Managing Software Project And Advanced Topic

Woo Young Moon, Ph.D.
Professional Engineer Information Management
PMP, CFPS

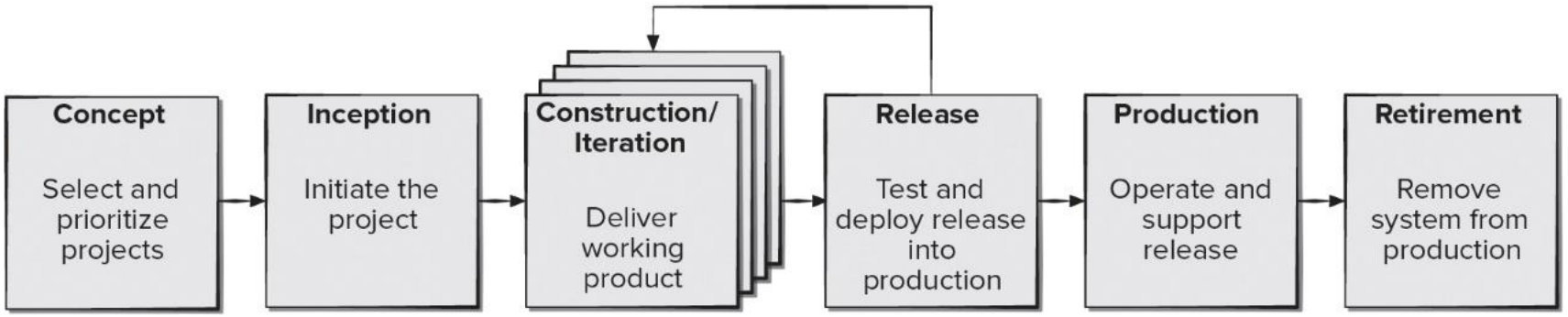
Mobile 010-8709-1714
Wooyoungmoon@soongsil.ac.kr

A Strategy for Software Support

A Strategy for Software Support

- Change Occurs
  - when errors are corrected,
  - when the software is adapted to a new environment,
  - when customers request new features or functions,
  - when the application is reengineered to provide benefit in a modern context
- Software Support
  - 버그 수정, 환경 변화에 따른 소프트웨어 적응, 이해 관계자 요구에 따른 소프트웨어 향상, 더 나은 기능과 성능을 달성하기 위한 소프트웨어 리엔지니어링 등 일련의 활동을 포함하며 이 과정에서 품질을 보장하고 변경 사항을 관리해야 함
    - 소프트웨어 지원: 소프트웨어가 사용되는 전체 기간 동안 제공되는 활동
    - 버그 수정: 소프트웨어의 오류 또는 결함을 식별하고 수정하는 작업
    - 환경 변화: 운영 체제, 하드웨어, 다른 소프트웨어 등 소프트웨어가 속한 환경의 변경
    - 이해 관계자 요구: 소프트웨어 제작, 사용, 유지보수에 이익이 있는 사람이나 조직의 요구 사항
    - 소프트웨어 리엔지니어링: 기존 소프트웨어 코드를 구조적 으로 변경하여 기능이나 성능을 향상시키는 작업

Prototype Evolution Process Model



- 프로토타입 진화 프로세스 모델.
  - 프로세스의 단계는 개념, 시작, 구성 또는 반복, 릴리스, 생산 및 폐기.
  - Concept: 제품이 선택, 우선순위가 지정.
  - Inception : 프로젝트를 시작.
  - Construction/Iteration : 작동하는 제품을 제공
  - Release : 테스트하고 프로덕션에 배포. 프로세스는 건설 단계로 돌아가거나 생산 단계로 넘어감
  - Production : 생산 단계가 운영되고 릴리스를 지원
  - Retirement : 마지막 단계에서 시스템은 생산에서 제거.

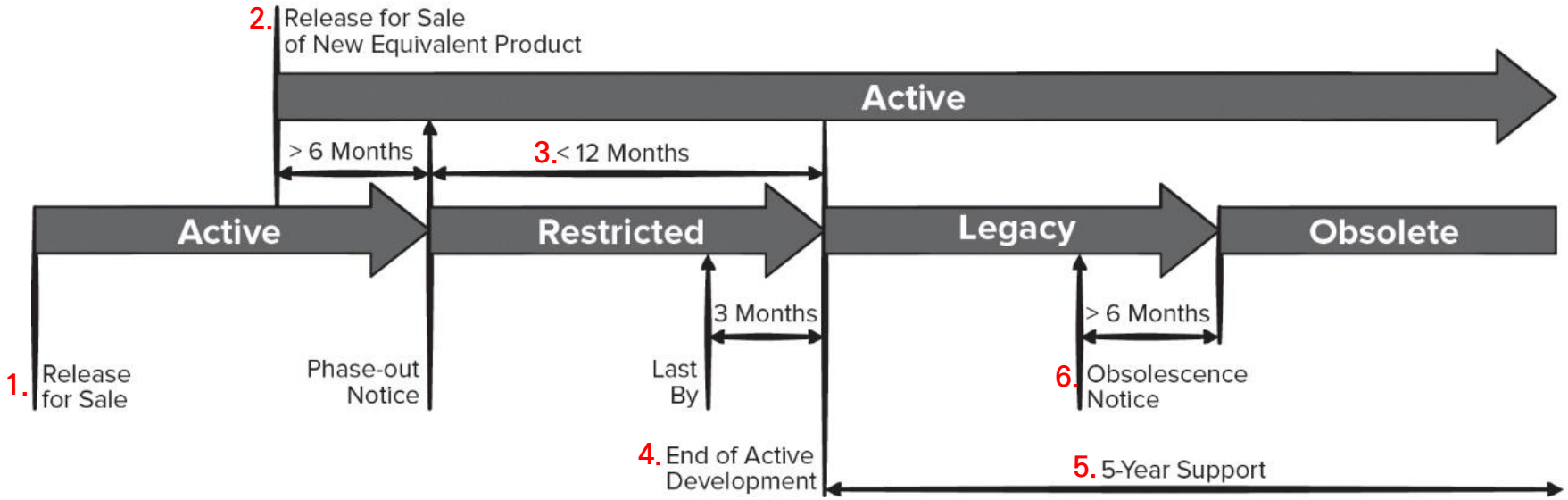
Lehman's Laws of Software Evolution

- 소프트웨어는 지속적으로 변화한다는 것을 전제로, 시스템 변화에 대해 일반적으로 적용되는 원칙제시.
  - S-타입(Static Type): 단순 계산 프로그램
  - P-타입(Practical Type): 요구되는 동작이 명확하여 변화가 거의 일어나지 않는 프로그램
  - E-타입(Embedded Type): 실제 일상(업무)에서 수행하는 활동을 위해 만들어진 프로그램
- Continuing Change 지속적 변경(1974년) : 소프트웨어는 계속 진화하며 요구사항에 의해 계속적으로 변경되어야 함, 소프트웨어는 자체적으로 갱신 불가하며 인간의 의지의 개입이 필요
- Increasing Complexity 복잡도 증가(1974년) : 변경이 가해질수록 구조는 복잡해짐, 복잡도는 이를 유지하거나 줄이고자 하는 특별한 작업을 하지 않는 한 계속 증가
- Self Regulation 자가 조절(1974년) : 프로그램별로 변경되는 사항은 고유한 패턴/추세가 있음, 복잡성을 단순화 시키려는 인간 의지의 개입
- Conservation of Organisational Stability 조직적 안정성(1978년) : 변화하는 시스템의 평균적인 효과성은 제품 생애 전반 동안 불변, 조직의 경영 사정, 전사적 안정성 추구 등에 따라 효과성은 크게 변하지 않음
- Conservation of Familiarity 익숙함의 보존(1978년) : 소프트웨어 각 버전의 변화는 일정함, 소프트웨어는 규칙적인 수행결과와 추이를 보여주기 때문에 계측 가능
- Continuing Growth 지속적 성장(1991년) : 소프트웨어의 기능성은 사용자 만족도를 유지하기 위해 증가되어야 함
- Declining Quality 품질 감소(1996년) 소프트웨어는 엄격하게 관리 및 운영되지 않거나, 환경 변화에 적응하지 않으면 품질 하락
- Feedback System 피드백 시스템(1996년) :시스템의 지속적인 변화 또는 진화를 유지하려면 성능을 모니터링 할 수단이 필요.
- https://itwiki.kr/w/%EB%A6%AC%EB%A8%BC_%EC%86%8C%ED%94%84%ED%8A%B8%EC%9B%A8%EC%96%B4_%EB%B3%80%ED%99%94_%EB%B2%95%EC%B9%99

Software Support

- **Software support can be considered an umbrella activity that includes:**
  - Change management.
  - Proactive risk management.
  - Process management.
  - Configuration management.
  - Quality assurance.
  - Release management.

Software Release and Retirement



An example of a timeline displays the phases of a software release and retirement.

1. An active software is released for sale.

2. 기존 소프트웨어의 단계적 중단 공지가 발표되기 최소 6개월 전에 A release for sale of a new equivalent product는 발표되어야 함.

3. 단계적 중단 통지 후 소프트웨어의 사용 기간은 12개월 미만임.

4. The last by to End of active development period is of 3 months.

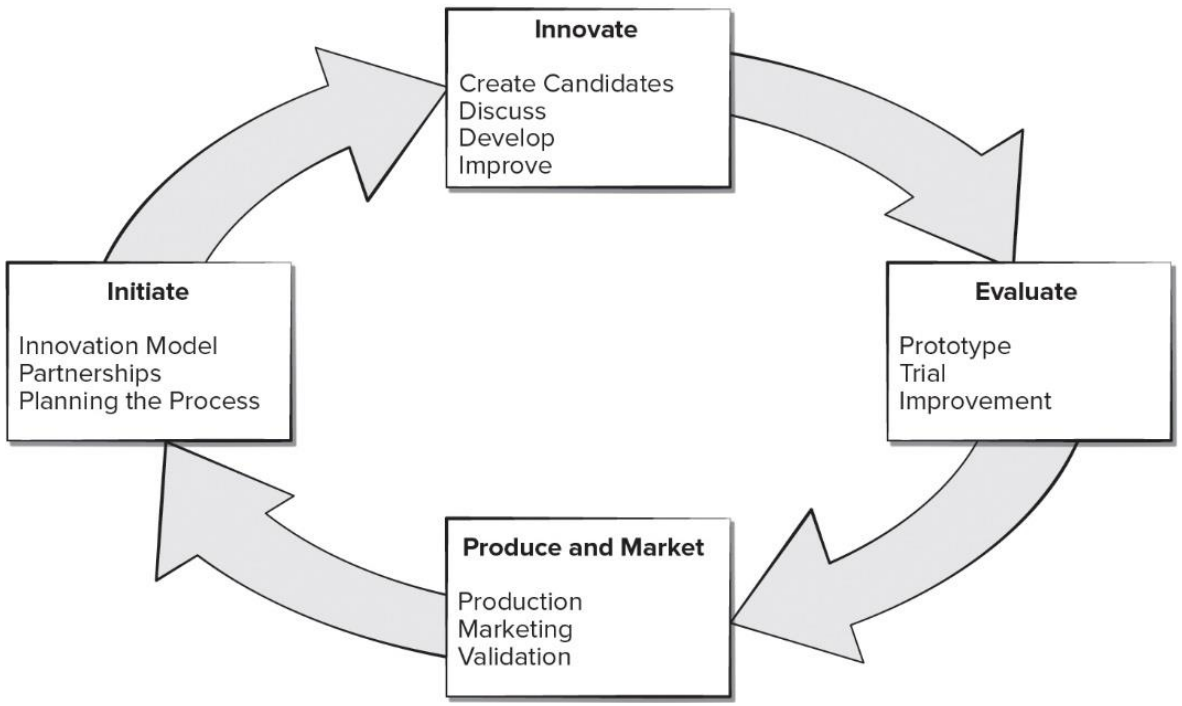
5. After the restricted phase 레거시 및 폐기 단계는 5년 동안 지속되며 5년 지원이 제공됨.

6. 레거시 단계에서 단종 알림(Obsolescence notice)은 소프트웨어가 단종되기 최소 6개월 전에 제공되어야 함

Release Management

- **Release management - process that brings high-quality code from developer's workspace to the end user includes:**
  - Code change integration.
  - Continuous integration.
    - developers regularly merge their code changes into a central repository, after which automated builds and tests are run.
  - Build system specifications.
  - Infrastructure-as-code.
    - IaC Terraform . 선언적 또는 스크립팅된 정의(코드)를 통해 하드웨어, 가상 리소스, 플랫폼, 컨테이너 시스템, 서비스 및 토폴로지를 포함한 인프라를 프로비저닝 및 관리하는 것을 의미
    - https://velog.io/@moonhj3117/Terraform-Module-%EC%82%AC%EC%9A%A9%EB%B2%95
  - Deployment and release.
  - Retirement.

Iterative Software Support Model



- A circular flowchart displays an iterative software support model.
- The components in the model are innovate, evaluate, produce and market, and initiate.
  - The intiate include innovation model, partnerships, and planning the process
  - The innovate phase create, discuss, develop, and improve.
  - The evaluate phase include prototype, trial, and improvement.
  - The produce and market include production, marketing, and validation.

Software Supportability

- 소프트웨어 지원성(Supportability):
  - 전체 제품 수명에 대해 소프트웨어 시스템을 지원하는 능력.
  - 이것은 필요한 수요나 요구 사항을 만족시킬뿐 아니 장비, 지원 인프라, 추가 소프트웨어, 설비, 인력 또는 소프트웨어를 운영하도록 유지하고 그 기능을 충족시킬 수 있는 기타 리소스를 제공하는 것을 의미."
    [Software-supportability.org, 2008, 소프트웨어 지원 가능성과 안정성 측면, 방법 및 기술을 홍보하고 공개하는 데 전념하는 비영리 그룹]
- 소프트웨어에는 운영 환경에서 bug가 발생했을 때 지원 담당자를 도와주는 기능이 포함되어야 함(실수하지 않아도 결함이 발생).
  - 예) 지원 담당자는 이미 발생한 모든 결함(특성, 원인 및 해결 방법)이 포함된 데이터베이스에 액세스할 수 있어야 함.

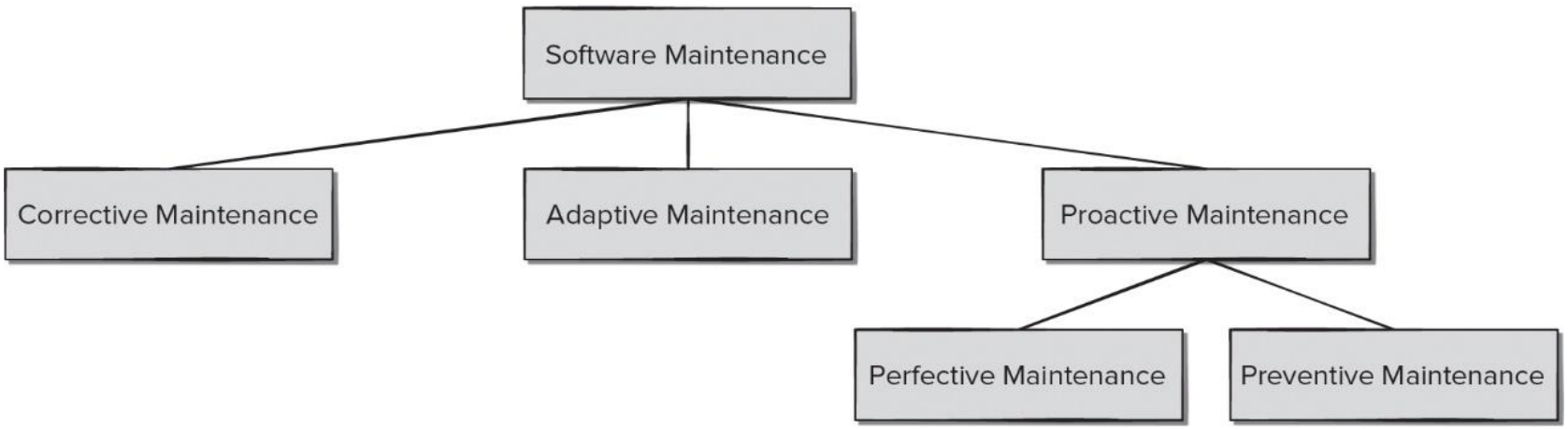
Software Maintenance

- Maintenance begins almost immediately
  - Software is released to end-users, and:
  - Within days, bug reports filter back to the software engineering organization. (버그 보고서가 필터링되어 소프트웨어 엔지니어링 조직으로 다시 전달)
  - Within weeks, one class of users indicates that the software must be changed so that it can accommodate the special needs of their environment. (해당 환경의 특별한 요구 사항을 수용할 수 있도록 소프트웨어를 변경)
  - Within months, another corporate group who wanted nothing to do with the software when it was released, now recognizes that it may provide them with benefits and want few enhancements to make it work better for their needs. (소프트웨어가 출시되었을 때 그 소프트웨어와 아무 관련이 없던 또 다른 기업은 이 소프트웨어가 자신들에게 예상치 못한 이익을 제공할 수 있음을 인식하고 그들이 그것을 활용하기 위해 몇 가지 개선 사항이 필요함)
- All of this work is software maintenance

Maintainable Software

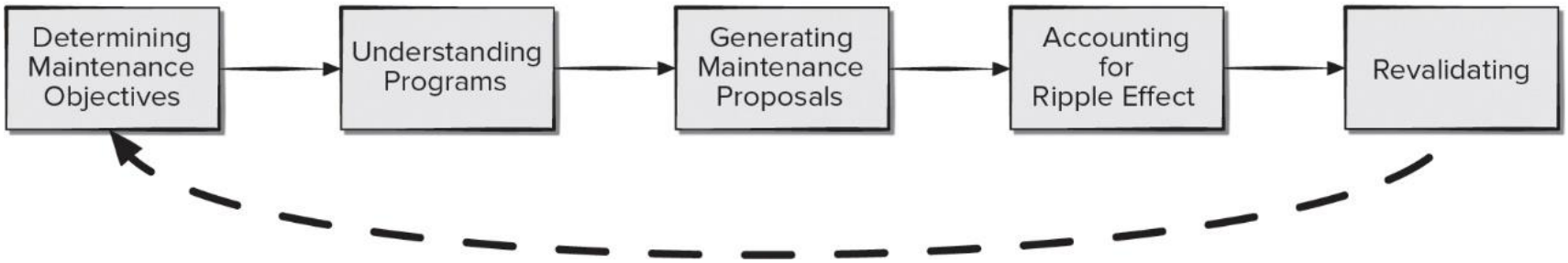
- 유지 관리 가능한 소프트웨어, 유지보수성이란 무엇인가?
  - 유지보수성은 기존 소프트웨어를 수정, 조정, 또는 향상할 수 있는 용이성을 나타내는 지표
  - 효과적인 모듈성과 이해하기 쉬운 디자인 패턴을 사용.
  - 잘 정의된 코딩 표준 및 규칙을 사용하여 구성되어 자체 문서화되고 이해하기 쉬운 소스 코드로 이어짐
- 소프트웨어가 릴리즈되기 전, 잠재적인 유지보수 문제를 해결하기 위한 다양한 품질 보증 기법 적용.
  - 변경해야 할 때 그들이 없을 수 있음을 인식하는 소프트웨어 엔지니어에 의해 만들어짐. (It has been created by software engineers who recognize that they may not be around when changes must be made)
- 따라서 소프트웨어 design 및 implementation은 변경을 수행해야하는 사람을 “assist ” 할 수 있어야 함

Maintenance Types



- Software maintenance is divided into corrective maintenance, adaptive maintenance, and proactive maintenance. The proactive maintenance is further divided into perfective maintenance and preventive maintenance.
- Corrective Maintenance: 에러를 고치는 유지보수 (bug fix), 잠재적인 오류를 찾아 수정
- Adaptive Manintenance :Requirement 또는 환경의 변화에 따라 바뀌는 것들에 대한 유지보수
- Preventive Maintenance: 완벽한 기능을 갖도록 하는 유지보수, 기능의 수정, 추가, 전반적인 기능 개선
- Preventive Manintenance :프로그램의 변경을 예측하여 준비

Maintenance Tasks



- An illustration displays maintenance tasks.
- Figure shows a set of generic tasks that should be completed as part of a controlled software maintenance process.
- The steps include determining maintenance objectives, understanding programs, generating maintenance proposals, accounting for ripple effect, and revalidation.
- After revalidation, the tasks repeated with determining maintenance objectives.

Agile Maintenance 1

Heeager and Rose [Hee15] suggest nine heuristics to help the maintenance become more agile.

1. Use sprints to organize the maintenance work. Balance the goal of keeping customers happy with technical needs of the developers.
   스프린트를 사용하여 유지 관리 작업을 구성, 고객 만족을 유지한다는 목표와 개발자의 기술 요구 사항 간의 균형
2. Allow urgent customer requests to interrupt scheduled maintenance sprints, make time for them during sprint planning.
   유지 관리 스프린트 계획 중 긴급한 고객 요청을 허용
3. Facilitate team learning by ensuring that more experienced developers are able to mentor less experienced team members.
   경험이 부족한 팀 구성원을 멘토링할 수 있도록 하여 팀 학습을 촉진
4. Allow multiple team members to accept customer requests as they and coordinate their processing with maintenance team members.
   여러 팀 구성원이 고객 요청이 발생하면 수락하고 다른 유지 관리 팀 구성원과 처리를 조정할 수 있도록 허용

Agile Maintenance 2

5. Balance the use of written documentation with face-to-face communication to ensure planning meeting time is used wisely. 서면 문서 사용과 대면 의사소통의 균형을 유지
6. Write informal use cases to supplement documentation being used for communications with stakeholders. 다른 문서를 보완하기 위해 informal use cae 사용
7. Have developers test each other's work (both defect repairs and new feature implementations). 서로의 작업(결함 수리 및 새로운 기능 구현 모두)을 테스트
8. Make sure developers are empowered to share knowledge with one another. Motivates people to improve the skills and knowledge. 서로 지식을 공유할 수 있는 권한을 부여, 기술과 지식을 향상시키도록 동기를 부여
9. Keep planning meetings short, frequent, and focused. 회의를 짧고, 자주, 집중적으로 계획

Reverse Engineering

- Definition
  - 시스템의 구성요소와 상호관계를 식별하고 시스템을 다른 형태로 표현하기 위해 대상 시스템을 분석하는 프로세스, 재 문서화(redocumentation)와 설계 복구
    https://en.wikipedia.org/wiki/Reverse_engineering
  - 소프트웨어 유지 관리 및 개선을 위한 기본 소스 코드에 대한 이해를 높이는 데 도움이 될 수 있으며, 관련 정보를 추출하여 소프트웨어 개발에 대한 결정을 내릴 수 있으며, 코드의 그래픽 표현은 소스 코드에 대한 대체 보기를 제공
- Reverse Engineering to Understand Data
  - First reengineering task often begins by constructing UML class diagram.
  - 유지 관리되는 시스템의 품질이 낮고 합리적인 문서가 부족한 경우 많으며 기술적 부채 (technical debt)는 개발자가 기능을 문서화하지 않거나 더 큰 소프트웨어 시스템에 미치는 영향을 고려하지 않고 기능을 추가함으로써 발생하는 경우 발생
  - Reverse engineering requires developers to evaluate the old software system by examining its (often undocumented) source code, developing a meaningful specifica- tion of the processing being performed, the user interface that was used, and the program data structures or associated database 소스 코드를 검사하고, 수행 중인 처리, 사용된 사용자 인터페이스, 프로그램 데이터 구조 또는 관련 데이터베이스에 대한 의미 있는 사양을 개발하여 기존 소프트웨어 시스템을 평가

- Reverse Engineering to Understand Data (cont.)
  - At the program level, internal program data structures must often be reverse engineered as part of an overall reengineering effort.
  - At the system level, global data structures (e.g., files, databases) are often reengineered to accommodate new database management paradigms (e.g., the move from flat file to relational or object oriented database systems). 전역 데이터 구조(예: 파일, 데이터베이스)가 종종 재설계
- Reverse Engineering of Internal Data Structures
  - focuses on the definition of object classes.
  - This is accomplished by examining the program code with the intent of grouping related program variables together. 관련 프로그램 변수를 함께 그룹화하려는 의도로 프로그램 코드를 검사함으로써 수행
  - In many cases, the data organization within the code suggests several abstract data types. 여러 추상 데이터 유형을 제안
  - For example, record structures, files, lists, and other data structures often provide initial suggestions for possible classes

- Reverse Engineering of Database Structure
  - Reengineering one database schema into another requires an understanding of existing objects and their relationships. 하나의 데이터베이스 스키마를 다른 데이터베이스 스키마로 리엔지니어링하려면 기존 개체와 해당 관계에 대한 이해가 필요
  - may be used to define the existing data model as a precursor to reengineering a new database model
    - (1) 초기 객체 모델 구축
    - (2) 후보 키 결정(속성은 다른 레코드나 테이블을 가리키는 데 사용되는지 여부를 결정하기 위해 검사. 포인터 역할을 하는 키는 후보 키가 됨)
    - (3) 임시 클래스를 개선
    - (4) 일반화
    - (5) CRC Approach와 유사한 방식을 사용하여 연관성 발견

Reverse Engineering

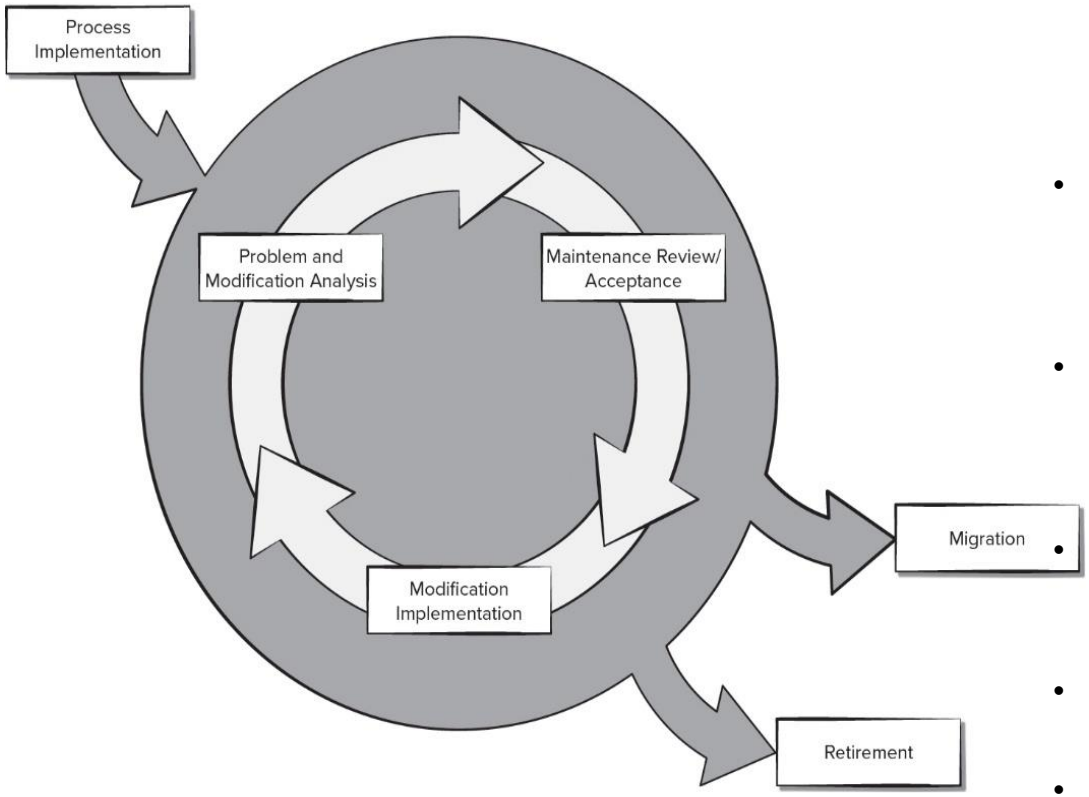
- Reverse engineering to understand processing
  - attempts to understand procedural abstractions in source code.
  - 절차적 추상화를 이해하기 위해 시스템, 프로그램, 구성 요소, 패턴 및 명령문 등 다양한 추상화 수준에서 코드를 분석
  - 리버스 엔지니어링 작업을 수행하기 전에 전체 애플리케이션의 전반적인 기능을 이해. 컨텍스트를 설정하고 대규모 시스템 내 애플리케이션 간의 상호 운용성 문제에 대한 통찰력을 제공
  - Code: 일반적인 절차 패턴을 나타내는 코드 섹션을 찾음. 모듈 내에서 처리할 데이터, 처리를 수행, 처리 결과를 준비, 데이터 유효성 검사 및 경계 확인
  - Automated tools can be used to help you understand the semantics of existing code. The output of this process is then passed to restructuring and forward engineering tools to complete the reengineering process.

Reverse Engineering

- Reverse engineering to understand user interfaces
  - may need to be done as part of the maintenance task (for example, adding a GUI).
  - Merlo and his colleagues suggest three basic questions that must be answered as reverse engineering of the UI commences:
    - 인터페이스가 처리해야 하는 기본 동작(예: 키 입력 및 마우스 클릭)은?
    - 이러한 행동에 대한 시스템의 행동 반응을 간략하게 설명하면?

    “replacement"란 무엇? 더 정확하게는 어떤 인터페이스 등가 개념이 여기서 관련되나?

Proactive Software Support Model



Software maintenance and support process

- 소프트웨어 엔지니어링의 목표가 시기적절하고 비용 효율적인 방식으로 고객 요구 사항을 충족하는 고품질 제품을 제공하는 것이라면 SW Support는 다른 소프트웨어 엔지니어링 활동과 마찬가지로 불필요한 재작업을 피할 수 있도록 관리되는 프로세스 사용
- 문제 발생 후에는 시간과 비용이 많이 드는 프로세스일 수 있으므로 문제가 발생하기 전에 문제를 예측하고 고객 우려 사항에 대응하는 데 필요한 작업 일정을 잡는 것이 중요 (사전예방)
- Proactive software support(사전예방적 소프트웨어 지원)은 소프트웨어 엔지니어가 문제가 발생하기 전에 문제를 식별하고 해결하는 데 도움이 되는 도구와 프로세스를 의미
- 개발자는 소프트웨어 제품에 품질 문제가 있을 수 있음을 시사하는 지표를 검색한 후 modification 혹은 최신 버전으로 마이그레이션하여 해결
- 품질을 향상시키고 유지 관리를 더 쉽게 만들기 위해 재구성되거나 리팩터링
- 문제가 너무 심각하여 고객이 제품을 포기하기 전에 개발자가 제품 폐기 계획을 세우고 대체 제품 개발

Software Analytics and Proactive Maintenance

- There are currently three dominant uses for artificial intelligence methods in software engineering work
  - Probabilistic reasoning techniques can be used to model software reliability (확률추론)
  - Machine learning can be used to automate the process of discovering root causes of software failures before they occur by predicting the presence of defects likely to cause these failures
  - Search based software engineering may be used to assist developers in identifying useful test cases to make regression testing more effective 소프트웨어 엔지니어링 문제 에 유전자 알고리즘 , 금기 검색 과 같은 메타휴리스틱(최적화 문제를 제한된 시간과 자원으로 효율적으로 풀기 위한 알고리즘) 검색 기술을 적용합니다. 소프트웨어 엔지니어링 의 많은 활동 https://en.wikipedia.org/wiki/Search-based_software_engineering

1. Be sure you are using analytics to identify meaningful development problems, or you will get no buy in from the software engineers.
   - 의미 있는 개발 문제를 식별하기 위해 분석을 사용하고 있는지 확인하십시오. 그렇지 않으면 소프트웨어 엔지니어로부터 동의를 얻지 못함
   - 분석을 사용하여 제품에 아직 발견되지 않은 결함에 대한 추정을 기반으로 한 결함 발견 비율, 작동 중 결함 발견 사이의 시간, 결함을 수리하는 데 필요한 노력을 추정할 수 있다고 제안
   - 최종 사용자가 적극적으로 사용할 수 있도록 시스템을 출시한 후 시스템 유지 관리에 할당해야 하는 비용과 시간 측면에서 더 나은 계획

Software Analytics and Proactive Maintenance

2. The analytics must make use of application domain knowledge to be useful to developers (this implies the use of experts to validate the analytics).
   개발자에게 유용하도록 애플리케이션 도메인 지식을 활용
3. Developing analytics requires iterative and timely feedback from the intended users. 사용자의 반복적이고 시기적절한 피드백이 필요
4. Make sure the analytics are scalable to larger problems and customizable to incorporate new discoveries made over time. 더 큰 문제로 확장 가능하고 새로운 발견을 통합하도록 사용자 정의
5. Evaluation criteria used needs to be correlated to real software engineering practices. 사용된 평가기준은 실제 소프트웨어 엔지니어링 실무와 연관

Role of Social Media

- Many online stores allow users to provide feedback on the apps by posting ratings or comments.
- The feedback found in these reviews may contain usage scenarios, bug reports, or feature requests. 온라인 스토어 리뷰 분석, 버그 보고, 기능 요청, 사용자가 앱을 어떻게 사용하는지에 대한 구체적인 예시
- Mining these reports can help developers identify potential maintenance and software evolution tasks. 잠재적인 유지 관리 및 소프트웨어 발전 작업을 식별하는 데 도움
- Many companies maintain Facebook pages or Twitter feeds to support their user communities.
- Some companies encourage product users to send program crash information for analysis by the support team members. 제품 사용자에게 프로그램 crash 정보를 보내도록 권장
- Some companies use the questionable practice of tracking how and where products are used by customers without their knowledge. 고객이 알지 못하는 사이에 고객이 제품을 사용하는 방법과 위치를 추적하는 의심스러운 관행

Cost of Support 1

- Before an organization attempts to modify or replace an existing application, it should perform a cost-benefit analysis
- Cost-Benefit Analysis: 기업이 내릴 결정과 포기할 결정을 분석하는 데 사용하는 체계적인 프로세스. 상황이나 행동에서 기대되는 잠재적 보상을 합산한 다음 해당 행동을 취하는 데 드는 총 비용을 차감. [sneed,H "Planning the Reengineering of Legacy System", 1995, IEEE=
- Nine parameters are defined:
  - P1 = current annual maintenance cost for an application. 현재 연간 유지 관리 비용.
  - P2 = current annual operation cost for an application. 연간 운영 비용
  - P3 = current annual business value of an application. 연간 비즈니스 가치
  - P4 = predicted annual maintenance cost after reengineering. 리엔지니어링 이후 예상 연간 유지관리 비용
  - P5 = predicted annual operations cost after reengineering. 리엔지니어링 이후 예상 연간 운영 비용
  - P6 = predicted annual business value after reengineering. 리엔지니어링 이후 연간 비즈니스 가치
  - P7 = estimated reengineering costs. 예상 리엔지니어링 비용
  - P8 = estimated reengineering calendar time. 예상 리엔지니어링 일정 시간.
  - P9 = reengineering risk factor (P9 = 1.0 is nominal). 리엔지니어링 위험 요소(P9 = 1.0이 명목임).
  - L = expected life of the system. 시스템의 예상 수명

Cost of Support 2

- The cost associated with continuing maintenance of a candidate application (that is, reengineering is not performed) can be defined as:리엔지니어링이 수행되지 않음

$$C_{maint} = [P_3 - (P_1 + P_2)] \times L$$

- The costs associated with reengineering are defined using the following relationship: 리엔지니어링과 관련된 비용

$$C_{reeng} = [P_6 - (P_4 + P_5) \times (L - P_8) - (P_7 \times P_9)]$$

- Using the costs presented in equations above, the overall benefit of reengineering can be computed as: 리엔지니어링의 전반적인 이점

$$\text{Cost benefit} = C_{reeng} - C_{maint}$$

- 가장 높은 비용 대비 이점을 보이는 애플리케이션은 사전 유지 관리 또는 개선의 대상이 될 수 있으며, 다른 애플리케이션에 대한 작업은 리소스를 사용할 수 있을 때까지 연기

Refactoring

- "Refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure". – MartinFowler

- 소프트웨어 리팩토링(구조 조정이라고도 함)은 향후 변경 사항에 적응할 수 있도록 소스 코드 및/또는 데이터를 수정, 가독성과 유지보수성을 높이는 작업
  - 일반적으로 리팩토링은 전체 프로그램 아키텍처를 수정하지 않음
  - 개별 모듈의 설계 세부 사항과 모듈 내에 정의된 로컬 데이터 구조에 초점을 맞추는 경향
  - 리팩토링 노력이 모듈 경계를 넘어 소프트웨어 아키텍처까지 확장된다면 구조 조정은 포워드 엔지니어링
  - 리팩토링은 기술적 내부 작업이 필요하더라도 애플리케이션의 기본 아키텍처가 견고할 때 발생
  - 소프트웨어의 주요 부분이 서비스 가능하고 모든 모듈 및 데이터의 하위 집합에만 광범위한 수정이 필요할 때 시작.

Data Refactoring

- 데이터 리팩토링을 시작하기 전에 소스 코드 분석이라는 리버스 엔지니어링 활동을 수행.
- Data analysis.
  - 데이터 정의, 파일 설명, I/O 및 인터페이스 설명을 포함하는 모든 프로그래밍 언어 명령문이 평가됨.
  - 데이터 items와 objects를 추출하고, 데이터 흐름에 대한 정보를 얻고, 구현된 기존 데이터 구조를 이해하는 것이 목적
- Data redesign
  - 가장 간단한 형태의 data record standardization step은 기존 데이터 구조나 파일 형식 내에서 데이터 항목 이름이나 물리적 기록 형식 간의 일관성을 달성하기 위해 데이터 정의를 명확하게 함.
- data name rationalization
  - 모든 데이터 명명 규칙이 로컬 표준을 준수하고 시스템을 통한 데이터 흐름에서 별칭이 제거되도록 보장
- 리팩토링이 표준화와 합리화를 넘어설 때 기존 데이터 구조를 물리적으로 수정하여 데이터 디자인을 더욱 효과적.
- 한 파일 형식에서 다른 파일 형식으로의 변환을 의미할 수 있으며, 어떤 경우에는 한 데이터베이스 유형에서 다른 유형의 데이터베이스로의 변환을 의미할 수도 있음.

Code Refactoring

- 코드 리팩토링은 동일한 기능을 생성하지만 원본 프로그램보다 품질이 더 높은 디자인을 생성하기 위해 수행.
- 목표는 "spaghetti-bowl" 코드를 가져와 제품에 대해 정의된 품질 요소를 준수하는 디자인을 도출하는 것.
- 또 다른 접근 방식은 안티 패턴을 사용하여 잘못된 코드 설계 관행을 식별하고 가능한 솔루션을 제안하는 것. (너무 많은인수, 사용되지 않는 메소드, 잘못된 추상화 등)
- 코드 리팩토링은 디버깅이나 리엔지니어링이 아닌 작은 소프트웨어 변경과 관련된 즉각적인 문제를 완화할 수 있음.
- 코드 리팩토링의 실질적인 이점은 데이터와 아키텍처도 리팩토링되어야 얻을 수 있음
- 예) Clean Code
  - 객체의 생성에 유의미한 이름 사용 : 명확한 코드, 구현을 드러내는 이름은 피하라
  - 함수는 하나의 역할 , 지정된 이름아래에서 한단계 수준의 추상화 수준 유지(SRP)
  - 명령과 조회를 분리
  - 오류코드 보다는 예외 활용

Architectural Refactoring

- Architectural refactoring as one of the design trade-off options for dealing with a messy program (cost and time-consuming process이지만 trade-off를 고려해야함)

1. You can struggle through modification after modification, fighting the ad hoc design and tangled source code to implement the necessary changes. 수정 후 수정을 통해 어려움을 겪을 수 있으며, 필요한 변경 사항을 구현하기 위해 임시 설계 및 얽힌 소스 코드와 싸울 수 있음
2. You can attempt to understand the broader inner workings of the program to make modifications more effectively. 효과적으로 수정하기 위해 프로그램의 더 넓은 내부 작동 방식을 이해하려고 시도
3. You can revise (redesign, recode, and test) those portions of the software that require modification, applying a meaningful software engineering approach to all revised segments. 수정된 모든 세그먼트에 의미 있는 소프트웨어 엔지니어링 접근 방식을 적용
4. You can completely redo (redesign, recode, and test) the program, using reengineering tools to understand the current design. 리엔지니어링 도구를 사용하여 프로그램을 완전히 다시 실행(재설계, 재코딩 및 테스트)

Architectural Refactoring

- 단 하나의 "올바른" 옵션은 없습니다. There is no single “correct” option
- 상황에 따라 다른 옵션이 더 바람직하더라도 첫 번째 옵션이 필요할 수 있습니다.
- 개발 또는 지원 조직은 유지 관리 요청이 접수될 때까지 기다리지 않고 inventory analysis를 사용하여 다음과 같은 프로그램을 선택.

  (1) 미리 선택된 기간 동안 계속 사용.

  (2) 현재 성공적으로 사용되고 있음.

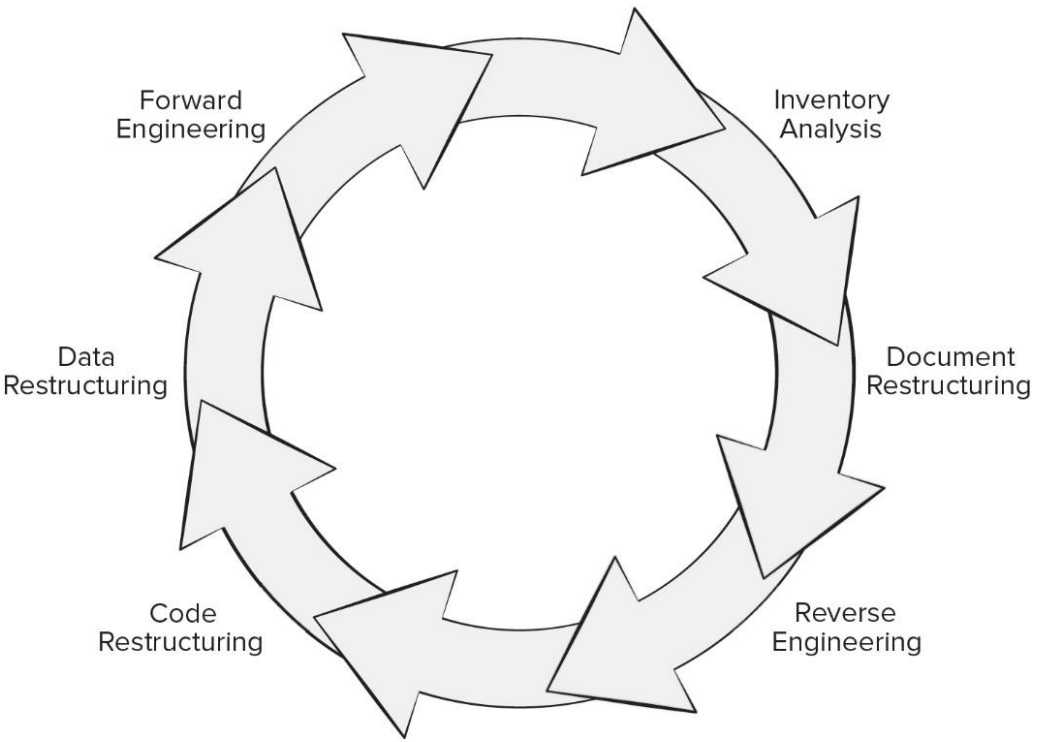
  (3) 가까운 미래에 대대적인 수정이나 개선이 이루어질 가능성이 높음.

  그런 다음 옵션 2, 3 또는 4가 적용됩니다.

Software Evolution

- 리엔지니어링에는 시간이 걸리고 상당한 비용이 소요되며 즉각적인 문제로 인해 점유될 수 있는 자원을 흡수.
- 모든 조직에는 소프트웨어 리엔지니어링을 위한 실용적인 전략이 필요
- 파레토 원칙을 적용하고 문제의 80%를 차지하는 20%의 소프트웨어에 리엔지니어링 프로세스를 적용하는 것을 고려
- Consider the following arguments
  - The cost to maintain one line of source code may be 20 to 40 times the cost of initial development of that line. 초기 개발 비용의 20~40배
  - Redesign of the software architecture (program and/or data structure), using modern design concepts, can greatly facilitate future maintenance. 재설계하면 향후 유지 관리가 크게 용이
  - Because a prototype of the software already exists, development productivity should be much higher than average. 개발 생산성은 훨씬 높아야 함
  - Tools for reengineering will automate some parts of the job. 도구는 작업의 일부 부분을 자동화
  - A complete software configuration (documents, programs and data) when the evolutionary preventive maintenance is done.

Reengineering Process Model



- A circular flowchart displays reengineering process model.
- The components in the model are forward engineering, inventory analysis, document restructuring, reverse engineering, code restructuring, and data restructuring.
- The process continues with forward engineering after data restructuring.

Inventory Analysis

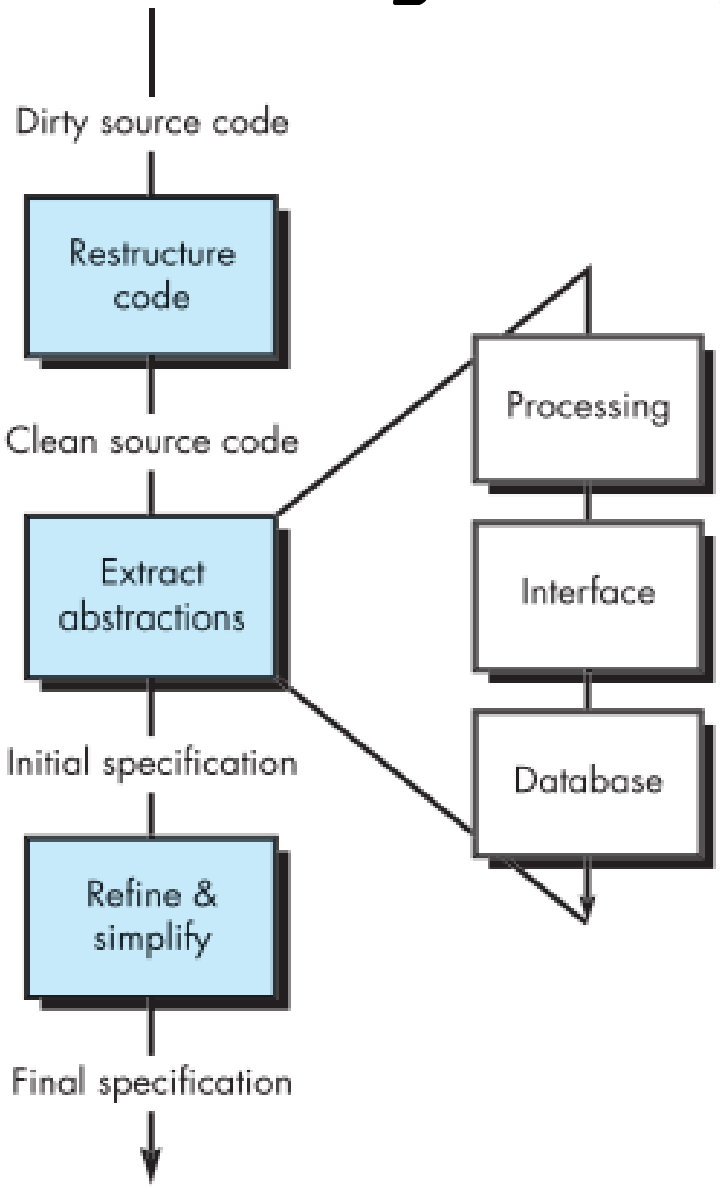
- 모든 애플리케이션을 포함하는 목록 구축
- 예를 들어, 기준 목록을 설정.
  - 응용 프로그램의 이름
  - 원래 만들어진 연도
  - 실질적으로 변경된 횟수
  - 이러한 변경을 수행하는 데 적용된 총 노력
  - 마지막 실질적 변경 날짜
  - 마지막 변경을 수행하기 위해 적용된 노력
  - 관련 시스템
  - 인터페이스하는 응용 프로그램,
- 리엔지니어링 후보를 선택하기 위해 분석하고 우선 순위를 지정합니다.

Document Restructuring

- 빈약한 문서는 많은 레거시 시스템의 트레이드마크.
- 그러나 그것에 관해 무엇을 할 수 있고, 선택은 무엇인가?
- Options
  - 문서 작성에는 너무 많은 시간이 소요되며 비용이 많이듬.
    - 시스템이 잘 작동하면 그대로 두어라.
    - 어떤 경우에는 이것이 올바른 접근 방식.
  - 문서를 업데이트해야 하지만 리소스가 제한되어 있음.
    - 우리는 " 변경사항이 있을 경우만 문서화 " 접근 방식.
    - 변경 신청서를 완전히 다시 문서화할 필요가 없을 수도 있음.
    - 꼭 필요한 만큼만 문서화
  - 이 시스템은 업무상 중요하므로 완전히 다시 문서화해야 함.
    - 이 경우에도 지능적인 접근 방식은 문서를 최소한으로 줄이는 것.

Reverse Engineering

- Reverse Engineering(역공학)
  - 소스코드보다 높은 추상화 수준분석
  - 기존 프로그램으로
    - 데이터
    - 아키텍처
    - 절차적 설계정보 추출



Code Restructuring

- 소스 코드는 리팩토링 도구를 사용하여 분석
- 잘못 디자인된 코드가 재 설계됨
- 구조화된 프로그래밍 위반 사항이 기록되고 코드가 리팩토링됨(이 작업은 자동으로 수행될 수 있음).
- 리팩터링된 결과 코드는 검토 및 테스트되어 이상이 발생하지 않았는지 확인.
- 내부 코드 문서가 업데이트

Data Restructuring

- 상대적으로 낮은 수준의 추상화에서 발생하는 코드 재구성과 달리 데이터 구조화는 본격적인 리엔지니어링 활동.
- 대부분의 경우 데이터 재구성은 리버스 엔지니어링 활동으로 시작.
  - 현재 데이터 아키텍처가 분석되고 필요한 데이터 모델이 정의.
  - 데이터 객체와 속성이 식별되고 기존 데이터 구조의 품질이 검토.
  - 데이터 구조가 취약한 경우(예: flat files are currently implemented , 관계형 접근 방식이 처리를 크게 단순화할 때) 데이터 리엔지니어링.
- 데이터 아키텍처는 프로그램 아키텍처와 이를 채우는 알고리즘에 큰 영향을 미치기 때문에 데이터를 변경하면 아키텍처/코드 수준의 대규모 변경이 필요함.

Forward Engineering

- In an ideal world, applications would be rebuilt using an automated "reengineering engine. 이상적인 세계에서는 자동화된 "리엔지니어링 엔진"을 사용하여 애플리케이션을 재구축
- Forward engineering recovers design information from existing software and uses this information to alter or reconstitute the existing system to improve its overall quality. 기존 소프트웨어에서 설계 정보를 복구하고 이 정보를 사용하여 기존 시스템을 변경하거나 재구성하여 전반적인 품질을 향상
- Reengineered software re-creates the function of the existing system and adds new functions and/or improves overall performance. 기존 시스템의 기능을 재창조하고 새로운 기능을 추가하거나 전반적인 성능을 향상
- Forward engineering does not simply create a modern equivalent of an older program – the redeveloped program extends the capabilities of the older application. 단순히 기존 프로그램과 동등한 최신 버전을 만드는 것이 아님. 기존 애플리케이션의 기능을 확장

Software Process Improvement

What is Software Process Improvement?

- SPI implies that:
  - Elements of an effective software process can be defined in an effective manner. 효과적인 소프트웨어 프로세스의 요소가 효과적인 방식으로 정의될 수 있음을 의미
  - An existing organizational approach to software development can be assessed against those elements, 기존의 조직적 접근 방식을 해당 요소에 대해 평가
  - A meaningful strategy for improvement can be defined. 의미 있는 전략을 정의

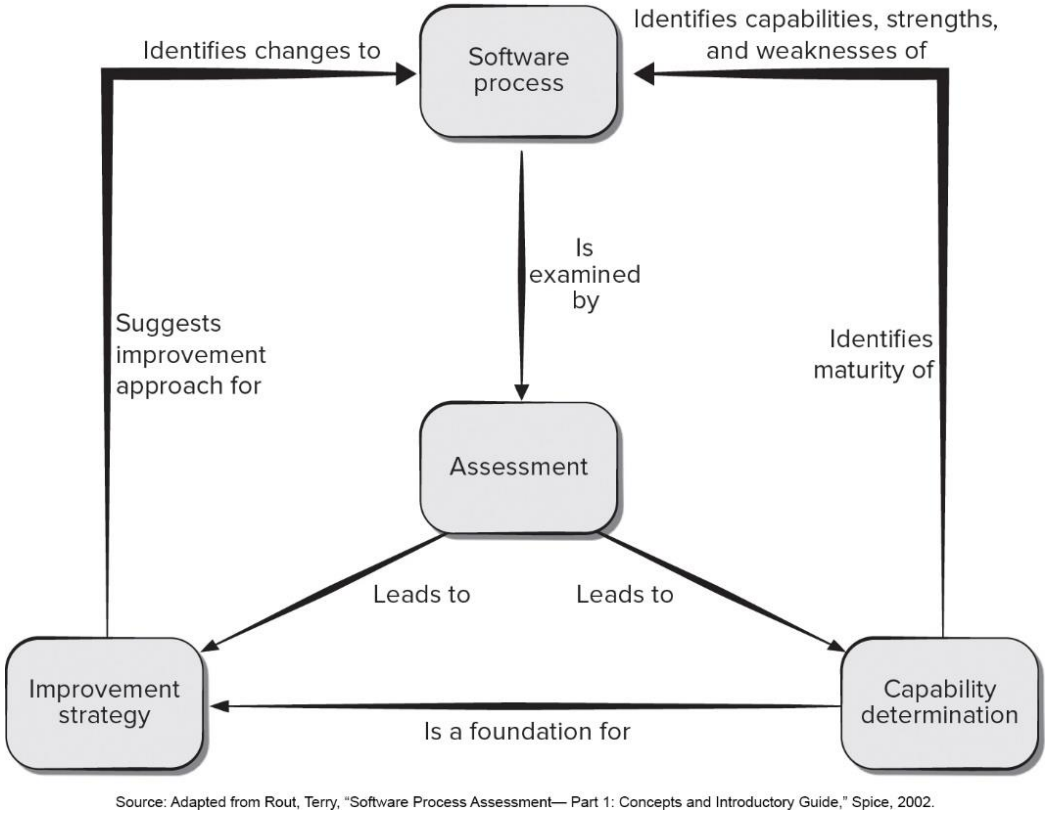
- The SPI 전략은 소프트웨어 개발에 대해 더 집중적이고, 더 반복 가능하며, 더 신뢰할 수 있는 방식(제품 품질 및 납품 적시성 측면)으로 전환되어야 하며 SPI 전략을 구현하는 데 필요한 노력과 시간은 측정 가능해야 함
- 개선된 프로세스와 관행의 결과로 시간과 비용이 소요되는 소프트웨어 "문제"가 감소
- 최종 사용자에게 전달되는 결함 수를 줄이고, 품질 문제로 인한 재작업량을 줄이고, 소프트웨어 유지 관리 및 지원과 관련된 비용을 줄이고, 소프트웨어가 늦게 전달될 때 발생하는 간접 비용을 줄여야 함

SPI Framework

1. 효과적인 소프트웨어 프로세스를 달성하려면 반드시 존재해야 하는 일련의 특성.
2. 이러한 특성이 존재하는지 여부를 평가하는 방법.
3. 평가 결과를 요약하는 메커니즘.
4. 취약하거나 누락된 것으로 밝혀진 프로세스 특성을 구현하는 조직을 지원하기 위한 전략.

- An SPI framework assesses the “maturity” of an organization’s software process and provides a qualitative indication of a maturity level. 프로세스의 "성숙도"를 평가하고 성숙도 수준에 대한 질적 지표를 제공

SPI Framework Elements



Source: Adapted from Rout, Terry, "Software Process Assessment— Part 1: Concepts and Introductory Guide," Spice, 2002.

- An illustration displays SPI framework elements.
- The elements are software process, assessment, capability determined, and improvement strategy.
- The Software process is examined by assessment and that leads to improvement strategy, and capability determination.
- The capability determination is a foundation for improvement strategy.
- The capability determination identifies capabilities, strengths, and weakness of the software process, and identifies maturity of it.
- The improvement strategy identifies changes to software process, and suggest improvement approach for it.

Establishing SPI Frameworks 1

- As an SPI framework is applied, an organization must establish mechanisms to: 조직은 다음을 위한 메커니즘을 확립

1. Support technology transition. 기술 전환을 지원
2. Determine the degree to which an organization is ready to absorb process changes that are proposed. 제안된 프로세스 변경 사항을 어느 정도 흡수할 준비가 되어 있는지를 결정
3. Measure the degree to which changes have been adopted. 변경 사항이 채택된 정도를 측정

Establishing SPI Frameworks 2

- A maturity model is applied within the context of an SPI framework. (프로세스 성숙도 모델)
- The intent of the maturity model is to provide an overall indication of the "process maturity" exhibited by a software organization by showing:
  - An indication of the quality of the software process. 소프트웨어 개발 프로세스의 효과성, 효율성, 일관성 등을 나타내는 척도
  - The degree to which practitioner's understand and apply the process, 실무자의 프로세스 이해 및 적용
  - The general state of software engineering practice. 조직 전체의 소프트웨어 개발 문화, 툴, 방법론 등을 포함한 일반적인 실무 상태
- 프로세스 성숙도 모델의 중요성:
  - 개선 영역 파악 : 평가함으로 강점과 약점 파악, 개선 영역 식별
  - 벤치마킹 : 다른 조직과 비교하여 프로세스 성숙도 평가
  - 고객 만족 향상 : 프로젝트 성공 가능성이 높아지고 결함이 적은 고품질의 SW 제작
  - 프로세스 개선 계획 수립 : 구체적인 프로세스 개선 계획 수립

Is SPI for Everyone?

- Can a small company initiate SPI activities and do it successfully?
  - 오늘날 모든 소프트웨어 개발의 상당 부분은 직원이 100명 미만인 회사(또는 24명 미만의 스타트업의 경우)에서 수행
- Answer: a qualified "yes."
- It should come as no surprise that small organizations are more informal, apply fewer standard practices, and tend to be self-organizing.(소규모 조직이 더 비공식적이고, 더 적은 수의 표준 관행을 적용하며, 자체 조직화되는 경향)
  - 많은 프로젝트에 대한 정량적 분석을 통해 소규모 조직이 선호하는 민첩한 프로젝트 방법론이 프로세스 효율성을 높이고 고객 만족도를 높일 수 있는 것으로 나타남
- SPI will be approved and implemented only after its proponents demonstrate financial leverage.(재정적 영향력은 기술적 이점(예: 현장으로 전달되는 결함 감소, 재작업 감소, 유지 관리 비용 절감, 시장 출시 기간 단축 등)을 검토하고 이를 달러로 환산하여 입증)

SPI Process 1

- Assessment examines a wide range of actions and tasks that will lead to a high-quality process.(평가에서는 고품질 프로세스로 이어지는 광범위한 조치와 작업을 검사)
  - Consistency. Are important activities, actions and tasks applied consistently across all software projects and by all software teams? (모든 소프트웨어 프로젝트와 모든 소프트웨어 팀에서 중요한 활동, 작업 및 작업이 일관되게 적용)
  - Sophistication(복잡화). Are management and technical actions performed with a level of sophistication that implies a thorough understanding of best practice? 모범 사례에 대한 철저한 이해를 의미하는 정교한 수준으로 관리 및 기술 작업이 수행
  - Acceptance(수락). Is the software process and software engineering practice widely accepted by management and technical staff? 소프트웨어 프로세스와 소프트웨어 엔지니어링 관행이 경영진과 기술 직원에게 널리 받아들여지고 있나?
  - Commitment(약속). Has management committed the resources required to achieve consistency, sophistication and acceptance? 경영진이 일관성, 정교함, 수용성을 달성하는 데 필요한 자원을 투입
- Gap analysis—The difference between local application and best practice represents a “gap” that offers opportunities for improvement. 개선 기회를 제공하는 " gap "을 나타냄

- Three types of Education and Training: (실무자, 기술 관리자 및 고위 관리자를 위한 교육 및 훈련으로 일반적인 소프트웨어 엔지니어링 개념 및 방법, 특정 기술 및 도구, 커뮤니케이션 및 품질 지향 교육)
  - Generic concepts and methods. Directed toward both managers and practitioners, this category stresses both process and practice. (관리자와 실무자 모두를 대상으로 하는 이 카테고리는 프로세스와 실무를 모두 강조)
  - Specific technology and tools. Directed primarily toward practitioners, this category stresses technologies and tools that have been adopted for local use. (실무자를 대상으로 하는 이 카테고리는 현지 사용을 위해 채택된 기술과 도구를 강조)
  - Business communication and quality-related topics. Directed toward all stakeholders, this category focuses on “soft” topics that help enable better communication among stakeholders and foster a greater quality focus. (모든 이해관계자를 대상으로 하는 이 카테고리는 이해관계자 간의 더 나은 의사소통을 가능하게 하고 더 큰 품질 초점을 조성하는 데 도움이 되는 "소프트" 주제에 중점)

- Selection and Justification:
  - 초기 평가 활동이 완료되고 교육이 시작되면 소프트웨어 조직은 selection and justification을 시작
  - 소프트웨어 프로세스 특성과 특정 소프트웨어 엔지니어링 방법 및 도구를 선택하는 활동 중 발생
    - 조직, 이해관계자 및 구축하는 소프트웨어에 가장 적합한 프로세스 모델을 선택
      - 적용될 프레임워크 활동 세트, 생성될 주요 작업 제품, 팀이 진행 상황을 평가할 수 있는 품질 보증 체크포인트를 결정
      - SPI 평가 활동에서 특정 약점이 있음을 나타내는 경우(예: 공식적인 SQA 기능이 없음) 이러한 약점을 직접 해결하는 프로세스 특성에 주의를 집중
    - 각 프레임워크 활동(예: 모델링)에 대한 적응형 작업 분석을 개발하여 일반적인 프로젝트에 적용될 작업 세트를 정의
      - 이러한 작업을 달성하기 위해 적용할 수 있는 소프트웨어 엔지니어링 방법도 고려
    - 선택이 이루어지면 이해가 강화되도록 교육과 훈련을 조정
      - 선택을 할 때 조직의 문화와 각 선택이 이끌어낼 수용 수준을 반드시 고려.
      - 잘못된 선택이 좋은 것보다 더 많은 해를 끼칠 수 있다는 것은 사실
      - 프로세스 특성이나 기술 요소가 조직의 요구 사항을 충족할 수 있는 좋은 기회를 갖고 있는 한, 때로는 완벽한 솔루션을 기다리는 것보다 방아쇠를 당겨 선택하는 것이 더 나을 때도 있음.

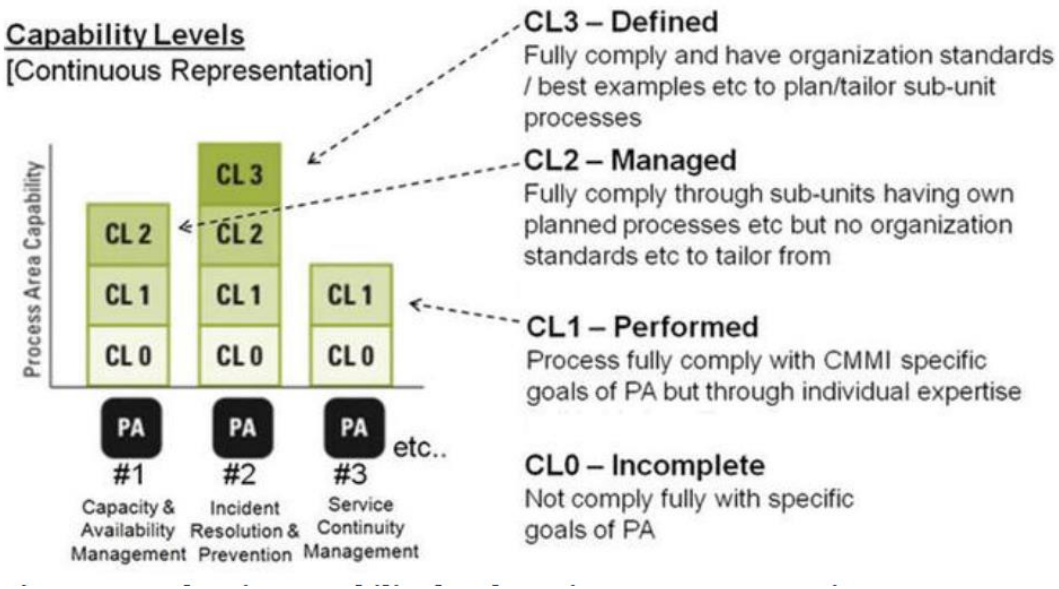
- Installation/Migration:
  - Software process redesign (SPR) is concerned with identification, application, and refinement of new ways to dramatically improve and transform software processes.
    (설치 및 마이그레이션은 SPR(소프트웨어 프로세스 재설계) 활동.)
  - Three different SPR process models are considered:
    - The existing ("as-is") process,
    - A transitional ("here-to-there") process, and
    - The target ("to be") process.

- Evaluation:
  - SPI(Software Process Improvement) 활동의 효과를 측정하는 것은 프로세스 개선의 진전 상황을 파악하고 실질적인 이익을 확인하는 데 중요
  - 변화의 도입 및 활용 정도:
    - 이는 프로세스 개선을 위해 도입된 새로운 방법, 툴, 기준 등이 프로젝트에서 얼마나 잘 사용되고 있는지를 평가하는 부분.개발 팀 구성원들이 새로운 프로세스를 이해하고 실제 작업에 적용하는 정도를 파악
  - 소프트웨어 품질 및 프로세스 개선 효과
    - 프로세스 개선을 통해 소프트웨어 품질이 향상되었는지 측정
  - 프로세스 및 조직 문화 변화
    - 프로세스 개선 활동에 대한 조직 구성원들의 참여도와 의견 수렴 여부
    - 지속적인 개선 문화가 형성되었는지 여부
    - 프로세스 개선에 대한 조직의 리더십과 지원
- Quantitative metrics are collected from projects that have used the transitional or “to be” process and compared with similar metrics that were collected for projects that were conducted under the “as is” process.
  - 정량적 지표 수집 (Quantitative metrics collection): "변경 전" 프로세스와 "변경 후" 프로세스를 사용하여 진행된 프로젝트에서 수집된 객관적인 측정 데이터를 비교 (예:결함 발생 건수, 개발 기간, 개발 비용 등을 비교)

Capability Security Model Integration (CMMI)

- The original Capability Maturity Model (CMM) was developed and upgraded by the Software Engineering Institute throughout the 1990s as a complete SPI framework.
- Today, it has evolved into the Capability Maturity Model Integration (CMMI)
  - a comprehensive process–meta model that is predicated on a set of system and software engineering capabilities that should be present as organizations reach different levels of process capability and maturity.
- process meta–model that is predicated on a set of system and software engineering capabilities that should be present in organizations at each maturity level:
  - As a “continuous” model.
  - As a “staged” model.
- Defines each process area in terms of “specific goals” and the “specific practices” required:
  - Specific goals establish the characteristics that must exist if the activities implied by a process area are to be effective.
  - Specific practices refine a goal into a set of process–related activities.

CMMI Process Levels [1]



- Level 0: Incomplete. 프로세스가 임시적임을 의미. 소수의 능력에 달려 있으며 사용 가능한 프로세스에 기반을 두지 않음
- Level 1: Performed. 구축된 제품과 제공되는 서비스가 대부분의 경우 잘 작동합니다. 그러나 예산과 일정은 항상 영향을 받고 문서화된 계획에서 벗어남.
- Level 2: Managed. 프로젝트가 조직에서 정의한 프로세스를 준수한다는 것을 의미.
  - 프로젝트와 해당 작업 결과물을 모니터링, 제어및 검토.
  - 프로젝트와 작업 결과물이 프로세스 설명을 준수하는지 평가.
  - 경영진은 정의된 간격(예: 주요 마일스톤 및 주요 작업 완료 시)으로 작업 산출물의 상태를 알고 있음.
  - 관련 이해관계자 간의 약속이 가시화됨

CMMI Process Levels 2

- Level 3: Defined.
  - 모든 역량 레벨 2 기준이 달성되었으며 프로세스는 조직의 지침에 따라 맞춤화.
  - 조직의 프로세스 자산에 프로세스 개선 정보를 제공.
- Level 4: Quantitatively managed.
  - 모든 역량 레벨 3 기준이 달성되었으며 프로세스 영역은 측정 및 정량적 평가를 사용하여 제어 및 개선
  - 품질 및 프로세스 성과에 대한 정량적 목표는 프로세스를 관리하는 데 사용.
- Level 5: Optimized.
  - 모든 역량 레벨 4 기준이 달성되었으며 프로세스 영역은 변화하는 고객 요구 사항을 충족하기 위해 정량적 수단을 사용하여 조정 및 최적화

Business Objectives	Measure	Min Value	Max Value	Unit of Measure	Indicator Definition
To Deliver On Time	Schedule Variance	-10%	10%	Percentage	Extent to which number of days the delivery was varied from the planned delivery date (against planned and revised schedule)
To Deliver On Time	Effort Variance	-15%	15%	Percentage	Extent to which actual effort (measured in person hours) deviates from planned effort for a planned release/test cycle (against planned and revised effort)

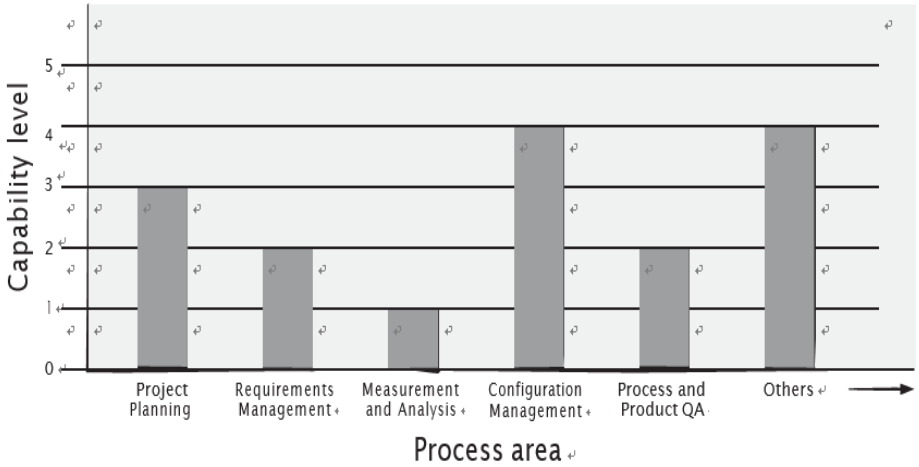
An example quantitative objective

CMMI Process Levels 2

Business Objectives	Measure	Min Value	Max Value	Unit of Measure	Indicator Definition
To Deliver On Time	Schedule Variance	-10%	10%	Percentage	Extent to which number of days the delivery was varied from the planned delivery date (against planned and revised schedule)
To Deliver On Time	Effort Variance	-15%	15%	Percentage	Extent to which actual effort (measured in person hours) deviates from planned effort for a planned release/test cycle (against planned and revised effort)

An example quantitative objective

- the measure schedule variance is −10% to 10%, and the measure effort variance is −15% to 15%



Process Area	Abbreviation	Category	Maturity Level
Causal Analysis and Resolution	CAR	Support	5
Configuration Management	CM	Support	2
Decision Analysis and Resolution	DAR	Support	3
Integrated Project Management	IPM	Project Management	3
Measurement and Analysis	MA	Support	2
Organizational Process Definition	OPD	Process Management	3
Organizational Process Focus	OPF	Process Management	3
Organizational Performance Management	OPM	Process Management	5
Organizational Process Performance	OPP	Process Management	4
Organizational Training	OT	Process Management	3
Product Integration	PI	Engineering	3
Project Monitoring and Control	PMC	Project Management	2
Project Planning	PP	Project Management	2
Process and Product Quality Assurance	PPQA	Support	2
Quantitative Project Management	QPM	Project Management	4
Requirements Development	RD	Engineering	3
Requirements Management	REQM	Project Management	2
Risk Management	RSKM	Project Management	3
Supplier Agreement Management	SAM	Project Management	2
Technical Solution	TS	Engineering	3
Validation	VAL	Engineering	3
Verification	VER	Engineering	3

People CMM

- Provides a roadmap for implementing workforce practices that continuously improve the capability of an organization's workforce. (조직 인력의 역량을 지속적으로 향상시키는 인력 관행을 구현하기 위한 로드맵을 제공)

- Defines a set of five organizational maturity levels that provide an indication of the relative sophistication of workforce practices and processes. (인력 관행 및 프로세스의 상대적 정교함을 나타내는 5가지 조직 성숙도 수준을 정의)

Other SPI Frameworks

- SPICE – an international initiative to support the International Standard ISO 15504-5:2015 and ISO 12207:2017

  SPICE provides a framework to assess a process and provide information on the strengths, weaknesses, and capabilities to help an organization achieve its goals.(https://en.wikipedia.org/wiki/ISO/IEC_15504, 프로세스 평가 (SPICE (소프트웨어 프로세스 개선 및 기능 결정) 라고도 함)는 컴퓨터 소프트웨어 개발 프로세스 및 관련 비즈니스 관리 기능 에 대한 기술 표준)

- TickIT – an auditing method that assesses an organization compliance to ISO Standard 9001:2015

  ISO 9001:2015 has adopted a "plan-do-check-act" cycle that is applied to the quality management elements of a software project. (소프트웨어 개발 및 컴퓨터 업계 기업을 위한 인증 프로그램, 소프트웨어 품질을 향상, https://en.wikipedia.org/wiki/TickIT)

SPI Return on Investment

- How do you know that we'll achieve a reasonable return for the money we're spending?

$$\text{ROI} = \frac{\text{S(benefits)} - \text{S(costs)}}{\text{S(costs)}} \times 100\%$$

- Benefits include the cost savings associated with higher product quality (fewer defects), less rework, reduced effort associated with changes, and the income that accrues from shorter time-to-market. (제품 품질 향상(결함 감소)과 관련된 비용 절감, 재작업 감소, 변경과 관련된 노력 감소, 출시 시간 단축으로 인한 수익 등이 포함)

- Costs include both direct SPI costs (for example, training, measurement) and indirect costs associated with greater emphasis on quality control and change management activities and more rigorous application of software engineering methods (for example, the creation of a design model).(직접적인 SPI 비용(예: 교육, 측정)과 품질 관리 및 변경 관리 활동에 대한 강조 및 소프트웨어 엔지니어링 방법의 보다 엄격한 적용(예: 설계 모델 생성)과 관련된 간접 비용이 모두 포함)

SPI Trends

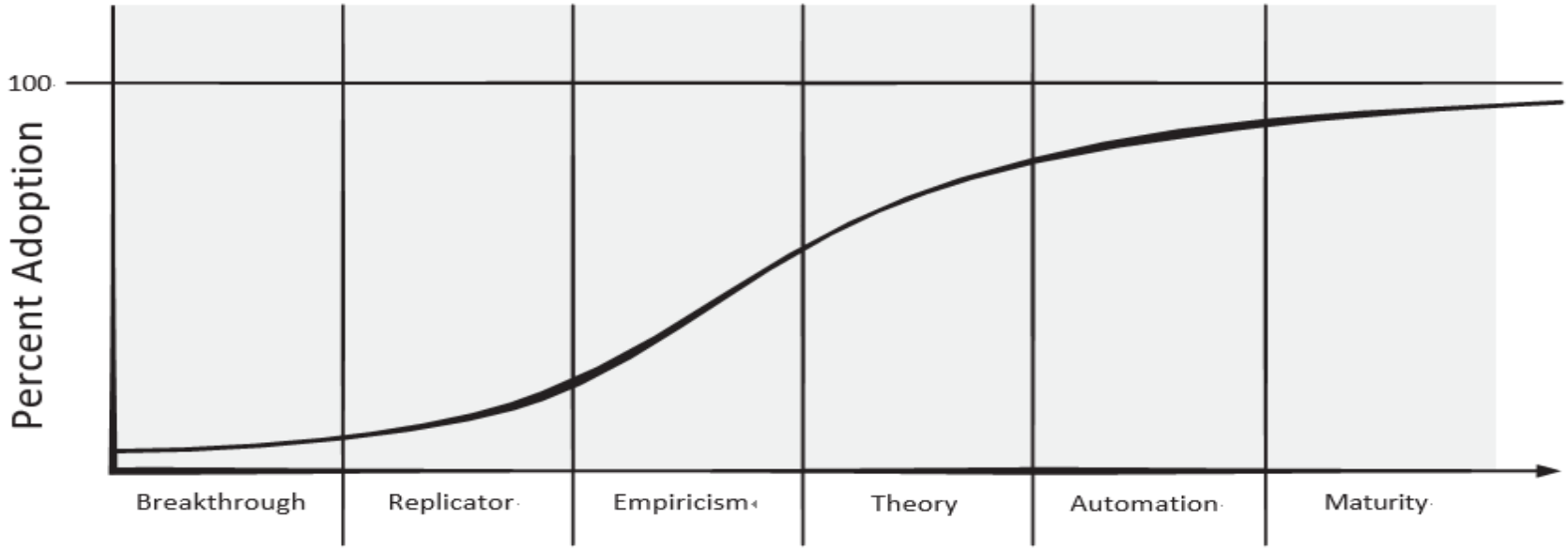
- 지난 35년 동안 많은 기업에서는 조직 변화와 기술 전환에 영향을 주기 위해 SPI 프레임워크를 적용하여 소프트웨어 엔지니어링 관행을 개선하려고 시도 , 절반 이상이 실패
- SEI CMMI와 같은 SPI 프레임워크의 일반적인 애플리케이션 비용이 1인당 $25,000~$70,000이며 완료하는 데 수년이 걸릴 수 있다고 보고
- 복잡한 프레임워크 모델이 더 간단한 모델로 대체될 수 있으며 수십 개의 핵심 사례와 수백 개의 보충 사례 대신 Agile SPI 프레임워크는 몇 가지 핵심 사례(예: 이 책 전체에서 논의된 프레임워크 활동과 유사)만 강조
- 교육과 훈련은 비용이 많이 들 수 있으므로 최소화(및 합리화) 하는 노력
- 조직 문화를 변화시키려는 광범위한 시도(모든 정치적 위험을 수반함)보다는 문화 변화가 현실 세계에서와 같이 전환점에 도달할 때까지 한 번에 한 소그룹씩 진행
- 모든 것과 마찬가지로 이러한 자산은 반복되는 교리가 되는 것이 아니라 더 우수하고 단순하며 민첩한 SPI 모델의 기초 역할

Emerging Trends in Software Engineering

Technology Evolution

- Challenges faced when trying to isolate technology trends:(기술 동향을 분리하려고 할 때 직면하는 과제)
  - *What Factors Determine the Success of a Trend*? (트렌드의 성공을 결정하는 요소)
  - *What Lifecycle Does a Trend Follow*? (추세는 어떤 수명주기)
  - *How Early Can a Successful Trend be Identified*? (성공적인 추세를 얼마나 일찍 식별)
  - *What Aspects of Evolution are Controllable*? (진화의 어떤 측면을 통제)
- Ray Kurzweil argues that technological evolution is similar to biological evolution but occurs at a rate that is orders of magnitude faster. (기술 진화가 생물학적 진화와 유사하지만 훨씬 더 빠른 속도로 발생한다고 주장)
  - Evolution (whether biological or technological) occurs as a result of positive feedback—"the more capable methods resulting from one stage of evolutionary progress are used to create the next stage." 진화적 진보의 한 단계에서 얻은 더 유능한 방법이 다음 단계를 만드는 데 사용

Technology Innovation Cycle



- A graph plots a technology innovation cycle.
- The graph is plotted for percent adoption versus 6 criteria.
- The curve begins at breakthrough at about 5 percent. 문제가 인식되고 실행 가능한 솔루션을 찾기 위한 반복적인 시도
- The next phase is replicator where the curve rises 20 percent. 초기 획기적인 작업은 복제기 단계에서 재현되어 더 널리 사용
- Next at empiricism the curve rise to 50 percent. 경험주의는 기술의 사용을 지배하는 경험적 규칙의 생성으로 이어짐
- Between theory and automation, the curve rises 90 percent. 자동화 단계에서 자동화된 도구의 생성으로 이어지는 더 광범위한 사용 이론으로 이어짐
- In maturity the curve plateau's at around 95 percent. 마지막으로 기술이 성숙해 널리 사용

Innovation Life Cycle

- 많은 연구 및 기술 동향이 결코 성숙해지지 않는다는 점에 유의
- 실제로 소프트웨어 엔지니어링 영역에서 "유망한" 기술의 대부분은 몇 년 동안 광범위한 관심을 받은 후 전용 지지자 집단에 의해 틈새 시장에 사용
  - 이러한 기술이 장점이 부족하다는 의미가 아니라 혁신 수명주기를 통한 여정이 길고 어렵다는 점.
- Kurzweil은 컴퓨팅 기술이 기술 형성 기간 동안 상대적으로 느린 성장을 보이고 성장 기간 동안 급속한 가속을 보인 후 기술이 한계에 도달함에 따라 평준화 기간을 보이는 "S-곡선"을 통해 진화한다는 데 동의
- 오늘날 우리는 현대 컴퓨팅 기술의 S자 곡선 무릎에 서 있음
- 초기 성장과 앞으로의 폭발적인 성장 사이의 전환기에 있음
- 향후 20~40년 동안 컴퓨팅 성능에 있어서 극적인(심지어 놀라운) 변화를 보게 될 것이라는 의미
- 20년 안에 기술 진화가 점점 더 빠른 속도로 가속화되어 궁극적으로 매혹적인 방식으로 인간 지능과 융합되고 확장되는 비생물학적 지능의 시대로 이어질 것이라고 제안
- 2040년에는 극단적인 계산, 인공 지능, 기계 학습, 나노 기술, 대규모 고대역폭 유비쿼터스 네트워크, 로봇공학의 조합이 우리를 다른 세상으로 이끌 것임
- 아마도 우리가 아직 이해할 수 없는 형태의 소프트웨어는 계속될 것이며 이 새로운 세계의 중심에 거주
- 소프트웨어 엔지니어링은 사라지지 않을 것입니다.

Observing Software Engineering Trends

- Barry Boehm suggests that:

  software engineers face the formidable challenges of dealing with rapid change, uncertainty and emergence, dependability, diversity, and interdependence, but they also have opportunities to make significant contributions that will make a difference for the better.

  소프트웨어 엔지니어는 급격한 변화, 불확실성 및 출현, 신뢰성, 다양성 및 상호 의존성을 처리하는 엄청난 과제에 직면. 그러나 그들은 또한 세상을 변화시킬 중요한 기여를 할 수 있는 기회도

- But what of more modest, short-term innovations, tools, and methods?
  (앞으로 몇 년 동안 이러한 과제에 직면할 수 있게 해주는 추세는 무엇?)

- Gartner Group – has developed a *hype cycle for emerging technologies*
  - "하이프 사이클"은 단기적인 기술 통합에 대한 현실적인 관점을 제시
  - 장기적인 추세는 기하급수적. 모든 소프트웨어 엔지니어링 기술이 과대광고 주기를 끝까지 통과하는 것은 아님.
  - 어떤 경우에는 환멸이 정당화되고 기술이 무명으로 전락하기도 합니다

Hype Cycle

- The graph plots the hype cycle.
- The graph is plotted for visibility versus five phases.

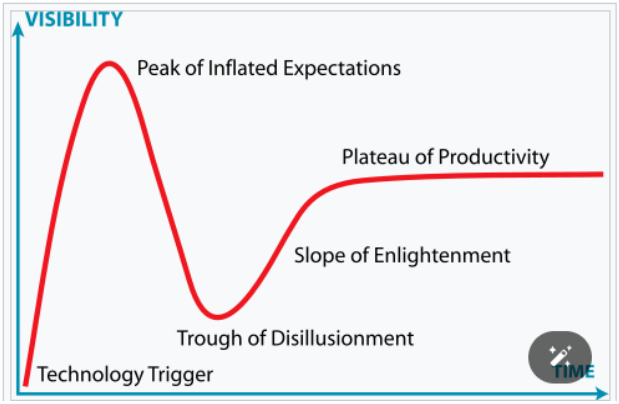
1. 기술 트리거 : 잠재적인 기술 혁신이 일을 시작. 초기 개념 증명 기사와 미디어의 관심은 상당한 홍보를 촉발합니다. 종종 사용 가능한 제품이 존재하지 않으며 상업적 생존 가능성도 입증되지 않음.

2. 부풀려진 기대의 정점 :초기 홍보는 수많은 성공 사례를 만들어내며 종종 수많은 실패를 동반합니다. 일부 회사는 조치. 대부분은 그렇지 않음.

3. 환멸의 골짜기 : 실험과 구현이 실패하면 관심이 약해짐. 기술 생산자는 흔들리거나 실패. 살아남은 공급자가 얼리 어답터가 만족할 만큼 제품을 개선하는 경우에만 투자가 계속됨.

4. 깨달음의 경사 :기술이 기업에 어떻게 도움이 될 수 있는지에 대한 더 많은 사례가 구체화되고 더 널리 이해. 2세대 및 3세대 제품은 기술 제공업체에서 등장. 더 많은 기업이 파일럿에 자금을 지원

5. 생산성의 고원 : 주류 채택이 시작. 제공자 생존 가능성을 평가하는 기준이 보다 명확하게 정의. 이 기술의 광범위한 시장 적용 가능성과 관련성은 확실히 성과를 거두고 있습니다. 기술이 틈새시장 이상의 영역을 갖고 있다면 계속해서 성장할 것입니다.



- https://en.wikipedia.org/wiki/Gartner_hype_cycle#Five_phases

Identifying Soft Trends

- Connectivity and collaboration has already led to a software teams that do not occupy the same physical space. 연결성과 협업(고대역폭 통신을 통해 가능)으로 인해 이미 동일한 물리적 공간을 차지하지 않는 소프트웨어 팀(현지 상황에서 재택근무 및 시간제 고용)이 발생
  - 소프트웨어 엔지니어링은 즉각적 요구를 충족할 수 있을 만큼 민첩하면서도 서로 다른 그룹을 조정할 수 있을 만큼 규율이 뛰어난 "분산된 글로벌 엔지니어링 팀"을 위한 포괄적인 프로세스 모델로 대응
  - Globalization – leads to a diverse workforce.
- 일부 세계 지역(예: 미국과 유럽)에서는 인구가 노령화되고 있음. 이러한 부인할 수 없는 인구통계학적(및 문화적 추세)은 경험이 풍부한 많은 소프트웨어 엔지니어와 관리자가 향후 수십 년 동안 현장을 떠날 것임을 의미.
  - 소프트웨어 엔지니어링 커뮤니티는 이러한 노령화된 관리자와 기술자의 지식을 포착하는 실행 가능한 메커니즘으로 대응해야 함(예: 패턴 사용(14장)은 올바른 방향으로 나아가는 단계임).
  - 그래야 미래 세대의 소프트웨어에서 사용할 수 있음
- 우리의 트렌드는 유행이 아님. 우리의 트렌드는 지속. 우리의 트렌드는 진화
  - 그들은 근본적인 힘, 첫 번째 원인, 기본적인 인간 요구, 태도, 열망을 나타냄
  - 그들은 우리가 세상을 탐색하고, 현재 일어나고 있는 일과 이유를 이해하고, 아직 다가올 일에 대비하는 데 도움을 줌

Managing Complexity

- 1982년에는 백만 줄의 소스 코드(LOC)를 포함하는 메인프레임 기반 시스템은 상당히 큰 것으로 간주
- 오늘날 소형 디지털 장치에는 일반적으로 운영 체제 기능을 위한 수백만 개의 LOC와 함께 60,000~200,000줄의 맞춤형 소프트웨어 라인이 포함되어 있으며 1,000만 ~ 5,000만 줄의 코드를 포함하는 최신 컴퓨터 기반 시스템이 일반적
- 외부 세계, 다른 상호 운용 가능한 시스템, 인터넷(또는 그 후속 제품) 및 이 컴퓨팅 괴물이 성공적으로 작동하기위해 수백만 개의 내부 구성 요소에 대한 10억 개의 LOC 시스템 인터페이스
  - 이러한 모든 연결이 정보의 올바른 흐름을 보장하는 신뢰할 수 있는 방법이 있나?.
  - 작업 흐름을 관리하고 진행 상황을 추적하는 방법은 무엇?
- 불일치, 모호함, 누락, 명백한 오류를 발견하고 수정하는 방식으로 수만 가지 요구 사항, 제약 조건 및 제한 사항을 어떻게 분석할 수 있나?
- 이 규모의 시스템을 처리할 수 있을 만큼 견고한 설계 아키텍처를 어떻게 만들 수 있나?
- 어떻게 의미 있는 방식으로 검증과 검증을 수행할 수 있나?
- 초기에 소프트웨어 엔지니어들은 임시적인 방식으로만 복잡성을 관리하려고 시도. 오늘날 우리는 복잡성을 통제하기 위해 프로세스, 방법 및 도구를 사용. 하지만 우리의 현재 접근 방식이 내일의 작업에 적합한가? 데이터 과학 기술과 인공 지능 기술의 광범위한 사용을 통하여 복잡성 관리

Open-World Software

- Open-world software
  - software that is designed to adapt to a continually changing environment ‘by self-organizing its structure and self-adapting its behavior.” (구조를 자체 구성하고 동작을 자체 적응함으로써' 지속적으로 변화하는 환경에 적응하도록 설계된 소프트웨어)
- Concepts such as *ambient intelligence, context-aware applications,* and *pervasive/ubiquitous computing*
  - all focus on integrating software-based systems into an environment far broader than anything to date.(모두 소프트웨어 기반 시스템을 지금까지의 그 어떤 것보다 훨씬 더 광범위한 환경에 통합하는 데 중점을 둠)
- Engineering of *variability intensive systems* focuses on systems that can be highly variable during all software engineering activities and we need to improve our understanding of how to design and manage them in a cost-effective manner. (모든 소프트웨어 엔지니어링 활동 중에 매우 가변적일 수 있는 시스템에 중점을 두고 있으며 비용 효율적인 방식으로 시스템을 설계하고 관리)

Emergent Requirements

- As systems become more complex requirements will emerge as everyone involved in the engineering and construction of a complex system learns more about the systems, the environment in which it resides, and the users who will interact with it. (복잡한 시스템의 엔지니어링 및 구축에 관련된 모든 사람, 상호 작용할 사용자에 대해 더 많이 배우게 되면서 요구 사항이 등장)
  - 오픈 월드 시스템의 출현으로 긴급 요구 사항(및 거의 연속적인 변경)이 표준이 될 수 있음
  - 사용자 시나리오(예: 사용 사례) 생성으로 시작
    - 이해관계자에 대한 향상된 지식획득 및 공유
    - 반복에 대한 강조
    - 의사소통 및 조정

- This implies several of software engineering trends:
  - Process models must be designed to embrace change and adopt the basic tenets of the agile philosophy. (프로세스 모델은 변화를 수용하고 애자일 철학의 기본 원칙을 채택하도록 설계)
  - Methods that yield engineering models (for example, requirements and design models) must be used judiciously because those models will change repeatedly as more knowledge about the system is acquired.(엔지니어링 모델(예: 요구 사항 및 설계 모델)을 생성하는 방법은 시스템에 대한 더 많은 지식이 획득됨에 따라 해당 모델이 반복적으로 변경, 신중하게 사용)
  - Tools that support both process and methods must make adaptation and change easy.(프로세스와 방법을 모두 지원하는 도구는 적응과 변경을 쉽게 만들어주어야 함)

Model-Driven Software Development

- Model-drive software development
  - couples domain specific modeling languages with transformation engines and generators in a way that facilitates the representation of abstraction at high levels and then transforms it into lower levels.(높은 수준에서 추상화 표현을 용이하게 한 다음 이를 낮은 수준으로 변환하는 방식으로 도메인별 모델링 언어를 변환 엔진 및 생성기와 결합)

- Domain-specific modeling languages (DSMLs) represent application structure, behavior, and requirements within a particular application domains are described with meta-models. (특정 애플리케이션 도메인 내의 애플리케이션 구조, 동작 및 요구 사항을 메타 모델로 설명)
- These DSML meta-models are tuned to design concepts inherent in the application domain and can therefore represent relationships and constraints among design elements in an efficient manner. (DSML 메타 모델은 응용 프로그램 도메인에 내재된 설계 개념에 맞게 조정되므로 효율적인 방식으로 설계 요소 간의 관계와 제약 조건을 나타낼 수 있음)

Concluding Comments

Importance of Software

- The functionality delivered by software differentiates products, systems, and services and provides competitive advantage in the marketplace. (소프트웨어가 제공하는 기능은 제품, 시스템 및 서비스를 차별화하고 시장에서 경쟁 우위를 제공)
- The programs, documents, and data that are software help to generate the most important commodity that any individual, business, or government can acquire—information. (프로그램, 문서 및 데이터는 개인, 기업 또는 정부가 얻을 수 있는 가장 중요한 상품인 정보를 생성하는 데 도움)
- The growing pervasiveness of computer software will present dramatic challenges for society as a whole. (소프트웨어의 확산이 점점 더 확산되면서 사회 전체에 극적인 도전이 제시)
- Whenever a technology has broad impact (to help or harm) it must be handled with care. (기술이 광범위한 영향(도움 또는 해로움)을 미칠 때마다 조심스럽게 다루어야 함)

The Way People Build Systems

- As program sizes continue to increase team sizes will increase as well. (프로그램 규모가 계속 증가함에 따라 팀 규모도 증가)
- Experience indicates that as the number of people on a project team increases, the overall productivity of the group will suffer because of communications issues.(프로젝트 팀의 인원 수가 증가하면 커뮤니케이션 문제로 인해 그룹의 전반적인 생산성이 저하)
- Too much time is spent sharing small amounts of information among teams and team members, and important information may falls through the cracks. (소량의 정보를 공유하는 데 너무 많은 시간이 소요되며, 중요한 정보가 누락)
- Software engineers must plan for radical changes in the ways individuals and teams communicate with one another (for example, uses of social media and cloud services). (개인과 팀이 서로 통신하는 방식(예: 소셜 미디어 및 클라우드 서비스 사용)의 급격한 변화에 대비)

Knowledge Transfer

- Communication is the transfer of knowledge, and the acquisition (and transfer) of knowledge is changing in profound ways. (의사소통은 지식의 전달이며, 지식의 획득(및 전달)은 심오한 방식으로 변화)
- Sophisticated search engines may allow social networking and crowd sourcing to morph into serious developer tools. (정교한 검색 엔진을 사용하면 소셜 네트워킹과 크라우드 소싱이 심각한 개발자 도구로 변모)
- As mobile applications provide better for better developer synergy, the speed and quality of knowledge transfer will grow exponentially. (모바일 애플리케이션이 더 나은 개발자 시너지를 위해 더 나은 기능을 제공함에 따라 지식 전달의 속도와 품질은 기하급수적으로 향상)
- People themselves are likely to be slow to embrace the changes needed for a new software development culture. (새로운 소프트웨어 개발 문화에 필요한 변화를 느리게 받아들이는 경향)

Software Engineering Ethical Principles

소프트웨어 엔지니어는 소프트웨어의 분석, 사양, 설계, 개발, 테스트 및 유지 관리를 유익하고 존경받는 직업으로 만들기 위해 최선

1. PUBLIC – SE's shall act consistently with the public interest. (소프트웨어 엔지니어는 공익에 부합하도록 행동)
2. CLIENT AND EMPLOYER – SE's shall act in a manner that in best interests of their client and employer consistent with the public interest. (소프트웨어 엔지니어는 공익과 일관되게 고객 및 고용주의 이익을 극대화하는 방식)
3. PRODUCT – SE's shall ensure that their products and related modifications meet the highest professional standards possible. (제품 및 관련 수정 사항이 가능한 최고의 전문 표준을 충족하는지 확인)
4. JUDGMENT – SE's shall maintain integrity and independence in their professional judgment. (소프트웨어 엔지니어는 전문적인 판단에 있어 무결성과 독립성을 유지)
5. MANAGEMENT – SE managers and shall promote an ethical approach to management of software development and maintenance. (소프트웨어 개발 및 유지 관리에 대한 윤리적 접근 방식)
6. PROFESSION – SE's shall advance the integrity and reputation of the profession consistent with the public interest. (공익에 부합하는 직업의 진실성과 명성을 높여야 함)
7. COLLEAGUES – SE's shall be fair and supportive of colleagues. (동료를 공정하게 대하고 지원)
8. SELF – SE's shall participate in lifelong learning regarding the practice of their profession and promote an ethical approach to the practice of the profession. (자신의 직업 수행과 관련하여 평생 학습에 참여해야 하며 직업 수행에 대한 윤리적 접근 방식을 장려)

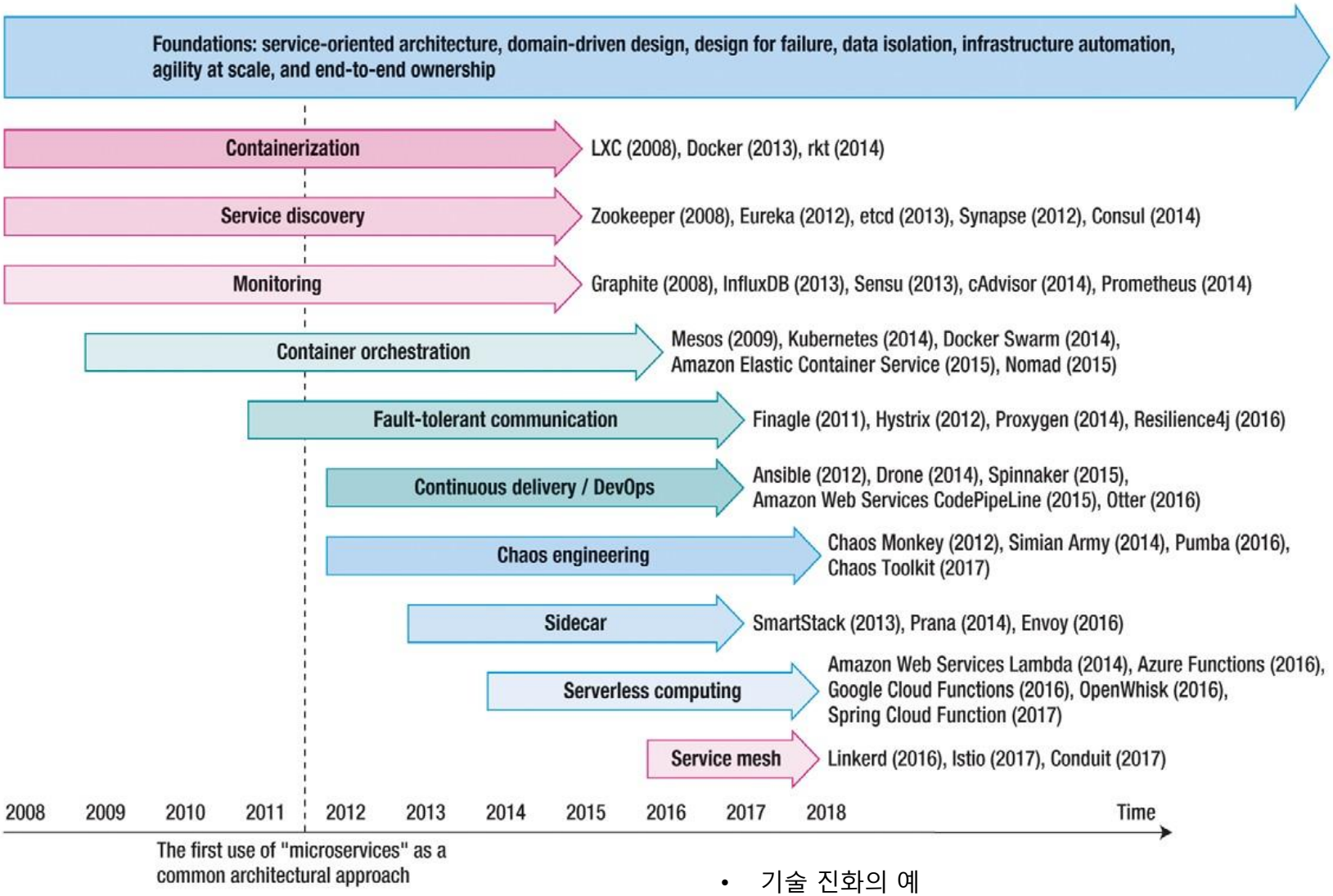
미래의 역량

- SW 개발자로서 10-20년 이상을 어떻게 살아 남을 수 있을까?
  - SW 2.0, Low-code/No-Code platform등과 같은 새로운 솔루션, 기법들이 나의 미래를 위협하고 있지는 않나?

- 환경 변화에 적응이 필요
  - Industrial revolution (4,5,6th,…), 짧은 시간에 이루어 질 것이다
  - Homohundred : 사람이 백세까지 살아간다.

- Technology evolution에 대한 심층적인 이해가 중요
  - 패러다임의 변화를 이해하고, 향후의 추세에 대한 예측 필요
  - 변하는 기술 자체보다는 이러한 진화의 근본(원인, 필요성)에 대한 이해(How?)

미래의 역량



- 기술 진화의 예
  - Microservice (출처: IEEE Software, May/June 2018)

미래의 역량

- **차별화된 역량 필요**
  - Commodity:
    - Open source, github, stackoverflow,..
  - Value-added: 품질 차별화:
    - Safe, Secure, Sustainable,..
    - Better, Faster, Cheaper
  - Innovative:
    - New business idea
    - Story telling, …
    - Creativity

END