1 2019 Paper 7 Question 8

https://www.cl.cam.ac.uk/teaching/exams/pastpapers/y2019p7q8.pdf

In this question you will be asked to reflect on a project you have been involved in or observed, in which a design evolved, or could have evolved, through applying a theory of user behaviour. You may refer to a Part IB group project, practical work from IA/IB Interaction Design or a project outside the Computer Science Tripos. You are advised to read the whole question before choosing a project to describe.

(a) Describe the project in one or two sentences.

The IA Interaction Design project: designing a weather app for usage by Rowers in Cambridge. The app would function as a replacement weather app, but providing rowers with specific information tailored to rowing – with a focus on the CUBC flag (which indicates what skill is required to go out at this specific time).

(b) Describe the intended users of this system, and the benefits that they would obtain through using this system.

The users of this system would be Cambridge rowers – they would be able to quickly check and get tailored information to whether they were allowed to go rowing. The current website is archaic, and rowers who wake up at 4/5am aren't mentally prepared to check the weather on such an outdated website. so true

(c) Describe a theory of user-behaviour that is relevant to the project, explaining why it is relevant.

"Show that it means to apply theory"
Doesn't want a particular type of theory.

Goal-oriented search. The way in which users search for information on an interface is via a goal-oriented search. The stages of a goal-oriented search are: ✓

- Goal

  The user determines what their goal is.

- Availability

  The user searches for something which they believe may help them find their goal.

- Match

  The user finds and selects the thing which they believe may help them achieve their goal.

- Feedback

  The user observes the change caused by their action and determines whether their goal is achieved and if not, whether the option they chose was correct or helped achieve their goal.

If it's in any way a design task, define your user.
Then systematically define what the design would be – doesn't matter what the design would be.

Memorise the taxonomies / frameworks / methods. This is the only thing they can ask for.

Be systematic to get teh top marks. Systematically apply theories.

(d) Would application of this theory be formative or summative? Explain why.

Application of this theory would be formative. The application of goal-oriented search in user-centered design is via Cognitive Walkthrough. can be The researchers establish a task which a user may want to perform and attempt to perform it, noting down any issues they encounter.

Since Cognitive Walkthrough finds specific issues with an interface, researchers can use this information to determine how to improve an interface. Therefore the evaluation would be formative.

(e) How are the opportunities for design evolution different, when either summative or formative evaluation methods are applied? Your answer should refer to the roles for divergence and convergence in a design process.

Summative evaluation is an evaluation which tells you whether a product is good or not. These do not explicitly help discover how to *improve* a product. For example

We're talking more about evaluating users

We will go hard in one of the two evaluation frameworks because it's the most useful.
The main purpose of this part of the course is to teach you to be systematic.
Memorise frameworks, taxonomies and models of users.
Be systematic.
Commit to one of these frameworks and be systematic in choosing it o evaluate things.
If you choose one of these frameworks, and you can apply it properly (without making things up, you're doing things well!)

Closeness of mapping
Closeness of mamping to the domain it's been written for. You want to see whether this thing translates to me given the domain I come from.

Role Expressiveness
Relates to the purpose of what this does .Is this defining a function / running a function. You're looking at the components, the sequence of it.
You want to know what the thing does. IE you want to make it clear what a given line does (not really related to names).
It's not looking at what the words mean. Do you understand the purpose of each of the words.

Abstraction
Availability and types of abstraction mechanisms.
Measuring how much abstraction is used (not conflicting with others).
Evaluation of how abstract the notation being used is.

The idea is these are not replacing each other -- you tune each of them. There is no such thing as a perfect combination. You find an optima.

Provisionality
Ability to edit things

Premature Commitment
Avoiding forcing users to make big commitments.

Verbose Java

This is trying to solve problems with the language that are solved by an IDE.
It's restrictive and naive to think people aren't using an IDE.
Syntax highlihgting is a significantly more useful secondary notation.
This langauge would just be forcing secondary notation on users.

Design systems that take into account secondary notation in IDEs.

When you design frontend systems, make backend concepts be frontend concepts as well.
IE "principle of least privilidge" is backend concept
"minimum default structure" is frontend concept

In formative evaluation, we use a component-centric evaluation and compare it to itself.

In summative evaluation we compare versions and see whether they're better or worse. We try to evaluate whether the whole is better or worse.

In summative, you get rid of alternatives.
In formative, you search for areas in which to explore alternatives.

Theory is "things that are well-known to be useful".
In HCI, theory involves the methods.
When we talk about a theory, we talk about theory "this is a useful model".

Apply a theory: apply methods / models / frameworks / taxonomies.

A/B tests give evidence as to the size and direction of an effect – however do not tell us how to increase the effect size.

Formative evaluation tells us how to improve a system. For example, in Design Ethnography we observe people using the system and query them about things they do. This gives large amounts of quantitative information which can be used to improve a product.

Formative evaluation is most useful for the divergent phase in the iterative design process – we're given new ideas about how to improve systems so can add these new systems. While summative evaluation is better for the convergent phase – it tells us whether a system is good and which systems are better – we can use this to eliminate the worst interfaces.

(f) Explain what kind of evidence would be required when applying this theory in your project, and how you would obtain it, noting whether this evidence would involve quantitative or qualitative data.

Goal-oriented search provides a concrete theory for how humans attempt to achieve their goals on an interface with which they are unfamiliar. This generates qualitative data. This would generate qualitative evidence that the designers found with their own system.

(g) Describe how your project team applied, or could have applied, a method that would improve the reliability of *quantitative* data.

We could have used Design Ethnography to observe usage of the product and determine whether real user problems aligned with the problems we observed during Cognitive Walkthrough. If they did not, we would need to investigate why.

(h) Describe how your project team applied, or could have applied, a method that would improve the reliability of *qualitative* data.

Get more of it. Using a larger data sample decreases the variance. After the product was complete, we could have used an A/B test to perform formative evaluation and determine whether rowers found our app to be better than the alternatives on the market. A/B tests generate quantitative data.

The way of increasing reliability of data is to minimise variance. This can be done by taking a larger sample.

What did they mean by "Theory"... under my understanding, the question confused lots of different topics – theories of user behaviour with evaluation methods with empirical evaluation methods.

2 2019 Paper 7 Question 9

(a) If HCI methods were applied to the design of a programming language and tools, what research questions might be explored, according to the concerns of first wave, second wave and third wave HCI respectively?

- First Wave

First-wave HCI was primarily concerned with efficiency of usage by professionals. In the context of a programming langauge, "efficiency" relates to productivity of users and quality of source code generated.

Research questions that might be explored under the concerns of first-wave HCI may include:

https://www.cl.cam.ac.uk/teaching/exams/pastpapers/y2019p7q9.pdf

  - How can we design the language to minimise the error rate of expert programmers

    What is the optimal tradeoff between type annotations and type inference?

    Should there be restrictions on the format of variable / function names so they're not confused.
  - How can we design the language to maximise the rate at which expert programmers can generated code
  - What language features or restrictions maximise the quality of code that programmers write
  - What langauge features can we include/exclude to maximise the quality of compiled code.

- Second-wave HCI

Second-wave HCI was characterised by designing usable and learnable systems. These systems often included personalisation options – each user would have their own system.

  - What language features do users find hard-to-learn. Can we make them easier to learn?
  - How can we allow users to tailor their programming to their own personal taste without making code unreadable to other programmers.

    For example, how much liberty should we give programmers in code style – how many ways of doing things should there be? Consider list comprehension – it tailors programming style but can be unreadable for some people.

- Third-wave

Third-wave HCI was characterised by interfaces designed to maximise user experience. Due to the arrival of ubiquitous computing, usage of interfaces became discretionary so interfaces had to become prettier to retain users. Users now use many interfaces; all serious programmers know many languages.

Converting these principles to programming languages leads to research questions such as the following:

  - What language features do users *like* using? Why?
  - How can we write a language that users *understand* and *enjoy*?

(b) What empirical methods might be appropriate for studying programming activity from the perspective of each of these three waves.

- A/B tests

  Implementing language features and testing whether they increase or decrease efficiency or compiled code quality.
- Design Ethnography

  Observing workers to see how they use language features, what they use right and what they use wrong.
- Grounded Theory – looking at data online to see how people feel about different features, what do they find annoying and what do they like?

(c) Consider a programming language that has been proposed for the specification of firework displays. Suggest an analytical method that would be appropriate for evaluating and refining the usability of this language and associated tools.

Cognitive Dimensions of Notations

Languages have a broader audience. You're not building systems to do a task you have in mind.

Main trend: software had lifecycles. You would update software.

So development has a lifecycle. So different people have to work on the same things down the line

ie Java

C# is C++ but with java-style syntax. It was more verbose and easier to read and develop on.

3rd wave just wants "make it work"!
C# is halfway between C++ and Java. It's syntactically abstract.

In 2nd/3rd wave IDEs arrived this made the interface nicer. It's about usability.

Jupyter Notebook is 3rd-wave

Emergence / Popularisation of OOP

Breaking apart of systems into components -- related to the lifecycle. Suddenly you can employ someone usefully to design the relationships. So you now have two teams which work on the same software and never talk to each other but will make compatible software.

IE C -> C++

Software becomes the same because it's so easy to differentiate on experience.

So 3rd wave is designed to optimise experience. IE Python -- people like programming in it.

Target users for python was mathematicians. But was just fun and easy so people liked it.

IE open-source software is often written in Python.

Python is similar to how you would write pseudocode.

Now, you have so much more code, systems are so much bigger -- I just want a language which breaks things down into smaller parts. Users of programming languages are making personal websites, fun flash games, making really basic stuff -- so why have a language which is brutally overpowered.

very nice

(d) In terms of the analytical evaluation method proposed in part (*c*), define the target user, the nature of their task and several specific usability requirements that would result from that task.

The target user is an amateur fireworks enthusiast who is organising a mid-size display for their local community. They want to make it perfect, however the display shall take a long time so will not have time to fine-tune everything.

- Premature commitment

The language should have negligible premature commitment. Designing art is an iterative process and involves trial-and-error. Commitment to launching or styling a display in a particular way would limit the creativity and greatly increase the time commitment.

- Progressive evaluation

The display should have very high progressive evaluation. The display is aesthetic and the user should be able to check that the display *will* look good before the display itself. So they should be able to simulate the display. Furthermore, they should be able to start this simulation from any point in the display (ie they shouldn't have to resume a 30 minute display from the start every time they modify the finalé)

- Viscosity

The display should minimise viscosity. As stated before, art is an iterative process and so requires frequently editing previous work. This should be supported and taken into account in the programming language design.

- Error-proneness

The language should be designed to minimise common errors. For example, the language should support subtyping such that the user can create specific types of firework and operate on those rather than specifying the parameters or each firework (which they're likely to get wrong eventually).

- Abstraction

The language should have some abstraction – while the target user is not a professional programmer (and as such is unlikely to understand complex typing relations); it should allow some abstraction such that users can declare their own types of fireworks such that they don't have to repeatedly create fireworks with the same specification.

IE if the user has 50 fireworks of the same type, they shouldn't have to have 50 lines of
`Firework f = new Firework(250, 30, "red", 6);`).

(e) Choose one requirement identified in part (*d*) and describe in detail an empirical approach that you would take to evaluating whether this requirement has been met.

I choose error-proneness.

I would use design ethnography to determine whether this requirement has been met.

This would involve going and watching people use the programming language, and observing the types of error they make. If people consistently make the same types of error then the programming language clearly encourages them in some way to make this type of error. This would also involve talking to people to find out the *types* of error they *think* they're making. If this error differs to the types of errors they're actually making, we may investigate *why* this is the case. Is this because they spend so much time thinking about this error that they no longer make it? Perhaps the errors they actually make are small but the errors they think they make are significant when

Search / transcription and exploratory design are important parts of programming -- but they're still not the primary thing that you're doing.

Search
A lot of what we do is search for the particular pieces or make sense of what's going on.

Exploratory design:
A lot of the time we test whether things work.

These three tasks tend to be offloaded to the IDE. When we design a language, we design the language only -- we have ot assume the IDE has a lot of additional notation. So we don't have to hyperfixate on this being in the language.

Important in this case! IE don't allow people to do dngerous things
Especially matters in fireworks.

In a complex system, we always work with multiple notations.

Hidden dependencies is a big thing.

Double-check the difference between premature commitment and provisionality.

In Cognitive Walkthrough, you define a set of representative tasks.
How are you going to make an exhaustive list of tasks for a programming language... impossible! It doesn't make sense!

There's no such thing as a representative task.

As soon as you start evaluating representative tasks, you start identifying that the programs themselves and the way you're writing this code becomes a complex system. Is this a user problem, a language problem, a OS problem or a philosophy problem. Is it too verbose, not verbose enough.
This is analagous to what a wicked problem is. Writing code starts becoming analgous to a wicked problem.

Use of a programming language becomes a complex system rather than a simple system.

The language itself is not a complex system. But what you BUILD with it and HOW you program with it becomes remeniscent of a complex system.

Cognitive Walkthrough fails with complex systems. You come up with problems that contradict each other.

This question wants you to relaise this is a complex system and to use an actually useful framework.
CDN (Cognitive Dimensions of Notations).

This defines the two main components of a system.

Notation -- elements of abstraction (buttons, space etc) is a notation. Anything that has meaning is a notation.

The environment is a space (a system in which you can change, edit, manipulate) notation.

It's VERY generic -- because it's meant to be general.

It tries to create the most generic potential framework for evaluating complex systems.

A complex system has no perfect design.

Dimensions: You may improve one. But this will change other dimensions. It was built to remove the case of goal-oriented design, perfect design. Move towards designing against a wicked problem -- you can make things better or worse and priorities (ie these are the 5 most important dimensions).

Notational activities: (searching/ modification etc) -- you have main activities and secondary activities towards which you optimise. IE you may want certain activities to be easiest

Memorise the list for the exam.

You prioritize different activities and dimensions.

they does happen and we should instead focus efforts in ensuring the language doesn't encourage them.

3 2020 Paper 7 Question 9

https://www.cl.cam.ac.uk/teaching/exams/pastpapers/y2020p7q9.pdf

related to your previous Q, lets discuss

You are designing a new syntax for a programming language like Java, with the intention of making it more approachable to students by using English words instead of punctuation symbols.

(a) How does a HCI designer use a theory of human behaviour in their design practice?

Theories of human behaviour can be used to model how humans will interact with an interface and thereby predict the efficiency of it and discover usability problems without the need for expensive human trials.

(b) What is an empirical method? Give an example and short description of that method.

An empirical method is an evaluation method which is based around the observation and measurement of users. They should be used in the late stages of development – observation of users is summative – we discover whether a product works or not but aren't often told how to improve it.

Randomised Control Trials (RCTs) are a form of empirical evaluation. They are used to compare two products (interfaces). I describe A/B tests (a subtype of RCTs).

- Researchers first think of a task which they feel well-represents the question they wish to answer.
- Next, find an unbiased group of representative users.
- Randomly partition them into two groups.
- Each group performs the task on each of the interfaces.
- Finally, collect data about how long it took users, how many mistakes they made etc.
- Significance tests can then be performed to determine whether one of the interfaces was better than the other.

Many large companies use RCTs – they run hundreds or thousands of experiments simultaneously and determine whether or not the change has a positive or negative impact. However, RCTs cannot determine how to make an observed effect bigger (they are only useful for summative evaluation; not formative evaluation).

(c) Describe in terms of the Cognitive Dimensions two trade-offs involved between the existing Java syntax and the new proposed syntax.

- Replacing symbols with words will increase the closeness of mapping and role expressiveness – by replacing symbols with english words which explicitly state the functionality, it becomes very clear to the user what a particular thing does.

  However, this change will increase diffuseness. English words are inherently longer than single symbols. This will increase the time required by experts to write basic programs. Similarly, the consistency is increased – users who are used to other programming languages would find it hard to transition since the syntax is so different.

- the programmers must decide how far to segment symbols – for example `<` has different uses (in generics and bitwise operations). Simply replacing symbols with their names will have no advantage. Rather, the designers must decide to replace

will discuss

a symbol with a set of words in given situations. The decision of how much to segment given symbols leads to some questions.

Increasing the segmentation of symbols increases cognitive load – the programmer must remember more notation. However, it will also decrease the hidden dependencies. If we were to replace brackets with different words for each application of a bracket, the programmer could immedately see what a particular bracket related to – ie) vs `close_function_call`.

(d) How does the programming environment relate to this analysis?

Cognitive dimensions of notations are used for analysis of a particular interface in a **specific use case** by a **specific user**. So if the programming environment is different, the whole analysis would change.

For example, if the use case were for professional programmers, they may find the increased diffuseness an insurmountable downside.

not sure what you mean by 'specific', like real scenario or "type"?

(e) A manager makes a proposal to try different permutations of syntaxes and measure students' performance using each permutation. How would you measure and compare the students' performance?

I would suggest running an A/B test. I would first select a number of representative tasks – these tasks should reflect what students would actually have to use. Since the target audience for this language is students, these could be extracted from real syllabus (ie ticks).

Next, we would give students several lectures in the language (this is for use by students so we wish for it to be similar to the real use-case) before setting them a number of tasks. We would then evaluate the time taken and errors made by the students on each of the syntaxes.

This could be tested for statistical significance and we could use the syntax on which the students performed the best.

(f) What are the likely strengths and weaknesses of the approach in Part (*e*)?

This is not reflective of the case which students would actually use the language. Students use languages for prolonged periods of time.

We're unlikely to get statistically significant results. Subtle changes in syntax are likely to lead to <1% changes in efficiency. We would need tens or hundreds of thousands of students to run this trial to achieve any statistically significant result. There is a very high variance in the ability of students at programming – orders of magnitude. This would dominate the time taken – some students are better programmers than others.

If the tests were done online, the students may use the internet and the results would not test their ability – rather the difficulty of converting code online into this language.

If the tests were done in-person, the experimenters would affect the test. Students could ask for hints or help when stuck – which would skew the results.

The students may not be fully engaged with the test, creating an unrealistic environment. Personally, I would not put effort into learning a language I did not expect would be used anywhere. Many students may share the same sentiment and therefore the test would be invalid.