Final

Chapter 8

Program testing goals

- To demonstrate - meets its requirements

Validation and defect testing(검증 및 결함)

- Validation testing - meets its requirements(요구사항 만족하는지)
- Defect testing - discover faults or defects(결함 버그 찾기)

Verification vs Validation

- Verification - building product right(명세서)
- Validation - building right product(유저가 필요로 하는지)

Inspections and testing

- Inspections: static verification, 소스코드 확인
  - 구현 전에 가능
  - any representation에 적용 가능(requirements, design, …
  - error에 가려진 error 찾을 수 있음
  - Incomplete versions can be inspected(미완성 버전도 인스펙션 가능)
  - Cannot check non-functional(퍼포먼스 같은 비기능적 체크 못함)
- Testing: dynamic verification, 실행해서 확인
- Both should be used during V&V(V&V에 Inspection and testing 다 써야)

Stages of testing

- Development testing
  - Unit testing: object, methods,
    - Automated Unit test 가급적 해야, [Setup, Call, Assertion]

        - Partition testing: common characteristics, equivalence partition or domain
        - Guideline-based testing: test case: chosen from each partition, force error, buffers to overflow, invalid output, … 파티션 나눈대로 가이드라인 tst
    - Component testing
        - Interface: behaves according to its specification(요구사항에 맞게 인터페이스 작동하는지)
            - Parameter interface
            - Shared memory interfaces
            - Procedural interfaces
            - Message passing interfaces
            - Errors: misuse, misunderstanding, timing errors
    - System testing
    - forces these interactions to occur: 오류 일부러 만들어보기
- Release testing: separate testing team before release to user
- User testing

Test-driven development

- Interleave testing and code development(테스트와 코드개발 섞어가며)
- Develop code incrementally, dont move until pass(코드 한번 완성되면 다음코드)
- Can be used in Agile, Plan-driven(애자일, 플랜드리븐 둘 다 적용 가능)
- Start: identifying what required
- Automated test, new test will fail
- Benefits:
    - Code coverage
    - Regression testing: (재귀 테스트: A, AB, ABC.. 예전 코드를 망가트리는지 테스트가능)

test changes have not broken previous work code, automated라서 비싸지않음

- Simplified debugging
- System documentation(문서화 쉬움)

Release testing

- Outside development team(타 팀이 진행)
- to convince system is good enough for use(충분히 쓸만한지)
- Black-box testing process
- Defect testing, Validation testing(버그와, 충분히 쓸만한지 확인)
- Performance testing
  - Part of release testing
  - Performance test: load is steadily increased, until sys performance unacceptable
  - Stress test: deliberately overloaded(의도적 과부하) to test its failure

User testing

- Essential, even when Comprehensive system carried out
- Alpha test: with development team
- Beta test: with user and dev team
- Acceptance: for custom system, user test

Stages in the acceptance testing process

Define criteria, Plan testing, Derive tests, Run tests, Negotiate result, Reject/Accept

DPDRNR

Key points:

- Dev test only show the “presence of errors”, cannot demonstrate that no fault

Chapter 9 - SW Evolution

- SW Change is inevitable
- Majority of budget - devoted to changing and evolving rather than new

Spiral model(나선형 무한회전) (SIVO)

- Specification
- Implemention
- Validation
- Operation
- Evolution and servicing
  - Development - Evolution - Servicing - Retirement(개발 진화 서비스 은퇴)
  - Evolution: 여전히 기능 추가도 하고, 서비스 중에 있음
  - Servicing: 기능추가는 없고 버그수정 정도만
  - Phase-out: 단계적 폐지

Evolution processes:

- Change identification and evolution continues(시스템 수명동안 지속)
- SW type, Development Process, Skills and experiance of 개발자 에 따라 달라진다.

Change implementation:(구현 변경)

- Proposed changes → Req analysis → Req updating → SW Development
- original team 아닐 경우 프로그램 이해가 우선

Urgent change requests:

- without going all stages, 걍 바로 ㄱ

Agile and evolution:

- Automated regression testing: 유용함
- 애자일과 진화는 거의 하나 수준, 애자일이 곧 진화

Handover Problem

- 개발은 애자일, 유지보수 플랜 → 문서가 없음
- 개발은 플랜, 유지보수 애자일 → 자동회 테스트 툴 없음

Legacy system components

- System hardware Legacy
- Support software
- Application software
- Application data

Legacy system replacement

- 시스템 specification 사라졌거나
- 시스템과 비즈니스 프로세스 타이트하게 통합됐거나
- 레거시 시스템에 포함된 문서화되지 않은 규칙들
- 새로운 시스템이 딜레이되거나, 예산초과 일수도

Legacy change 비싼 이유

- 일관성 없는 프로그래밍 스타일
- 구식 언어라 개발자 없음
- 시스템 문서 부실
- 시스템 구조 이미 저하됨
- 하도 패치해서 최적화 이해 어렵
- 데이터 오류, 중복, 불일치

Legacy 처리 방법

- Scrap: 폐기,
- Continue: 유지
- Transform: 유지보수성 개선, 변형
- Replace: 새로운 시스템으로 대체

Low qual, Low value: Scrap

Low qual, high val: re-engineer or replace

Higu qual, low val: scrap or maintain

High qual, high val: maintain(continue)

Interoperability: 다른 시스템과 상호작용 하는데 문제 없는가

Configuration management: 레거시 유지하면서 버전관리 잘 되었는지

System measurement:

- number of sys change requests: 높을수록 구림
- number of diff user interfaces used: 많을수록 불안
- volume of data: 많을수록 불안

Software maintenance

Type of maintenance:

- Fault repairs: 문제 수정
- Environmental adaption: 플랫폼이나 환경에 맞추기
- Functionality addition and modification: 새로운 기능이나 요구사항 추가

Maintenance cost:

- 유지보수 할 수록 cost 올라감, 시스템이 오래될수록 유지보수 비용 높아짐
- 레거시 유지보다 새로운거 만드는게 쌀수도

Software reengineering

- Refactoring:
  - 리엔지니어링과 다름
  - Preventative maintenance, 미래 문제 생길 확률 줄이도록
  - Continuous 한 작업임, 계속해서 하는거임
  - 구조 및 코드 저하 방지
- Reengineering:
  - 유지보수 쉽도록 함
  - 리스크, 비용 축소. 하지만 항상 축소되는건 아님, 더 들수도
  - 레가시를 이용해서 유지보수 가능한 새로운 시스템 만드는 것, 몇년에 한번이라던지
  - Source code translation
  - Reverse engineering

- Program structure improvement
- Program modularisation
- Data reengineering

Chapter 10 - Dependable systems

Dependability - {1) reliability, 2) availability, 3) security}

이 세 가지는 서로 의존적임(관계가 있음)

Causes of failure:

1) HW, 2) SW, 3) Operational failure(사람이 잘못한 것)

- Availability: 시스템이 잘 돌아가고 있을 확률
- Reliability: 사용자가 원하는대로 정확히 서비스해줄 확률
- Safety: 주변 환경에 데미지를 입히지 않음
- Security: 외부 공격으로부터 의도적 접근을 방어할 수 있냐
- Resilience: 크리티컬한 문제가 생겨도 계속해서 서비스해줄수 있냐

Dependibility Cost:

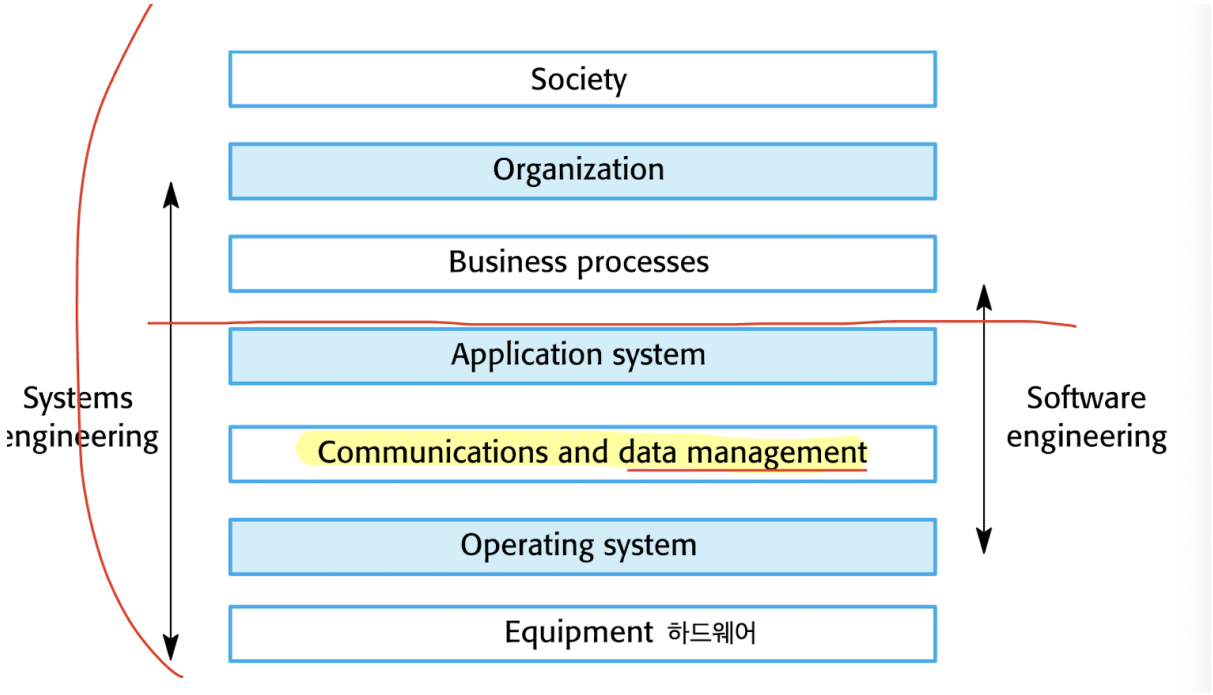
- exponentially 증가,
- 하드웨어 비용
- 고도화된 테스팅에 따른 인건비

May be 덜 신뢰도있는 것 쓰고 실패해서 돈내는게 나을수도

Sociotechnical systems

Software engineering is not isolated activity

사람, 사회조직 등과 연관되어 관계, 서로 연관성 있다



Regulation and compliance: 규정 및 준수

핵, 공기질, 메디컬 등은 Regulator에 의해 approved 되야 한다.

Redundancy and diversity

- Redundancy: 중복성, 중요한 컴포넌트의 경우 문제 생겼을 때 대체 가능하도록 여러개 만들어 두는 것, E커머스의 백업 서버
- Diversity: 같은 기능을 다른 방식, 다른 컴포넌트로 제공하는 것. 문제가 발생해도 같은 방식으로 망가지지 않도록, 윈도우와 리눅스를 사용한다던지.
- 이 두가지 사용하면 비용 올라가는데, Resilience 올려줌
- 이 둘은 Verification and validation 중요하다
- System complexity를 증가시킨다
- Unanticipated interactions and dependencies로 에러를 만들 수 있다. (중복된 컴포넌트 간 상호작용 및 의존성으로 인해)

Chapter 21- Real-time SW Engineering

Responsiveness

- Real-time에서, 반응성은 critical
- non-real-time에서, correctness는 정확한 output

- real-time에서, correctness는 정확한 output + 걸리는 시간 까지 만족해야

Real-time system

- Result와 Time까지 정확히 작동해야 하는 시스템
- Soft real-time: 시간에 문제 생길경우 품질이 Degraded
- Hard real-time: 시간에 문제 생길경우 결과 자체가 Incorrect

Embedded system design

- HW는 수정 불가능하기 때문에, 어떤 타이밍에 무슨 서비스 할지 미리 생각하고 HW SW 구현

Reactive systems

- Stimulus에 대응하여 작동하는 Real-time system
- Periodic stimuli: 주기적으로 정해진 시간에 예측 가능하게 자극이 발생(온도센서 등)
  Predictable time intervals
- Aperiodic stimuli: 비주기적, 침입이 감지되면 알람이 온다던지
  Irregularly and Unpredictably
- 이러한 Stimuli들은 inturrupt를 발생시킬 수 있다
- System architecture는 stimulus handlers 사이 fast switching을 허용해야
- Architecture에 Sequential loop은 적합하지 않다
- Stimuli가 다르면 타이밍 요구사항이 다르기 때문에 우선순위 고려하여 아키텍처 만들어야
  (더 중요한 자극이 들어오면 먼저 처리)
- System Elements
  - Sensor control processes
  - Data processor
  - Actuator control processes

Process coordination

- Process들은 coordinated(공동 작업)하고, share information
- coordination mechanism은 mutual exclusion(상호배제)를 보장해야 함

- 한 쪽이 shared resource를 수정하고 있으면, 다른 process는 그 리소스를 수정할 수 없음
- Process 사이 exchange를 고려할 때, 프로세스 속도가 다른걸 고려해야

Mutual exclusion

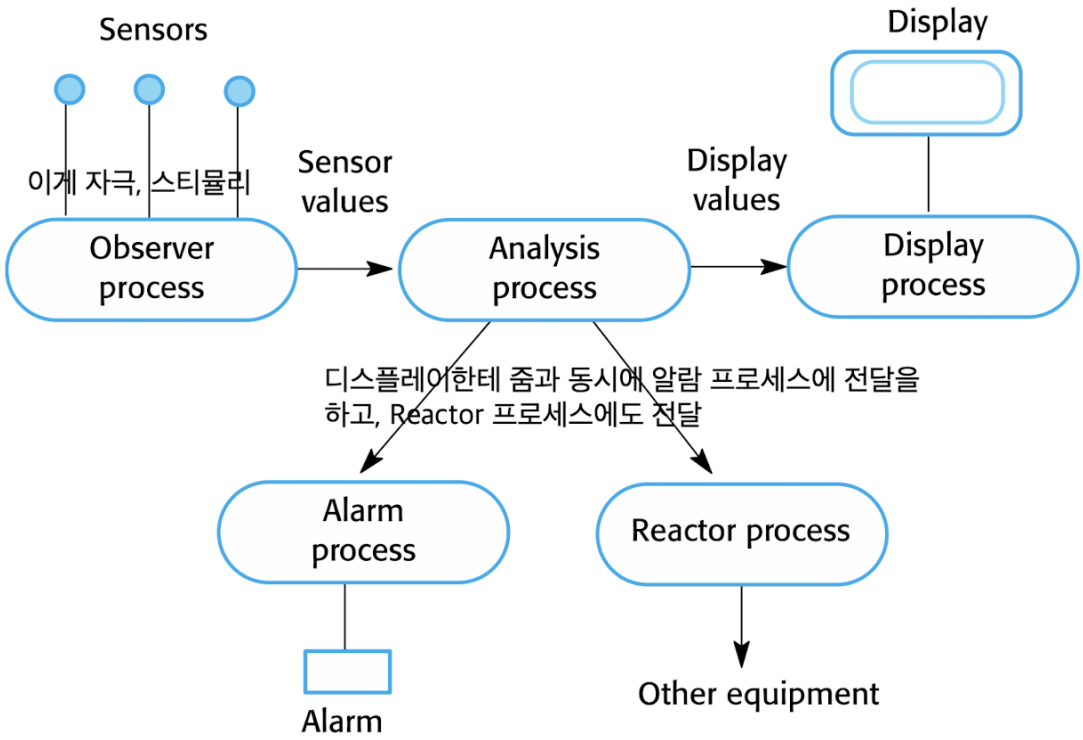
- 버퍼 데이터 생성기와 데이터 소모기는 같은 원소에 동시 접근하면 안됨
- 버퍼가 가득차면 프로듀서 멈추고, 텅 비면 컨슈머 멈춰야함

Real time programming

- C같은 로우레벨 씀, 자바는 느려서 안씀, overhead 일으킴

Architectural patterns for real-time software

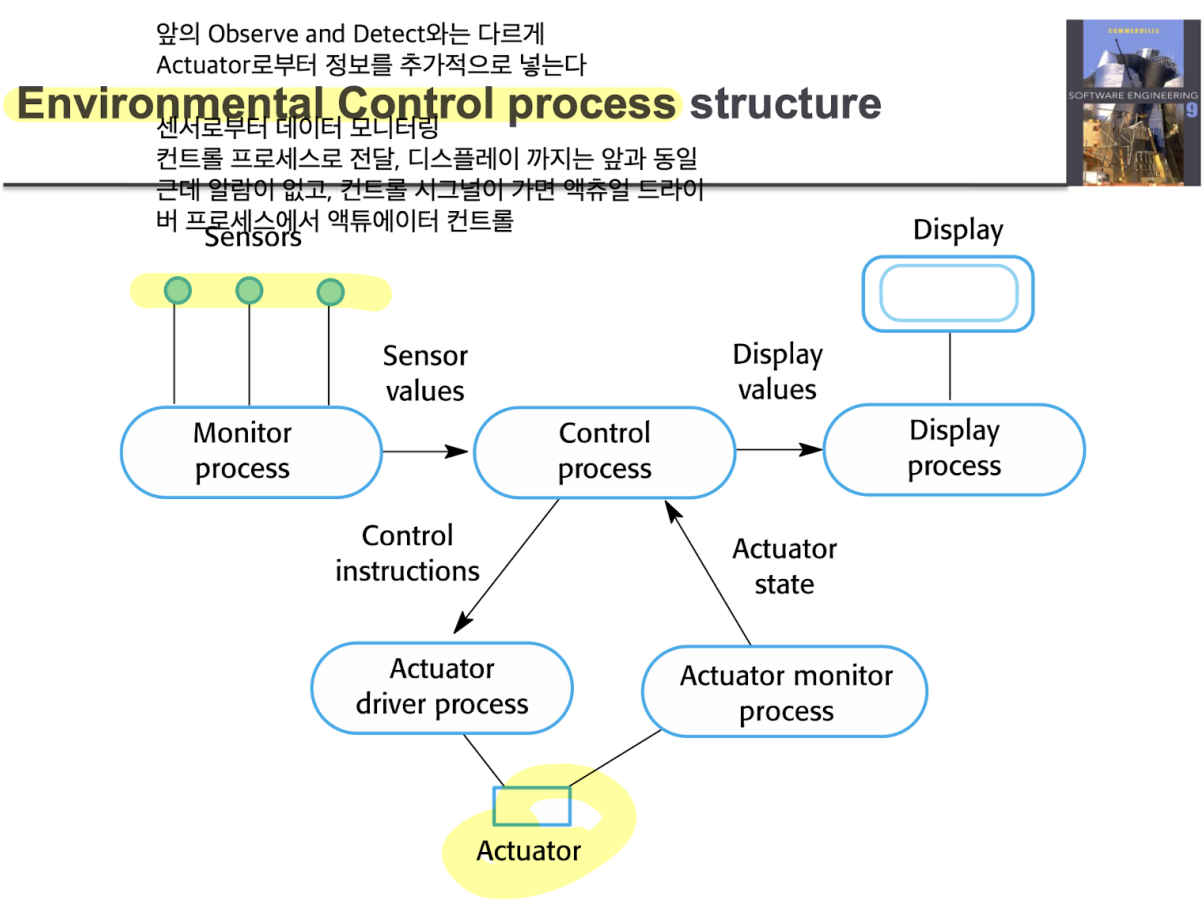
Observe and react:



/2014 Chapter 21. Real-time Software Engineering

- 여러개의 센서가 주기적으로 모니터되고 디스플레이 되는 상황에서 씀

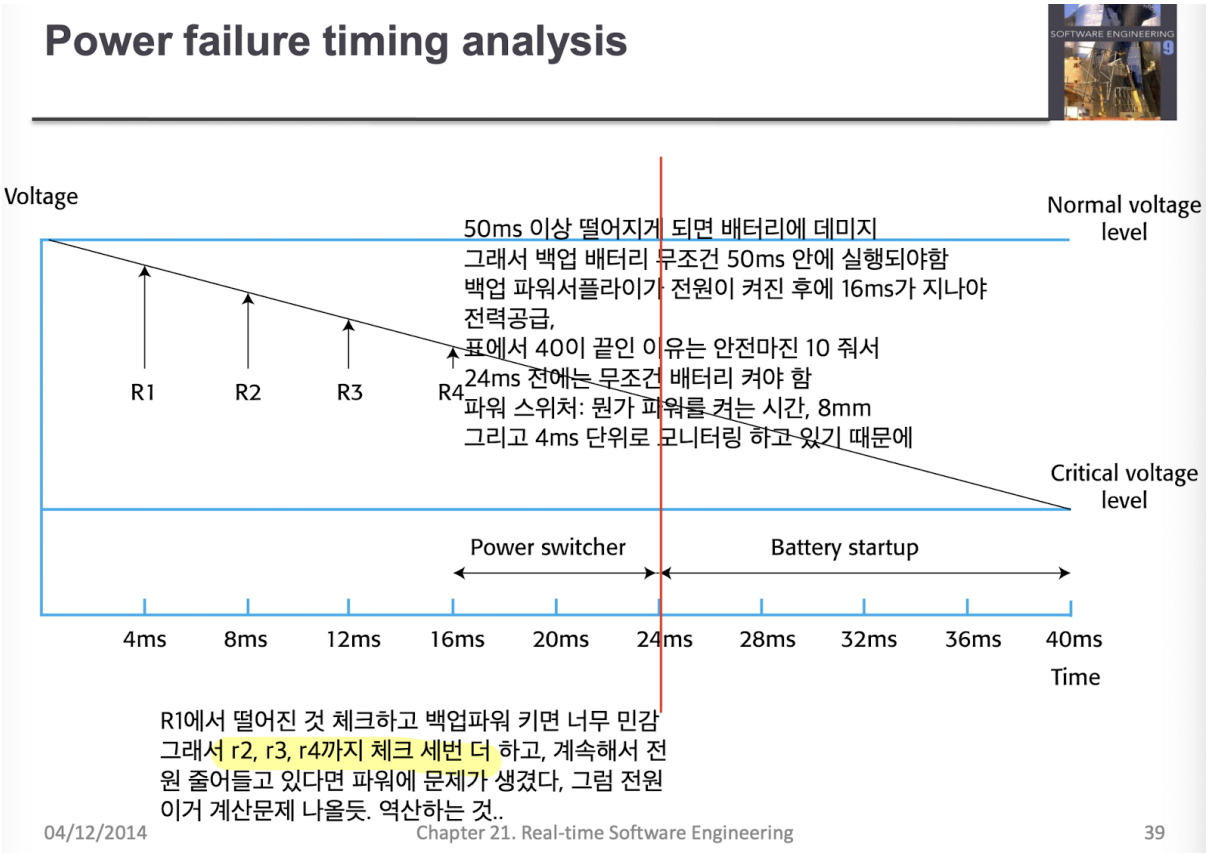
Environmental Control:



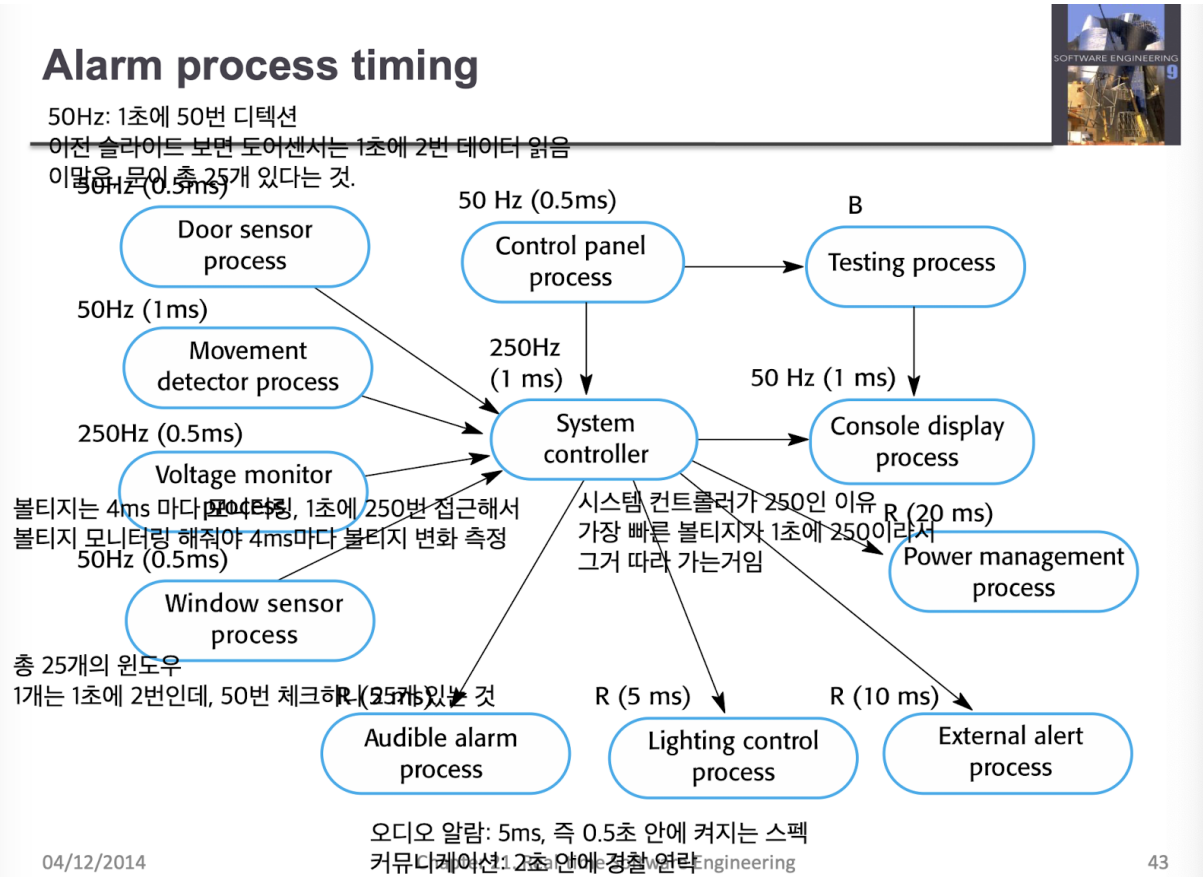
- 시스템이 센서를 가지고 있고, 환경 정보가 주어질 때, Actuator로 환경을 변화시키는 상황에서 씀(자동차 ABS)

Timing analysis

- Real-time system correctness는 시간, 시간 만족시키기 위해
- Factors
  - Deadlines: Stimuli가 들어왔을 때, 몇 초 안에 처리되야 하는지
  - Frequency: 1초동안 프로세스가 몇 번 실행되야 데드라인을 만족시킬 수 있는지
  - Execution time: Stimuli를 처리하고, Response를 만드는데까지 걸리는 시간



Worst-case execution time이 8ms(switcher)



Real-time OS

- Process management, Resource allocation
- May be based on standard kernal
- 보통 do not include file management
- Components:
  - Real-time clock
  - Interrupt handler
  - Scheduler
  - Resource manager
  - Dispatcher
- Non-stop Components:
  - Configuration manager: dynamic reconfiguration
    동적으로 돌아가면서도 수리 할 수 있도록 해주는 매니저(백그라운드 업데이트 등)

    - Fault manager:
      SW, HW에 faults 발생 시 detect하고, 적절한 액션 취함(백업 디스크 스위칭 등)

Process management

- Concurrent(동시 실행)하는 process 어떻게 관리할건지
- Periodic process: pre-specified(미리 정해진) 인터벌에 따라 실행됨
- Real-time clock 사용
    - Process period: execution 사이의 시간
    - Process deadline: processing 완성되는 시간
- Interrupt level priority: 최상위 우선순위
- Clock level priority: Periodic하게 돌아가는 process allocation

Chapter 22- Project Management

- Accordance: 화합 유지
- Budget and schedule: 이것 지키는게 PM이 필요한 이유
- Maintain a cohorent and well-functioning dev team: 일관성 있게 팀 유지
- SW Product: intangible(형태가 없다)

Management Activities

- Project planning: estimate and scheduling project(예측하고 계획세우기)
- Risk management: 리스크를 assess(평가), monitor(관측), take action(대응)
- People management: 인력 관리
- Reporting: 상사에 보고
- Proposal writing: 제안서 써서 프로젝트 따오기

Risk Management

- Identifying risks(리스크 식별), Minimise their effect(영향 최소화)
- SW 자체로 Inherent Uncertainties(내재된 불확실성)이 있음, 그래서 중요
- Anticipate risks(예측), Understand the impact(영향 이해), Avoid risks(피하기)

Risk classification

- Project risks → schedule or resources
- Product risks → quality or performance
- Business risks → developing or procuring software

Risk management process(식별 분석 계획 모니터, iapm)

- Risk identification: 뭐가 리스크인지 분석
- Risk analysis: 리스크의 결과를 예측
- Risk planning: avoid or minimise the effect(피하거나 영향 최소화)
- Risk monitoring: 관찰(프로젝트 내내)

Risk analysis

- Probability: very low - low - moderate - high - very high
- Risk consequence: catastrophic - serious - tolerable - insignificant

Risk planning

- Avoidance strategies: 발생 확률을 줄임
- Minimization strategies: 리스크 발생 시 임팩트 최소화
- Contigency plans: 만일의 사태 어떻게 대처할지

Risk monitoring

- Regularly(계속해서) Assess each identified risk less or more probable(평가해줘야 함, 확률을)

People management

- Consistency: 다 동등하게 대우
- Respect:
- Inclusion
- Honesty

Personality type

- Task-oriented: 일중독자

- Interaction-oriented: 친목왕
- Self-oriented: 자기 잘나가는게 주목적

Teamwork

- cohesive(화합)이 중요하다

Chapter 23 - Project planning

Project planning

- Work를 Breaking down, 쪼개서 팀에게 할당해준다. 문제를 예측하고, 잠정적인 솔루션을 준비한다.

Planning stages(물론 플래닝은 계속해야함)

- Proposal stage: 입찰용 견적서 제안서
- Start up: who will work, how project broken down into increments, how resource
- Project 도중 주기적(Periodically)으로 Plan modif 해줘야함

Proposal planning:

- 가격 견적서 제안하기 위해서 필요
- Pricing에는 인건비, 장비비, SW비용 다 포함

Proj startup planning

- Agile development에도 필요함.
- System requirements에 대해 더 이해하나, design이나 구현 정보는 없을 때
- Monitoring mechanism도 만들어둬야 함

Development Planning

- 도중, 가격예측과 리스크도 변동된다

SW Pricing

Under pricing: 계약을 따내기 위해서 낮게 부른다.

Increased pricing: 고정계약일 경우, 리스크 때문에 높게 부른다.

Plan-driven development

장점: 프로젝트 진행 도중 발생할 Potential problems, dependencies을 미리 확인 가능

단점: early decisions have to be revised(어차피 먼저 결정해봐야 나중에 상황은바뀜)

- Project Planning은 Iterative process임, 프로젝트 진행 내내 반복되는

Planning assumptions

- 좀 Realistic하게 계획을 세워야한다
- 문제는 어차피 발생하고 딜레이는 …
- Contingency: 만일의 사태, 이 역시 플랜에 고려해야한다

Risk mitigation(완화)

- 심각한 문제가 생길 것 같으면, Risk mitigate actions을 취해라, 망할 확률 줄이고싶으면.
- 이러한 액션들의 Conjunctions(r결합)을 위해서, 플랜을 다시 짜야한다.
- Project constraints, deliverable 들을 다시 재고려해볼 수도 있다.

Project Scheduling

- 일을 아주 잘게 쪼개고, 언제 어떻게 이게 실행될건지 계획 세움
- 누가 할건지도 정해야 함
- 각 작업마다 리소르를 얼마나 쓸지 예측해야 함
- Activities:
  - Split 하고, task 별 시간과 자원 예측
  - Organize tasks concurrently: 동시에 최적화되게
  - Minimize task dependencies: 작업 간 의존성 최소화하기
  - Dependent on PM’s intuition and experience: PM의 직관과 경험에 달렸다.

Scheduling problems

- 사람 수이 비례해서(proportional) 생산성 올라가는거 아님
- 추가 인력 투입은, 오히려 프로젝트 지연시킬 수 있음.

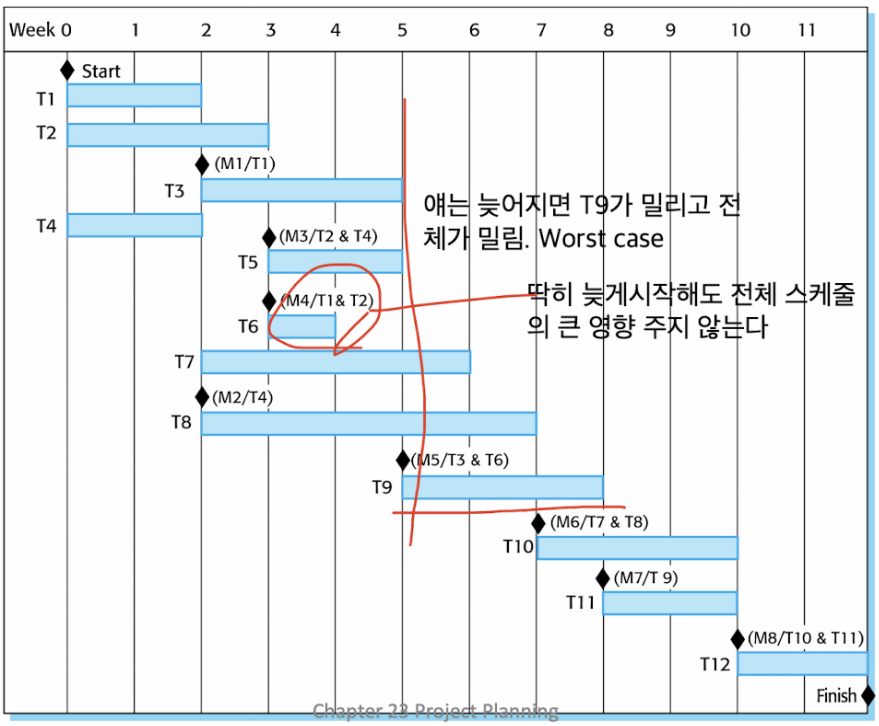
Tasks, durations, and dependencies

Task	Effort (person-days)	Duration (days)	Dependencies
T1	15	10	
T2	8	15	
T3	20	15	T1 (M1)
T4	5	10	
T5	5	10	T2, T4 (M3)
T6	10	5	T1, T2 (M4)
T7	25	20	T1 (M1)
T8	75	25	T4 (M2)
T9	10	15	T3, T6 (M5)
T10	20	15	T7, T8 (M6)
T11	10	10	T9 (M7)
T12	20	10	T10, T11 (M8)

T1의 M1이란 결과를 기준으로 T3이 진행된다

Activity bar chart

하나가 밀리면 전체가 밀리는 애들이 있음. Worst case를 계산할 때는 제일 뒷부분부터 역산으로 계산하면 됨..



얘는 늦어지면 T9가 밀리고 전체가 밀림. Worst case

딱히 늦게시작해도 전체 스케줄의 큰 영향 주지 않는다

Agile planning

- Iterative하게 그냥 애자일로 쭉 진행함
- Plan-driven과 다르게, 구현할 기능은 계획하는게 아니라 개발 도중 결정됨
- Stages:
  - Release Planning: 이 기능 구현하려면 이런게 필요하다~ 미리 정의
  - Iteration Planning: 2~4주 단위로 싸이클 반복..