디자인 전략(Design Strategies)

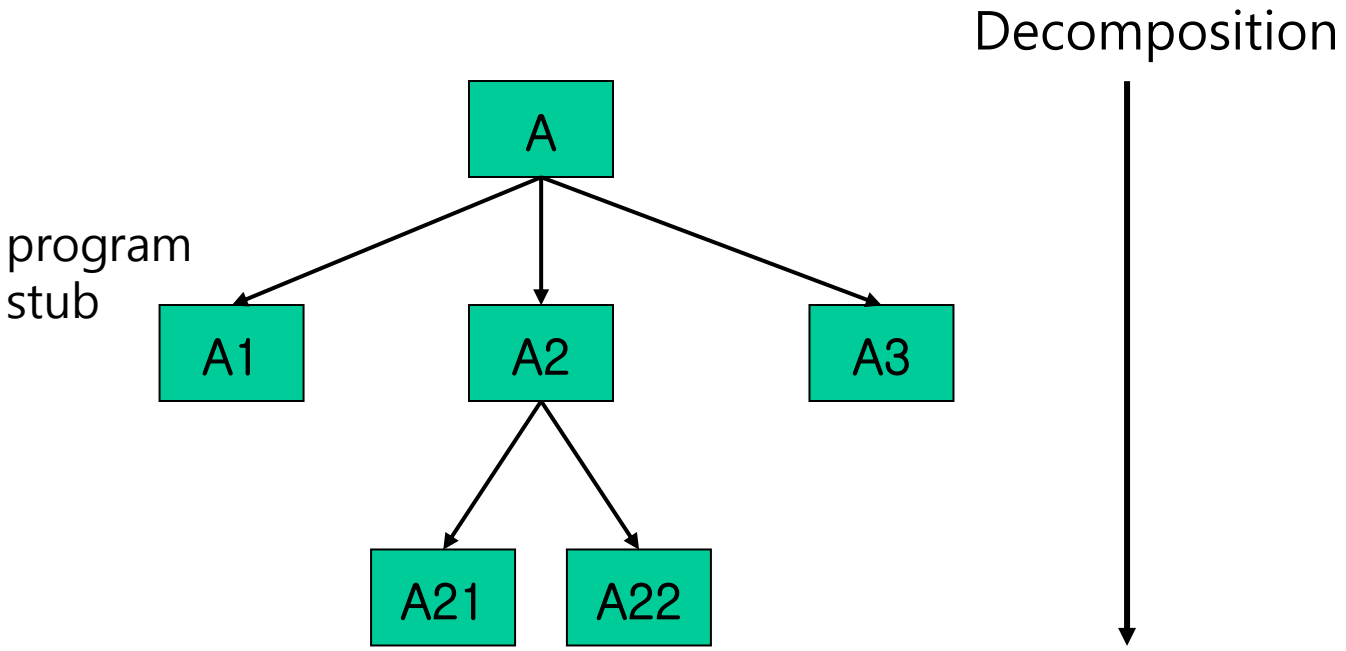
프로그램 설계 전략

- 크게 2가지 설계 전략으로 나뉘어 질 수 있다
  - **Top-down (하향식 설계전략)**
  - **Bottom-up (상향식 설계 전략)**

- 설계 전략을 세우는 목적은
  - 프로그래밍 하는 과정을 체계화 하기 위해서
  - 문제를 해결하는 프레임워크(기반이 되는 틀) 제공
  - 프로그램을 모듈화 하기 위해서

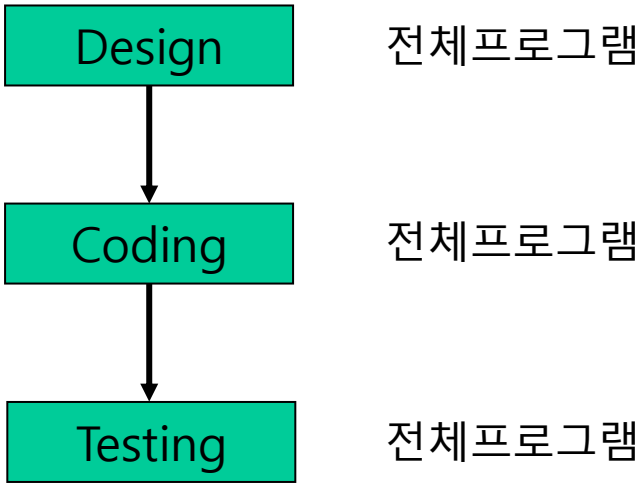
Top-Down Strategy(하향식 설계전략)

- **단계적 세분화**(stepwise refinement) 개념에 기반을 두고 프로그램을 설계, 코딩, 그리고 테스트하는 순차적 방법
- **계층적 구조를 가진 모듈화 된 프로그램**을 설계할 수 있다
- 점진적 혹은 단계적 접근방식이다
-
- 문제 정의와 분석(조사)로 시작하여 구현 계획을 제시하는데, 결과 프로그램의 일반적인 구조를 도출한다



Program stub :
아직 개발되지 않은 함수 혹은 모듈에 대해 테스팅을 위해서 단지 return statement 만 넣어서 만든 아직 내용이 구현되지 않은 function 모듈

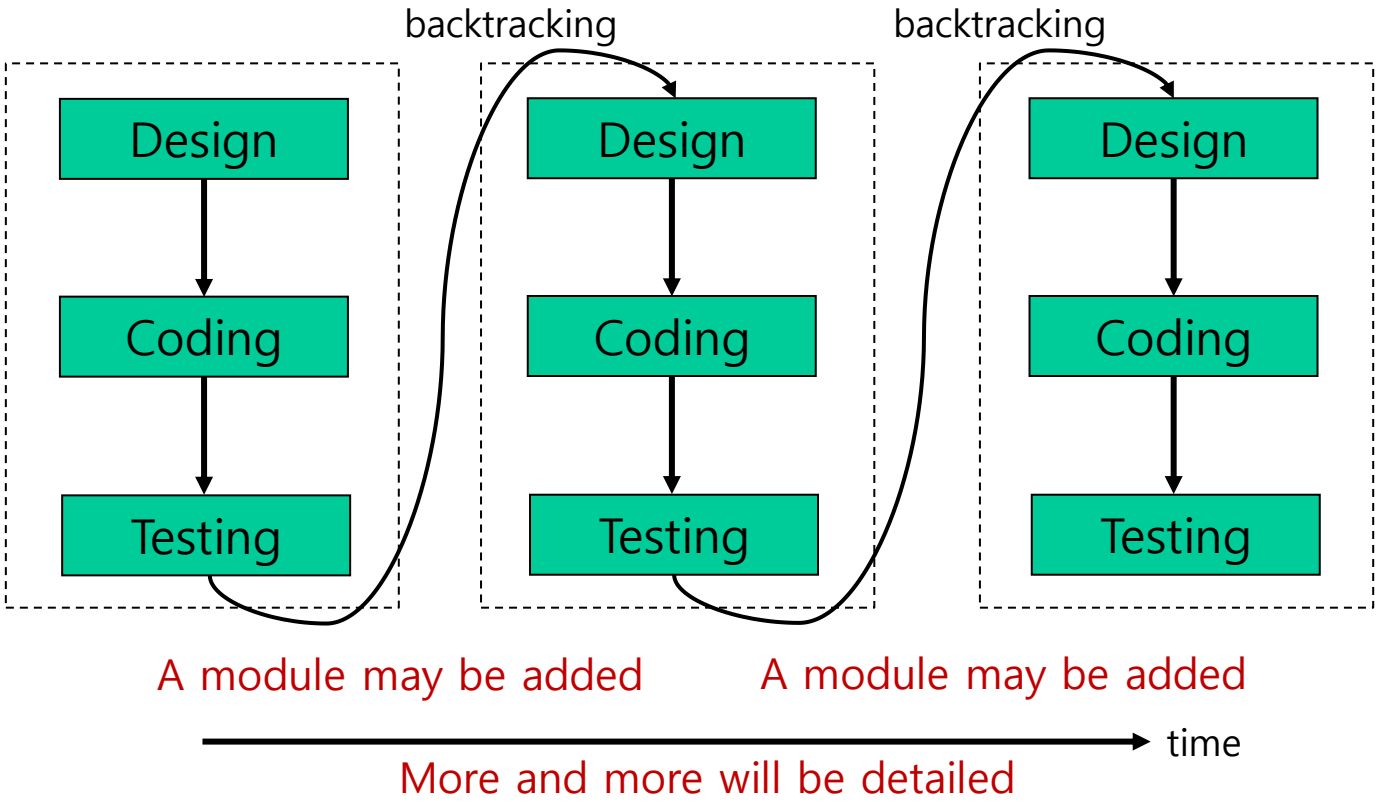
all-in-one approach – 여러분이 하는 코딩방식



All-in-one approach is generally not successful

1) delay for finding serious problem (심각한 문제의 발견 시점이 늦다)
2) 한꺼번에 모든 인적 물적 자원과 시간을 필요로 한다

- Top-down approach by **stepwise refinement**

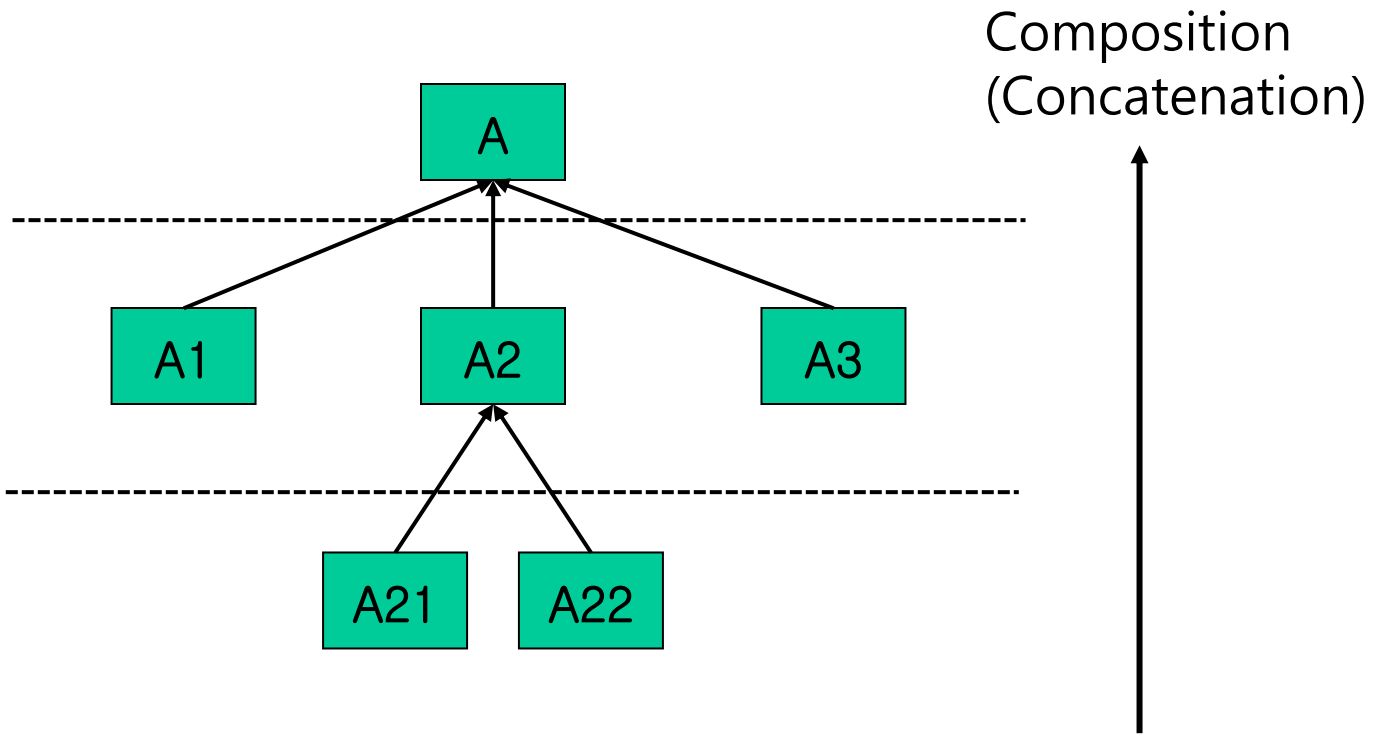


단점

- 공통 함수(common functions)들이 확인 되지 않을 수 있다
- 개발 과정 기간이 매우 길다
- 사용자 레벨에서 아직 완료되지 않은 상태에서 demonstration을 하기 위해서는, 아직 개발되지 않은 모듈들에 대해 dummy routine(***program stub***) 들을 사용해야 하는 경우가 종종 있다
- 완전한 프로그램이 개발될 때 까지 성능에 대한 평가가 지연된다
- 그러므로 개발 사양 등이 잘 정의된(well-defined) 환경에서 사용하기 적합하다
- 대부분의 응용 프로그램이 이에 해당한다고 볼 수 있다

Bottom-up Strategy(상향식 설계전략)

- begins with an outline of the proposed program
  - outline includes the general functional and data components of the pro gram
  - 설계자는 제일 먼저, 문제 해결을 위해 기반을 제공해 줄 primitive object 들, action들, 그리고 그들 사이의 relationship을 찾아서 확인하는 노력 이 필요하다
  - 그러므로, 상향식 설계 전략의 성패는 그 시스템을 구현하는데 충분 한 proper primitive idea set를 찾아 내는데 있다고 볼 수 있다
  - 그리하여 개발하고자 하는 프로그램의 초기 버전(initial solution)이 일 단 제안된다. 왜냐하면 프로그램의 가장 낮은 단계의 콤포넌트들이 개발 되기 때문이다
- levels of abstraction 개념에 기반을 두고 있다



Program module A2 is constructed by combining low-level components A21 and A22

- **특징**
  - 시스템의 개발 중간중간에 subsystem에 대한 성능의 평가가 가능하다
  - 그러므로, 아직 사양이 완벽하게 정의되지 않은(ill-defined) 개발 환경에서 적용하기 적합하다
- **단점**
  - undefined 된 데이터가 발생될 수 있다
  - 모듈간의 인터페이스에 대한 테스트가 약할 수 있다

Program Modularization(프로그램 모듈화)

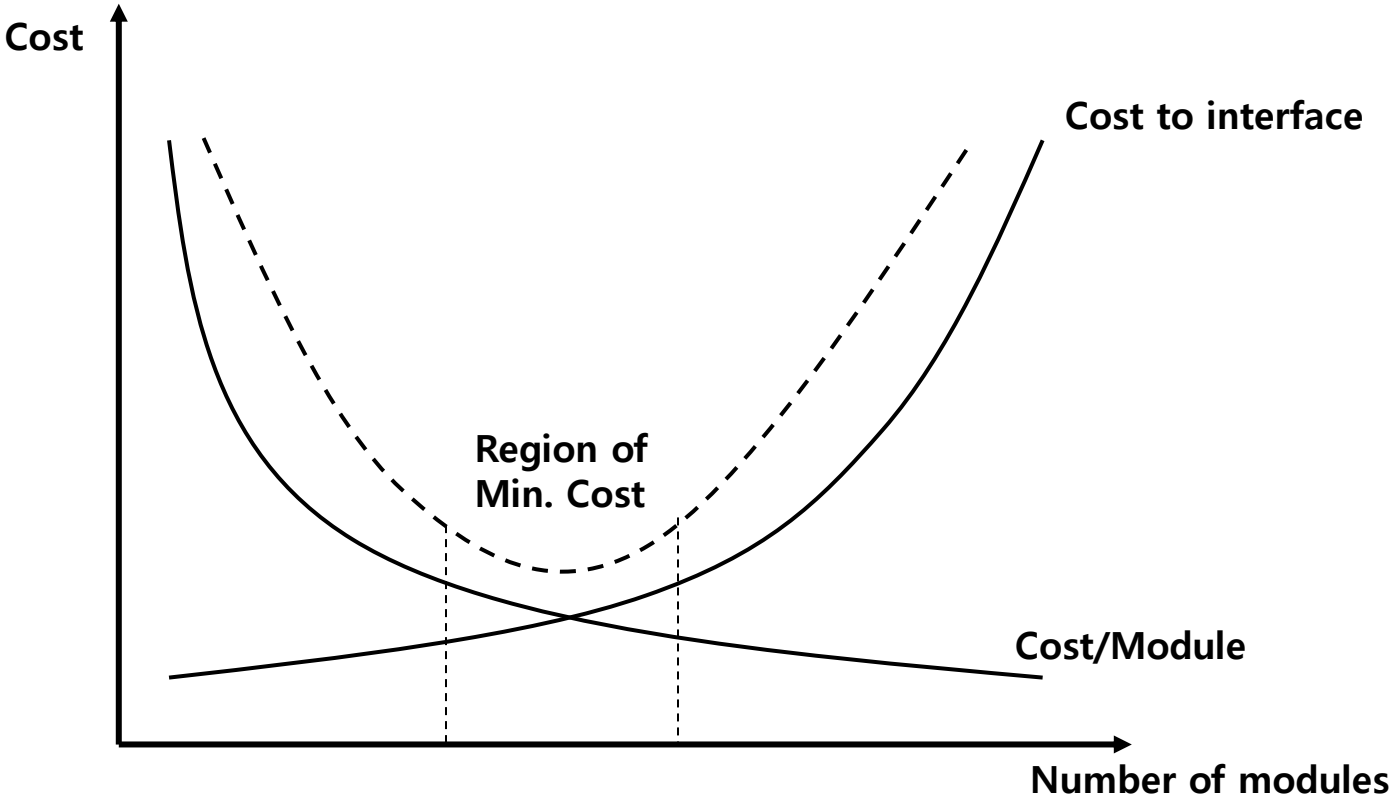
- **Why modularization ?**
- C(x)를 problem x의 복잡도(complexity)라고 하고 E(x)를 problem x를 해결하는데 필요한 cost라고 하자. 그러면,

- For two problems P1, and P2

  If C(P1) > C(P2), then E(P1) > E(P2)

  그런데, 일반적으로

  C(P1 + P2) > C(P1) + C(P2), 그러므로 E(P1 + P2) > E(P1) + E(P2)


Cost
Cost to interface
Region of
Min. Cost
Cost/Module
Number of modules

효과적인 모듈화를 위한 고려 사항

- Modular decomposability
- Modular composability(reusability)
- Modular understanability
- Modular protection(integrity)
- Modular continuity

- **효과적인 모듈의 설계**
  - functional independence (기능적 독립성)을 가능한 높인다
  - 모듈의 응집도(cohesion)를 높인다
  - 모듈간의 간섭도(coupling)를 낮춘다
  - 적정한 fan-out 모듈의 수
    - The number of modules that are directly controlled by the other module
  - 적정한 fan-in 모듈의 수
    - how many modules directly control a given module
  - visibility의 향상
    - the set of program components that may be invoked or used as data by a given component, even when this is accomplished indirectly --- program depth, width
  - connectivity의 향상
    - the set of components that are directly invoked or used as data by a given component – program width, fan-out
  - Component-oriented programming (COP)
    - OOP의 reusability(재사용성)를 효과적으로 또 체계적으로 변형시킨 개념

- fan-in and fan-out