Cloud Foundry and Japanese-style System Integration

- Lessons Learned from a Pilot Project -

Isoo Ueno

NTT Software Innovation Center

Isoo Ueno

- Researcher / Engineer
- Cloud Foundry
- CF 100-day Challenge (Please search the articles (JP/EN) on web)
- Big data, IoT
- Web hosting, Authentication
- Multi-agent, Artificial life

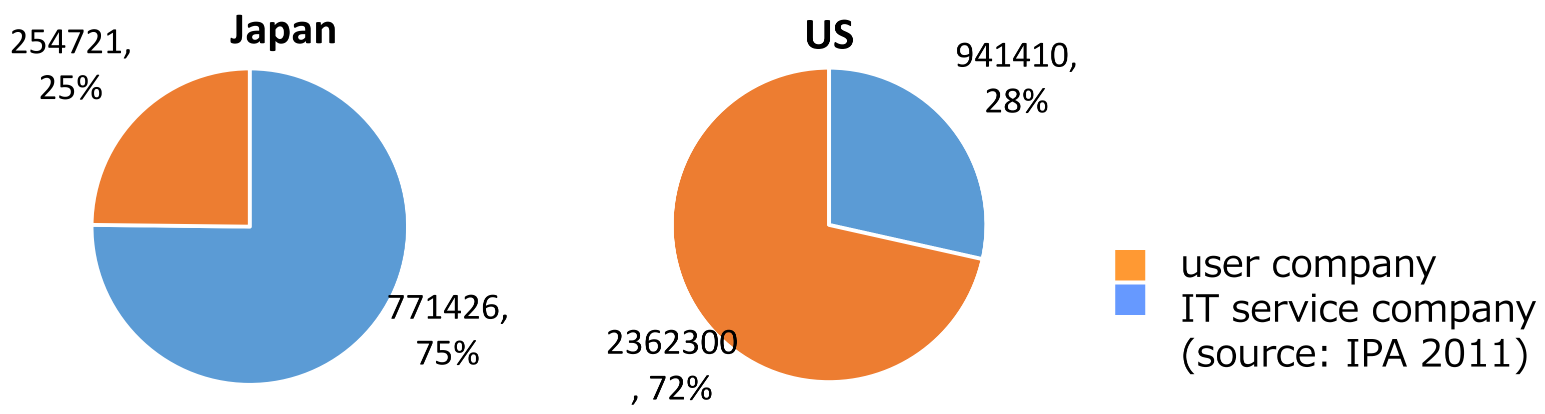
- Karate

Outline

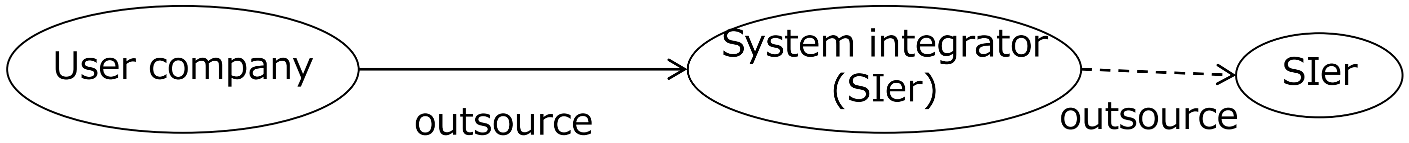
Japanese-style System Integration

Japanese-style System Integration

- IT engineers



- Japanese-style system integration

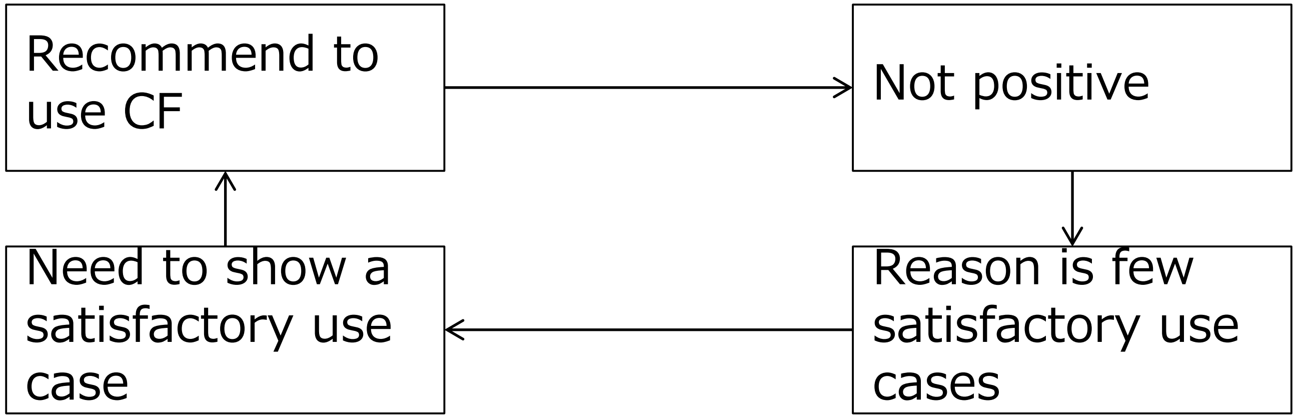


➢ Outsourcing (Client -> Contractor)
➢ Waterfall

Chicken-and-Egg Dilemma

Chicken-and-Egg Dilemma

Trying to introduce Cloud Foundry …


Recommend to use CF
Not positive
Reason is few satisfactory use cases
Need to show a satisfactory use case

- “Already have environment and systems…”
- “Our way is not so bad…”
- “Often heard PaaS, Cloud Foundy, but…”

Launch a pilot project !
- Develop a test application connecting to an existing system
- Evaluate CF based development and operations

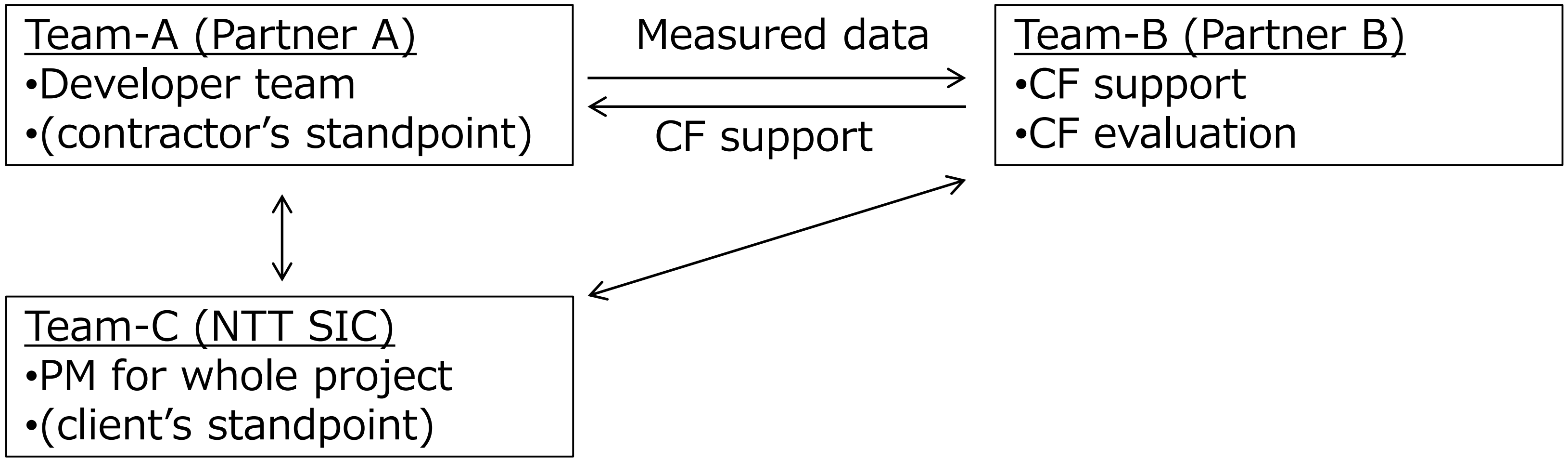
Pilot Project

Pilot Project

- Purpose

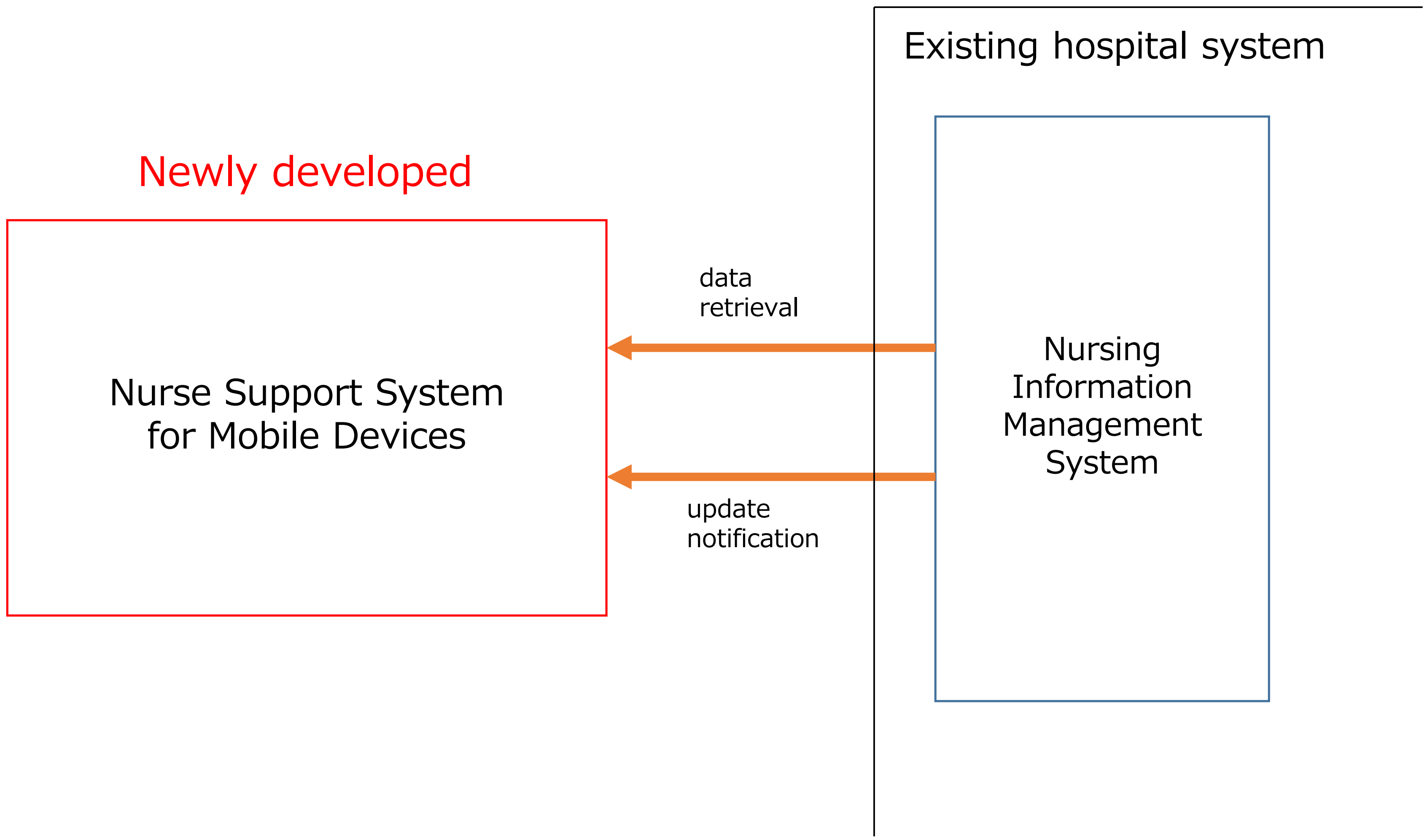
    1. Evaluate CF based development and operations
    2. Introduce CF to our software developers
    3. Recognize hurdles for developers to use CF without PaaS experiences

- Project formation



Pilot Project

- Medical Information System (evaluation version)
  - written in Java
  - 5KLOC


Existing hospital system
Newly developed
Nurse Support System for Mobile Devices
data retrieval
update notification
Nursing Information Management System

Pilot Project

- Communications
  - Weekly face-to-face meeting
  - JIRA – ticket base development
  - Redmine – document sharing
  - Slack – real time communications
  - Github – source code management

  * Working agreement is revised when needed.

- Development process
  - Waterfall base
  - Introducing weekly pseudo sprints that manage budget (time estimation) and actual results in ticket base

Evaluating the Project

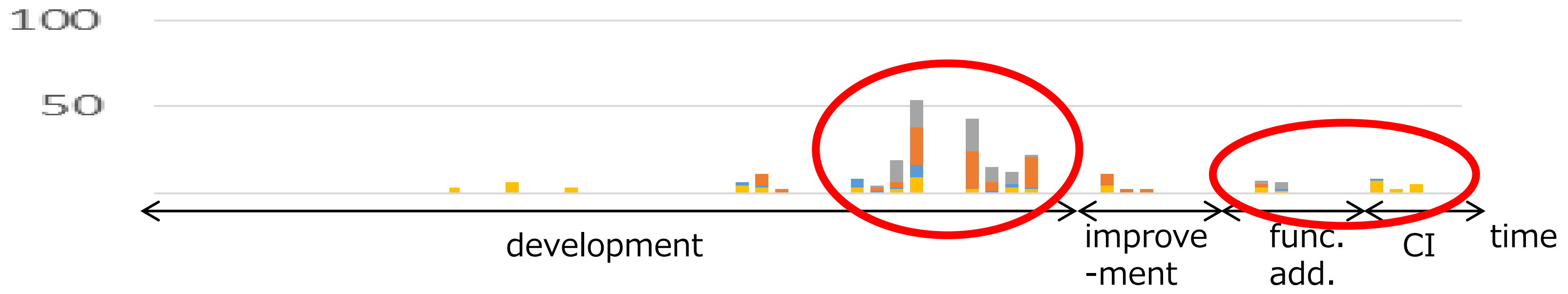
Evaluating the Project

- Development process evaluation
  - ➢ JIRA ticket records showing work time of estimation and actual results
  - ➢ Logs of CF operations
  - ➢ Design document (frequency, quantity)
  - ➢ Questionnaires (from the developer team)
  - ➢ Interviews (for each member of the developer team)
- Operations process evaluation
  - ➢ Questionnaires / interviews about:
    - these experiences through the whole process
    - operational hands-on done in the project
      - ✓ Log management
      - ✓ Scaling operations
      - ✓ Security updates

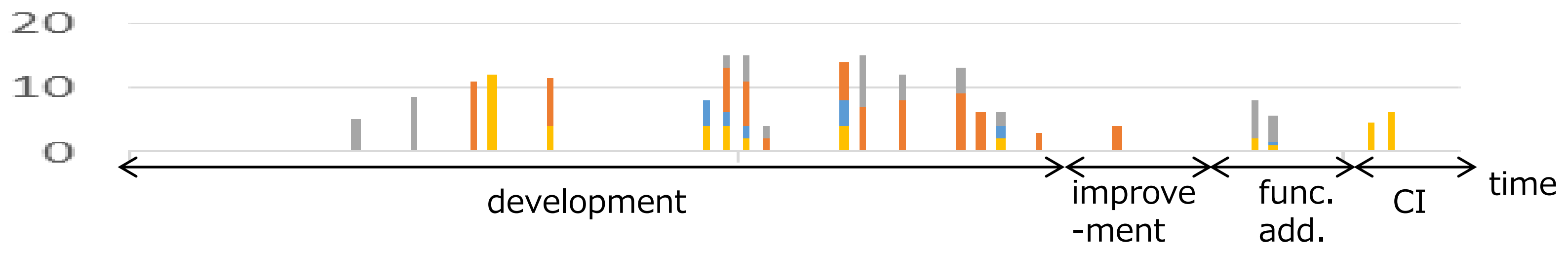
Evaluation

Evaluation: Performance of CF deployment

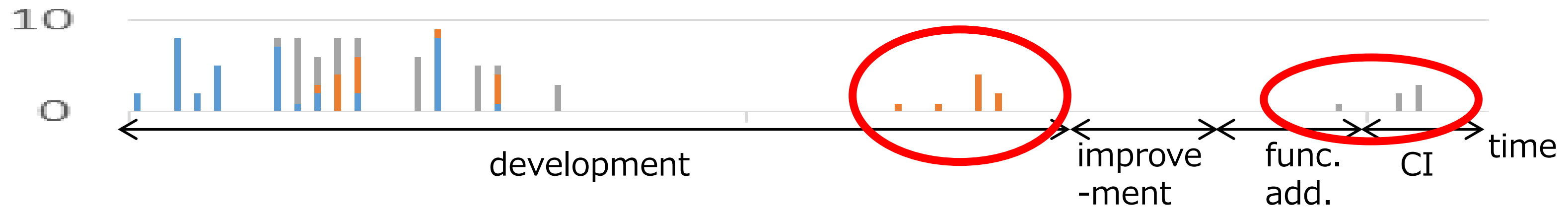
- Frequency of CF push trials per day



- Approx. work time (consuming for dev) (hours) per day



- Frequency of design document updates per day



Evaluation: Comparison of Work Time of Estimation and Result

- Work time of estimation and results (collected from JIRA tickets)

category	estimated time [H]	result time [H]	result/estimation
Total	250.2	187.1	74.8%
Patient data retrieval servlet	57.2	45.6	79.7%
Update info retrieval servlet	28	14.5	51.8%
Patient info update monitoring app.	97	77.5	79.9%
Hospital info retrieval app.	68	49.5	72.8%

  - ➢ As work time for development excluding CF study related things, result time is held in 50-80% of the estimated time (<u>approx. 75% in average</u>).

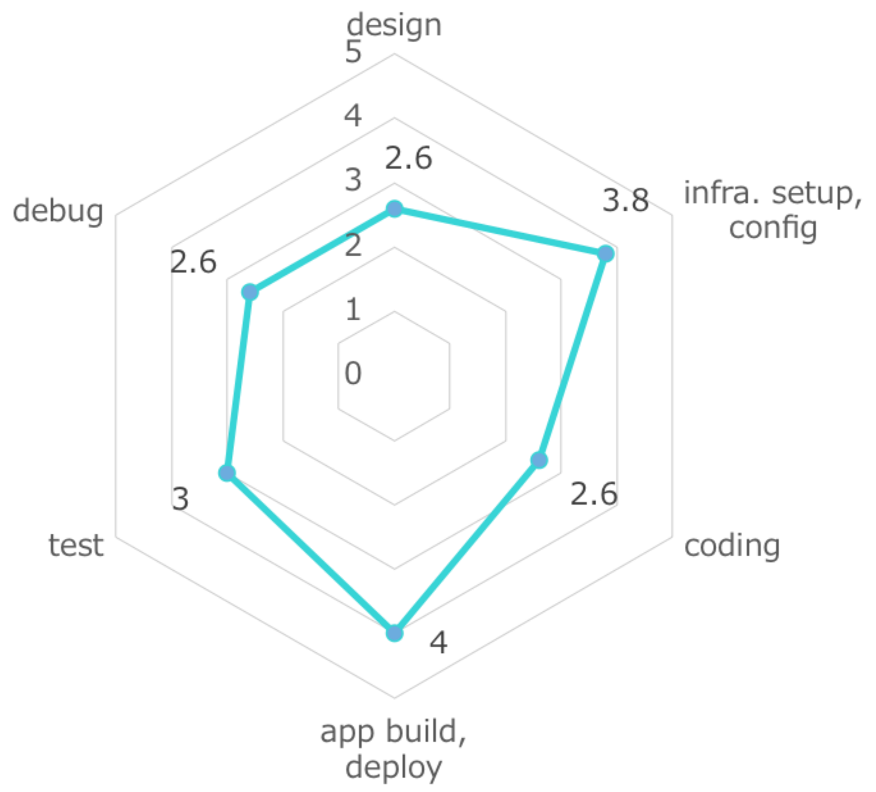
- A developer’s voice (from questionnaires)

  - ➢ “Estimated time was counted longer than usual. If including CF research, study, and related trial and error, consumed time will be approx. <u>1.5 times as long as usual</u>.”

Evaluation: Effectiveness of Development Aspect of CF

(from questionnaires)
How much (1-5) do you rate effectiveness of each action (design / infra. setup, config. / coding / app. build, deploy / test / debug) using CF?

- Five developers' average



Actions rated higher/lower than 3.0 (standard level):

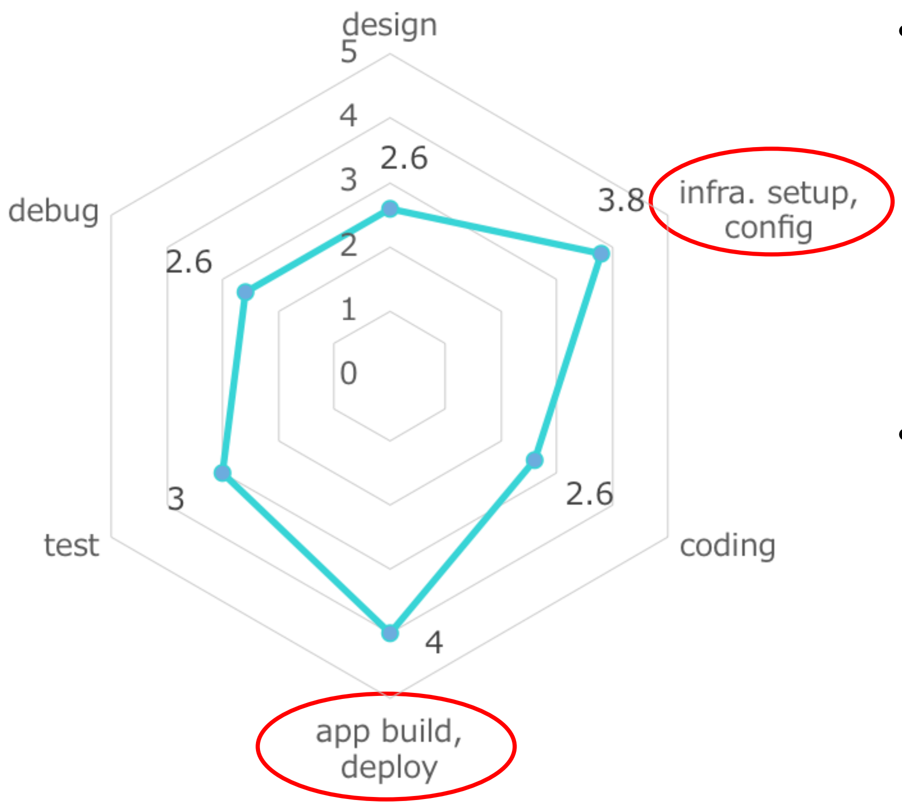
Higher: "infra. setup, config", "app build, deploy"

Lower: "design", "coding", "debug"

Evaluation: Effectiveness of Development Aspect of CF

(from questionnaires)
How much (1-5) do you rate effectiveness of each action (design / infra. setup, config. / coding / app. build, deploy / test / debug) using CF?

[Actions rated higher than 3.0 (standard level)]

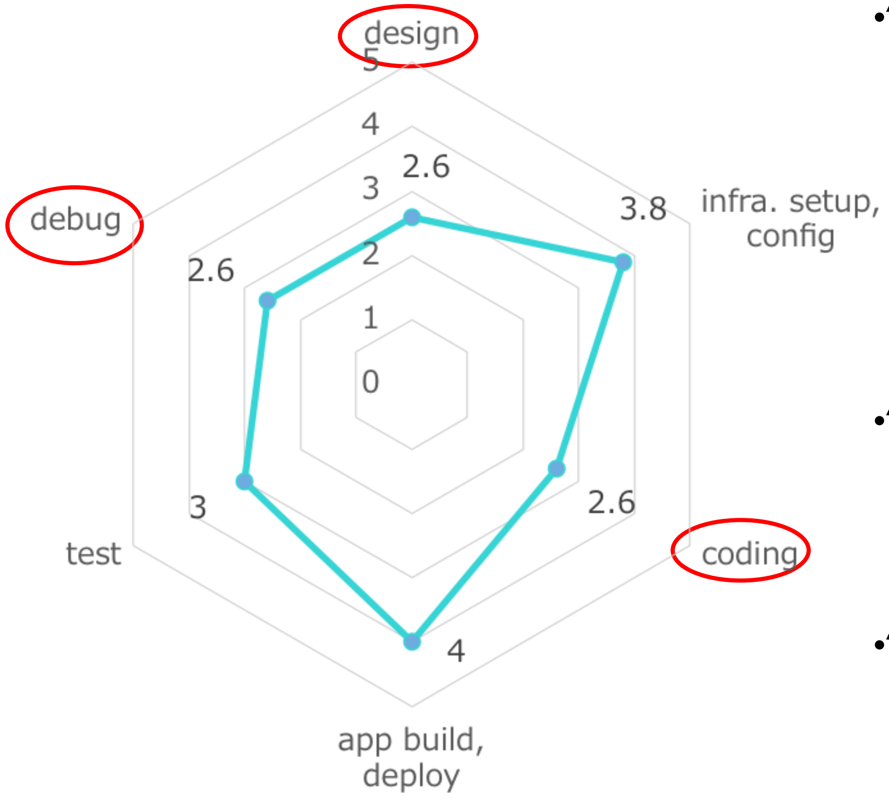


- "infra. setup, config": 3.8
  - (positive comments)
    - ➢ Infra. is automatically set up by few procedures
    - ➢ No difference b/w env's of test and production
- "app build, deploy": 4.0
  - (positive comments)
    - ➢ Only 1 command can deploy an app.
    - ➢ Manifest.yml can ease the procedure more
  - (negative comments)
    - ➢ Uneasy because buildpack is black box

Evaluation: Effectiveness of Development Aspect of CF

(from questionnaires)
How much (1-5) do you rate effectiveness of each action (design / infra. setup, config. / coding / app. build, deploy / test / debug) using CF?

[Actions rated lower than 3.0 (standard level)]



•"design": 2.6
(positive comments)
- Can conduct trial and error flexibly
- Advantage in Non-functional design

(negative comment)
- A kind of complex system containing a special feature is hard to be designed and executed on CF

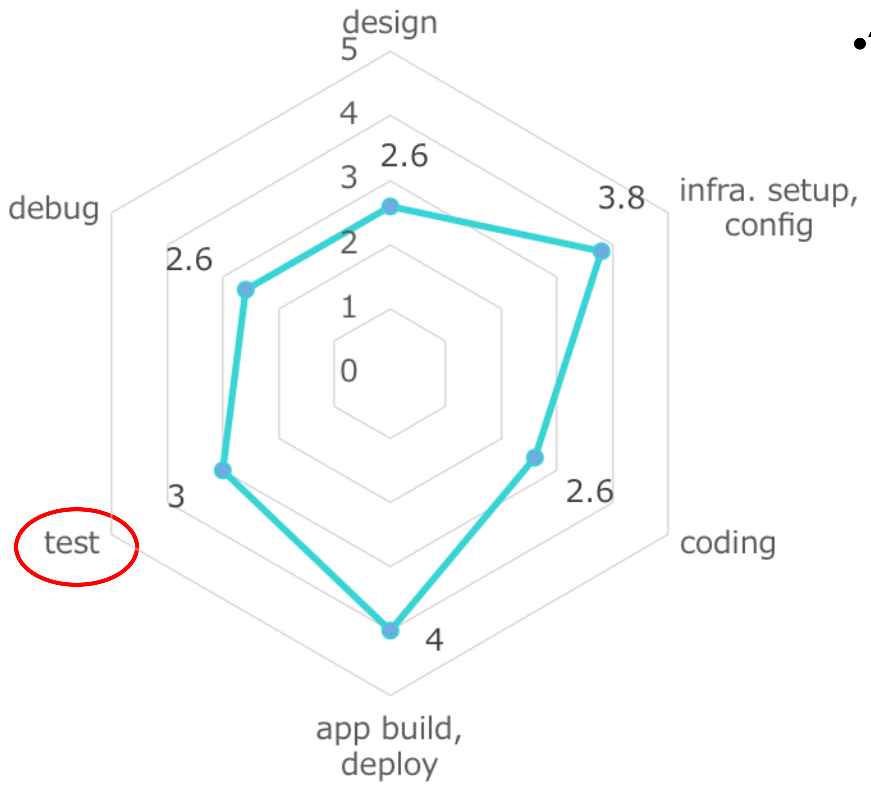
•"coding": 2.6
(negative comments)
- CF-depend coding
- Want a various sample codes

•"debug": 2.6
(negative comments)
- Cannot see if something inside CF happen, that will confuse us to detect the cause
- Sometimes deploy time becomes long that makes the debug time long

Evaluation: Effectiveness of Development Aspect of CF

(from questionnaires)
How much (1-5) do you rate effectiveness of each action (design / infra. setup, config. / coding / app. build, deploy / test / debug) using CF?

[Actions rated as 3.0 (standard level)]



- "test": 3.0
  - (positive comment)
    - ➢ CI makes us find bugs easily
  - (negative comments)
    - ➢ Test of internal processing is not suited
    - ➢ Unless using CF, UT automation is possible
    - ➢ CI takes much work load, that makes a small project hard to adopt CI

Evaluation: Effectiveness of Operations Aspect of CF

(from Questionnaires/Interviews)

item	Positive evaluation	Negative evaluation
Infra. setup, config.	• Suppress work load of build • Suppress trouble by no difference b/w dev/ops • Ecosystem like monitoring is fruitful	• Feel anxiety for over load of NW IO and disk IO about resource limitation by virtualization
Production release	• Can drastically reduce work load and risk in production release while they cost much so far	• Feel anxiety for black boxes like Buildpack
Monitoring	(became out of evaluation target)	
Log management	• Easy to confirm logs if using a tool like Kibana • Centralized management, and can set up env. for flexible output	• Need work load to build sufficient log management env. • Need to prepare log centralized env. in dev side, so easiness, which is a merit of CF, decreases
Security update	• Smooth cutover by blue-green deployment though it cost too much work load so far	• Need to take care of blue-green usage because old/new versions exist momentarily at the same time when in switching
High load measurement	• Can reduce burden and risk for non-functional design by utilizing flexible cloud resources and easy scale operations	• Need to reinforce cloud resources as well as executing scaling by CF

Evaluation: Learning CF

Lectures, hands-on, demo held in this project:

- Cloud Foundry (PaaS) lecture/hands-on for beginners
- CI/CD hands-on
- Log storing
- Operational hands-on (scaling, security updates)

Required info/lectures to be considered next time (from questionnaires/interviews/f2f mtg):

[Required information]

- **Japanese document**
- **Trouble shooting related info**
- **Sample codes for use cases like DB connection, etc.**
- **CF's able/unable item list, and some other info about CF's functions**

[Required lectures and hands-on]

- **CF operational hands-on including abnormal system like typical troubleshooting in CF deploy or operations**
- **CI (have to bring up engineers who can manage CI environment)**

Evaluation: Developers’ Voices (on CF as Cloud)

Less anxiety about capacity

- “In waterfall, it’s hard to determine HW and the spec, and the anxiety continues until the end. Since CF can support scaling or distribute applications easily in downstream process, such initial burden could be lowered.”

Less (or no) anxiety about environmental difference

- “CF can easily prepare any number of development environment identical to the production one. And, you can use resources as much as you need on demand.”

Being afraid of IO / NW hogs

- “In general cloud contract, CPU, memory, and HDD resources are clearly separated and guaranteed. However, file IO and NW are shared with others, that would be impacted by a neighboring heavy user. It’d be great If CF could resolve this issue.”

Evaluation: Developers’ Voices (on CI)

- “If the project is to make, test, and introduce small scale applications quite frequently, CI would be effective because the work load for env. setup could be reduced.”
- “Even in waterfall, if development is repeated because of function addition, etc., CI’s merits could be utilized.”
- “Tasks to create test code are added because of automated test.”
- “From the long term view, there are merits to adopt CI.”

Evaluation: Developers’ Voices (on Agile, etc.)

- “An ideal figure is to make scrum involving end users utilizing tickets, chatops, etc, and to improve applications quickly reflecting end users’ requests with test build by CI.”

- “In that case, strong responsibility and leadership are required in client side.”

Feeling safe with “guide” of PaaS, even in agile “freedom”

- “Not “free and flexible”, but “able to realize quickly by some tuned templates.” It’d be great if CF could be used like that as PaaS layer (under SaaS).”

- “CF’s ecosystem looks fruitful having many tools that can support operations and maintenances.”

Lessons Learned

Lessons Learned

Should determine the followings before starting development:

1. How to obtain the required technical skills
2. Target range to adapt CF
3. Development process

Lessons Learned: 1. How to obtain the required technical skills

- In the beginning, need to consider to cover all the technical matters:
  - prototyping
  - design
  - test automation
  - various kinds of troubleshooting, etc.

- How to obtain the required skills are case by case:
  - training courses
  - hiring resources
  - support contract, etc.

- Several ways of training (case by case):
  - Information
  - Lecture
  - Demo
  - Hands-on, etc.

Lessons Learned: 2. Target range to adapt CF

- Safety is to limit the CF coverage to simple web app.
- Stateless application
- Complicated internal process might be able to be excluded to another server app.

Lessons Learned: 3. Development process

- Development method (waterfall, agile, etc.)
- Support tools (JIRA, Redmine, etc.)

Conclusion

Conclusion

- Pilot project
  - ➢ Japanese-style system integration
  - ➢ Developers without PaaS experiences
  - ➢ CF based development

- CF worked effectively even in waterfall

- Skills should be required accordingly depending on the case
  - ➢ information/lectures/demo/hands-on, etc.