

# Exploration of the bloating of software

by *Gerrit Muller* Buskerud University College

e-mail: [gaudisite@gmail.com](mailto:gaudisite@gmail.com)

[www.gaudisite.nl](http://www.gaudisite.nl)

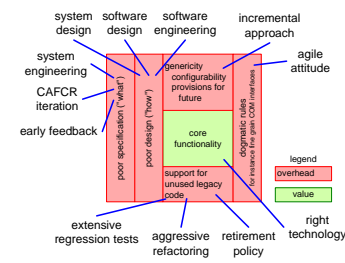
## Abstract

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.

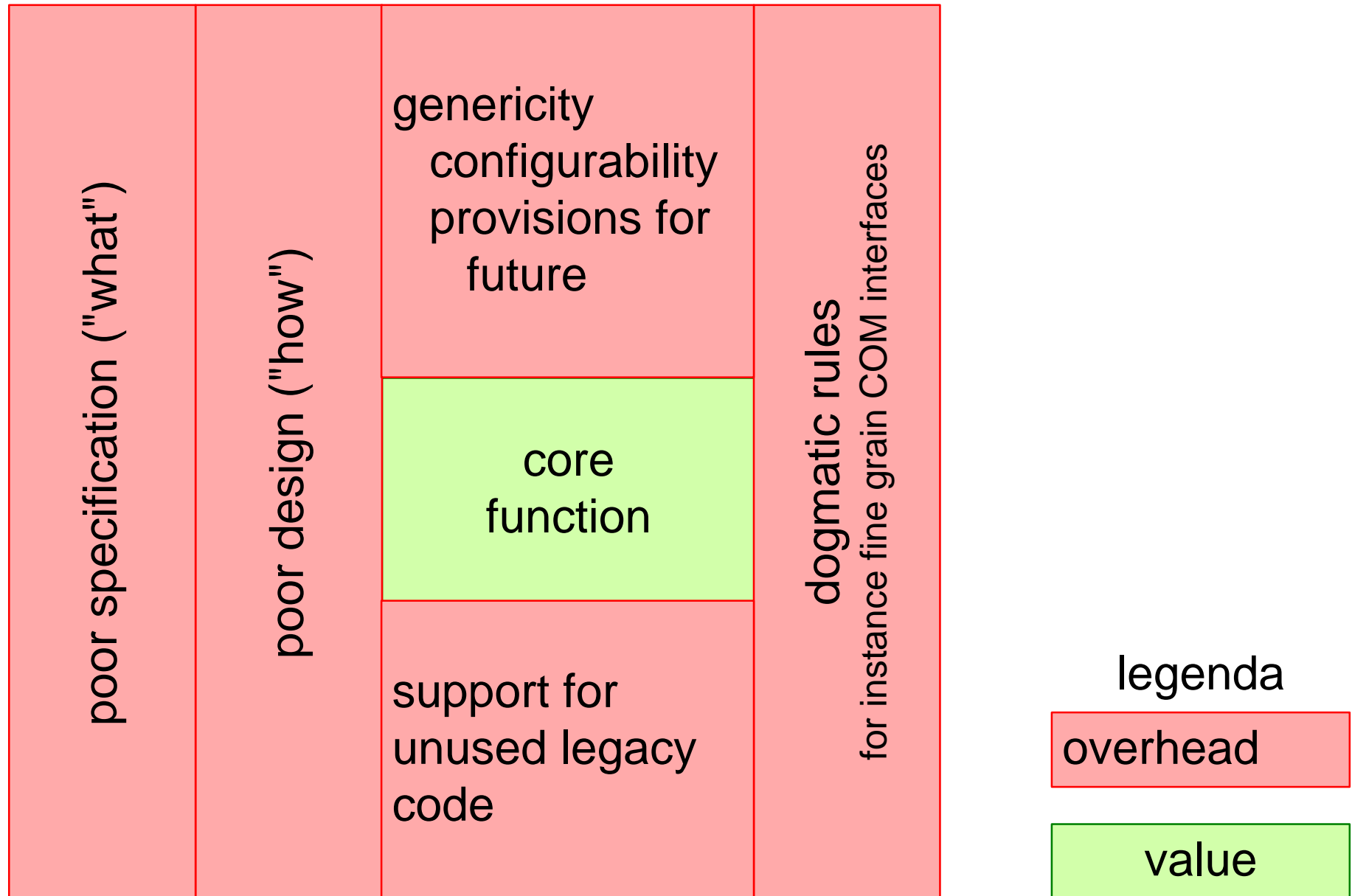
### Distribution

This article or presentation is written as part of the Gaudí project. The Gaudí project philosophy is to improve by obtaining frequent feedback. Frequent feedback is pursued by an open creation process. This document is published as intermediate or nearly mature version to get feedback. Further distribution is allowed as long as the document remains complete and unchanged.

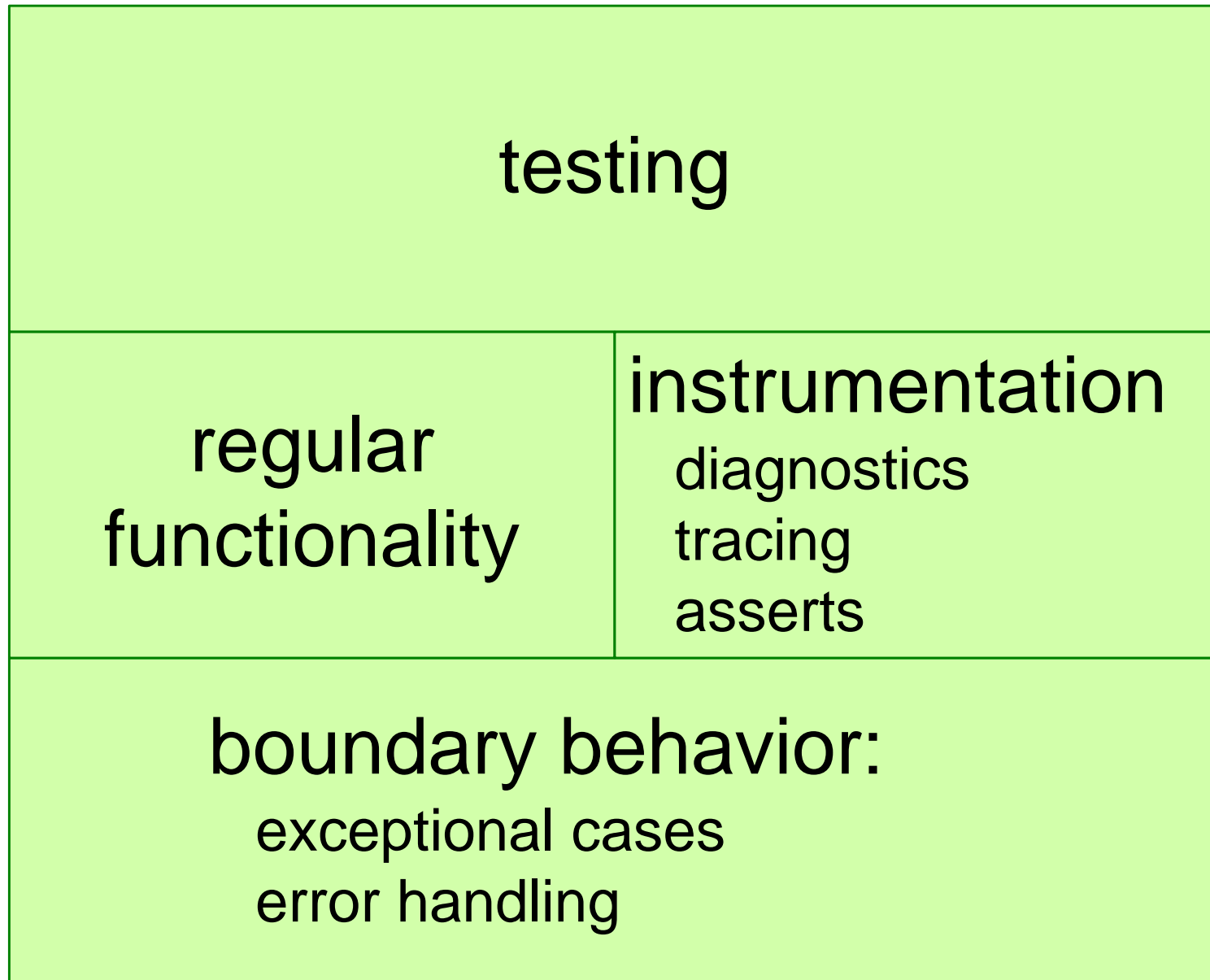
October 20, 2017  
status: finished  
version: 1.2



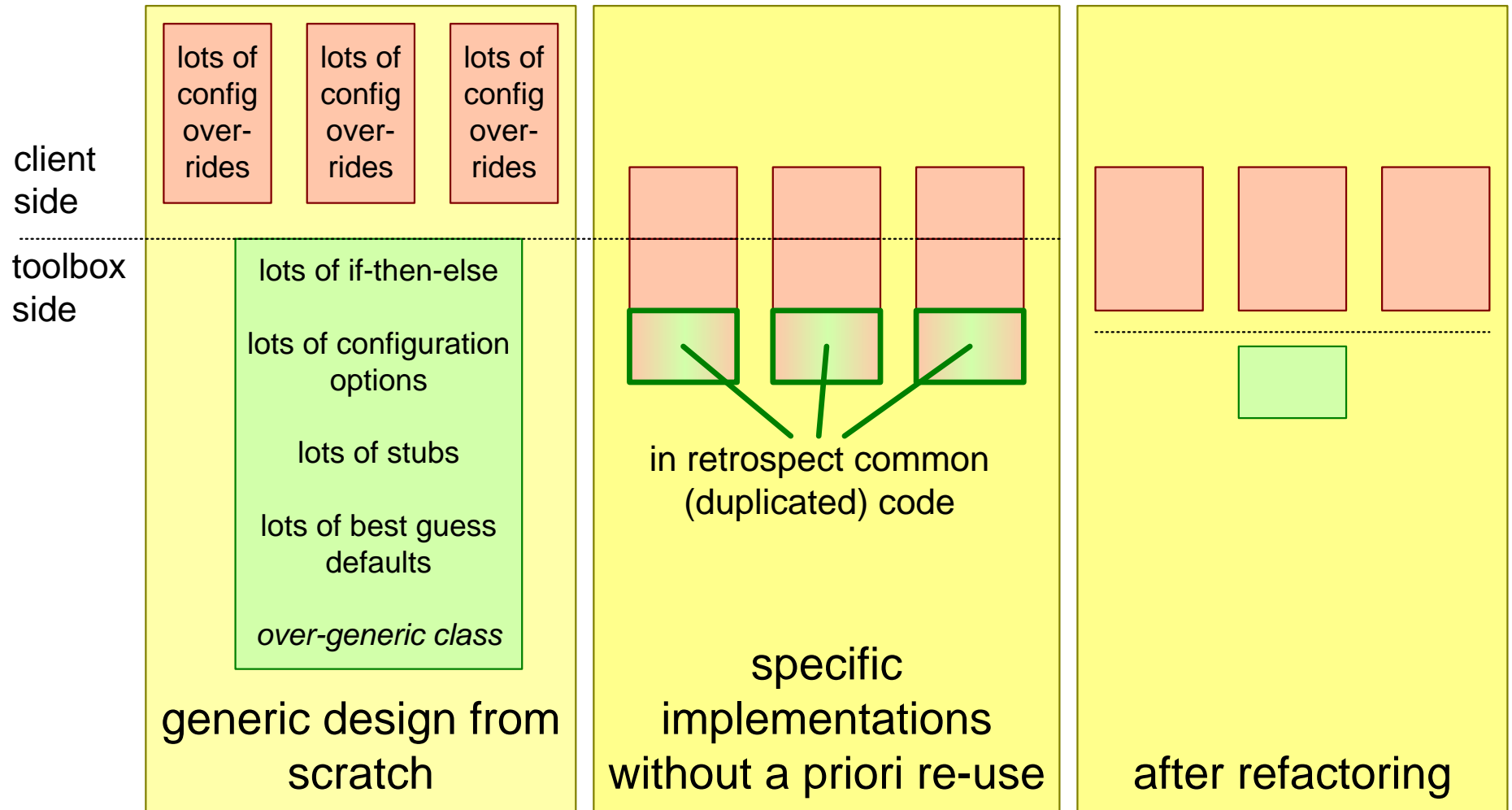
# Exploring bloating: main causes



# Necessary functionality $\gg$ the intended regular function

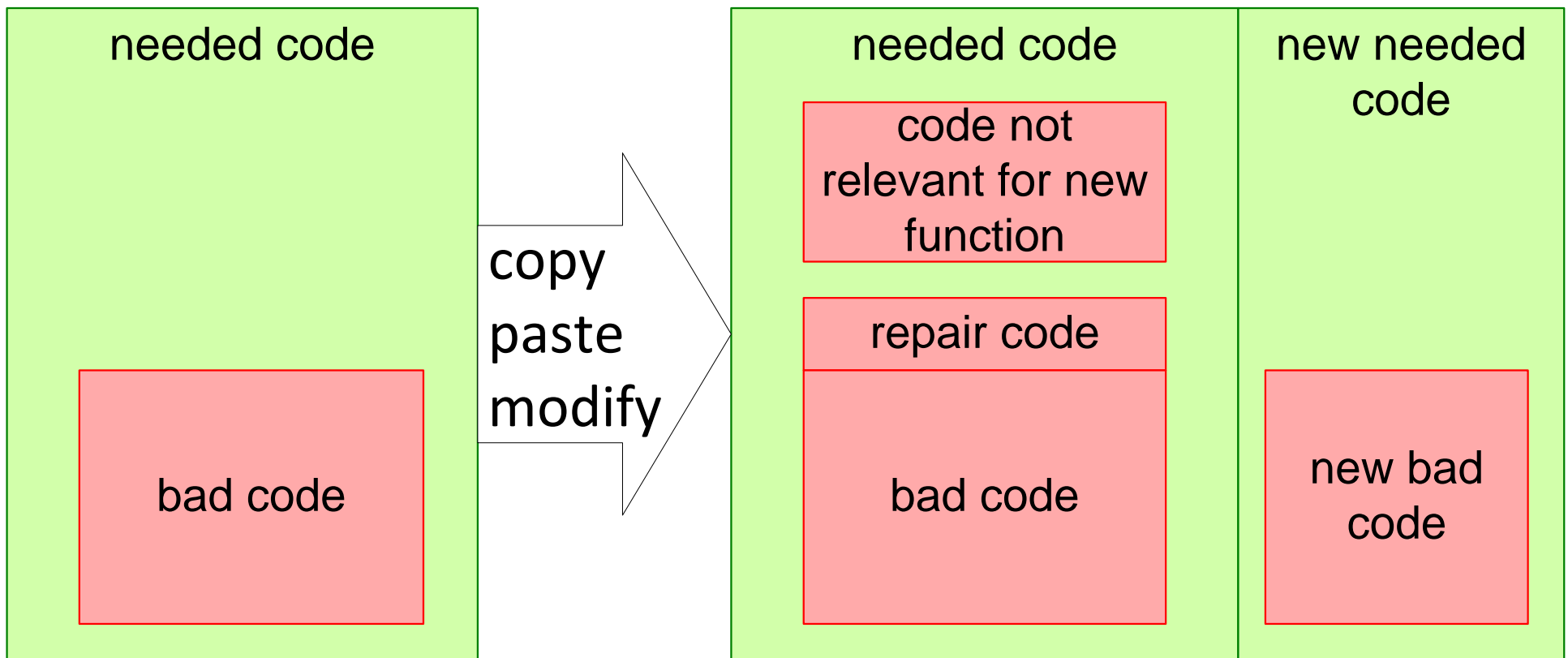


# The danger of being generic: bloating

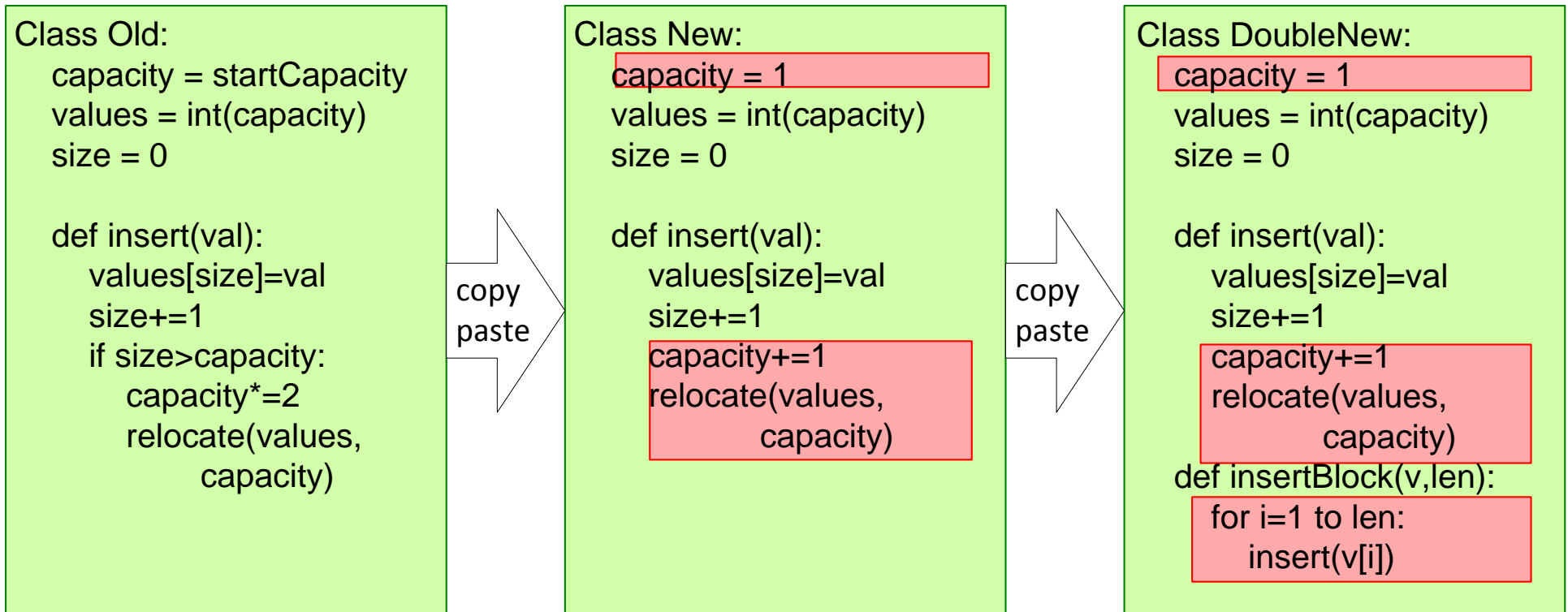


"Real-life" example: redesigned *Tool* super-class and descendants, ca 1994

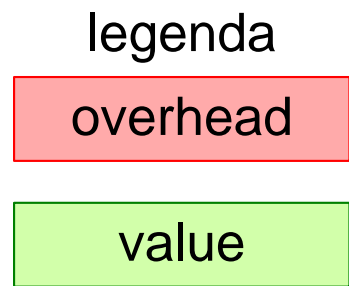
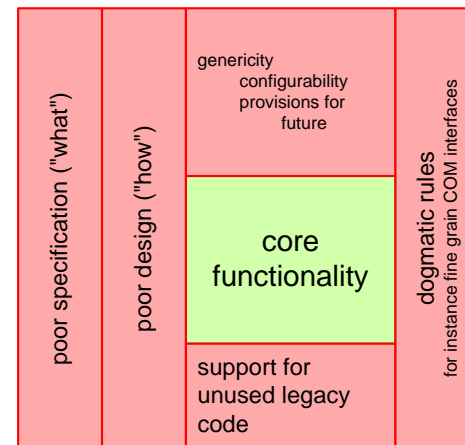
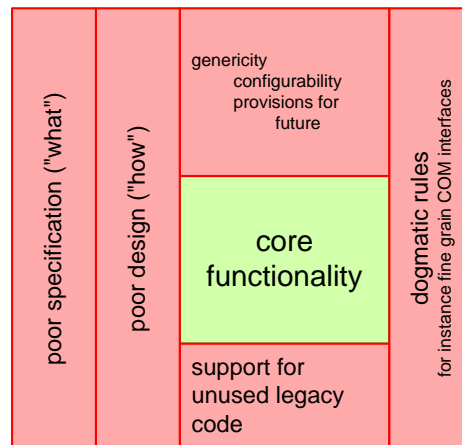
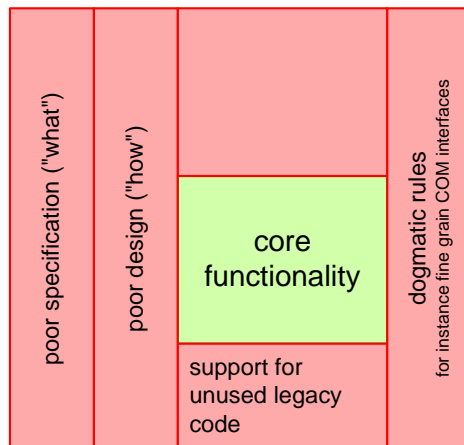
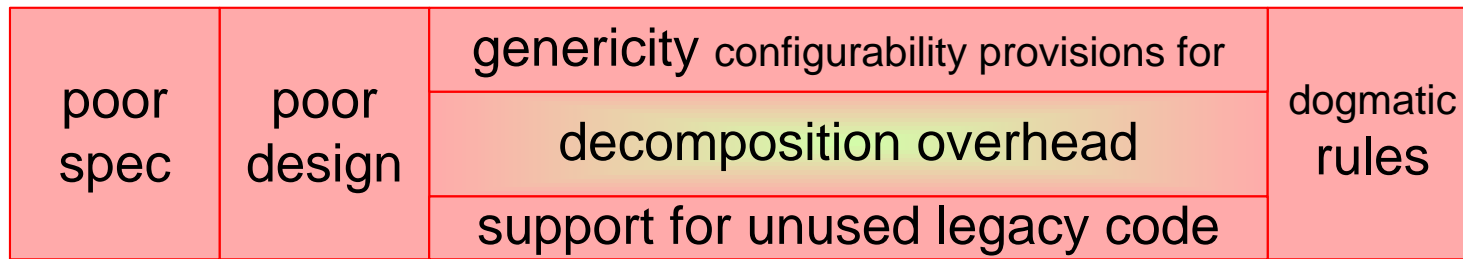
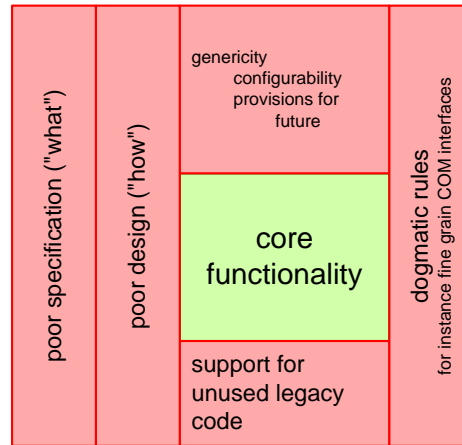
# Shit propagation via copy paste



# Example of shit propagation

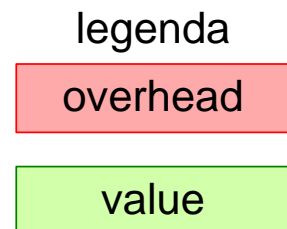
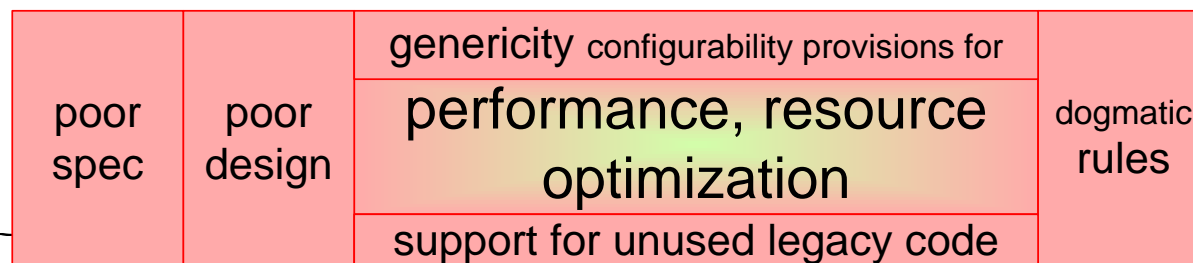
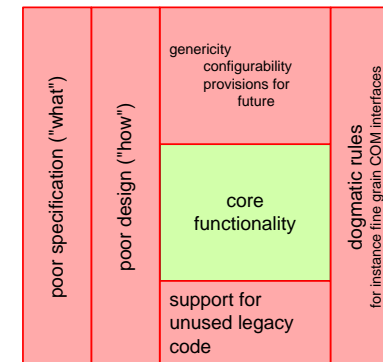
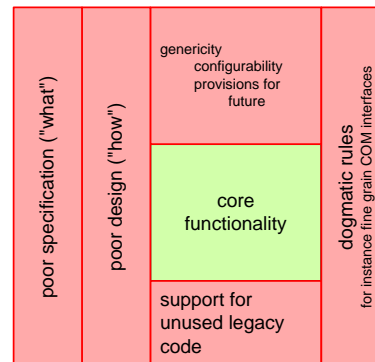
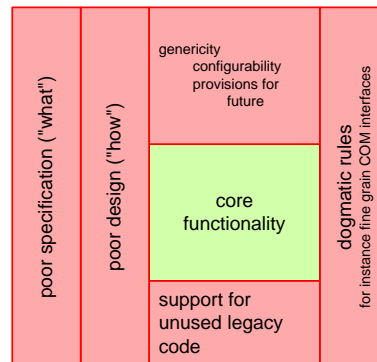
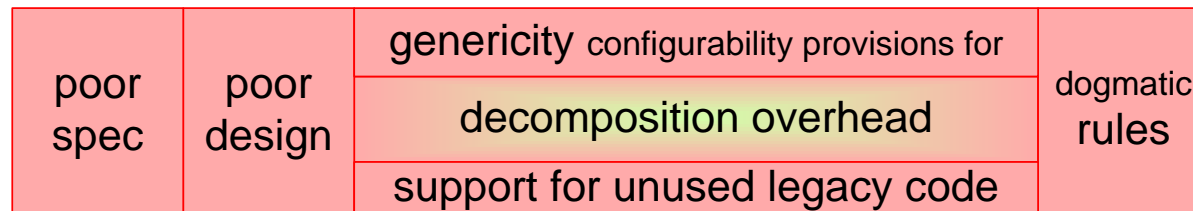
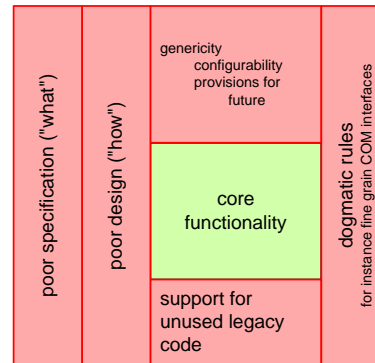


# Bloating causes more bloating



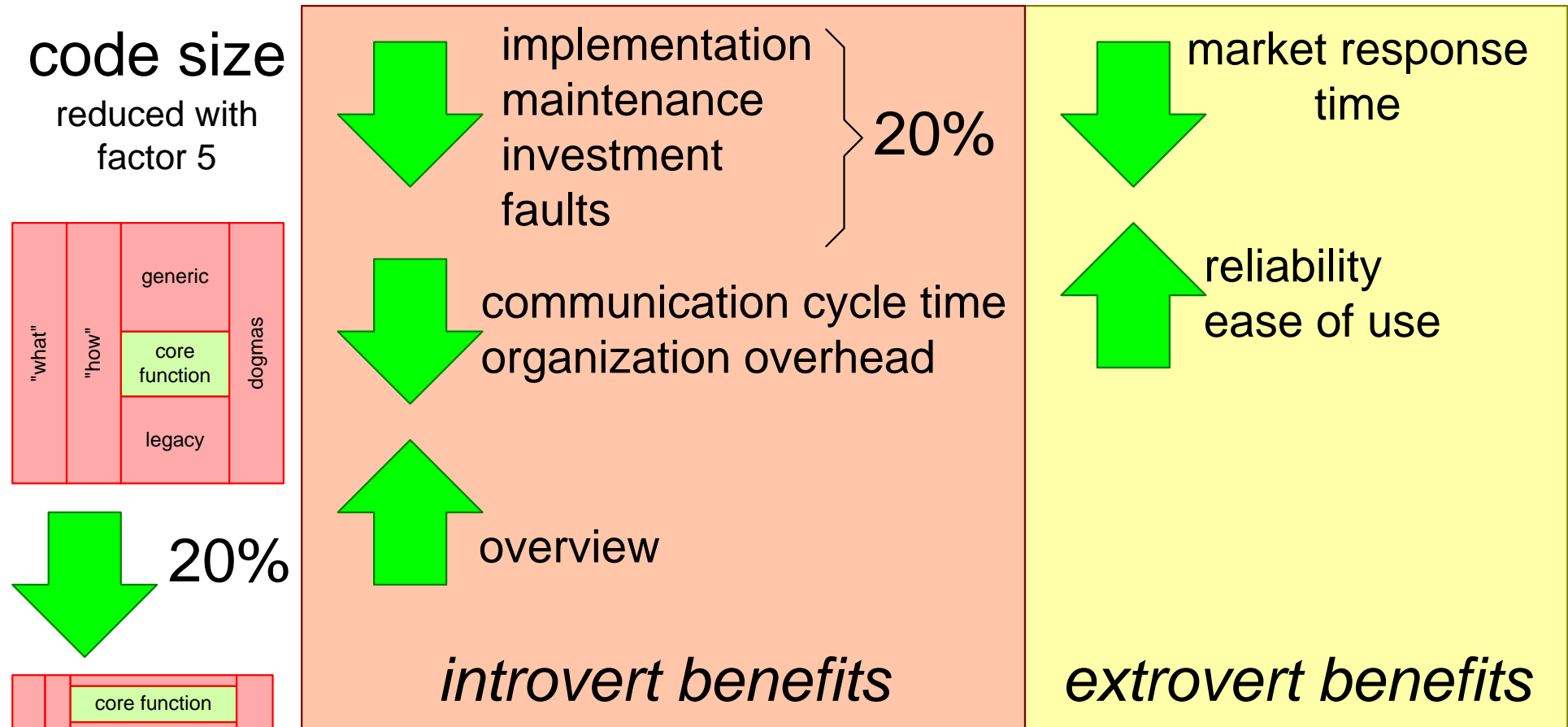
# Causes even more bloating...

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

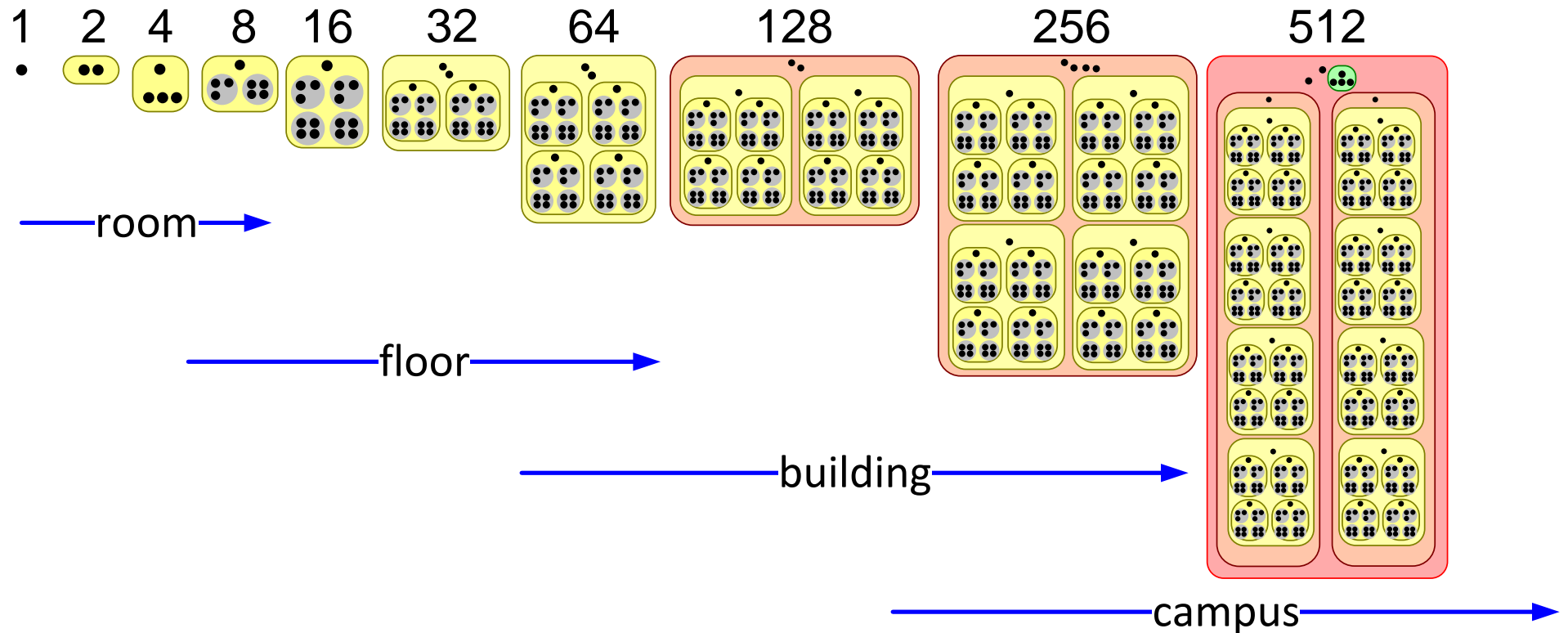




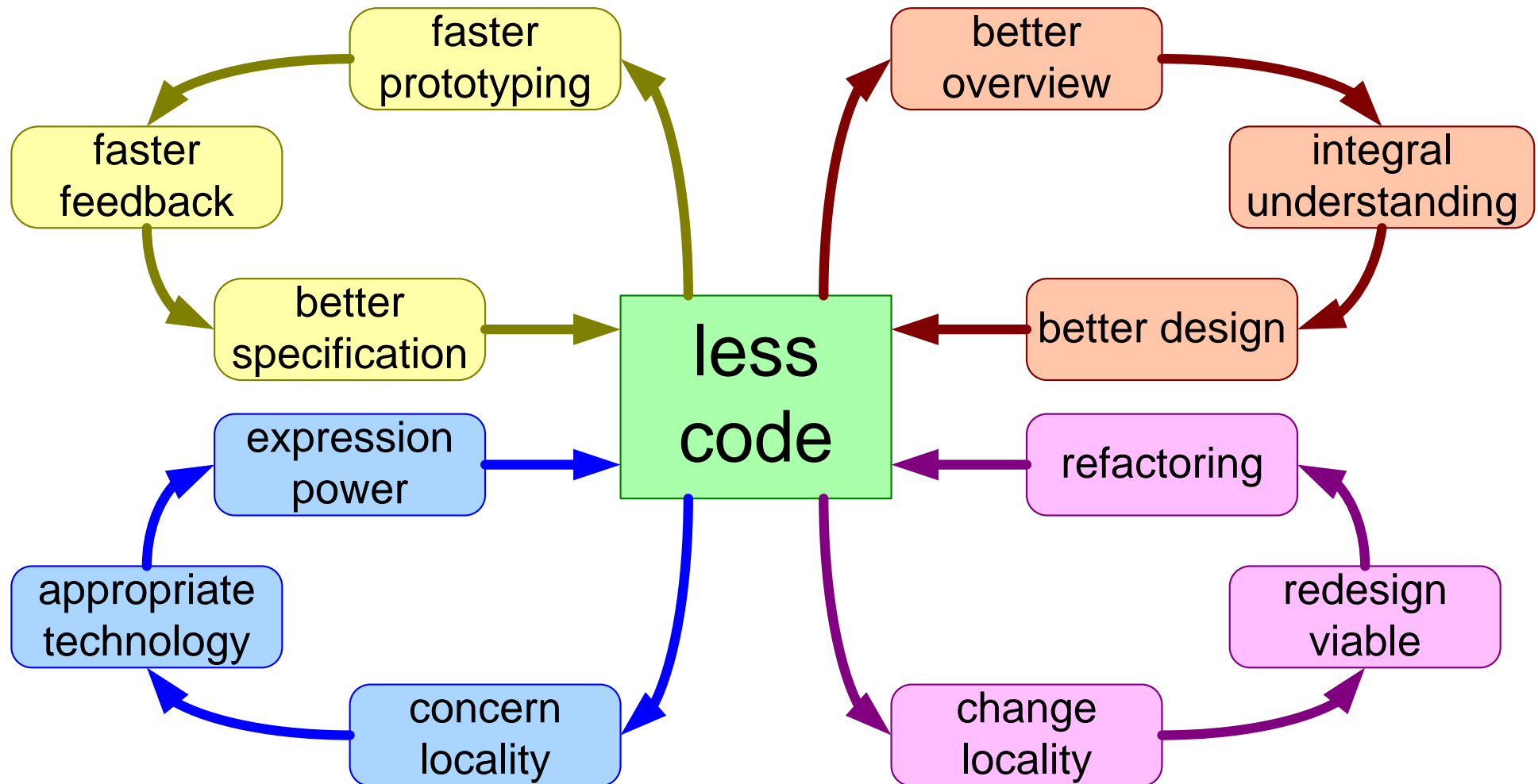
# What if we remove half of the bloating?



# Impact of size on organization, location, process

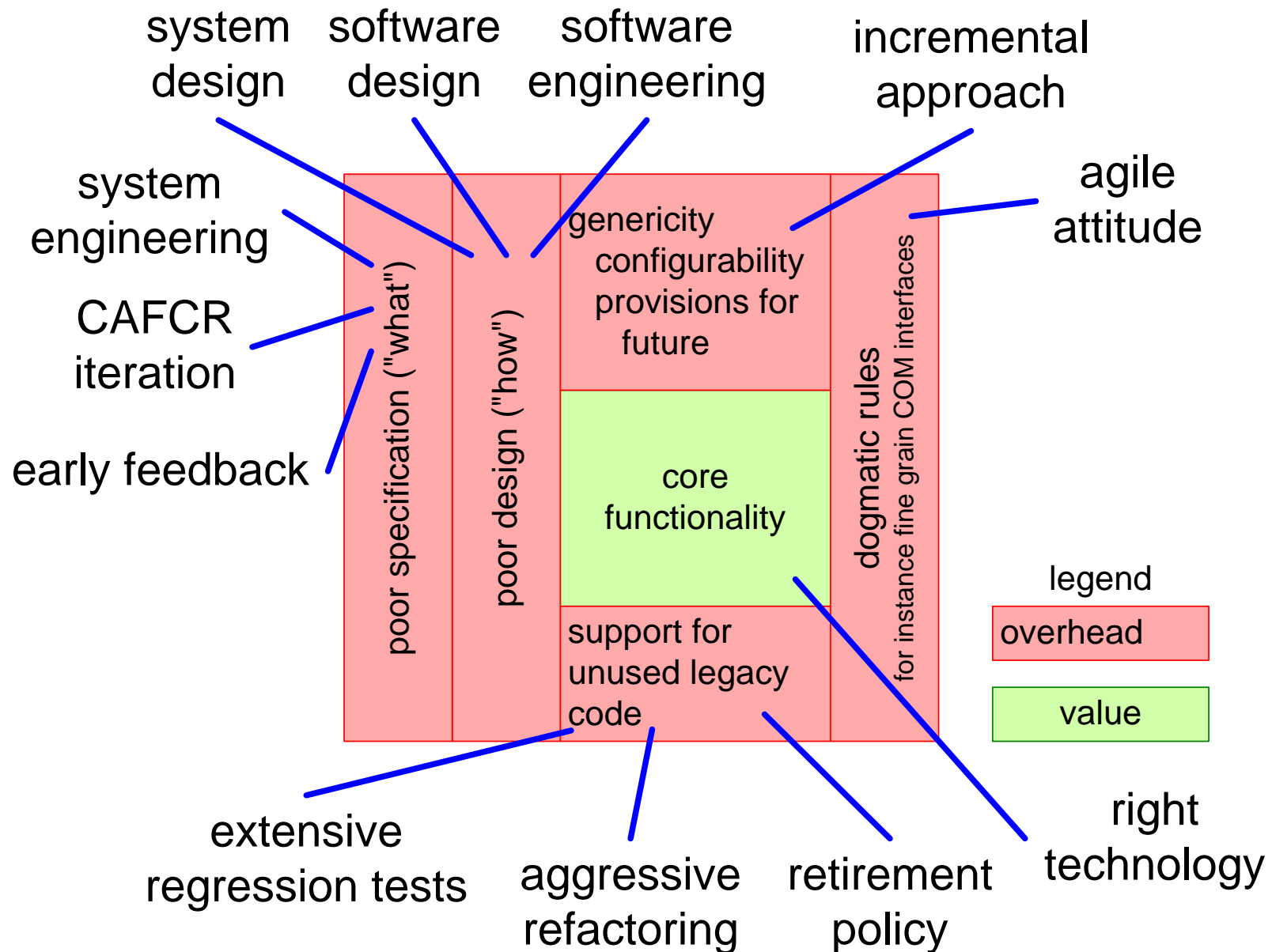


# Anti bloating multiplier



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

# How to reduce bloating

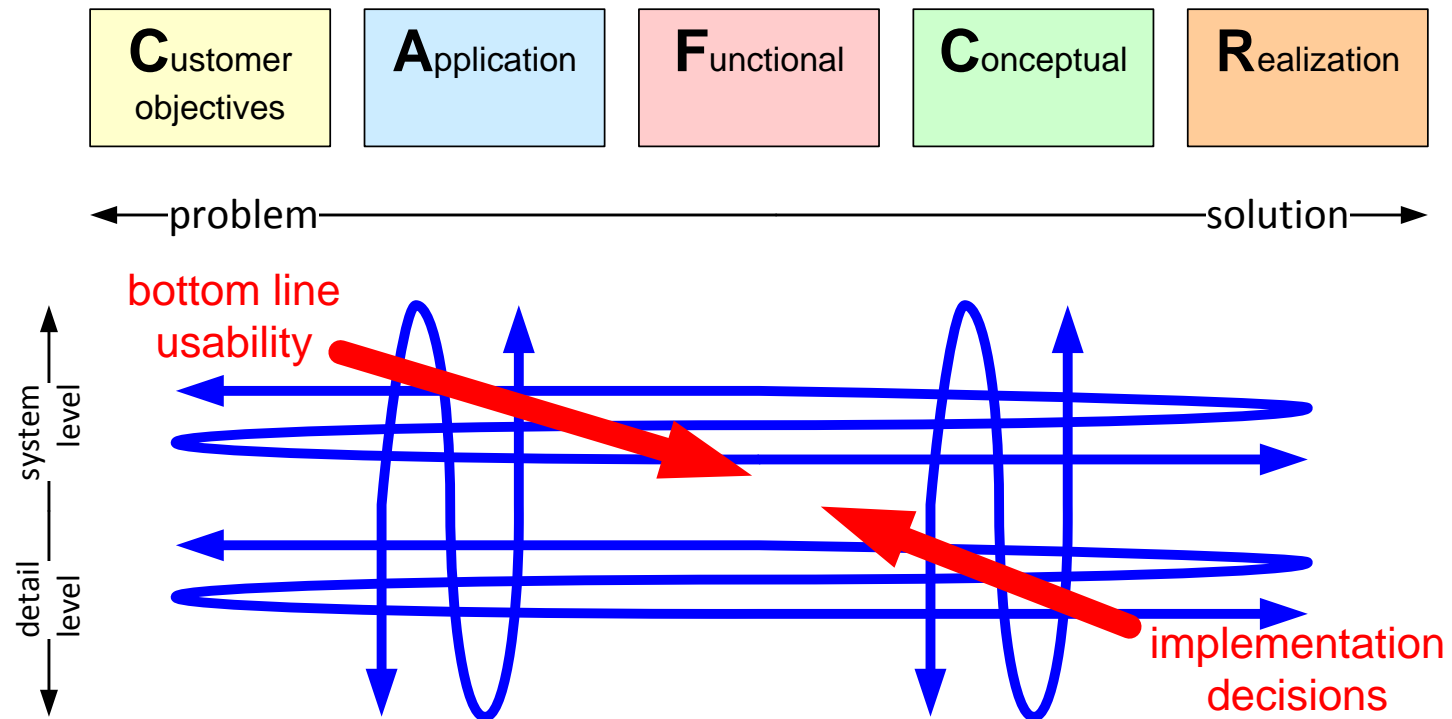


# Improving the specification

poor specification ("what")

system engineering: mature discipline, checklists, literature

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

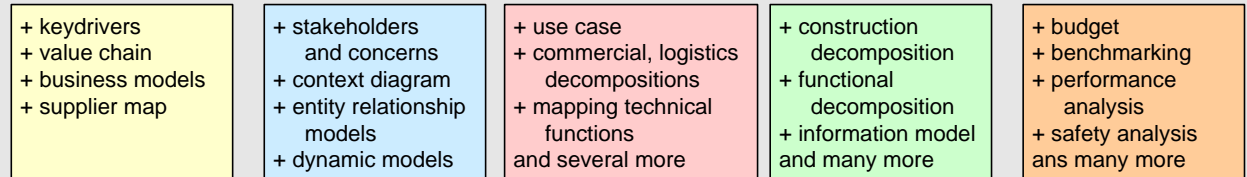


# Improve design: use multiple views and methods

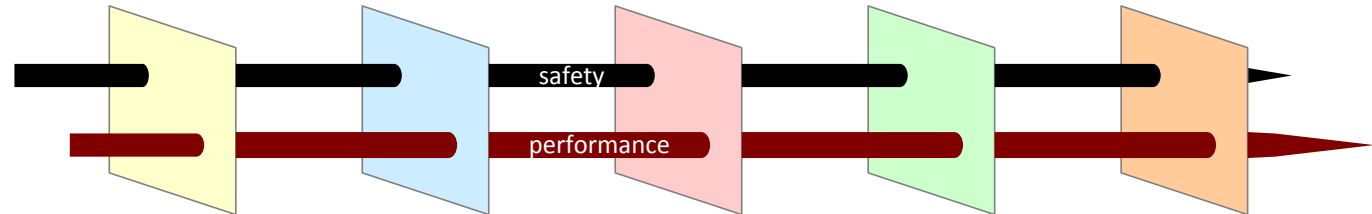
architecture decomposition



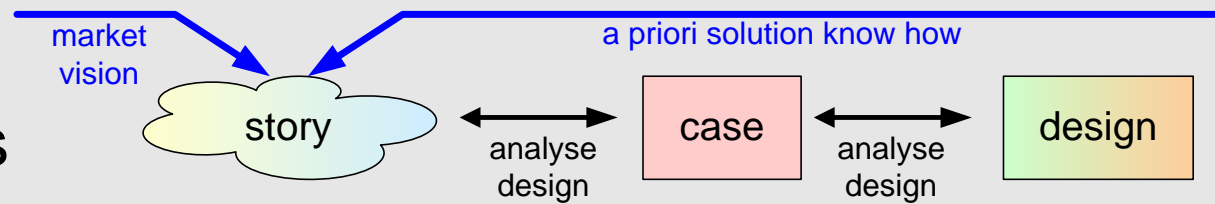
submethods per view



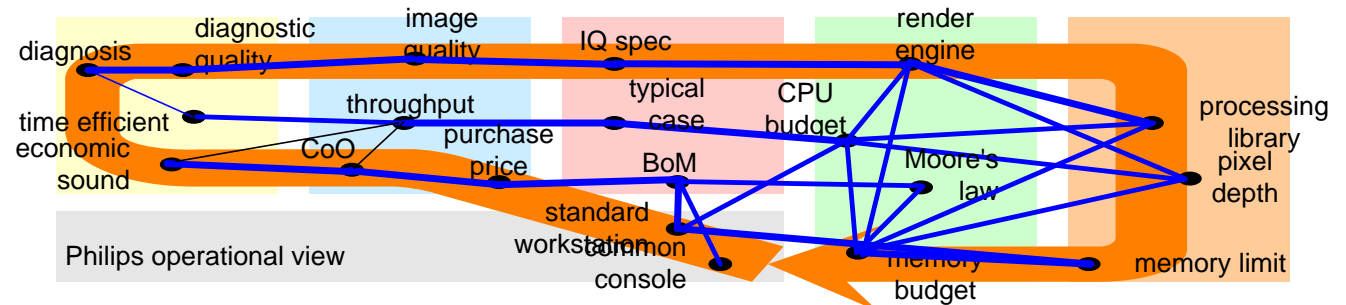
integration via qualities



explore specific details



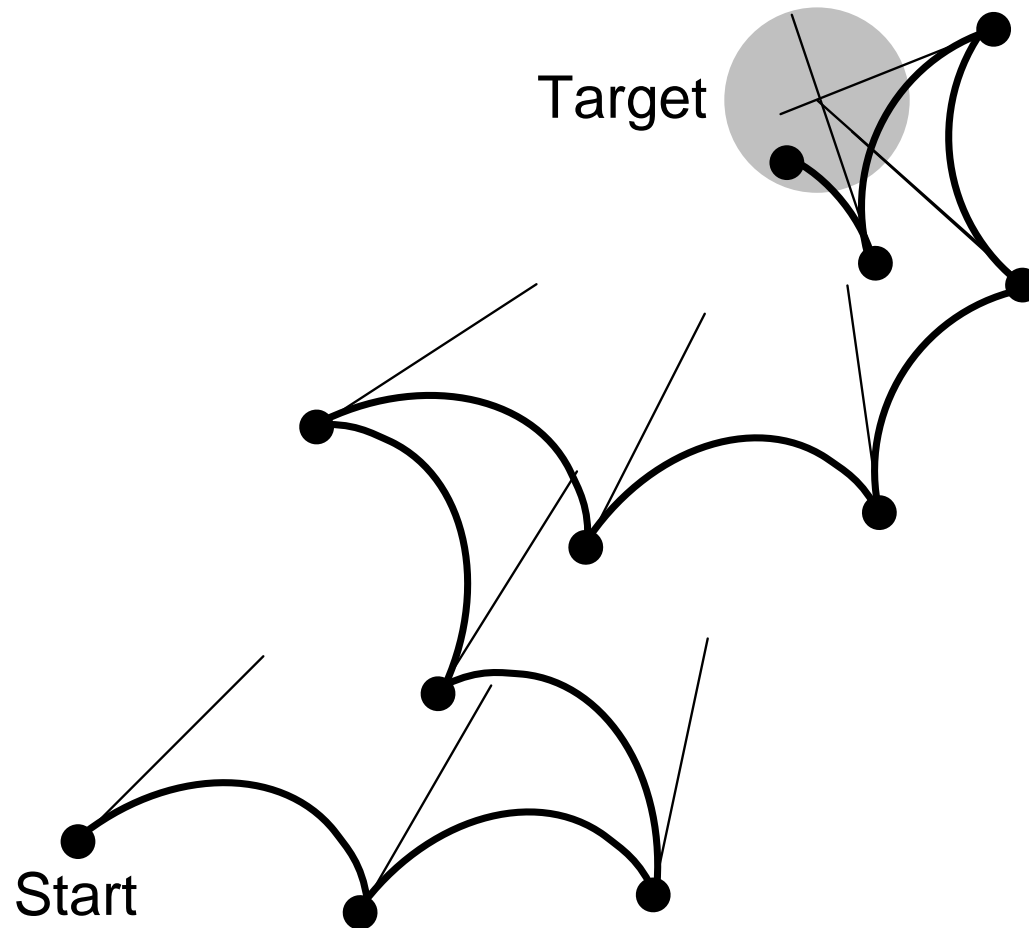
reasoning



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

# Feedback

stepsize: 3 months  
elapsed time: 25 months

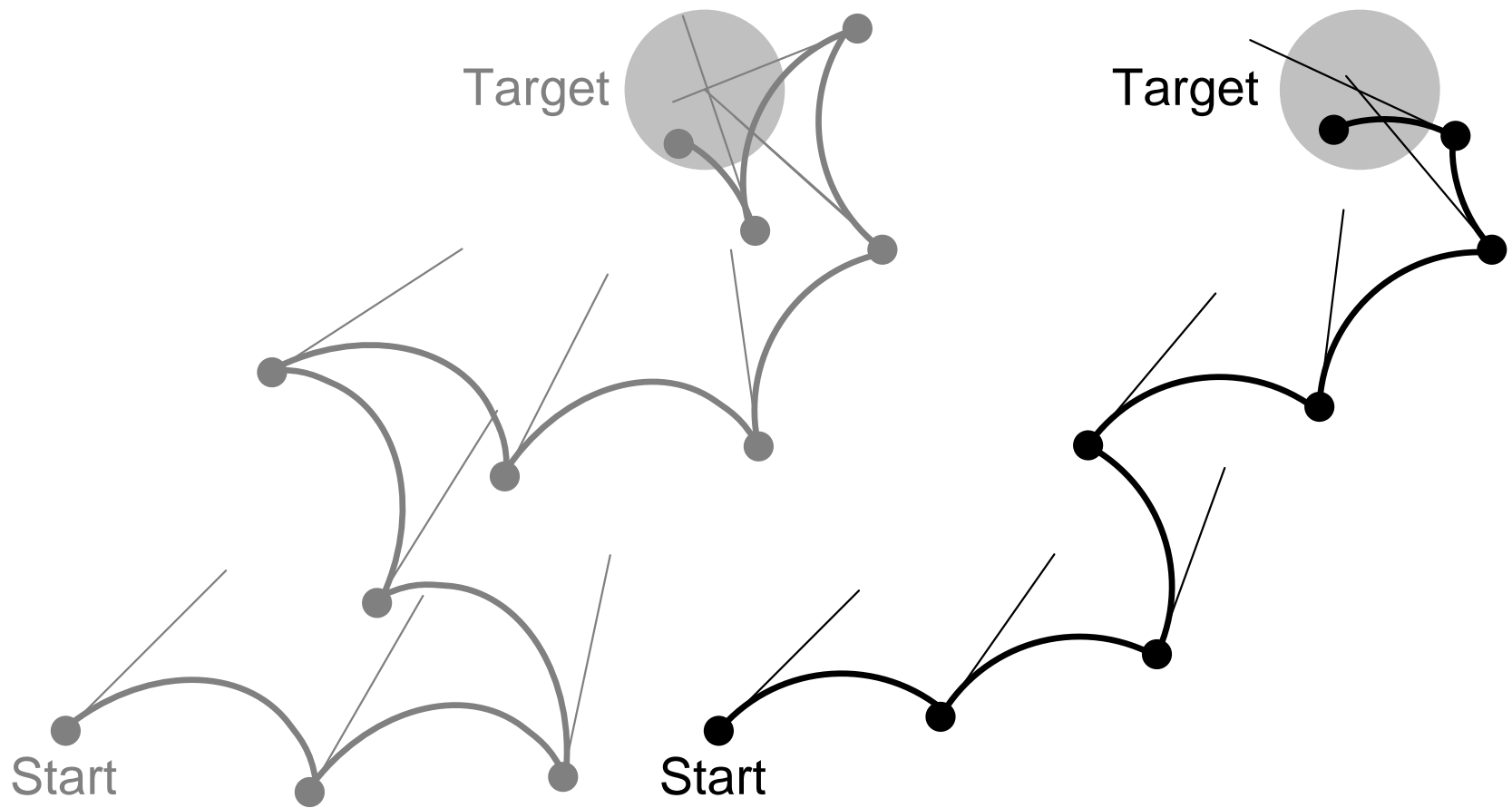


# Feedback (2)

stepsize:  
elapsed time

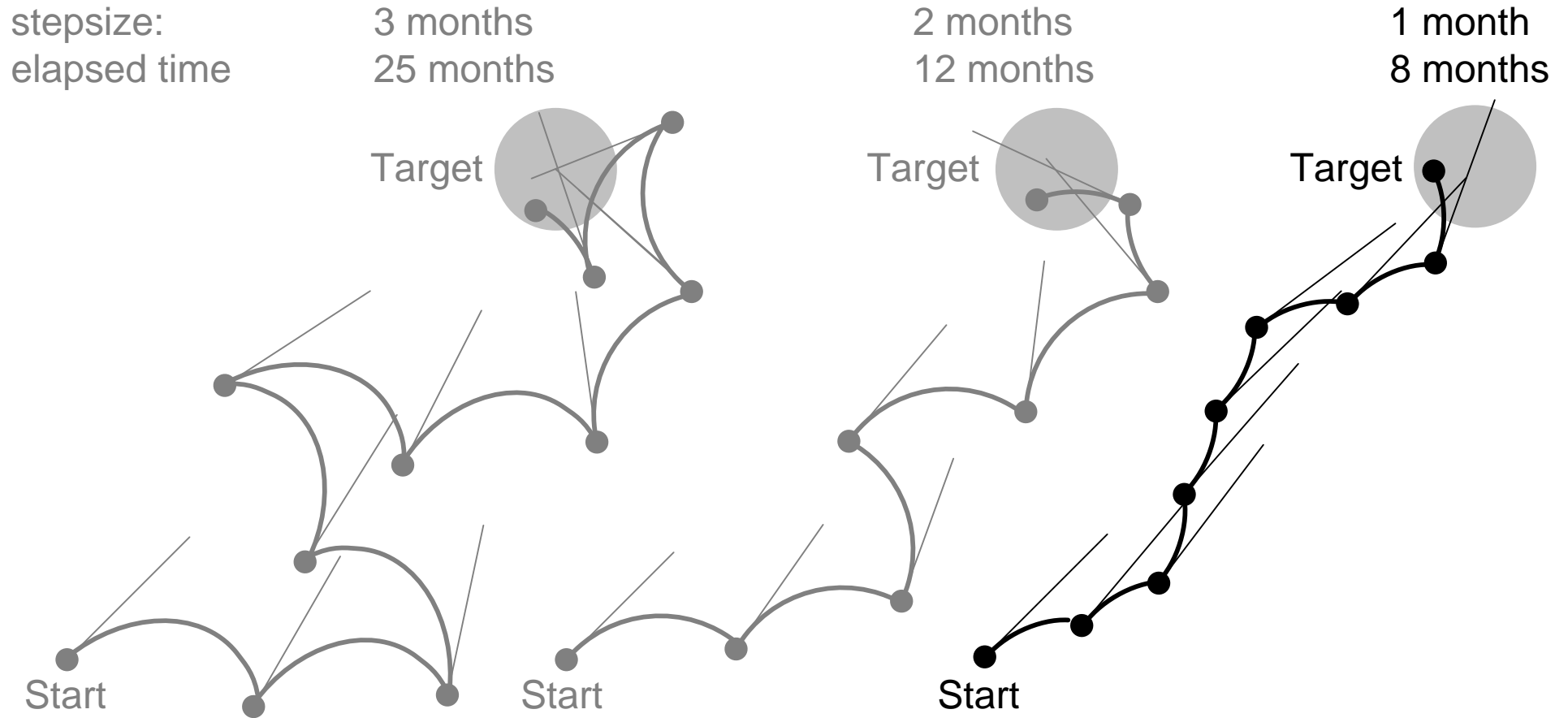
3 months  
25 months

2 months  
12 months



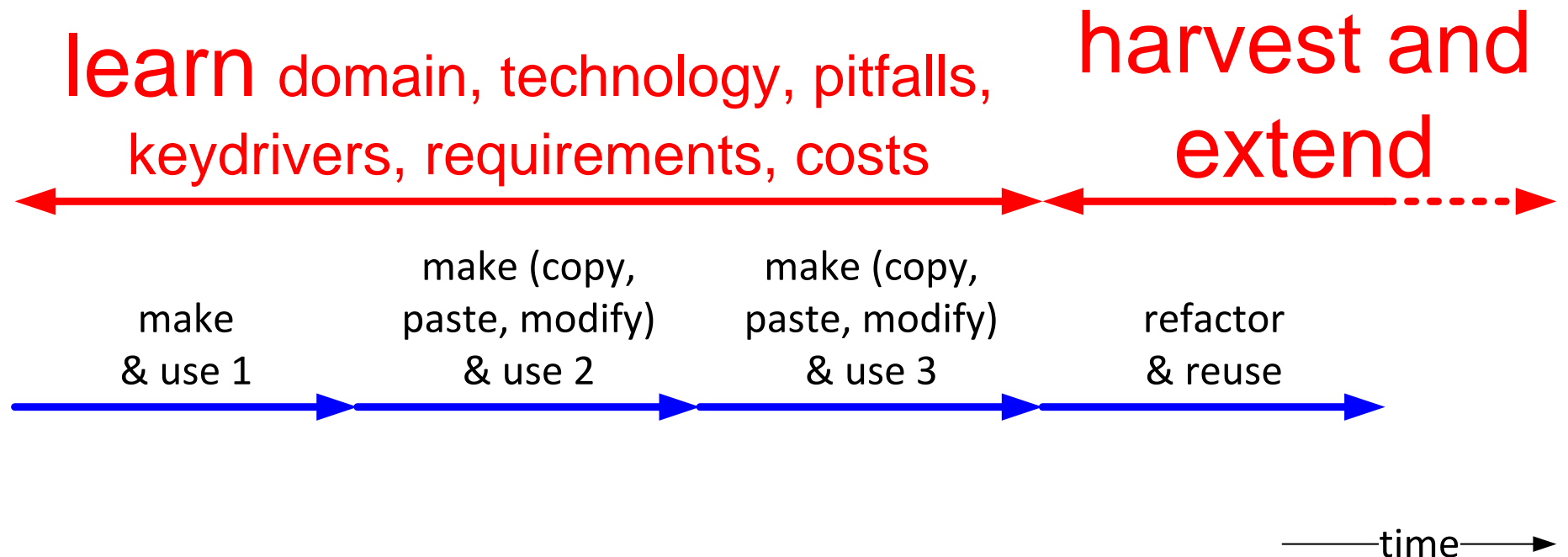


# Feedback (3)



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

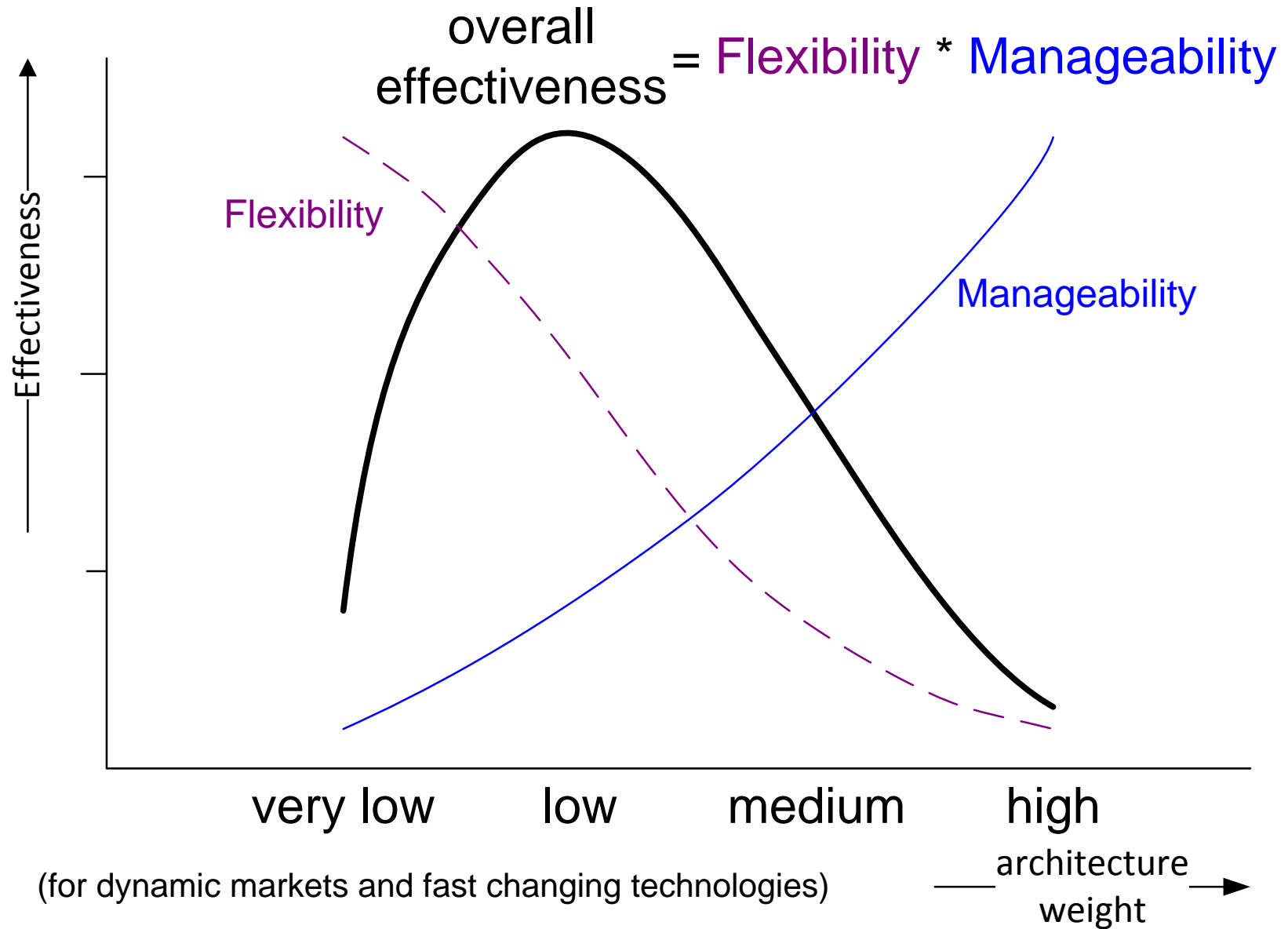
small hard real-time or extremely performance critical

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