A PHILOSOPHY OF SOFTWARE DESIGN

Authors John Ousterhout

DESIGN COMPLEXITY

Complexity is anything related to the structure of a software that makes it hard to understand or modify. It can be defined as:

$$C = \sum_p c_p \cdot t_p$$

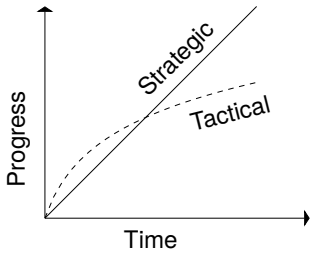
Where p is the number of components, c_p the complexity of each component, et t_p the fraction time accorded to it.

NATURE OF COMPLEXITY

Depedencies and **obscurity** causes complexity. It can be evaluated by questionning the 3 points:

- **Change amplification**: amount of code affected by each design decision;
- **Cognitive load**: how much a developer need to know to complete a task;
- **Unknown unknows**: obviousness of which piece of code must be modified to complete a task.

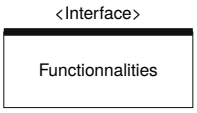
STRATEGIC VS. TACTICAL



Tactical programming make things work, quickly. Strategic programming invest time on design.

At the beginning, a tactical approach to programming will make progress more quickly than a strategic approach. However, complexity accumulates more rapidly under the tactical approach, wich reduces productivity.

DEEP VS. SHALLOW



(a) Shallow module

(b) Deep module

Deep modules have a simple interface and powerful functionnalities, **shallow modules** have complex interface, not much functionnality and hide does not hide complexity.

TOGETHER OR APART ?

Modules, classes or functions should be together if:

- information is shared,
- it simplifies interface,
- it avoids ressources duplication.

Keep a separation between specialized and general entities.

COMMENTS

Comments should capture information that was in the mind of the designer and could not be represented in the code.

NAMING

“The greater the distance between a name’s declaration and its uses, the longer the name should be.”

Andrew Gerrand