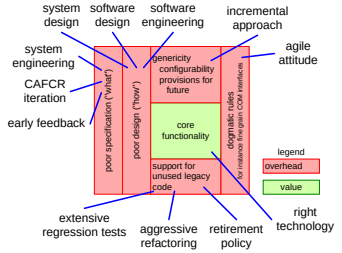
Exploration of the bloating of software

by *Gerrit Muller* University of South-Eastern Norway-NISE
e-mail: gaudisite@gmail.com
www.gaudisite.nl

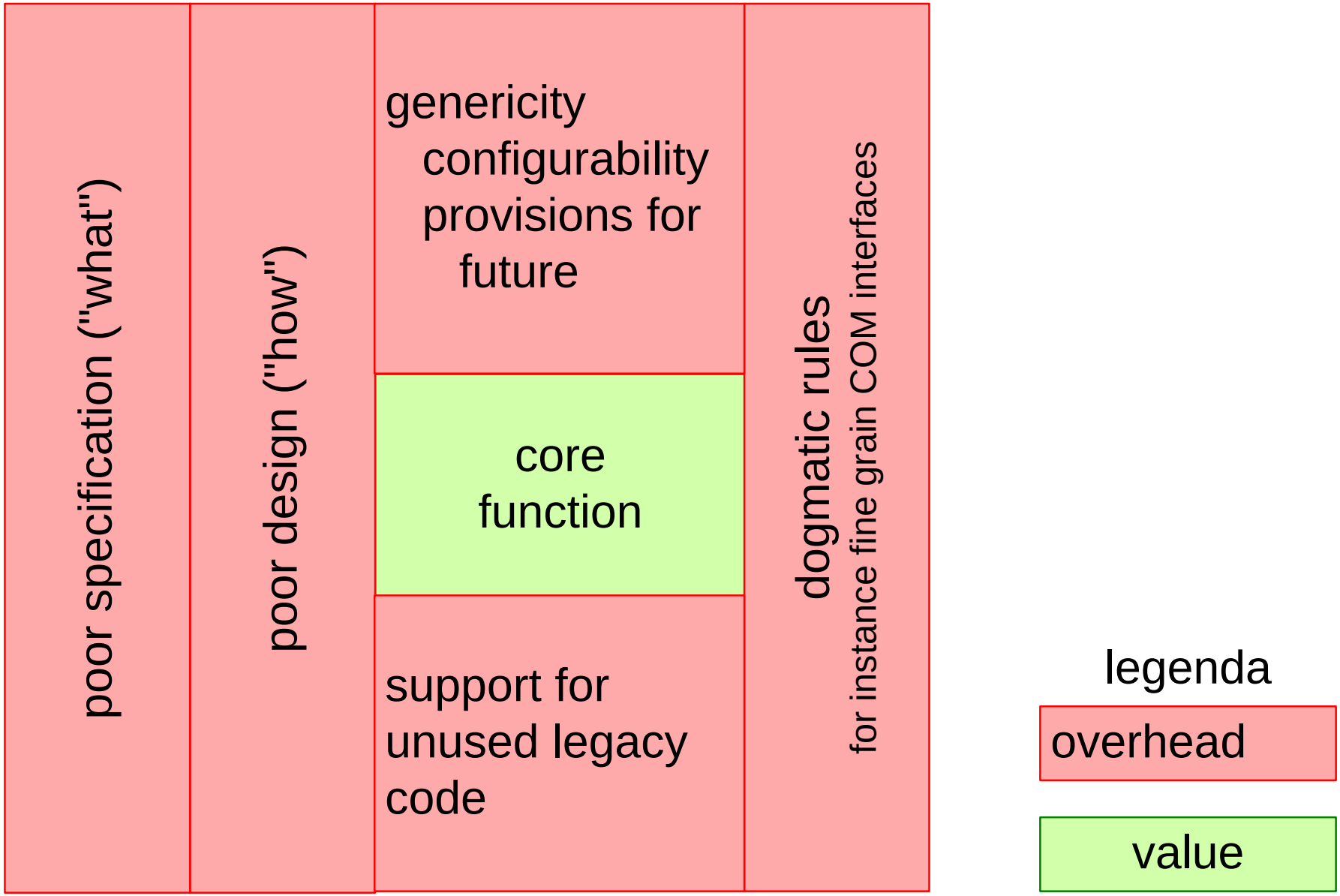
Abstract

Present-day products contain one order of magnitude more software code than is actually needed. The causes of this bloating are explored. If we are able to reduce the bloating significantly, then the product creation process is simplified tremendously. Potential handles to attack the bloating are discussed.

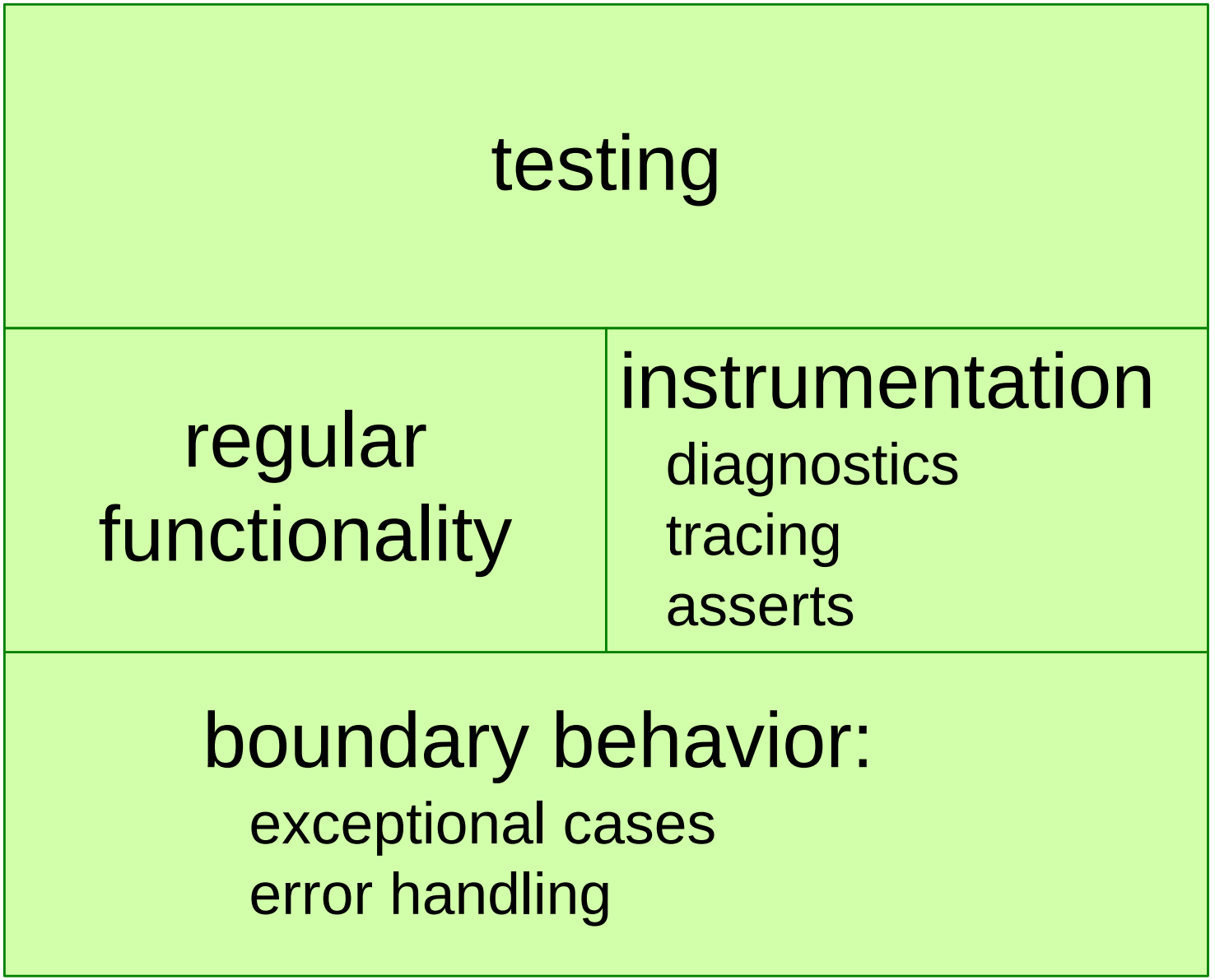


August 21, 2020
status: finished
version: 1.2

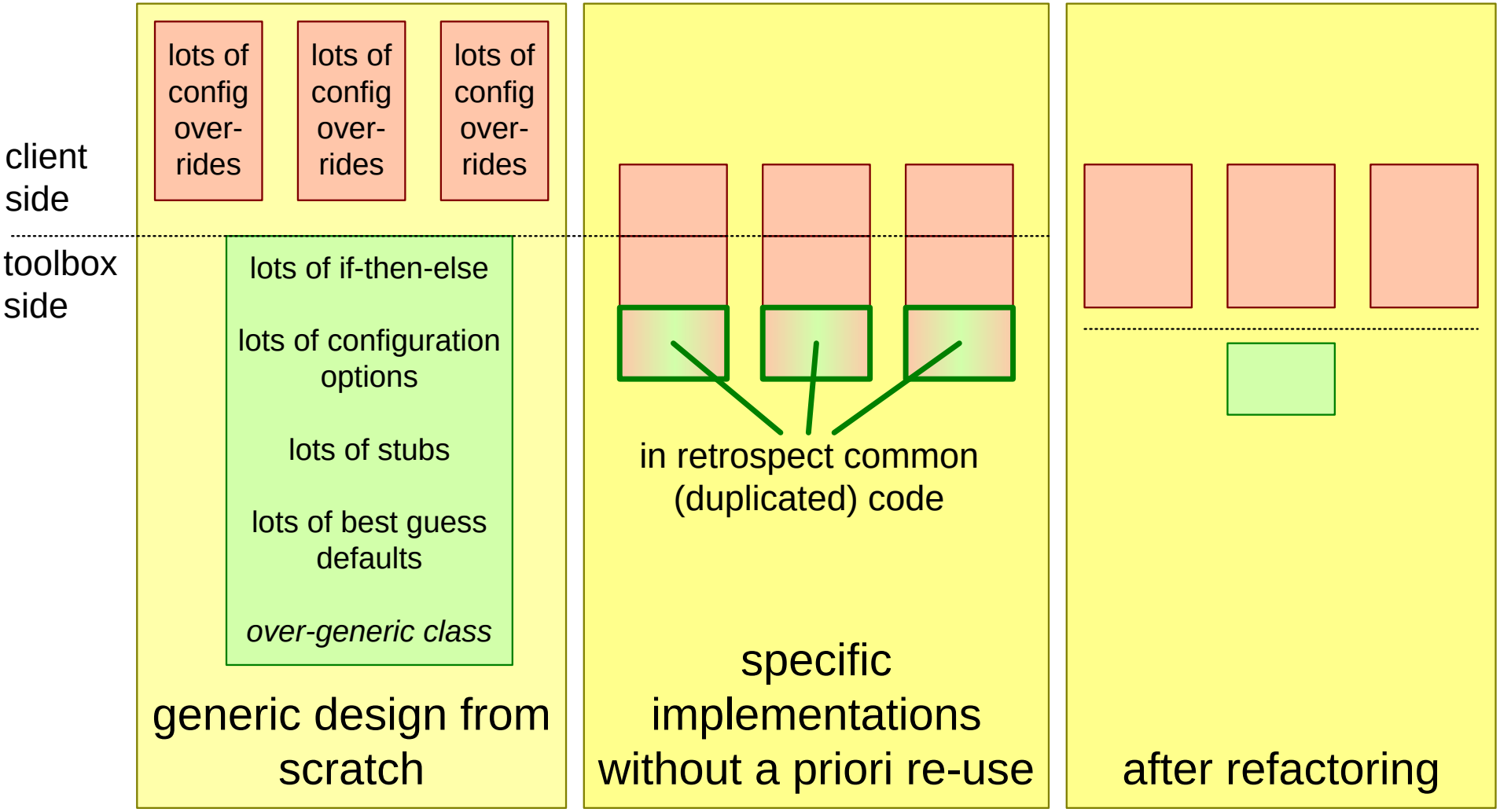
Exploring bloating: main causes



Necessary functionality ≫ the intended regular function

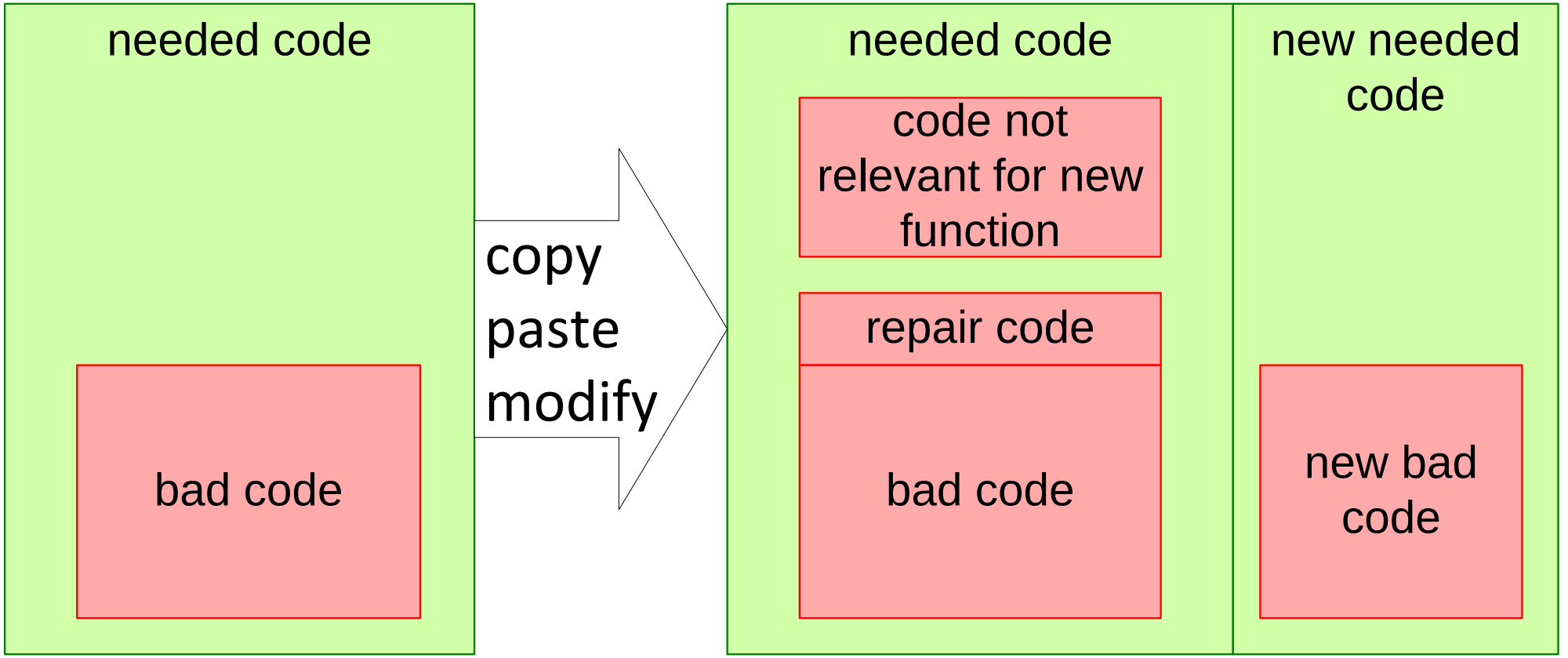


The danger of being generic: bloating

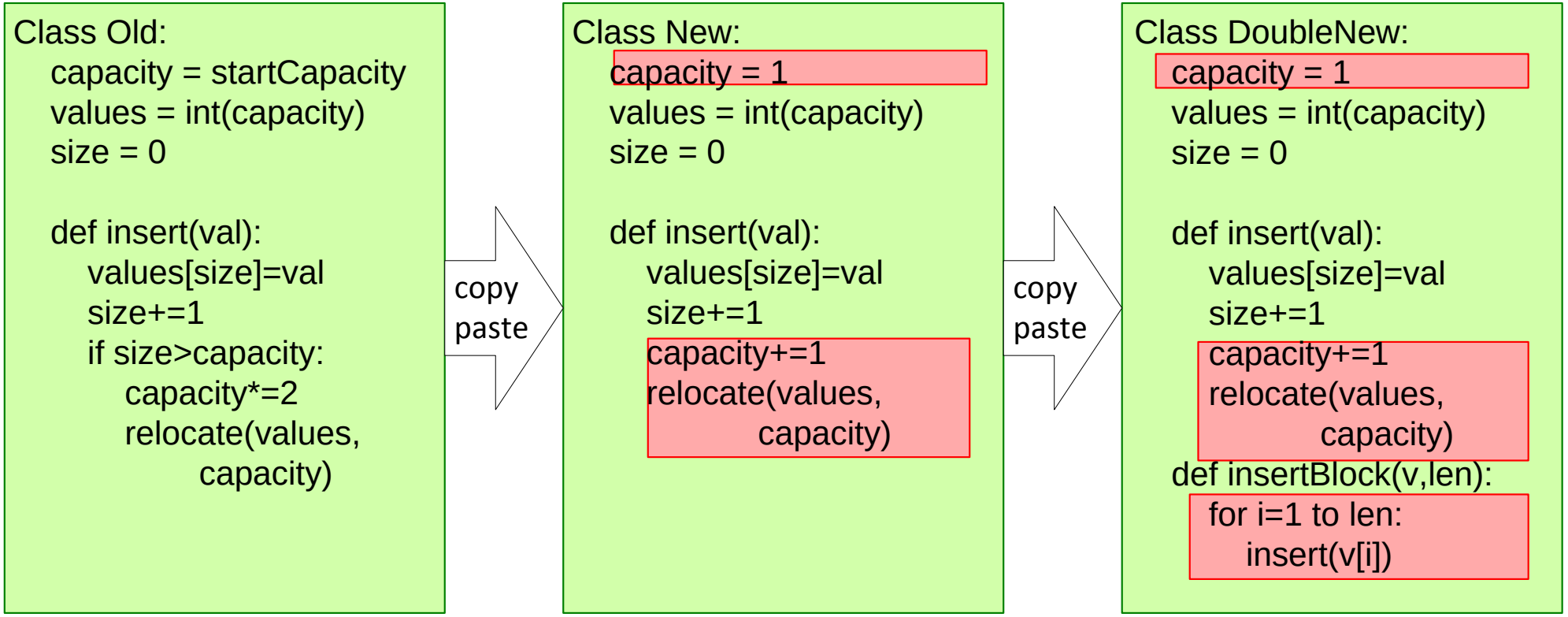


"Real-life" example: redesigned *Tool* super-class and descendants, ca 1994

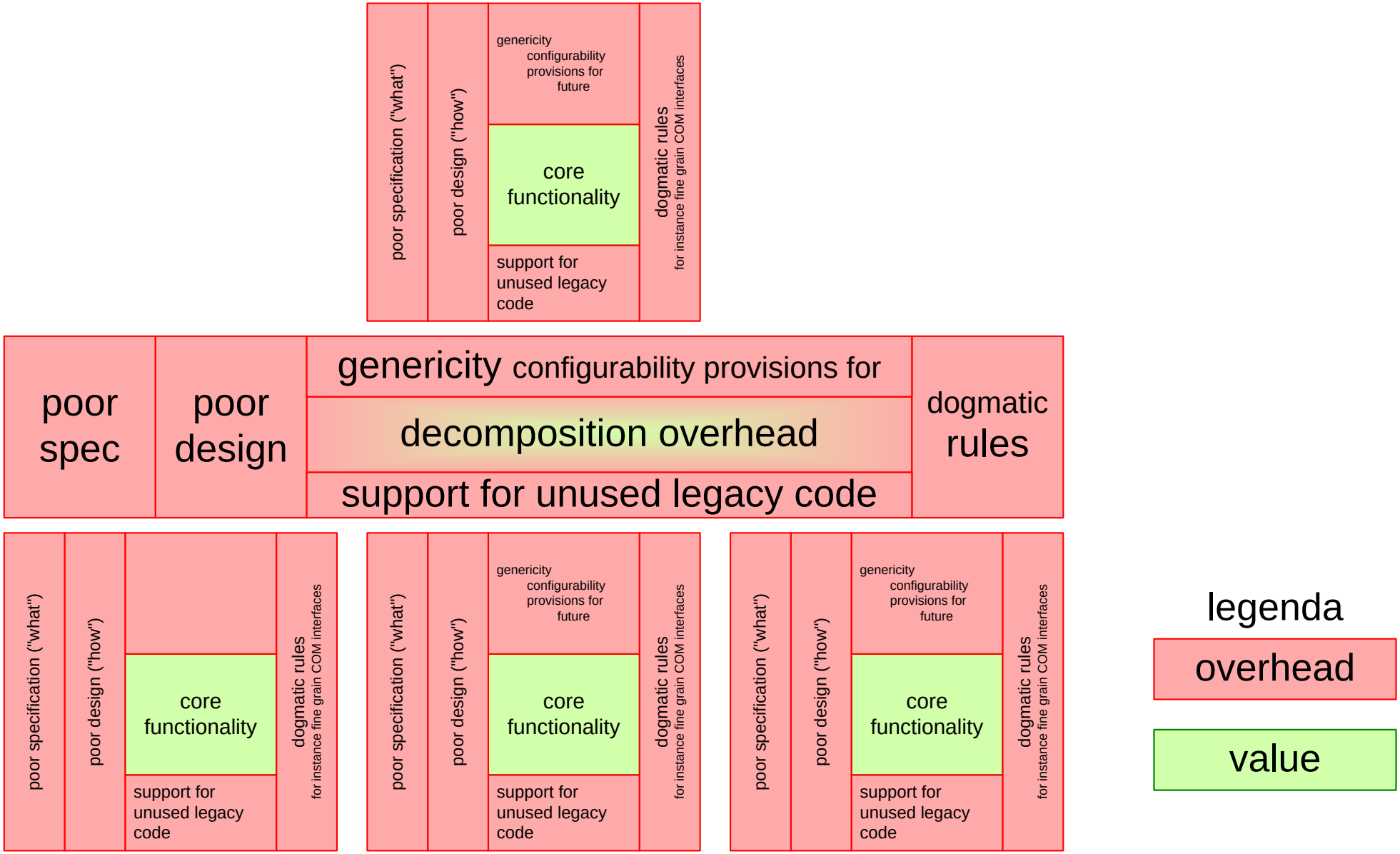
Shit propagation via copy paste



Example of shit propagation

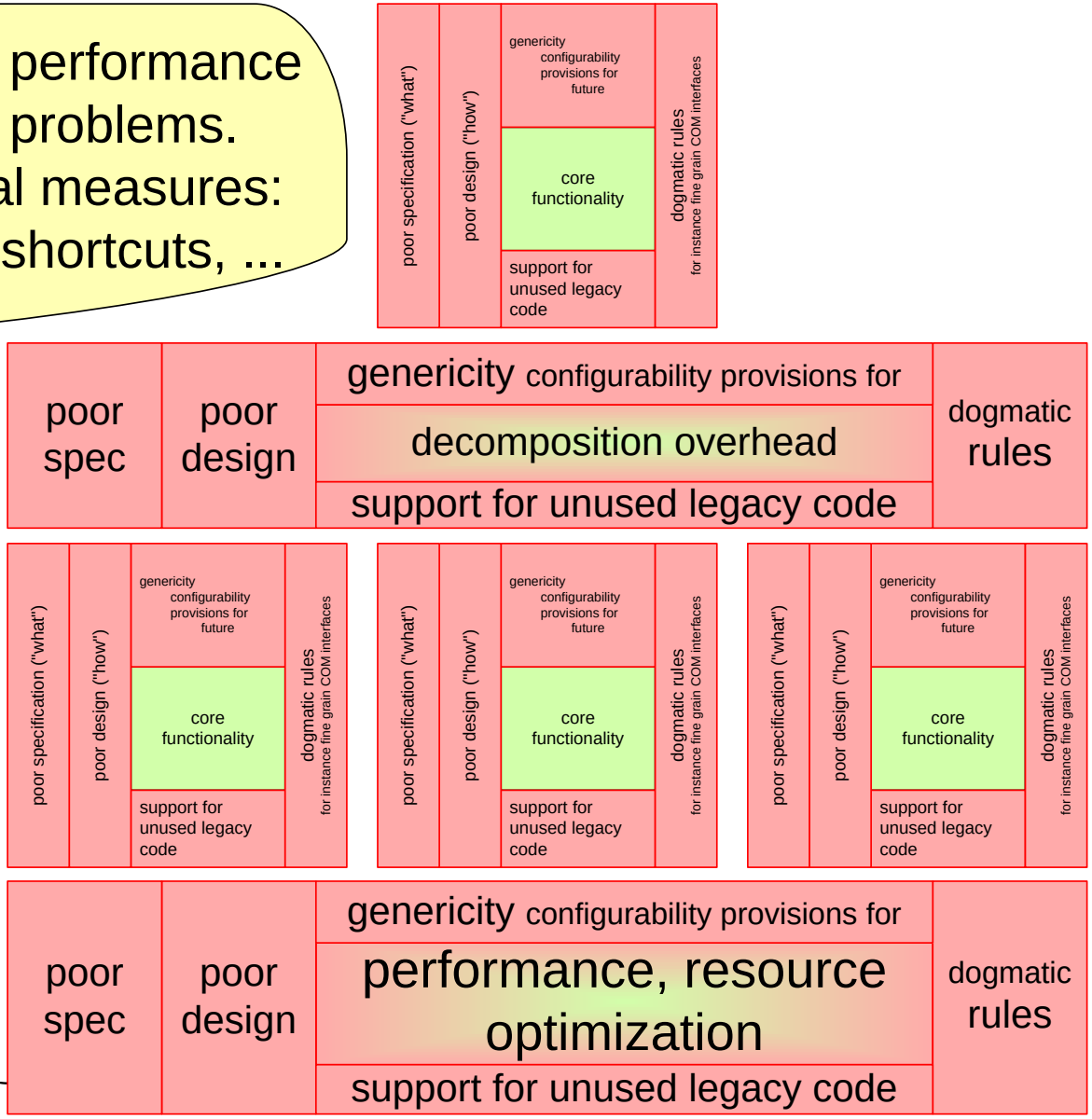


Bloating causes more bloating

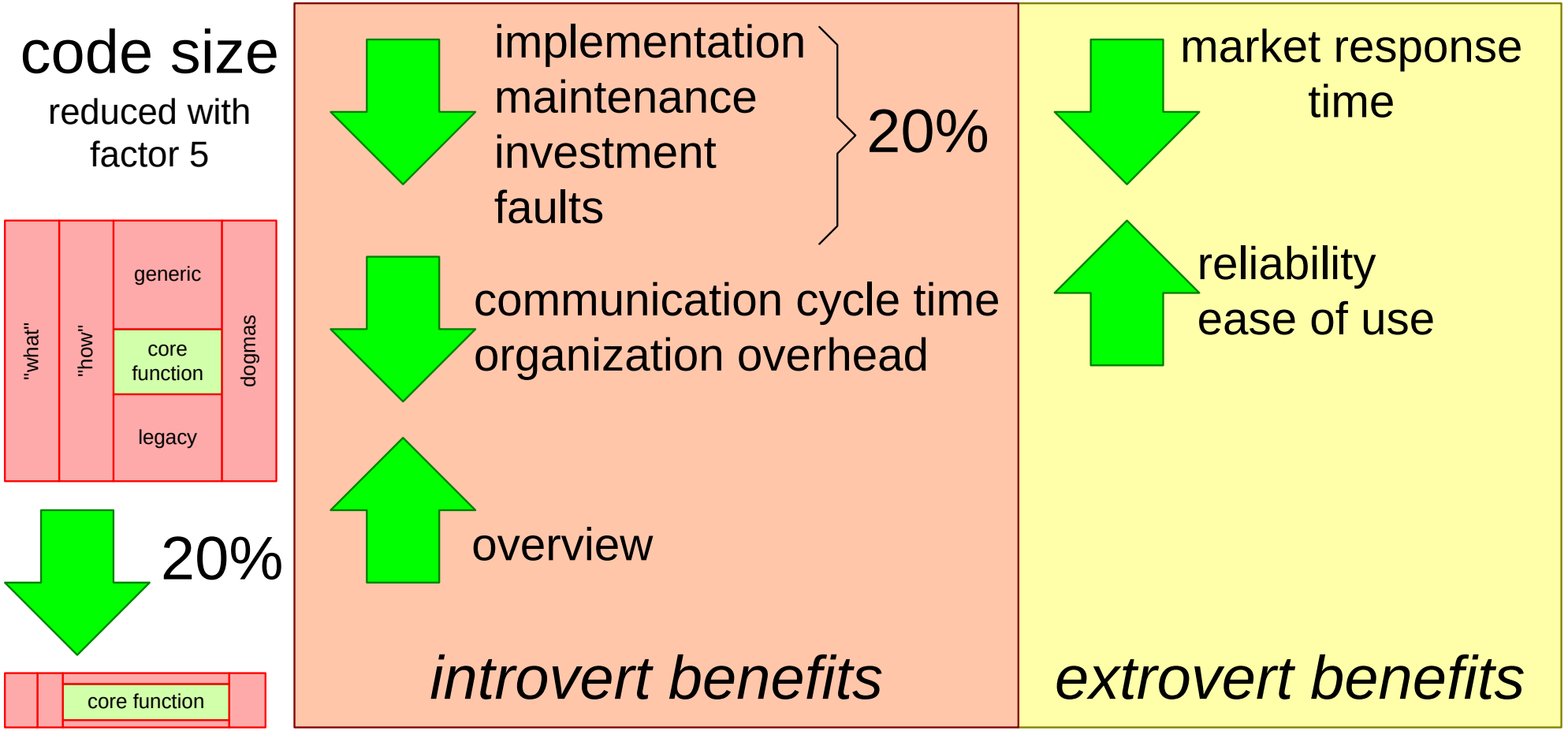


Causes even more bloating...

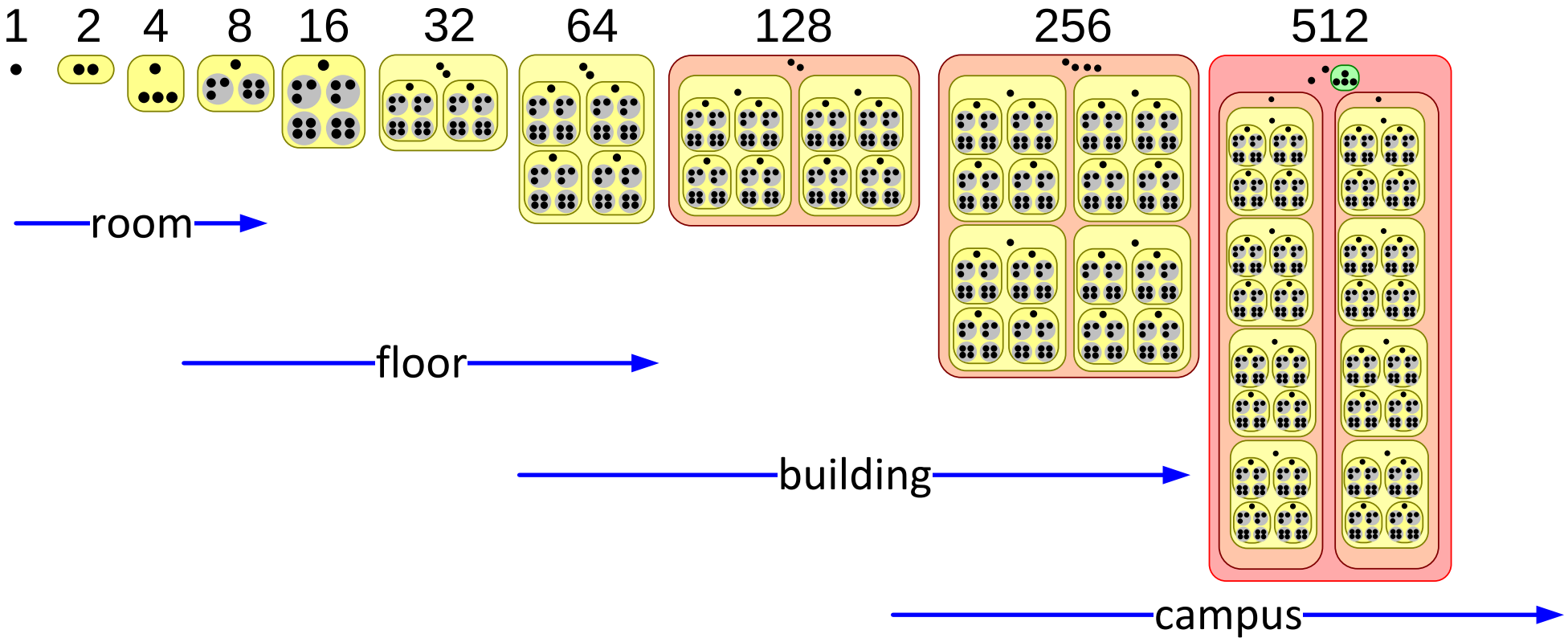
Bloating causes performance and resource problems.
Solution: special measures: memory pools, shortcuts, ...



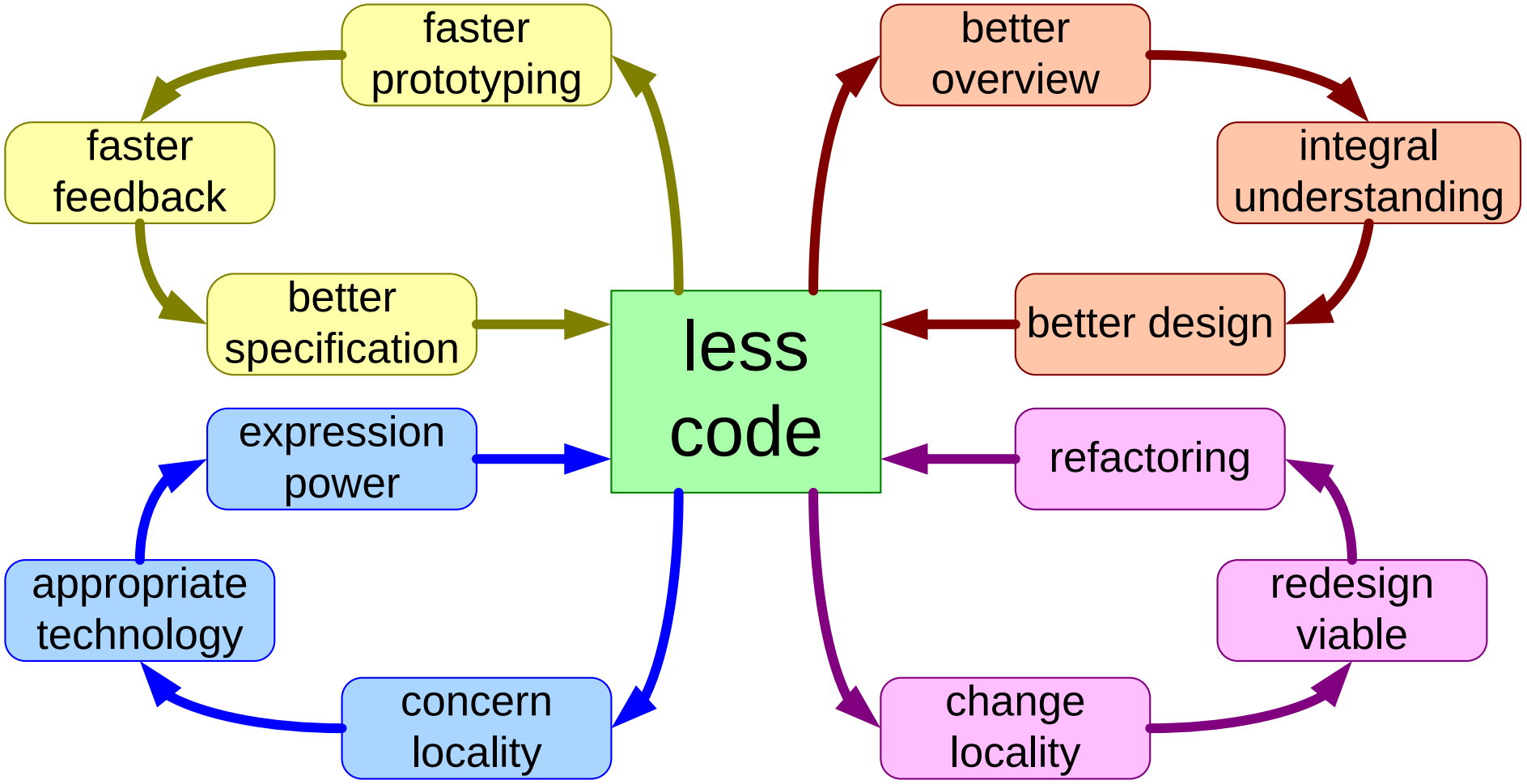
What if we remove half of the bloating?



Impact of size on organization, location, process

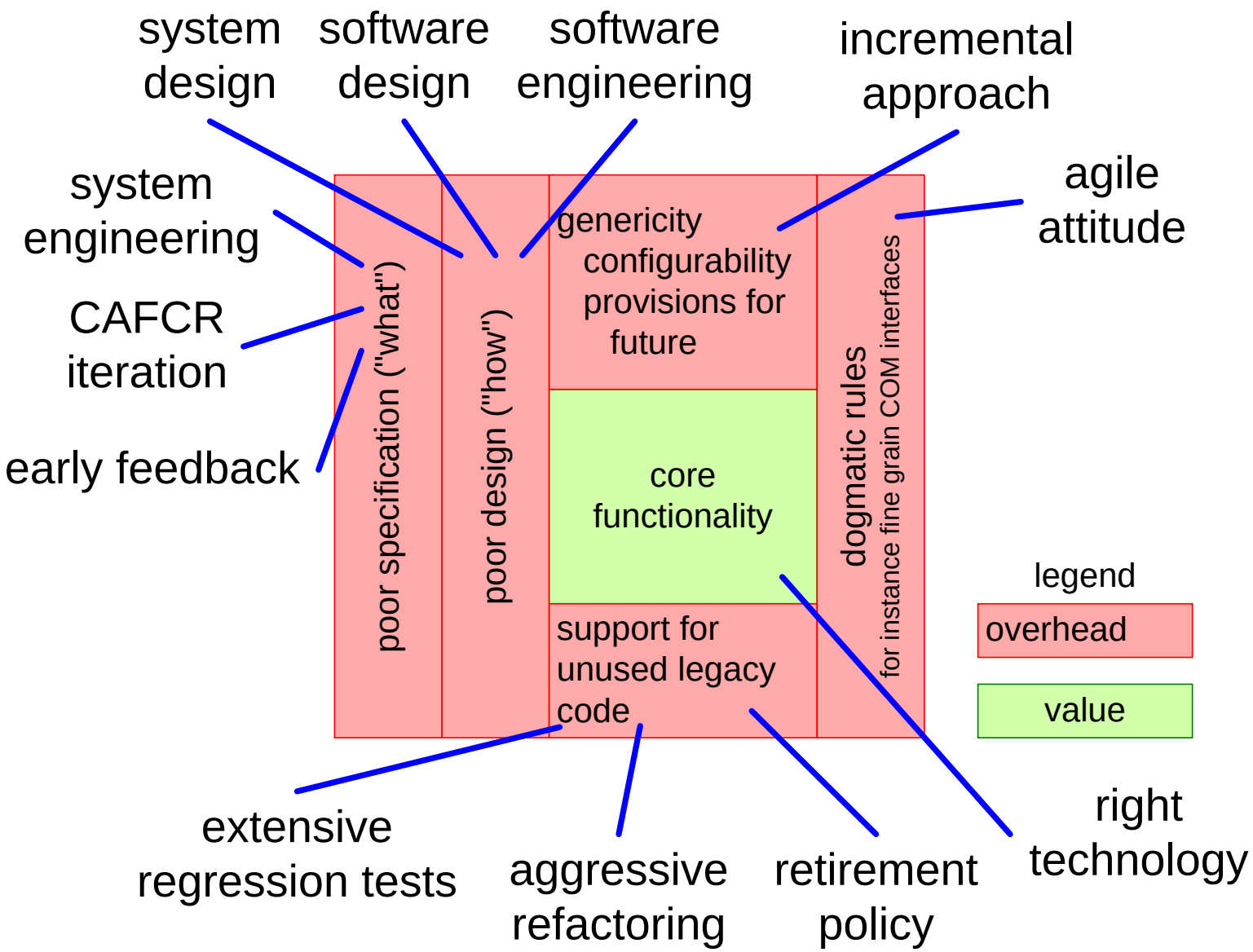


Anti bloating multiplier



same type of diagram can be made for less people
(less communication, space, organization, bureaucracy)

How to reduce bloating

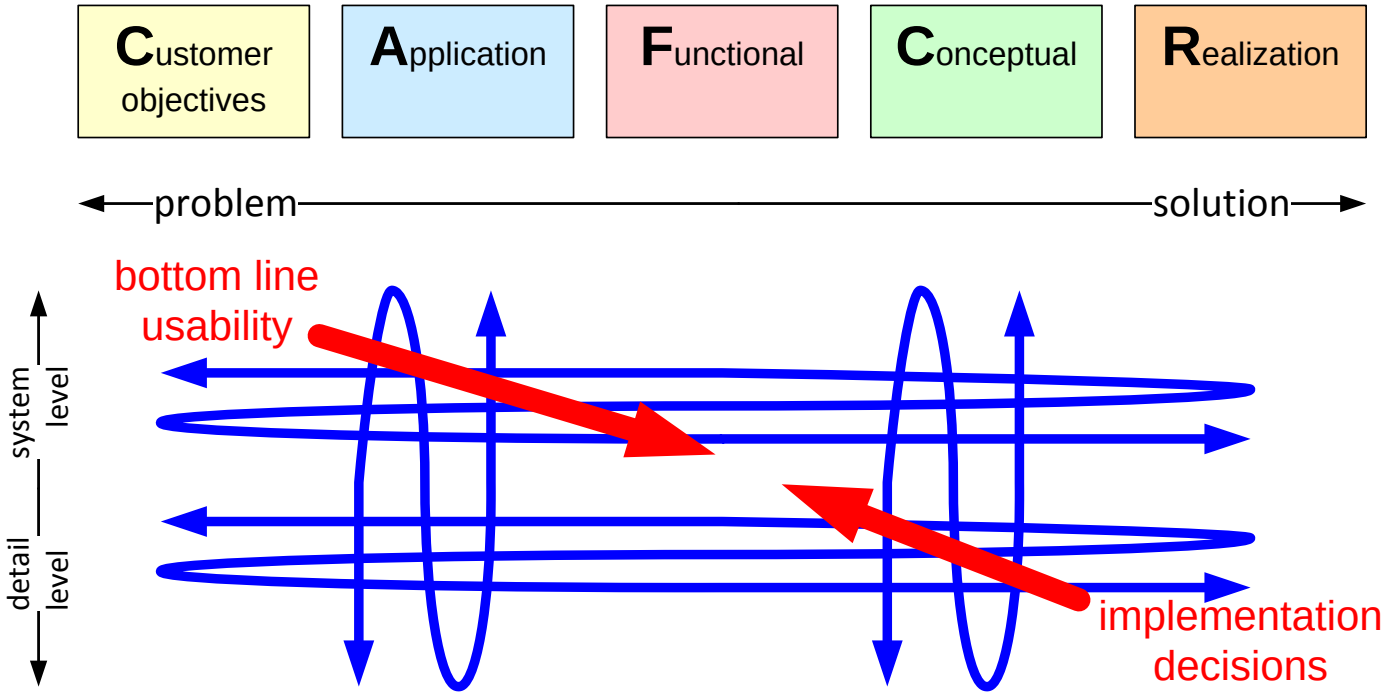


Improving the specification

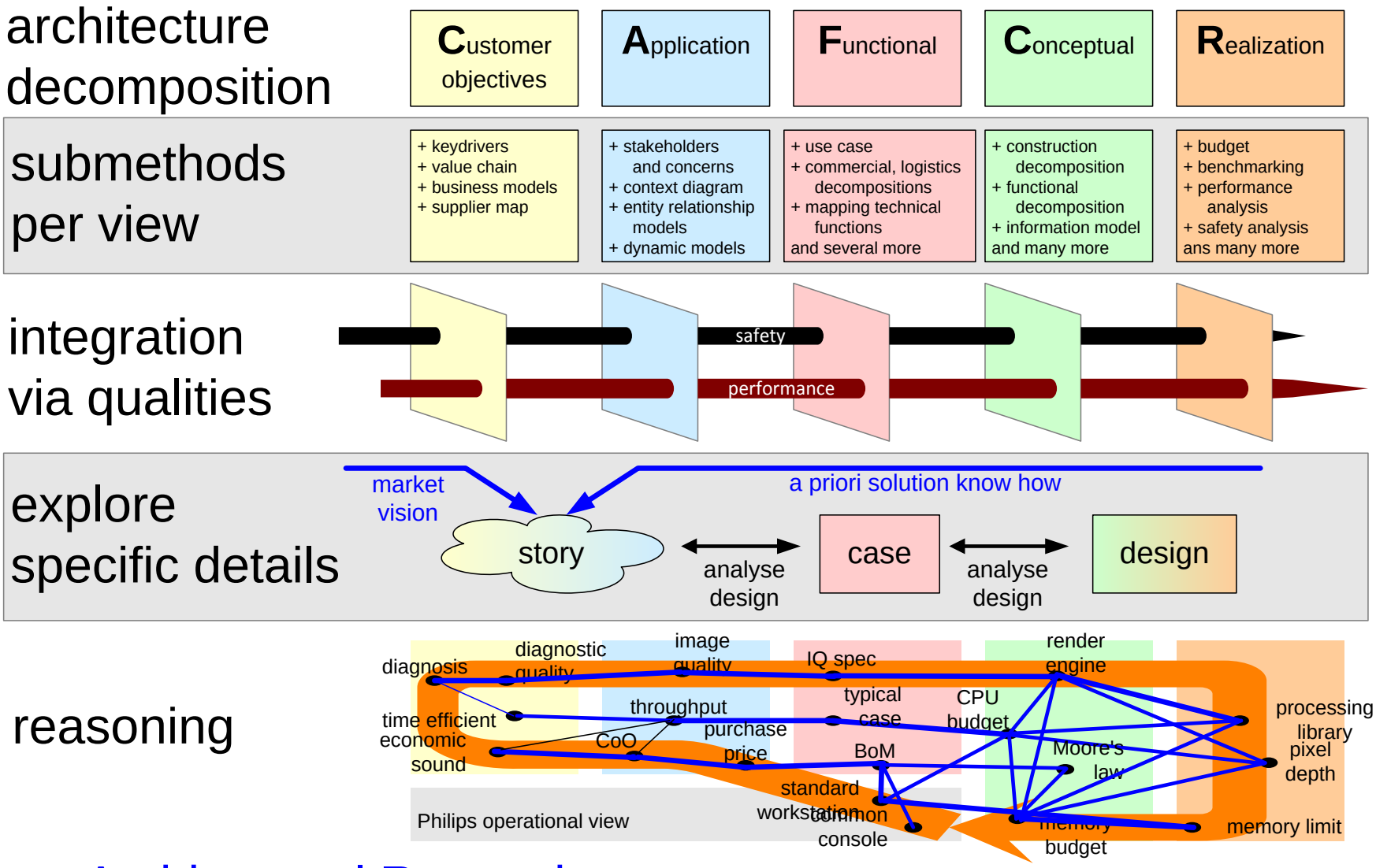
poor specification ("what")

system engineering: mature discipline, checklists, literature

CAFCR iteration, early **feedback:** learn **why**



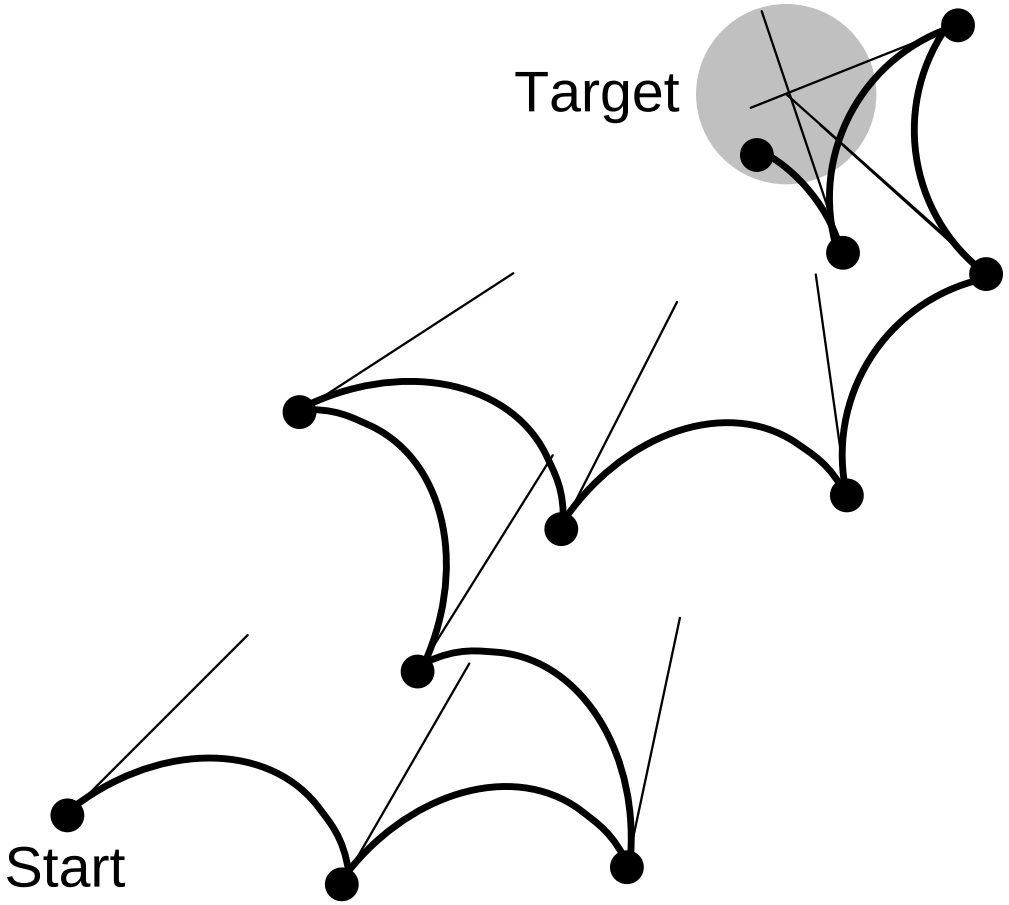
Improve design: use multiple views and methods



See: Architectural Reasoning http://www.extra.research.philips.com/natlab/sysarch/ArchitecturalReasoning.html

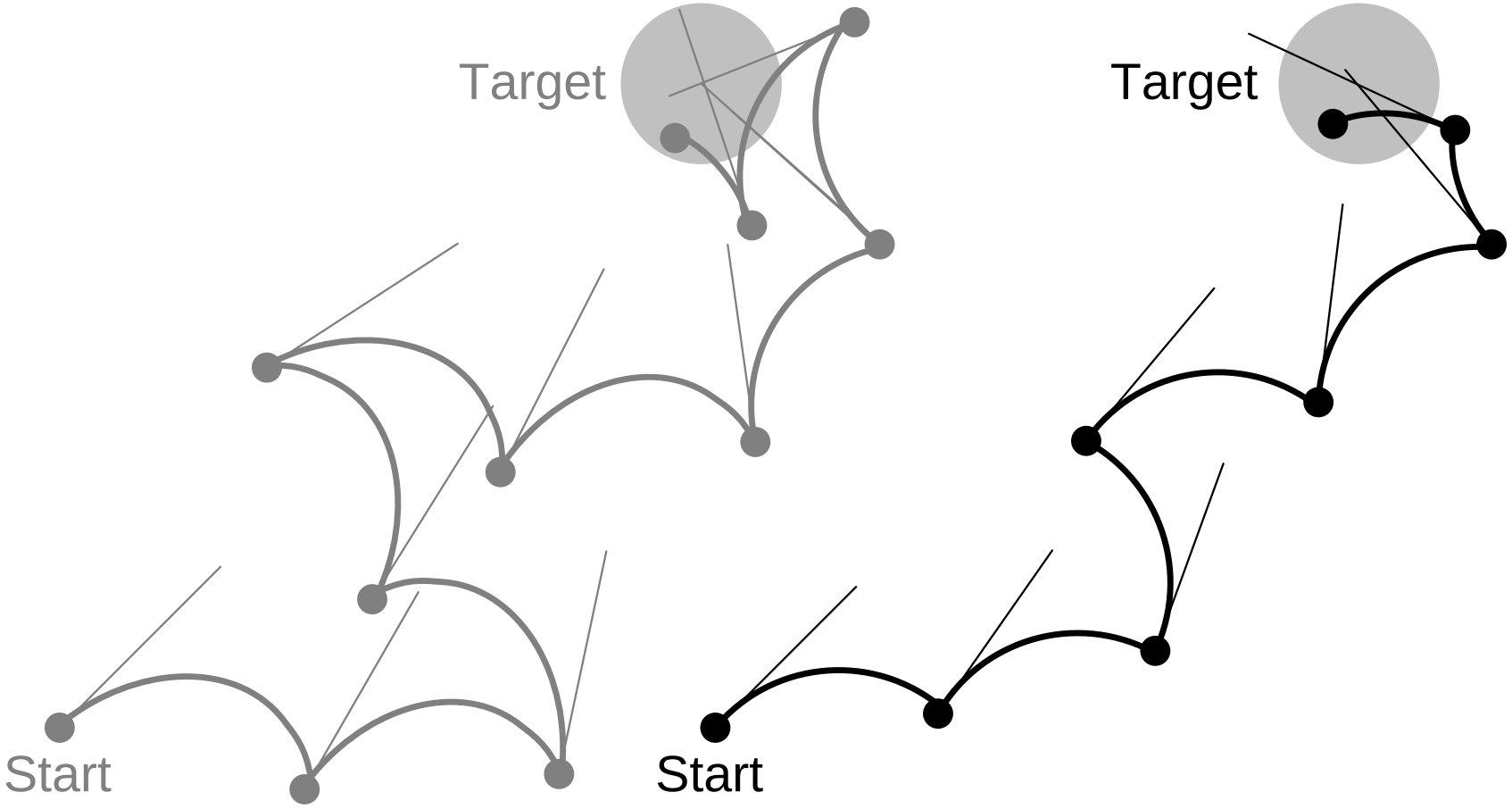
Feedback

stepsize: 3 months
elapsed time: 25 months

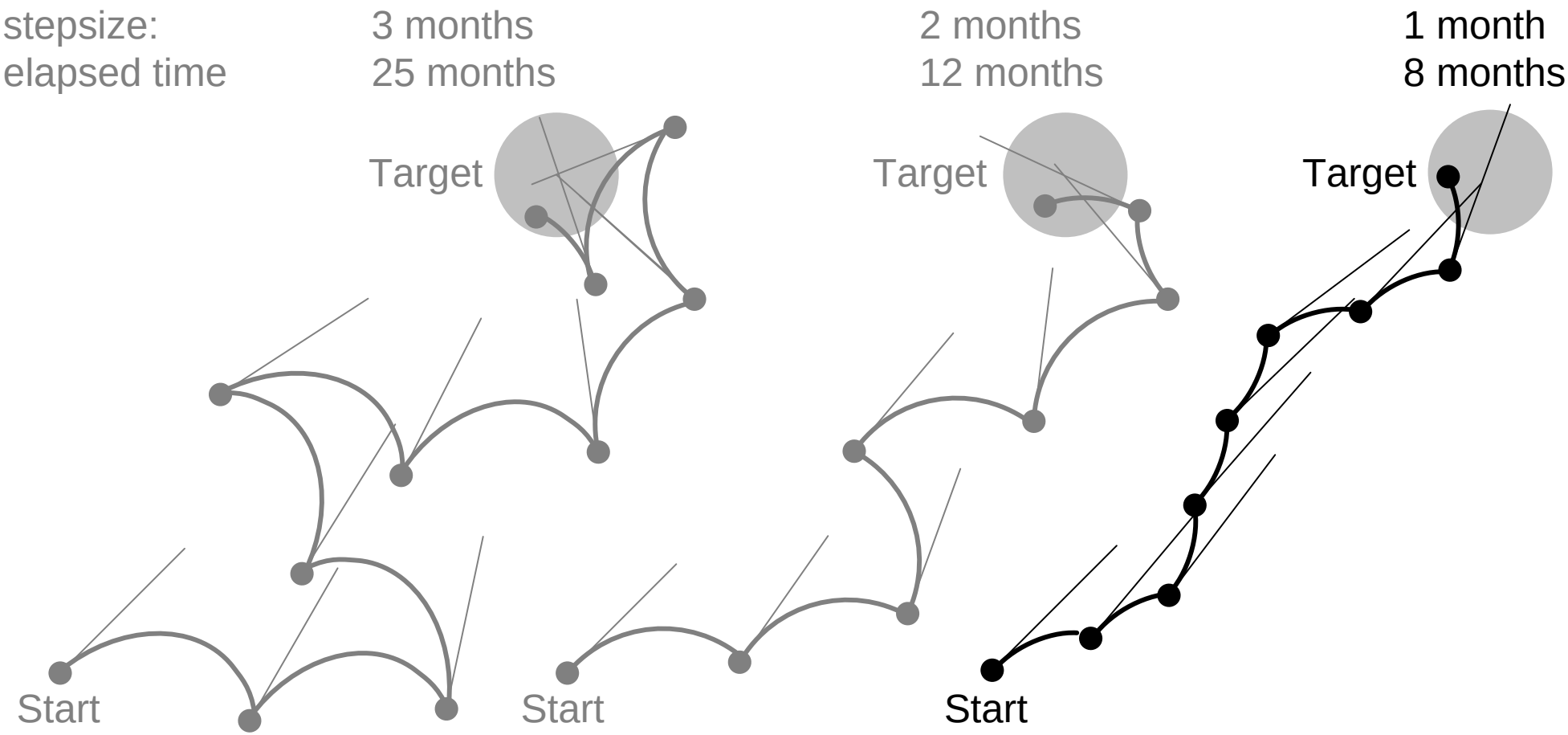


Feedback (2)

stepsize: 3 months 2 months
elapsed time 25 months 12 months

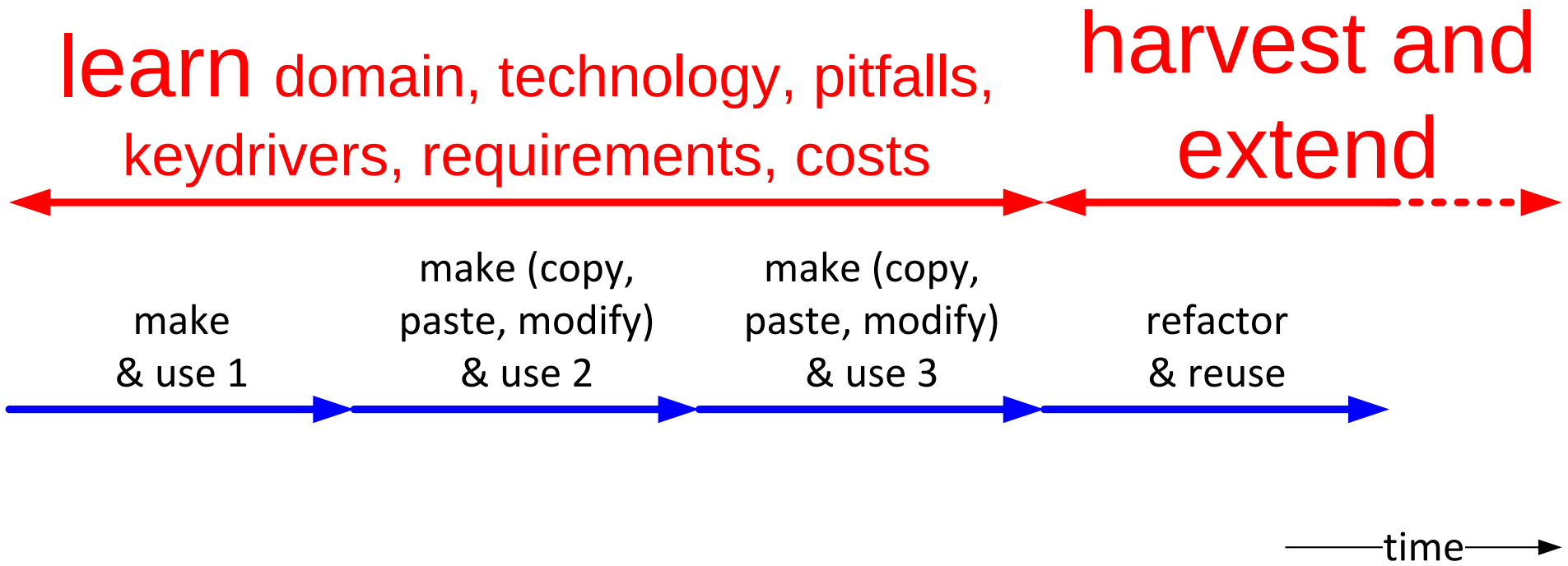


Feedback (3)



Small feedback cycles result in Faster Time to Market

Lesson learned about reuse

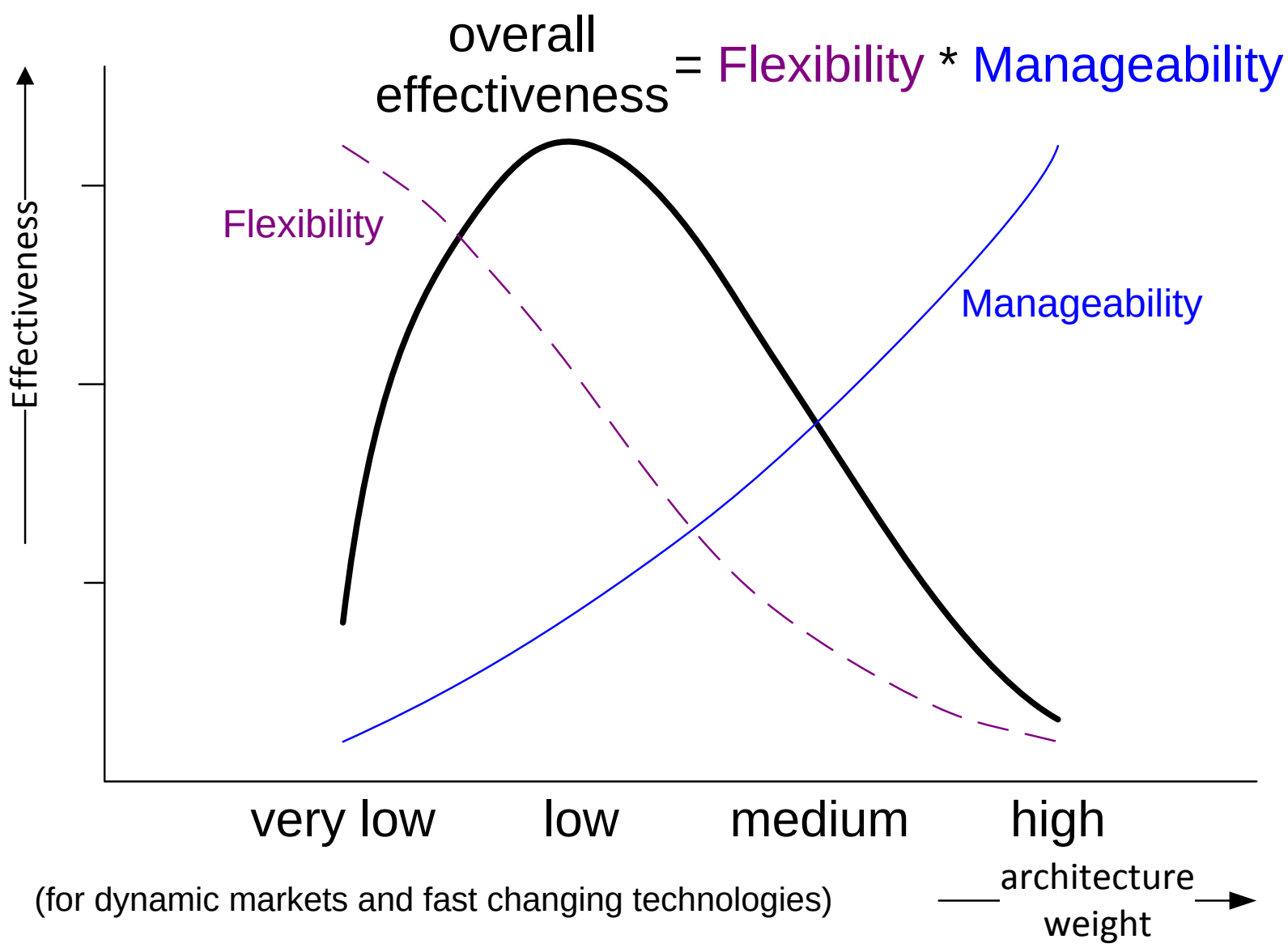


heuristic: use 3 times before factoring out the generic parts

Examples of ”right” technology choices

UI prototyping:	GUI editor/generator
non hard real time textual, algorithmic, networking:	 Python
small hard real-time or extremely performance critical	hand optimized
highly repeatable problem	dedicated generator tools

Keep the architecture weight low



support for unused legacy code

retirement policy

make explicit what can not be used anymore

aggressive refactoring

cleanup

extensive regression tests

reduce fear
reduce surprises