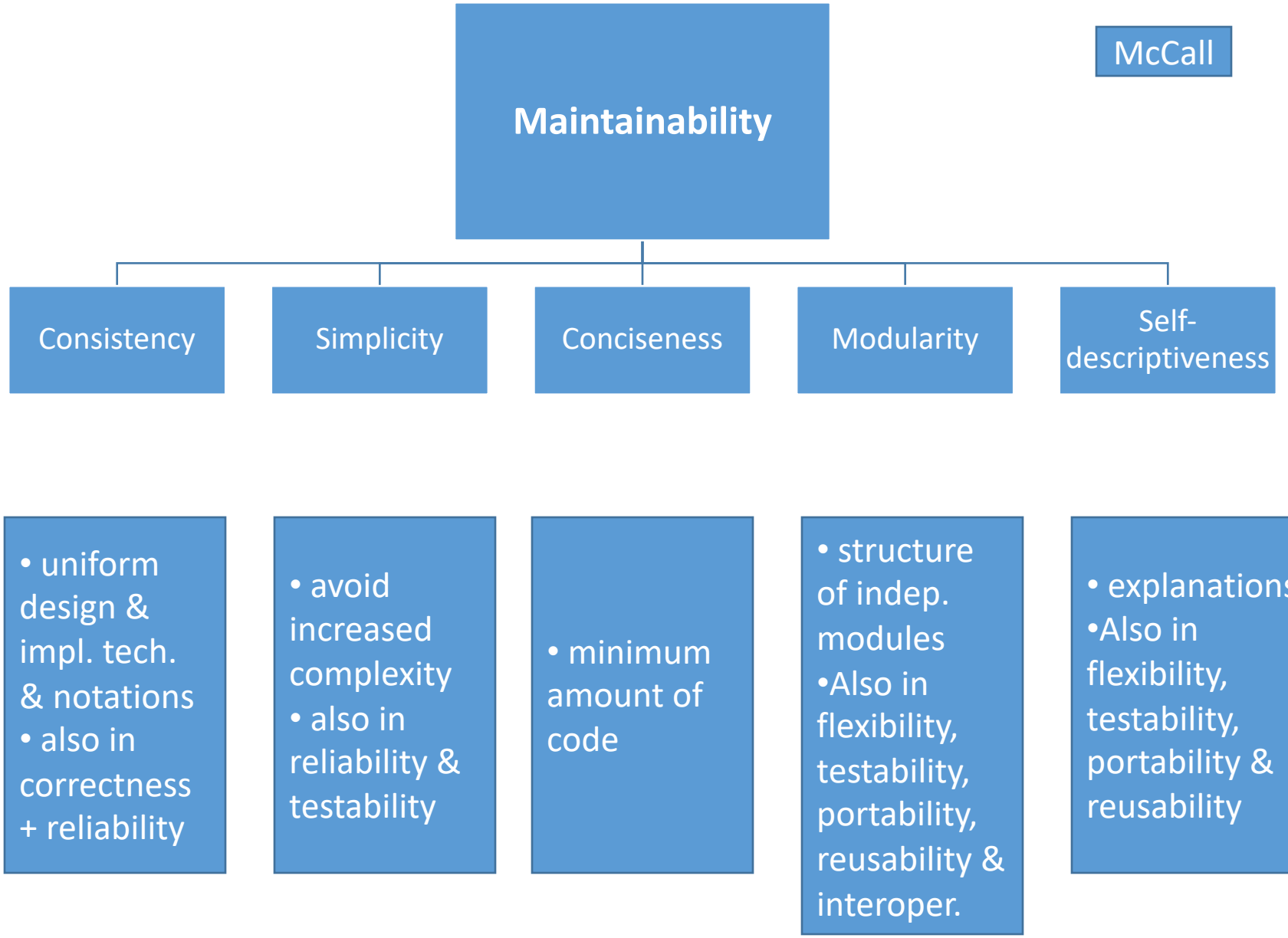

Quality software
Portability
Product transition
Product revision
Maintainability
Reusability
Interoperability
Product operation
Correctness
Reliability
Efficiency
Integrity
Usability
Testability

Maintainability [McCall]

Definition: *Effort required to locate and fix an error in an operational program*

Impact:

Measured: design + implementation	Realized in: maintenance + transition


Maintainability
McCall
Consistency
Simplicity
Conciseness
Modularity
Self-descriptiveness
• uniform design & impl. tech. & notations
• also in correctness + reliability
• avoid increased complexity
• also in reliability & testability
• minimum amount of code
• structure of indep. modules
•Also in flexibility, testability, portability, reusability & interoper.
• explanations
•Also in flexibility, testability, portability & reusability

Maintainability – ISO9126

- Analyzability
- Changeability
- Stability – encapsulation and data hiding
- Testing

Maintainability ISO 25010

- Modularity
- Reusability
- Analysability
- Modifiability
- Testability

Maintainability influenced by **Analyzability**

- Locate causes of failure - correct
- Locate parts for modification – extend

- Readability
- Comprehensibility
- Traceability

Maintainability influenced by…**Modifiability**

- Easily identify elements to change
- Changes compared to the specification
- Changes affect rest of the system

Maintainability influenced by…**Testability**

- Unit testing
- Integration testing
- System testing
- Test coverage

Maintainability influenced by…

- style: name convention, indent
- simple !!!
- comments

Maintainability measured …

- 3 purposes:
  1. System maintainability over time – disintegrate as it evolves
  2. Compare different systems performing the same task
  3. Evaluate parts – less maintainable – target for refactoring

Maintenance - classification

Corrective Maintenance — Correct any errors

Adaptive Maintenance — Adapt to change

Preventive Maintenance — Prevent the system to be obsolete (updates)

Perfective Maintenance — Improve (performance, efficiency)

How to measure maintainability?

Maintainability measured …

- Maintainability index (different formula)

MI = 171-5.2 x ln(aveV)
-0.23 x aveV(g')
-16.2 x ln(aveLOC)
+ 50 x sin sqrt(2.4 PerCM)
http://www.virtualmachinery.com/sidebar4.htm
[-100, 200]

MI = MAX(0,(171 - 5.2 * ln(Halstead Volume) –
0.23 * (Cyclomatic Complexity) –
16.2 * ln(Lines of Code))*100 / 171)
[0,100]
0-9 =bad; 10-19= satisfactory; 20-100 = acceptable

Parameter	Name	Measures
aveV	*Average Halstead complexity*	*Computational density*
aveV (g')	*Average extended cyclomatic complexity*	*Logical complexity*
aveLOC	*Average count of lines of code*	*Code size*
PerCM	*Average percent of lines of comments*	*Human insight*

Halstead complexity:

$$V = N \times \log_2(n)$$

N = no of operators

n = no of distinct operators

Cyclomatic complexity:

= number of linearly independent paths through a program

Maintainability Index measured …

- Tools:
  - JHawk
  - Metrics – .NET framework
  - Radon - Python

Critics to MI

- Average values are used in the computation, ignoring the real distribution of values
- defined threshold values - not accurate
- defined for modular and procedural programming languages, not OOP – ignoring inheritance, coupling and cohesion
- van Deursen, A.: Think Twice before Using the Maintainability Index (2014)

Maintainability for OOP: metrics high influence

- WMC – Weighted Methods per Class
- DIT – Depth of Inheritance Tree
- NOC – Number of Children
- CBO – coupling between objects (method call, field access, inheritance, exceptions, …)
- …

Technical Debt

- Computed instead of maintainability
- Introduces by Cunningham (1992): metaphor from the financial sector
- = the debt that is accumulated when development of new features is prioritized over fixing known issues (principal) + interest (not fixing at time)
- See https://martinfowler.com/bliki/TechnicalDebt.html

Martin Fowler (OOAD, patterns, UML, agile, refactoring)

https://www.stepsize.com/blog/technical-debt-horror-stories

Types of TD [Fowler quadrant - https://martinfowler.com/bliki/TechnicalDebtQuadrant.html]

	Reckless	Prudent
Deliberate	"We don't have time for design"	"We must ship now and deal with consequences (later)"
Inadvertent	"What's Layering?"	"Now we know how we should have done it"

TD classification

- Code debt (technical debt)
- Architectural TD
- Documentation debt
- Test debt
- Versioning debt
- Requirement debt …

Computing TD – case study SonarQube

Debt = duplication + violations + comments + coverage + complexity + design

Where:

duplication = cost_to_fix_one_block * duplicated_blocks

violations = cost_to_fix_one_violation * mandatory_violations

comments = cost_to_comment_one_API * public_undocumented_API

coverage = cost_to_cover_one_of_complexity *
uncovered_complexity_by_tests

design = cost_to_cut_an_edge_between_two_files * package_edges_weigh

complexity = cost_to_split_a_method
* (function_complexity_distribution ≥ 8)
+ cost_to_split_a_class
* (class_complexity_distribution ≥ 60)

[O. Gaudin, "Evaluate your technical debt with Sonar," Sonar, Jun, 2009]

Cost (in man-hours)	Default value (man hour)
cost_to_fix_one_block	2
cost_to_fix_one_violation	0.1
cost_to_comment_one_API	0.2
cost_to_cover_one_of_complexity	0.2
cost_to_split_a_method	0.5
cost_to_split_a_class	8
cost_to_cut_an_edge_between_two_files	4

TD in SonarQube

Code smells

Bugs

Vulnerabilities

Technical debt = effort to fix code smells

Maintainability Rating (sqale_rating)
Default Maintainability Rating grid is:
A=0-0.05, B=0.06-0.1, C=0.11-0.20, D=0.21-0.5, E=0.51-1

A = remediation cost is <=5% of the time that has already gone into the application

New 2024: Impact on maintainability

Several models for TD [S.Motogna – Today Software Magazine series]

Model	Year	Reference	Tools
SQALE	2010	Letouzey	SonarQube, Square, Ndepend
CQM	2012	CQM	Kiuwan
CAST	2012	Curtis	CAST Software
Design Flaw	2012	Marinescu	inFusion (not available)
SIG	2011	SIG/TUViT	SIG method
CISQ	2018	CISQ, Curtis	standard

Portability

Portability

- Definition: *Effort required to transfer a program from one hardware configuration and/or software system environment to another.*
- Impact:
  - Measured:
    - Design
    - Code
  - Realized: transition

McCall

Portability

- Modularity
  - struct. of indep. modules
  - in other 6 factors
- Self-descriptiveness
  - explanations
  - Also in maintainability, testability, portability & reusability
- Machine Independence
  - hardware indep.
  - also in reusability
- Software System Independence
  - indep. of OS, utilities, I/O
  - also in reusability

ISO25010

Portability

- **Adaptability**: Degree to which a product or system can effectively and efficiently be adapted for different or evolving hardware, software or other operational or usage environments
- **Instalability**: Degree of effectiveness and efficiency with which a product or system can be successfully installed and/or uninstalled in a specified environment
- **Replaceability**: Degree to which a product can replace another specified software product for the same purpose in the same environment.

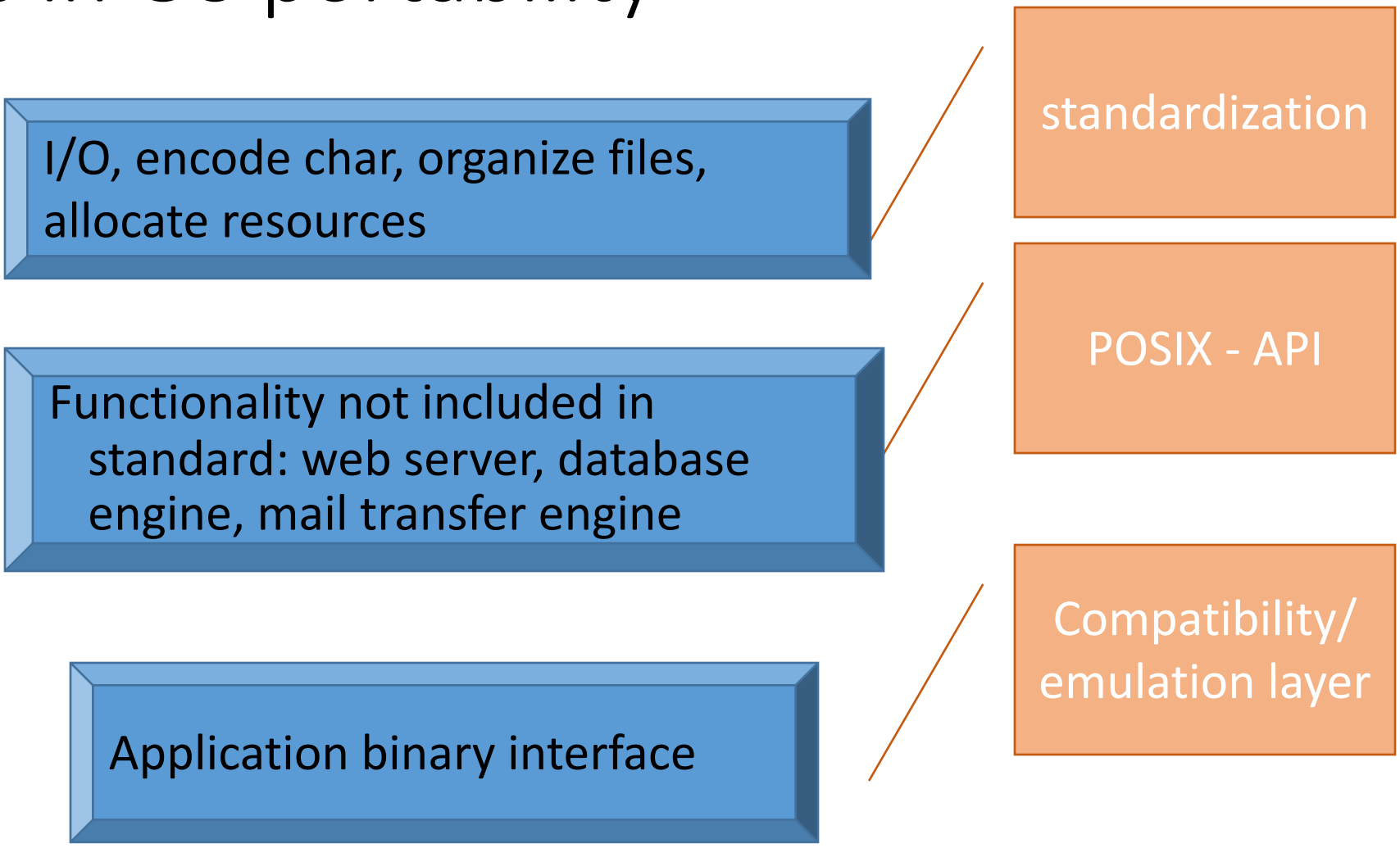
Why?

- Software life ≈15 years vs. hardware life ≈ 4-5 years
- Software implemented ≈ 3 hardware config.
- Porting less expensive than implementing
- Greater market

The 7 dimensions of portability

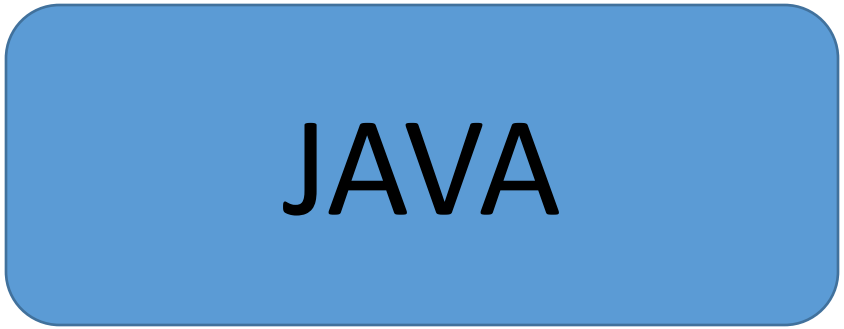
1. Operating systems
2. Processor architecture
3. Compiler & language features
4. GUI environment
5. Regions
6. Hardware devices
7. …

Issues in OS portability



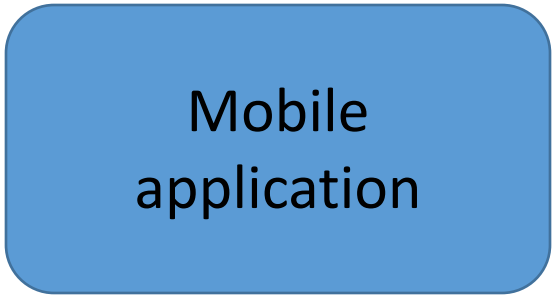
Issues in Processor Architecture portability

- Data type properties – size + operations
- Data representation – alignment
- Machine-specific code



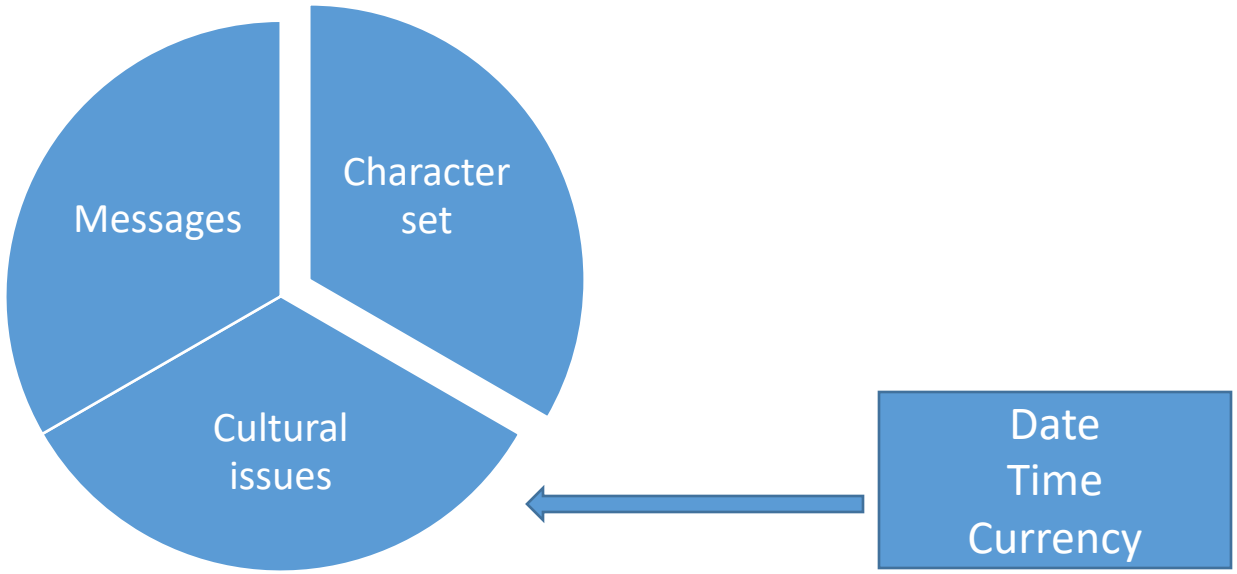
Approaches in GUI portability

- Ignore: platform dependent app.
- Emulation layer: use libraries for transformation
- Portability layer: isolate GUI elem.
- Portable platform: ex. Java – own API for GUI
- HTML or AJAX-based layer



Issues in region portability

- **Internationalization**: *generalization process of creating programs that can be easily ported across different regions*
- **Localization:** *particularization effort required to port a program to a given region*



Portability today

SOA

- Less technology dependent
- Internal platform independence through SOA

Open Source

- Less coding, more composition
- Portability of parts

Tinderbox
http://www-archive.mozilla.org/projects/tinderbox/

Portability metrics