NAIST Interview Q&A

画像

Final year project of my Bachelor study
→ conducted an anthropological research to investigate the fandom of pop-rock singers/ groups such as Superfly and Every Little Thing in Japan
→ Thesis Statement
I argued that not only do fans support their favorite groups individually, but they also collaborate to support the groups, and gain materials and intimacy through "emotional resonance", which can facilitate cooperation by interpreting lyrics of their favourite songs and commonly used terms by themselves and examining various strategies of purchasing official goods together.
→ Definition of emotional resonance
the shared affective resonance produced by and through feelings such as joyfulness, sadness and nostalgia which fans feel when they are worshipping their favourite pop-rock groups
→ Why is emotional resonance important?
This “emotional resonance” is indispensable to pop-rock group fandom as it strengthens the bond between fans and singers and within the fan community. It also plays an important role in maintaining the pop-rock group fandom in the long run.
→ Methodology
a) read a bunch of academic journals to study about fandom of singers in Japan and some Western countries such as the USA
b) attended more than 10 concerts and fans meetings to observe their behaviours
c) conducted interviews with 12 pop-rock fans and a questionnaire study to gain a deeper understanding of the reasons for consuming concert goods and organizing/attending fans meetings.
→ Importance of the research
a) offers an insight into the fandom of pop-rock musicians which is significant consumer behaviour in the Japanese music industry.
b) helps people understand the cooperative space created by pop-rock groups’ fans through their creative engagement and externalization of private and collective emotion.

Explain the research plan in 1/3 minutes
→ **Research Questions:**

How does leadership of lead developers/ project managers affect the quality of code review?
→ Reason for conducting this research? (Motivation)
→ Definition of leadership ⇒ Introduce the sub-research questions
→ Methodology
→ Importance of this research(Who benefit from this research?)

<u>What is leadership? (Definition provided by Hogan and Kaiser)</u>
The ability of building effective teams and persuading people (i.e. team members) to pursue a common goal.
→ Building effective teams: setting rules (⇒ First research question:
What kind of rules do lead developers/ project managers usually set for opening a pull request and conducting a code review?)
→ Persuading people to pursue a common goal
(⇒ Second research question:
How do lead developers/ project managers implement rule to ensure useful code review are conducted constantly?)

<u>Difference of role between project manager and lead developer</u>
Project manager:
→ Responsible for communicating with the client, closer to the business side.
→ Usually manage the whole progress of the project
⇒ In this research, it is expected that PM is the one
(1) to set rules related to management such as deadline of code review, assignation of code review
(2) to briefly understand the difficulty for developers to follow rules, make suitable adjustment related to management
(3) to encourage developers to follow rules
Lead developer:
→ responsible for handling technical affairs and improve the quality of code
⇒ In this research, it is expected that lead developer is the one
(1) to set rules related to the technical stuff such as the use of keyword, limitation of stop word, format of pull request, etc.
(2) understand deeply the difficulty for developers to follow rules and make suitable adjustment from the perspective of technical skills

<u>Elaborate on the journals cited in the research proposal</u>
Hogan, R., & Kaiser, R. B. (2005). What we know about leadership. Review of

general psychology, 9(2), 169-180.ISO 690 Rahman, M. M., Roy, C. K., & Kula, R. G. (2017, May).
→ Leadership mainly concerns building and maintaining effective teams, persuading people to give up their selfish pursuits and pursue a common goal.
→ Personality of leader is an important factor of leadership and it affects the perfomance of the team.
→ Competency Model proposed by Hogan Warrenfetlz:
(a) intrapersonal skills (e.g. regulating one's emotion and easily acommadating to the authority)
(b) interpersonal skills (e.g. building and maintaing relationships)
(c) business skills (e.g. planning, budgeting, coordinating, monitoring business activities
(d) leadership skills (e.g. building and motivating a high-perfomance team)
→ Implicit Models of Leadership
(a) able to make good decision spontaneously
(b) competent (a contributing resource for the group)
(c) able to project to a vision, explain to the group the purpose, meaning and signifance of its key undertaking)
→ Leadership of CEO affects the dynamics and culture of top management team and the characteristics of the top management team influence the perfomance of the organization
→ Component of organizational effectivness
(a) talented personnel
(b) motivated personnel
(c) talented management team
(d) effective strategy for outperforming the competition
(e) a set of monitoring systems that allow senior leadership to keep track of the talent of the staff, the performance of the management group as well as the effectivness of the business strategy

Thongtanunam, P., Tantithamthavorn, C., Kula, R. G., Yoshida, N., Iida, H., & Matsumoto, K. I. (2015, March). Who should review my code? a file location-based code-reviewer recommendation approach for modern code review. In *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)* (pp. 141-150). IEEE.
→ It is discovered that 4~30% of reviews have code-reviewer assignment problem. These reviews significantly take 12 days longer to approce a code change.
→ Modern Code Review(MCR) is the most prevalent approach used by developers

when conducting code review.
Mechanism of MCR:
when a code change, i.e. patch, is submitted for review, the author wil invite a set of code reviewers to review the code change. Then the code reviewers will discuss the change and suggest fixes. The code change will be integrated to the main version control system eventually when one or mode code-reviewers approve the change.
Problem of MCR: time is required to decide who review the patch.
→ **REVFINDER, a code-reviewer recommendation tool has been purposed. With the use of REVFINDER, files that are located in similar file paths would be managed and reviewed by similar experienced code reviewers.**
It is found that **PEVFINDER accurately recommended 79% of reviews with a top 10 recommendations, enabling developers not to decide reviewers manully and eventually letting them focusing on review task.**

Ruangwan, S., Thongtanunam, P., Ihara, A., & Matsumoto, K. (2019). The impact of human factors on the participation decision of reviewers in modern code review. *Empirical Software Engineering*, *24*(2), 973-1016.
→ It is found that 16%-66% of patches have at least one invited reviewer who did not respond to the review invitation. Moreover the more the reviewers are invited, the higher the rate that the reviewers do not respond to the review invitation.
→ Human factors such as **review workload** and **familiarity between reviewers and patch authors** play an indispensible role in predicting whether an invited reviewers will take part in a review or not.
→ The **review participation rate** is the most influential factor on the participation decision of an invited reviewer. In addition, the code authoring experience of an invited reviewer also contributes to an increasing relationship with the likelihood.
(Review participation rate = responded review invitations/ received review invitations)

Masudur Rahman, M., Roy, C. K., & Kula, R. G. (2018). Predicting Usefulness of Code Review Comments using Textual Features and Developer Experience. *arXiv e-prints*, arXiv-1807.
(The Methodology may be useful for the research plan)
→ A useful comment (code review) usually triggers one or more code changes within its vicinity (i.e. 1-10 lines) in the subsequent commits (i.e. patches) of a pull request.
(In other words, A non-useful comment refers to the one which triggers code changes in unrelated places or does not trigger any changes at all.)
⇒ **Definition of useful code review used in this research**

→ **Useful review comments share more vocabulary with the changed code, and contain more relevant code elements (e.g., program entity names) and relatively less stop words**
(Useful code review comments are significantly different than non-useful comments in terms of several textual properties of the comments such as ***code element ratio, stop word ratio*** **and** ***conceptual similarity***, and **not significantly different in terms of** ***reading ease*** **and** ***question ratio***.)
→ **Reviewers' past experience positively influence the usefulness of their comments**
(Prior experience with a file being reviewed in terms of authorship and reviewing activity of the developers can significantly influence the usefulness of their review comments. However, such experience of the developers and the usefulness of review comments provided by them always do not have a linear relationship)
→ Reviewer's experience is more effective than textual content of a review comment in predicting usefulness of the comment. However, textual features could be more meaningful to the developers, and its addition to the model also marginally improves the performance

Baysal, O., Kononenko, O., Holmes, R., & Godfrey, M. W. (2016). Investigating technical and non-technical factors influencing modern code review. *Empirical Software Engineering*, *21*(3), 932-959.
→ Both technical (path size and component) and non-technical (organization, patch writer experience and reviewer activity) factors affect review timeline when studying the effect of individual variables.
→ **Patch writer experience** affects code review outcome. Factors like **priority**, **organization and review queue** also have an effect on patch acceptance.
What does "organization" mean?
→ number of developers working on a component
→ organizational distance between developers (working at the same company? having a close relationship with each other?)
→ organizational code ownership

Have you searched for others thesis other than the ones cited in the research proposal? If yes, elaborate on the content of them.
→ mainly read many journals related to factors of code review

Wang, D., Xiao, T., Thongtanunam, P., Kula, R. G., & Matsumoto, K. (2021).

Understanding shared links and their intentions to meet information needs in modern code review. *Empirical Software Engineering*, *26*(5), 1-32.
→ **R01: To what extent do developers share links in the review discussion?**
Results show that in the past five years, 25% and 20% of the reviews have at least one link shared in a review discussion within the OpenStack and Qt. 93% and 80% of shared links are the internal links that are directly related to the project.
Not only internal links, bug reports and source code are also shared in review discussions.
→ **R02: Does the number of links shared in the review discussion correlate with review time?**
Results show that the more the internal link, the longer the review time is. But there is no significant correlation between external link and review time for OpenStack and Qt. Futhermore.
→ **R03: What are the common intentions of links shared in the review discussion?**
(1) Providing Context, (2) Elaboration, (3) Clarification, (4) Explaining Necessity, (5) Proposing Improvement, (6) Suggesting Experts and (7) Informing Splitted Requests for the common intentions.
Especially for internal links, it is found that the most popular intention is to provide context. On the other thand for the external link, elaboration of review disccusions (providing a refrence or related information) is the most prevalent intention.

Based on the results of the research, the following suggestions have been made for patch authors, review teams and researchers.
For **patch authors**, they are advised to r**ead the project related guidelines** in order to have a better **understanding environment** before submission. Moreover, it is suggested that the information brought by the shared links is helping as an indicator for review teams to clearly understand the patch implementation context.
For **review teams**, they are advised to share links to **transfer necessary information** for guiding the patch author and the review process. Links usually provide a concise and explicit answer when comparing to explain it in prose so sharing links is an alternative of automatic change summarization to satisfy the information requirement.
For **researchers**,
1. The **mechanism to automatically recommend related patches** can be improved not only based on the similar change history, but also **considering the patch contents**.
2. A functionality to detect alternative solution patches (i.e., patches aim to achieve the same objective) is needed, since the empirical study shows that the second most

requent intention of sharing review links is to explain necessity.
3. A tool to **recommend guideline and tutorial related link** would be especially useful for **novice developers** and help them to be familiar with the project environment.

Hirao, T., Kula, R. G., Ihara, A., & Matsumoto, K. (2019). Understanding developer commenting in code reviews. *IEICE Transactions on Information and Systems*, *102*(12), 2423-2432.
→ The number of both general and inline comments varies among reviews and acress studied systems.
→ The number of comments tends to increase steadily over time in the largest studied system(Chromium). Moreover, the number of comments tends to stabilize in other studied systems, implying that reviewers are more likely to participate more actively rather than just commenting.
→ **Human experience and patch property features** have the **strongest impact** on general and inline comments, respectively. (The largest the patch churn is, the more likely the review wil receive comments and words.)
→ It is suggested that the experience of the author and reviewers and the property size of patches should be considered more meticulously before the start of the discussion.

______________(Other theoretical journals)______________
Viera, A. J., & Garrett, J. M. (2005). Understanding interobserver agreement: the kappa statistic. *Fam med*, *37*(5), 360-363.
→ Kappa statistic is the most commonly used statistic for measuring the agreement between two or more observers who should inlcude a statistic that takes into account the fact that observers will sometimes agree or disagree simply by chance.
→ A kappa of 1 indicates perfect agreement while a kappa of 0 indicates agreement equivalent to chance.
→ Calculation of kappa agreement: (Number of response of extremely agree + Number of response of extremely disagree)/ total number of data

Methodology of the research (Is it practical? How to collect data?)
- conduct a empirical study by joining around 20 or more software development teams
(→ beforehand conduct a preliminary survey to check the existence of rules and leader (lead developer/ project manager
→ How to approach the team: through using my personal connection in my

workplace as well as the online developer community I have been joining (i.e.IT KINGDOM))

↓ **Evidence/ data of rules (⇒ 1st sub research questions)**

1. Collect data about rules written down on README files or other crucial documents (e.g. Google document, Notion) for development.

⇒ First, classify the rules manually. After collecting a certain amount of data, develop an automatic classification system.

↓**Evidence/ data showing the effect of leadership on the quality of code review (⇒ 2nd sub research question)**

2. Check the pull request in GitHub regularly to investigate whether developers follow the rules stricly and whether the rules facilitate useful code review.
3. Attend meetings of the teams and keep a record of all messages in Slack to understand how the lead developers/ project managers guide other developers to follow rules and to improve the quality of code review

(Reference: "An Analysis of GitHub Newcomers' Onboarding to OSS Project")

Why do you want to conduct this research (Motivation)?
（Why is this research important? Why do you want to study leadership?)

- realize that leadership really plays an important role in ensuring the quality of code after participating in several commercial and private collaborative software development project

⇒ Leadership is important
because with competent leader, developers
1) have a clear direction on how to improve the quality of pull request and code review by following rules (⇒ example of rules)
2) are encouraged to contribute to code review in an effective way

⇒ Without competent leader, developers do not know how to generate a useful code review for the project and have low motivation to contribute to code review.

⇒ leadership can be observed from the aspect of rule setting and guiding/ encouraging the developers to follow rules.

Reason for setting the first sub-research question(What kind of rules do lead developers/ project managers usually set for opening a pull request and conducting a code review?)

→ **Why studying pull request?**
the quality of pull request affects the motivation of code reviewer

- **Why rules are set? (How happen if rules are set and how happen if rules are not set?)**
- Rules are important to ensure that all developers know **what to do in each process specifically** and **all members work for a common goal.**
- for successful projects, lead developers usually set rules in detail and both lead developers and project manager take their initiative to encourage other developers to follow.
- Without rules, the progress of the project becomes really slow (e.g. No one takes the responsibility to conduct code review) and the code is usually chaotic (e.g. no fixed format of code, existence of repeated code or similar code)

Examples of rules frequently set:
→ Whether to use static checker (e.g. eslint) and automatic code formatter(e.g. prettier) before giving a pull request
→ Format of pull request (Attach the link of the ticket of task management tool. Describe the change from the point of view of the developers/ users. How? Why?)
→ Keywords (for example, words used in the ticket of task managment tool like Jira or Redmine)
→ Limitation of stop words
What are stop words?
→ a set of commonly used in any languages and these words usually don't carry any important meaning
→ Types of stop words:
1. determiners (e.g. a, an, the, another)
2. coordinating conjunctions (e.g. not, or, but)
3. preposition (e.g. in, under, below, towards)

→ Whether or not check the validation of code (ensure the code runs well) (Who is responsible for this check? developers? reviewers? Both?)
→Who is responsible for the code review?
(Assign one person for reviewing all the code? assign a person randomly every day with the use of self-made lucky draw web application? Apply the REVFINDER model (files that are located in similar file paths would be managed and reviewed by similar experienced code reviewers) suggested by Masudur Rahman)
→ Existence of the deadline of code review

(Reference:
https://smltar.com/stopwords.html
https://kavita-ganesan.com/what-are-stop-words/#.YhNA2pNBzDI)

<u>Reason for setting the second questions(How do lead developers/ project managers implement rule to ensure useful code review are conducted constantly?)</u>
→ Rules are not useful if they are irrational or are not followed by any members.
→ Leadership is not just only about rules, but it is also related to the means of convincing members to achieve a common goal.
 (How the lead developer/ project manager motivates others to follow rules to help each other through conducting useful code review)
 ⇒ e.g. explain to the group the meaning, significance of the rules, care about the needs of other developers, collect opinions from developers

<u>Importance of this research/ Contribution (Who benefit from this research? How?)</u>
<u>Developers</u>
→ understand the role of lead developers/ project managers and the reasons of setting strict rules and encouraing members to follow.
→ give practical ideas related to rules setting to raise of the quality of the code project
<u>Leaders</u> (Lead developers/ project managers/ CTO)
→ understand what kind of rules should be set and how to manage a software development companies for purusing a common goal and striving for excellence
→ (for CTO only) have a better undestanding on how to choose a suitable leader
<u>Researchers/ Scholars</u>
→ understand a brand-new and important human factor of code review (i.e.leadership) which have not been mentioned for a long period of time in the academic field of software engineering
→ understand a whole picture of human factors of code review as leadership (rule setting) is basically built up on the existence discovery such as patch writer experience, review workload,etc.

<u>Threat to the Validity of this research(課題点)</u>
→ sample size may be too small and cannot be representative for all open source system.
 How to avoid/ minimize the effect:
 If time is limited and I cannot expand the sample size of case study, I will collect sample from the list of repositories in OSS and use them as my data ⇒ and then use automatic classification system
 (which I’ve mentioned in the methodology verbally)

OR
"not avoidable"

(→ the existence of the uncontrollable variable ⇒ personality of the lead developers/ project managers
→ Why is it a threat ?
Hogan and Kaiser suggested that personality of leader changes the style of management.
⇒ a great factor affecting the result of the second sub-research questions
⇒ cannot be avoided but (...use the kappa statistic ...))

Expected outcome/ result (得られる見込み)
→ For projects with many useful code review, it is expected that
(1) rules are set explicitly in many aspects such as format of pull request, assignation of code reviewer, deadline of code review, etc.
(2) Lead developers/ project managers take their initiative to
(a) understand the current situation of other developers
(b) listen to the opinions of developers related to rule setting
(c) understand the difficulty for developers to follow some rules
(d) encourage developers to follow rules and give useful code review constantly
(d is the most important because leadership is about how to encourage others to pursue a common goal ⇒ refer to the definition of leadership)
(e) make suitable adjustment regarding rules.

To what extent are you planning to complete the research during the master programme ? (修士課程でどこまでやれればいい？)
During the master programme, at least
→ Collect, classify manually and analyze the data (rules) written on README file or other documents from 10 collaborative software development teams
→ Collect and analyze the data of pull request page, slack messages, etc. to observe the influence of leadership on the quality of code review from 10 collaborative software development team
If time is allowed,
→ Expand the sample size to 20 groups
→ Develop a automatic classification system for (a) classifying rules and (b) studying the effect of rules on the quality of code review

Programming experience (What programming language?)
→ started learning programming since May 2020
→ learned how to create homepage, corporate websites and landing page with the use of HTML, CSS, jQuery, WordPress
→ gained a sense of satisfaction after creating several websites in a short-term freelance job
→ took a further step and learned JavaScript(React, Next.js, TypeScript) in detail
→ Now, I am working as a frontend developer and usually write Next.js code to develop the User Interface of the web application and fetch data from the database with the use of API.
→ sometimes write fastAPI (well-known framework of Python) code for fixing bugs or conducting minor changes of API.

Difference between characteristics of programming language you have used (TypeScript, Python)
→ Typing
TypeScript: Static typing (give our code more structure, enhance debugging and refactoring, make our code look like self-documented, has compiler alert about type related mistakes)
Python: Dynamic typing (types are only determined at runtime, causing it to consume chunks of memory during execution ⇒ not suitable for memory sensitive processes)
→ Usage
TypeScript: frontend, backend, mobile application development, desktop application development
Python: backend, desktop application development, machine learning, Data Science, AI
→ Speed
a) Coding Speed
TypeScript: verbose
Python: clean and easy to write
b) Runtime execution for large scale application
Large applications require tasks which are tedious on the compiler and slow down the execution time so TypeScript is better.
c) Raw performance
→ coding memory-intensive tasks (e.g. games, 3D graphics) in Python ⇒ CPU takes a hit and significant drop in performance will occur.
→ Python is not asyncronous at its core ⇒ require AsyncIO library to achieve asynchronous programming

(Reference:
General difference: https://blog.logrocket.com/why-is-typescript-surpassing-python/
Gramatical difference: https://www.freecodecamp.org/news/python-vs-javascript-what-are-the-key-differences-between-the-two-popular-programming-languages/)

What is Object-oriented programming (OOP) ?
→ a computer programming model that organizes software design around data, or objects, rather than functions and logic
→ Structure of OOP
a) Class: user-defined data types that act as the blueprint for individual objects, attributes and methods
b) Object: instance of a classes created with specifically defined data
c) Method: the inforamation that is stored and is usually defined in the Class template
d) Attribute: represent behavior and might return information about an object, or update an object's data.
→ Characteristics of OOP
a) Inheritance: Child classes inherit data and behaviors from parent classes
b) Encapsulation: contain information in an object, exposing only selected information
c) Abstraction: only exposing high level public methods (public instance) for accessing an object
d) Polymorphism: many methods can do the same task

(References:
https://www.techtarget.com/searchapparchitecture/definition/object-oriented-programming-OOP
https://www.educative.io/blog/object-oriented-programming)

Why do you want to change your academic field from Japanese Studies to Information Science?
→ work as a professionalist in Japan in the future (stand out of the crowd of generalist by being a professionist)
→ interested and currently engaged in software engineering by working as software engineer
→ in order to build a successful career, I want to study more about infomation science comprehensively and conduct meaningful research related to software engineering

Why do you want to study at NAIST?

1) take a master degree of Information Science **in Japan**
unlike other universities like JAIST, your school welcomes students with different academic background.
(JAIST, Tsukuba University: not welcome students graduated with a Bachelor of Arts)
As a graduate of a bachelor of arts, I would like to take this opportunity to change my own academic field.
(→ **Why study for a master degree in Japan?**
want to work in Japan as a software engineer in the future
→ Japanese companies tend to hire candidates gruduated from university/ graduate school in Japan rather than the one graduated from overseas universities.)
2) feel secure in NAIST
a) Students and professors are kind and willing to provide me with practical advise for my research as well as this interview.
b) Admission staff in NAIST
→ are experienced and responsible in handling the application of overseas candidate from the entrance exam application to the COE application
→ speak fluent English and Japanese so I can communicate with them without facing any language barrier
3) want to conduct research related to software engineering
reasons written in the answer of the next question (↓)

Why do you want to conduct research in software engineering laboratory?
→ don't have much computer science knowledge since I graduated with Bachelor of Arts
→ easy and practical for me to conduct research related to my current work field (i.e. software development)
→ interested in software engineering (feel excited in software development based on my past experience)
→ code review is basically part of my life and daily routine as I am working as a software engineer currently ⇒ easy to collect data and obtain inspiration from my daily life.
(**Why choose SE lab in NAIST but not the one in other universities?**
→ Prof. Masami Hagiya in Tokyo University: too many academic field such as theory of computing, formal methods, DNA computer, hard to focus on a specific topic
→ Fukazawa Lab in Waseda University: Usability, Accessibility, IoT/ cloud service ⇒ too technical and I have less interested in these area)

Courses I would like to take in NAIST
→ Programming Course (learn Python more)
→ Data Science (learn how to construct models and conduct small data analysis ⇒ useful for analyzing the data of my research)
→ Software Engineering (build a strong foundation for my research and learn more about software documentation and communication which is definitly useful for my research topic)
→ Advanced Algorithm Design (learn algorithm which is the basic of computer science for work)

Career path after graduating from NAIST
- work as a software engineer in well-known big corporations like Yahoo, Mercari, etc.
  so what I need to and am going to do are:
- participate in several internship programmes offered by well-known company
- devote myself to the research so that the HR staff and developers of the company think that my research is attractive

How many lines of code have you ever written? (コードを何行ぐらい書いたことがあるか？)
countless (more than 30,000 lines)
Based on the counting system of forkwell,
TypeScript: 19,054 lines
JavaScript: 10,295 lines
(only count the lines of code of my personal repositories)

Result from the analytical system of forkwell :
超絶技巧のTypeScript忍者で**2021年**から累計**19054行（上位5.5%）**変更しました。

他には、**JavaScript**を**2020年**から使いはじめ、**累計10295行（上位24.6%）**変更しました。

直近半年では主に**JavaScript、TypeScript**を書いています。土日は異常なくらい多くのコードを書きます。深夜は他の人より明らかに多くのコードを書きます。

半年間でアクティブに開発しているリポジトリは
gabriel6181997/postapp_backend(58 commits)です。他にも
gabriel6181997/postapp_frontend(33 commits)、
gabriel6181997/node_chat_app(29 commits)を開発しています。

Contribution to OSS (Open source software)

→ At work, develop an analytical web application for analyzing the current situation of the logistic factory as a frontend developer
(cannot reveal the detail because of the non-disclosure agreement with the company)

→ In my private life
(1) create a website for myself with the use of Next.js and microCMS
(2) join several collaborative software development project
⇒ e.g. sow-agree (a web-application for farmers to manage the delivery of the harvest)
https://sow-agree.web.app/

→ All the code have been uploaded to GitHub
Link of my GitHub account: https://github.com/gabriel6181997

Perferences of Laboratory

Perferences	Name of Laboratory	Professor	Main Topic of the laboratory
1	Software Engineering (ソフトウェア工学)	松本　健一	Code review, npm package, activities of software engineer,etc.
2	Software Design and Analysis (ソフトウェア設計学)	飯田　元（はじむ）	Software Repository Mining,Software Analytics, Machine Learning
3	Internet Architecture and Systems (情報基盤システム学)	藤川　和利（かずとし）	Networking, Cyber Security
4	Social Computing (ソーシャル・コンピューティング)	赤牧　英治	Sociology,Psychology
5	Cybernetics & Reality Engineering (サイバネティクス・リアリティ工学)	清川　清	Virtual Reality, Augmented Reality

Questions to ask the professor

1. I have heard that students have opportunities to give a presentation regarding their research topic in some conferences? What conferences do students usually participate in? How many times do students usually participate in the conferences during their study of master degree?
(→ Appealing point: cherish the opportunities to join the conference and let more people know my research topic)

2. If I pass the entrance exam, I will have about 3 weeks left behind the start of the 1st quartile. Is there anything I can do to prepare for the study at NAIST?
(→ What I’ve been doing: reading journals related to my research topic)
3. Is high level of Mathematics ability a requirement to pass or get a satisfactory grade for most of the courses?
(→ I’ve studied Maths so hard for the entrance exam. What do students who are not good at Maths usually do to catch up with the required Mathematics ability? How do they learn the required Mathematics?)