Human Computer Interaction

Needfinding and Funtional Reactive Programming

Prof. Andrew D. Bagdanov

Dipartimento di Ingegneria dell'Informazione
Università degli Studi di Firenze
andrew.bagdanov AT unifi.it

November 22, 2017

Outline

News

News

Latest and Greatest

- On Friday, December 1, we will discuss the following paper:

  N. Cila, I. Smit, E. Giaccardi, B. Kröse, "Products as Agents: Metaphors for Designing the Products of the IoT Age." In: Proceedings of CHI, 2017.

Project Information

- I have uploaded sample projects to a dedicated topic in the course Moodle.
- There is also a link to the Google Document for the sample project document template there.

Let's talk projects

Let's talk projects

- How is everyone progressing in terms of narrowing down a project?
- I will propose another two projects today (and revisit one from last week, after we see what FRP is all about).
- But first, what are your thoughts?

Proposal: phone localization for TUI

- In this project we will experiment with techniques for determining the location and orientation of smartphones placed on top of interactive tabletops.
- We will begin with a literature survey on low-level techniques for using structured information hiding to visually communicate location information.
- We will experiment with different spacer techniques to ensure optimal positioning of the camera for detecting pixel patterns used for communicating localization information.
- You could think of this project as using steganography for phone localization on interactive tables.

Proposal: mouse endpoint prediction

- In this project we will implement the endpoint target prediction algorithm described in:

  P. Pasqual, J. Wobbrock, "Mouse pointing endpoint prediction using kinematic template matching." Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, 2014.

- The idea behind this work is to predict which UI component the user will eventually click through analysis of mouse trajectories.
- This type of predictive technology can be used to customize UI behavior.
- In this project we will reproduce the testing protocol described in the article using a simple Kivy application.

Tips for zeroing in on a project

- Try to determine what type of technology interests you the most:
  - a more technical, experimental project;
  - a project centered around an embedded application;
  - a project centered around a mobile application; or
  - a project centered around a web application;
- If you can at least begin to narrow down the possibilities, I can help you a lot more.

Needfinding: Tools and Techniques

NF: Tools and Techniques

- As we saw last time, there are two basic kinds of needfinding techniques: observations and interviews.
- The most important thing about needfinding in design is that we look without knowing what we are looking for.
- We must trust that our ability to define the problem will emerge during the need finding process.
- Needfinding is an essential aspect of Human-centered Design because we explicitly look to users for design inspiration.

NF: Tools and Techniques

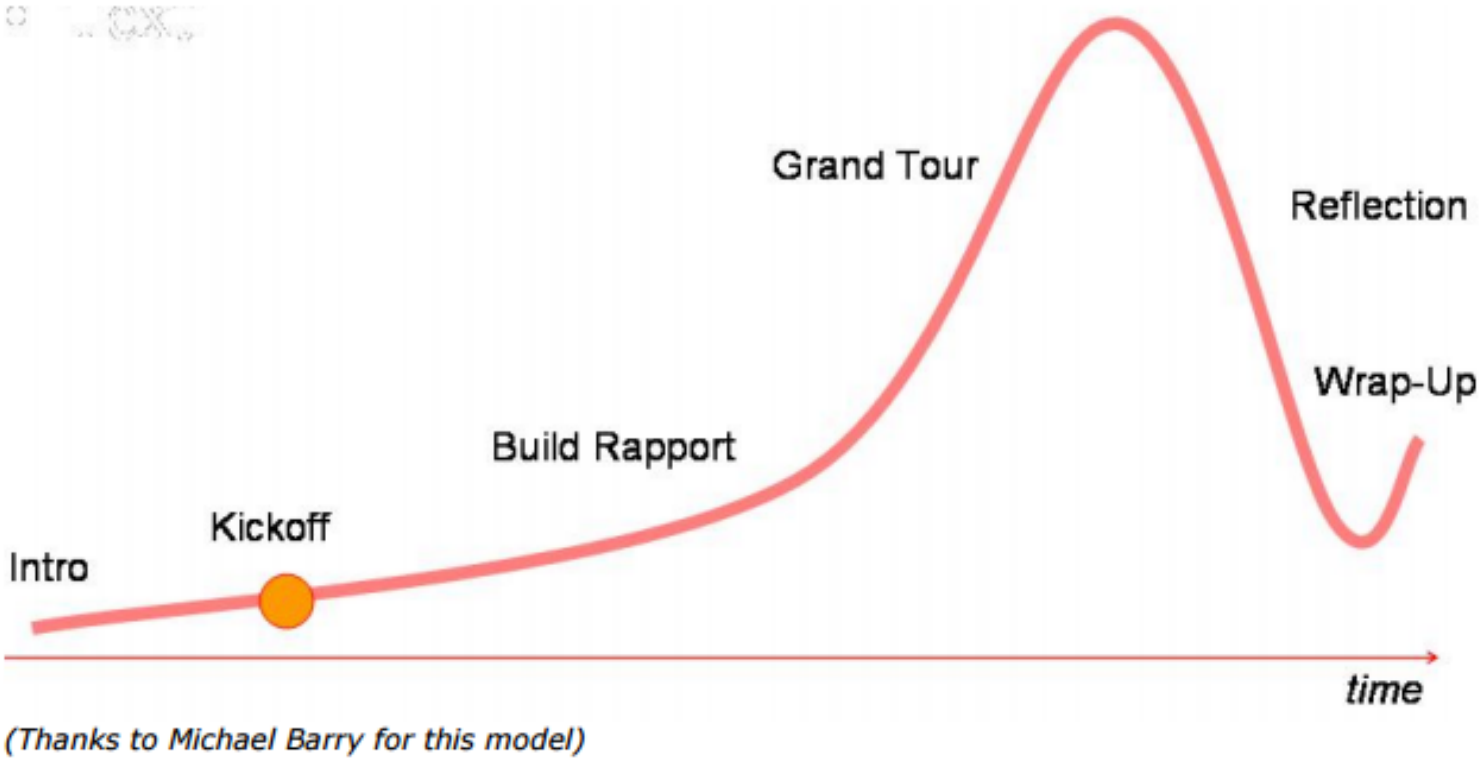
- At the highest level, good needfinding techniques need:
  - Good subjects: interesting, open, expert, unusual, observant, etc.
  - Good environments: in situ, friendly, "safe", etc.
  - Structure (but also flexibility): have a plan and goals, but pursue new opportunities.
  - Great stories: capture images, quotes, etc. that bring your interview to life.
  - Iterative designs: iterate interview structure or observation techniques based on results.
  - Avoid Hawthorne effect: distortion of behavior due to observation.
  - Room for silence: if you listen, they will speak.

NF: Purpose and Process

- Through needfinding we want to:
  - Uncover Latent Needs: Gaps in Use, Usability and Meaning. We look for surprises, differences between what people say they do and what actually do.
  - Gain Empathy for Users: the emotions that guide behaviors.
  - Look for Extreme Users: users who are pushing the system may reveal needs before the mainstream.
- The process involves bringing back Stories and Artifacts (photos, drawings quotes) that communicate your insights.

NF: Interviewing

- Visually, the interview process can be imagined like so:



(Thanks to Michael Barry for this model)

NF: Interviewing

- Here are some truncated examples of questions for each stage:
  - Introduction: "Hi, I'm a student studying coffee. I'm interested in hearing about your experience with coffee. There are no right or wrong answers, I just want to hear what you have to say."
  - Kick-off: "Do you drink coffee?"
  - Build rapport: "Did you have a coffee today? How was it? Do you have a favorite coffee?"
  - Grand Tour: "Can you describe your most memorable coffee experience? Why was it so unique? What happened?"
  - Reflection: "If you were designing the ultimate coffee shop based on your ideal experience. . . "

NF: Observation

- Observation is the other main needfinding technique:
- What it is: viewing users and their behavior in context.
- When to do it: when you want to see users in their element and learn about their experience
- How to do it:
  - Deep hanging out: spend time in the vicinity of the subjects
  - Walk in the subjects shoes: assume the role of the subject
  - Ask for a tour from an insider
  - Paparazzi: observe and photograph anonymously
  - Other: security cameras, head cameras for subjects, etc.
- The most important thing about needfinding in design thinking is that we look without presupposing what we are looking for.

NF: Lead User Interviews

- Lead user interviews are a great way to get deep and informed input right from the start.
- What it is: interviews with fanatics (the most rabid consumers of an experience).
- When to do it: when you want to see the future of usage, or understand an experience from the perspective of its most critical subject.
- How:
  - Look for the most extreme users (use your network, look for blog owners, etc).
  - Make contact, and state your interest in them and their views
  - Ask questions that are more open ended and blunt "Why do you think people pay more for coffee now than they did in the past?" "What do you think we need to know?"
  - Engage them in an ongoing way: lead users make great testers for your prototypes.

NF: Expert Interviews

- Expert interviews are another way to inform design right from the beginning.
- What they are: Interviews with subjects who have domain expertise.
- When to do it: when you need to come up to speed on the context of your design quickly.
- How to do it:
  - Ask open ended questions that allow them to educate you. "I am a designer, so I don't really understand coffee chemistry. As a food chemist, can you explain to me how coffee 'works'?"
  - Try to understand their role in the user's experience. "If I come in to your store to get a coffee, what, as the barista, do you do?"
  - Gather ideas for needs that experts may uniquely see (e.g. "What bothers you most about you job?").

NF: Surveys

- Surveys are an excellent way to gather quantitative information about a design domain.
- What they are: a series of carefully structured questions that can be used to draw conclusions about user attitudes, likes and desires.
- When to use them: when we want to get many perspectives rapidly or quantify insights
- How:
  - Identify screener (criteria for filtering) for subjects.
  - Develop and sequence questions.
  - Include explanation of survey in survey instructions.
  - Provide progress updates through survey.
  - Ask at least one open ended question.
  - Test the survey before releasing it.
  - `http://www.surveymonkey.com` and other tools are very useful.

NF: Advice on Observing People

1.

Cast aside your biases, listen and observe

Let subjects tell their own story, and listen for the things that elicit emotion, cause them concern or frustration.

"If you want to find out what people really need, you have to forget about your problems and worry about their lives."

NF: Advice on Observing People

2.

Note the contradictions between what people say and what they do

Opportunities for innovation lie within the disconnect between action and words.

NF: Advice on Observing People

3.

Listen to people's personal stories

Let them relate their successes and failures.

Stories encompass the implicit rules that govern and organize peoples lives and reveal what they find normal, acceptable and true. They reveal moral codes, sources of pride, shames, shoulds and should-nots.

NF: Advice on Observing People

4.

Watch for "work arounds"

People make do and work around the shortcomings of products and situations.

In everyday life, we all come up with "work arounds," clumsy or clever, that we usually are totally unaware of.

You must take note.

NF: Advice on Observing People

5.

Distinguish between needs and solutions.

Needs open up possibilities, solutions constrain them.

If you start with a solution then you may overlook the possibility of coming up with an entirely new and revolutionary product or service.

NF: Advice on Observing People

6.

Look beyond the obvious.

Your research may seem so routine and familiar that you feel there is nothing new to be learned.

Boredom and frustration easily set in. Stay alert.

The epiphanies and insights emerge from the nuances.

Needfinding: Working with Personas

NF: Personas

- Personas are humanized descriptions of users.
- They are useful for anything that has to do with making stuff or selling stuff.
- In marketing classes, we learn that the way you describe a market is something like "males 25-35 with incomes over $30,000/year".
- But who is that guy? How would you figure out what innovative new product or feature he might like?
- Personas are used extensively in marketing, but they are an extremely useful tool for identifying archetypal users of your systems.
- Here I will give some general advice for building personas for your design process.

NF: Personas

- Good personas tell a story.
- This isn't to discount the importance of being rigorous and quantitative, it's to anchor that rigor in a meaningful starting point and identify the right tools for that starting point.
- Avoid bullet points: this "plastic persona" you see is the opposite of what you want.

- Women
- Age 28-45
- Have kids
- Socialize with other mom's
- Online with Facebook
- 86% said they'd like to be more organized
- 70% said they'd use an application that organizes them

NF: Personas

- The photo is clipart from the Internet, the description is a set of generic bullet points and they don't say much of anything about who this person really is or what they need.
- Get real photos, create a collage of real stuff and if you're saying that your target persona Tweets or posts certain types of content online, get examples.
- If you're saying that they hate doing paperwork, get samples of the kind of paperwork they have to do.
- Write full sentences that deliver a narrative.

NF: Personas

- The 'organic persona' is a better start: there's an actual paragraph of description, and that the photo was taken with an iPhone out in the field, which is where you should be when you're developing personas.
- One of the top failure modes of persona creation is that they have their humanity peeled away or just never acquire it.
- Giving them a name is a trick that helps humanize them.

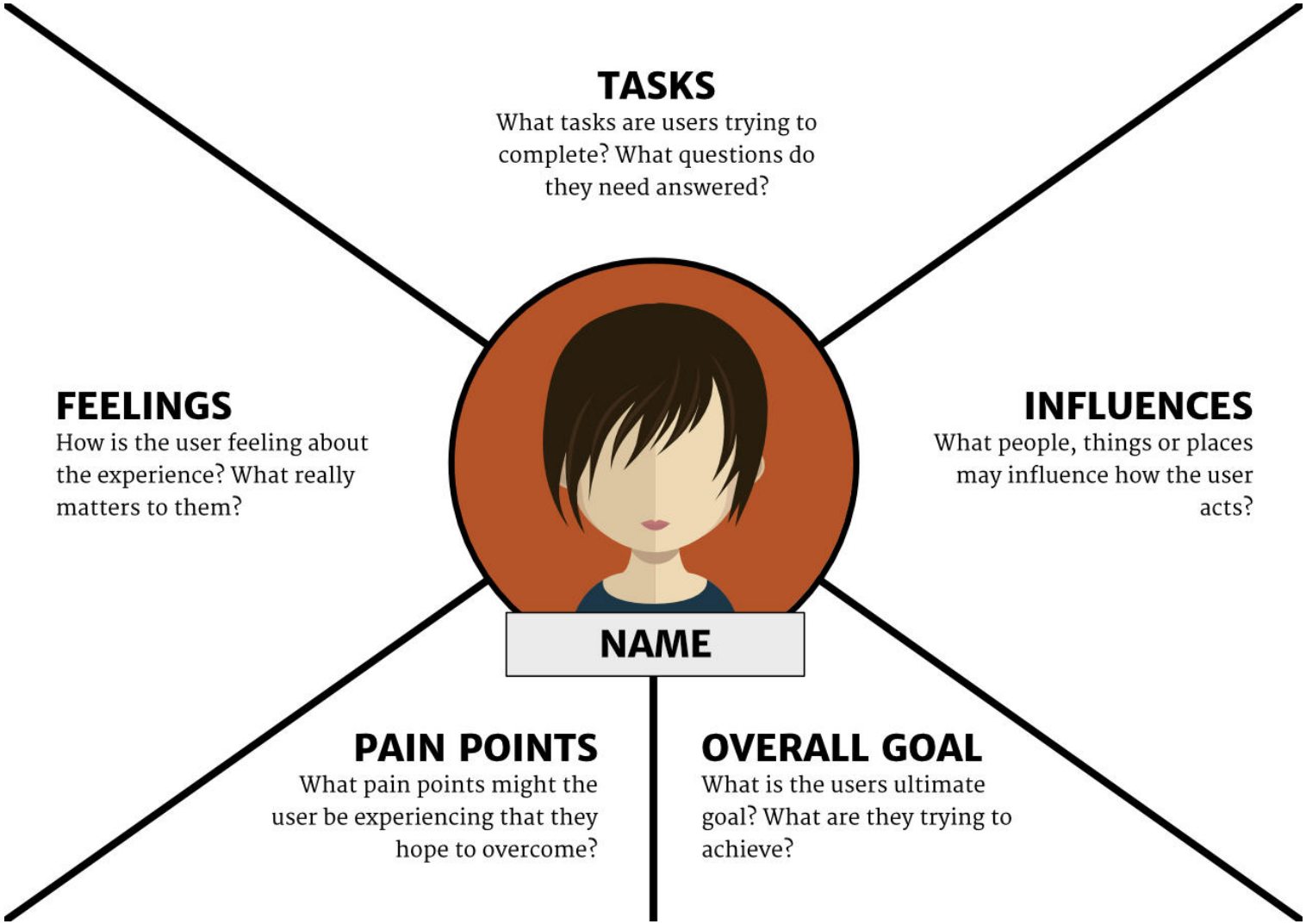
NF: Personas

- Once you've humanized your persona, you'll want to operationalize it in your particular area (online banking, dating, shopping for power tools, whatever).
- The Think-See-Feel-Do checklist is a good way to frame this.

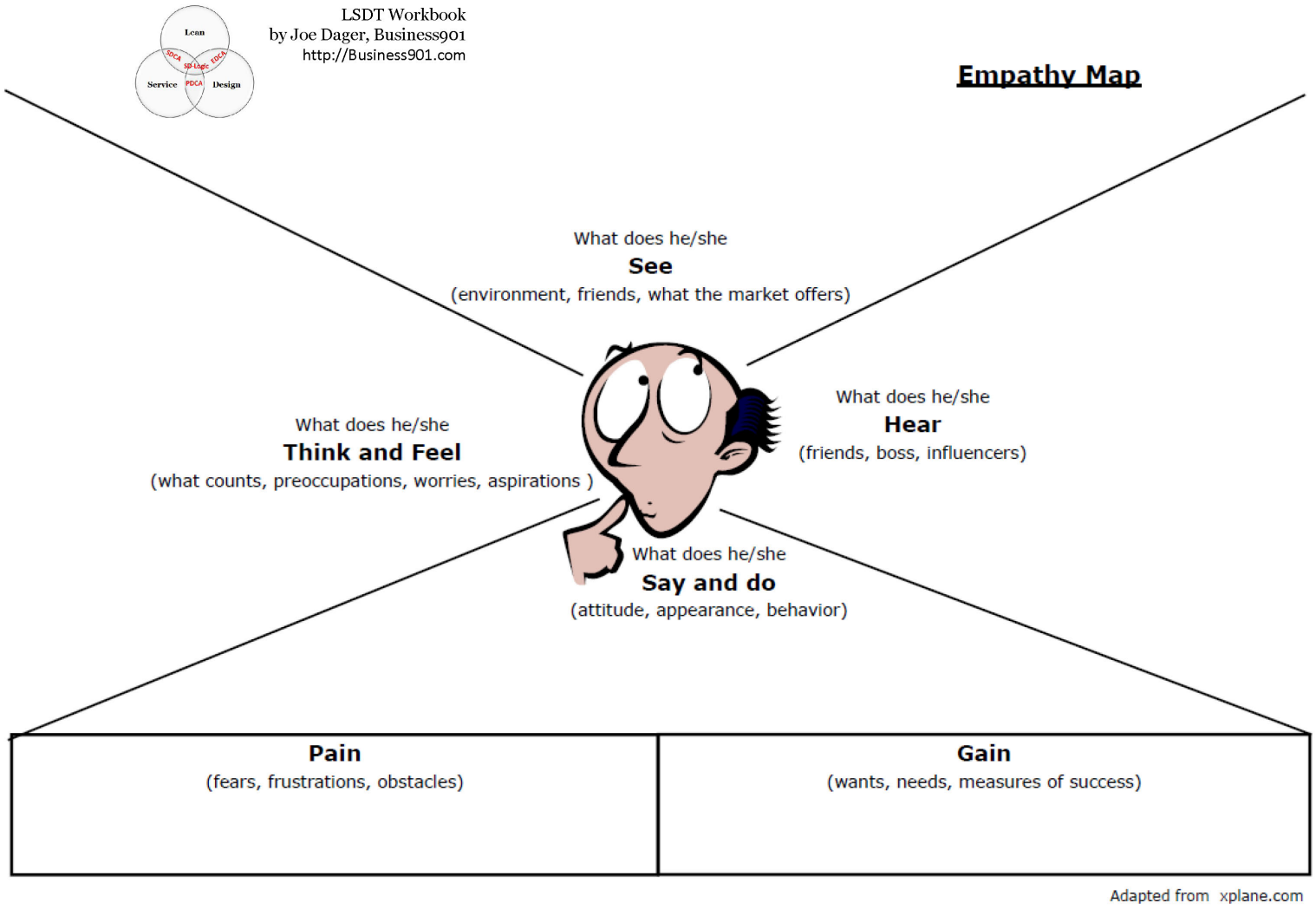
THINK	What is their point of view on your area of interest? What do they like, dislike about it? What's the difference in their view between how it is and how it should be?
SEE	What influences and informs them about your area of interest? Where do they get that? Peers? Media?
FEEL	What are the underlying emotional drivers in the area? How does that influence what they do and their interest in alternatives?
DO	When you observe them, what do they actually do?

- This website has excellent resources for working with personas.

NF: Empathy Maps



NF: Empathy Maps



Functional Reactive Programming (FRP)

Functional Reactive Programming (FRP)

- Functional Reactive Programming grew from a sequence of basic observations:
  - Real programs have to deal with the real world.
  - The real world is constrained by time.
  - The real world produces unpredictable inputs to your program.
  - Usual solution: Callbacks and event loops.
- Can we extend the benefits of functional programming to handling these characteristics of the world?
- Without jumping off into a huge tangent, suffice it to say that functional programming is a paradigm that concentrates on the functional abstractions of programs, in contrast to object-oriented programming which concentrates on the data objects and encapsulation.
- There has been a minor functional programming renaissance with the surge of interest in languages like Haskell, Clojure and Scala.

Functional Reactive Programming

- FRP divides inputs into two basic classes:
  - Behaviors or signals: Functions of time.
  - Events: Temporal sequences of discrete values.
- An FRP language must include a means of altering or replacing a program based on event occurrences.
- This is the basis of FRP's reactivity.
- These abstractions may be reified in an FRP language or may form the basis of other abstractions, but they must be present.

The Reactive Manifesto

- The Reactive Manifesto begins:

> *"Organisations working in disparate domains are independently discovering patterns for building software that look the same. These systems are more robust, more resilient, more flexible and better positioned to meet modern demands.*
>
> *[. . .]*
>
> *We believe that a coherent approach to systems architecture is needed, and we believe that all necessary aspects are already recognised individually: we want systems that are Responsive, Resilient, Elastic and Message Driven. We call these Reactive Systems.*
>
> *Systems built as Reactive Systems are more flexible, loosely-coupled and scalable. This makes them easier to develop and amenable to change. They are significantly more tolerant of failure and when failure does occur they meet it with elegance rather than disaster. Reactive Systems are highly responsive, giving users effective interactive feedback."*

The Reactive Manifesto

- Reactive Systems are:
  - Responsive: The system responds in a timely manner if at all possible. Responsiveness is the cornerstone of usability and utility, but more than that, responsiveness means that problems may be detected quickly and dealt with effectively.
  - Resilient: The system stays responsive in the face of failure. This applies not only to highly-available, mission critical systems — any system that is not resilient will be unresponsive after a failure.
  - Elastic: The system stays responsive under varying workload. Reactive Systems can react to changes in the input rate by increasing or decreasing the resources allocated to service these inputs.
  - Message Driven: Reactive Systems rely on asynchronous message-passing to establish a boundary between components that ensures loose coupling, isolation and location transparency. This boundary also provides the means to delegate failures as messages.

Reactive Programming

- This philosophy of programming and architecture design has been embraced by parts of the web application and GUI development communities.
- Though it is being championed from many corners, it can be difficult to nail down a precise definition of Reactive Programming.
- The Reactive Manifesto is written to convince managers, and the definition from Wikipedia is far too generic:

  > *In computing, reactive programming is a programming paradigm oriented around data flows and the propagation of change. This means that it should be possible to express static or dynamic data flows with ease in the programming languages used, and that the underlying execution model will automatically propagate changes through the data flow.*

- And Microsoft has a typical Microsofty defintion: "Rx = Observables + LINQ + Schedulers". Thanks, MSDN...
- If we try to tear it down to its essence, we arrive at something like:

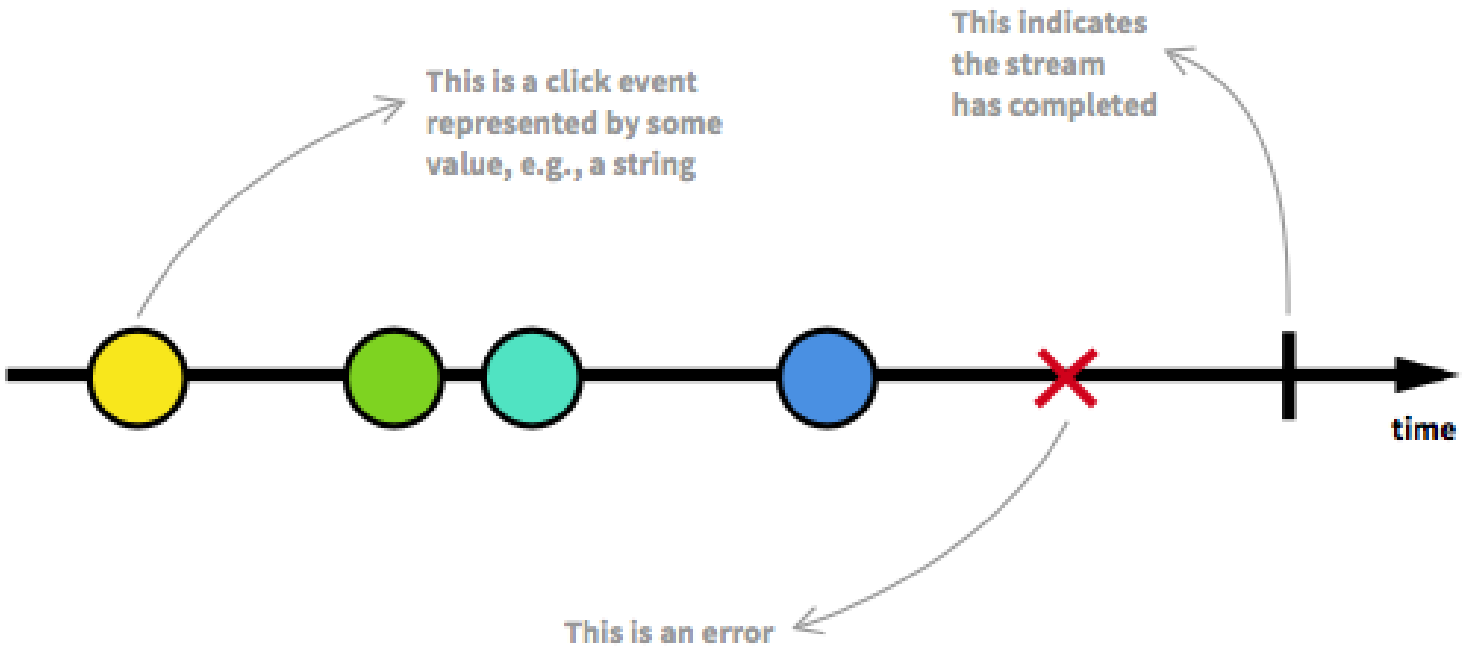
  Reactive programming is programming with asynchronous data streams.

Reactive Programming

- This really isn't anything new – event buses or your typical click events are really an asynchronous event stream, on which you can observe and do some side effects.
- Reactive programming is this idea taken to the extreme: you can create data streams of anything, not just from click and hover events.
- Streams are cheap and ubiquitous, anything can be a stream: variables, user inputs, properties, caches, data structures, etc.
- For example, imagine your Twitter feed as a data stream in the same fashion that click events are.
- You can listen to that stream and react accordingly.

Putting the "F" in RP

- Importantly: in Reactive Programming you are given a rich toolbox of functions to combine, create and filter any of those streams.
- This is where the functional magic kicks in: a stream can be used as an input to another one, and even multiple streams can be used as inputs to another stream.
- You can merge two streams, you can filter a stream to get another one that has only those events you are interested in
- You can map data values from one stream to another new one.



Functional Reactive Programming

- A stream is a sequence of ongoing events ordered in time.
- It can emit three different things: a value (of some type), an error, or a completed signal.
- Consider that the completed takes place, for instance, when the current window or view containing that button is closed.
- We capture these emitted events only asynchronously, by defining a function executed when a value is emitted, another function when an error is emitted, and another function when completed is emitted.
- Sometimes these last two can be omitted and you can just focus on defining the function for values.
- The "listening"' to the stream is called subscribing, the functions we are defining are observers, and the stream is the subject (or observable) being observed.
- This is precisely the Observer Design Pattern.

- Sometimes it is convenient to use an ASCII shorthand notation for drawing FRP diagrams and reasoning about stream processing logic:

```
--a---b-c---d---X---|->

a, b, c, d are emitted values
X is an error
| is the 'completed' signal
---> is the timeline
```

- This should all feel very familiar already.
- Let's do something new: we are going to create new click event streams transformed out of the original click event stream.

Functional Reactive Programming

- First, we will make a counter stream that indicates how many times a button was clicked.
- In most Reactive libraries, each stream has many functions attached to it, such as map, filter, scan, etc.
- When you call one of these functions, such as `click_stream.map(f)`, it returns a new stream based on the click stream.
- It does not modify the original click stream in any way.
- This is a property called immutability, and much of the theoretical foundation of FRP is based on immutability.
- This allows us to chain (compose) functions like `click_stream.map(f).scan(g)`.

- Our stream diagram might look like this:

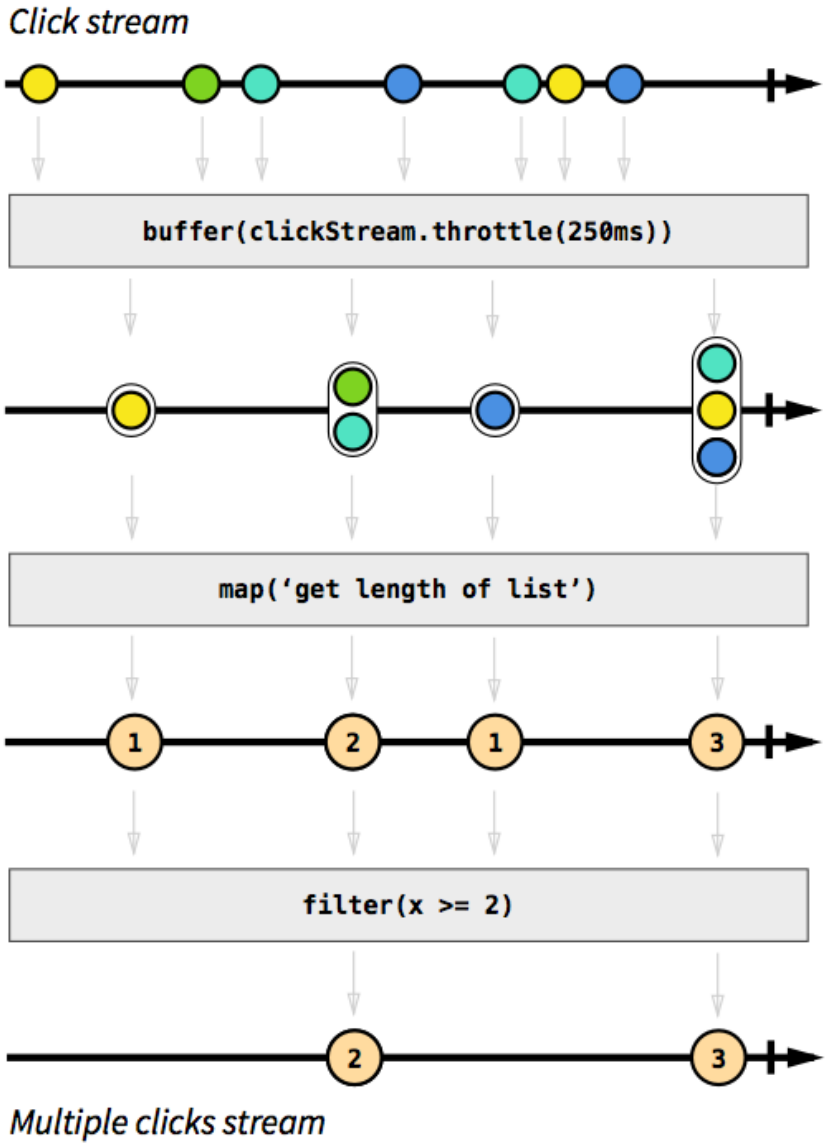
```
click_stream:    ---c----c--c----c------c-->
                 vvvvv map(c becomes 1) vvvv
                 ---1----1--1----1------1-->
                 vvvvvvvvv scan(+) vvvvvvvvv
counter_stream:  ---1----2--3----4------5-->
```

- The `map(f)` function replaces (into the new stream) each emitted value according to a function `f` you provide: in our case, we mapped to the number 1 on each click.
- The `scan(g)` function aggregates all previous values on the stream, producing value `x = g(accumulated, current)`, where `g` was simply the add function in this example.
- Then, counter-stream emits the total number of clicks whenever a click happens.

Functional Reactive Programming

- To illustrate the power of FRP, say that you want to have a stream of double click events.
- To make it even more interesting, let's say we want the new stream to consider triple clicks as double clicks, or in general, multiple clicks (two or more).
- First, let's imagine how we might do that in a traditional imperative and stateful fashion.
- It doesn't take much thinking to see it's fairly messy and involves variables to keep state and some fiddling with time intervals.
- Well, in FRP it's pretty simple: in fact, the logic is just 4 lines of code.
- But let's ignore code for now: thinking in diagrams is the best way to understand and build streams.

Functional Reactive Programming



Functional Reactive Programming

- Grey boxes are functions transforming one stream into another.
- First we accumulate clicks in lists, whenever 250 milliseconds of "event silence" has happened (that's what `buffer(stream.throttle(250ms))` does, in a nutshell.
- The result is a stream of lists, from which we apply `map()` to map each list to an integer matching the length of that list.
- Finally, we ignore 1 integers using the filter(x >= 2) function.
- That's it: 3 operations to produce our intended stream.
- We can then subscribe ("listen") to it to react accordingly how we wish.
- This example is just the tip of the iceberg: you can apply the same operations on different kinds of streams, and there are many other stream manipulation methods.

The real "F" in FRP

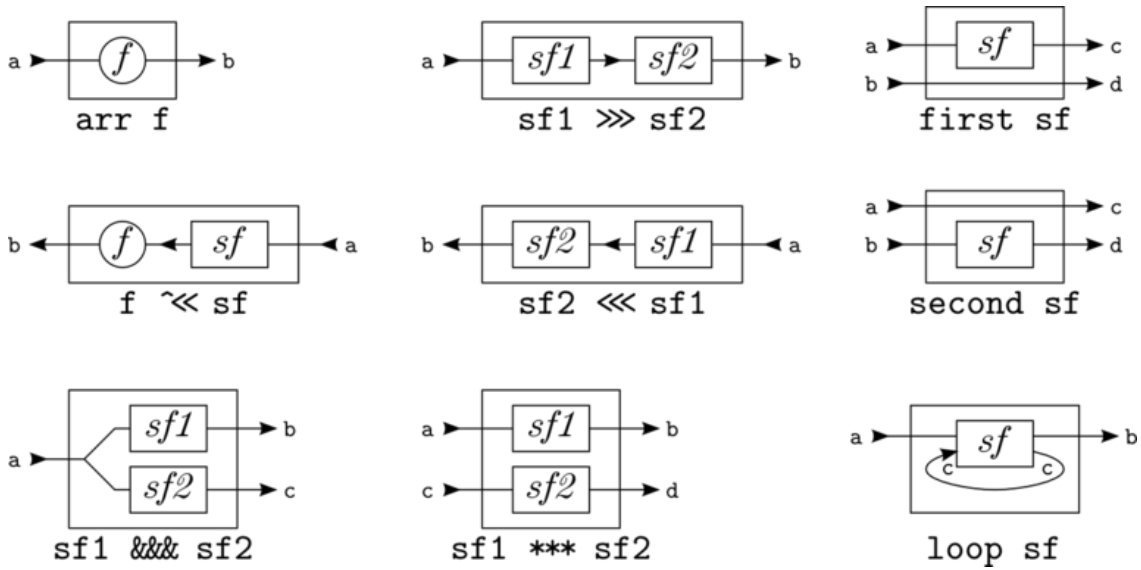
- Now is probably a good time to comment on the origins and some of the ongoing polemic surrounding FRP.
- FRP has taken many forms since its introduction in 1997, and there is much debate about what FRP is and what FRP isn't.
- One axis of contention is discrete vs. continuous semantics:
  - Discrete time semantics: formulations such as Event-Driven FRP and Elm require that updates are discrete and event-driven. These formulations have pushed for practical FRP, focusing on semantics that have a simple API that can be implemented efficiently in a setting such as robotics or in a web-browser.
  - Continuous time semantics: the earliest formulation of FRP used continuous semantics, aiming to abstract over many operational details that are not important to the meaning of a program. A key property of this formulation is modeling values that vary over continuous time, called behaviors or signals.

The real "F" in FRP

- This semantic model of FRP in side-effect free languages is typically in terms of continuous functions, and typically over time.
- These distinctions are largely unimportant for us, though it can be useful to understand the origins of FRP.
- The original goal of FRP was to develop programming systems in side-effect free languages like Haskell so that it is easier to reason about the semantics of time-dependent programs.
- Purists, in fact, insist on distinguishing between Reactive Systems, and *Functional* Reactive Systems.
- To be truly Functional, time semantics should be continuous, and referential transparency preserved by avoiding side effects.
- In any case, most of the concepts and terminology translate into Reactive Programming systems (including functional aspects like `map` and `scan`).

Advantages of FRP

- One of the main advantages of FRP is the composable nature of streams and stream processing.
- Just having event streams is already an advantage: you still will attach listeners to them in the end to execute effects, but between the source and the destination you have a powerful abstraction.
- Event streams have a lot of higher-order functions to easily deal with them, and for composing them without writing massive amounts of error-prone boilerplate code.



FRP: Where to go from here?

- There is an abundance of information about Reactive and Functional Reactive Programming.
- For a basic motivational overview, I highly recommend this technical report on the limitations of the observer pattern.
- We will have one final laboratory on FRP, and we will use the RxPy reactive programming system:

  `https://github.com/ReactiveX/RxPY`
- Not to mention Microsoft's own Reactive Extensions:

  `https://msdn.microsoft.com/en-us/data/gg577609`
- Under Windows (if you're into that sort of thing), there is a strong community behind ReactiveUI:

  `http://reactiveui.net/`
- And for Web application development, RxJS:

  `https://github.com/Reactive-Extensions/RxJS`
- Or a general, multi-platform entry point:

  `http://reactivex.io/`

A project proposal, revisited

FRP bindings for Kivy (revisited)

- In this project we will implement a binding between the an implementation of Functional Reactive Programming and Kivy (this will make more sense after next Monday's lecture).
- We will use as a basis the RxPy implementation of FRP: https://github.com/ReactiveX/RxPY
- We will then investigate different methods of integrating RxPy into the Kivy scheduler.
- We will implement selected method(s) according to the Kivy coding style and best practices, including unit tests where appropriate.
- Finally, we will implement a series of example programs showing off the use and features of our FRP in Kivy.
- The finished package, after evaluation, will be published on Github under a Kivy-compatible open source license.

Homework

Homework

Exercise 17.1: Zeroing in on a Project

During the weekend, spend some time thinking about a final project for the course. Try to determine:

1. Will it will be an individual or group project?
2. Which type of project: technical, standard, scientific, or some affine combination of the three?
3. What might be a source of users to select interviewees from?