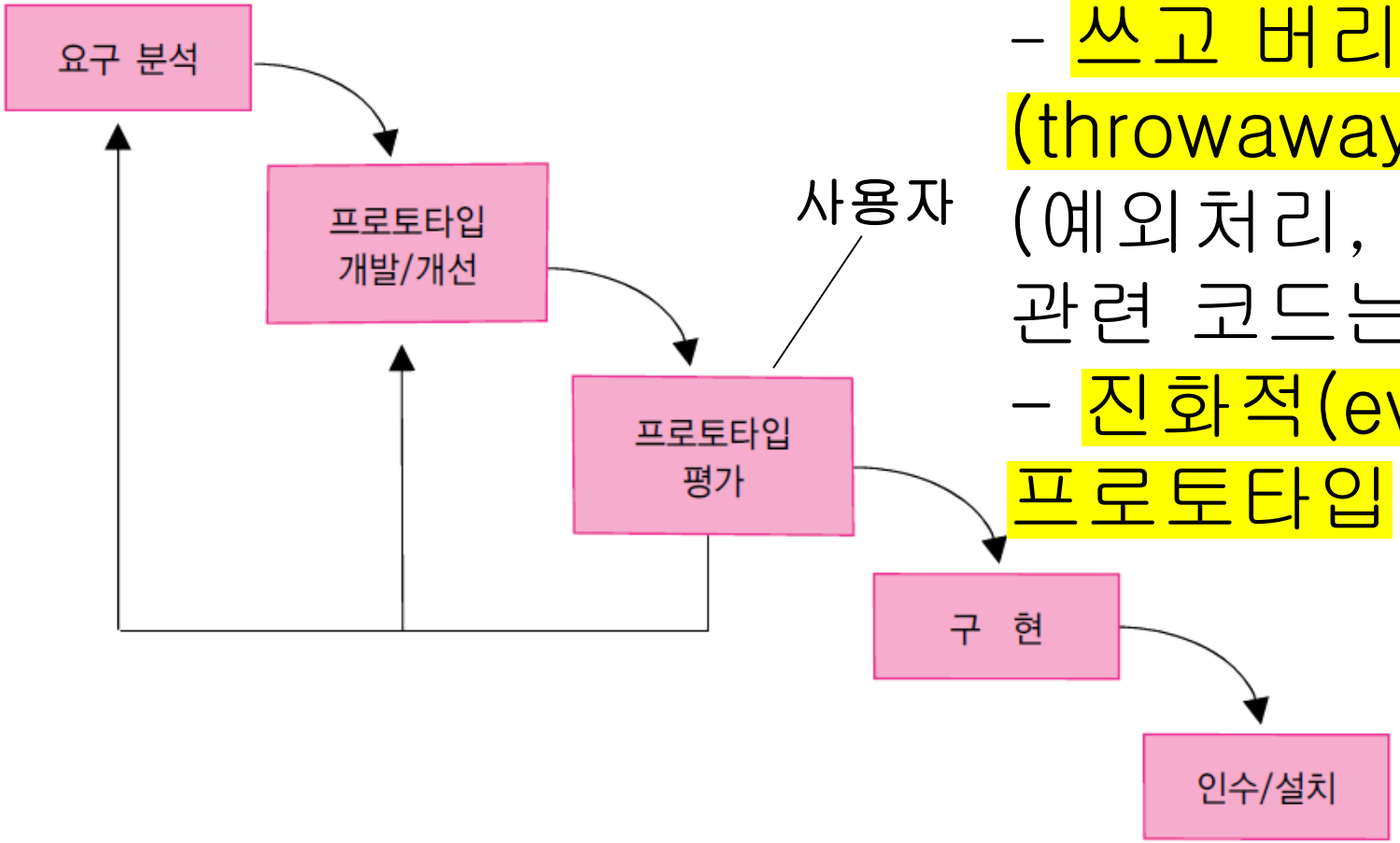
(2) 프로토타이핑 모델

- **Rapid Prototyping Model**

동기: 사용자가 한꺼번에 완전한 요구를 낼 수가 없기에.



- 쓰고 버리는 (throwaway) 프로토타입. (예외처리, 표준준수 등 관련 코드는 제외시킴)
- 진화적(evolutionary) 프로토타입

프로토타이핑 모델

- **프로토타입(시범 시스템)의 적용**
  - 사용자의 요구를 더 정확히 추출
  - 알고리즘의 타당성, 운영체제와의 조화, 인터페이스의 시험 제작

- **프로토타이핑 도구**
  - 화면 생성기
  - 비주얼 프로그래밍 (ex: scratch), 4세대 언어 (report generation, data management 용도 등) 등

프로토타이핑 모델

- **공동의 참조 모델**
  - 사용자와 개발자의 의사소통을 도와주는 좋은 매개체

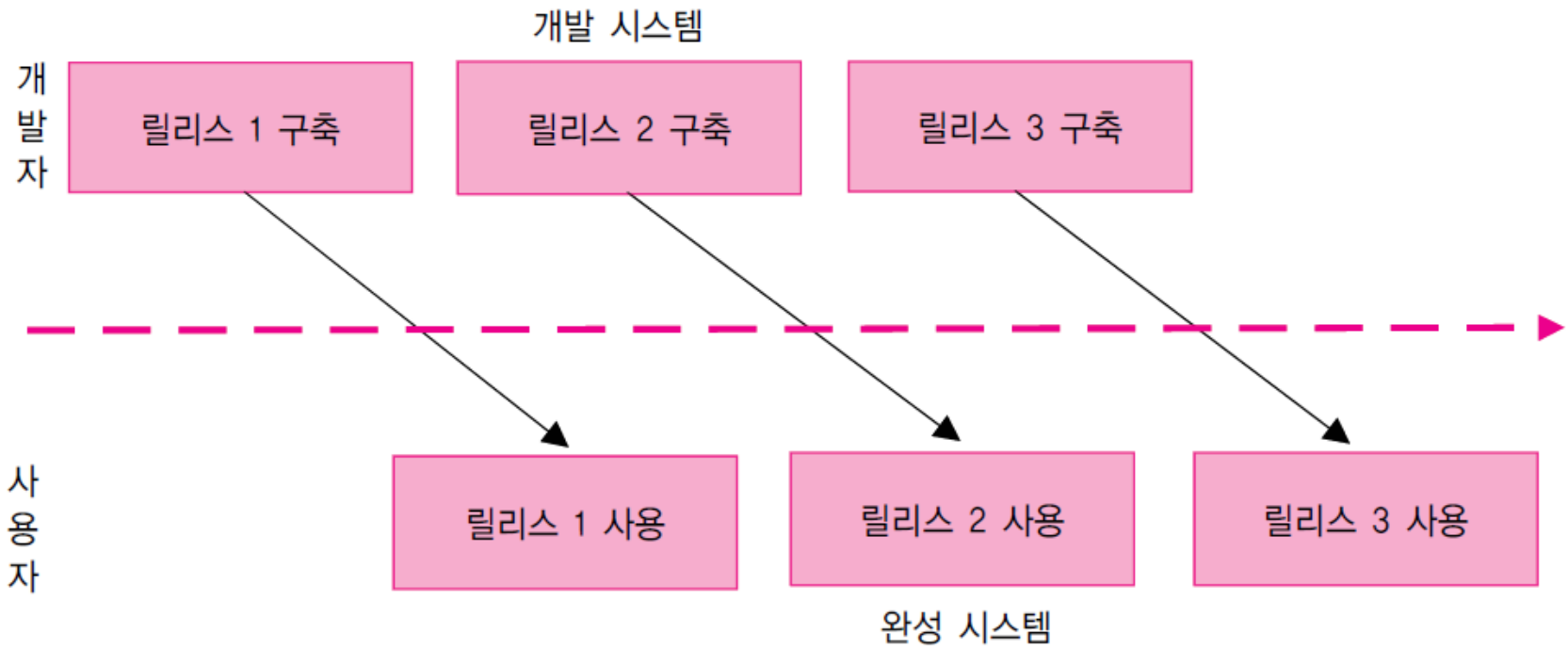
- **프로토타입의 목적**
  - 단순한 요구 추출 – 만들고 버림
  - 제작 가능성 타진 - 개발 단계에서 (사용자가 사용하게 함으로써)유지보수가 이루어짐

프로토타이핑 모델의 장단점

- **장점**
  - 사용자의 의견 반영이 잘 됨
  - 사용자가 더 관심을 가지고 참여할 수 있고 개발자는 요구를 더 정확히 도출할 수 있음
- **단점**
  - (발주자의) 오해, 기대심리 유발
  - (프로토타입핑 과정의)관리가 어려움(중간 산출물 정의가 난해)
- **적용**
  - 개발 착수 시점에 요구가 불투명할 때
  - 실험적으로 실현 가능성을 타진해 보고 싶을 때
  - 혁신적인 기술을 사용해 보고 싶을 때

(3) 진화적 모델

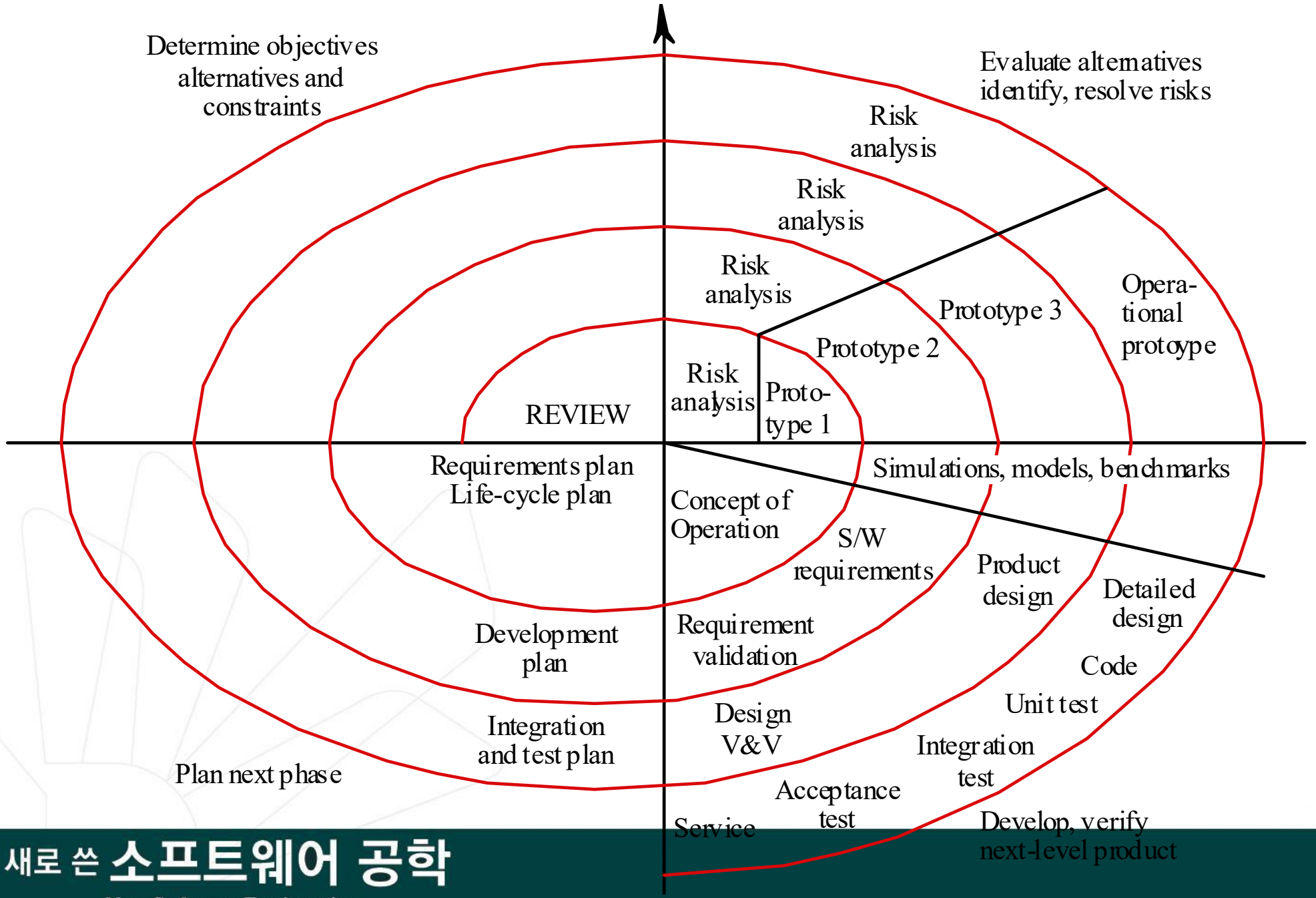
- **개발 사이클이 짧은 환경**
  - 빠른 시간 안에 시장에 출시하여야 이윤에 직결
  - 개발 시간을 줄이는 법 – 시스템을 나누어 릴리스



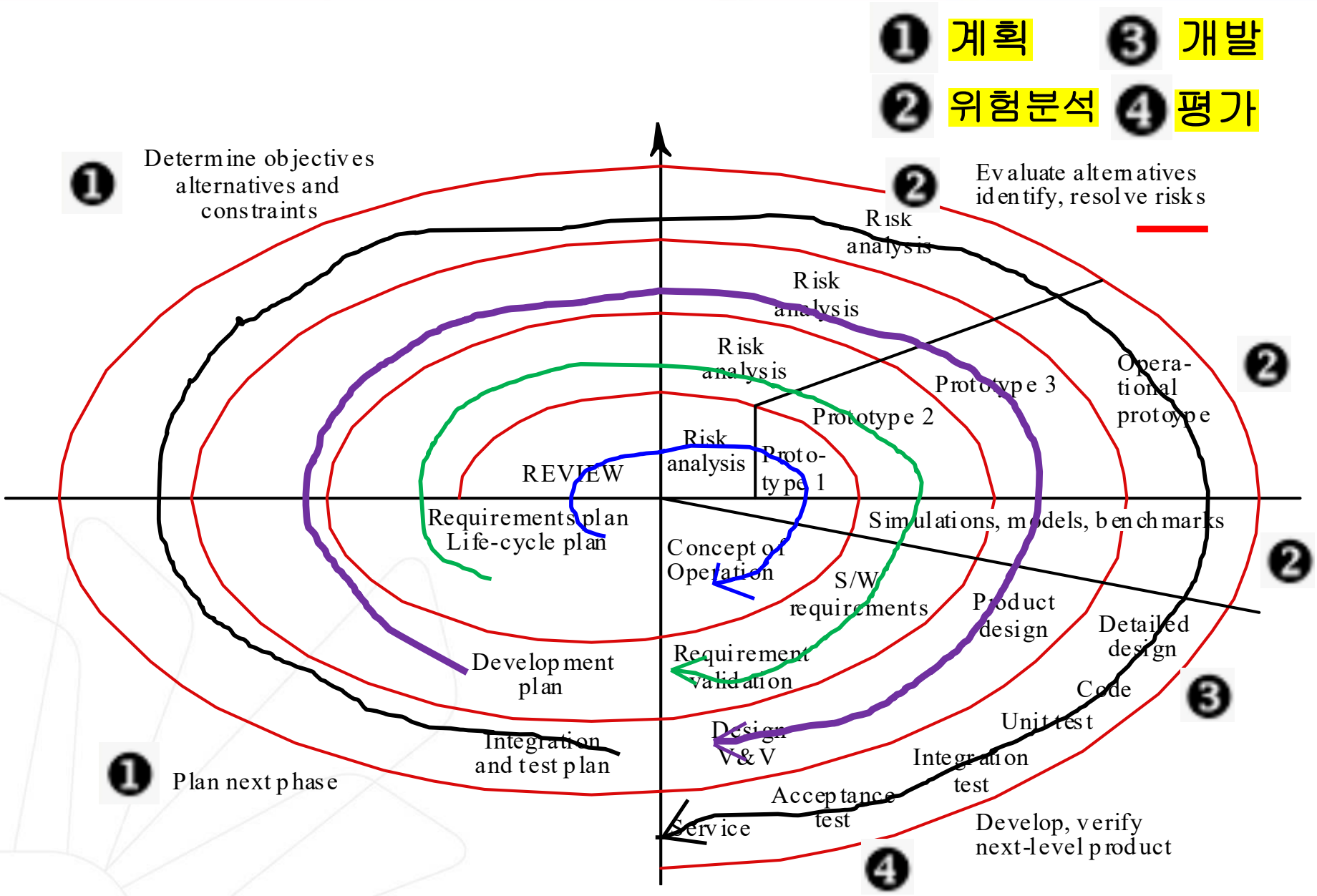
진화적 모델

- **릴리스 구성 방법**
  - 점증적 (stepwise increments) 방법 – 기능별로 릴리스
  - ~~반복적~~ 단계적 상세화(stepwise refinement) 방법 - 릴리스 할 때마다 기능의 완성도를 높임
- **단계적 개발**
  - 기능이 부족하더라도 초기에 사용 교육 가능
  - 처음 시장에 내놓는 소프트웨어는 시장을 빨리 형성시킬 수 있음
  - 자주 릴리스 하면 가동 중인 시스템에서 일어나는 예상하지 못했던 문제를 신속 꾸준히 고쳐나갈 수 있음.
  - 개발 팀이 릴리스마다 다른 전문 영역에 초점 둘 수 있음.

(4) 나선형(spiral) 모델



나선형(spiral) 모델



나선형(spiral) 모델

- **소프트웨어의 기능을 나누어 점증적 개발**
  - 실패의 위험을 줄임
  - 테스트 용이
  - 피드백
- **여러 번의 점증적인 릴리스(incremental releases)**
- **Boehm이 제안**

진화적 모델의 성격을 가짐

참고: incremental builds of the product or continuous refinement through repetitions(주기)

나선형(spiral) 모델

- 각 주기의 진화 단계들
  - 계획 수립(planning): 목표, 기능 선택, 제약 조건의 결정
  - 위험 분석(risk analysis): 기능 선택의 우선순위, 위험요소의 분석
  - 개발(engineering): 선택된 기능의 개발
  - 평가(evaluation): 개발 결과의 평가

나선형(spiral) 모델의 장단점

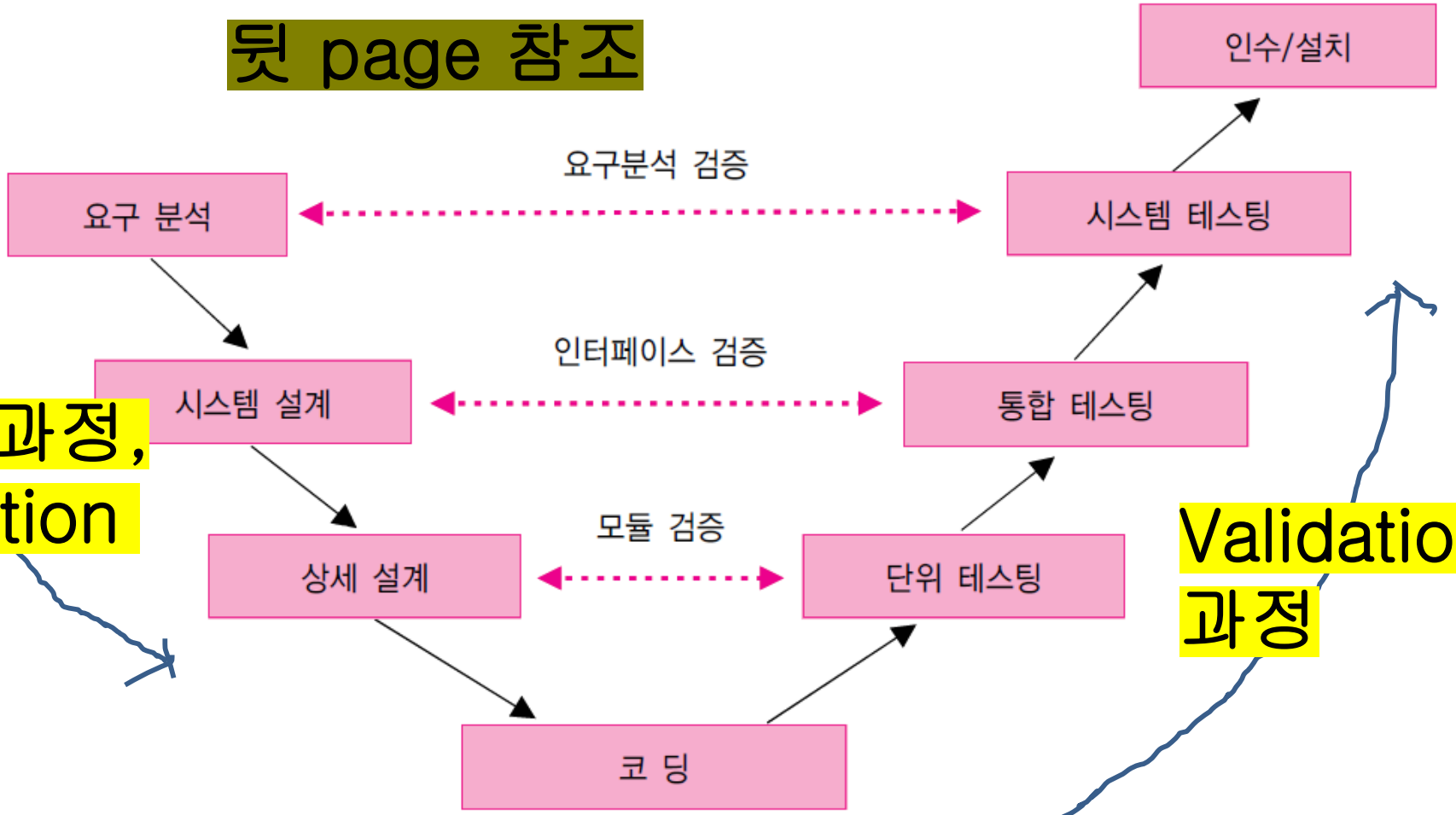
- **장점**
  - 대규모 시스템 개발에 적합 - risk reduction mechanism
  - 반복적인 개발 및 테스트 - 강인성 향상
  - 한 사이클에 추가 못한 기능은 다음 단계에 추가 가능
- **단점**
  - 관리가 중요
  - 위험 분석이 중요 ➔ 만약 위험분석이 잘못되면 ?
  - 새로운 모형

나선형(spiral) 모델의 장단점

- **적용**
  - 재정적 또는 기술적으로 위험 부담이 큰 경우
  - 요구 사항이나 아키텍처 이해에 어려운 경우

(5) V 모델 (V&V 모델)

뒷 page 참조

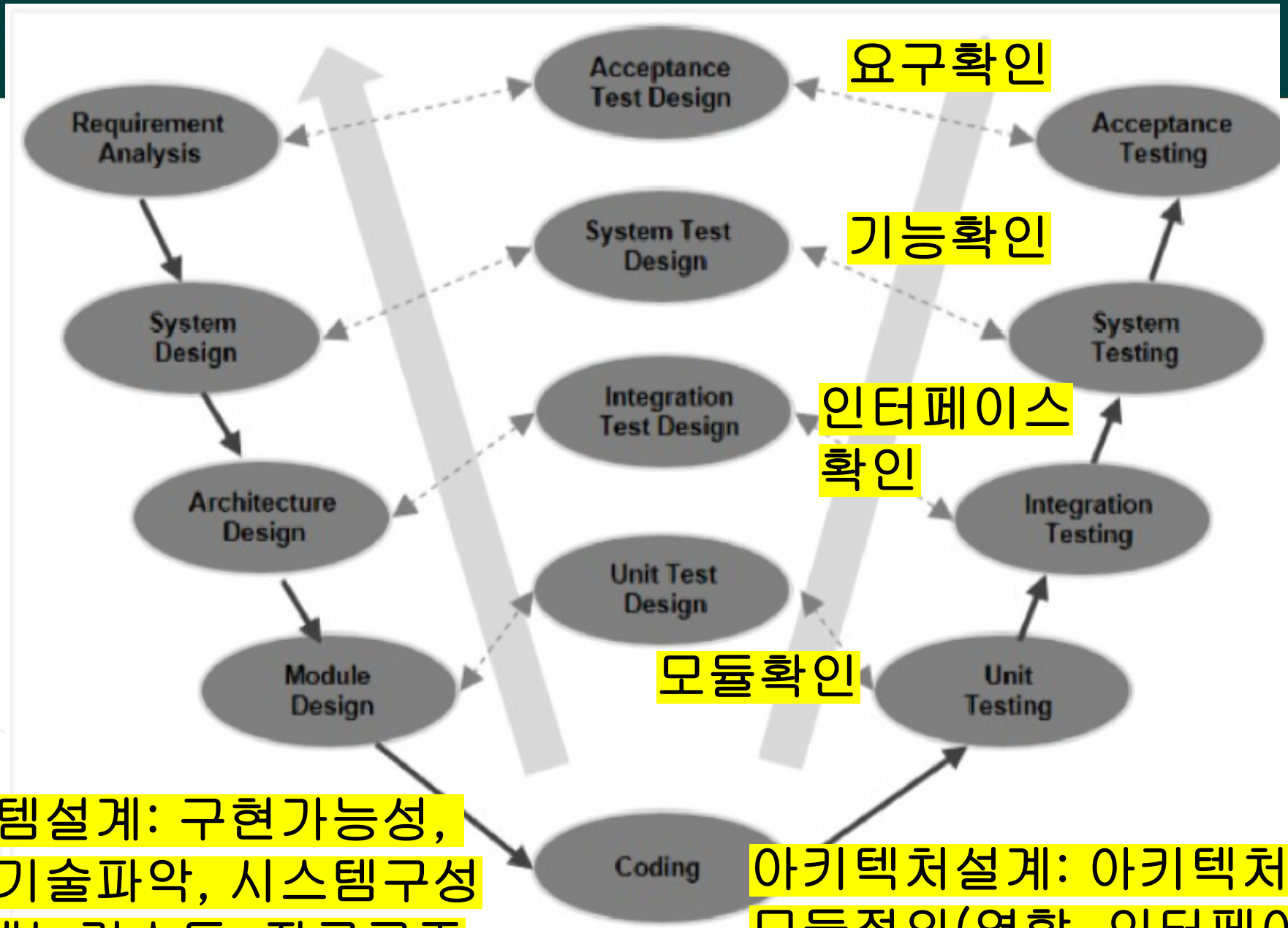


상세화 과정, Verification 과정

Validation 과정

Verification: 요구에 맞추어 잘 진행되었는 지(정적인 검증). 개발자 관점

Validation: 요구에 맞추어 잘 동작하는지 (코드 실행 중 에라는 없는 지, 실행 결과가 맞는 지). 사용자 관점.



요구확인

기능확인

인터페이스 확인

모듈확인

시스템설계: 구현가능성, 필요기술파악, 시스템구성도, 메뉴리스트, 자료구조(ER 다이어그램 등).

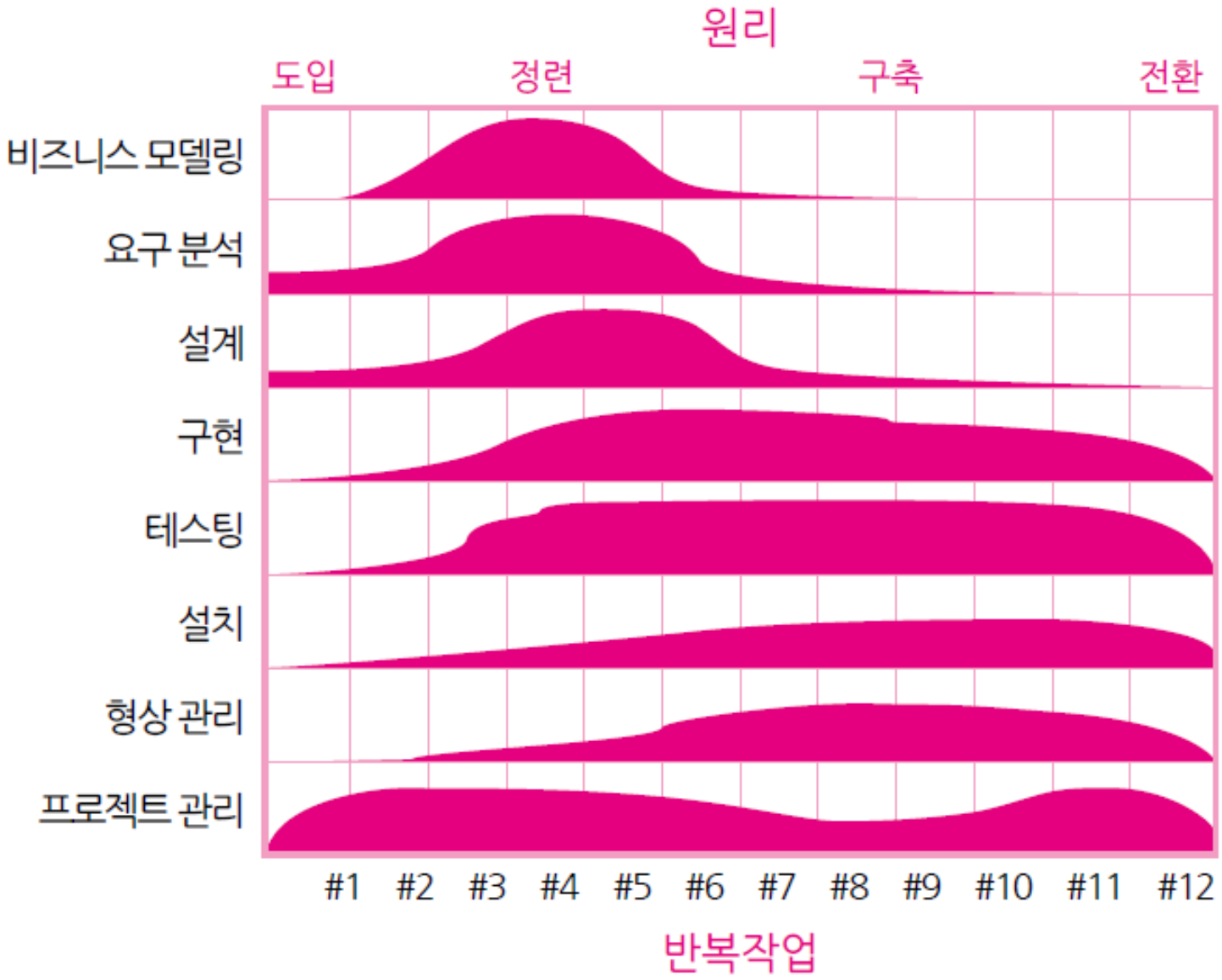
아키텍처설계: 아키텍처 정의, 모듈정의(역할, 인터페이스, 의존성), 타시스템과 통신, 데이터 테이블 목록)

V 모델

- 폭포수 모형의 변형
  - 감추어진 반복과 재 작업을 드러냄 ← 검증 과 테스트 계획 작성을 통해
  - 작업과 결과의 검증에 초점
- 장점
  - 오류를 줄일 수 있음
- 단점
  - 반복이 없어 변경을 다루기가 쉽지 않음
- 적용: 신뢰성이 높이 요구되는 분야 → 의료, 원자력, 우주

(6) Unified 프로세스

반복적이고 점증적



도입: 계획 수립, 간단한 사용사례, 아키텍처

정련: 대부분 사용사례, 아키텍쳐

구축: 나머지 사용사례, 통합

전환: 베타 테스팅, 사용자 인수

- **사용 사례(Use Case)**
  - '누구'(Actor)가 시스템을 '어떤 특정 업무'(a specific task)를 위해 사용하는 사용 '시나리오' (uses scenario)를 따라 사용하는 케이스 .]

  - EX) An airline's online booking system
- *A customer browsing flight schedules and prices*
- *A customer selecting a flight date and time*
- *A customer adding on lounge access and free checked bags*
- *A customer paying with a personal credit card*
- *A customer paying with loyalty miles*

Unified 프로세스

- 사용 사례 중심의 프로세스
- 시스템 개발 초기에 아키텍처와 전체적인 구조를 확정
- 아키텍처 중심
- 반복적이고 점증적