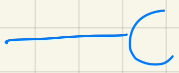
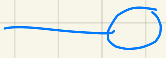
project manager, end user อยู่ใน stakeholder

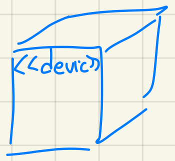
logical view $\xrightarrow{\text{rep}}$ {class diagram, state diagram} = functional design

software element map to hardware node in physical view

 ต้องการ interface  มี interface

physical result of development process → artifact

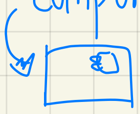
merge, access, import เป็นลักษณะที่ใช้บนเส้นของ package diagram



activity diagram → เหมือน flow chart มาก
↳ มี concurrency, activity, decision

activity, sequence = process view

componet ใน component diagram แยกย่อย → independent, encapsulated unit ที่ในระบบ
↳ clarify dependency rela., ไม่ใช่ class เป็นส่วนประกอบ



4+1 : scenario = use case

programming language → large procedural ที่ใช้ main and subroutine

layer system → seperation of concern, sand boxing, abstraction

การคุยกันระหว่าง n-tier arch → ใช้รูปแบบของ message { req, res }
layer → componet

interpreter → เขียน macro, add-on, abstract platform detail

event-based arch → event generator กับ event consumer หลวมสุดๆ ผ่าน eventbus สื่อกัน
↳ มีเทคนิค "semaphor" ใช้เพื่อเช็คว่า resource นั้นๆ ถูก access โดย process อื่นไหม

feedforward loop → การทำงานที่วนลูปเรื่อยๆ

step ใน process control system → {monitor, analyse, plan, execute}

procedural programming & main and subroutine → เหมาะกับ ระบบคำนวณ, วิเคราะห์, gen report

= data flow arch

pipe and filter → ที่ส่งข้อมูลเป็นทอดๆ → ทำให้ช้าลง แต่ → loose coupling, ลดความซับซ้อนของ data

system performance → throughput, latency

system ที่ usability → ใช้ง่าย, ลดความผิดพลาด user, ให้ user ควบคุมได้ดี

promote concept integrity → กำหนด convention, ทำ code review, good doc → อย่าทำ subteam

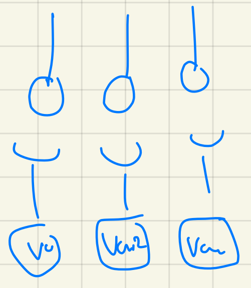
ตัวที่ทำให้ระบบต้อง respone "stimulus (กระตุ้น)" → กระทบต่อ artifact (คนละอันกับ deploy dis)
↳ mode ของระบบ "environment"

ใน quantity scenario ⇒ environment → recovering from error

ใน architecture tradeoff → คนที่เกี่ยว → stakeholder, project decision maker, evaluation team

potential ที่จะเกิดระบบล้ม → qualtiy ยังไม่ได้ → "risk scenario"

ถ้าไม่แยกทั้ง 2 ทีมออกจากกันในเชิงกายภาพ → tightly coupling



adapation

product-line → ไม่ได้ทำใน up-front deve team ลดลง

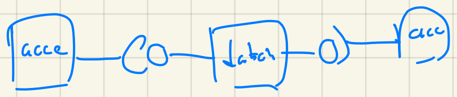
---

development view = package diagram

abstract data → ถูกกำหนดโดย dev

subroutine → pros → programmer จะเขียนเอง function กลับมาใช้, ดีกับปัญหาการคำนวณ

data acc



layer architecture base on abstract layer

machine that host server → serve host

maintainance down time = ช่วงไม่ใช้งาน

different between products
= variatio

complex system → MAPE

n-tier จะ drawbacks คือ resource เปลืองกว่าเดิมแบบ client-serv

interpreter → ไม่ใช่เหมือน os kernel

dev → สน flexibility → รองรับการเปลี่ยนแปลง req

security, performance → attr refine

อย่าให้ quantity ระบบเท่ากันทุกด้าน → ตามแต่ละระบบ only