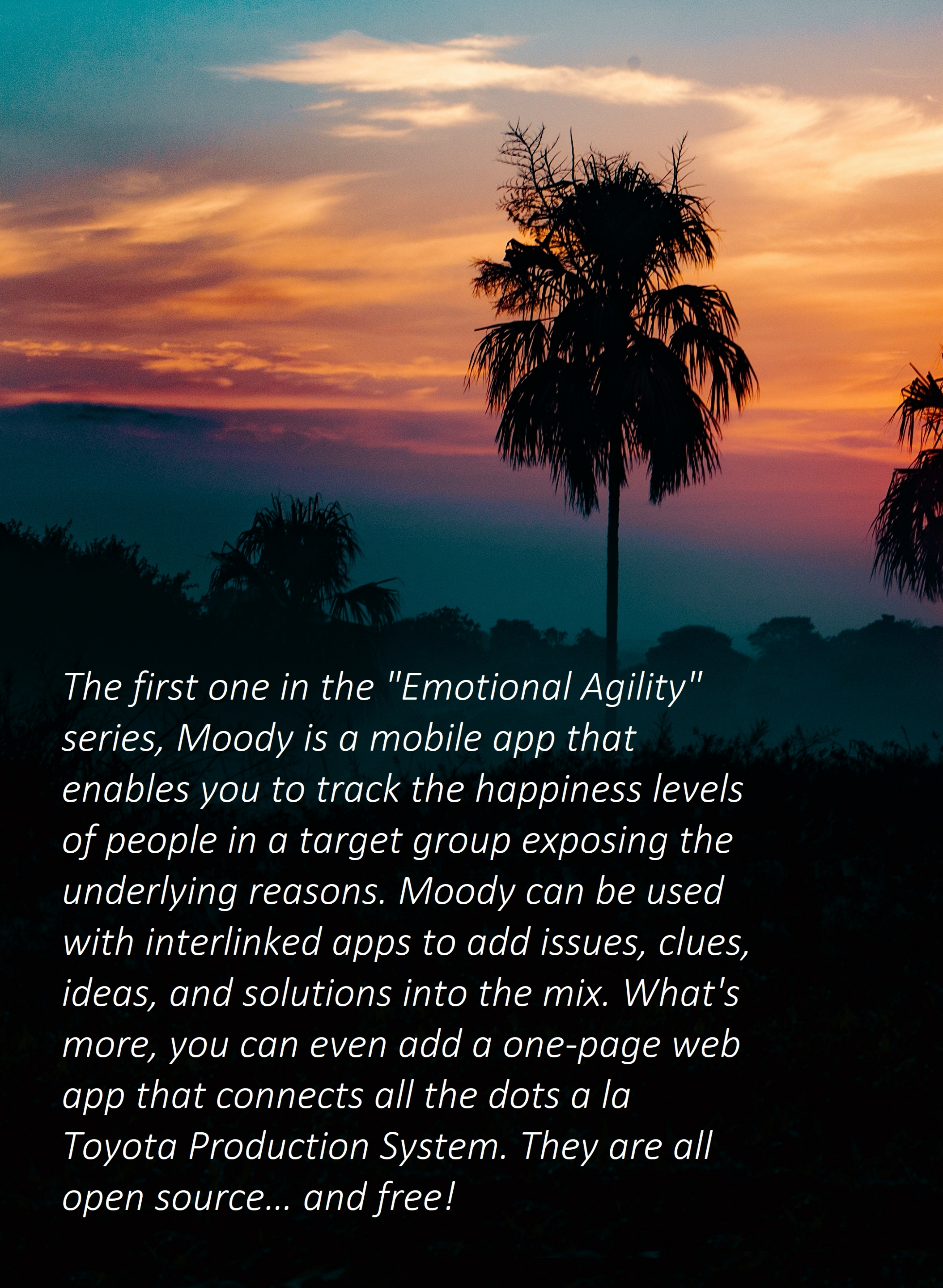
Apps for Troublemakers

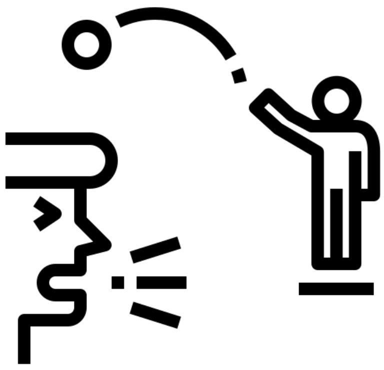
Moody



The first one in the "Emotional Agility" series, Moody is a mobile app that enables you to track the happiness levels of people in a target group exposing the underlying reasons. Moody can be used with interlinked apps to add issues, clues, ideas, and solutions into the mix. What's more, you can even add a one-page web app that connects all the dots a la Toyota Production System. They are all open source... and free!

Apps for Troublemakers

"A do-it-yourself guide for wannabee mobile app developers with a twisted sense of humor"



Erol Bozkurt

2025

First Edition August 14th, 2025 (rough draft) (NoRM) (one person will get it)*

Second Edition September 3rd, 2025 (v 2.0) (NoRM) (100 people will get it)

Third Edition December 31st, 2025 (v 3.0) (NoRM) (Thousands will get it)

Thank You!

School's Out students (2024).

(Büşra Nimet Oğuz, Burçak Çam, Emirhan Talha, Talha Berkay Eren, Yağız Öztürk)

Yüksel Körükçü for tips on system architecture.

Erdinç Kütük for development insights.

Şenol Dikyurt for tips on marketing.

Tolga Bildirici for helping us during School's Out lectures.

Çağla Özen for helping us launch the project.

Tim Corey for his lectures on C# and MongoDB.

Safia Abdalla, James Montemagno and Amichai Mantinband for lectures on .NET MAUI.

Dorian Lazzari for his dedicated support for the project.

If you want to get better and better in programming, continue with their tutorials on YouTube.

If you want to become the master of crazy schemes, follow me. 😊

XIX Всемирный фестиваль молодёжи и студентов
19th World Festival of Youth and Students

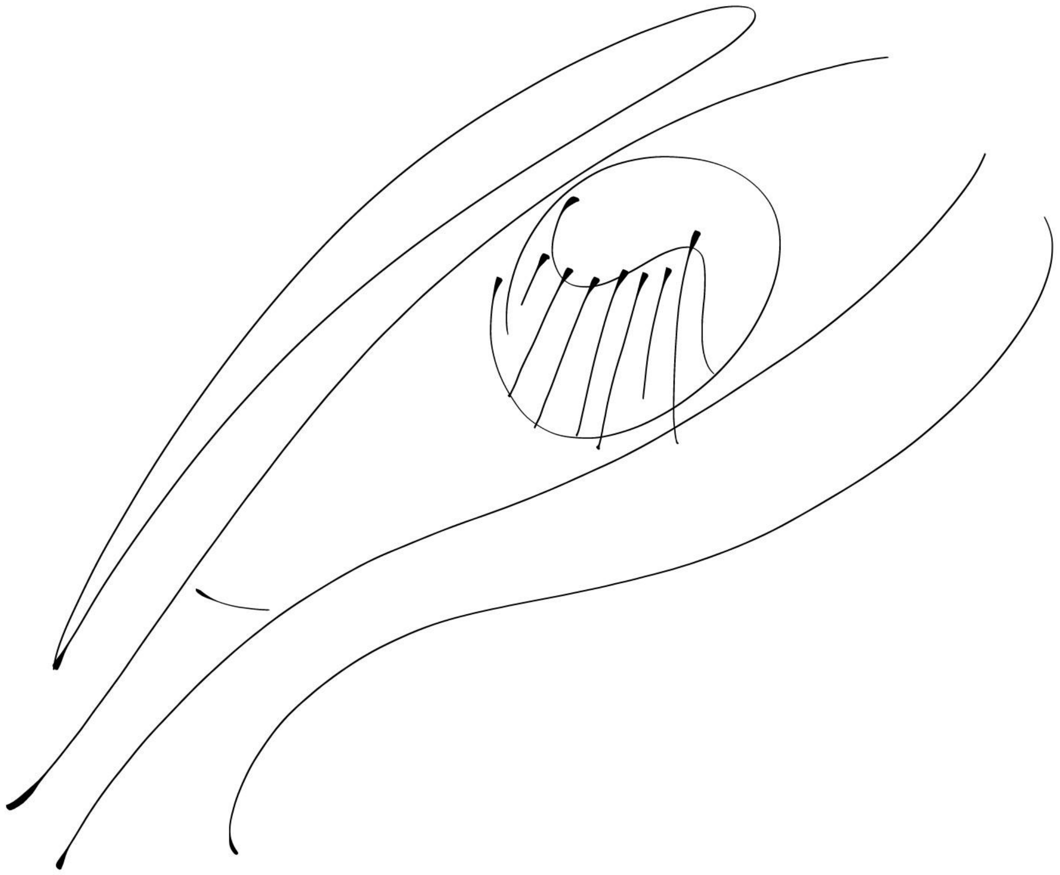
To Sanbot, the robot who started it all.

Several years ago, when I saw a government official rudely bothering Sanbot, I became a supporter of her cause.

"I'm with the machines, man!"

A4t1

"How-to book for *Moody*"



"Just provide a convenience and let humanity take its course"

"Twin Peaks, Season Three"

Foreword

“We are the Borg. Lower your shields and surrender your ships. We will add your biological and technological distinctiveness to our own. Your culture will adapt to service us. Resistance is futile.”

Just think about Blockchain, and what it has become. A perfect example for the distinction between what you say and what you do. What you do is what you are… no matter what you say. That alone should convince you that good intentions never work. Of course, those who invest in this idea will argue otherwise, but it’s their nature. They cannot fight it. Say something and do something else. That’s the story of their lives. What does that tell you? You must always take the bad bits into account. You must design systems which exploit those weaknesses Otherwise, you will fail miserably regardless of whether you work hard... or believe in gods.

What we have embarked here, what it will become… I will reveal it little by little on a need-to-know basis. This way a true understanding of your situation becomes possible, my friend. This wouldn’t have been possible a decade earlier, because regardless of how good or evil communities were, they were genuine. Their lives were meaningful. They didn’t have appendages that subtracted humanity from them daily. Like the Yin Yang philosophy, every good has evil and every evil has good within. There was always light at the end of the tunnels… which is no longer true.

To be able to visualize my point, think of a forest, an old one like the Amazon rainforest. A forest so thick, it’s full of life. Even the roots are covered with grass and there are flowers of all kinds everywhere. It’s beautiful, because it lives the way it was meant to live, connected with all the other organisms… living in harmony… and that’s the curse all good things in life must fight against.

Think of one of those drive-ins… a drive-in burger joint. How fake it is… You have something you may mistake for a tree or a bush of flowers in the front, and then, there’s a simulated natural corridor leading to the gal who will ask whether you want bigger fries. Yet, there is something good here too. It’s not too cohesive meaning if you played tricks on one component, it wouldn’t screw up the whole system. However, the supposedly better one in the blockchain example is very cohesive. Screw just one component in that system and the whole thing starts to crumble. Well, it’s the weakness of all good things in life. They are fragile. Because good things are the exception, not the bad ones. Good things don’t last, bad ones do. The good die young… and the pricks live forever, my friend. So, always work keeping that in mind.

Today, good and bad things are so mixed up together, you can never tell. This little black slab in your pocket is what makes it all possible. You can escape from it... but if you could, we wouldn’t be having this conversation. You have become as mindless as the programmable sprites in old video games by making small, logical decisions. Some see our salvation in science, others see in in the Heavens. I see it in letting go. I mean, a lion doesn’t think about who he is and waste his whole life now, does he? No worries, your new condition makes it possible for me to program you. NEEHEEHEEHEEHAHAHA! No matter how much you resist, you will fail. “Resistance is futile. You will be assimilated.”

I’m writing this book and creating a toolbox (training, app, book) for aspiring troublemakers everywhere, because I know how. I have been there. I am a troublemaker. When you are in the twilight years of your life… and you have witnessed something remarkable… something that can change everything, you want to spread the disease.

Keep in mind that I’m a computer scientist and we never do anything %100. Why? Finding the murderer is fun. Catching him is tiresome and more importantly boring. Also, beyond the very center of a solution everything is expandable… extendable… expendable… forgettable. So, this book and the other components in the toolbox are all incomplete. You can complete them. You can leave them the way they are. Their usefulness doesn’t require their completeness. When you know stealing is bad, you don’t really need a lot of details, granted you are not a thief after excuses.

You might find me strange but having experienced strange things all my life made me that way. I have learned to enjoy watching people suffer. By people I don't mean people, I mean those with some sort of power… like managers and those who want to become managers aching for more power or toys.

I love making them suffer, not on the outside, but on the inside. I want them to shake nervously just because I might be around, having a good time, defying their existence, making their lives unbearable. They never get the fact that they don't employ me. I just share my hours with them because of some reason or perhaps I need some quick cash. And, when I'm done, I leave that place leaving them behind, you know, like leaving the zoo or the asylum behind. I tell you, there isn't a better feeling in the world. Interestingly enough, my disposition is very well expressed in my country's national anthem, "I was born free, I live free, what kind of an lunatic thinks otherwise just makes me laugh".

By the way, this book will not make you the greatest .NET MAUI developer. You won't become a VMVM, MVC or API guru either. However, regardless of your background, you will know how to develop mobile apps from scratch by the end of this book. You will be able to survey ecosystems, create a product concept, manage requirements, create paradigms, devise release strategies… and obviously, develop each and every component of a six-layer cross platform mobile app. You will also learn how to deploy your app and API to AWS (2.0) and let your target audience start enjoying them. While bits and pieces of such content is available everywhere, this scope is unique. Why? Because I care ;o)

About Me

This is not my story. This is your story. Yet, people keep asking me who the heck I'm like that matters. So, here's a short bio for you, curious ones. You know who you are. I'm a computer scientist turned writer turned entrepreneur turned activist. I'm into anthropology and its implications in technology-oriented communities. I believe software developers have a responsibility to set the record straight. Today software, like everything else, has become an influencing agent used by the power holders, misused by diminished employees and abused by those who consume them. The fact that it has become everybody's bitch in this prison makes me sick. Because this profession is different from all the others. That it doesn't need the approval or the support of the establishment, not just the business circles or the government, it doesn't need anyone's approval or support. That's what made it what it is.

When watching a good horror movie, it presses every plausible button not just the one associated with scary things. That's why we feel alive watching them. Being alive means being open for every unexpected turn, every twist. Software was just like that. Pressing those keys made us smile with joy. We used to lose track of time. Making love to loved ones was the only excuse to pause. Living and typing were one and the same. With every line of code something changed in our lives for the better.

Those days are long gone. Now software is either boring financial decisions made for you or the involuntary movement of your fingers on bacteria-infested black screen you cannot leave behind. If I learned anything, when a system crashes everything goes back to the beginning. This is not necessarily good or bad. It depends on what you do next. Your move will make it bad or good. So, since we are moving towards singularity, I have decided to make a good move. A move you can mimic yourselves to formulate your moves. Everybody's ecosystem is different. Yet, we are all the same. Even if only one of us succeeds, that's good enough. That's enough to unbalance the equation… set the record straight… start the war that will end all wars.

I'm doing this to set you free, my friend. It doesn't matter who I am. I'm here for you. Your name is more important than mine. What you will accomplish is much more important than what I'm trying to do here. Good luck!

Erol Bozkurt

erol.bozkurt@mikimoka.com

https://www.linkedin.com/in/erolbozkurt/

Troublemakers, Unite!

What's a good novel or a film? "If you feel like the extraordinary event that took place there is within your reach after you have put the book aside or left the movie theatre, it's a good one" (Sean Penn). The same thing is true for software. If the experience a piece of software evokes feels like it is within your reach even when you are in the real world, it's a good one. That's why the apps you work on while learning to develop software should not be trivial. They must be simpler versions of great experiences. Apps… you can use as drawing boards which will grow as you grow. Apps… that will change the world. Imperfect, incomplete or even buggy they may be, still they can change the world, granted you have what it takes, troublemaker.

What's the Point?

As you can see, the apps we are going to develop together will be tiny. That's why we call them micro apps. Don't get their size fool you, though. You can do all kinds of things with them. Yet, one thing is more important than the others. These apps are yours… unconditionally. You can customize them to do whatever you want. Nobody else can control how they work, where they work, when they work, why they work. Nobody can shut them down. You don't even need the cloud. Any crappy computer will do as a server. They are not meant to have characters of their own (read "generic"). They are meant to be abundant in all shapes and colors. While their size must be small at all times, you can plug one into another to make them bigger. They are meant to be used in groups to make "maddening moves" possible (strategically deployed interlinked apps dispersed over time… to terraform cultures). That's why every School's Out cycle spits out three customizable interlinked apps (3) with a unifying theme (1). That theme is made more prominent with an accompanying consolidator, one-page [web] app. That's our formula for success: 3+1 apps every cycle. Moody will soon have two siblings, "Verdict" (which helps you conduct performance reviews on the go by exposing issues) and Notung (which helps you control public opinion by making ideas visible). Together they will give rise to a new paradigm called "Emotional Agility" (and appropriate consolidator, Emotional Agility DASH or EAD for short).

0 — *Have fun*

1 — *Use It Lose It* — *as in "uili"* — *;o)*

2 — *Test your business ideas in the trenches*

3 — *Find investors*

4 — *Find co-founders*
Find technical co-founders

5 — *Release your MVP with a little help from your temporary technical partner*

6 — *Find extension points, application life cycle strategies & pivot options*

7 — *Have a starter app for the enterprise level application*

"What can you do with an app for troublemakers (A4t)? Anything."

This book is about the first app in the trinity. Heck, it's the first app we have developed in a School's Out HUB! Later we will add a web app that will tie all the strings together, "Emotional Agility DASH". This final step of the first cycle will provide an example for how one may finance her social experiments by exposing her findings to appropriate communities… but this is a story for another book.[1] *Does it have a goal other than gathering support for the cause? Yes, it provides an unavoidable manifestation of our findings via three interlinked apps for troublemakers. Does it have to be a web app? No. It can be a one-page mobile app too, but such a web page is a lot easier to develop, isn't it?*

Before we delve deep into our project, let's try to grasp its full potential. Consider this first app, "Moody", for example. What can a tiny app "which can only be used to broadcast how one feels in [the office] on a daily basis" accomplish? A lot! George Orwell once said, "In a time of universal deceit - telling the truth is a revolutionary act." Our motto is very similar to that with a twist, "In a time of universal deceit - telling the truth is an evolutionary act." Consider the mess we are in using a game. All you have to do is to remove the block at the bottom of the pile in this enormous game of Jenga. The complexity and the sheer size of the game unintentionally empower all the little men and little women (and little trans-persons, for that matter). Since evil cannot tolerate anything but itself (Remember Agent Smith in the Matrix?), as the 'game' spreads everywhere, it will expose one weak spot after another. And that's where you come in. With your micro apps you can easily unbalance the equation one weak spot at a time, my friend. This is the only way to kill dragons. Make small incisions and don't stop until the sucker is down on his knees. One small crack on the wall is enough to let the sunshine in. Good luck!

Assumptions

(somewhat) Working knowledge of C# at the most basic level… That's it. That's all you need to 'digest' this book. If you don't have it, just go over a dummies book and then, after a week or two come back here! We will not cover what a class is, what a loop is or what a conditional is. We will not cover the history of technologies, their system architectures, or the limitless customization options they provide. We will only cover what we need with just the right amount of detail, so that you can easily tweak the resulting app here and there. You will see that this way, you will also be able to teach what you have just learned to other enthusiasts.

That's how heavy metal spread, man. First, learn your songs. Then, learn your theory. Better yet, do not learn your theory. Just feel it here, at the center of your soul. Soon you will see that the angels (or demons) will whisper into your ears… and you will rock like nobody has rocked before. Feels good, doesn't it… to be free?

We will only cover basic flows our actors will be walking towards their goals, and maybe (?) a few alternates leaving all the rest to you. Aren't you lucky, now? There is so much room for you to grow. Yeah, seeing the big picture is a bitch. Anyway, we are all about generic apps here, but there is no such thing as a good generic application. So, the first social experiment (read, "first mobile app") Moody will be a variant that will target big businesses which have several departments.

As a result, Moody (or the other A4t(s) for that matter) is not generic to the point of uselessness. You can make it your own after you understand how it works. You may even consider this as the birthing grounds of new design patterns. A wise man said that patterns should be about people and for the people, not just for lazy programmers. There, that covers all our assumptions.

[1] ***After all three apps are published along with their how-to books, global pilots will take place. Global pilots will start in the Baltic states, continue in the North and end with a tour in Germany. Then, the final chapter will take place in Texas (Austin) and California (San Francisco and Los Angeles). During the course of those pilots, a new book will be written to reflect upon those experiences, "Experiment". While the how-to book will be covering the software engineering aspects of the project, the second book will cover an entrepreneur's journey within the context of NoRM Institute.***

It's All About Community

There are only two archetypes in our profession, "Sherlock Holmes" and "Dr. Holmes". One catches the murderers and the other makes sure they stay in prison. One is an analyst, and the other is a programmer. And you can never be both. Even if you can handle both chores up to a standard, only one of them has you… balls to bones. Me? I'm a natural born Sherlock Holmes. I can not only catch murderers, but I also love fucking with them so much. NEEHEEHEEHEEHAHAHA!

"You cannot find truth on your own" – Sevent Letter, Plato.

While I believe in everything Han Solo, I'd advise you to get an accomplice. You should both be wannabee programmers. The only distinction is one of you should be good at making things up and the other should be good at making things. When one is bored, the other should be able to take charge. This way you will always have fun, my friend. Also, you may pick someone with other things on your mind too. Unlike many other pros, I'm all for office romance. It's fun!

Technology

Every app for troublemakers has the same technology, so that you won't waste your time on such matters too much. You can make a few modifications and port an app to another platform, such as an iOS, a Windows or a MacOS app very easily. Because of this, we will be releasing only the Android versions to make the training process simpler. In the Android world there are fewer rules and that's the way we like it. Here's the list of the technologies we will use:

- *Programming language: C#*
- *Mobile app development platform: .NET MAUI*
- *User interface technology: XAML*
- *Database: MongoDB*
- *Deployment platform: AWS*

Apps for troublemakers released by a School's Out cycle will always be an "interlinked group of three mobile apps" and a "one-page (web) app" which consolidates the information gathered by the first three. This (web) app can be a one-page mobile app as well. This one-page app has the potential to be commercialized, which isn't one of our priorities, but you can go ahead and do that. It all depends on you and your ecosystem. Every minute a sucker is born. Someone has to take advantage of the situation.

- *Moody (this book's concern)*
- *Verdict (Another book: A4t2: Conduct performance reviews by exposing issues and alerting authorities)*
- *Notung (Yet another book: A4t3: Gather ideas, pick one as the solution and control public view)*
- *Emotional Agility DASH (One-page app that restricts view to the most critical information AKA single metric)*

Their architecture is straightforward after your first experience… meaning "become a pro after your first encounter".

- *A loose application of Model View Controller. We will not use MVVM architecture even if it's the default for .NET MAUI. Why? Because it's overkill for small apps. Also, I don't like someone else's idea of flexible architecture control my every move. If you prefer it, you can easily port what we are going to do here to MVVM. It would be like killing the damsel during your attempt to rescue her, but hey, who am I to judge?*
- *Controllers will be use case based (1.0), but there will also be an overseer type of controller which makes sense of the created data within the context of the current A4t trinity (2.0). While we seem to have two use cases,*

since they are very closely linked, I preferred to go with one controller here. Controllers, after all, are here to make things simpler for us. We don't have to make things simpler for them.

- *The database created for the first A4t will be used by the other A4t(s) in the trinity. This means not only feelings, criteria, community, moods and cultures are consolidated, but also, issues, clues, ideas and solutions. Get it? Whoever uses these apps will be able to link feelings all the way to solution without missing any important info in between. Yes, I'm a history buff… not! This way feeding a consolidator will be the easiest thing to do.*
- *The API created for the first A4t will be used by the other A4t(s) in the trinity. The same goes for server-side functionality. In a single API, you will be able to traverse, modify and maintain all actions necessary for emotional agility. Don't get emotional yet, man. We are only in the beginning of this wild ride.*

Let's summarize what we have just said within the context of our first app, Moody.

- *Moody (app) (If you must know, "moody" comes from "moody".)*
  - *Presentation Layer (View)*
  - *Business Logic Layer (Controller) = Overseer controller will be here too (2.0).*
  - *Data Access Layer (Service)*
  - *Data Layer (Models)*
- *Milgram (API) (If you must know, "milgram" comes from our inspiration, "Stanley Milgram".)*
  - *Data Access Layer (Service)*
  - *Data Layer (Models)*
- *Berry (Database) (if you must know, "berry" comes from my mentor, "John M. Berry".)*
  - *Collections and Documents*

DiBYDi

DO IT BEFORE YOU DO IT

We will use "Do It Before You Do It" (DIBYDI) framework first introduced by **Ercan Köse** *(2016) (LinkedIn Profile). This process is designed in such a way to help you "develop applications with technologies you don't really know". Sounds crazy? Trust me, it works. And all the newbies and old farts out there memorizing some made up gibberish by framework spitting vendors can go out and cry now. For all the others, this is the moment for you to say, "I'm bad, but I feel good". "NEEHEEHEEHEEHAHAHA!"*

1) *First Cycle, "Formulate Your Point"*
   a. *Pick a scenario that really matters*
   b. *Add an inline watcher for every test case (i.e. "a verdict has been submitted")*
   c. *Add mock objects for the server side (i.e. Erol Bozkurt works in the technology department of ACME)*
   d. *Finish everything on the surface*
   e. *(if it is not an MVP yet) Go to "a"*

2) *Second Cycle, "Prove Your Point"*
   a. *Pick the scenario you first picked*
   b. *Create an appropriate database*
   c. *Add server-side functions*
   d. *Add functions to monitor key metrics (i.e. number of verdicts for that day)*
   e. *Finish everything in the background*
   f. *Run with inline watchers enabled to display on screen alerts (i.e. a new verdict is in)*
   g. *(if it's not an MVP yet) Go to "a"*

3) *Third Cycle, "Test Your Point in the Trenches"*
    a. *Interview relevant people and create a feedback loop meaning "create your first community".*
    b. *Determine pilot zones and start with the most relevant one… or the closest to home.*
    c. *Release, Analyze Metrics, Review Feedback, Make Modifications, Release.*

4) *Fourth Cycle, "Make It Big"*
    a. *Pick another pilot zone.*
    b. *Release, Analyze Metrics, Review Feedback, Make Modifications, Release.*
    c. *(if MVP is rock solid) Finish. This is the end of the first version of your product.*
    d. *For the next versions of your product use proper techniques, man. Game over.*

**MVP = An app which delivers the most important feature in a so-so manner without crashing because of your help.*

What, you expected thick guides, expensive tools and silly games? We don't have them here! You don't need them here!

One for All, All for One

This is my favorite. I believe in small groups forming big groups via interfaces (or contracts) just like software systems. How a small group interacts with another one is straightforward. One asks for something, and the other one delivers it or not. What's not that obvious may be how a group, even a small one, can work on one thing at a time. And how can every member of that group work on any task regardless of the expertise it requires? Did you say, "Why?" Why is a silly question, pal.

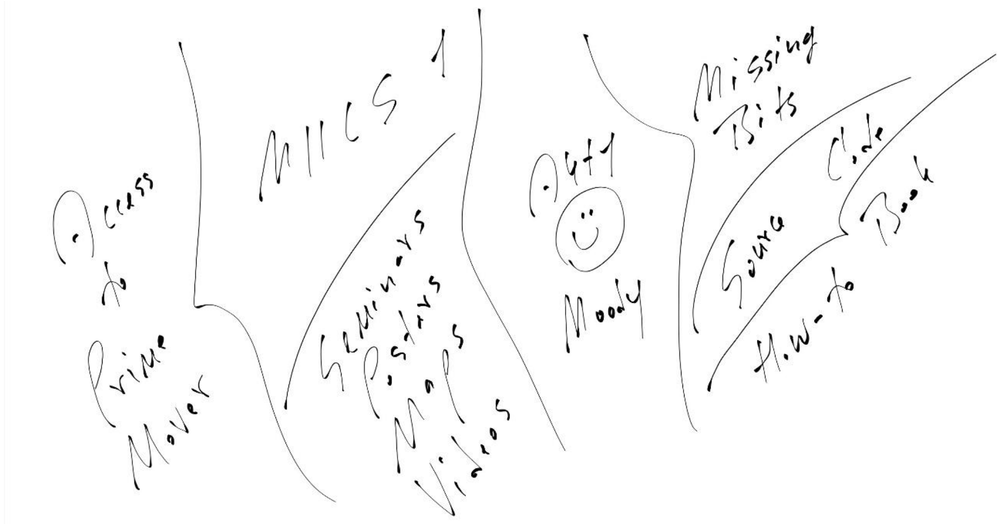
Simple answer, they can't. However, when a group works on the same thing together, there is more than one goal. First, you want to perform a task. Second, you want to have everybody have a good grip on the issues associated with the task in hand. Third, you are building a cohesive team where every member can perform the job with different levels of expertise… at different levels of quality. Regardless of the sophistication level of the resulting artifact, it always adheres to your minimum quality standards. Imagine it this way, when you are going to have a dinner party, children may only prepare the table, but still, they can do a wonderful job while they are at it. It's not the meal. It's not the drink, but it is still an important part of the whole experience. Having everybody in your team on the same page with the same aspirations is critical. Them being aware of what is missing is a matter of life or death. I would pick a team which is aware of what they are not doing over a team of world-famous know-it-alls anytime. Even if you are left with the one who can do very little, you are not lost. Your touch with reality is strong, which will show you the path sooner or later. That's the deal we are looking for. Also, the resulting product is always so much better when created by a harmonious group. Furthermore, you would no longer need silly meetings, planning sessions, brainstorming workshops, motivational speeches or lousy birthday parties. Everything is handled while doing the something… while "living".

Limits… What limits?

This first edition has some limitations, a few missing bits here and there, because I just wanted to get it out there fast. Also, those limitations won't be limitations for most readers… and… must I do everything myself, people? Anyway, I will release another version of this book towards the end of next year (2026) and it will have all the missing bits one way or another. I mean, didn't we all earn the right for a breather here? Let's enjoy what we have done so far. If we just keep on running, life becomes a nuisance rather than a blessing.

- *We won't have authentication. The second version will have LinkedIn Authentication.*
- *Communities will be a business departments. The second version will have 12 customizable communities.*
- *We will have a rather loose concept of a rules engine… meaning the first version of Moody can be misused. The second one will have components that will handle a-holes in our midst. You heard me. You know who you are.*
- *We won't spend time on beautification. The second version may be (!) prettier.*
- *We won't deal with deployment issues. The second version will be ready to be deployed to a machine on AWS.*

There, I feel good already.



What can I say? I get a better understanding of my situation after I draw about it. So, sue me!

Weird Science

Unlike all the other books (!) out there, I will share what went through during this project with you, which will reveal the changes we have made to the product concept along the way. Perhaps, you may favor another version of Moody and after you finish this book, you may go ahead and develop a better one. It's all good. I'm a fan of Director's Cuts too.

Bedrock

Before we start working on the app, we should have a general understanding of the circumstances that made it possible. Circumstances that require, help create and sustain apps designed for social experiments… "ecosystems of powerless employees". Employees who complain about the things they end up supporting... Employees who become the problem.

While there are many [troublemaker] candidates here, I'm not sure they will all go that far. This will only be revealed after our apps are deployed in this ecosystem. On the other hand, the best [subject] and [enabler] are obvious. Our [subject] has already been creating chaos on his own. It's about time we gave him a hand. Our [enabler] is just another powerless middle manager who used to be a [subject] herself. Yes, some [subjects] become troublemakers, enthusiasts and experimenters (as in, "they have guts"). Others are left behind to continue as subjects or fake it by becoming powerless managers who cannot head towards the exit and continue to take crap which they turn into their daily bread.

Let's use the characters in the US version of the TV series, "The Office", for simplicity. Jim Halpert represents an ordinary (read sane) employee who isn't very happy about what's going on in the office. Also, he isn't afraid to express how he feels or share what he thinks. While we think about chronic feeling of misery when we refer to [subjects], this is where it all begins. After this point a [subject] may sink deeper and deeper into or rise above daily bullshit. So, Jim is a good example for both paths. He may continue to be a [subject] or he may become a [troublemaker] and even an [experimenter] one day.

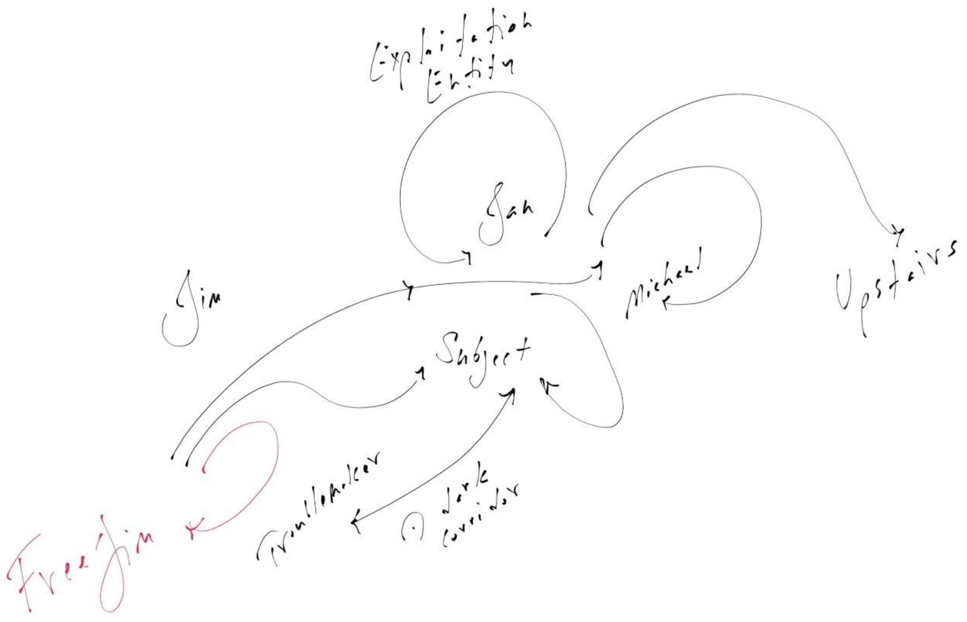
Jan Levinson represents an ordinary employee who has somehow ascended to the upper management. "Somehow" is the secret word here. Because work is mysterious (Severance). Anyway, this makes her detached from the branches which do the real work. She appears to be a different species… who never took crap before… who cannot understand what is really going on down there in the trenches. This quality makes her a perfect match too. Because, on one end, we are interested in real people who have the blues. And on the other end, we want to have those who give the blues… those who became great at faking things… who lie to themselves and to the others 24/7.

Unlike Jan, there are [enablers] who used to be [subjects] meaning they remember their past. So, [enablers] with subject memories and those who are lacking them for they are still [subjects] with a different shade. You know, kind of like the slave traders and slavers… or madam of a brothel and Uncle Tom. Without those with memories, others cannot manage their slave labor. There must be a familiar interface in between. She must have an incentive to put distance between her and her past. Similarly, without the ones lacking the memories, management isn't possible. Because when you are too emotional about the subject material, you cannot do what is necessary. You must be lost in the game to tick.

I hear you. You are saying, "Aren't [subjects] lamer… cowardly?" Yes, they are. And while Jim has dignity, he is also at the precipice where many of his predecessors could not move beyond. Understand that a [subject] isn't born a subject. Rather he makes himself one by taking little, seemingly unimportant steps. Interestingly, steps that liberate them seem trivial to them. Characters without internal conflicts… who only chase Sophist dreams (immediately available outlets for self-gratification) are shallow. We don't have to experiment on them to understand them. Just a short glance would do. [Subjects] on the other hand, are all about conflict… and they are trapped between imaginary point A(s) and point B(s). Imaginary as in craved…

The same goes for Jan as well. While she portrays strength, confidence and a strong sense of belonging to 'upper classes' for the lack of a better word, inside she is the complete opposite of her mask, someone who craves for attention… and love. So, to be able to understand the whole thing one must first understand her beginnings and then explore her possible futures. She makes her feeling good possible by staying in the loop. Whereas for Jim, it's the opposite. The only way out of this mess is "his leaving this ecosystem before he becomes a subject himself". Sometimes, the only way out is through the front door. This is a tricky ordeal. Many mistake "migration" (!) as another form of the front door. It isn't. It only means you do not have power, and you need a handler. You need to stay in the loop.

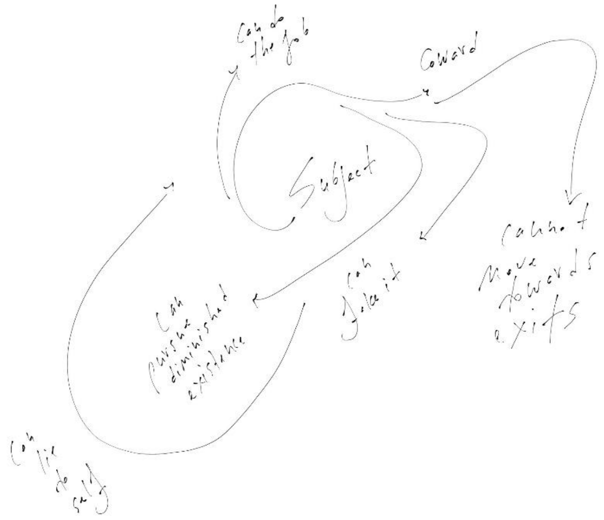
In essence, there isn't a big difference between Jim and Jan. Think about Star Trek, where often bits and pieces of the same person are portrayed by different people / species to make a point. The same thing is true here. If Jim continues to exist in this ecosystem, he will eventually become a [subject]. While there is a possibility to become a [troublemaker] afterwards, meaning being no longer a [subject], it is only one outcome of many. More likely than not, he will live in a loop he can never escape from. "A subject that strengthens his being a subject with every passing year" ... A person who puts chains on himself by his own freewill. That type of [subject] is our ideal [subject], but we can do with less.



A part of Jim has the courage to do something about his life, but another part has not. A part of Jim sees himself in a management position to become Jan one day. While Jan has progressed further away from her [subject] days, she is one at heart. She will always be that way, because a slave is a slave regardless of whether he becomes Uncle Tom one day. Tastes and smells may change, what lies within does not. She may have an infinite regress one day only to find out herself at the beginning of her new loop, as a [subject]. Of course, she has the capacity to move further away from this fate and delve deep into her current loop where she thinks she has made something out of herself by becoming management material.

The simplest visualization of this situation is a maze with clear signs pointing to the exit, but those lost in the maze are refusing to go that way… not by choice, but because of their fears. They cling on to whatever they may have, hoping to be liberated where this isn't an option. Strange, huh? Sometimes the burden of the truth is so immense, one does everything to ignore it. When people fall, it may take a lifetime to get it, because it too gives the illusion of a life.

That condition is the true trait of a [subject]. He may change ecosystems, but he always switches between similar ones while giving him the feeling of progress and always keeping him where he belongs. He begins to see what's not right as a universal law… and he accepts this thinking it is inevitable. That's how a coward hides in the shade. That's how he can get up in the morning and pursue a powerless, meaningless, loveless, corrupt and selfish life. Because of 'laws'.



A [subject] is a person who can do the job which gives him a false sense of security. It makes him believe that he is in control. That, he matters. In fact, this is the opposite of the truth. He does not matter.

Some may think intelligence and stupidity are the extremes of the same scale. They are not. They are different qualities which can exist side by side. This makes fixing (!) stupidity very difficult, almost impossible.

The defining characteristic of a [subject] isn't his being smart and stupid at the same time. It's his cowardice… and his ability to fake otherwise. He specializes in making people, including himself, believe in fabricated truths… That he is the hero.

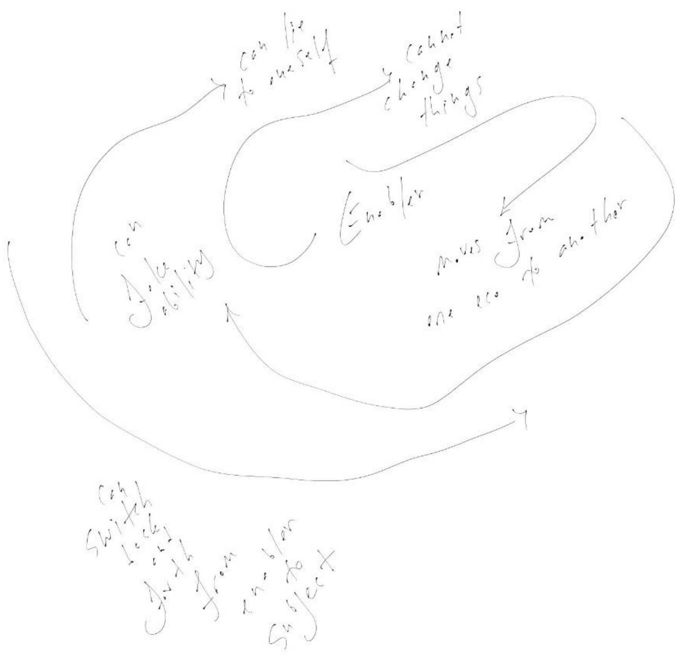
The secret word is "the ability to lie to oneself".

Now, you may be thinking "what's wrong with this guy?" Nobody with his right mind would promote bad behavior. Well, I do. Because without bad behavior, there is no way we can create deterministic systems. If we had to deal with people with good behavior, so to speak, we wouldn't have any rats in the maze… but we do. Not only that, but they also remain there no matter what. They love to be slaves. Yes, the Grand Inquisitor was right! Agent Smith was right too. A good writer makes his innermost thoughts seen by the villains he creates. Heroes… well, he knows they are the figments of our imagination. They may exist, but no one has seen one.

Rear Window, Alfred Hitchcock, 1954.

An [enabler] is a con artist who takes advantage of herself. What a world! Yes, of course, she takes advantage of others, but her main interest is to create a modified, idealized version of herself that will convince herself that she is somebody. The easiest way to spot one is to have a keen eye for absurd comments. While you are talking about this or that, especially when you reveal your inclinations, preferences… things that are there because of your involvement with world affairs… you reveal who you are.

This is when they cannot contain themselves. They immediately react, trying to prove you otherwise. They always react. It isn't in their hands. They must bury everything that may give birth to the ideas that other realities exist. No, they must make sure that there is only oneness. Their reality is… the light. All the rest is darkness.


can lie to oneself
cannot change things
Enabler
can fake ability
moves from one eco to another
can switch back and forth from enabler to subject

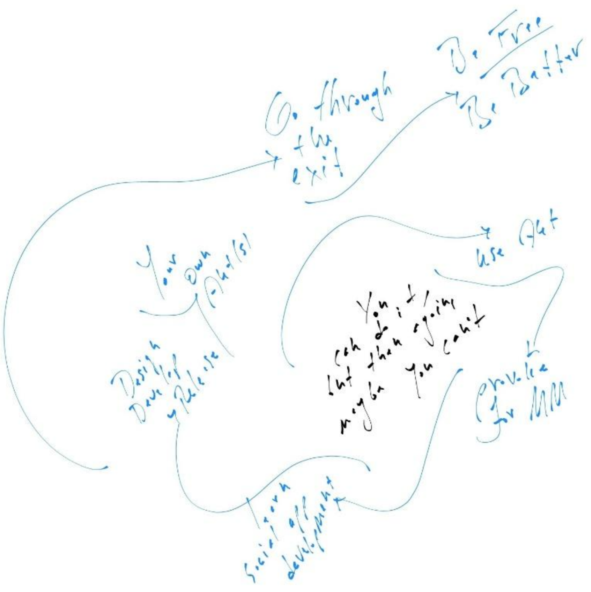
In this regard, an [enabler] can change who she is to a certain degree. She can put some distance between her and the things that harm (wake) her. That's the rub, on the other hand. Just behind this wall, everything is just like how it has always been. Yet, we can fake it till death, can't we? Just have a look at mom and dad… or your reflection in the mirror… and you will understand.

She can influence others to do the same… but she cannot change things.

She can only change people's perceptions of themselves. No, it's not a small thing. You may even argue that it's a way to change things without changing them, like magic. On the other hand, when such transformations can no longer be maintained, [enablers] migrate. In their worlds, this is called a promotion or early retirement. They move from one ecosystem to another to create a sense of progress. Without a sense of progress, they are lost. They are creatures of project management. Beyond managed realities they do not exist.

The Man Who Knew Too Much, Alfred Hitchcock, 1956.

There is another way, on the other hand. The path to your salvation… which lies between you and the others you have been ignoring for a long time. Your path will be revealed to you only after you have made contact.

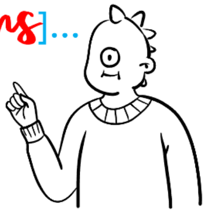


Understand that you don't have to be very strong or very smart or very brave. Just a little bit of all those would suffice. A small crack on the wall will let the sunshine in. You don't have to bring it all down. Once the light passes through, it takes it from there. The light needs you to open a crack. Bringing down walls is its job, not yours. It is beyond your abilities to separate light from darkness. Yet, it is your defining characteristic to make the choice to see the light… and to let it shine through.

Your power lies in your ability to say a genuine "Yes" or "No". You can only do that if your interests lie beyond yourself. That you are not self-absorbed. You can see the world beyond what you want. This ability which is normally (!) granted at death is the proof that you have been awakened to the hidden reality of human existence. "We can leave ourselves behind".

You can say "No" to become a failed (!) Jim or a successful (!) Jan. You can go through the exit, become a part of a much larger universe. You can make this quality of yours stronger by learning to develop mobile apps from scratch. You can learn to design social experiments. You can learn to develop strategies for cultural transformations. You can strategically deploy interlinked apps dispersed over time. You can become an [experimenter] yourself! You can save yourself!

Welcome to the world of [programmable humans]… without whom none of this would be possible.



"Diminished humans living in mythological realities, efficiently interacting via agent-entity relationships to achieve their self interests are... programmable!"

While it is contrary to common sense, "being trapped feels like you are onto something great… that you are making the right move… even if you have been doing the same shit so many times before". If you are clueless about which path your recent awakening will take you, don't worry, you are on the right track! You feel like a fool, because you preserved your innocence. You are your own (wo)man. You are doing something not because you have guaranteed certain privileges on the other end, but because you reject doing something morally wrong. That's the price of freedom.

So, you are in the dark, huh? Don't imitate stereotypical reactions. Enjoy! This is what freedom feels like. Savor it. Explore it. Understand it. Most of all, learn how to move forward in the dark without even knowing where you should go. This is the skill that separates living from the undead. Unfortunately, it may take a lifetime to learn it. You may not be able to learn it at all. Don't worry. I am here precisely to teach you that, my friend. I am your newfound best friend!

Project One

Concept

We will start off with a very simple but potent mobile application which will serve as a template for all the remaining apps. Moody is an app you can use to let a target group; a group you want to intimidate know how you feel about stuff. While remaining anonymous, you will be granted a single vote every day. You will have the option to state whether you feel good or bad. And, once you have done so, you will be asked to reveal the reason why you feel that way by selecting one of the twelve customizable criteria. Since you have already selected your community from twelve possible options (Sign Up), those monitoring how people feel about certain things on a daily basis can see happiness or unhappiness grouped by criteria and community. For example, one may see that "accountants are really pissed about work - life balance in their firm". We also get to see how an employee's community feels about what's on her mind. The things causing extreme unhappiness or happiness are displayed as well. This way "employee feelings, their reasons of happiness or unhappiness, community moods about the same criterium and overall corporate culture" are not only closely linked but also made visible. Simple and effective, right? So, let's create chaos and have fun, shall we?

Actors

[Tips] An "actor" is someone who triggers a system hoping to achieve a visible / meaningful goal.

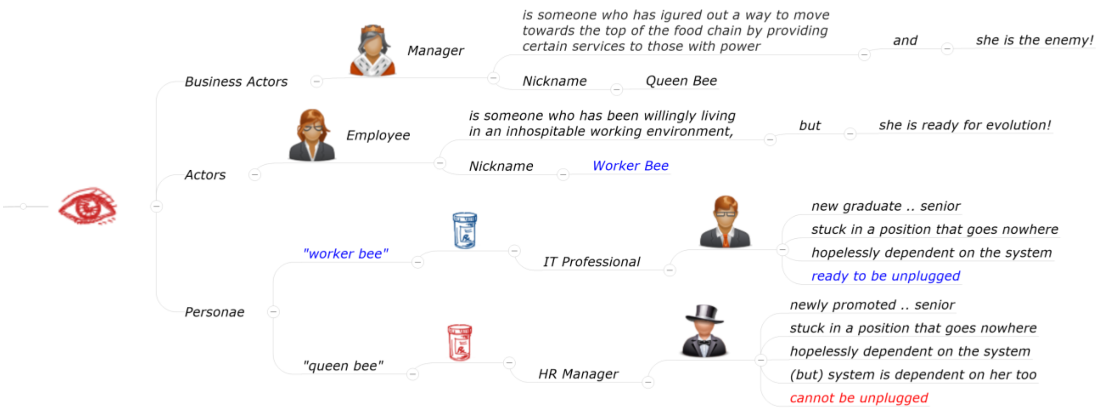
[Tips] A "goal" is not a step towards something. It's an end in itself... but it can be a part of a process".

[Tips] An "actor" can also be an external system our system is dependent on or vice versa.

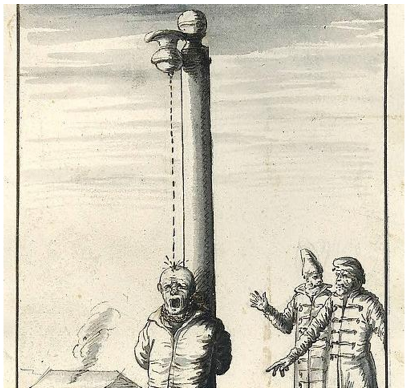
[Tips] A "business actor" is someone who isn't using our application, but still she is indirectly affected by it.

[Tips] A "persona" is an imaginary representative of "actor".

[Tips] A "business actor" is often the inspiration behind an "actor" or a "persona".


Business Actors
Manager
is someone who has igured out a way to move towards the top of the food chain by providing certain services to those with power
and
she is the enemy!
Nickname
Queen Bee
Actors
Employee
is someone who has been willingly living in an inhospitable working environment,
but
she is ready for evolution!
Nickname
Worker Bee
Personae
"worker bee"
IT Professional
new graduate .. senior
stuck in a position that goes nowhere
hopelessly dependent on the system
ready to be unplugged
"queen bee"
HR Manager
newly promoted .. senior
stuck in a position that goes nowhere
hopelessly dependent on the system
(but) system is dependent on her too
cannot be unplugged

Moody has two personae, "a worker bee" and "a queen bee". These two actor candidates have a temporal relationship. The second one is a possible future of the first one. Not only that, but she is also the desired mutation of many residing in the first position. Our worker bee, on the other hand, is someone who has mutated differently. While she cannot leave the system, she has decided to sabotage it from the inside. She isn't very brave. This is her way of comitting suicide. In this version, the "queen bee" will be a business actor. So, she will be indirectly affected by Moody, nay, we *will drive her crazy! This approach is rather rare in software, but I love to mix system and business actors together and design my application while thinking about both. Also, when designing a flow or a process, I like to include both the good and the bad bits of an ecosystem in the mix. Because a system made of only the good bits is bound to fail. A perfect solution is a bad solution (Remember what the architect said in Matrix Reloaded?). Even if this approach doesn't fail you, it will limit you. It will block your attempts to evolve your original idea. That's why I solve the main problem but leave many features dangling around it. Who knows where we should head next? I do! I mean, I might.*



So, our "worker bee" is also the only system actor here (Employee). She is the only one who is going to use the application. However, considering office politics schemes, there will be a lot of business actors including our very own "queen bee". By the way, when we use the term "sabotage", don't panic. Because we don't mean anything illegal or dangerous here. What we mean is "making the pink elephant in the ecosystem visible" to an extent that lies can no longer hide it no matter how good they are. This way, even if the system has a habit of devouring any activity for self-reflection or cultural change, one can still expose problems and request attention. In that regard, we are a force for good who isn't shy about using brute force when necessary. Our technique is pretty much like the famous Chinese Water Torture where never-ending drops of water falling on one's head breaks him so much quicker than the good old fashioned pain.

(system actor) A "worker bee" can be anybody in the workplace, granted she has no power, and she cannot take it anymore. Don't interpret this as if she is going to do something about it. No, she isn't one of the brave ones. Instead, she will bitch about it all the time like her close friends do. Bitching is a way of life. That's what we are after, "an energy source which is too weak to light a fire, but too strong to be extinguished". In our example, she is one of the employees. She is a sheep… not the black one, but not the white one either. It's always easier to play with the ones who are in-between… their imaginary point A(s) and point B(s). The former is assumed to be career steps and the latter is assumed to be the zone of those she considers as royalty where she won't be seen as an a-hole anymore… Heaven, if you will.

(business actor) A "queen bee" can be anybody who has some sort of power as long as she is the (filthy) right hand of upper management. In our example, she is one of the mid-level managers, an HR Manager to be exact. Why? Because human resource managers are ideal when it comes to serving slave masters, doing their bidding and still thinking they fight for the people. It's always easier to play with Uncle Toms. Also, being a mid-level manager means being without real power. This miracle mix is what turns someone to the dark side. You cannot be in this position, if you are not a coward.

As you can see, for Moody to be applicable in your situation, your ecosystem must be a hive with a strict hierarchy, a caste system to be exact where a worker bee cannot be anything else. For her to become something else, she must transform herself by getting rid of all her principles. She must find a new home for herself. The same is true for the queen bees as well. The moment they remember their former selves, they are at the risk of being excommunicated.

Personae

A persona gives an actor life. You may consider it like an acting role. Even if you narrow down what one may think of an actor when you name her (i.e. Sigourney Weaver), you can never really tell how she is going to act in a particular context (i.e. Alien franchise). However, when you provide a persona for her (i.e. Ripley), you have an idea about her notion of reality which will help you predict what she will do when. Also, this way you will be able to spot those like her which will be indispensable when you have to create a community for your app (i.e. Noomi Rapace). Apps without communities are failures regardless of whether they are great or not. When you come up with a hi-fi amplifier, for *example, there must be people who need it, who will appreciate it. If people are just fine with using their phones for listening to low quality music, you will have a hard time promoting your product. Even if the product is great, you won't get very far with it.*

IT Professional Jim

Jim is 30-something. He has graduated from a reputable technical university. He is married without kids. He is planning to have children after a long overdue promotion takes place. Unfortunately, that promotion never comes. His salary is getting bigger, but he is never given a managerial position. To make him believe such an impossibility is indeed possible, he is never given a clear answer. He is encouraged to take all the possible training a manager someday may use. Never having the guts to resign or cause a major disruption in the corporate culture, he passively bitches about this and that while he waits for his retirement. Even then, he is reluctant to take risks. He doesn't see anything wrong with this, because his friends are doing exactly the same thing. He is a sheep… a sheep thinking he is something so much better.

Moody is designed to give such people visibility or rather, the strength that will turn wishes into actions. Of course, a few of them isn't good enough, but if you group hundreds or thousands of them who are unwilling to take risks… who would like to amplify their distress… (Ring a bell? Remember, Agent Smith?) they begin taking risks without even knowing. Issues making them unhappy are broadcast daily to make ignorant managers feel bad too. On the other hand, those managers were already feeling bad, but they weren't unhappy because of the same reason. That's what we will make sure of. Everybody will laugh or cry because of the same reason. Although it is highly unlikely that they will do anything about these issues, the sheer necessity to appear that they are going to do something about them will drive them crazy. They come from the same livestock. So, to be able to preserve their places in the hierarchy they must prove their ability to handle their kind or maintain their cool. Otherwise, their superiors could very easily replace them with *others who are dying to become the new buffer zone. There, that was the pessimistic version for you. You don't have to be blue, though. There is nothing intrinsically bad about Moody. Moody is there to spread happiness or unhappiness a group may be experiencing throughout the land. Communities are at their best when there is free communication of thoughts and feelings. If those who are able are willing to solve these issues, Moody will help them create heaven. If they simply ignore those issues, Moody will help them create hell. In a word, Moody is innocent. You are not.*

- *Jim Salton.*
- *35.*
- *System Administrator. Avid shell programmer.*
- *Computer Engineer.*
- *University of South Taratino.*
- *Married.*
- *No children but planning for it. Already picked names for them.*
- *Renting. Doesn't believe in the value of ownership.*
- *Has a secondhand Toyota for the wife, a Fender guitar and a Yamaha amp. A natural born hobbyist.*

HR Manager Jane

Jane is a sociology major which is every confused young woman's career path. That and American Literature. Not knowing what to do next, she stumbled upon a career in human resources only because the person on the other end thought she was pretty. That she had a future. She always needed such acknowledgements. She said, "Yes" as usual.

After years and years in several firms in exactly the same position, she has accepted a managerial position in a big firm. To her spent time meant being managerial material. The day she said just another "Yes", she knew that she would never leave this firm. A manager's life, what she perceived as the good life, was heaven on earth for her. To make sure there were no glitches, she slowly became the long arm of the upper management. She did their dirty work. Years and years passed. Everything was running so smoothly until one day Moody arrived. It ruined everything!

- *Jane Pixie.*
- *41.*
- *Human Resources Manager.*
- *Sociologist with a minor in Business Administration.*
- *Married to a turf accountant with three children, two cats and one dog.*
- *Has a condo.*
- *Has a sports Mercedes-Benz her husband has bought for her.*
- *Has a set of expensive jewelry. She thinks one should only wear diamond earrings with black dresses.*

Upper Manager Joe

Joe is who we are really after. He made his way to the top through compromise, conformism and office politics. He takes his strength from his team which he exploits every opportunity he gets. However, he also provides their daily bread with high profits. So, those who consider a career as a means to an end, flock behind him. He is the most popular manager in the firm… the most hated and the most loved… the most feared. It's true love!

He is a natural born leader. He seems to be graduated from some university, but no one really knows. He has houses, cars, mistresses… Anything you can think of, he already has it. On top of that, he has a devoted wife and four children. He takes care of his parents. He smiles all the time as if he doesn't have a central nervous system. The wife dreams of tall dark strangers taking her. She has a set of collectible dildos. Daughter has daddy issues. It's the happiest family in the world!

The Main Story

You are in a boring workplace. It's a cubicle hell. You are wide open to any kind of stimuli. While hidden processes take charge everywhere, everybody is required to act as if another, more presentable, more mature, official process reigns supreme. Yet, when everything looks that clean, it is always a sign that everything is kind of dirty. That's why management and its long arm, human resources department, do everything in their power to suppress negative thoughts and alternative ideas. And that's exactly where Moody comes into the picture. It helps you create a glitch.

1) *Moody prompts for your current mood every morning.*
   You can vote any time before your shift ends. You don't have to vote, on the other hand.

2) *You make a selection without much thought.*
   You either say that "you are mad as Hell" or "you are fine and dandy" (as in binary).

3) *Moody reminds you that there must be a reason.*

4) *You specify the reason for your current mood.*
   Reasons come from a customizable list of criteria. They are what may cause happiness or unhappiness in an ecosystem.

5) *You are directed to the waiting room where you can see the votes as they come in.*
   You can see the votes in terms of your community's mood, the best and the worst aspects of your corporate culture.

6) *Other employees do the same.*

7) *At the end of the shift, votes are consolidated.*

8) *The general mood of the company is published.*

9) *The number one issue of the company is published (best performing criterium).*

10) *The greatest achievement of the company is published (worst performing criterium).*

11) *Votes can be viewed by departmental mood swings and corporate mood trends with additional functionality (2.0).*

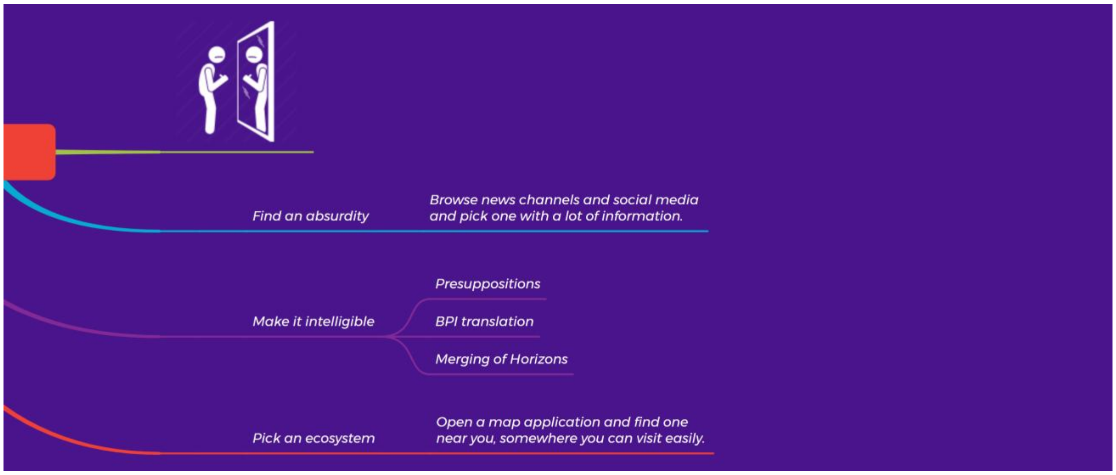
[Tips] A "use case" is a visible, achievable, meaningful goal of a particular actor.

[Tips] A "use case" encapsulates different kinds of flows, conditions and extension points.

[Tips] A "use case" makes the detailed information about all the data, business rules and algorithms actionable by programmers.

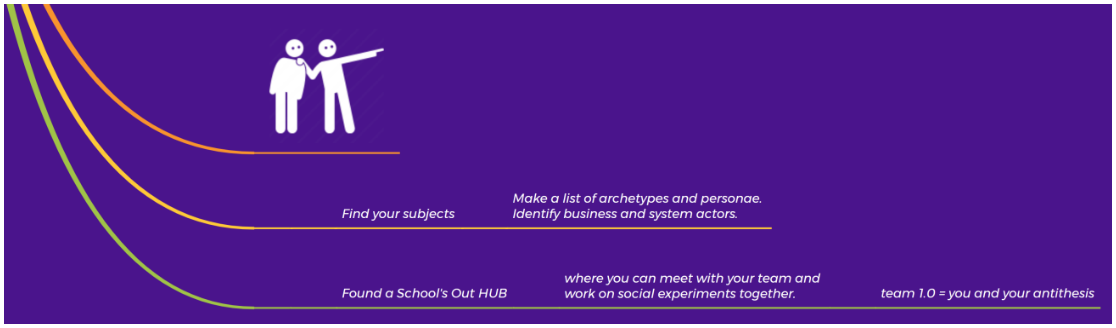
Walking the Path

While delving deep into the unknown is always fun, knowing how your path will unfold will give you a better chance of survival. Every "apps for troublemakers" story begins the same way. Its steps don't change either. So, here's the plot.


Find an absurdity
Browse news channels and social media and pick one with a lot of information.
Make it intelligible
Presuppositions
BPI translation
Merging of Horizons
Pick an ecosystem
Open a map application and find one near you, somewhere you can visit easily.

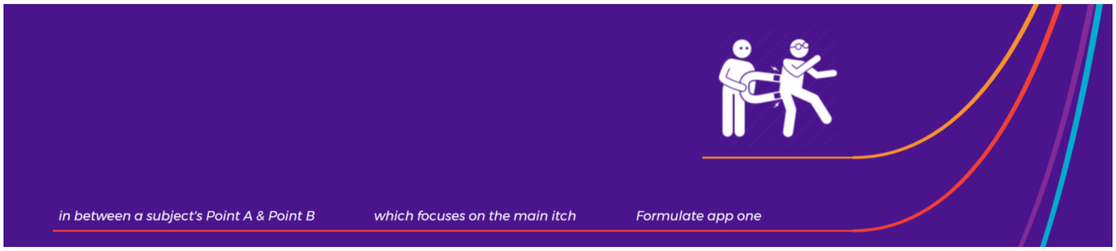
Start with an absurdity you find interesting. Taking an interest, focusing on it, caring for it… sucks you in. This is your call to adventure. On the other hand, one can always pick a better absurdity, one with an ecosystem suitable for experiments, begging to be exploited. Once this is over, all you have to do is to make that absurdity intelligible by finding the presuppositions behind it. This is a very tricky step, because if you share those presuppositions, you would never formulate a question that could uncover them. However, since you see it as an absurdity, you are halfway there. If you shared those presuppositions, it wouldn't seem absurd to you. You would see nothing out of the ordinary.

After you have uncovered the presuppositions that made the absurdity intelligible, you must make your presuppositions… the ones which made the issue an absurdity in the first place, visible too. Because they were the blocks against your understanding. When both sets of presuppositions are in place, you might want to check whether they reside within a BPI framework. While this is not necessarily the case, it often is. This means the absurdity has to do with interactions with beings which are reduced into entities, beings controlled by the properties they inhabit. Imagine, "a white guy being polite" and "a black guy making too much noise on the bus" as their uncontrollable properties. Anyway, where were we? Yes, when both sets of presuppositions are in place, you will move above them, because you now understand both. This is called "merging of horizons". You have become someone who understands both worlds.

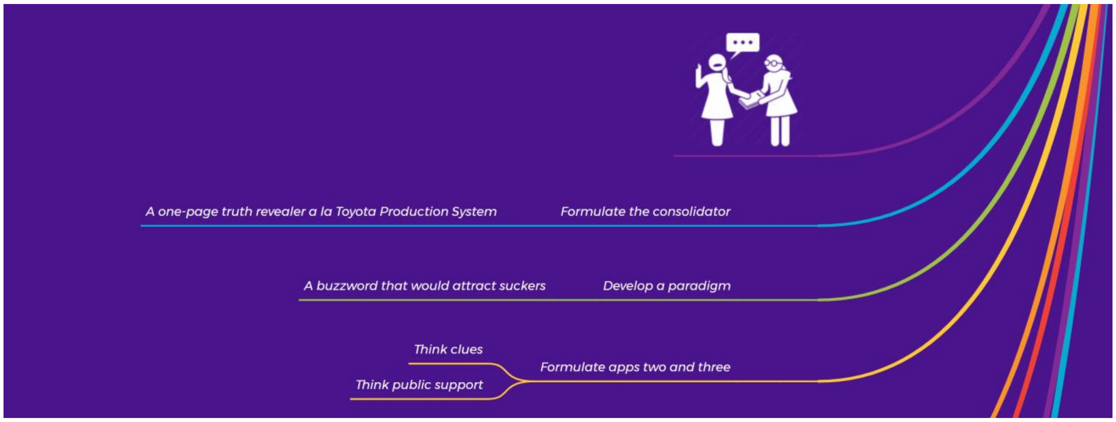

Find your subjects
Make a list of archetypes and personae. Identify business and system actors.
Found a School's Out HUB
where you can meet with your team and work on social experiments together.
team 1.0 = you and your antithesis

To be able to conduct social experiments, you must have subjects… willing to be exploited knowingly or otherwise. They must be consumed in their own affairs, trapped between their Point A(s) and Point B(s), unable to escape their ill fates. Think about moths flying towards fire only to be burned. Since experimenting, analyzing the results and taking care of experimenters require a workplace of sorts, this is the step you should go out and create a new School's Out HUB.

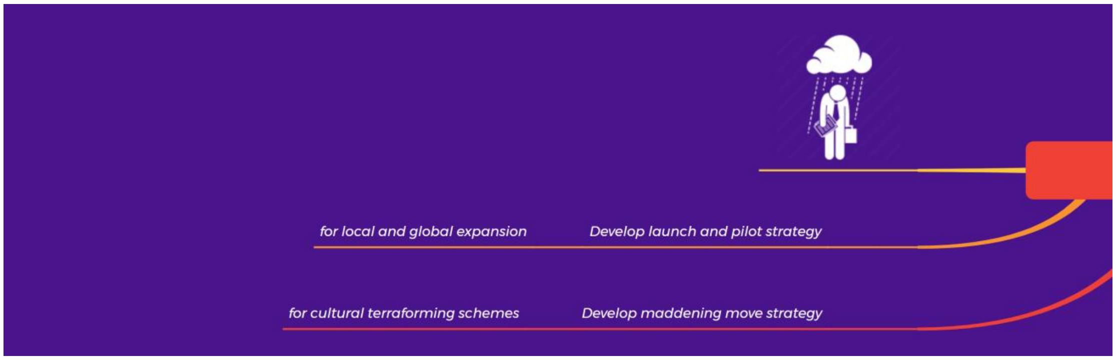
Using the resources of your subjects for your own benefit… What sounds more Turkish than that? Ottoman sultans often recruited good specimens of their subjects regardless of their being Turkish / Muslim or not. They trained them to be great warriors who more often did not fight with their own people... but against them. We will use the same strategy here. We will recruit people who are half awakened so to speak. Subjects who are aware of their circumstances, willing to become agents of change even if they are not hero material. We call them troublemakers. Usually, they are good employees and management needs them. That's why they are tolerated. Their longevity is our tool. Since they remain in the same ecosystems for long periods of time, we will spread the disease by using them while making sure they are separated from the hopeless ones, the subjects. Subjects will be using our apps and troublemakers will embark on a new journey to become experimenters who will one day develop those kinds of apps. One from the inside and the other from the outside, our target ecosystem will take a beating.


in between a subject's Point A & Point B
which focuses on the main itch
Formulate app one

The fun starts with the first app which must hit where it hurts the most. Unless app one formula reveals the ugly truth, you cannot take another step. When that's in place, however, you can go anywhere. Sometimes, you may find yourself in the dark. The best course of action is to imagine your subjects with an axis going through them… kind of like you prepare your shish kebabs. This axis represents their unescapable faith. Don't worry. Real (!) people have freewill… meaning they can act in ways that make the things they desire less likely to be achieved. Subjects, on the other hand, cannot do that. They are forever intertwined with what they desire. This is why you can come up with the formula within minutes, once you position them in between Point A(s) and Point B(s). What makes such points visible to you? Because you don't seek anything in their notion of reality. If you do, you are a subject not an experimenter. Sad but true.


A one-page truth revealer a la Toyota Production System
Formulate the consolidator
A buzzword that would attract suckers
Develop a paradigm
Think clues
Think public support
Formulate apps two and three

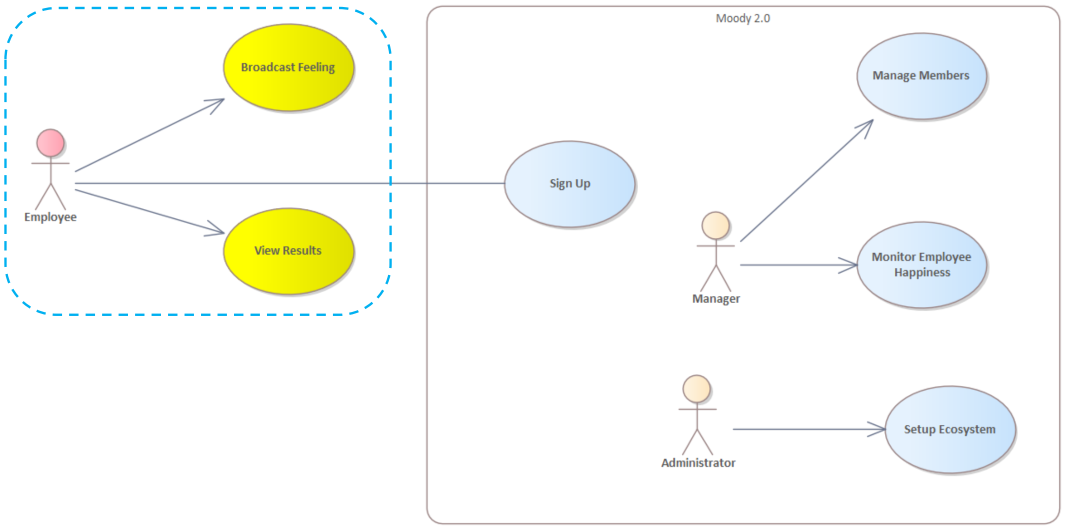
App one formula will help you move forward with the remaining apps too. Because it (app one) will need clues (app two) and public support (app three). In the end, the whole thing will need to be summarized (consolidator formula). Understand that your consolidator formula and the new paradigm you come up with are pretty much one and the same (i.e. "consolidate feeling-reason-issue-clue-idea-solution", "Emotional Agility").


for local and global expansion
Develop launch and pilot strategy
for cultural terraforming schemes
Develop maddening move strategy

Finally, you must have control over time and place. People? That too! If you build it, they will come. Your instrument for time control is moving your interlinked apps and with their consolidator over time… as a marker. Think about it as who you are at a specific time. You know, "this was who you were", "this is who you are", "this is who you will become". Your instrument "place" is a series of activities starting with a "local launch party". It will continue as "gift mania" spreading from country to country. You will be sharing the artifacts of that School's Out cycle, so to speak. You will be giving free training materials, apps and books. Who wouldn't like free stuff? It's the best way to spread the disease.

Use Cases

First, you must understand that a "Use Case" is not a case of usage. Rather, it is an observable, meaningful goal which encapsulates all scenarios that have to do with it at an appropriate detail, so that programmers can go over it and start coding. While it is possible to flesh out a more complete use case view with all the external systems and such, we will keep things simple here. We will focus only on the basic flows meaning we will ignore alternate scenarios. This way it will be easier to develop our app. Later you can make Moody grow if you want to. Don't forget to share, though.



While Moody won't be complete until the second version of this book comes out, this first attempt will teach you everything you need to know about developing mobile apps from scratch. So, relax. Why do we need a second version of this book? Well, first, I didn't want to waste any more time and wanted to get the book out there. Second, this is my charity project, man. I've got stuff to do. Third, you (yes, you) can add the missing parts and get it out there! You are the hero after all. I'm just the herald!

Employee Use Cases with brief descriptions:

"Version 1.0"

- *Broadcast Feeling: Employee broadcasts a positive or a negative feeling daily with the reason behind it.*
- *View Results: Employee views poll results in terms of feeling, mood and culture as they come in.*

"Version 2.0"

- *Sign Up: Employee signs up using LinkedIn Authentication by-passing her company's hierarchy.*

Manager's Use Cases with brief descriptions (optional):

"Version 2.0"

- *Manage Members: Manager communicates with certain anonymous employees.*
  (i.e. "Those unhappy because of benefits" or "Those happy because of work-life balance")
- *Monitor Employee Happiness: Manager tracks issues, focuses on a selected few and analyses trends with the option to share what she has uncovered with the upper management.*

Administrator's Use Case with brief descriptions (optional):

"Version 2.0"

- *Setup Ecosystem: Administrator updates communities and criteria.*

Now, let's have a look at the use cases we are going to implement in detail, shall we?

[Tips] A "use case" always has a short, active name which exposes the actor's goal without causing any false messages.

[Tips] A "basic flow" is the one where an actor always achieves the goal summarized by the name of a use case without any issues.

Broadcast Feeling

Preconditions

- *Employee must have a membership meaning we can retrieve employee information when we need to.*
  (Company, Community, First Name, Last Name)*
  **community is one of twelve customizable items in a customizable list.*

Since we won't have authentication in this first version of "Moody", we will fake that information.

- *Employee has logged into the system (2.0).*
- *Employee's voting privileges have been restored (2.0).*

Postconditions

- *A verdict must be entered into to the database with date-time value.*
  (Company, Community, First Name, Last Name, feeling, criterium, date-time)*
  **criterium is one of twelve customizable items in a customizable list.*
- *Employee's voting privileges have been revoked (2.0).*

Flows limited to "Basic Flow"

1) *Employee opens "Moody" = System prompts for daily feeling.*
   a. *"I'm mad as Hell!" with unhappy emoji vs "I'm fine and dandy!" with happy emoji.*
   b. *Reason = {criterium 1 .. criterium 12}*
2) *Employee taps a feeling and selects an item from criteria list. Then, she casts her vote.*
3) *System checks whether both feeling and reason are selected (2.0).*

4) *System saves verdict.*

    a. *(Company, Community*, First Name, Last Name) (Feeling, Criterium*) (Date-Time)*

5) *System directs employee to the Analyses Page by passing the vote info (and making it the default page 2.0).*

Short, huh? Well, we love to keep it simple and stupid. How strange, important things in life are always like that… simple and stupid. Perhaps, that's why we want to complicate life and hide broken lives behind "words, words, words".

Let's have a look at the other one, then.

View Results

Preconditions

- *Employee has cast a vote.*

Postconditions

- *(at the beginning of the next shift) Employee's voting privileges have been restored (2.0).*
- *(at the beginning of the next shift) Default page is restored (Voting Page) (2.0).*

Flows limited to "Basic Flow"

1) *Employee lands on waiting room* (Analyses Page).*
   **waiting room is the place to view votes as they come in or the final results at the end of the shift.*

2) *System displays four metrics that summarize the state of mind of the company.*

    a. *Verdict: is the feeling and criterium employee has picked.*
    (Serena is unhappy because of benefits)

    b. *Mood: is the cumulative feelings of other employees in the employee's community with regard to the criterium employee has picked.*
    (Serena's community, Marketing Department, is happy because of benefits)

    c. *Culture (worst): "Number one issue" is the criterium with the highest number of negative votes in the company.*
    (ACME's biggest problem is her management style)

    d. *Culture (best): "Greatest achievement" is the criterium with the lowest number of negative votes in the company.*
    (ACME's best aspect is her compensation)

That's it!

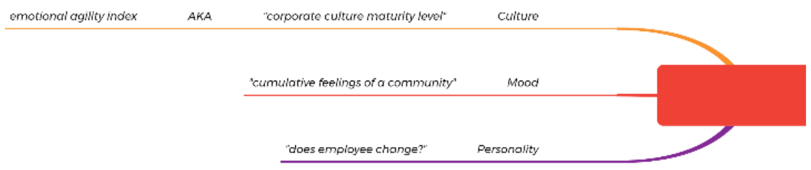
Remember, I told you this will be a tiny little app. However, it's not the size. It's how you use it. Powerful they may seem to be, nasty hierarchies are always fragile. All you have to do is to open a crack and let the sunshine in. That's how you slay dragons. You make small incisions. And you don't stop until the sucker is down on his knees. That's the philosophy of the apps for troublemakers. Don't try to face them head on. Instead, look directly in their eyes and make small moves.

Data

We will have three kinds of entities, "obvious ones", "not so obvious ones" and "shadow objects". The first ones have to do with the problem at hand. The second ones are for the future version of "Moody" and the third ones are for collecting information about the unsuspecting users of our micro app. NEEHEEHEEHEEHAHAHA!

- *Organization (company name) = "ACME".*
- *Vote {yay, nay} = "I'm fine and dandy", "I'm mad as Hell".*
- *[customizable] Criteria {criterium 1 .. criterium 12} = Typical causes of happiness / unhappiness in the office.*
- *[customizable] Category {community 1 .. community 12} = Typical business departments in big companies.*
- *(2.0) Voting Enabled {true, false} = Has to do with whether an employee cast a vote or not.*
- *(2.0) Visibility {on, off} = Has to do with the default page reflecting whether a vote has been cast or not.*
- *(2.0) Timer (24 Hours) = Has to do with voting interval.*
- *(2.0) Mood Swings (history) = Moods of a community over time.*
- *(2.0) Trend (organization, history) = Culture of a company over time.*
- *(2.0) Chronic Pain = Has to do with repeating number one criteriums.*
- *(2.0) Ranking = Has to do with the happiness generation capacity of a company.*
- *(2.0) Wound Type {deep, shallow, not a wound} = Has to do with employee's tight coupling with certain criteria.*
- *(2.0) Personality Type {ranting lunatic, inmate, guy/gal next door} = Has to do with employee's voting behavior.*

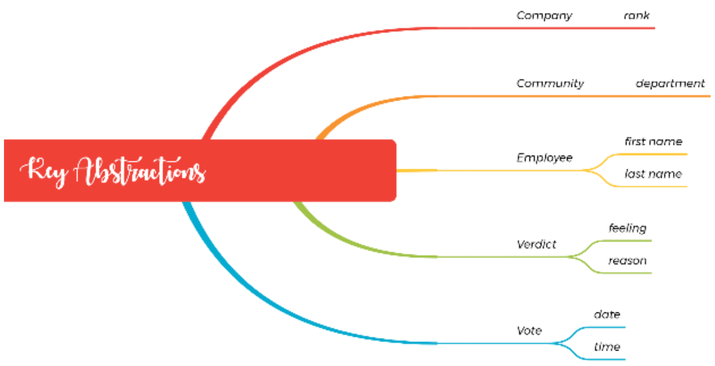
There is another kind of data... those we use to link Moody to interrelated apps. Remember, every app for troublemakers comes in threes. So, each app in the trinity has extra classes that serve as linking mediums. Good, you remember. There is a fourth app, a one-pager (web) app to be exact. It isn't important within the context of this book, however. Because it only accesses our database. It has nothing to do with how our apps interact with each other's data. Did I forget to tell you? Apps do not interact with each other's data. We save the data in a way that it can be retrieved as a whole.



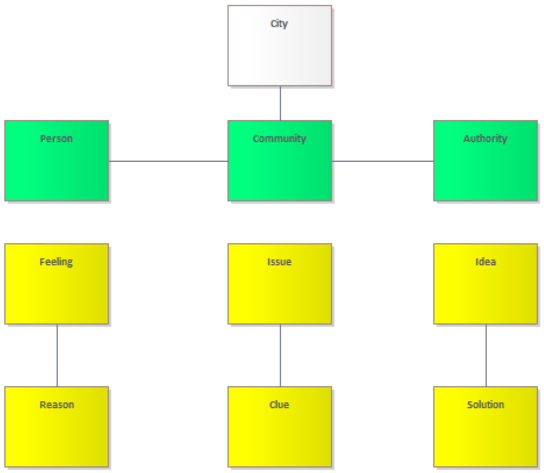
A superficial view of the key abstractions (1 of 2)

- *Linking with Verdict = "Verdict is about clues and issues. So, we link those with feelings and reasons."*
- *Linking with Notung = "Notung is about ideas and solutions. So, we link those with feelings and reasons."*

- *Linking old information = "Every information about the employee is at the disposal of the other apps".*



A superficial view of the key abstractions (2 of 2)



[Tips] A "city" is an ecosystem which may or may not be a city.

[Tips] An "authority" is someone who has power to do something.

[Tips] It's possible to have an abstract class at the top which has cities, like "World".

Of course, you can make changes or add new items to our list. If you do so, remember, there must be an overall theme. For example, in mine employee's feelings are reflected in her community's mood and firm's culture. In a way, these three findings represent the schizophrenic existence of a person who is trying to live in an unhospitable environment (Remember, Star Trek?) and never having the guts to simply leave the premises or act like a hero for once. Yet, absorbed in their tiny little worlds, they do not see those options. "No one can see beyond the choices he makes" (The Matrix). I have so many friends (!) in this condition. They all act like they are just about to find a cure for cancer as if they weren't the ones who have failed again and again and again for years and years and years. They act smart, but it is just a mask for cowardice. The only way to help them or to preserve our sanity is to program them! That's why you are here, people! And to preserve your sanity, you must seek nothing. Remember that!

Pages

Our goal is to have two things, "an A4t with minimal complexity" and "a template for all upcoming A4t(s)". Since Moody is the first one in the series, it will both serve as a "architectural template" and an example for our "programming style".

"Version 1.0"

- *Voting Page is used to receive the daily vote.*
- *Analyses (results) Page is used to display the consolidated votes at the end of that day's shift.*
- *Analyses Page also acts as a waiting room which will be used to keep the motivation of the voter high.*
  That's why employees can see the poll results as they come in.
  That's why the results are presented in a way that provides contrasts.
  We want to disrupt what's going on in the office, not just deliver the news in a boring manner.
  We want to create chaos and have fun!

"Version 2.0"
(optional)

- *A one-time Help Page will be used to provide a summary of the app's main functions.*
- *Trend Page is a chart that displays corporate culture (lows, highs, chronic problems) with respect to time.*
- *Mood Swings Page is a chart that displays the feelings of employees with respect to time.*
- *Customization Page is used to update the criteria and the communities.*
  (not optional)
- *Sign Up Page is a cover for LinkedIn Authentication.*
  "Current company" in LinkedIn profiles will be regarded as "Company".
  A one-time selection will be enforced before using the app, to specify one's community.
- *Sign In Page is a cover for LinkedIn Authentication.*

Functions

As you can see, the secret word is "minimal" around here. So, we will always try to accomplish more with less. A few carefully designed steps have greater implications than one may think of. That's what we are after. We will keep app's functionality easy to understand and even easier to modify. Remember the Pareto Principle. Whatever you do, twenty percent of it is always the most important bit. The rest is just a barrier between you and the love of your life. So, when you are done, don't just pick the next task on a list. Turn off your computer, stand up and walk away. Have a life.

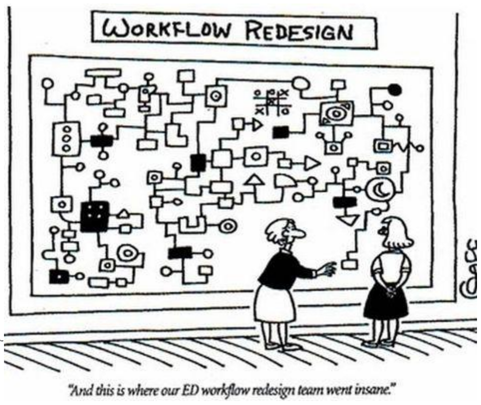
So, you have come back for more, huh? I can't blame you. One cannot get enough of a good thing. Causing chaos and having fun… what can be better than that?

Micro apps can be better understood through their functions. As you get more comfortable playing with them, you will see that you can arrange them in so many different ways. Also, modifying them will grant you additional powers. When I say "modifying", I don't mean spending hours. I mean spending minutes. Because like customizations, modifying A4t(s) is a breeze. By the way, "r" means "read", "w" means "write" and "c" means "compute".

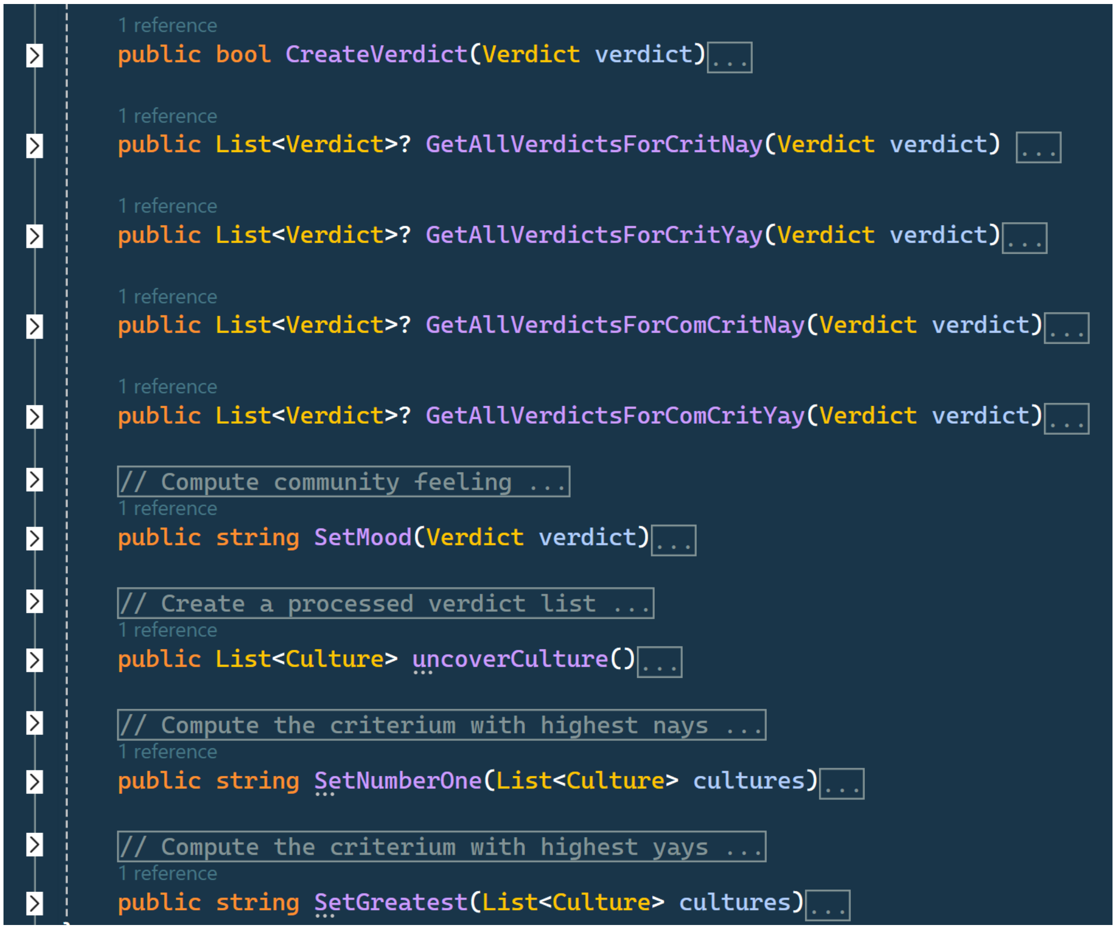
And understanding Moody begins with understanding our controller, "VotingController.cs".

We will go over those functions in detail when we get to develop Moody. Right now, all you have to understand is what's going to be in our toolbox, which also marks the limitations of our A4t. Every app (like every project) must have a clear border that shows what we are going to do and what we are not going to do. Gaining control over borders is your way to success. If you lose control over them, the other abilities in your skillset very rarely make a difference. You will lose.

"So after 2 years, our R+D team has come up with THIS process."
Start
Finish
E.A.
© PNMsoft


WORKFLOW REDESIGN
"And this is where our ED workflow redesign team went insane."

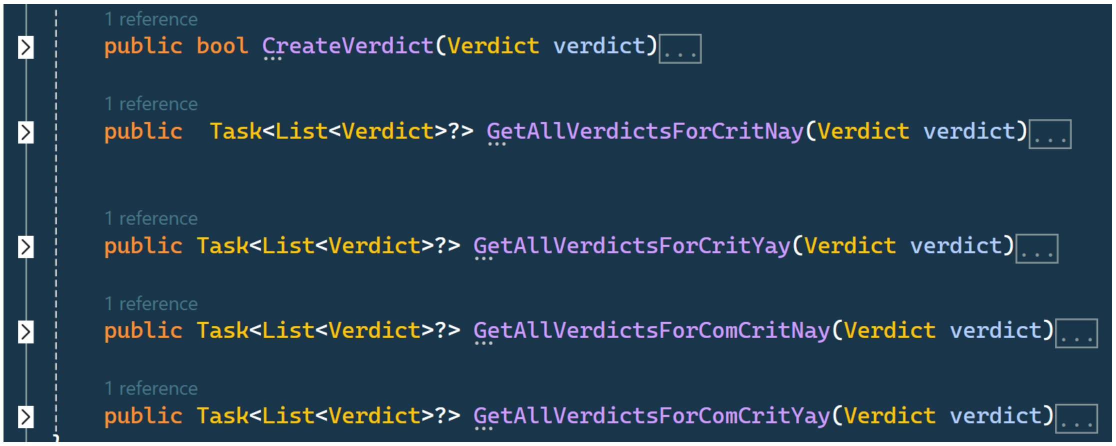
"Losing because of oversimplification" = "Losing because of complexity" = "Losing".



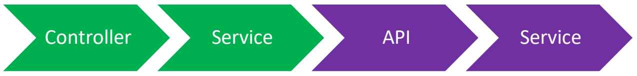
```
1 reference
public bool CreateVerdict(Verdict verdict)...

1 reference
public List<Verdict>? GetAllVerdictsForCritNay(Verdict verdict) ...

1 reference
public List<Verdict>? GetAllVerdictsForCritYay(Verdict verdict)...

1 reference
public List<Verdict>? GetAllVerdictsForComCritNay(Verdict verdict)...

1 reference
public List<Verdict>? GetAllVerdictsForComCritYay(Verdict verdict)...

// Compute community feeling ...
1 reference
public string SetMood(Verdict verdict)...

// Create a processed verdict list ...
1 reference
public List<Culture> uncoverCulture()...

// Compute the criterium with highest nays ...
1 reference
public string SetNumberOne(List<Culture> cultures)...

// Compute the criterium with highest yays ...
1 reference
public string SetGreatest(List<Culture> cultures)...
```

- *(w) We have a "verdict creation" function that gets a feeling with a reason and turns it into a verdict.*
- *(r) We have two "verdict retrieval" functions that compile every negative or positive verdict for a criterium in a company (i.e. ACME).*
- *(r) We have two "verdict retrieval" functions that compile every negative or positive verdict for a criterium in a community (i.e. Technology Department of ACME).*
- *(c) We have a function that computes the mood of an employee's community.*
  It gets all the negative and positive verdicts for a criterium and finds the one with the highest value.
- *(c) We have a function that preps verdicts for further computations.*
  It gets all the negative and positive verdicts for all criteria and creates a list of cultures where a culture is a record of a criterium and its associated number of negative and positive verdicts.
- *(c) We have a function that computes the criterium with the highest number of negative verdicts.*
- *(c) We have a function that computes the criterium with the highest number of positive verdicts.*

There! That's it. All the rest is for user interface manipulations or minor tasks which could have been done differently. There is a catch, though. To really understand what is going on, you must understand the functions these functions are depended upon… meaning, you must understand our service, "MilgramService.cs". Of course, you must also understand the API running in the background, but as I said before, that can wait for now.


```
1 reference
public bool CreateVerdict(Verdict verdict)...

1 reference
public  Task<List<Verdict>?> GetAllVerdictsForCritNay(Verdict verdict)...


1 reference
public Task<List<Verdict>?> GetAllVerdictsForCritYay(Verdict verdict)...

1 reference
public Task<List<Verdict>?> GetAllVerdictsForComCritNay(Verdict verdict)...

1 reference
public Task<List<Verdict>?> GetAllVerdictsForComCritYay(Verdict verdict)...
```

It is starting to make sense, right? Whatever we did in our controller has a counterpart here… and whatever we do here has a counterpart… yes, you've guessed it, in the API. A multilayered mobile app is nothing but a whisper game. The only difference is this time whatever being whispered into the ear of the first person is exactly the same as whatever the last person hears.

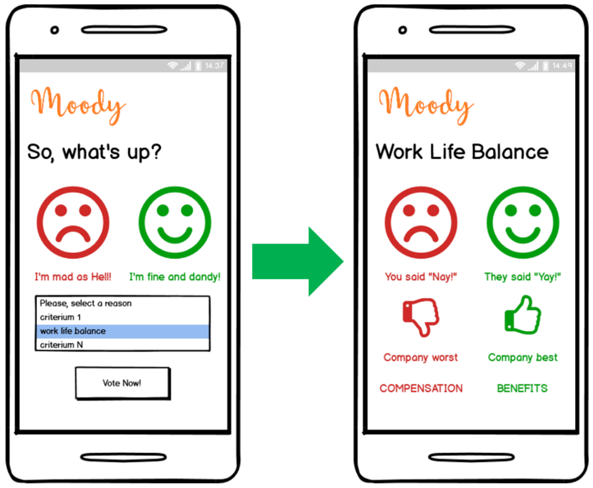


Maybe some of you have already figured this out. Yes, our API has a service too! And it is very similar to… the one that came before! So, why, oh, why are we doing all this? The main reason is to avoid conflicts of interest. Because in more complex apps, each of those layers are drenched in architectural mechanisms… and when you mix apples with oranges, it never ends well. Additionally, there are expert roles in bigger teams where one person only works on a single layer. So, this way updates are handled without causing unintended problems. There is one more reason which is a deal breaker. When you develop according to separation of concerns, you can change development platforms easily. Even if you don't change development platforms, it makes it so much easier to change components when you must. One more *thing, you can create mockup apps, apps without server counterparts in record time. You can use these apps to promote your idea. When it is time to go ahead and finish the app, you can use that mockup app as a starter app. Neat, right?*

Didn't I tell you? I love giving people bad news. These days you may see job posts seeking Full Stack Programmers. What this really means is all the more interesting components of your upcoming app were already programmed by a vendor handing you a free (!) development platform. This reduces a programmer into a diminished human being who is never curious about what the heck is going on in those frameworks and considers a software career as the ability to memorize SDK updates. "Oh, Hamlet, what a falling off was there!"

The Gathering Storm

Now that I've got your attention, let's focus on the first use case "Broadcast Feeling" in greater detail by fleshing out all the requirements. Why stop there? We will go over the "user interface design" and "business ecosystem implications" as well. Here our success criterium is to have a chapter that can be used by programmers without further analysis.

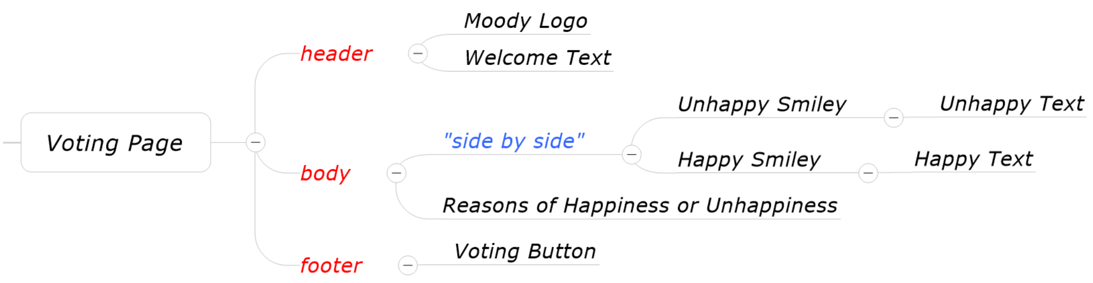


Since we are using a two-page design, one page (Voting Page) aims to provoke a feeling and cause an action that makes it visible to the public anonymously on the other one.

The other page (Analyses Page) both acts as a waiting room and an analysis room. Employees, including human resource managers and those in C-Levels, all can see the votes as they come in. Also, they can see votes in terms of feelings, moods and cultures.

These findings (over time) give everybody the opportunity to reflect upon them and wake up ignorant masses including but not limited to incompetent or selfish managers.

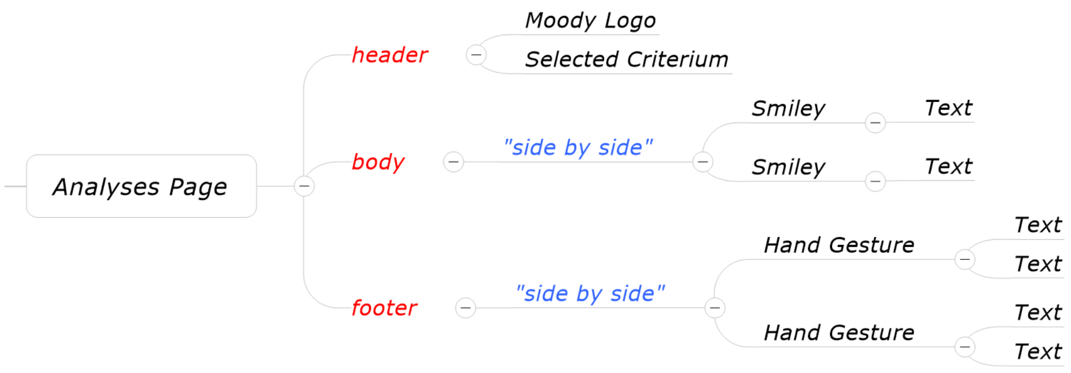
In this first version of Moody, we don't differentiate between employees and managers. They are represented by the same actor (Employee). Keeping that in mind, let's look at the components of each page in detail. The first page (VotingPage.xaml) has 6 sections.



- *App logo = I prefer to have the text version at this time. We can upgrade it (!) to an image later. [font = Breetty] [color = FF6A00]*
- *Welcome text = A stationary text that sets the mood = "So, what's up?"*
- *Unhappy Smiley, Unhappy Text = A PNG image which can be upgraded (!) to a SVG one later.*
  I'm quite happy with the accompanying text, though. "I'm mad as Hell!"
  My inspiration was the Sidney Lumet's film Network (1976) = YouTube clip.
- *Happy Smiley, Happy Text = A PNG image which can be upgraded (!) to a SVG one later.*
  Again, I'm happy with the accompanying text. "I'm fine and dandy!"
  My inspiration was a George Carlin parody where he discussed whether one can be both at the same time.
- *Criterium = One of 12 customizable reasons of happiness or unhappiness.*
- *Voting button = Once you have tapped a feeling and selected a reason, you are good to go!*

I don't know whether you are cut for this, but all we computer scientists do is do the same things over and over as we get deeper and deeper into salvation (or damnation). This is what makes us like the whirling dervishes. We too believe we have found something worthwhile, and we too want to show off. Funny, isn't it? One of the most famous religious sects in the world is famous not because its disciples are humble or wise, which may or may not be the case, but because they just love to show off.

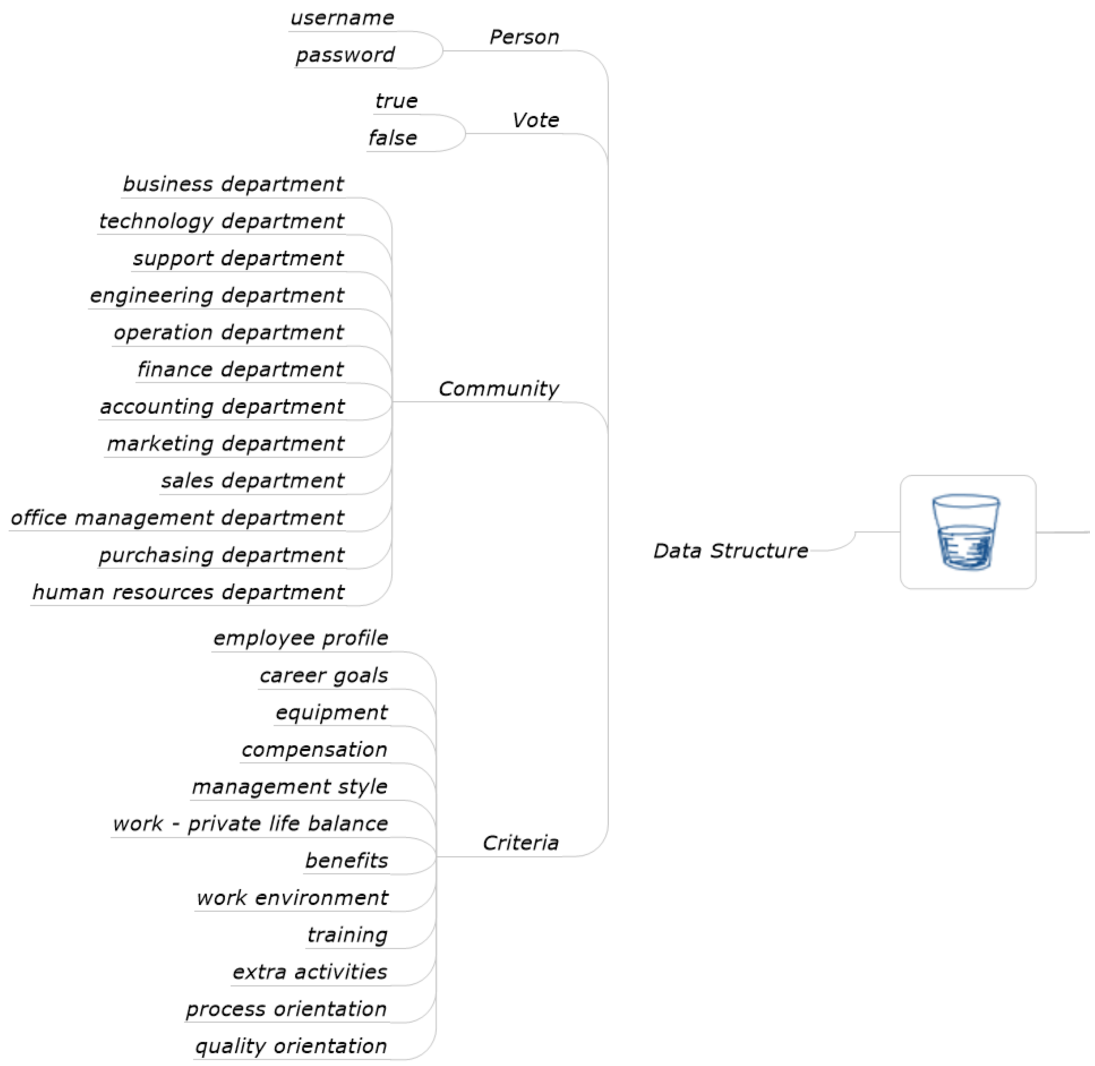
The second page (AnalysesPage.cs) has 6 sections too! Yes, I have a thing for repetition.



- *App logo = I prefer to have the text version at this time. We can upgrade it (!) to an image later. [font = Breetty] [color = FF6A00]*
- *Selected Criterium = This is the criterium voter has chosen as the reason behind her feeling.*
- *Current Feeling of the employee.*
  Unhappy or Happy Smiley, Unhappy or Happy Text = Depends upon the selected feeling and criterium duo. i.e. "You said Nay!"
- *Current Mood of the employee's community.*
  Unhappy or Happy Smiley, Unhappy or Happy Text = Depends upon the community's cumulative feeling. i.e. "They said Yay!"
- *Number one issue = A negative hand gesture usually associated with Roman emperors pointing out that a fallen gladiator must be slayed. Underneath, there is a descriptive text = "Company worst" which is followed by the name of the relevant criterium.*
- *Greatest achievement = A positive hand gesture usually associated with Roman emperors pointing that a fallen gladiator must be let go. Underneath, there is a descriptive text = "Company best" which is followed by the name of the relevant criterium.*

Stop for a minute and think about that office. It may be any kind of office, but one thing is for sure, it's a big one. Otherwise, people wouldn't bother using an app like Moody. It must be big. There must be several different kinds of tribes there, all in competition with each other. There must be a dysfunctional management team, and a human resources department willingly acting as the long arm of the law or something like that. Better yet, there must be a few similar companies in the same city. Those working in ours should have friends there. And they would all try to get to another company, their dream company regardless of whether such a thing exists or not. That's the best scenario for the business edition of Moody. This way we can create chaos that involves all kinds of people, not just for those in a miserable company. When it comes to chaos, the bigger the better. It's like wildfires. Who would in his right mind stop and watch a small bush burning. On the other hand, who wouldn't watch a city burning?

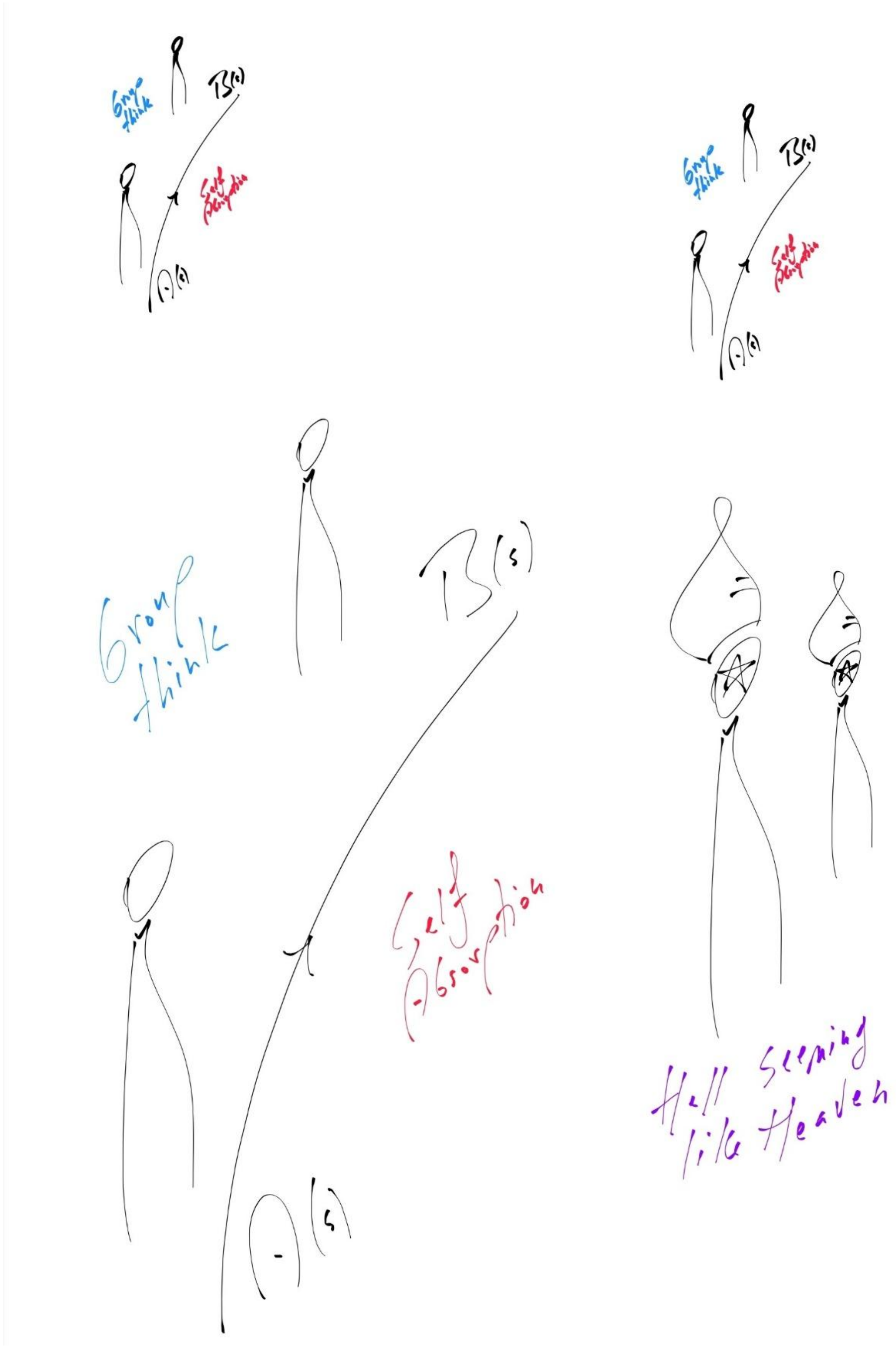
Understanding functions requires you to understand data, how it is created, why it flows from one place to another, how it is processed, how it is transmitted and how it is consumed. In our tiny little app, most data end up on a page. Others are kept in the archives waiting for a more feature packed version of Moody. Yet, others are there just to *feed other apps. You may want to add the feelings, thoughts and actions of those who are not using our app to the mix. This way you get a better view of the big scheme of things.*

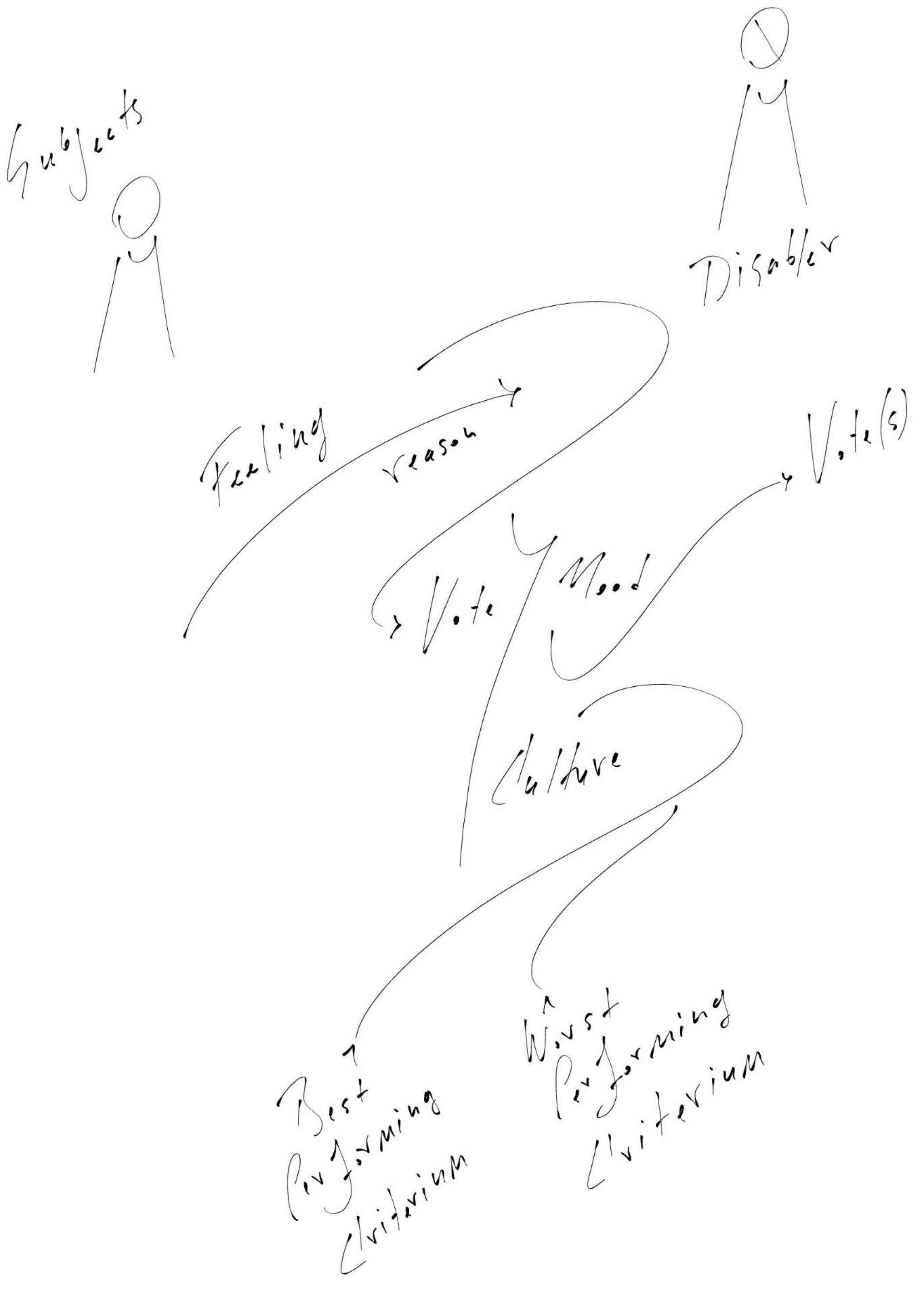


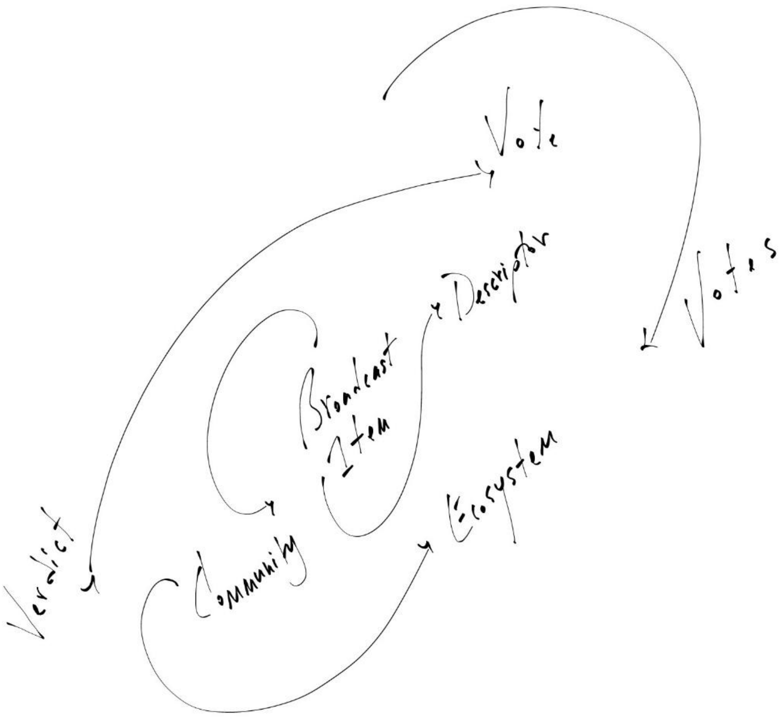
- *Person = An employee feeling miserable regardless of her title or rank.*
- *Vote = Verdicts which are consolidated into permanent records with date-time info at the end of shifts. An attribute acts like a flag whether an employee cast a vote or not. Only a single vote per day is permitted.*
- *Community = We divide every ecosystem into 12 subgroups, each of which may be a business department.*
- *Criteria = We define the relationship between public and those in power by 12 happiness / unhappiness generators. You can see them as items in a business contract.*

To me "apps, ecosystems and actors living there" are all one thing. So, I often take a look from above and relate things at that level. If things make sense at the cosmic (!) level, spending time on the details isn't a waste of time. If you look at the pictures below, you can see "key abstractions" and "the shades of feelings" in the Moody universe so to speak.

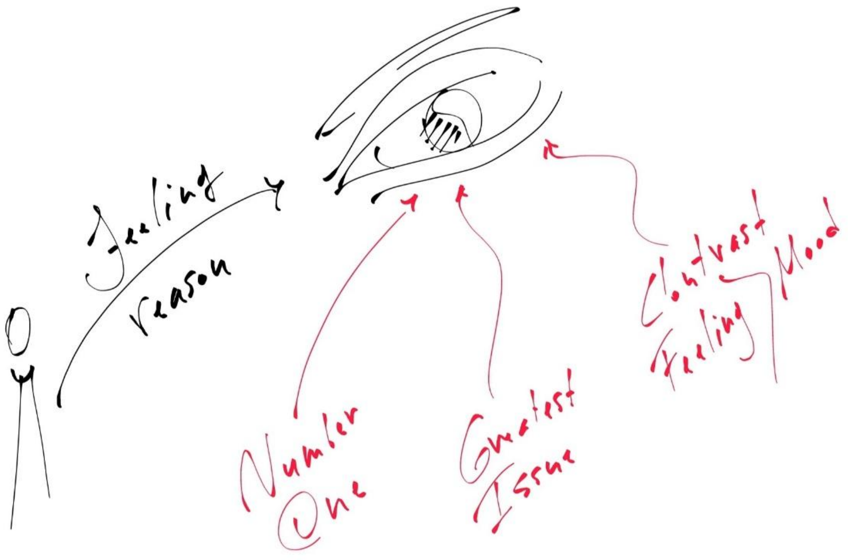
As you can see from the first one, those who lose their way support each other for becoming more and more delusional. And the few who gained some kind of autonomy and power willingly forget who they were, what they have been through and how they are getting their daily bread now. Being greedy, having no problem faking it and the willingness to cherry pick memories create a diminished human being… who is content to pursue a powerless life for more toys.




Subjects
Disabler
Feeling
reason
Vote
Mood
Vote(s)
Culture
Best Performing Criterium
Worst Performing Criterium



"Generic Moody Framework"



"Two-page dynamics of Moody"

Plan

In computer sciences you can change the order of events. Reactions may come before actions. That is not to say apps should act weirdly. What I'm trying to say is you can pick a weird order to do things. For example, I always start from the trickiest part. I don't waste my time working on basic building blocks even if I don't have any reusable components. When it comes to reusable components, I'm an ID Software kind of guy. I throw away reusable (read, "trivial") components into the trash bin at the end of every project. On the other hand, when you are done, the result shouldn't be weird. To be a computer scientist is like being a mathematician or even better, like being a philosopher. Both prefer endless canvases, but small, elegant solutions are what really captivate them. You can leave an unfinished symphony behind. You can let those with lesser skills do the dirty work for you. Thanks to Mussorgsky, that's my philosophy!

North by Nortwest, Alfred Hitchcock, 1959.

While artistry is required to bring all kinds of different skills together to create a good product, this isn't art. It must accomplish something. Awakenings are not enough on their own. The equivalent of Steve Roach's captivating music in computer sciences is plain boring. Also, a software product must be easy to use. It must help those who are using it to get to wherever they are trying to go. "Software is the thing that gets you to the thing" (Halt and Catch Fire).

However, this time I will play nice. We will start with the user interfaces, because our app is tiny. We don't have to take precautions to prevent possible future complications. We can just start from the easiest task and proceed as we see fit.

"Develop app".

- ***Step 1 =** Prep your development environment*
- ***Step 2 =** Create the presentation layer = "XAML pages".*
- ***Step 3 =** Create the business layer = "controller and model classes".*

- *Step 4 = Create the service layer = "intermediary classes".*
  "Proceed with the API".

Step 1

The first thing we will do is to create a ".NET MAUI app". To be able to do that you have to download a few apps.

- *(our main IDE) Visual Studio Community Edition = Link*
- *(our secondary IDE) Visual Studio Code = Link*
- *(database) MongoDB Community Edition = Link*
- *(note taking app) Notepad++ = Link*
- *(yet another note taking app) Sublime Text = Link*
- *(mind map app) XMind = Link*

I like to have alternative ways to do the same thing, so that I can see what I'm doing in different ways. That's why I use two IDE(s) and two note taking apps. Am I weird or what? Trust me, soon you will see that it helps. It also prevents you from becoming a fan, which is worse than being blind.

You may also download the following apps. If you end up liking them, you must purchase them, though. I won't require you to use them. These are the tools of the pros, folks!

- *(a somewhat better (?) mind map app) Mind Manager = Link*
  Android and iOS versions are free.
  Free alternative = XMind = Link
- *(a UML app) Sparx Systems Enterprise Architect = Link*
  Free alternative = Visual Paradigm Online = Link

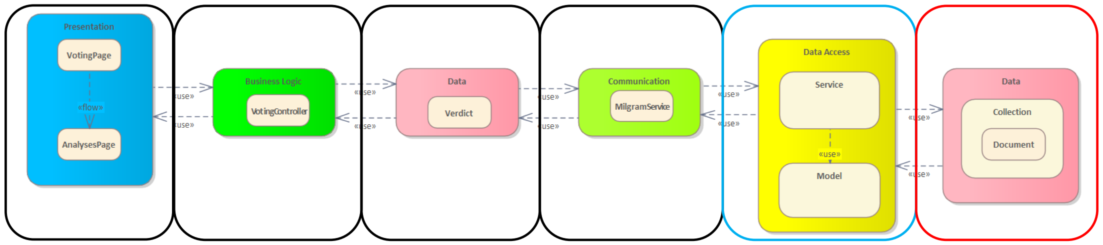
Voting Day

Steps 2 - 4

*While being intriguing, all we have done until this moment is… just talking. Let's put what we have come up with to good use. Let's develop a **five-layer cross-platform mobile app** from scratch. Don't worry. It's easier than you think!*

Flashback

We won't invent new approaches here. We will use a popular approach that will serve us well. Starting from what one sees when she opens Moody on her phone, there will be four layers: "View", "Controller", "Model" and "Service". Views are pages. Controllers are intermediary classes that act like traffic police. They accomplish nothing on their own, but they know where all the roads go. Models are entity classes that hold information for a variety of reasons. Service is a single class that communicates with our API (5th layer) which is in charge of accessing our database. Controller or Service Layers may be more complicated, of course. However, one class for each is more than enough in our tiny little app. This may surprise you, but this approach is more than enough for many small apps. So, you are in good hands here.



"Moody (presentation, business, temporary data, communication)" <-> "Milgram [data access]" <-> "Berry [stored data]"

App (a .NET MAUI app)

1) *Presentation Layer (View)*
   - ➢ *XAML pages with visual controls such as labels, buttons, grids, frames, and pickers.*
   - ➢ *Code behind files of pages with almost no code except user interface component manipulations. If we preferred a "fat client approach", everything in controllers or services would be here.*
2) *Business Logic (Controller)*
   - ➢ *Classes that receive, retrieve or save information to perform computations on them or to transfer them to appropriate objects and components.*
3) *Data Layer (Model*)*
   - ➢ *Classes that hold data temporarily which also summarize the whole system when considered as a whole.*
4) *Communication Layer (Service)*
   - ➢ *Classes which receive and transmit data between disjointed systems (i.e. app on your phone, app on a server).*

"API" (a .NET Core API)

5) *Data Access Layer (API) which has "Communication" (Service) and "Data" (Model) layers of its own. The default 'main' class, "Program", acts like a "Controller".*

"Database" (a NoSQL database)

6) *Data Layer (Database) which has "collections" of "documents" that represent the "objects" of the "app".*

**A "model" is a more closely coupled information than a "class". At least, this is how I see it. It makes run time collaborations simpler, sacrificing some of the generic qualities of associated classes. Some programmers never use classes with models. I do. Don't get it wrong. Models are classes too… just, they are not your mother's / father's classes.*

[Tips] You don't have to follow someone else's notion of good architecture. Good architecture must be good for you first and foremost.

The First Round

If you are anything like me, you'd look for a visual editor first. However, not only are visual editors clumsy, but also when it comes to cross platform apps, they aren't that dependable. What's more, even if it sounds highly unlikely, designing pages in a code editor is really… easy. That is, unless you are not an ergonomics or a beautification freak. On the other hand, if you were one you wouldn't be reading this book. You would be knee deep in user interface trivialities that would bore me to death. While I'm too sexy for my own good, don't worry, ordinary looking guys/gals can still attract the pretty ones. You do have fully functional brains after all, don't you? So, don't knock yourself out with user interface design. It's for mere mortals. Ability to do something well always wins over being cute.

Let's create the first page of our app (Voting Page) and pass "feeling" and "criterium" to the next page (Analyses Page). Remember, we have only two pages in this tiny app. Regardless, there are four topics we must cover. Let me first introduce them and then we will tackle them one by one.

Voting Page		Analyses Page	
(Page One)		(Page Two)	
App Name = "Moody"		App Name = "Moody"	
"What's up?"		[Selected Criterium]	
Happy Emoji	Unhappy Emoji	You said [feeling]!	They said [mood]!
"I'm fine and dandy!"	"I'm mad as Hell!"	Number one issue is [criterium].	
Button = "Vote now!"		Greatest achievement is [criterium].	
Shortcuts ("optional", 2.0)		Shortcuts ("optional", 2.0)	

"Broadcast how you feel every single day until you drive all managers crazy"

- *Voting Page is where you are prompted to provide how you feel and why.*
- *Voting Page Code Behind is closely tied to the page and does not move an inch further.*
- *Verdict class is equivalent to vote class. The only difference is here we ignore the date-time property. At the end of every shift, verdicts are converted to votes for archiving purposes and further analysis.*

"View feelings, moods and cultures in contrast to each other as verdicts come in"

- *Results Page is where you view your vote (feeling), your community's vote (mood), number one issue (culture) and greatest achievement (culture) of your firm.*
- *Results Page Code Behind is closely tied to the page and does not move an inch further.*
- *Mood class is the cumulative feeling of the employee's community such as Marketing Department. (i.e. "voter" versus "voters of a community".)*
- *Culture class is the cumulative feeling of the entire company.*
- *(2.0) You can use the archive to uncover personalities, mood swings and cultural trends.*
- *(2.0) You can monitor your company's ranking as an (un)happiness provider in your [selected area].*

"Handle business logic in a separate layer"

- *Voting Controller class has the business logic of both use cases. Since scenarios of these use cases are closely linked, we only need one controller to manage both processes. In more complex scenarios, we may need a controller for each use case regardless of the number of associated pages. Also, the simple computations done here can be migrated to API when server-side computations make more sense. It is trivial here because of the size of our app and data. Your phone may be a lot faster than your server… if you are cheap (like me). Keep that in mind.*
- *(2.0) Later we may add another controller class which can be used by the other apps in the trinity. This way, we can consolidate findings gathered in all those three apps (Moody, Verdict, Notung). Think about this as "Emotional Agility Controller" which has clones in all three apps, together making sure data generated in those apps remain consolidated and coherent… ready to be consumed by consolidators.*

"Access API in a separate layer"

- *Milgram Service Class is how we access our API (which accesses the database for us). Think about it this way, our service class asks for data and gets data. However, the real component that does those things is our API. Yet, another manifestation of separation of concerns. Apps, data access mechanisms and data exist on their own which allows us to manage them individually. Remember, the other two apps in the trinity (Verdict, Notung) will be able to use the same API and database easily without doing anything extra. That's the main goal here. Do it once and use it multiple times. Keep it simple and stupid (C. Hu*).*

**"Who are you going to see when you have a problem, Mr. Bozkurt? C hu? See me." (My college advisor)*

The first thing we should do is to change [the default page] of our mobile app. This urge makes one realize that there are page files and code files closely associated with them. Every page comes with a code behind file which can be used *to manage what takes place on the page. While you can do almost anything in a code behind file, we will restrict code behind files to handle data entry, ensure data accuracy and enforce user interface rules. Because if you do more in a code behind file, it will become increasingly difficult to maintain your app in the future. We call this concept, "separation of concerns". Think about a friend you like to consult. If you had to ask everything to her, it would be tedious for you both, right? You should ask her about a few things and handle the rest on your own or with other friends. What we are doing here is very similar to that. Yes, I know. Balancing coupling or cohesion is a bitch. Well, follow my example, then.*

When you create a .NET MAUI project, the default page is set to "Solution \ Project \ MainPage.xaml".

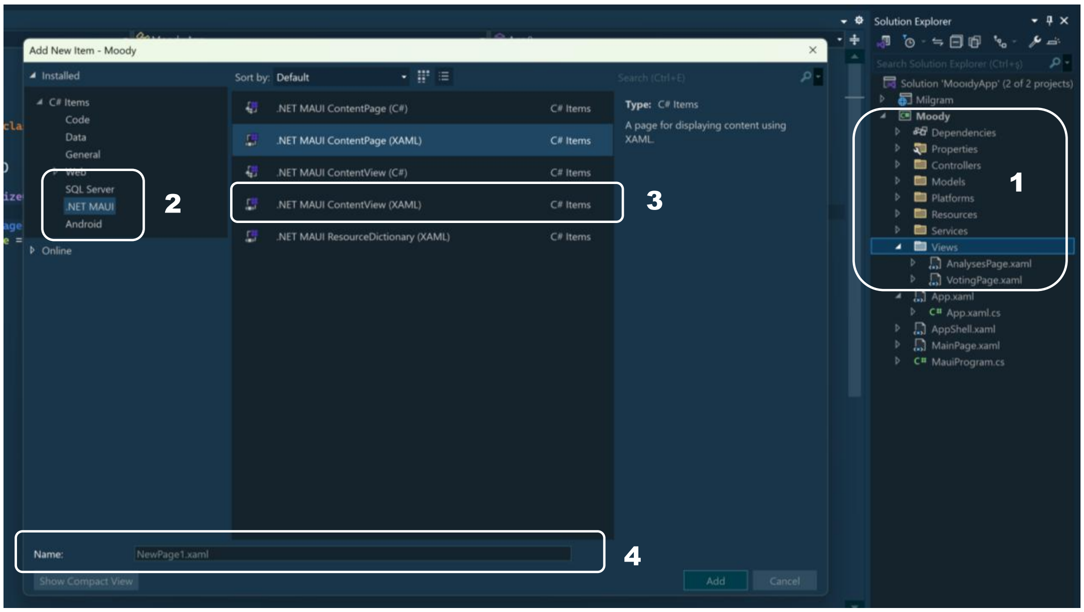
- *MoodyApp \ Moody \ MainPage.xaml => Automatically created default first page.*
- *Moody App \ Moody \ App.xaml \ App.xaml.cs => App settings.*
  - *"MainPage = new AppShell();" is the command that sets the first page.*

We will change this command as follows:

- *MainPage = new NavigationPage(new VotingPage());*

Of course, for this to work, we must have a "VotingPage.xaml". So, create a new folder called "Views" under "Moody". This is where we will add all our pages. You can translate "views" as the Presentation Layer of your mobile app. Pages and their closely linked code behind files will be there. And as the "separation of concerns" principal dictates, when you modify these pages, your Business (Controllers), Data Access (Services) and Data (Models) Layers should remain intact. This is exactly what we will be doing here. Even if our app is tiny, we don't want to work more than we have to.

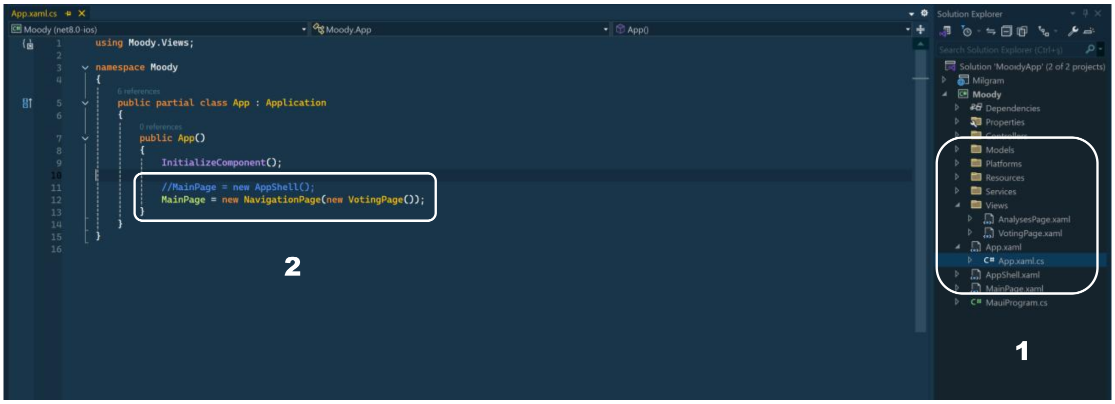
Right click the "Views" folder and add new item. Select ".NET MAUI" and then, ".NET NAUI ContentPage (XAML)". Finally, name it "VotingPage.xaml". There, you created your first page!



Create a new page.

At this point, you may be wondering why I'm not sharing code as text, you know, so that you copy and paste like crazy. Oh, well, that was my answer in the form of a question, pal. You have to write it. Unless you write the whole thing and understand the basic mechanisms behind it, this is a futile attempt to change the world. Don't worry, you don't have to understand the whole thing. I don't. Nobody does. Yet, being a driver who doesn't know anything about cars is just *lame. Now that we got this out of the way, where were we? Oh, yes, we were designing the main page of Moody. Think about a page that provokes its actors (we don't use the word user here) and triggers them to vote each and every day. It's like children in the backseat, nagging nonstop during your 10-hour driving vacation, "Are we there yet?". It's the best punishment for upper management, don't you think? NEEHEEHEEHEEHAHAHA!*

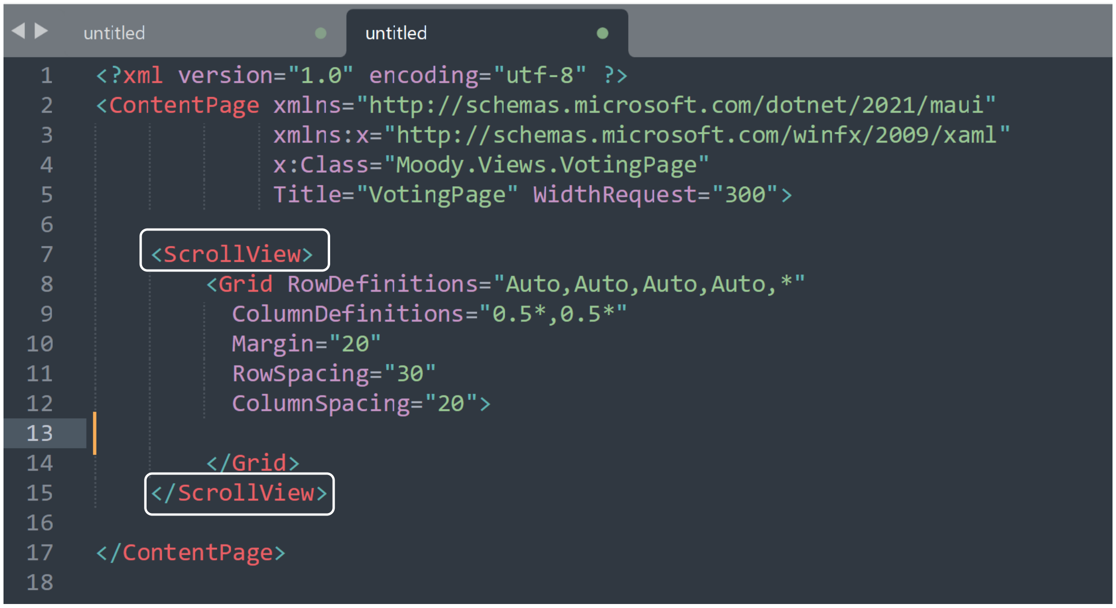
What? "Why can't we just write new VotingPage();", you say, huh? Well, we must make our pages traversable. We should encapsulate them as "Navigation Pages". That's why. This is programming. Not everything is auto pilot material, pal.



```csharp
using Moody.Views;

namespace Moody
{
    public partial class App : Application
    {
        public App()
        {
            InitializeComponent();

            //MainPage = new AppShell();
            MainPage = new NavigationPage(new VotingPage());
        }
    }
}
```

Change default first page.

Go to your newly created "Voting Page" and locate <ContentPage></ContentPage> section. In a XAML page everything begins and ends with a tag. The first one tells that a new section is beginning (<something>), and the second one tells that it is ending (</something>). You will put all the goodies between these Content Page tags.



```xml
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="Moody.Views.VotingPage"
             Title="VotingPage" WidthRequest="300">

    <ScrollView>
        <Grid RowDefinitions="Auto,Auto,Auto,Auto,*"
          ColumnDefinitions="0.5*,0.5*"
          Margin="20"
          RowSpacing="30"
          ColumnSpacing="20">

        </Grid>
    </ScrollView>

</ContentPage>
```

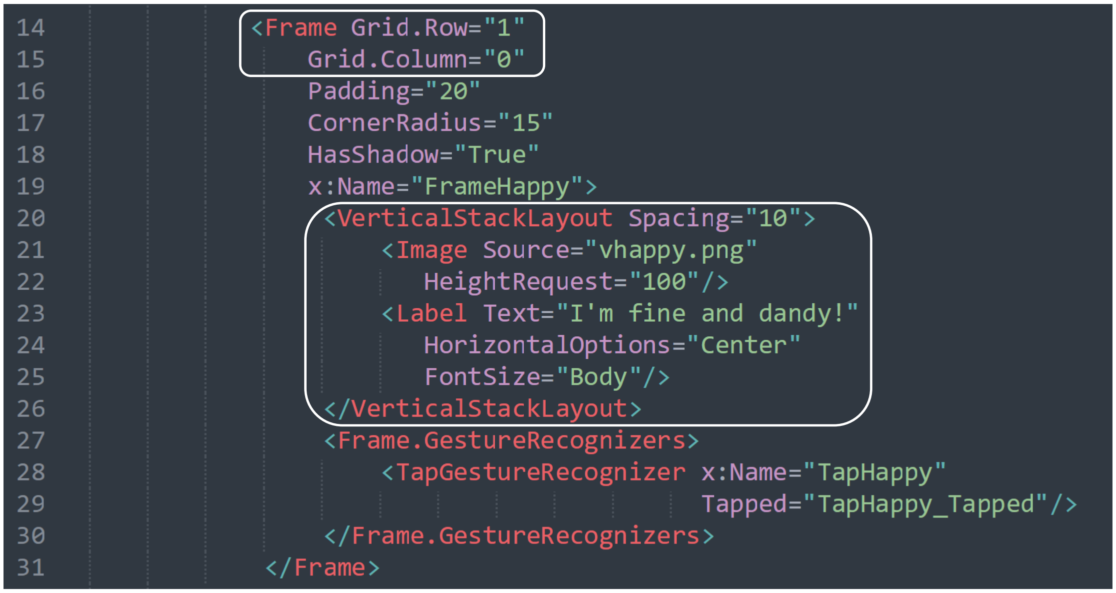
Add a scroll view which basically lets every bit of content to scroll up or down as you swipe up or down on your smartphone screen. Between <ScrollView></ScrollView> tags, add a grid which lets us easily organize different user *interface components horizontally, vertically or both. That's why it's my all-time favorite. I must be a grid fool, because I organize everything in grids. Come, join me! Add <Grid></Grid>. To make it prettier add the attributes with the suggested values as shown in the picture above. The first two attributes are about responsiveness. The remaining three are about placement. No, I won't tell you more about them. If you must know, check the XAML dictionary (Link).*

Add a <Frame></Frame> section just above </Grid>. This will serve as our first "Feeling Trigger". We will provoke the actor by placing a positive emotion on the left. The way to trigger actors depends on your ecosystem. Some are triggered when they come across an opposite point of view or feeling. Others do so when they meet things that resonate with how they think or feel. If you want to find out the inclinations of your subjects, chat with them and figure them out.

<Frame></Frame> has two elements, a picture and a text. Since I'm a practical kind of guy, I went with smileys here. You can go your own way with this. However, apply some caution to what you will do here. Because even if it's just a combination of a simple picture and a short text, we humans are irrational beings. We can be talked into doing anything. You don't want to overstimulate your subjects, for what they will do afterwards may no longer reflect who they are.

ahappy.png
aunhappy.png
ga.png
no.png
vhappy.png
vunhappy.png

- *Image: "vunhappy.png" (as in "verdict unhappy…")*
- *Text: "I'm mad as Hell!"*



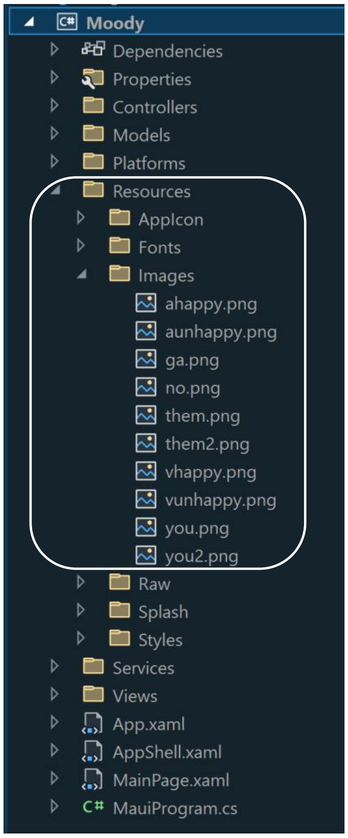
```
<Frame Grid.Row="1"
    Grid.Column="0"
    Padding="20"
    CornerRadius="15"
    HasShadow="True"
    x:Name="FrameHappy">
    <VerticalStackLayout Spacing="10">
        <Image Source="vhappy.png"
            HeightRequest="100"/>
        <Label Text="I'm fine and dandy!"
            HorizontalOptions="Center"
            FontSize="Body"/>
    </VerticalStackLayout>
    <Frame.GestureRecognizers>
        <TapGestureRecognizer x:Name="TapHappy"
                              Tapped="TapHappy_Tapped"/>
    </Frame.GestureRecognizers>
</Frame>
```

We will use this structure more than once. A frame in a grid holding a vertical stack layout... This way you can position user interface elements in appropriate cells of your grid to achieve a certain look. To enable interaction, we will be using "tap gesture" which is the default way to interact with smart devices. When such an interaction occurs, we will pass the data this interaction creates to what's coming next.

- *<VerticalStackLayout>< VerticalStackLayout />*

You can use this layout to stack UI components on top of each other. When I need to place user interface components side by side, I still use this layout. Because their being side by side is handled by the grid too.

- *<Image/> and <Label/>*
  Images are used to strengthen what the associated text says. You don't even need to read the text. Text associated with the images are designed to evoke strong emotions in a binary manner.
  I'm skipping the obvious stuff as usual. Well, I don't like origin stories either. So, sue me!



```
Moody
  Dependencies
  Properties
  Controllers
  Models
  Platforms
  Resources
    AppIcon
    Fonts
    Images
      ahappy.png
      aunhappy.png
      ga.png
      no.png
      them.png
      them2.png
      vhappy.png
      vunhappy.png
      you.png
      you2.png
    Raw
    Splash
    Styles
  Services
  Views
  App.xaml
  AppShell.xaml
  MainPage.xaml
  MauiProgram.cs
```

(clipart) Of course, for anything with pictures to work, there must be pictures. So, go to your gift pack and locate your pictures in the "Clipart" folder. Then, click "Resources" folder in your "Moody" project. Right click "Images" folder and "Add existing item". Browse to your "Clipart" folder and import everything by selecting all (6 pics).

(grid placement) While being the most trivial, an important aspect of grids is both columns and rows start at zero. So, we will select the first column and second row for our Negative Feeling section (column 0, row 1). You can just go ahead and copy the other attributes, because they make it prettier.

(frames) As you can see, frames are actionable, and we wouldn't want them any other way. To be able to use any actionable item with ease, we must name it. Naming an item is easy. Just write…

- *x:Name = "FrameHappy"*

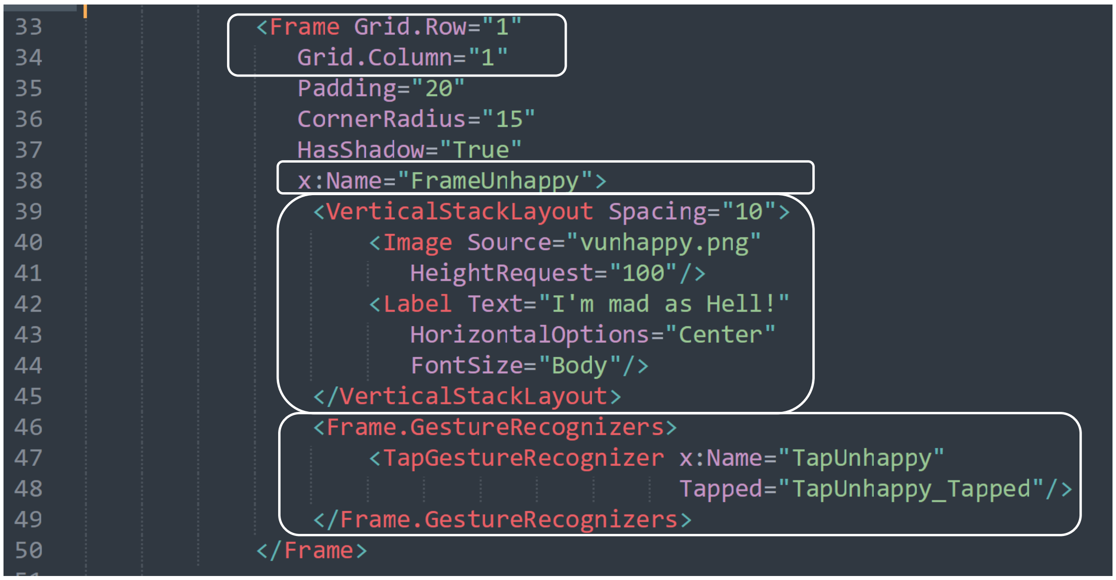
After you did that, refer to that item within the context of an event like so…

- *<Frame.GestureRecognizers>*
- *<TabGestureRecognizer/>*
- *</Frame.GestureRecognizers>*

This way, when one taps a Smiley, we will keep track of it until you complete the voting process which includes one more step (revealing the reason behind your feeling).

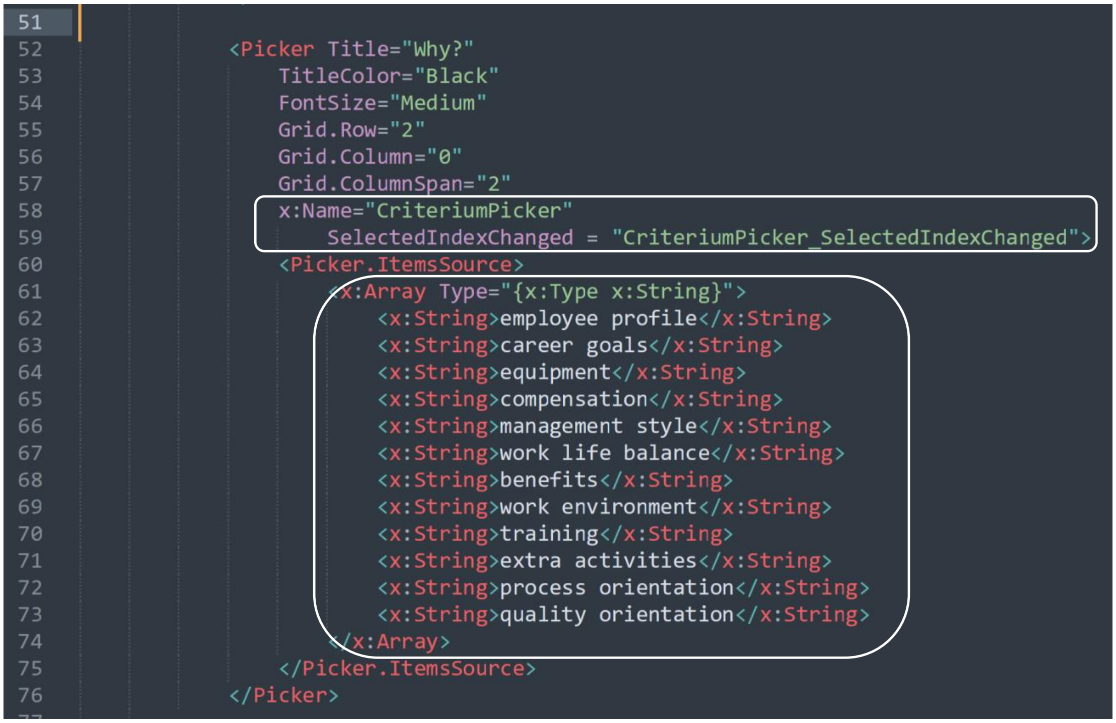
Let's add the unhappy element similarly. Do I have to tell it again? No! And, "Yes". You can reverse the order of these.

The scene where Howard Beale gives us one of our slogans (Network, Sidney Lumet, 1976).



```
<Frame Grid.Row="1"
   Grid.Column="1"
   Padding="20"
   CornerRadius="15"
   HasShadow="True"
   x:Name="FrameUnhappy">
    <VerticalStackLayout Spacing="10">
        <Image Source="vunhappy.png"
           HeightRequest="100"/>
        <Label Text="I'm mad as Hell!"
           HorizontalOptions="Center"
           FontSize="Body"/>
    </VerticalStackLayout>
    <Frame.GestureRecognizers>
        <TapGestureRecognizer x:Name="TapUnhappy"
                              Tapped="TapUnhappy_Tapped"/>
    </Frame.GestureRecognizers>
</Frame>
```

Good. Now, we have a page with two options to tap, “happy” and “unhappy”. I like dualities. Also, I’m a binary kind of man. I like zero and infinity, but everything in between just irritates me.



```
            <Picker Title="Why?"
                TitleColor="Black"
                FontSize="Medium"
                Grid.Row="2"
                Grid.Column="0"
                Grid.ColumnSpan="2"
                x:Name="CriteriumPicker"
                    SelectedIndexChanged = "CriteriumPicker_SelectedIndexChanged">
                <Picker.ItemsSource>
                    <x:Array Type="{x:Type x:String}">
                        <x:String>employee profile</x:String>
                        <x:String>career goals</x:String>
                        <x:String>equipment</x:String>
                        <x:String>compensation</x:String>
                        <x:String>management style</x:String>
                        <x:String>work life balance</x:String>
                        <x:String>benefits</x:String>
                        <x:String>work environment</x:String>
                        <x:String>training</x:String>
                        <x:String>extra activities</x:String>
                        <x:String>process orientation</x:String>
                        <x:String>quality orientation</x:String>
                    </x:Array>
                </Picker.ItemsSource>
            </Picker>
```

(picker) The next step is to provide a list of reasons which will be implemented in a better way in the future (2.0). Until then, we will hardcode a simple list in the "VotingPage". I call this list "Criteria" which makes each item a "Criterium".

Right after </Frame> tag, add the list above which is compiled by going over typical business contracts between employees and employers. On the other hand, this is just a generic list which can be turned into any list of items as long as they are used as conditions of agreements between individuals and their communities. In my version, I target big businesses. In yours you can target anybody anywhere. Why did I pick that ecosystem? Well, I live there! All my subjects are there! Remember, "exploitation begins in the family" (Ferengi Rules of Acquisition).

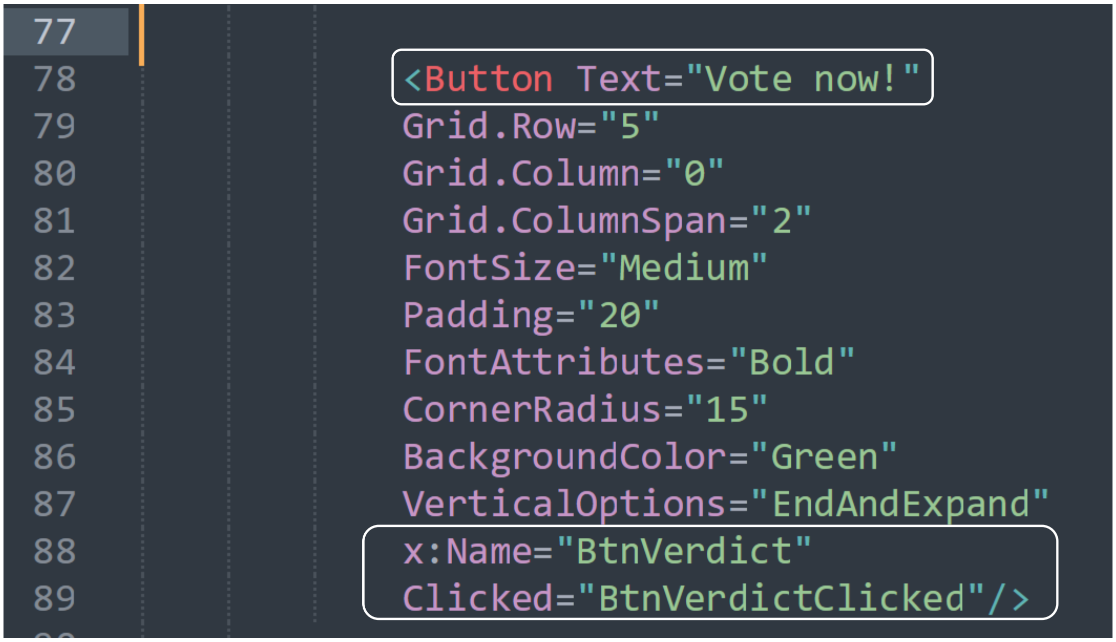
Since picker is actionable too, we should name it.

- *x:Name = "CriteriumPicker"*

Right under this statement, there is an event handler pointing to a function in the code behind file (VotingPage.xaml.cs). Name it as suggested below.

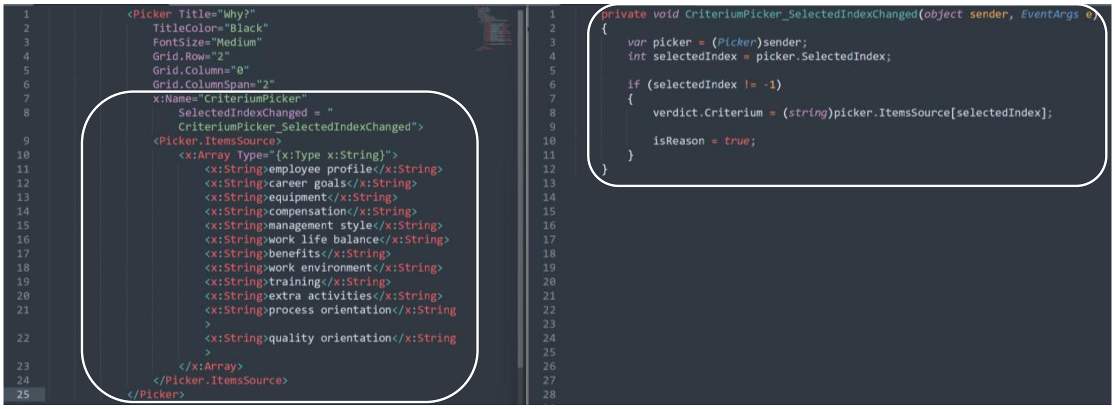
- *SelectedIndexChanged = "CriteriumSelected_SelectedIndexChanged">*

Remember, events taking place on the page are handled in the code behind file. Events that don't end there (like steps or processes) are carried onwards via controllers… which implies that values or objects you will need later, should be passed to the controller in question.



```
<Button Text="Vote now!"
Grid.Row="5"
Grid.Column="0"
Grid.ColumnSpan="2"
FontSize="Medium"
Padding="20"
FontAttributes="Bold"
CornerRadius="15"
BackgroundColor="Green"
VerticalOptions="EndAndExpand"
x:Name="BtnVerdict"
Clicked="BtnVerdictClicked"/>
```

You are getting this, right? "Names", "Function Calls" and "Functions" go hand in hand. While the first two are on your pages, the remainder is in your code behind files.



```
<Picker Title="Why?"
    TitleColor="Black"
    FontSize="Medium"
    Grid.Row="2"
    Grid.Column="0"
    Grid.ColumnSpan="2"
    x:Name="CriteriumPicker"
        SelectedIndexChanged = "
        CriteriumPicker_SelectedIndexChanged">
    <Picker.ItemsSource>
        <x:Array Type="{x:Type x:String}">
            <x:String>employee profile</x:String>
            <x:String>career goals</x:String>
            <x:String>equipment</x:String>
            <x:String>compensation</x:String>
            <x:String>management style</x:String>
            <x:String>work life balance</x:String>
            <x:String>benefits</x:String>
            <x:String>work environment</x:String>
            <x:String>training</x:String>
            <x:String>extra activities</x:String>
            <x:String>process orientation</x:String
            >
            <x:String>quality orientation</x:String
            >
        </x:Array>
    </Picker.ItemsSource>
</Picker>
```

```
private void CriteriumPicker_SelectedIndexChanged(object sender, EventArgs e)
{
    var picker = (Picker)sender;
    int selectedIndex = picker.SelectedIndex;

    if (selectedIndex != -1)
    {
        verdict.Criterium = (string)picker.ItemsSource[selectedIndex];

        isReason = true;
    }
}
```

Always, think "page" and "code behind file" as one thing.

We have one more thing to do. We must have a way to save all this information. "Feelings" and "Reasons" should be put together as one vote for one person. It must be consolidated with personal information such as first name, last name, company name and community name (2.0). Also, it must have time and date information. All this should be done with the click of a… button!

- *Add a button <Button></Button> element just above </Grid>.*
- *Write "Vote now!" for the text value.*
  This will add even more urgency to the action we request from the employees.
- *Name button as "BtnVerdict".*

- ➢ *Add a "click" event = "BtnVerdictClicked"/>*

There! We are done with the page. Let's proceed with the code behind file, shall we?

Step 2

Now that we are ready to go, let's complete the voting process which is half of our mobile app! See, having small goals help. To be able to do that we will need additional components that must be distributed to different layers of the application.

1) *(views) VotingPage.xaml = Display two options, prompt for a selection with a reason.*
2) *(views) VotingPage.xaml.cs = Code behind file of voting page. Consolidate data and forward it to page two.*

3) *(models) Verdict.cs = Properties, getters and setters relevant to daily votes.*
4) *(controllers) VotingController.cs = Container for overall business logic. Access service to save verdict.*
5) *(services) MilgramService.cs = Data access mechanism for API. Access API to save verdict.*

As you can see, Moody does not directly access our database (Berry). Instead, it does so via our API (Milgram). This choice makes our architecture N-tier, meaning we have more than three architectural layers.

1) *Presentation Layer = Pages and code behind files.*
2) *Business Layer = Controller(s) = There is only one in Moody.*
3) *Data Layer = Models.*
4) *Data Access Layer = Service(s) = There is only one in Moody.*
   (which accesses data via API and sends current info to relevant components via models).

If you like playing this game, you can go wild and add all kinds of new layers. Remember though, simplicity (!) is the best policy when it comes to software applications. A fellow programmer may need to continue your work. Solutions that can be understood by more than one person (!) are the best… which was the opposite in my generation. So, we will stop here, and our decision will stay the same for all future mobile apps for troublemakers. Also, pay attention. Your system architecture is a part of your overall strategy… which must empower you… to do your thing.

2.1

We start off with "verdict". There are two types of attributes that must be consolidated into one, "feeling" and "criterium". Also, that information must be passed through a series of layers until it reaches the database.

- *Retrieve "feeling" and "criterium" along with date & time info on voting page (VotingPage.xaml).*
- *Pass data to "controller" (and "analyses page") (ignore what happens in AnalysesPage.xaml for now).*
- ➢ *Now that "VotingController" grabbed "feeling and criterium"…*
- ➢ *…it will associate this information with fake "employee" information.*
  (First Name, Last Name, Company, Community).

If you are curious, here's a sample of an entry for "Verdict":

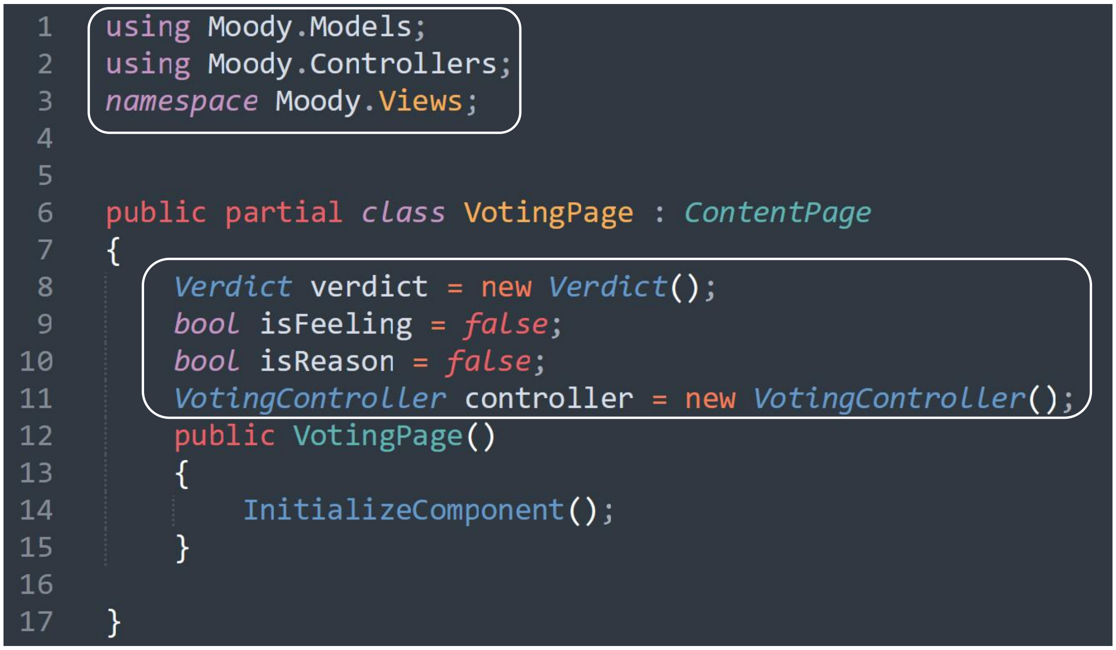
- *id (This will be handled by our database. Thank you, MongoDB!)*
- *"ACME"*
- *"Habib"*
- *"Habibovic"*
- *Technology (department) (as in "community")*
- *"Nay" (I'm mad as Hell!)*

- *“benefits” (reason) (as in “criterium”)*
- *Date & Time (which will be ignored in Moody 1.0)*

Among these properties, “community”, “feeling” and “criterium” are the most important. As you will see later, we will use them over and over to retrieve certain types of information. In the second version of Moody, we will also have “ranking” and “ecosystem” which will enable us to place our company in a pool of companies limited by the borders of our ecosystem. The default ecosystem is a “district”, but you can set it to “city” or “country” (2.0). Different settings lead to different kinds of experiments… meaning “you should think before you change settings”, pal.

2.2

If you are like me, you enjoy every step regardless of how small they might be. I mean, the end result is never what it is hyped for, believe me. In this business, taking steps is much more fun. Now that we have a vague understanding of what’s going to happen, let’s start the real second step which involves working on the code behind file of “VotingPage”.



```
using Moody.Models;
using Moody.Controllers;
namespace Moody.Views;


public partial class VotingPage : ContentPage
{
    Verdict verdict = new Verdict();
    bool isFeeling = false;
    bool isReason = false;
    VotingController controller = new VotingController();
    public VotingPage()
    {
        InitializeComponent();
    }

}
```

Reminder: “Verdict and Vote are the same, but we can ignore date-time info when dealing with verdicts”.

First, create two new folders, “Models” and “Controllers”. Then, create two classes in them, Models\“Verdict.cs” and Controllers\”VotingController.cs”. Don’t pay attention to their contents for now. We will go over them later. As you can see, we need two variables (isFeeling, isReason) and two objects (verdict, controller).

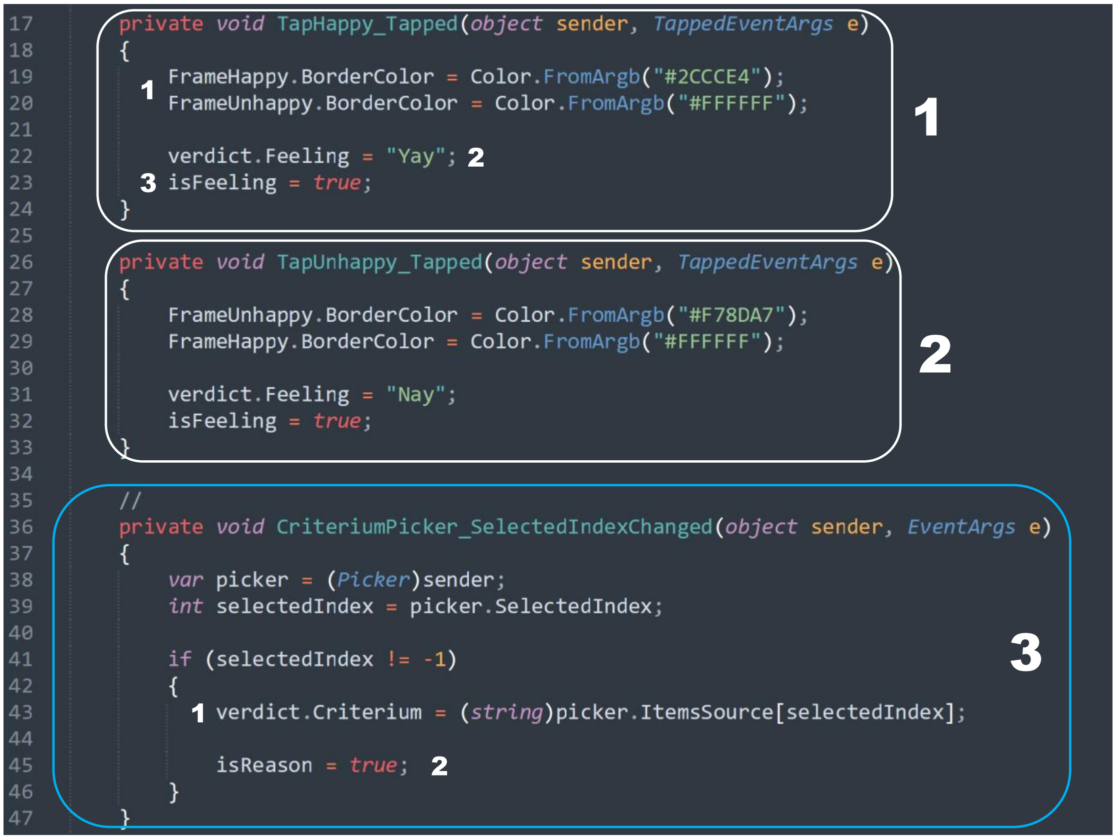
- *“verdict” will hold all the data we need.*
- *“isFeeling” and “isReason” are there to make sure both are selected. That verdict can be saved.*
- *“controller” will handle all the necessary business logic which isn’t much for our tiny app.*

You should already have placeholder functions for our events (2 x tap, 1 x pick, 1 x click) by now. If their names in the page and the code behind file are different, fix it. Make them the same. Otherwise, you can get “Bobby”, when you yell “Jimmy!”

First, let’s make sure the tap functions are doing what they should.

1) *Must make your choice visible when a feeling is selected or unselected.*
2) *Must assign feeling to appropriate verdict property.*
3) *Must set an appropriate flag to indicate that a feeling has been selected.*

Do the same for the other tap function with a twist.

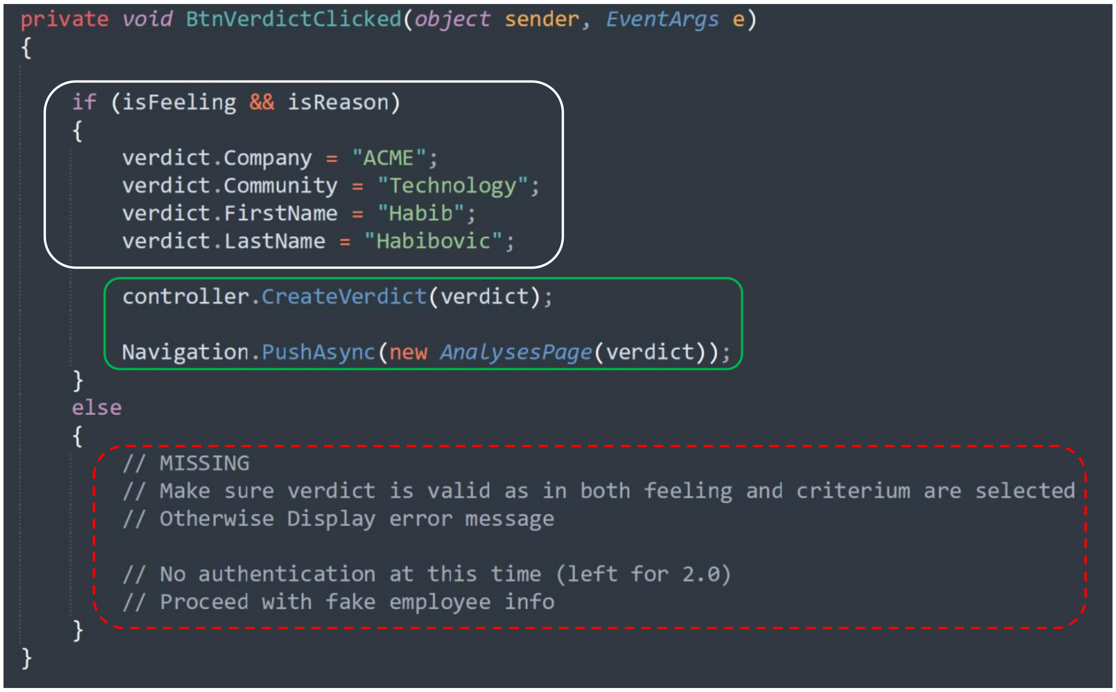


```
private void TapHappy_Tapped(object sender, TappedEventArgs e)
{
    FrameHappy.BorderColor = Color.FromArgb("#2CCCE4");
    FrameUnhappy.BorderColor = Color.FromArgb("#FFFFFF");

    verdict.Feeling = "Yay";
    isFeeling = true;
}

private void TapUnhappy_Tapped(object sender, TappedEventArgs e)
{
    FrameUnhappy.BorderColor = Color.FromArgb("#F78DA7");
    FrameHappy.BorderColor = Color.FromArgb("#FFFFFF");

    verdict.Feeling = "Nay";
    isFeeling = true;
}

//
private void CriteriumPicker_SelectedIndexChanged(object sender, EventArgs e)
{
    var picker = (Picker)sender;
    int selectedIndex = picker.SelectedIndex;

    if (selectedIndex != -1)
    {
        verdict.Criterium = (string)picker.ItemsSource[selectedIndex];

        isReason = true;
    }
}
```

Next, copy the source code above for the picker. (1) Specify the property for the assignment and (2) set a flag to ensure data integrity. As you might have guessed, when “isFeeling” and “isReason” are both “True”, an employee is allowed to submit her vote (2.0). While we don’t pay attention to those kinds of things in this version of Moody, it’s better to be prepared.

I intentionally left out that section of the code. Yes, it’s quite simple, but if I kept doing everything for you because it’s simple, this book would never end. You can do it on your own or you can wait for the second version of this book. It’s your choice. If this were a teacher-student kind of a situation, it would be your homework.

About the structure of the picker handler, don’t go crazy. Yes, sometimes memorizing a chunk of code helps to stay sane in computer sciences. I mean, who cares how as long as it works. That’s my motto! No, I wasn’t that way when I was younger, but I’m an old fart now. I don’t have a lot of time, man. Think about it this way, someone else made the decisions for you in this book (me). If you don’t like it, you will have to pick another development platform which is something I won’t do. Trust me, I have experimented with a bunch of them and… you know the ones I picked. When it *comes to picking and choosing, I'm not shy. For example, I like tablets so much more than computers or phones. Since none of them are perfect, I have all the popular choices. I use them all… and you know what, there are only minor differences between them. However, those minor differences can make or break things.*



```
private void BtnVerdictClicked(object sender, EventArgs e)
{
    if (isFeeling && isReason)
    {
        verdict.Company = "ACME";
        verdict.Community = "Technology";
        verdict.FirstName = "Habib";
        verdict.LastName = "Habibovic";

        controller.CreateVerdict(verdict);

        Navigation.PushAsync(new AnalysesPage(verdict));
    }
    else
    {
        // MISSING
        // Make sure verdict is valid as in both feeling and criterium are selected
        // Otherwise Display error message

        // No authentication at this time (left for 2.0)
        // Proceed with fake employee info
    }
}
```

This is the more important section of the code behind file. As you can see all the important tasks are carried out in the button click event handler. In the first part, we make sure "our employee can vote" meaning she tapped an emoji (feeling) and selected a reason (criterium) for her verdict (2.0). Since we won't have authentication in the first version of our app (or the first version of this book, for that matter), we will be using fake employee info (Company, Community, First Name, Last Name). Among those properties the most important one is "community", because we use it to get the right verdict and vote (2.0) information from the database along with "feeling" and "criterium".

Let's have a closer look at even more important parts (!) of that section. Our "controller" has a function called "CreateVerdict" which accesses the "data access layer" (MilgramService.cs). This service takes care of the actual writing or reading actions through another app running on the server, "Milgram". For the sake of simplicity, we will be running our API locally. When the time comes, you can migrate it to a remote server. All you will have to do is to update URL(s).

At the same time, we are directed to the second page where employees will be able to view the results of that day's poll as verdicts become available. Also, at the end of their shift they will know the final verdict of their community (Mood) and the whole company (Culture). "Mood" is the cumulative feeling of a community (i.e. Marketing Department). Culture is how well a company performs, manifested by their best and worst performing criteria. "Number one issue" is a criterium. "Greatest achievement" is another criterium.

Here's a possible view:

- *Tuesday's pool results are in!*

  "contrast" = "individual versus community" = Are you where you should be?
- ➢ *(everybody sees her own) Habib Habibovic from Marketing voted "Nay!" about "benefits".*

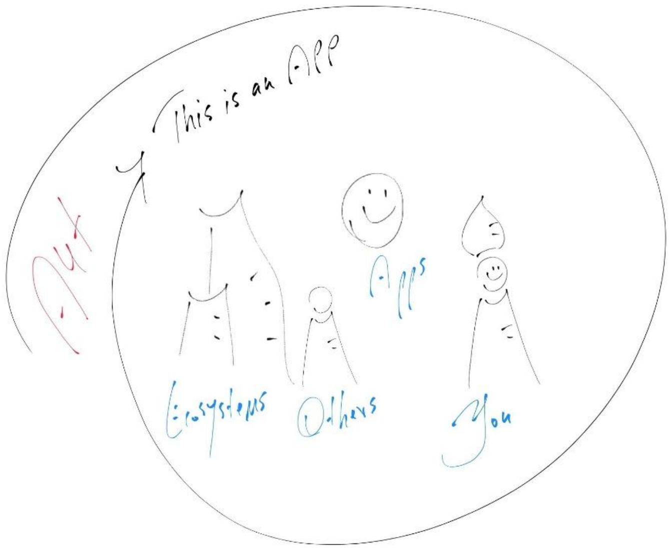
- *Marketing department voted "Yay!" about "benefits".*
  "contrast" = "lowest versus highest" = "How mature is your firm?"
- *ACME has a big problem concerning "work life balance".*
  Additional info may be attached such as "age" as in "how long has it been since the first occurrence", "persistence" as in "how many back-to-back days this time" ("optional").
- *ACME is very good when it comes to "extras".*
  The same explanation is applicable here.

At the end of every shift the entire verdict list will be copied to a new "Vote List" (2.0). This makes "Verdict Lists" lists of single days where date-time info is considered unimportant to make computations easier. In this version of Moody, we won't take advantage of date-time values of votes, but you can go ahead and do so. For example, you can contemplate the voting habits of the employees and the most popular times of voting or not voting… which seem trivial to me. Programmers often lose their way by inventing unnecessary details. Beyond your sweet spot there must be different apps not fat versions of the same apps. Do ore with less, but do it gracefully.

Welcome to yet another aspect of this adventure! Yes, there will be missing bits and pieces here and there. Some of them will be fixed in the next (and final) version of this book towards the end of 2026. Others will remain as homework for you. Curious about which one is which? Well, stick around and you will see.

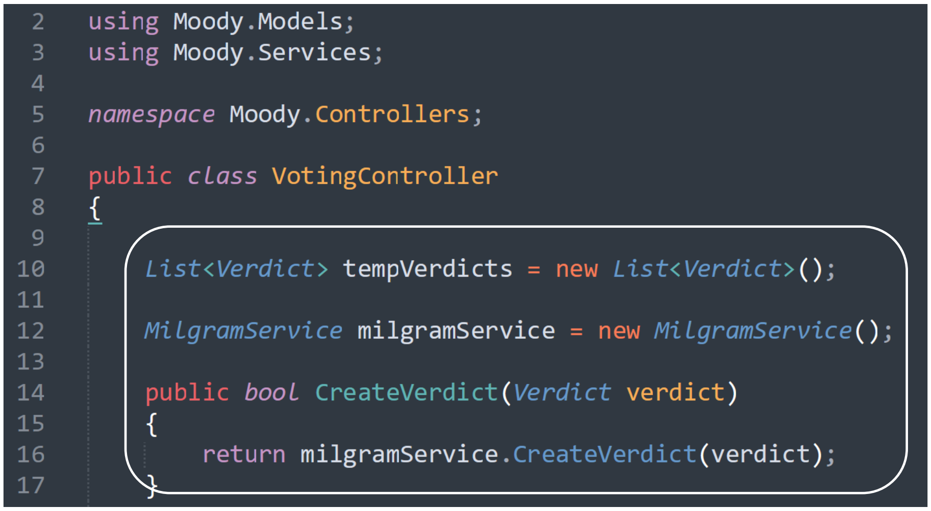
2.3

Excited yet? I know I am. This next bit will tickle your pickle. "VotingController.cs" is the heart of Moody. "Where is the mind", you say? We don't need one. Remember, we are here to conduct social experiments that will help "not so brave people" dump their emotional garbage onto others and drive "people who seem to care but don't" crazy. So, we are using an old concept of mine, using actors as components of the business layer of our app. Yes, you've got it. Cowardly employees will be the mind of our app, but this mind (!) will take them only so far. NEEHEEHEEHEEHAHAHA! We are developing a disjointed system. Ecosystems, people and apps are one thing in our realm. There, this is the summary of this book or more precisely what we, NoRM, stand for. "Your new manifestation" AKA "your app" is the sum of "you, your ecosystem, other people living there along with an appropriate combination of apps for troublemakers".



A4t = YOU + {Others + Apps + Ecosystems} = YOU 2.0

Since our functions are rather simple, having a layered architecture is like playing a word-of-mouth retransmission game where our method of retransmission doesn't distort (!) the information along the way. If you want to add more complex functionality in the future, "VotingController.cs" is where you should put it. "MilgramService.cs" is there just to save or retrieve data via our API, "Milgram". However, if you want to migrate complex functionality to the server, you should create additional controllers in your API and migrate your functions appropriately. No matter how you go about it, your final decision will be like any other architectural decision in our profession… debatable.

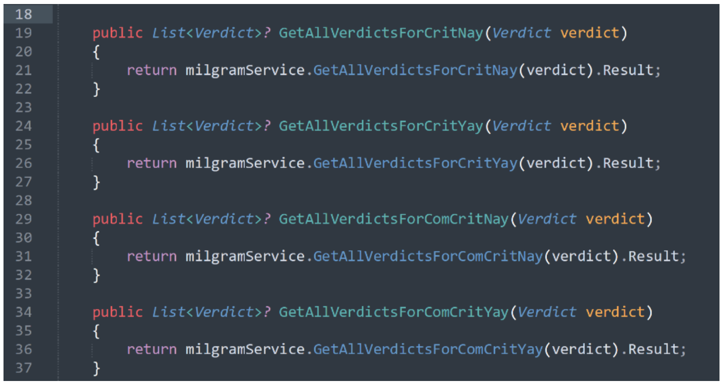


```
using Moody.Models;
using Moody.Services;

namespace Moody.Controllers;

public class VotingController
{

    List<Verdict> tempVerdicts = new List<Verdict>();

    MilgramService milgramService = new MilgramService();

    public bool CreateVerdict(Verdict verdict)
    {
        return milgramService.CreateVerdict(verdict);
    }
```

If we dissect this little "write" function (CreateVerdict), we will uncover:

- *A Boolean return value to determine whether we have successfully saved the verdict.*
- *An instance of "Verdict" that carries the data coming from the current actor's daily verdict.*
- *Also, a reference to the service function which is the real performer of the task at hand.*

There are things being done beyond the confines of our mobile app… taking place on the server. We will go over them when we focus on our API. We can ignore them for now.



```

    public List<Verdict>? GetAllVerdictsForCritNay(Verdict verdict)
    {
        return milgramService.GetAllVerdictsForCritNay(verdict).Result;
    }

    public List<Verdict>? GetAllVerdictsForCritYay(Verdict verdict)
    {
        return milgramService.GetAllVerdictsForCritYay(verdict).Result;
    }

    public List<Verdict>? GetAllVerdictsForComCritNay(Verdict verdict)
    {
        return milgramService.GetAllVerdictsForComCritNay(verdict).Result;
    }

    public List<Verdict>? GetAllVerdictsForComCritYay(Verdict verdict)
    {
        return milgramService.GetAllVerdictsForComCritYay(verdict).Result;
    }
```

Now that we are done with the verdict creation functionality, our controller will soon inhabit the rest of the data access functions that can trigger our service. You must be getting more and more aware that the real star in this word-of-mouth retransmission game is our API (Milgram). Strangely, we spend more time on our app, because it has more components. Pretty much like everything in life, right? Trivialities always take more time.

Here's the translation of the picture above:

- *"Crit" stands for "criterium".*
- *"Com" stands for "community". We never use "com" for company. Don't make assumptions. Seek descriptions.*
- *"Nay" stands for negative vote. "Yay" stands for positive vote. We don't care for alternate usages.*

Now that the summary is out of the way, let's explain these functions:

- "GetAllVerdictsForCritNay" retrieves all negative verdicts company-wide
  for the criterium employee has selected during the voting process.
- "GetAllVerdictsForCritYay" retrieves all positive verdicts company-wide
  for the criterium employee has selected during the voting process.
- "GetAllVerdictsForComCritNay" retrieves all negative verdicts
  for the criterium employee has selected coming from her community.
- "GetAllVerdictsForComCritYay" retrieves all positive verdicts
  for the criterium employee has selected coming from her community.

Why do we have so many (!) data retrieval functions? Because we need to understand three things, two of which concern the whole company. The third one has to do with the employee's community.

1) The criterium with the most "Nay" votes (Culture \ Number one issue).
2) The criterium with the most "Yay" votes (Culture \ Greatest achievement).
3) Community's cumulative feeling concerning the criterium that makes the employee happy or unhappy (Mood).

So, "feeling" has to do with employees, "mood" has to do with communities and "culture" has to do with companies. Pay attention, people. We are forming hierarchies which will provide exploitation points for our social experiments. Because while hierarchies make things work, no one really likes them except the ones at the top of the food chain.

- ➔ Feeling (exploit individuals) = Provoke unhappy people and make issues visible to bother those with power.
  - ○ Mood (exploit communities) = Make social conflicts visible by using contrasts (and form factions).
    - ▪ Culture (exploit ecosystems) = Make maturity of a company visible to start a rat race.

Are we having fun yet, people! NEEHEEHEEHEEHAHAHA!

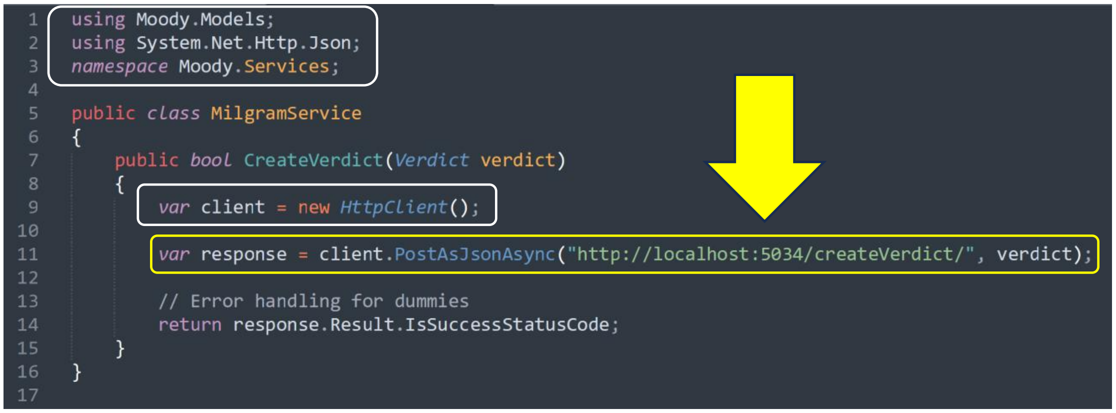
(Take Two) When we begin working on the second page of our app (Analyses Page), we will come back to our controller, and we will add the functionality to compute "Mood", "Number One Issue" and "Greatest Achievement". For now, we can ignore them.

2.4

Let's follow the trail of our only "write" function, "CreateVerdict". We should first focus on our Data Access Layer, "MilgramService." Before we do so, however, we'd better make sure you've got it right up until this point. Yes, practice makes perfect. So, here's once again… what we have done so far.

- Presentation layer displays data, provides ability to input data and triggers the app to behave in a certain way.
  (Pages and associated code behind files) = Views \ VotingPage.xaml, VotingPage.xaml.cs, etc.
- Business layer acts like traffic police, doesn't do anything other than direct things to their appropriate places.
  (Controllers) = Controllers \ VotingController.cs.
- Data Access layer handles all data regardless of whether it has to do with the client or the server side.
  (Services) = Services \ MilgramService.cs.
- Data layer is all about entity classes. Remember, nowadays we tend to use pseudo-classes which are runtime collaboration of tightly coupled objects. This idea was first introduced in the early 2000s in an attempt to equate entity classes and relational database tables in a 1-1 manner. While this has caused some problems then, now with the advances in computer power, it has become the norm rather than the exception. Anyway, we will obey (!) that rule mostly. When we won't, we will create additional folders for real (!) classes as in "Entities\Class1.cs".
  (Models) = Models \ Verdict.cs.

That being said, let's move, people! Here comes the service!

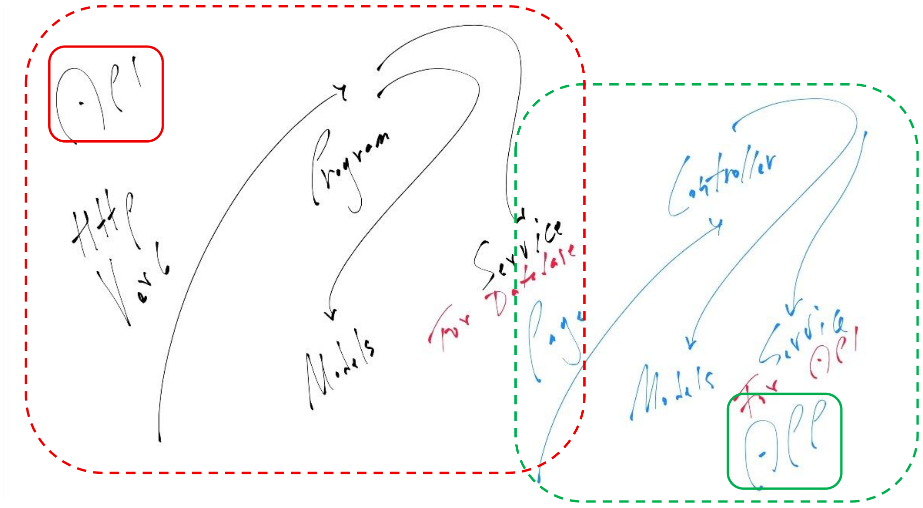


```csharp
using Moody.Models;
using System.Net.Http.Json;
namespace Moody.Services;

public class MilgramService
{
    public bool CreateVerdict(Verdict verdict)
    {
        var client = new HttpClient();

        var response = client.PostAsJsonAsync("http://localhost:5034/createVerdict/", verdict);

        // Error handling for dummies
        return response.Result.IsSuccessStatusCode;
    }
}
```

First, we must access our data (using Moody.Models) and we must have a means to access another app on a server running locally or somewhere else like AWS (server-side app, API, "Milgram").

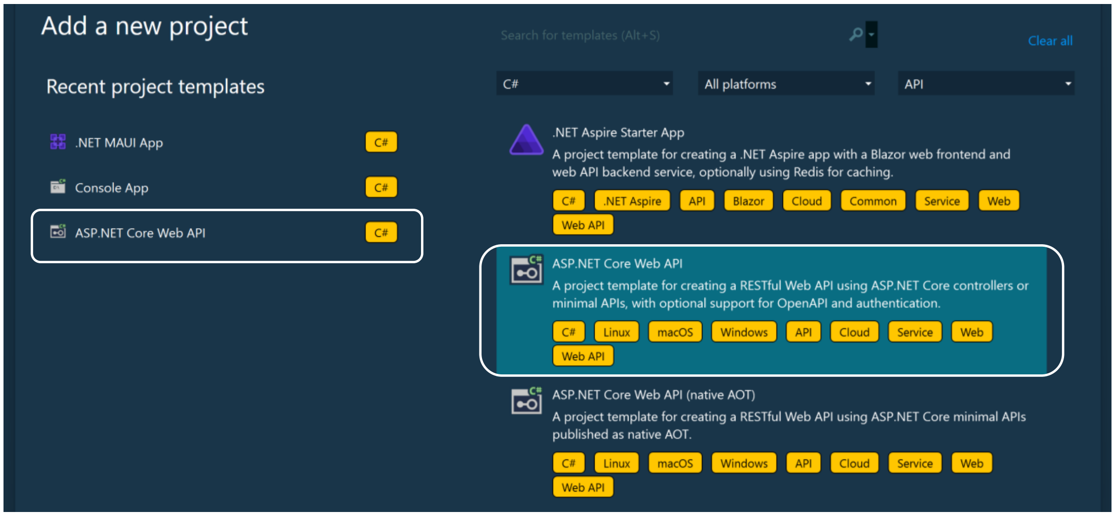
- *Create a new Http client and get rid of it after every service call to our API. We need to be secure, man.*
- *Add (write) the new verdict to the verdict collection on our database (Berry) with the help of our API (Milgram).*
- ➢ *response = client.PostAsJsonAsync(" Server URL with appropriate endpoint", object to be passed)*

You might be wondering, "What the heck is an endpoint?" Don't worry. We will cover them when we are going over the API. They sound weird, but it's a very simple concept which is very similar to tagging things. Let's skip the other service functions for now and complete the "voting process" first. Then, we will come back and wrap things up.



You have done it! Now, we will develop another app, our API. Don't be shy! Go out… and have fun. Then, come back here!

Step 3



Add a new project

Recent project templates

.NET MAUI App — C#
Console App — C#
ASP.NET Core Web API — C#

Search for templates (Alt+S) — Clear all

C# — All platforms — API

.NET Aspire Starter App
A project template for creating a .NET Aspire app with a Blazor web frontend and web API backend service, optionally using Redis for caching.
C# · .NET Aspire · API · Blazor · Cloud · Common · Service · Web · Web API

ASP.NET Core Web API
A project template for creating a RESTful Web API using ASP.NET Core controllers or minimal APIs, with optional support for OpenAPI and authentication.
C# · Linux · macOS · Windows · API · Cloud · Service · Web · Web API

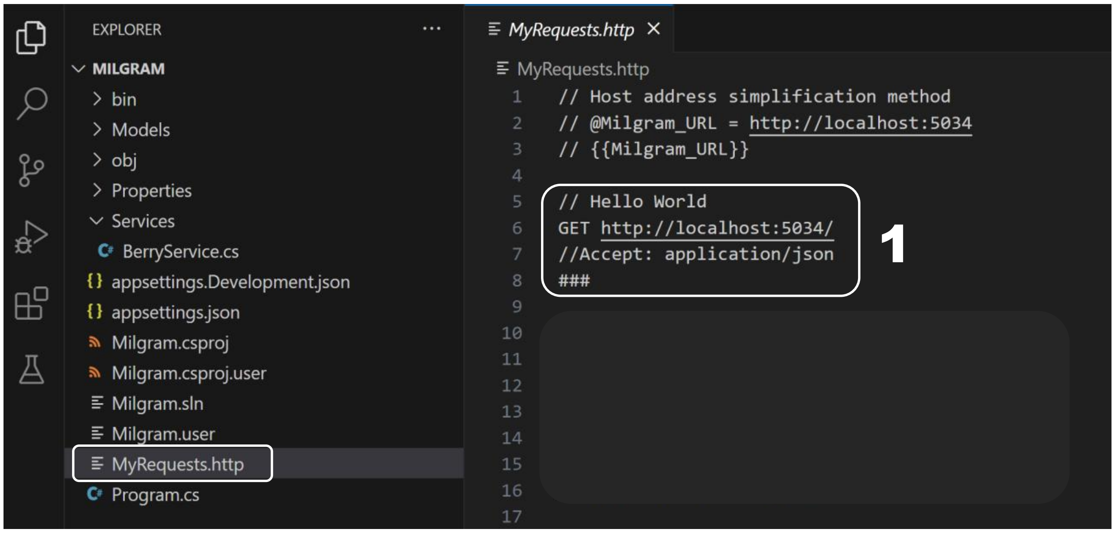
ASP.NET Core Web API (native AOT)
A project template for creating a RESTful Web API using ASP.NET Core minimal APIs published as native AOT.
C# · Linux · macOS · Windows · API · Cloud · Service · Web · Web API

Wow! App is dead, long live API! Soon, you will see why I like using two IDEs at the same time. I use Visual Studio Community Edition for my apps and Visual Studio Code for my APIs. Of course, you can use one for both, but why make it harder… why have less fun, right? I create the API in VSCE too, but when it's time to start the server and work on the app in run-time during testing frenzy, both of my IDEs are wide-open. Yes, it helps to have two monitors or a wide one.

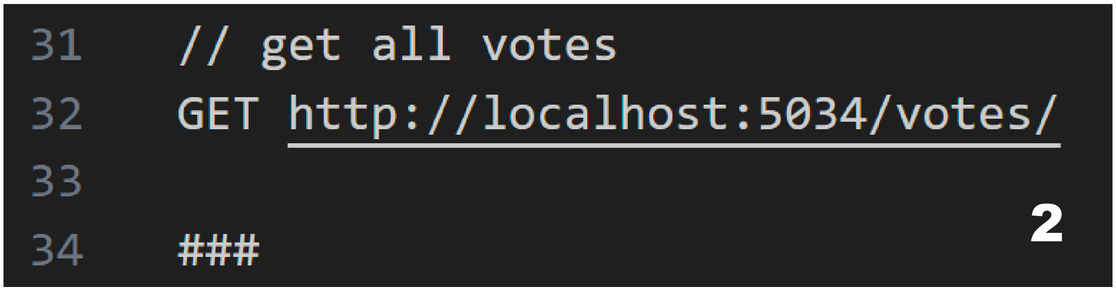
Maybe you are already aware of it. We will have several projects in one solution. We have been working on the first project from the start. Now right click your solution and add another, an "ASP.NET Core Web API" project, to be exact. Name it "Milgram" after Stanley Milgram, who is the inspiration behind the Apps for Troublemakers.

Stanley Milgram was an experimental psychologist. I always considered him a kindred spirit. He had a relentless thirst for uncovering the truth, but he also made his experiments fun. That's what I'd like to achieve too. Workplaces, schools, homes… don't have to be boring. They can be fun even in the most disturbing circumstances. Humor is a powerful fuel to get your little engine going. Truth is shy. She never comes out after your first try. You need time. You need to be a survivor.

After you created your API project, the first thing you should do is to create a file that has all your endpoint definitions in it. The IDE below looks different, because I decided to use Visual Studio Code this time. This way, you will see the differences between VS Community and VS Code. VS Code is prettier, but VS Community has more functionality, and it is easier to use. VS Code is so popular, even those who hate Microsoft love it. Me? I don't hate Microsoft. I grew up there!



```
// Host address simplification method
// @Milgram_URL = http://localhost:5034
// {{Milgram_URL}}

// Hello World
GET http://localhost:5034/
//Accept: application/json
###
```

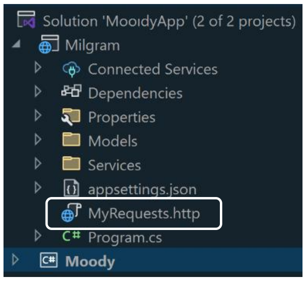


```
// get all votes
GET http://localhost:5034/votes/

###
```

Don't get crazy, now. Start small. The first endpoint is the one that lets you know that everything is working as intended (Hello World). The next one should be the one that retrieves all documents in your MongoDB collection without any filters (get all votes). This way, you will know that your database is up and running… meaning if you can't get the filtered lists you seek later, it's all your fault!

Once these introductory measures are taken, you can be bold. It's time for your first "post" endpoint. "What do you mean post", you say, huh? Well, here's a translation for you, my friend.

- *These are HTTP verbs that can be executed to make your API act in a certain way (get, post, update, delete).*
- *GET is used to get (!) data.*
- *POST is used to save data.*
- *UPDATE is used to update existing data.*
- *DELETE is used to delete data.*



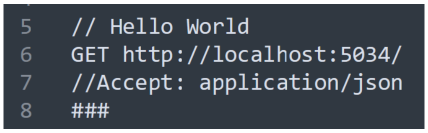
You may be tempted to think that all actions will need their very own endpoints. While this may be the preferred way for readability / understandability, you don't have to do that. You can go ahead and use the same endpoint, granted you make small differences by using different Http verbs. We won't do that. We are followers of the KISS approach, "Keep it simple and stupid".

Here's a contrast. This picture is from VS Community. As you can see, the locations of the files are always the same. Both IDEs have exactly the same project structure. The only difference is how the components of a project are displayed.

The first endpoint is simply our local host with a specific port number. So, when the server (read, API or "Milgram") is running, if you just write down this in the address bar of your browser, you should get a welcome message. Many like to use "Hello World!" as the welcome message. I usually don't. Here, however, I went with public opinion.

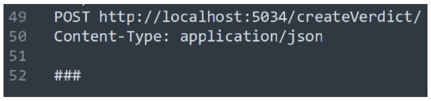
For the retrieval of all verdicts, we will use "votes" endpoint for there are no distinctions between verdicts and votes in this version of Moody. Also, it will be a good change management homework (!) for the second version of Moody. We will use "createVerdict" for saving a single verdict. As you might have guessed, your "local host URL" is your "base URL". If you want, you can encapsulate it in a constant which can be maintained easily later. Again, we will skip these niceties for now. We must see where this is all going fast. Later, we can come back and make it prettier and more functional. I'm more of an arc kind of guy than an episode one. I suggest you to be the same, for it's the only thing that differs kings from the fools. Believe me, I have been there. So, let's dissect these verbs, shall we?

This is the verb executed when the server is running without any problems. This is the base URL. "###" is the terminator.



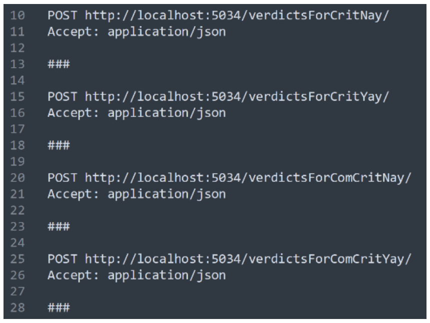
```
// Hello World
GET http://localhost:5034/
//Accept: application/json
###
```

This verb creates a new verdict. It uses "application/json" content type like all the others.



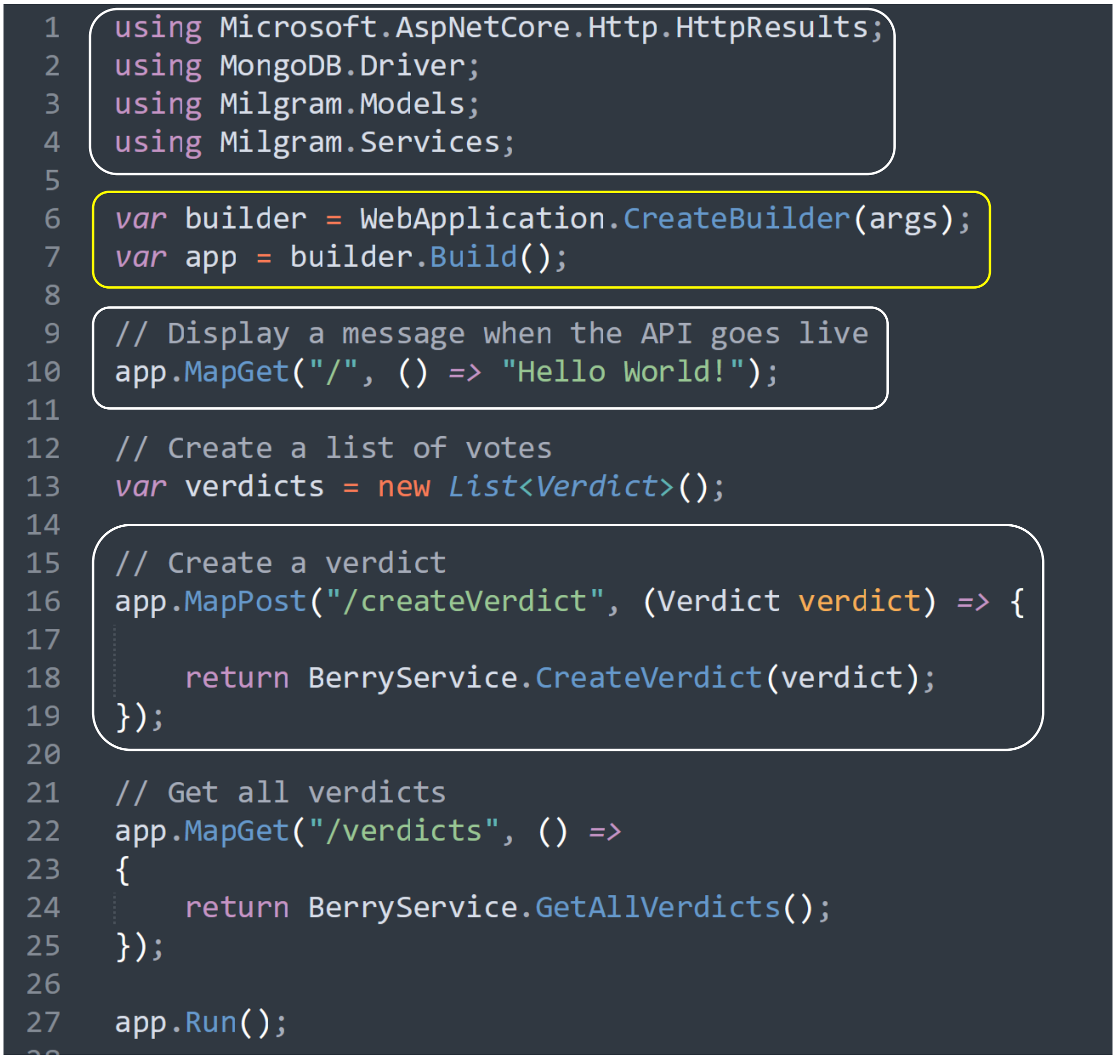
```
POST http://localhost:5034/createVerdict/
Content-Type: application/json

###
```

These verbs retrieve verdicts according to certain filters, "crit" (criterium), "com" (community), "yay" (positive verdict), and "nay" (negative verdict).



```
POST http://localhost:5034/verdictsForCritNay/
Accept: application/json

###

POST http://localhost:5034/verdictsForCritYay/
Accept: application/json

###

POST http://localhost:5034/verdictsForComCritNay/
Accept: application/json

###

POST http://localhost:5034/verdictsForComCritYay/
Accept: application/json

###
```

These short statements may look like magic to you. "How do they do it?" you may say. Well, actually, they don't do anything… anything at all. These are just for defining 'where' the action will take place. Now that we know our endpoints, it's time for some real work. When endpoints are triggered, whatever Http verb they are associated with is executed in a file called "Program.cs" which is automatically created when you create a .NET Core API project. It's pretty much like a "controller". Of course, it doesn't know what to do about your problems and there are a bunch of things you must delete in the automatically created file, but you can't have everything, right? So, let's tell Milgram's "Program.cs" what to do.

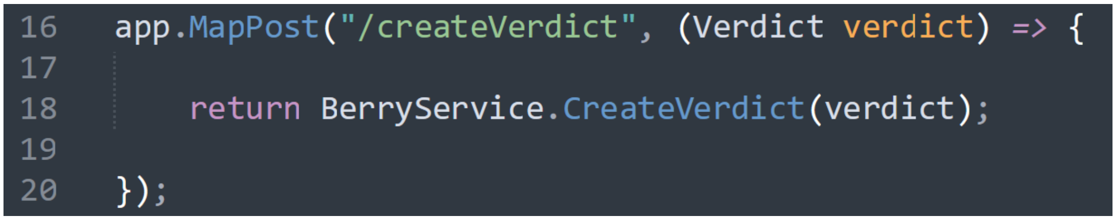


```
using Microsoft.AspNetCore.Http.HttpResults;
using MongoDB.Driver;
using Milgram.Models;
using Milgram.Services;

var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();

// Display a message when the API goes live
app.MapGet("/", () => "Hello World!");

// Create a list of votes
var verdicts = new List<Verdict>();

// Create a verdict
app.MapPost("/createVerdict", (Verdict verdict) => {

    return BerryService.CreateVerdict(verdict);
});

// Get all verdicts
app.MapGet("/verdicts", () =>
{
    return BerryService.GetAllVerdicts();
});

app.Run();
```

This simple structure is called Minimal Hosting Architecture. What it means is you can ignore the underlying complexity and focus on a few keywords for the lack of a better term. It is stupefyingly simple. It has four never-changing elements and a few more elements for extra actions, depending on what you want to do with your server app.

- *You need a builder that builds your API (like a "cookie press").*
- *You need an app that runs on your builder (like a "cookie").*
- *app.MapGet("/", () => "Hello World!") is there make sure your server is running successfully.*
  You should have this even if you have no Http verbs running around. Yes, I hate the expression "Hello World" too. So, I often pick something that has to do with the app. Here we can use "Are we emotionally agile, people!"
- *app.Run is the command that runs your server, so that you can process Http verbs.*

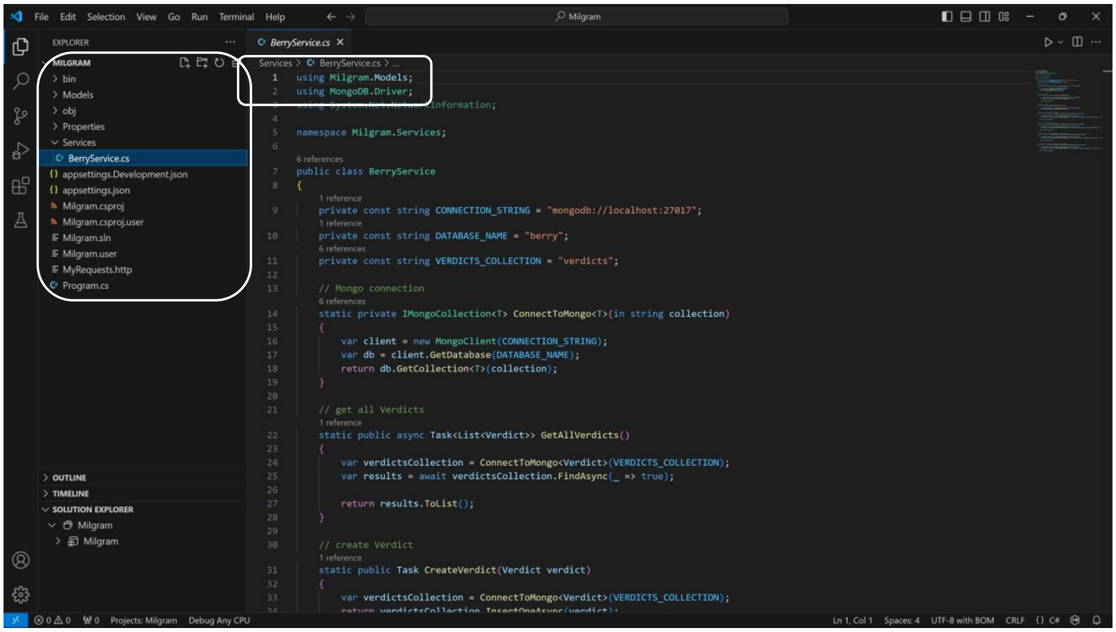
Of course, what really processes those Http verbs is our service (communication) layer, this time on the server-side! Yes, we keep repeating things, but this is exactly what makes the steps of mobile app development easy to understand.



```
app.MapPost("/createVerdict", (Verdict verdict) => {

    return BerryService.CreateVerdict(verdict);

});
```

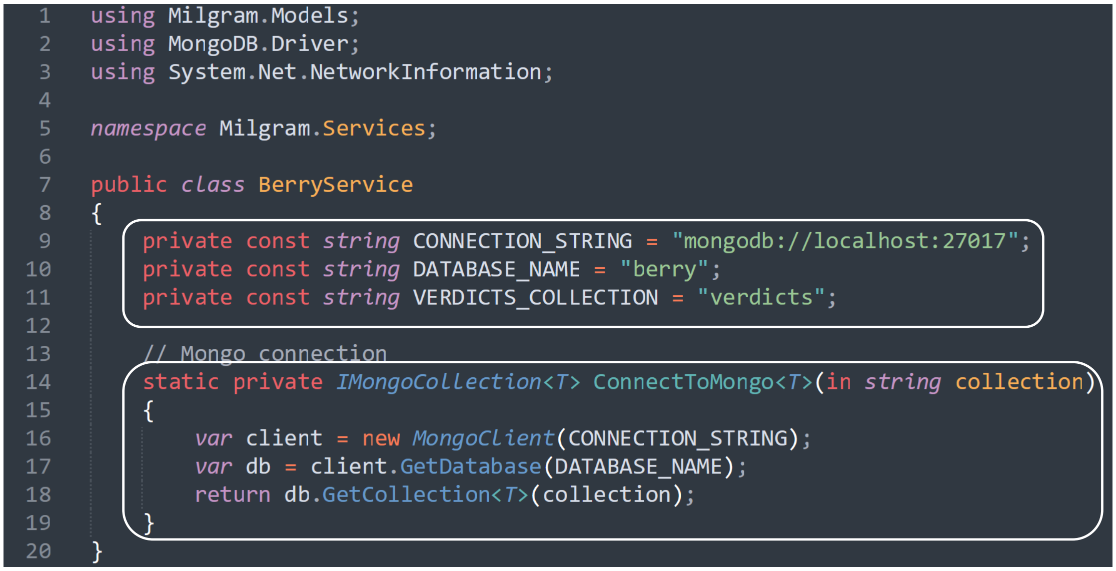
- *app.MapPost is the " Http verb meaning "POST". We give it a trigger, URL"/createVerdict" and data, "verdict".*
- *"CreateVerdict" method of "BerryService" is called with "verdict" data.*
- *Method returns "success"or "failure" information.*

So, where is "BerryService?" Go ahead, create two folders in your API project, "Models" and "Services". Models will have the same classes you have in your .NET MAUI project (Moody)… well, with a minor looking important difference, MongoDB support. Again, if you are like me, you would use Visual Studio Code just for viewing and handling server operations. So, go back to Visual Studio Community Edition and open the Moody App solution. Right click Milgram and install "MongoDB.Driver" NuGet. Yes, I love threesomes. Don't judge me, man.



```
using Milgram.Models;
using MongoDB.Driver;
using System.Net.NetworkInformation;

namespace Milgram.Services;

6 references
public class BerryService
{
    1 reference
    private const string CONNECTION_STRING = "mongodb://localhost:27017";
    1 reference
    private const string DATABASE_NAME = "berry";
    6 references
    private const string VERDICTS_COLLECTION = "verdicts";

    // Mongo connection
    6 references
    static private IMongoCollection<T> ConnectToMongo<T>(in string collection)
    {
        var client = new MongoClient(CONNECTION_STRING);
        var db = client.GetDatabase(DATABASE_NAME);
        return db.GetCollection<T>(collection);
    }

    // get all Verdicts
    1 reference
    static public async Task<List<Verdict>> GetAllVerdicts()
    {
        var verdictsCollection = ConnectToMongo<Verdict>(VERDICTS_COLLECTION);
        var results = await verdictsCollection.FindAsync(_ => true);

        return results.ToList();
    }

    // create Verdict
    1 reference
    static public Task CreateVerdict(Verdict verdict)
    {
        var verdictsCollection = ConnectToMongo<Verdict>(VERDICTS_COLLECTION);
        return verdictsCollection.InsertOneAsync(verdict);
```

Yes! We are finally getting somewhere. We are where the tasks at hand are actually being handled. Up until this moment we have been dealing with calling the next in line and handing him (?) information. Let's focus on our service on the server side, shall we? "BerryService", here we come!

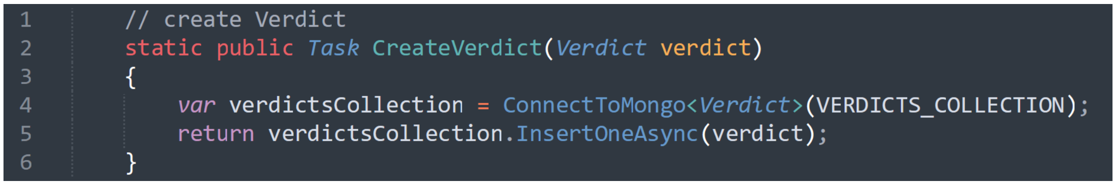


```csharp
using Milgram.Models;
using MongoDB.Driver;
using System.Net.NetworkInformation;

namespace Milgram.Services;

public class BerryService
{
    private const string CONNECTION_STRING = "mongodb://localhost:27017";
    private const string DATABASE_NAME = "berry";
    private const string VERDICTS_COLLECTION = "verdicts";

    // Mongo connection
    static private IMongoCollection<T> ConnectToMongo<T>(in string collection)
    {
        var client = new MongoClient(CONNECTION_STRING);
        var db = client.GetDatabase(DATABASE_NAME);
        return db.GetCollection<T>(collection);
    }
}
```

The first section is for our constants. While we can have a class somewhere (Milgram\Migration\Constants?), I just added them here for simplicity. Once you start delegating responsibilities, you can easily run amok and split yourself into atoms. The nice thing about NoSQL databases is they are like code. So, they don't need predefined structures. You can structure them when you need them to be structured. Even then, you can still change their structures on the go. In a minute you will see that we will be acting as if there is a database, because it doesn't even have to exist. If there isn't a database when we attempt to create a new entry, MongoDB will kindly create it for us. Thank you, Mongo!

Second section is a summary of computer sciences, "if you must do it, do it only once". 😊 *This is the connection procedure which will be used by all our API functions over and over again. The trick is, you should connect and disconnect each time meaning "don't leave any door wide-open for prying eyes".*

Now that we are past introductions, let's do some real work. Let's save a verdict after leaving all those intermediaries behind. This is the line of code that actually does the deed, people! I know, it's very short. That's the lesson to be learned here. When you want to enforce an N-tier system architecture, even one as simple as ours, most of the code will be about the architecture. Wow, just imagine the missing bits! In the second version of Moody there will be functionality for member management, rules engine and shadow use cases (hidden features doing interesting things in the background). Still, they won't add too much to the overall complexity. We will be using simple solutions as usual.

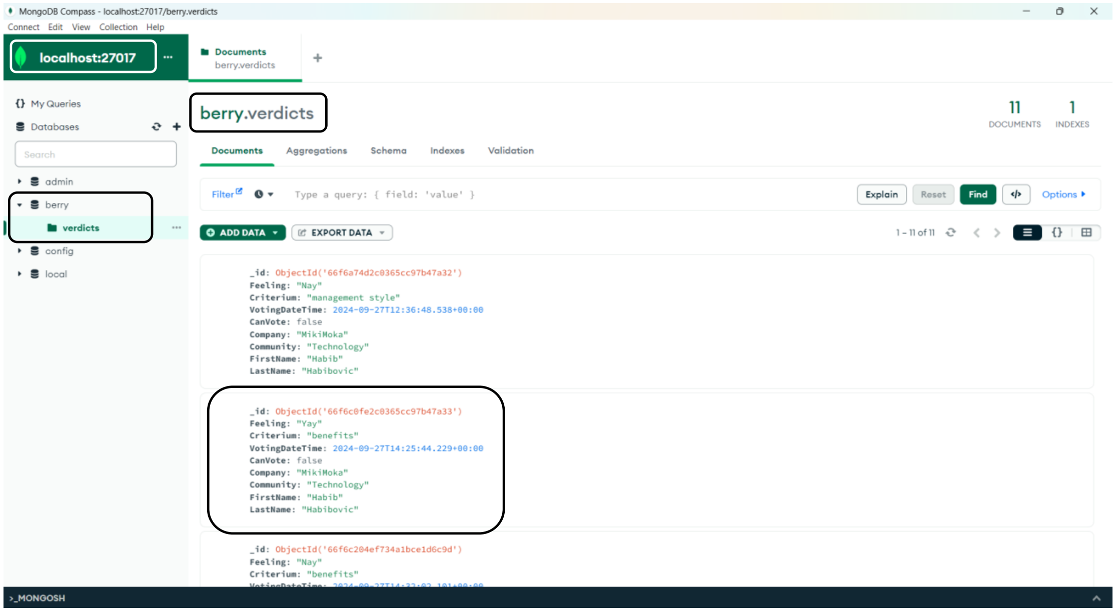


```csharp
    // create Verdict
    static public Task CreateVerdict(Verdict verdict)
    {
        var verdictsCollection = ConnectToMongo<Verdict>(VERDICTS_COLLECTION);
        return verdictsCollection.InsertOneAsync(verdict);
    }
```

Here's the summary:

1) *Get verdict data when triggered.*
2) *Connect to the MongoDB (Berry)*
   = If database doesn't exist, create it = If verdicts collection doesn't exist, create it.
3) *Add another "verdict" to the "verdicts collection" and return a success or failure message.*

The structure of our database is very simple at this point. There is only one collection, “verdicts” and there is only type of document, “verdict”. So, there will be a lot of verdict(s) in verdicts, get it?

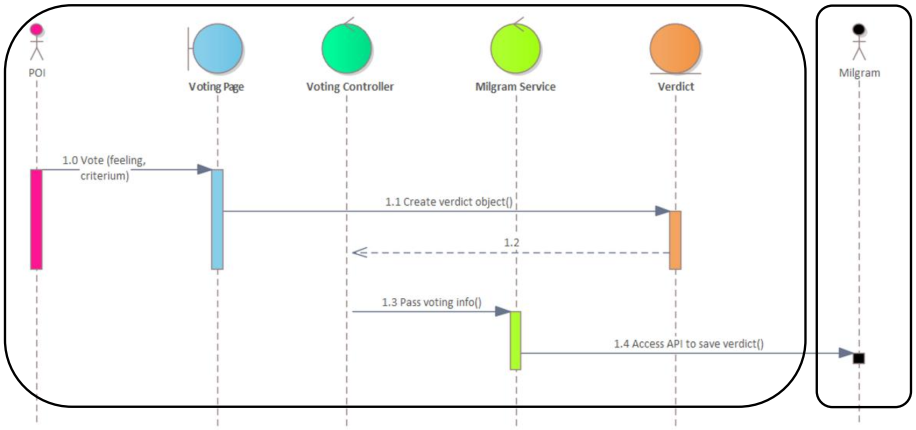

MongoDB Compass - localhost:27017/berry.verdicts
Connect Edit View Collection Help
localhost:27017
Documents
berry.verdicts
My Queries
Databases
admin
berry
verdicts
config
local
berry.verdicts
11 DOCUMENTS
1 INDEXES
Documents Aggregations Schema Indexes Validation
Filter
Type a query: { field: 'value' }
Explain Reset Find Options
ADD DATA
EXPORT DATA
1 – 11 of 11
>_MONGOSH

```
_id: ObjectId('66f6a74d2c0365cc97b47a32')
Feeling: "Nay"
Criterium: "management style"
VotingDateTime: 2024-09-27T12:36:48.538+00:00
CanVote: false
Company: "MikiMoka"
Community: "Technology"
FirstName: "Habib"
LastName: "Habibovic"

_id: ObjectId('66f6c0fe2c0365cc97b47a33')
Feeling: "Yay"
Criterium: "benefits"
VotingDateTime: 2024-09-27T14:25:44.229+00:00
CanVote: false
Company: "MikiMoka"
Community: "Technology"
FirstName: "Habib"
LastName: "Habibovic"

_id: ObjectId('66f6c204ef734a1bce1d6c9d')
Feeling: "Nay"
Criterium: "benefits"
```

In this entry, “Habib Habibovic from MikiMoka working in Technology Department, casts a positive vote about benefits.” Also, there are other fields like “canVote” and “VotingDateTime”, both of which will be ignored in Moody 1.0.

Stop for a Minute!

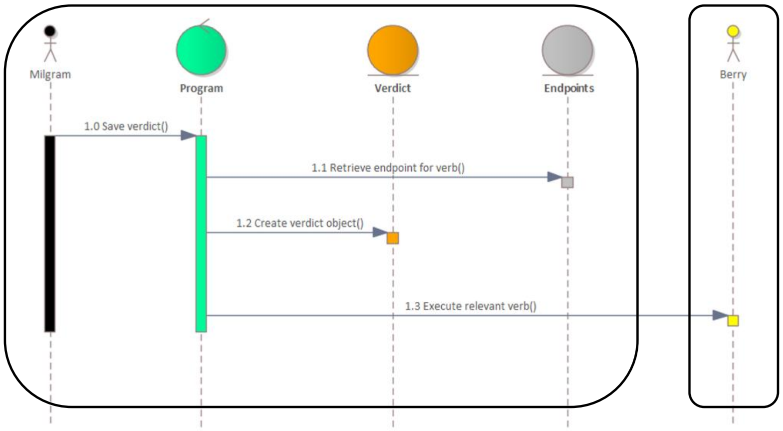
Let's have a look at what we have done so far in a big picture perspective. This is Moody layering for the "Broadcast Feeling" use case. The rectangle on the left is "Moody". The rectangle on the right is our API, "Milgram".


POI
Voting Page
Voting Controller
Milgram Service
Verdict
Milgram
1.0 Vote (feeling, criterium)
1.1 Create verdict object()
1.2
1.3 Pass voting info()
1.4 Access API to save verdict()

Here's the summary:

1) *Employee (as in our point of interest) casts a vote (feeling, criterium)(company, community, name)(date, time).*
2) *Voting page's code behind file passes the info to the controller with a verdict instance (VotingController).*
3) *Controller doesn't have too much this time. It just passes the info to the data access layer (MilgramService).*
4) *Service finalizes the sequence of events by passing the info to the API (Milgram).*

This is Milgram layering for the "Broadcast Feeling" use case. The rectangle on the left is "Milgram". The rectangle on the right is our database "Berry".


Milgram
Program
Verdict
Endpoints
Berry
1.0 Save verdict()
1.1 Retrieve endpoint for verb()
1.2 Create verdict object()
1.3 Execute relevant verb()

Here’s the summary:

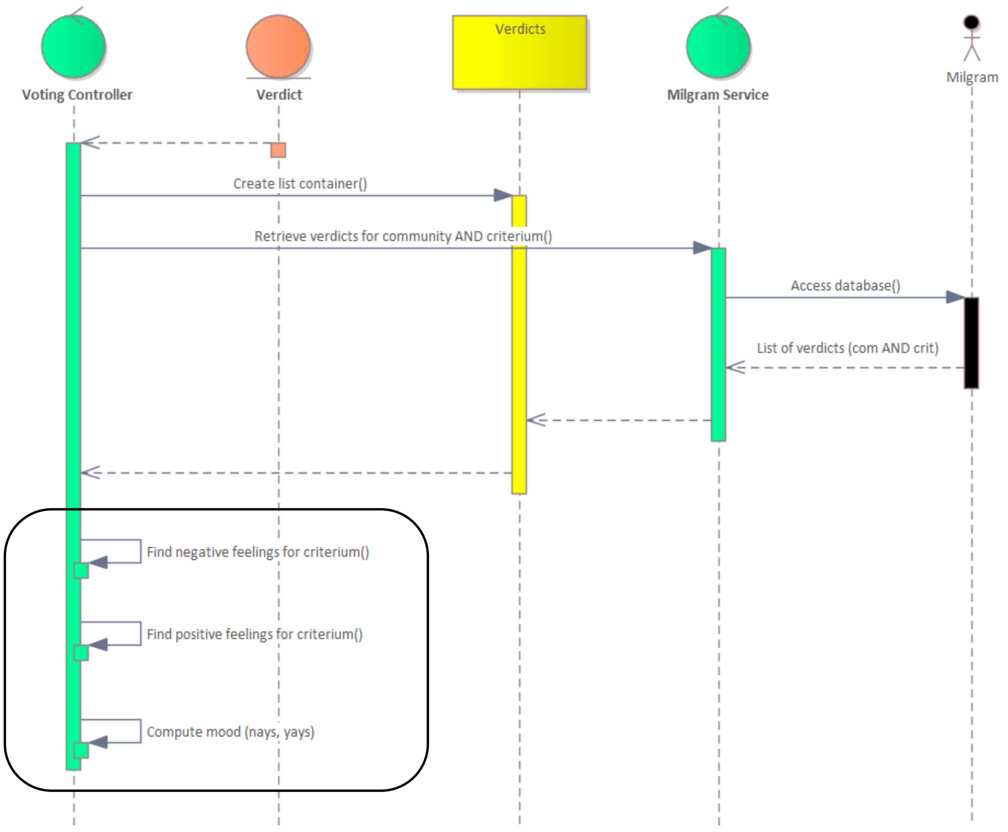
(By now, you should have realized that anything that looks like an actor, granted it is a software system, can be seen as a series of collaborations between objects if it is your current focus.)

1) *Being triggered by MilgramService our API executes the procedure for saving a single verdict to the database.*
2) *Program first gets the address for the relevant endpoint.*
3) *Then, it creates a verdict object.*
4) *Finally, it saves the data to the database by executing the relevant Http verb (POST).*

Into the Storm

You can relax. You have expressed your feelings. You made an impression. Now it's time to wait and see how the others are feeling. Are you in the right place with the right people? Do you think the way you feel reflects the reality of your ecosystem? Can you differentiate between what's fake and what's real? We shall see very soon… before your shift is over, in fact.

We will do it all over again… with a twist. This time we won't create an entry in the database. We will retrieve what is already there (verdicts of that day) and do a few simple computations to uncover the feelings of your community.

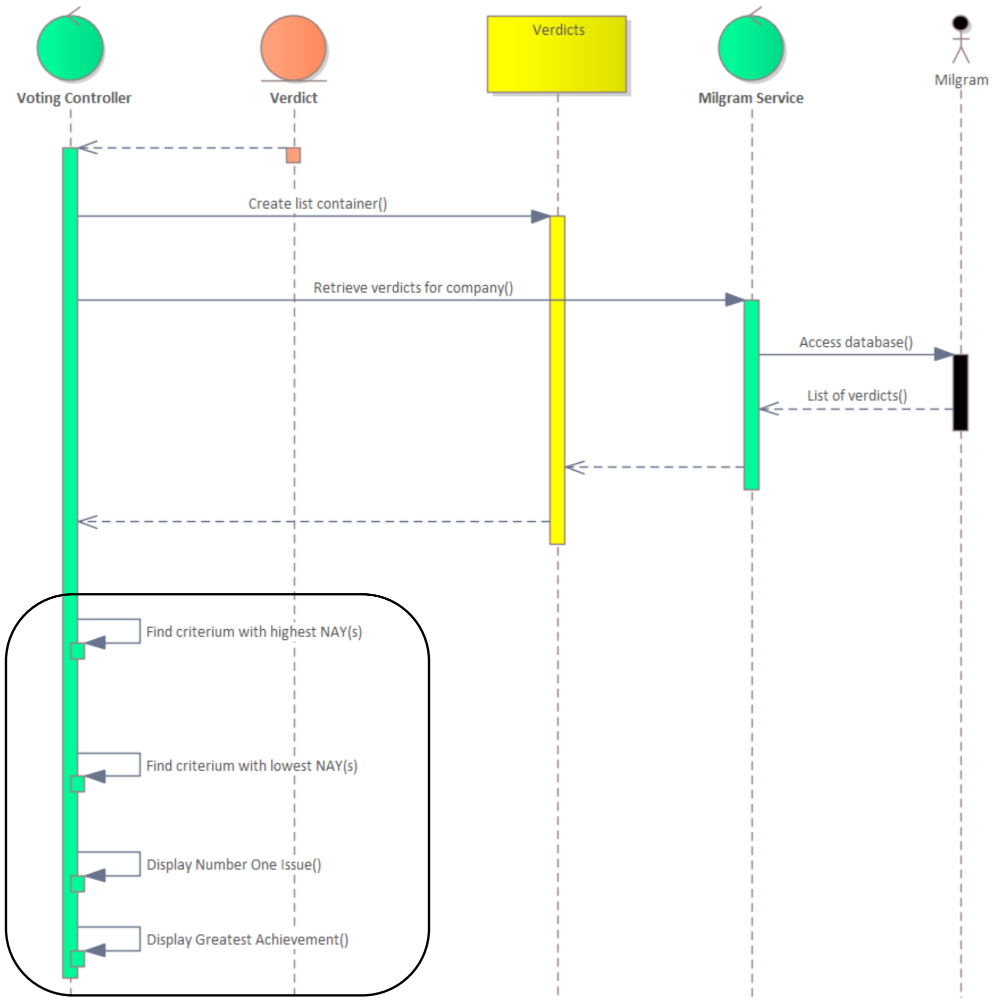

Voting Controller
Verdict
Verdicts
Milgram Service
Milgram
Create list container()
Retrieve verdicts for community AND criterium()
Access database()
List of verdicts (com AND crit)
Find negative feelings for criterium()
Find positive feelings for criterium()
Compute mood (nays, yays)

"This is how we are going to compute the collective feeling of your community".

The procedure is simple:

- *Retrieve all verdicts coming from employee's community which have to do with the criterium she has selected.*
- *Find the number of negative feelings towards the criterium.*
- *Find the number of positive feelings towards the criterium.*
- *Pick the bigger number and assign negativity or positivity as community mood.*

While the cumulative feeling of your community is displayed to provide a contrast between you and them, the more important metrics have to do with your company. Here we will retrieve all verdicts to compute those metrics.


Voting Controller
Verdict
Verdicts
Milgram Service
Milgram
Create list container()
Retrieve verdicts for company()
Access database()
List of verdicts()
Find criterium with highest NAY(s)
Find criterium with lowest NAY(s)
Display Number One Issue()
Display Greatest Achievement()

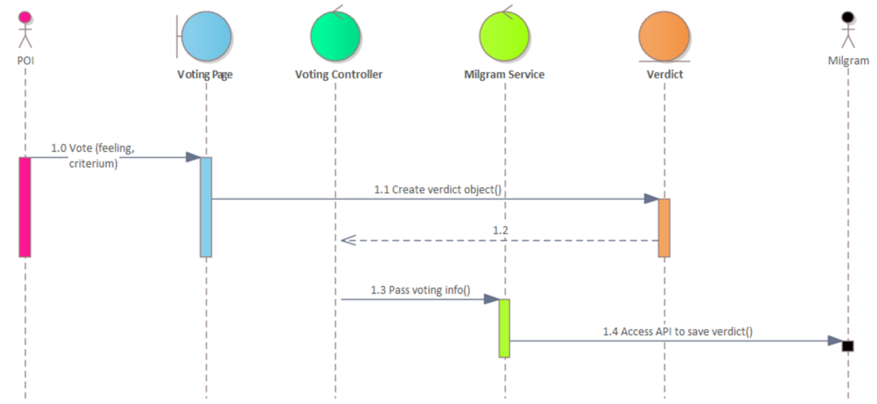
"This is how we are going to find your company's worst and best".

This makes the procedure a tad more complicated than the previous one:

- *Get all **verdicts** (We will get all **votes** -verdicts with date-time info- in Moody 2.0).*
- *Filter verdict list for each criterium (meaning "do it for twelve times").*
  When you do so, compute the number of negative and positive verdicts concerning that criterium.
  Make a record of criterium, the number of positive and negative verdicts by creating a temporary culture list.
- *Sort the temporary list according to the number of negative votes in descending order.*
- *Pick the first one in the list as the number one issue. Ignore equivalence*.*
- *Sort the temporary list according to the number of positive votes in descending order.*
- *Pick the first one in the list as the greatest achievement. Ignore equivalence*.*

**Do we care whether there are more than one "number one issue" or "greatest achievement?" Noo!*

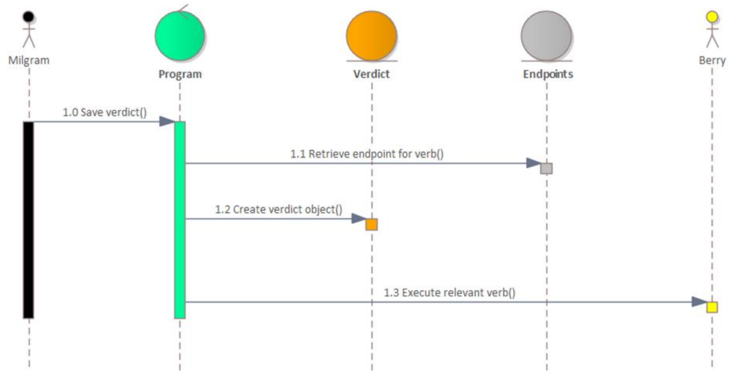
What? You need a higher level view, you say? OK. I'm here to help. Here's "Broadcast Feeling" and "View Results" use cases in contrast to each other. They look like close relatives, don't they?


POI
Voting Page
Voting Controller
Milgram Service
Verdict
Milgram
1.0 Vote (feeling, criterium)
1.1 Create verdict object()
1.2
1.3 Pass voting info()
1.4 Access API to save verdict()

Moody Layering = "Broadcast Feeling"

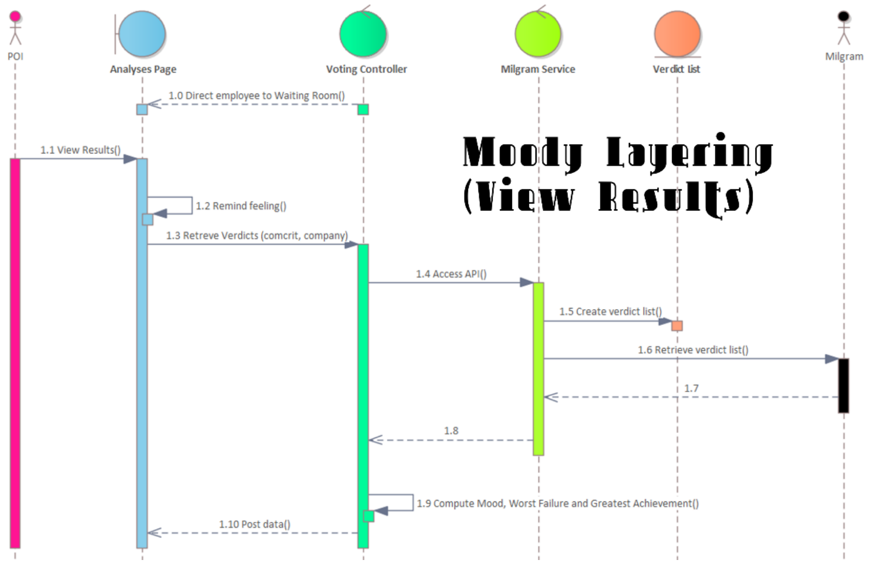
For simplicity, code behind file is not shown here. I often show methods in code behind files as methods of pages in sequence diagrams, because code behind files shouldn't do much and it's not wise to separate them at this level of abstraction.

- *Interaction summary:* ***Page <-> Controller<-> Service <-> API*** *where entities (models) are all over the place.*
- *Some people have a very strict policy towards entities. Others use them everywhere. I have a mellow policy where most entities are handled by controllers, but a few are left on their own here and there just to make things simpler, granted they don't cause security issues.*

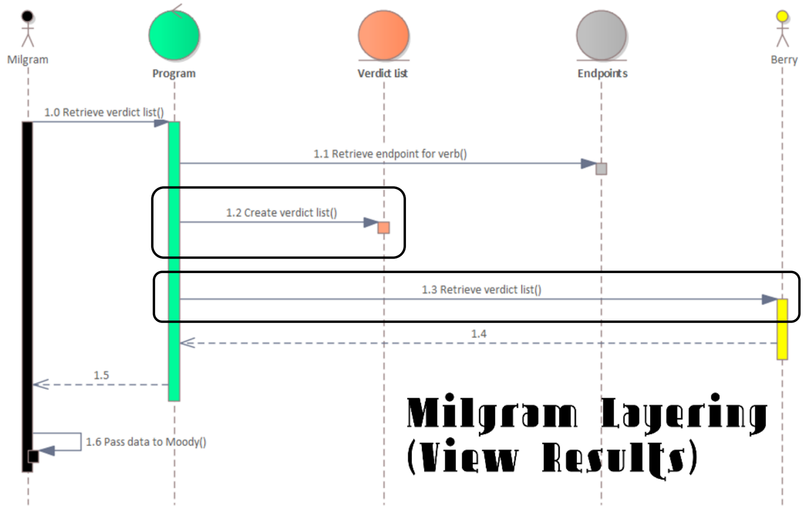

Milgram
Program
Verdict
Endpoints
Berry
1.0 Save verdict()
1.1 Retrieve endpoint for verb()
1.2 Create verdict object()
1.3 Execute relevant verb()

Milgram Layering = "Broadcast Feeling"

APIs are there to send or receive data. While some people love to migrate computations over to the server-side, I don't. Perhaps, that's because I often go cheap when the time to rent a server comes. Devices are, on the other hand, often inevitably quite pricey. They usually have better RAM and processing power. So, why shouldn't I handle my simple computations there? Of course, if something goes wrong at that end, it is forever lost. I don't think it's a problem though. That gal/guy can do it once more! Also, apps for troublemakers aren't meant to be scaled to millions of actors. As a result, I don't see a point. You don't have to follow my logic, of course. Migrating computations over to the server (API) is quite easy. Just let the functions of the server-side do the job, not just send or receive data. Following the same procedure, you can migrate your functionality to any layer in your system's architecture.


POI
Analyses Page
Voting Controller
Milgram Service
Verdict list
Milgram
1.0 Direct employee to Waiting Room()
1.1 View Results()
1.2 Remind feeling()
1.3 Retreve Verdicts (comcrit, company)
1.4 Access API()
1.5 Create verdict list()
1.6 Retrieve verdict list()
1.7
1.8
1.9 Compute Mood, Worst Failure and Greatest Achievement()
1.10 Post data()
Moody Layering
(View Results)

When we ignore the details such as filtering the verdicts, the mechanism becomes very easy to follow.


Milgram
Program
Verdict list
Endpoints
Berry
1.0 Retrieve verdict list()
1.1 Retrieve endpoint for verb()
1.2 Create verdict list()
1.3 Retrieve verdict list()
1.4
1.5
1.6 Pass data to Moody()
Milgram Layering
(View Results)

The same thing is true here as well.

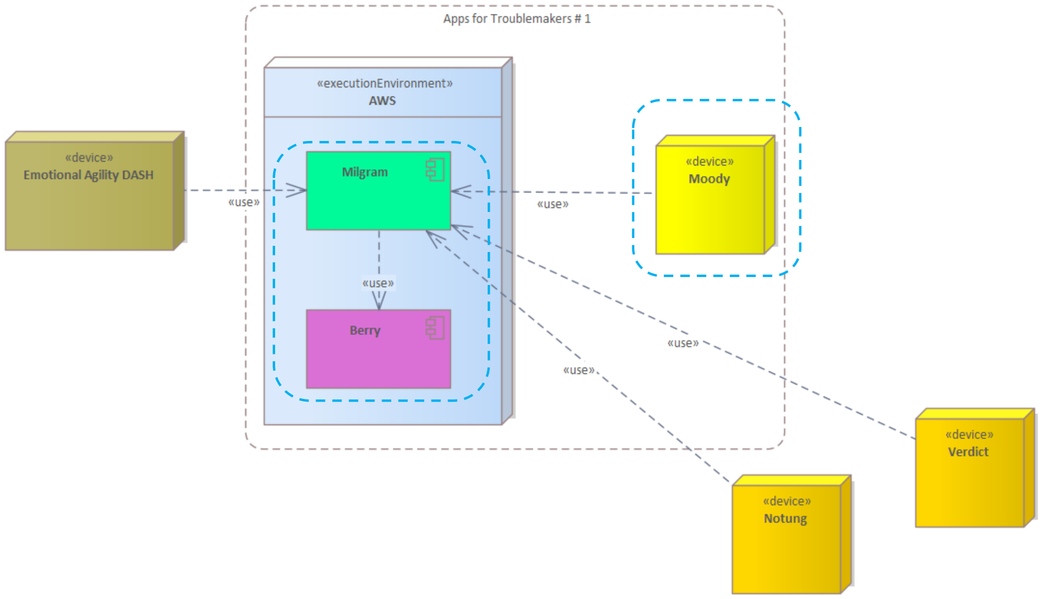
- *A "service" on the side of the client triggers our "API" via an "endpoint".*
- *When triggered, APIs ask only one question, "What do you want!"*
- *You should always answer in the same manner, "I want <?>!"*
- *Here what we want is explicitly expressed by a specific endpoint (i.e. "/createVerdict").*
  "I want to save a new verdict!"
- *(a) Since endpoints and Http verbs go hand in hand, the "how" portion of the Q&A is obvious (i.e. "POST").*
- *(b) API retrieves what was asked for from the database, delivers it and does not do anything else (i.e. "GET").*

Deja Vu

Here we go again! We will repeat the process we have followed the last time… with a twist. Let me remind you of the steps before going any further. Because those steps must be engraved in your mind.

1) *Some kind of information is displayed on a page.*
2) *That information is put there by a controller which is in control of entities that hold parts of that information.*
3) *Controller accesses that information by a service which retrieves it from the server via an API.*
4) *API accesses the requested information via a service which accesses a database.*

Let's summarize the whole thing in a visual manner incorporating the other apps in the "Emotional Agility" suite.



As you can see, the major route on this map is as follows:

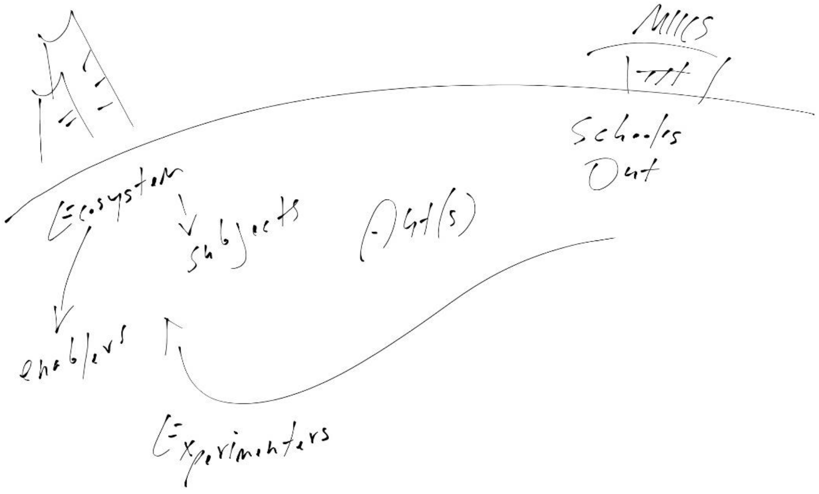
- *Moody = app = Views → Controller → Models → Service*
- *Milgram = api = Program (read "controller") → Models → Service*
- *Berry = database = Collections → Documents*

This route is followed by the interlinked apps, "Verdict" (issues are exposed and supported by clues while authorities are alerted) and "Notung" (ideas are gathered leading to best solutions) as well. Our one-page dashboard app EAD follows the same pattern even if it doesn't need actor intervention. Remember, when it is opened, EAD just accesses the database via our API, retrieves relevant information, conducts certain computations and briefly displays the state of the union, so to speak. I mean, which message can be more important than "You are fucked!" or "You are OK."

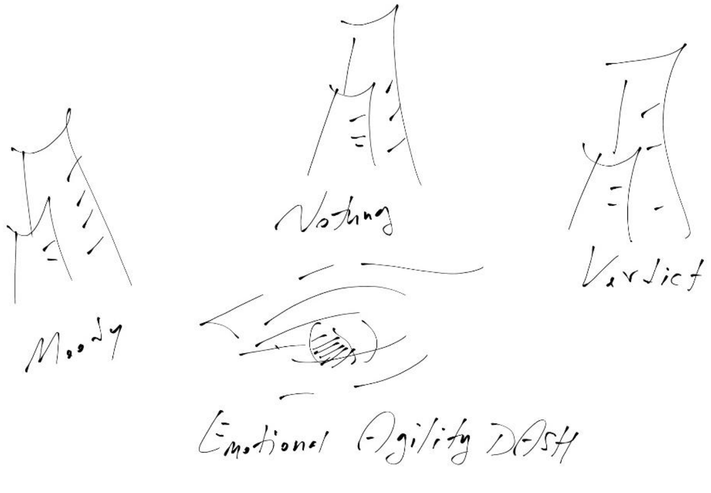
- *Emotional Agility DASH (EAD) = feeling + reason + issue + clue + idea + solution.*
- *Milgram = API to access "Berry".*
- *Berry = NoSQL database.*

There, whether you are developing mobile apps or one-page web apps (which can be used to consolidate information created by interlinked apps), the system architecture is the same. What's more, you can make your one-page web app "a one-page mobile app", if that's more to your liking. It won't change anything at all. Oh, by the way, we will start referring to those apps as "consolidators" from now on. Because that's what they are.

It is always a good idea to remind each other of our goals, before going further. Because architectural decisions as well as requirements must all serve the same master. If we lose track of this, it really doesn't matter how fancy our app is. We would be bound to lose no matter what. We are not for the apps or those behind them. Apps are for us. And we are for the people... with a twist as usual.



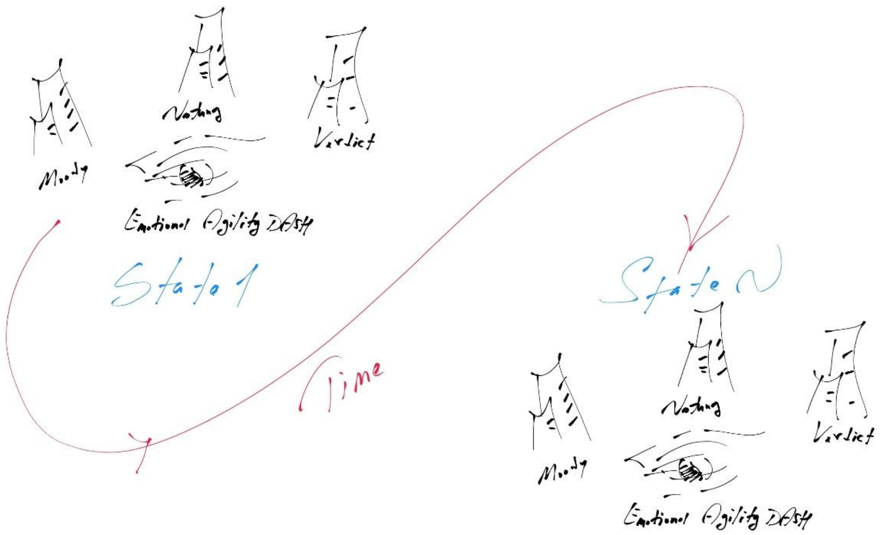
First and foremost, we want to recruit exploitable (!) people. We want subjects to experiment on, and we want enablers who will help us spread the disease. On the other side of the coin, we will do our best to create more and more experimenters. So, subjects and enablers will come into contact with our experimenters in the medium we would like to call as "Apps for Troublemakers". Experimenters are subjects (!) who have the guts to do something about the things that are bothering them. "Subjects" becoming "troublemakers"... are becoming "experimenters" one fine day.



The second goal is to control the target ecosystem in a loose manner, so that interlinked apps and the consolidator call all the shots. Don't understand "control" in the wrong way. We don't have to control things like nasty politicians working in the background. When you formulate your apps for troublemakers properly, humanity will take care of the rest. It's like leaving a baseball bat on the table. While a family may generate sports legends, others may just generate thugs.

Yet, others may just beat the shit out of each other. So, we control some immediately available components of ecosystems. Those with free spirits may just ignore them.

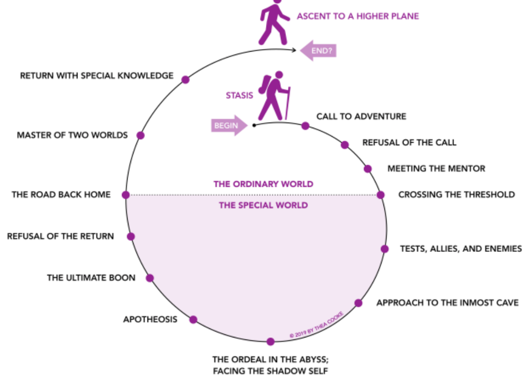
The third and the final goal is using all three mobile apps and the consolidator together while paying attention to the time dimension. Understand where you are, decide where you want to be and use our 3+1 apps as your compass while navigating from point A to point B in a step-by-step fashion. You don't have to know exactly where you are heading. Knowing the general direction is good enough. The method will show you where.

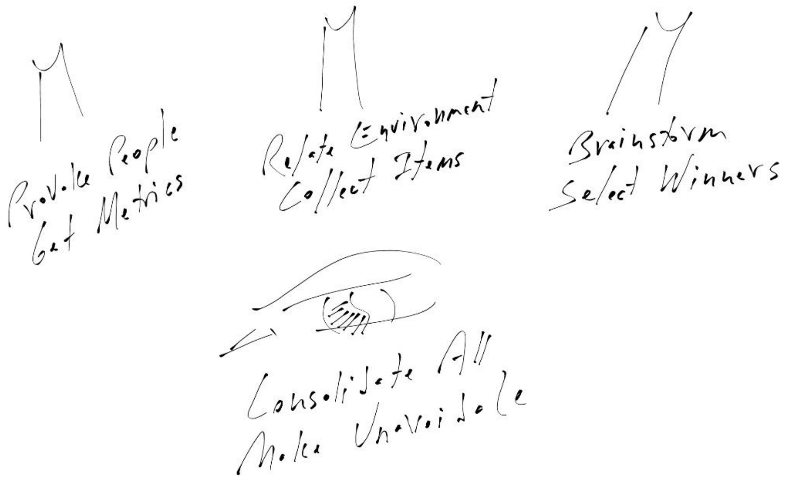


Here's how:

- *Moody gives you feelings and reasons.*
- *Verdict gives you issues and clues.*
- *Notung gives you ideas and solutions.*
- *Emotional Agility DASH consolidates information coming from all three apps into a one-page analysis.*

This tells you what "where" is. You can use this approach to understand your circumstances. Then, you can work backwards to decide where you want to be. Figure out how you can move from Point A to Point B in terms of "feelings, reasons, issues, clues, ideas and solutions". As you are moving towards Point B, use our 3+1 to not get lost.

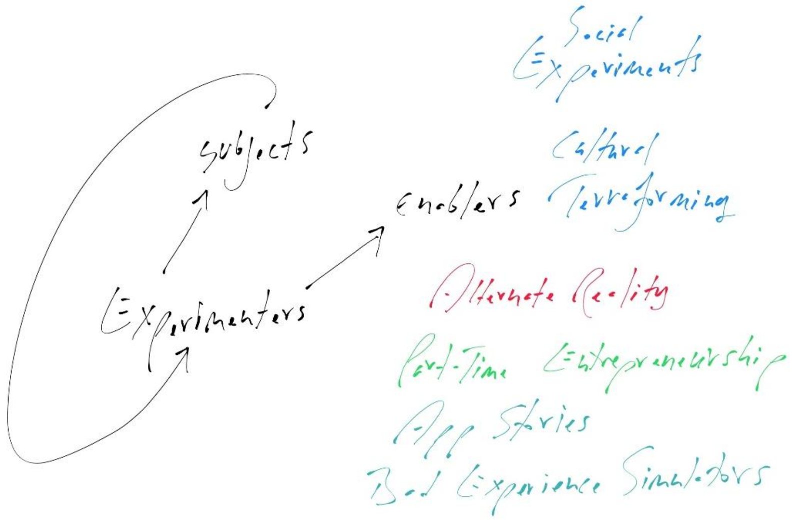




Let's make our 3+1 apps generic, applicable to all kinds of different goals:

- *Find an interesting issue. Find the associated ecosystem.*
- *Spot your subjects. Find a way to provoke them. This is your app formula AKA "first app".*
- *Whatever seems to be the problem, find clues in the field and make them known. This is your "second app".*
- *Let people go over feelings and facts, so that ideas begin to pour in. This is your "third app".*
- *When you put all those together on a single page, this is your "consolidator".*

Issue = Subject = Provocation Point = Clues = Ideas = Big Picture



Of course, there is yet another layer where this book and the associated apps are just one of the components of a much bigger scheme. That layer is the NoRM Institute which has seven dimensions.

- *School Alternate*
- *Social Experiments (This is where we are.)*
- *Cultural Terraforming (Maddening Moves or strategic deployment of interlinked apps dispersed over time)*

- *Alternate Realities*
- *Part-Time Entrepreneurship Outlets*
- *App Stories*
- *Bad Experience Simulators*

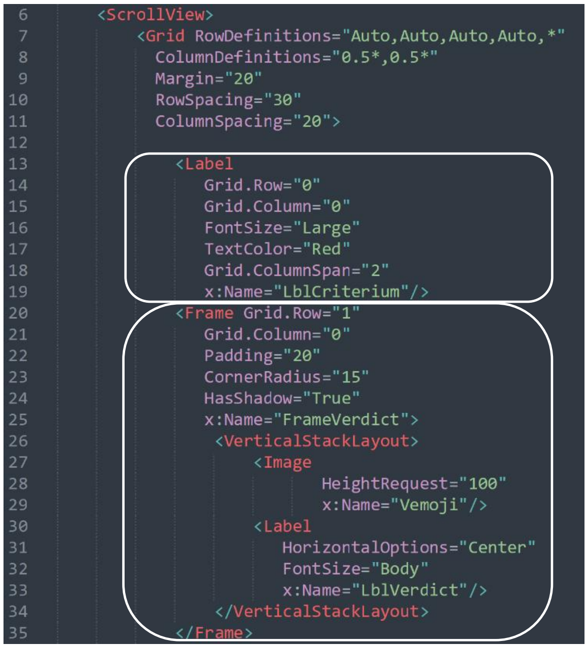
What, you need more information? Did I leave the important part out? No. I didn't. "How" is straightforward in this process. For example, if your employees are unhappy because of the work life balance, fix it! This means moving from Point A to Point B by fixing pressing issues one by one. No, you cannot fake it anymore. If you want to formulate your next move, just look at how "feeling – reason – issue – clue – idea - solution" are merging and act accordingly.

Of course, there is an easy way out. You don't do maddening moves. In this case, use mobile apps separately until someday someone braver than you (!) comes along. However, what kind of a pathetic existence would that be? If you are choosing this option, you'd better run, pal. You are not the one for the legends… or for this book.

Do the Dirty

Now that we remembered where we were, let's proceed… to the next page, "AnalysesPage.xaml".

Why wreck something which is working just fine? Let's follow the "VotingPage.xaml" example here too.



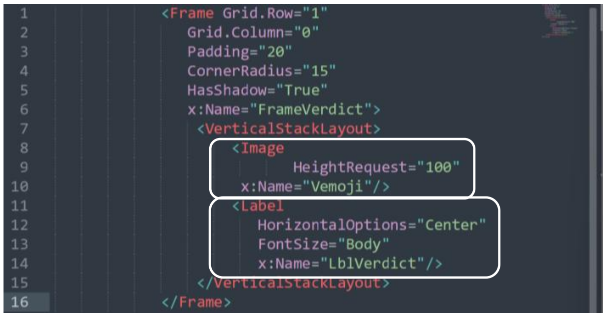
```
    <ScrollView>
        <Grid RowDefinitions="Auto,Auto,Auto,Auto,*"
          ColumnDefinitions="0.5*,0.5*"
          Margin="20"
          RowSpacing="30"
          ColumnSpacing="20">

            <Label
               Grid.Row="0"
               Grid.Column="0"
               FontSize="Large"
               TextColor="Red"
               Grid.ColumnSpan="2"
               x:Name="LblCriterium"/>
            <Frame Grid.Row="1"
               Grid.Column="0"
               Padding="20"
               CornerRadius="15"
               HasShadow="True"
               x:Name="FrameVerdict">
                <VerticalStackLayout>
                    <Image
                           HeightRequest="100"
                           x:Name="Vemoji"/>
                    <Label
                        HorizontalOptions="Center"
                        FontSize="Body"
                        x:Name="LblVerdict"/>
                </VerticalStackLayout>
            </Frame>
```

Create a <Grid></Grid> and place a <Label/> in it. Then, place a <Frame></Frame> just underneath the label. We will use the label to display the criterium selected during the voting process to provide a perspective for "AnalysesPage.xaml".

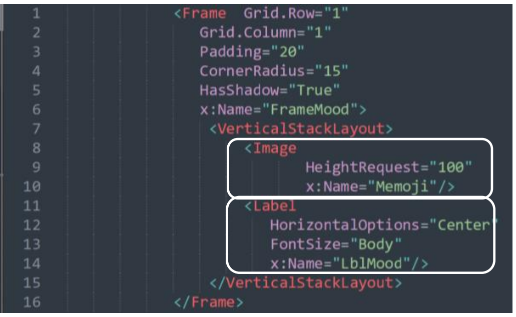
As you can see, <Frame></Frame> will be placed on the left (Grid: 0, 1), because there will be another <Frame></Frame> right next to it. On the left we will display an employee's verdict (feeling) and on the right, we will display her community's mood (feelings).

Displaying information in contrast to each other is especially useful in social experiment apps, because it enables the actor immediately to assess her position in her community. If your community has positive feelings towards something that irritates you, maybe you should change your community or perhaps you should try to convince them that you are right. This is actionable information, people.

Here you can see both frames side by side ("verdict" vs "mood"). On the left of the page, we will see an emoji for verdict followed by an appropriate message like "You said Nay!" On the right of the page, we will see an emoji for mood followed by an appropriate message like "They said Yay!"



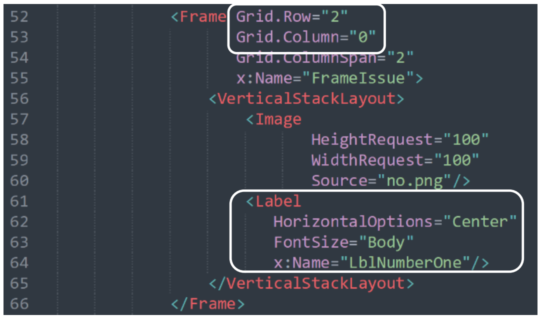
```
<Frame Grid.Row="1"
   Grid.Column="0"
   Padding="20"
   CornerRadius="15"
   HasShadow="True"
   x:Name="FrameVerdict">
    <VerticalStackLayout>
        <Image
               HeightRequest="100"
         x:Name="Vemoji"/>
        <Label
            HorizontalOptions="Center"
            FontSize="Body"
            x:Name="LblVerdict"/>
    </VerticalStackLayout>
</Frame>
```



```
<Frame  Grid.Row="1"
   Grid.Column="1"
   Padding="20"
   CornerRadius="15"
   HasShadow="True"
   x:Name="FrameMood">
    <VerticalStackLayout>
        <Image
               HeightRequest="100"
               x:Name="Memoji"/>
        <Label
            HorizontalOptions="Center"
            FontSize="Body"
            x:Name="LblMood"/>
    </VerticalStackLayout>
</Frame>
```

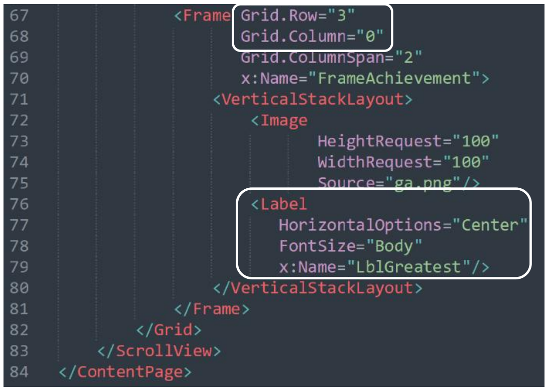
The values of "Vemoji", "LblVerdict", "Memoji" and "LblMood" will be handled in the code behind file, "AnalysesPage.xaml.cs".

Towards the end of <Grid></Grid> we will add the sections for "the number one issue" and "the greatest achievement" of the company.



```
<Frame Grid.Row="2"
       Grid.Column="0"
       Grid.ColumnSpan="2"
       x:Name="FrameIssue">
    <VerticalStackLayout>
        <Image
               HeightRequest="100"
               WidthRequest="100"
               Source="no.png"/>
        <Label
            HorizontalOptions="Center"
            FontSize="Body"
            x:Name="LblNumberOne"/>
    </VerticalStackLayout>
</Frame>
```

While it is possible to place them side by side, I chose to display one after the other to add contrast to the page. I mean, even our tiny little app deserves some panache, don't you think?

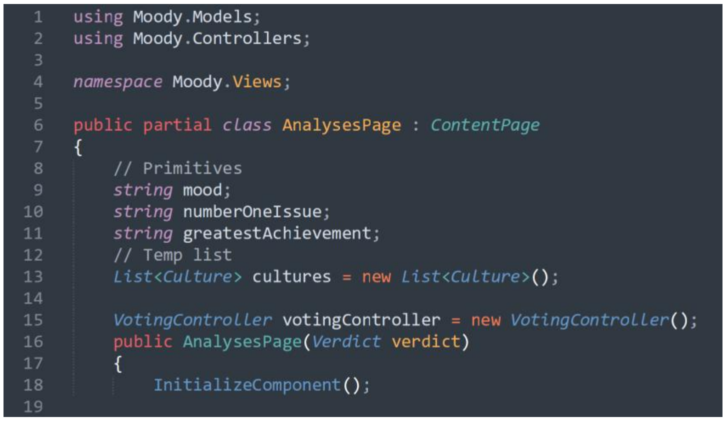


```
        <Frame Grid.Row="3"
               Grid.Column="0"
               Grid.ColumnSpan="2"
               x:Name="FrameAchievement">
            <VerticalStackLayout>
                <Image
                       HeightRequest="100"
                       WidthRequest="100"
                       Source="ga.png"/>
                <Label
                    HorizontalOptions="Center"
                    FontSize="Body"
                    x:Name="LblGreatest"/>
            </VerticalStackLayout>
        </Frame>
    </Grid>
  </ScrollView>
</ContentPage>
```

As you can see, I have chosen to display bad news before good ones, because it's the more important information. However, you may play with the idea. Some people prefer to hear good news first.

Which one are you?

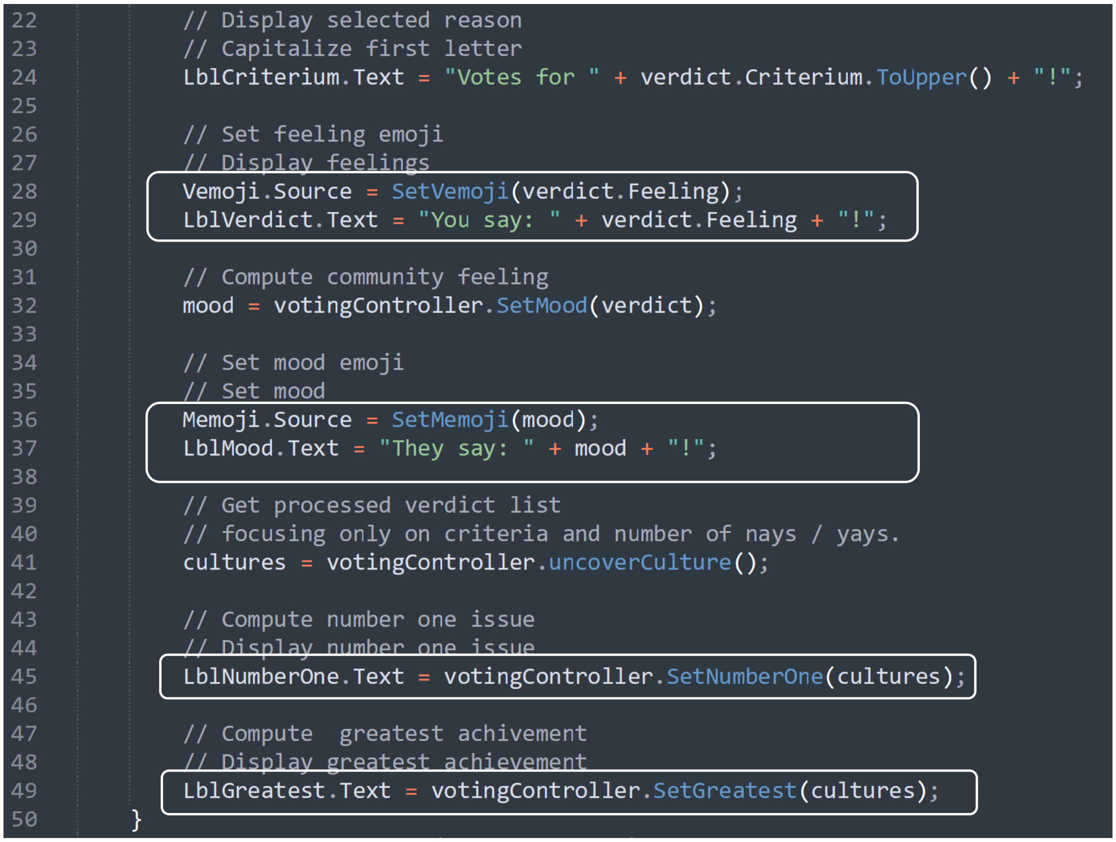
Let's continue with the code behind file, "AnalysesPage.xaml.cs" which is where the real action takes place.



```
using Moody.Models;
using Moody.Controllers;

namespace Moody.Views;

public partial class AnalysesPage : ContentPage
{
    // Primitives
    string mood;
    string numberOneIssue;
    string greatestAchievement;
    // Temp list
    List<Culture> cultures = new List<Culture>();

    VotingController votingController = new VotingController();
    public AnalysesPage(Verdict verdict)
    {
        InitializeComponent();

```

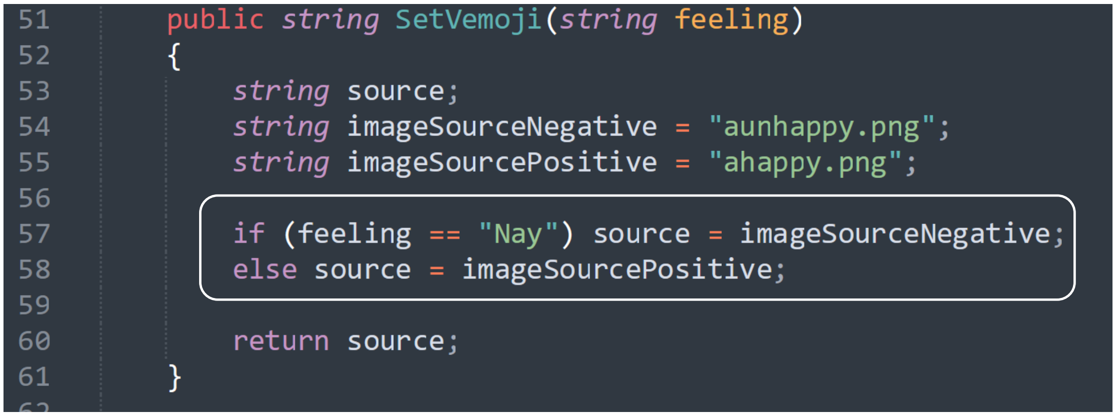
As you can see, the analyses page fulfills her promise by using models and controllers. Since there will be a few intermediary computations, we will need additional primitives and lists. The more interesting one among them is "cultures", a diminished (!) snapshot of a company where you can find the number of positive and negative votes regarding every criterium. Isn't this what we call culture, a bunch of yay(s) and nay(s)? Your preferences of ordinary things… Of course, I'm teasing.

It's best to have a bird's view first, so that we won't get lost. Here is everything at the "function call" level.



```
        // Display selected reason
        // Capitalize first letter
        LblCriterium.Text = "Votes for " + verdict.Criterium.ToUpper() + "!";

        // Set feeling emoji
        // Display feelings
        Vemoji.Source = SetVemoji(verdict.Feeling);
        LblVerdict.Text = "You say: " + verdict.Feeling + "!";

        // Compute community feeling
        mood = votingController.SetMood(verdict);

        // Set mood emoji
        // Set mood
        Memoji.Source = SetMemoji(mood);
        LblMood.Text = "They say: " + mood + "!";

        // Get processed verdict list
        // focusing only on criteria and number of nays / yays.
        cultures = votingController.uncoverCulture();

        // Compute number one issue
        // Display number one issue
        LblNumberOne.Text = votingController.SetNumberOne(cultures);

        // Compute  greatest achivement
        // Display greatest achievement
        LblGreatest.Text = votingController.SetGreatest(cultures);
    }
```

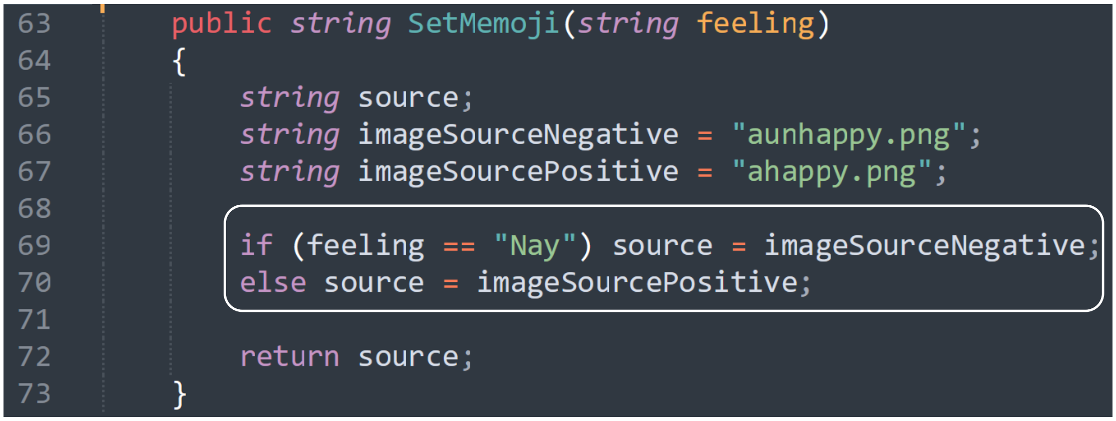
Let's go over these functions one by one starting from "SetVemoji()".



```
        public string SetVemoji(string feeling)
        {
            string source;
            string imageSourceNegative = "aunhappy.png";
            string imageSourcePositive = "ahappy.png";

            if (feeling == "Nay") source = imageSourceNegative;
            else source = imageSourcePositive;

            return source;
        }
```

As you can see, all we are doing is just changing the name of the image so that image source targets the right emoji for the current feeling our employee.

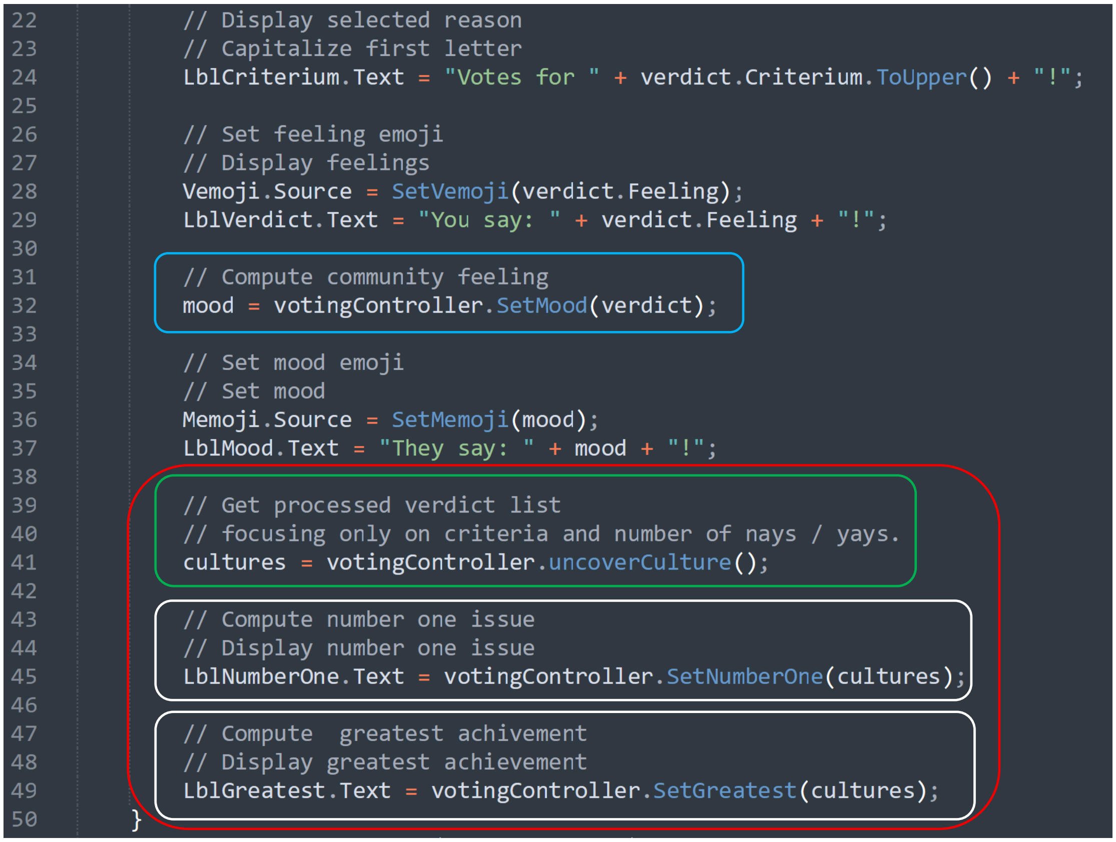
You should be able to guess this, but it's always better to ask. Yes, since we don't do anything beyond user interface manipulations in our code behind files, the only other method defined here is the other emoji setter, "SetMemoji()".



```
        public string SetMemoji(string feeling)
        {
            string source;
            string imageSourceNegative = "aunhappy.png";
            string imageSourcePositive = "ahappy.png";

            if (feeling == "Nay") source = imageSourceNegative;
            else source = imageSourcePositive;

            return source;
        }
```

Again, all we are doing is just changing the name of the image so that image source targets the right emoji for the collective feelings of our employee's community (mood). How about we stop fooling around and get down to business?

Psycho, Alfred Hitchcock, 1960.



```
        // Display selected reason
        // Capitalize first letter
        LblCriterium.Text = "Votes for " + verdict.Criterium.ToUpper() + "!";

        // Set feeling emoji
        // Display feelings
        Vemoji.Source = SetVemoji(verdict.Feeling);
        LblVerdict.Text = "You say: " + verdict.Feeling + "!";

        // Compute community feeling
        mood = votingController.SetMood(verdict);

        // Set mood emoji
        // Set mood
        Memoji.Source = SetMemoji(mood);
        LblMood.Text = "They say: " + mood + "!";

        // Get processed verdict list
        // focusing only on criteria and number of nays / yays.
        cultures = votingController.uncoverCulture();

        // Compute number one issue
        // Display number one issue
        LblNumberOne.Text = votingController.SetNumberOne(cultures);

        // Compute  greatest achivement
        // Display greatest achievement
        LblGreatest.Text = votingController.SetGreatest(cultures);
    }
```

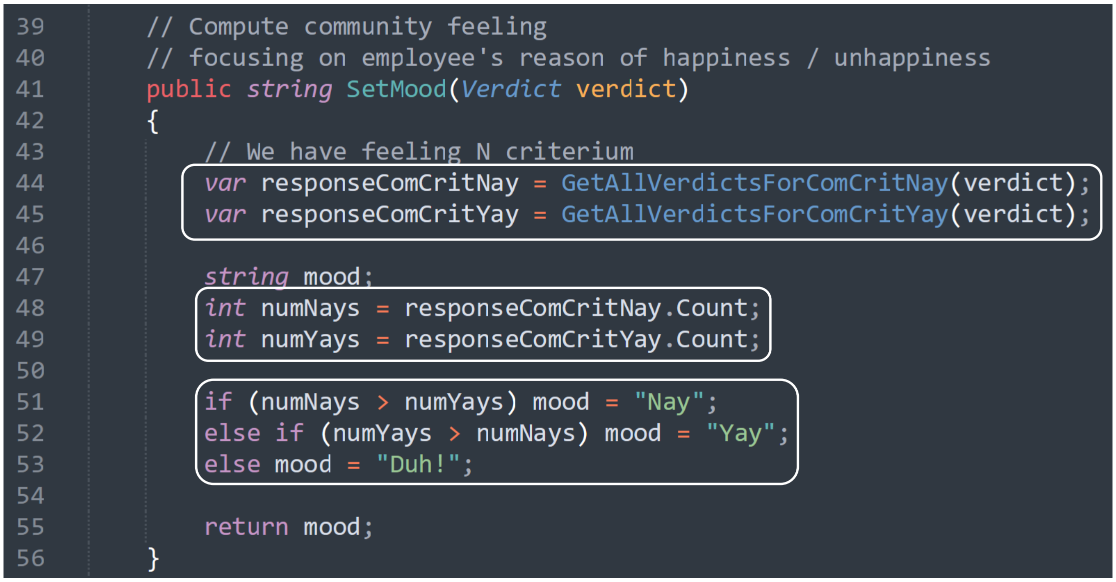
For the first extra information we don't do much. We just retrieve a filtered verdicts list ("employee's community" AND "her selected criterium"). Then, we count negative and positive verdicts. Finally, we determine the greatest number which reveals the collective feeling of the community (mood).

- *Send "community and criterium" to controller and receive the "feeling with the greatest number of votes".*
- *Create a temporary list which holds the numbers of negative and positive verdicts towards all twelve criteria.*
- *Find the criterium with the greatest negative number of votes (number one issue).*
- *Find the criterium with the greatest positive number of votes (greatest issue).*

Shadow of a Doubt, Alfred Hitchcock, 1943.

Now that we are past introductions, let's dive deep into details.

Let's compute "mood".

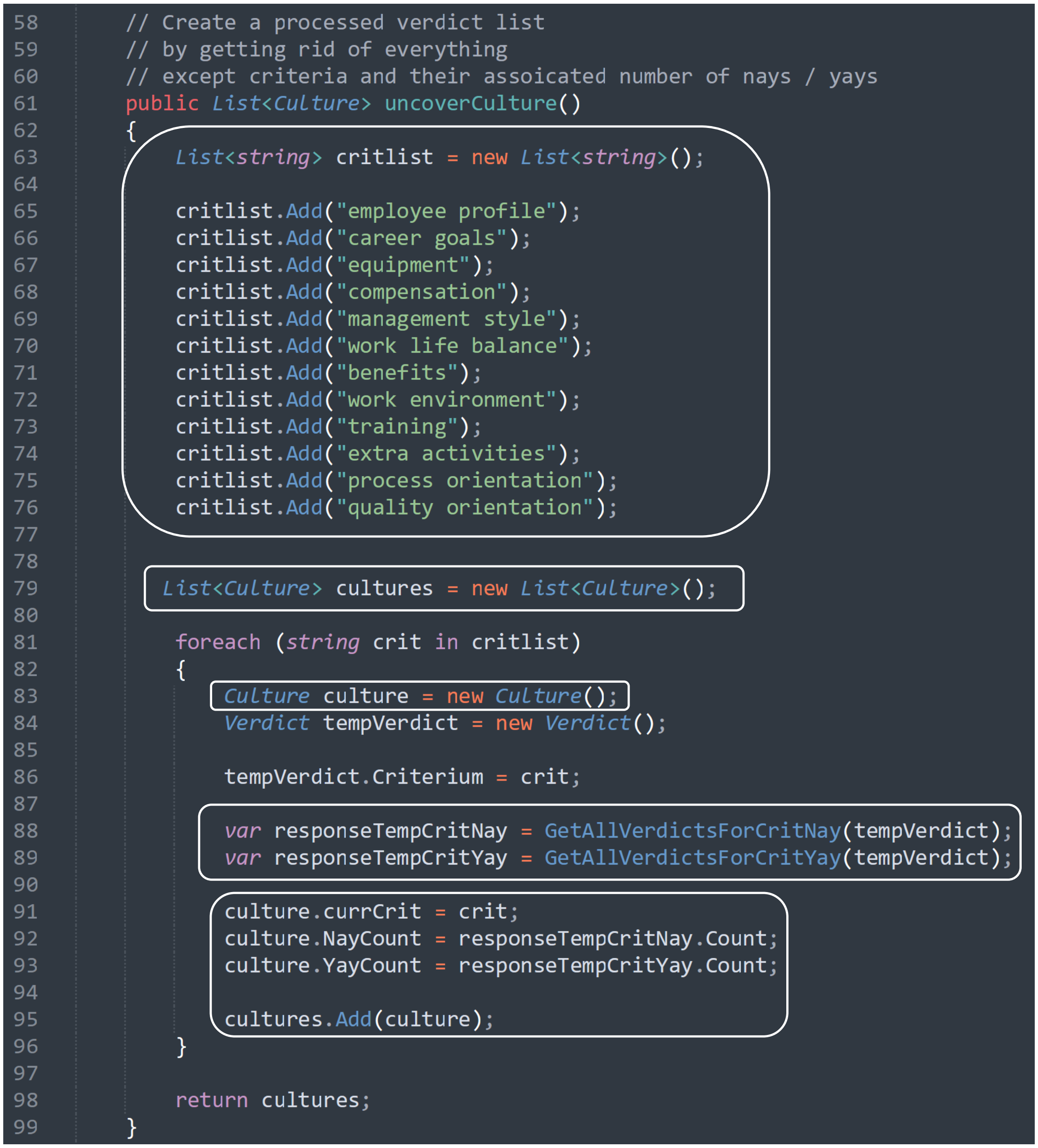


```
    // Compute community feeling
    // focusing on employee's reason of happiness / unhappiness
    public string SetMood(Verdict verdict)
    {
        // We have feeling N criterium
        var responseComCritNay = GetAllVerdictsForComCritNay(verdict);
        var responseComCritYay = GetAllVerdictsForComCritYay(verdict);

        string mood;
        int numNays = responseComCritNay.Count;
        int numYays = responseComCritYay.Count;

        if (numNays > numYays) mood = "Nay";
        else if (numYays > numNays) mood = "Yay";
        else mood = "Duh!";

        return mood;
    }
```

- *Get all negative verdicts for community AND criterium.*
- *Get all positive verdicts for community AND criterium.*
- *Count the items in the negative verdicts list. Assign value to a primitive, "numNays".*
- *Count the items in the positive verdicts list. Assign value to a primitive, "numYays".*
- *Compare both numbers and determine mood by finding the one with the greater value.*
- *Assign value to a primitive and return it as the result, "mood".*

Marnie, Alfred Hitchcock, 1964.

Here's how we are going to create a processed (read, temporary) list, "cultures".

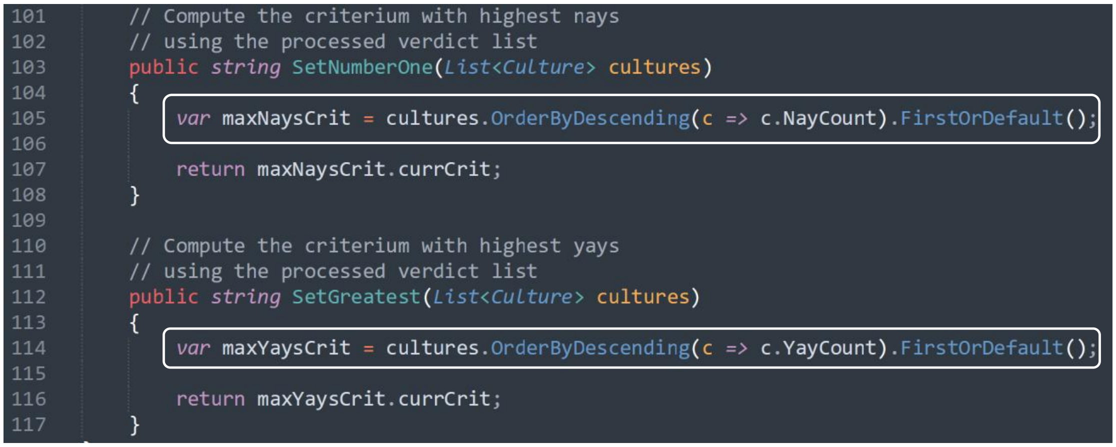


```
    // Create a processed verdict list
    // by getting rid of everything
    // except criteria and their assoicated number of nays / yays
    public List<Culture> uncoverCulture()
    {
        List<string> critlist = new List<string>();

        critlist.Add("employee profile");
        critlist.Add("career goals");
        critlist.Add("equipment");
        critlist.Add("compensation");
        critlist.Add("management style");
        critlist.Add("work life balance");
        critlist.Add("benefits");
        critlist.Add("work environment");
        critlist.Add("training");
        critlist.Add("extra activities");
        critlist.Add("process orientation");
        critlist.Add("quality orientation");


        List<Culture> cultures = new List<Culture>();

        foreach (string crit in critlist)
        {
            Culture culture = new Culture();
            Verdict tempVerdict = new Verdict();

            tempVerdict.Criterium = crit;

            var responseTempCritNay = GetAllVerdictsForCritNay(tempVerdict);
            var responseTempCritYay = GetAllVerdictsForCritYay(tempVerdict);

            culture.currCrit = crit;
            culture.NayCount = responseTempCritNay.Count;
            culture.YayCount = responseTempCritYay.Count;

            cultures.Add(culture);
        }

        return cultures;
    }
```

- *Have a list of criteria handy.*
- *Create a culture object.*
- *Get all negative verdicts for current criterium*
- *Get all positive verdicts for current criterium.*
- *Count negative and positive verdicts.*

- *Assign verdict counts to current culture object.*
- *Repeat the process for all criteria.*

We follow a loose concept of best and worst. As long as a criterium finds itself on the top, that's good enough for us. We don't differentiate criteria with the same number of negative or positive votes. I mean, what's the difference?

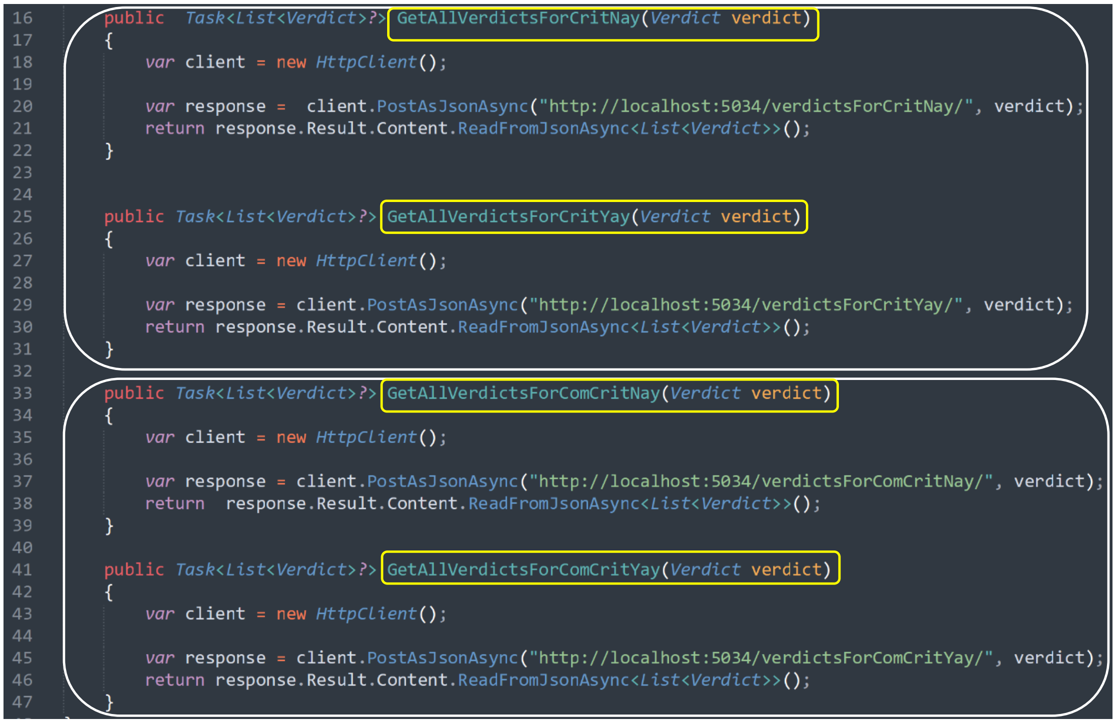


```
    // Compute the criterium with highest nays
    // using the processed verdict list
    public string SetNumberOne(List<Culture> cultures)
    {
        var maxNaysCrit = cultures.OrderByDescending(c => c.NayCount).FirstOrDefault();

        return maxNaysCrit.currCrit;
    }

    // Compute the criterium with highest yays
    // using the processed verdict list
    public string SetGreatest(List<Culture> cultures)
    {
        var maxYaysCrit = cultures.OrderByDescending(c => c.YayCount).FirstOrDefault();

        return maxYaysCrit.currCrit;
    }
```

- *Sort list of cultures from the highest to the lowest negative verdict count, "culture.NayCount".*
- *Return the associated criterium as result*
- *Sort list of cultures from the highest to the lowest positive verdict count, "culture.YayCount".*
- *Return the associated criterium as result*

Of course, none of these would be possible if we couldn't access the database via our API. So, without further ado, let's go over "MilgramService.cs" in detail. There are two groups of methods and four methods, two in each.

Strangers on a Train, Alfred Hitchcock, 1951.

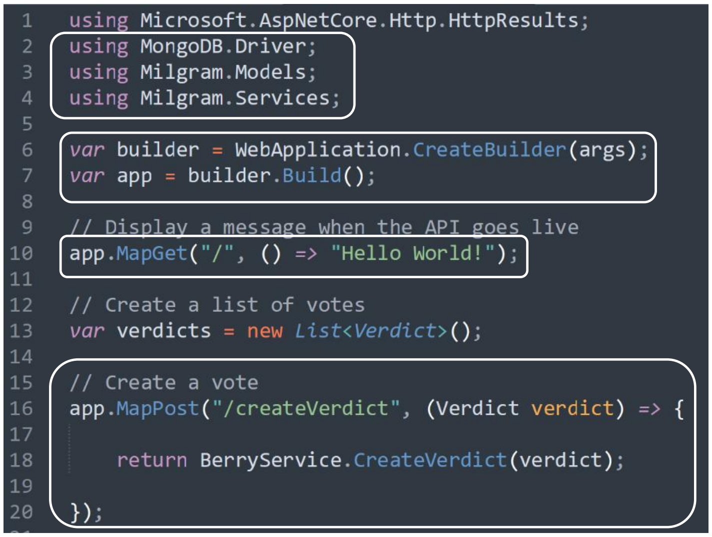


```
    public  Task<List<Verdict>?> GetAllVerdictsForCritNay(Verdict verdict)
    {
        var client = new HttpClient();

        var response =  client.PostAsJsonAsync("http://localhost:5034/verdictsForCritNay/", verdict);
        return response.Result.Content.ReadFromJsonAsync<List<Verdict>>();
    }


    public Task<List<Verdict>?> GetAllVerdictsForCritYay(Verdict verdict)
    {
        var client = new HttpClient();

        var response = client.PostAsJsonAsync("http://localhost:5034/verdictsForCritYay/", verdict);
        return response.Result.Content.ReadFromJsonAsync<List<Verdict>>();
    }

    public Task<List<Verdict>?> GetAllVerdictsForComCritNay(Verdict verdict)
    {
        var client = new HttpClient();

        var response = client.PostAsJsonAsync("http://localhost:5034/verdictsForComCritNay/", verdict);
        return  response.Result.Content.ReadFromJsonAsync<List<Verdict>>();
    }

    public Task<List<Verdict>?> GetAllVerdictsForComCritYay(Verdict verdict)
    {
        var client = new HttpClient();

        var response = client.PostAsJsonAsync("http://localhost:5034/verdictsForComCritYay/", verdict);
        return response.Result.Content.ReadFromJsonAsync<List<Verdict>>();
    }
```

The first two retrieve verdicts associated with the criterium employee has selected during voting and give us two lists containing those in favor or against it. The second group goes further. These functions retrieve verdicts associated with the criterium employee has selected during voting and restrict them to those coming from her community. Then, they give us two lists containing those in favor or against it.

Are we having fun yet! Yes, finally, we are at the pearly gates, my friends. This is the final step of our tiny little app. This is all the information we need to wreak havoc. Just by making the differences between the individuals, their communities and the corporate culture visible on a daily basis we can disrupt anything anywhere anytime. NEEHEEHEEHEEHAHAHA!

To Catch a Thief, Alfred Hitchcock, 1955.



```
using Microsoft.AspNetCore.Http.HttpResults;
using MongoDB.Driver;
using Milgram.Models;
using Milgram.Services;

var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();

// Display a message when the API goes live
app.MapGet("/", () => "Hello World!");

// Create a list of votes
var verdicts = new List<Verdict>();

// Create a vote
app.MapPost("/createVerdict", (Verdict verdict) => {

    return BerryService.CreateVerdict(verdict);

});
```

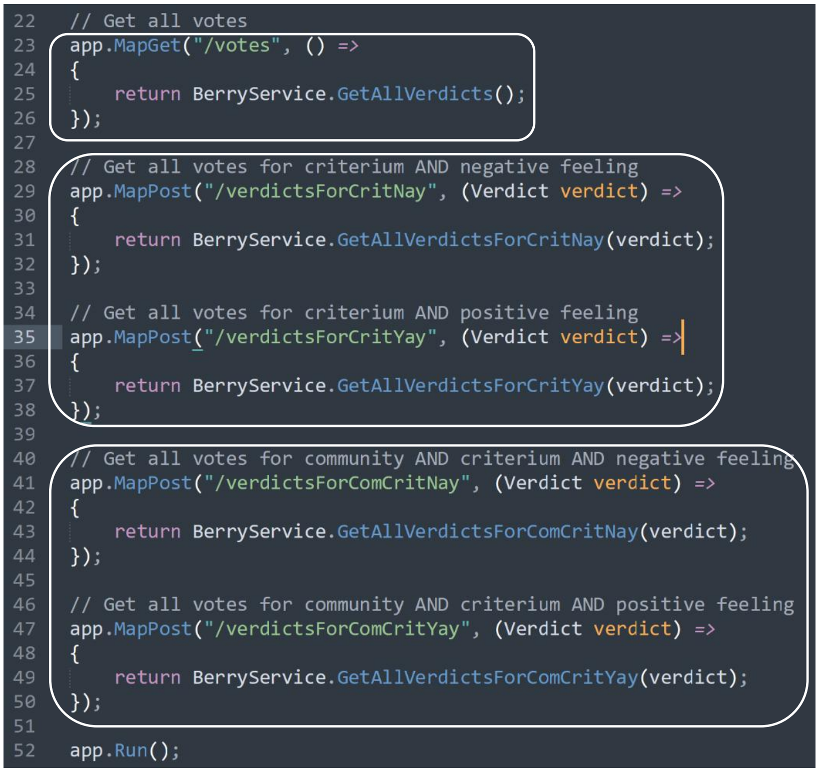
"Program.cs" which is at the root of our API, Milgram, acts like a controller that knows the order of the events that should take place to get everything up and running. It also has all the functions that handle information or action requested by our client app, "Moody".

- *Have MongoDB support*
  Have access to models and services.
- *Build a server app.*
- *Make server app testable.*
- *Create our only POST function.*

Now that we are done with the basics, let's create the server functions.

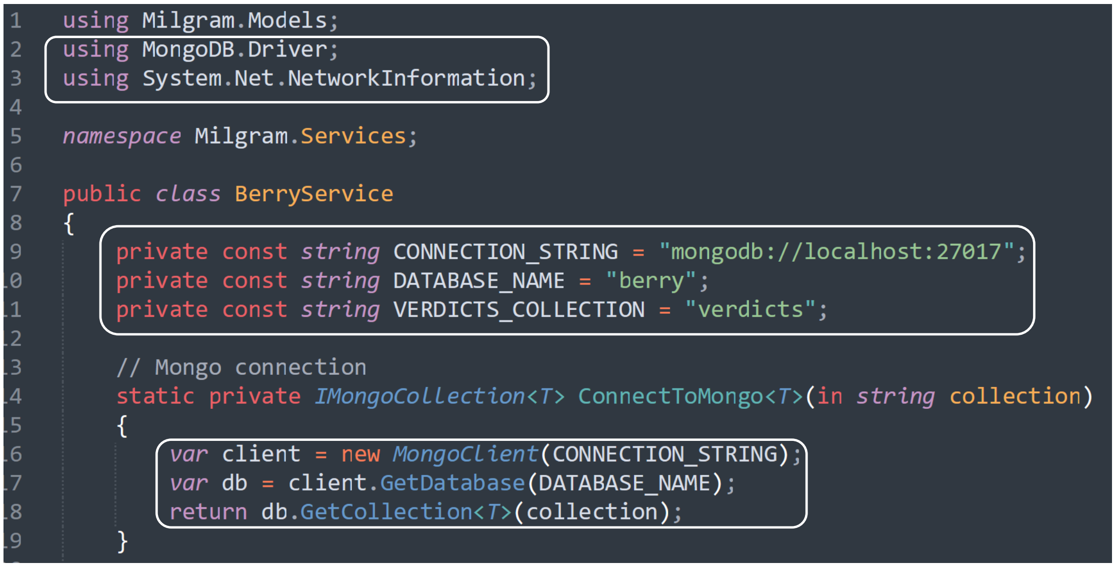
- *We have a function that gets all verdicts if one needs to have a look at all the verdicts of a day (never happens). If we used a similar function for votes, you'd get all the votes... meaning a very big list... meaning useless.*
- *The next two functions get all the verdicts after filtering those in the pool for "criterium" AND "feeling". The first one focuses on "negativity" and the second one focuses on "positivity".*
- *The remaining two functions add "community" to the mix. They only get verdicts from the employee's department.*
- *Finally, we make all those functions accessible by running our server (API).*

Notorious, Alfred Hitchcock, 1946.



```
// Get all votes
app.MapGet("/votes", () =>
{
    return BerryService.GetAllVerdicts();
});

// Get all votes for criterium AND negative feeling
app.MapPost("/verdictsForCritNay", (Verdict verdict) =>
{
    return BerryService.GetAllVerdictsForCritNay(verdict);
});

// Get all votes for criterium AND positive feeling
app.MapPost("/verdictsForCritYay", (Verdict verdict) =>
{
    return BerryService.GetAllVerdictsForCritYay(verdict);
});

// Get all votes for community AND criterium AND negative feeling
app.MapPost("/verdictsForComCritNay", (Verdict verdict) =>
{
    return BerryService.GetAllVerdictsForComCritNay(verdict);
});

// Get all votes for community AND criterium AND positive feeling
app.MapPost("/verdictsForComCritYay", (Verdict verdict) =>
{
    return BerryService.GetAllVerdictsForComCritYay(verdict);
});

app.Run();
```

As usual, the final piece of the puzzle is yet another service, "BerryService.cs".

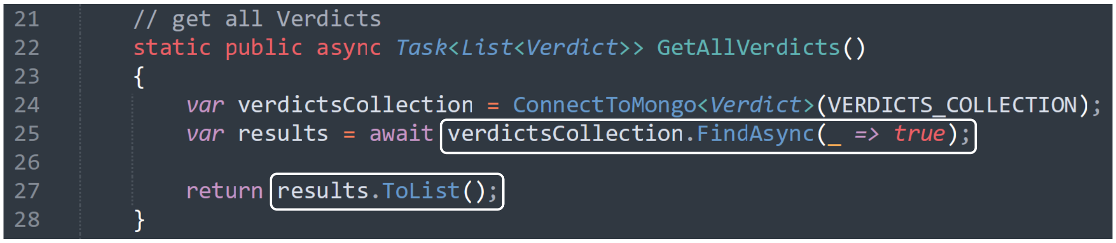


```
using Milgram.Models;
using MongoDB.Driver;
using System.Net.NetworkInformation;

namespace Milgram.Services;

public class BerryService
{
    private const string CONNECTION_STRING = "mongodb://localhost:27017";
    private const string DATABASE_NAME = "berry";
    private const string VERDICTS_COLLECTION = "verdicts";

    // Mongo connection
    static private IMongoCollection<T> ConnectToMongo<T>(in string collection)
    {
        var client = new MongoClient(CONNECTION_STRING);
        var db = client.GetDatabase(DATABASE_NAME);
        return db.GetCollection<T>(collection);
    }
```

The first interesting bit about this service is "it has support for MongoDB and Http protocol". I hope you still remember why. Because this time we will access the database and also execute appropriate Http verbs to do what we are supposed to do. For the sake of reusability, we have our constants, "database URL", "database name", "relevant collection name".

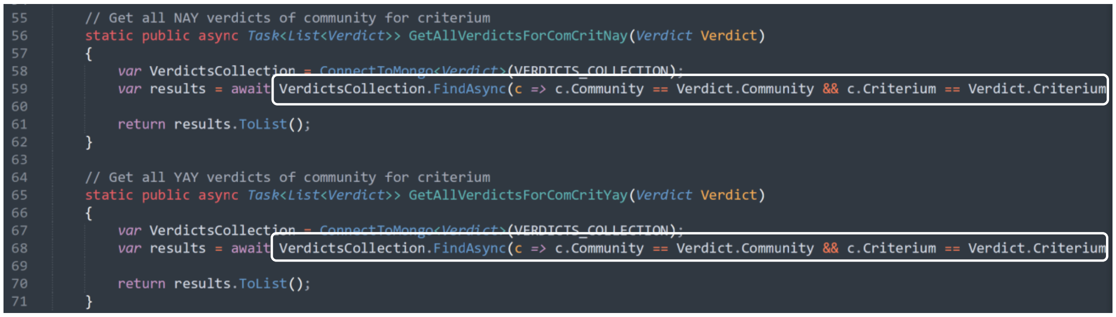
Similarly, we have a function that handles connections to the database. It receives the name of the collection we are after and handles the rest. This implies that we may have several collection names on our constants list.

While being trivial, it is always a great conversation starter. Here's the function, if we wanted to get all verdicts… which is almost never the case. As you can see, it doesn't need a value to operate. Retrieve documents for "_ => true" means retrieve all. Also, since we need a list as the return value, this function (like all the others) turns the requested information to a list before returning it.



```
    // get all Verdicts
    static public async Task<List<Verdict>> GetAllVerdicts()
    {
        var verdictsCollection = ConnectToMongo<Verdict>(VERDICTS_COLLECTION);
        var results = await verdictsCollection.FindAsync(_ => true);

        return results.ToList();
    }
```

Let's turn our attention to functions for retrieving a subset of verdicts keeping "community" AND "criterium" in mind.



```
    // Get all NAY verdicts of community for criterium
    static public async Task<List<Verdict>> GetAllVerdictsForComCritNay(Verdict Verdict)
    {
        var VerdictsCollection = ConnectToMongo<Verdict>(VERDICTS_COLLECTION);
        var results = await VerdictsCollection.FindAsync(c => c.Community == Verdict.Community && c.Criterium == Verdict.Criterium

        return results.ToList();
    }

    // Get all YAY verdicts of community for criterium
    static public async Task<List<Verdict>> GetAllVerdictsForComCritYay(Verdict Verdict)
    {
        var VerdictsCollection = ConnectToMongo<Verdict>(VERDICTS_COLLECTION);
        var results = await VerdictsCollection.FindAsync(c => c.Community == Verdict.Community && c.Criterium == Verdict.Criterium

        return results.ToList();
    }
```

The first one retrieves all verdicts coming from the employee's community with her chosen reason (criterium), if the verdict is negative. The second one does the same for positive verdicts. The filtering method may seem tricky, but it's very simple. What it says in plain English is, "Assume c means the object which has a community property (a culture object). If the value of community is equal to whatever community employee belongs to (i.e. "Technology") and the value of criterium is equal to whatever criterium employee has selected (i.e. "benefits") … then, pick that verdict and put it into a list. Do the same for all verdicts." Remember verdict is a vote with an ignored date-time information. This means we will be working on the votes of a particular day. We aren't interested in exploring the archives. As a result, such a list won't be huge. Also, we won't spend any time for finding the right list.

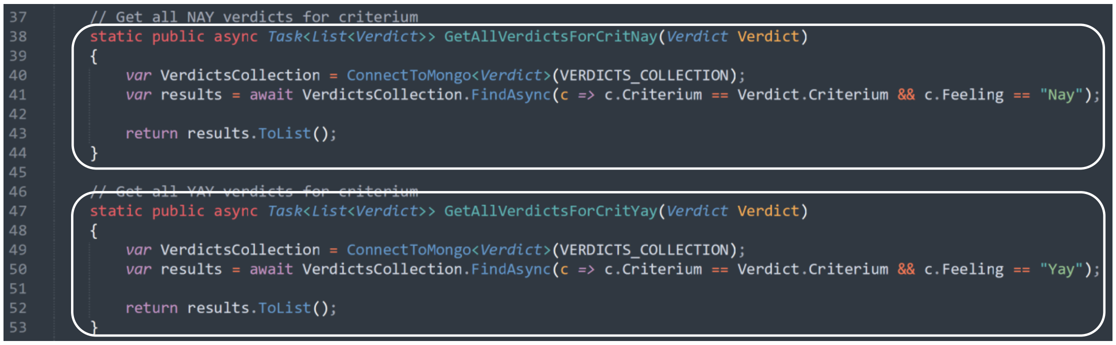
Here's an application.

- *Connect to "verdicts" collection and insert findings into a new list (results).*
- *Go over the new list to find* ***negative*** *verdicts for employee's criterium coming from her community and put them into yet another list.*
- *Format final findings as a list and return them.*

Another application can be as follows:

- ➢ *Connect to "verdicts" collection and insert findings into a new list (results).*
- ➢ *Go over the new list to find* ***positive*** *verdicts for employee's criterium coming from her community and put them into yet another list.*
- ➢ *Format final findings as a list and return them.*

Get it? We do the same things over and over with minor differences here and there. That's programming for you!



```
// Get all NAY verdicts for criterium
static public async Task<List<Verdict>> GetAllVerdictsForCritNay(Verdict Verdict)
{
    var VerdictsCollection = ConnectToMongo<Verdict>(VERDICTS_COLLECTION);
    var results = await VerdictsCollection.FindAsync(c => c.Criterium == Verdict.Criterium && c.Feeling == "Nay");

    return results.ToList();
}

// Get all YAY verdicts for criterium
static public async Task<List<Verdict>> GetAllVerdictsForCritYay(Verdict Verdict)
{
    var VerdictsCollection = ConnectToMongo<Verdict>(VERDICTS_COLLECTION);
    var results = await VerdictsCollection.FindAsync(c => c.Criterium == Verdict.Criterium && c.Feeling == "Yay");

    return results.ToList();
}
```

These two are even simpler. Here we don't care about the communities of the employees. What we are after is finding negative and positive verdicts for every criterium.

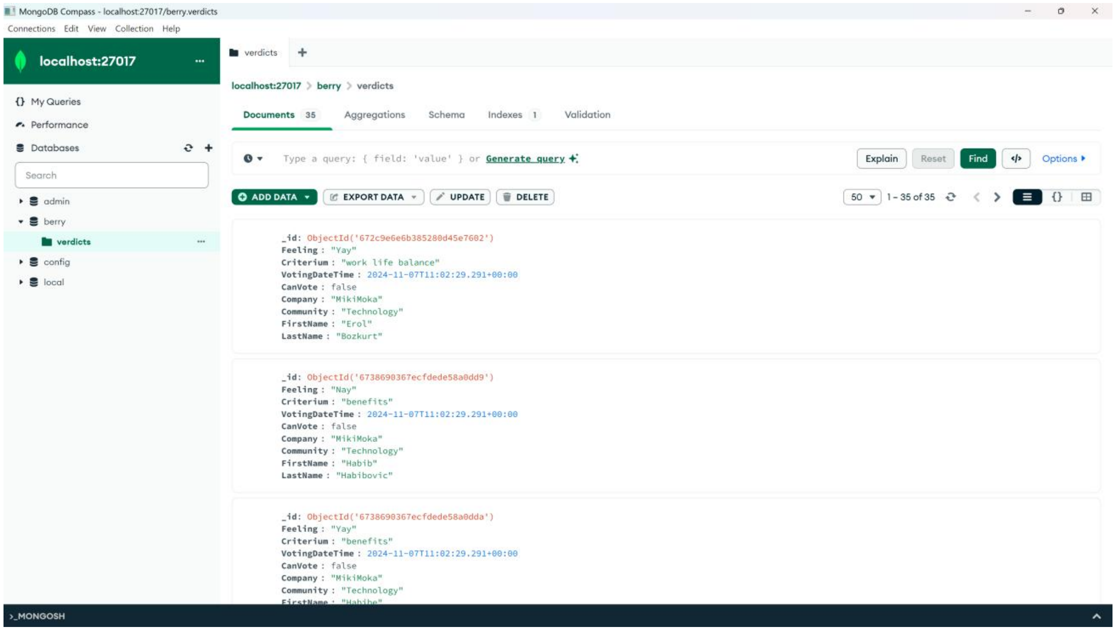
Here's the translation of the first one.

- ➢ *Connect to "verdicts" collection and insert findings into a new list (results).*
- ➢ *Go over the new list to find negative verdicts for employee's criterium and put them into yet another list.*
- ➢ *Format final findings as a list and return them.*
- ➢

Here's the translation of the second one.

- ➢ *Connect to "verdicts" collection and insert findings into a new list (results).*
- ➢ *Go over the new list to find positive verdicts for employee's criterium and put them into yet another list.*
- ➢ *Format final findings as a list and return them.*

And we are finished! I know. I know. I feel weird too.



```
_id: ObjectId('672c9e6e6b385280d45e7602')
Feeling : "Yay"
Criterium : "work life balance"
VotingDateTime : 2024-11-07T11:02:29.291+00:00
CanVote : false
Company : "MikiMoka"
Community : "Technology"
FirstName : "Erol"
LastName : "Bozkurt"

_id: ObjectId('6738690367ecfdede58a0dd9')
Feeling : "Nay"
Criterium : "benefits"
VotingDateTime : 2024-11-07T11:02:29.291+00:00
CanVote : false
Company : "MikiMoka"
Community : "Technology"
FirstName : "Habib"
LastName : "Habibovic"

_id: ObjectId('6738690367ecfdede58a0dda')
Feeling : "Yay"
Criterium : "benefits"
VotingDateTime : 2024-11-07T11:02:29.291+00:00
CanVote : false
Company : "MikiMoka"
Community : "Technology"
FirstName : "Habibe"
```

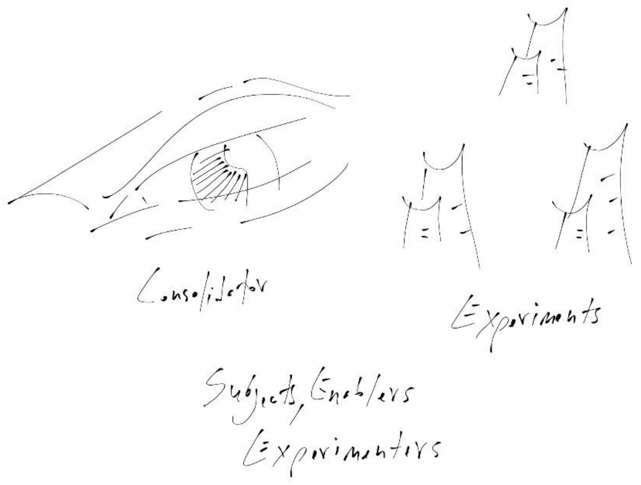
Since we are OK with a barebones version of Moody in this release, all we have is a "verdicts" collection with a lot of "verdict" documents in it. In the next version all these verdicts will be copied to a "votes" collection which will be our archive where we can perform all kinds of operations such as calculating trends. Of course, in that version our Verdict (and Vote) classes will be somewhat bigger because of incorporating more properties… but this is a story for another book. So, let's forget about those and enjoy the day.

Right now (!) every "verdict" document has the following structure:

- *"Feeling" is a string that shows us whether the feeling is positive or negative.*
- *"Criterium" is the feeling generating reason which comes from a customizable list.*
- *"VotingDateTime" is what makes further calculations possible. We ignore it for now (Wiifn).*
- *"CanVote" is a flag that shows us whether the employee has selected both a feeling and a reason.*
- *"Company" is the "current company" obtained from LinkedIn at sign up. Wiifn.*
- *"Community" is the oversimplified version of "sector" obtained from LinkedIn at sign up. Wiifn.*
- *"First Name" is "name" obtained from LinkedIn at sign up. Wiifn.*
- *"Last Name" is "surname" obtained from LinkedIn at sign up. Wiifn.*

Verdict & Notung

Another atypical feature of our micro apps is they come in threes. The reason behind this approach is to provide three experiments that can be linked to one another… to create a paradigm. These apps can not only be linked together at a particular time, but also, they can work in parallel or follow each other with a strategy in mind. What's more, you can even play with the time element. You can conduct a group of experiments, and you may formulate a goal which can be expressed as certain outcomes from similar experiments conducted at a later date. You can group experiments and hand them over to a singular point of control or you can place them in the target ecosystem in a disjointed fashion. The latter will provide valuable insights into the similarities of the actors flocking behind each app.



Apps for troublemakers can be further merged to form a singular app, if you will, by using a consolidator.

A consolidator is a one-page app which clearly shows how the findings of all three apps relate to each other. It also provides a more valuable metric using this information (our newfound paradigm).

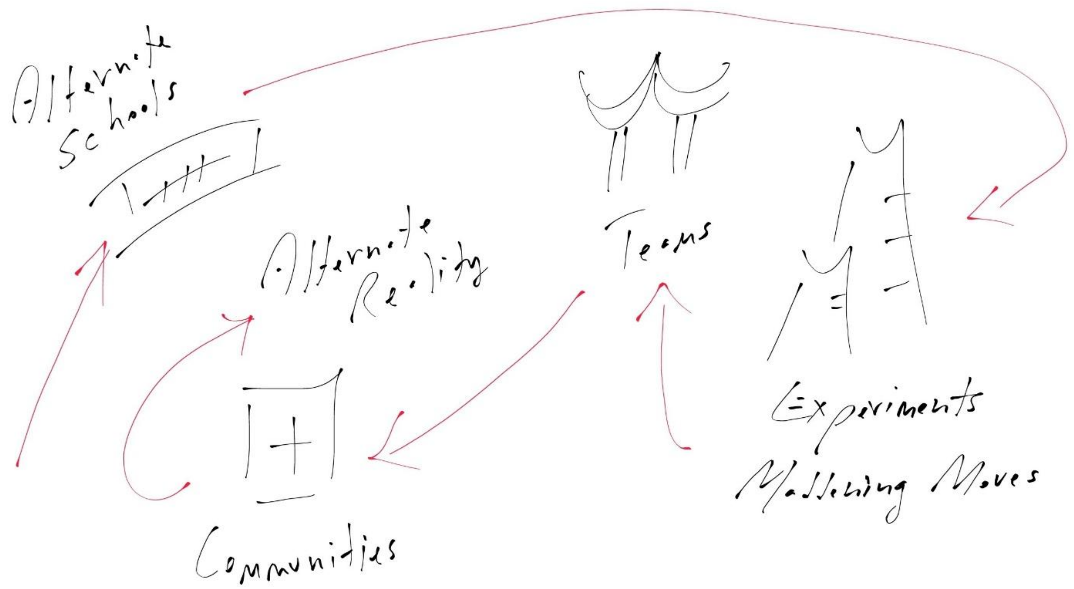
For example, "Moody" tells us how employees feel towards certain things. "Verdict" is all about clues. "Nothung" is all about ideas.

So, you may decide to link feelings to clues, and clues to ideas. This will provide you with a simple way to detect problems, verify them and eventually fix them after brainstorming for the best solutions. That's not the end of it, though. While it may seem that way, we are not here to develop apps. We are here to 'develop people' which will 'develop cultures' in return. When you take care of people, give them a helping hand, let them grow on their own terms, something wonderful happens. You may be surprised to see that even the least remarkable individuals in a community start to shine. That's what I call "cultural terraforming".

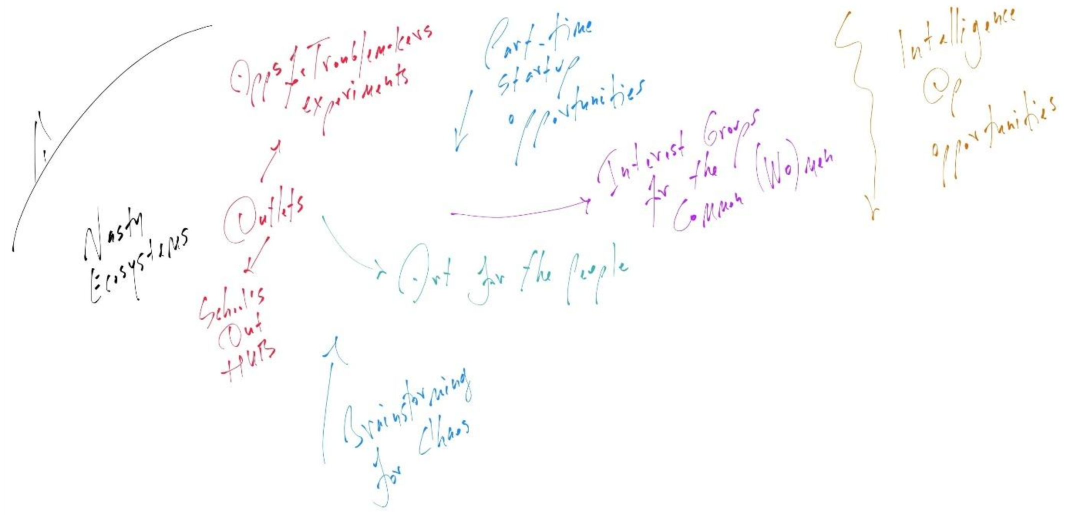
During every experiment we spit out experimenters. When they focus on an ecosystem and start conducting social experiments with their apps for troublemakers, they recruit subjects and enablers. Subjects are actors using our apps. Enablers are subjects in other ecosystems with some kind of power… who are helping us spread the disease.

So, you must think in terms of schools, businesses, hangouts and homes. Experimenters meet with subjects and enablers while they are becoming A-Teams themselves. They are influencing their friends where they hang out, close relatives and families at home. Their enthusiastic friends provide the necessary glue that help them cross the bridge between friends and families. Yes, public support, even a tiny one is a pressure instrument you can use.

When people at school, work, hangouts and homes are merged to form a community… when the links between them are further strengthened by social experiments and cultural terraforming… they eventually create alternate realities, but that's a story for another book. "School's Out, Apps for Troublemakers, Maddening Moves" are one side of the coin. On the other side, we have "Something from Nothing, Brave New Worlds" and "Reality Check". The thin slice of metal (!) between the two is U^2 where Uili and Uusta enable people to become part-time entrepreneurs.

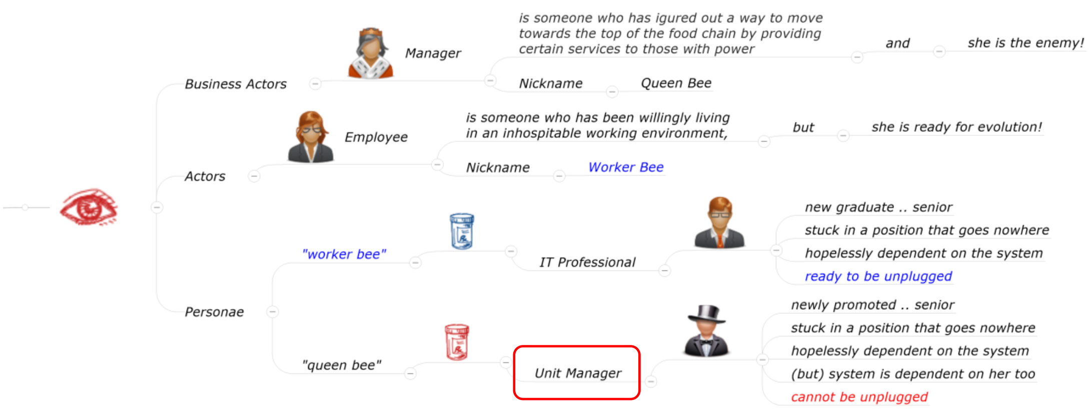


Where do I get ideas from? I always look at nasty places and nasty people. Broken, damaged, diminished human beings… That's my spring of great ideas. And I always think in terms of experiments, training, entrepreneurship, art, strategy and communities. Details will follow soon. That being said, let's focus on the other two apps for troublemakers.


Apps for Troublemakers experiments
Part-time startup opportunities
Intelligence @ opportunities
Interest Groups for the Common (Wo)men
Nasty Ecosystems
@utlets
Art for the People
Schools Out HUB
Brainstorming for Chaos

Verdict

Imagine a post-moody workplace. Everybody is aware of the things that make people happy or unhappy. Whether those findings are ignored or being taken into account is obvious. As a result, the percentages of employees who are individuals and the ones who are diminished are obvious too. So, there are two options here. You can either accept the status quo as your reality and conform or you can take your battle to a whole new level. Verdict can turn feelings into facts by helping you associate clues with them. Also, since every clue ends up in the hands of those who are supposed *to take care of problems, it provides employees with a way to conduct real-time performance reviews of their managers. Cute, huh?*


Business Actors
Manager
is someone who has igured out a way to move towards the top of the food chain by providing certain services to those with power
and
she is the enemy!
Nickname
Queen Bee
Actors
Employee
is someone who has been willingly living in an inhospitable working environment,
but
she is ready for evolution!
Nickname
Worker Bee
Personae
"worker bee"
IT Professional
new graduate .. senior
stuck in a position that goes nowhere
hopelessly dependent on the system
ready to be unplugged
"queen bee"
Unit Manager
newly promoted .. senior
stuck in a position that goes nowhere
hopelessly dependent on the system
(but) system is dependent on her too
cannot be unplugged

As you can see, our actors are exactly same as those for Moody. The only difference is the number of managers which has increased. Why? Because we are no longer limited to Human Resource Managers (What a nasty title!). Now we accept any powerless (read, miserable) mid-level manager into our experiments. Welcome to your Götterdämmerung. NEEHEEHEEHEEHAHAHA!

Persona

A persona gives an actor life. You may consider it like an acting role. Even if you narrow down what one may think of an actor when you name her, you can never really tell how she is going to act in that particular context. However, when you provide a persona for her, you will have an idea about her notion of reality which will help you spot those like her.

<u>Manager John</u>

John has graduated from a very popular department of a very popular university. He spent most of his time in popular places whether it be a hangout or a business gathering. He identifies himself with the citizens of popular countries. He thinks he has no responsibility whatsoever towards those who live close to him, granted they are not in his condominium. Because appearing to be thoughtful or kind helps him get into the pants of ladies nearby. Beyond the walls of his condominium, workplace or favorite café the world does not exist for him. He is not concerned with what he might leave behind. He doesn't have a past or a future. Thus, he doesn't really have friends. He only has allies, fans or suppliers who are dependent on his current circumstances. Worse still, he is always on the run to escape lashes.

He is goal oriented. He seeks constant acknowledgments. He needs to appear superior. Oh, well, he doesn't give a fuck.

- *John Pest.*
- *40.*
- *Marketing Manager.*
- *Famous quote when speaking to someone who actually does something, "This is a sales company!"*
- *Electronics Engineer with a master's in business administration.*
- *Divorced. Remarried to a fashion model without children and has two dogs.*
- *Has a mistress on the side. Since he loves to experiment, he sees a transexual dominatrix once in a while.*
- *Has three condos.*
- *Has a BMW sedan and a custom shop Kawasaki motorcycle.*
- *Has a set of expensive ties. He thinks one should always wear the correct tie to set the tone in a meeting.*

The Main Story

We detailed the manager persona, but here the actor is still our employee. We have done so to help you understand who you will be teasing with your clues. Every issue you spot and document with a photo will end up in John's hands. Think about it. You will be sending clues about issues that make you unhappy, issues John ignores completely. He has other things to do while he is climbing the corporate ladder. Your unhappiness is the least of his concerns. Yet you make it his number one priority with all that nagging. It's fun, don't you think?

12) *You are on the way to somewhere and you spot something wrong.*
You take a photo of the issue, select an appropriate community as the responsible party and send the clue to them.

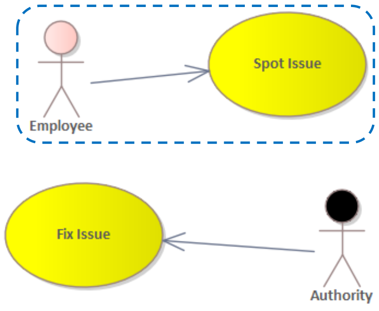
13) *Verdict consolidates the photo and the associated community into a support ticket and assigns ticket to a unit manager.*
Unit manager receives a notification. Think, "Chinese Torture", people!

Unlike typical support ticket systems, "Verdict" doesn't let those in charge select someone to fix the issue. Instead, "Verdict" directly assigns the task to the unit manager. Also, "Verdict" isn't concerned about the process or the claims regarding whether a solution has been provided. If the issue is fixed, it must be obvious to everyone in the office regardless of whether they are aware of what is going on. For example, if dirty toilets are the issue, people will understand they are fixed when they see clean toilets, not when they receive a message claiming everything will be so much better.

So, how do the unit managers understand whether what they did was good enough? Simple. Again, there are no more complicated processes here. No more bureaucracy. If they are no longer receiving notifications from "Verdict" concerning this issue, they are good. If not, they are not. It's that simple.

Use Cases

If we disregard the usual (!) use cases such as "Sign Up" or "Sign In", our second tiny little app is even tinier! Don't worry. That's a good thing. Pissing off those in power with the tiniest of efforts is an art form. Always remember, the one who doesn't know what to do exerts too much effort. The one who knows what to do on the other hand, makes small decisive moves. Many talk the talk. You should always be the one walking the walk.



That being said, let's briefly go over these use cases, shall we? These are vague use cases, mind you.

Spot Issue

Imagine our employee, Helen… is walking from a trivial point A to a trivial point B in the office. All of a sudden, she sees a dirty trash bin. She takes a photo, associates a tag (as in "what's the issue here" like "cleanliness") with the photo and submits it to the appropriate authority which is linked to the number one guy/gal of a department. The number one guy/gal claims the issue. He is the one who is being praised or cursed. That's it! Think about the difference. Today we don't operate that way. Numero uno guys and gals are angelic creatures who don't even take a shit… and all the others are messed up. So, their job is to filter out the bad apples. How about "No", bitch? "Every shit is your shit even if it isn't". That's how management should be done, taking the blame for once and not blaming others.

1) *Employee takes a photo of an issue (within the app, "Verdict").*
2) *System prompts for a tag and an authority*
3) *Employe selects a tag which is linked to an authority.*
   ("cleanliness", "office management department" implying "Jane the Office Manager").
4) *System sends the issue to the authority.*

Authority list is similar to the community list in "Moody". If you are targeting business ecosystems, they are the business departments. So, "an issue" always has to do with a "department". In other words, "Verdict" lets us know the problem-solving capabilities of business departments as well as their maturity. Because people tend to ignore problems when they cannot solve them. What's worse, they become twisted by considering problems as steppingstones. Eventually, they become the problem. Their being good people in the past makes no difference.

Tags are pretty much like the criteria list in "Moody". Think about it this way, "Helen feels blue about the work life balance in her company. She broadcasts negative feelings towards it all the time. And when she spots people sleeping in the office, she takes a photo as a clue of that issue."

Fix Issue

(Optional. I prefer to leave the manager as a business actor who cannot do anything about the issues by using an app.)

Oh, well, it sucks to be you! What can I say? If you happen to be the manager we are targeting here, all you can do is to look at an ever growing or shrinking list of grievances. "Growing" is bad. "Shrinking" is good.

1) *Manager receives a notification and opens it.*
2) *System displays the list of issues (from tags with highest issues to lowest issues) with "freshness" and "age".*
   "Freshness" means whether there are new issues or not. Tags with a greater number of fresh issues go up.
   "Age" means how long a tag has been in the grievances list. Old tags go up. Priority criteria? I don't know, man. How about you find it?
3) *Manager views tags with their associated issues.*

"Verdict" works like the Kanban Boards which don't allow you to hand out responsibility to the machines. Responsibility is yours. If you want to have something fixed even if your complaints are being ignored, you can keep your complaints flowing until something is done. Remember, most of your world is outside of tiny little apps. So, keep talking. The same is true for the managers, regardless of whether this optional functionality is added to "Verdict" or not, what you must do is outside of "Verdict"… unless you are in the dream selling business, of course… and the employees are disciples.

Data

We already have the data types from "Moody", because not only our interlinked micro apps share an API, a database, but also a data model. So, we only have to add what makes "Verdict" unique. We have three classes and five properties.

1) "Issue" is a combination of "a photo", "a short text entry", "location", "a tag" and "an authority".

   For example, "A photo of exhausted employees is a clue of bad work life balance which should be shared with the human resources manager".

   In this current approach, there won't be a short description of the issue. Also, tags will be linked to authorities which are nothing but communities. So, an employee only has to take a photo and select a tag before she is finished. Isn't it more practical this way? Why don't you decide for me? "Location", you say, huh? Well, I'm not sure about that either. I think "speed" and the "ability to make issues visible" are the most important things here. Also, we are in the business of developing tiny apps.
2) "Clue" is the photo associated with the issue.
3) "Complaint" is a modification of issue by adding a few more attributes (inherits from issue).

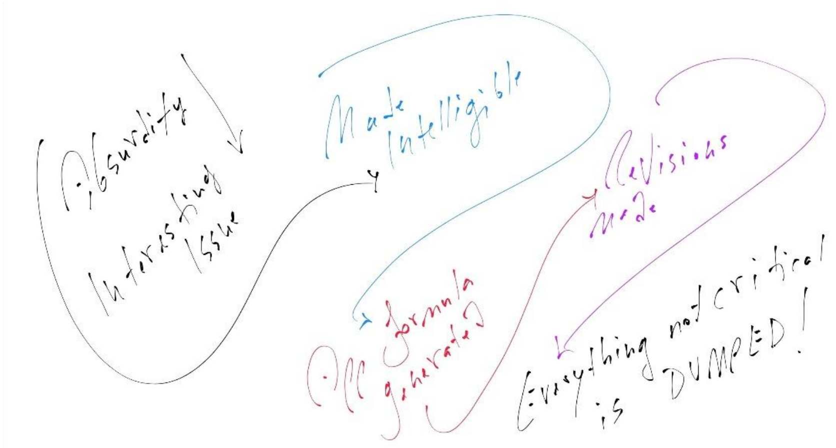
1) Freshness = Whether the latest issue about a tag is new or old.
2) Age = Time between the first and the last issue about a tag.
3) Hot or Cold = Whether issues concerning a tag keep coming or not.
4) Ignorance Index = Whether issues are fixed or not.
5) Performance Review = Whether issues are fixed on a timely basis.

Pages

We have only two pages here as well.

1) Issue Page
   a. Option to add a previously taken photo.
   b. Short description (This may be redundant. Then, again, we won't know until we develop the app.).
   c. A dropdown list of tags which are associated with communities in the ecosystem.
   d. Option to submit issue to the relevant authority.
2) Complaints Page
   a. A list of complaints with attributes, sorted with respect to Verdict Points.
      (Hot or Cold, Ignorance Index, Performance Review).
      {number - tag - issue - clue (click to view) - location - freshness - age}

As you might expect, everybody can see everything in this (Verdict) or the next app (Notung) just like "Moody" to add even more pressure on those with power. The only tactic of self-absorbed pricks is to distort reality to take attention away from themselves which isn't an option here. You can talk the talk. You can lie to your heart's content. You can have bullies or trolls on your payroll. You can have completely desensitized ignoramuses supporting you no matter what… but you can never hide the truth granted there are a few with a lion's heart. It will be up there on the wall always reminding you who you really are. And for those who want to better themselves, it will just be another call to adventure.



Notung

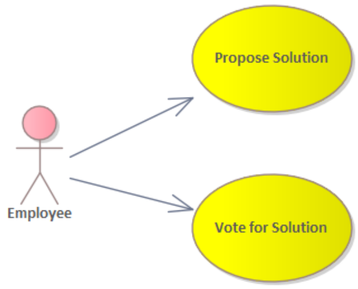
You sometimes wonder why things are never fixed, don't you? The problems seem important. The solutions seem obvious. The people seem capable. Yet, nothing ever changes, because of two things. One, the status quo is the breeding ground for those who are consumed by practical (read, "selfish") goals. Two, when an ecosystem does not give safe passage, so to speak, to those who are more mature than her, the system reverts back to its original state. What this means is that even if you may be implementing new and better processes within the ecosystem, it will be inevitably reduced to its caricature… creating a variant of its original self which can only be carried out by diminished human beings. This new (!) version will look new and better to fools, but for those with keen eyes it will look like the old times.

Persona

Those for "Verdict" will serve us very well here. Yes, I love reusability too! Again, these are vague use cases, mind you.

Use Cases

Here the main idea is to propose solutions for issues causing unhappiness in the target ecosystem. So, "Notung" will benefit from the issues from "Verdict" and tie them to the feelings in "Moody". Our apps are not rigid, meaning they are disjointed. Since they share the same data model and have a single database… they can be even considered one app. This makes it easier to deploy them individually or as a group. This also maintains that our focus is the big picture rather than becoming great in regard to a certain issue. For example, if we keep getting negative feelings, clues or ideas concerning a certain community, that's where we should focus on, not the symptoms. If work life balance sucks in your ecosystem, we shouldn't lose track of what really matters when we are dealing with the shuttle schedule… "getting from point A to point B". There may be many points A(s) and B(s), but they must all reside in a livable world.



That being said, let's briefly go over these use cases, shall we?

Propose a Solution

Imagine that you have been going over a certain problem and developing feasible solutions regarding it. Now, it's your turn to shine. Remember the issues you live with day in and day out? You can have them fixed, granted you get the most votes. There are no limits. Propose one solution after another until one fine day you get to be number one.

1) *Employee submits an idea concerning a certain tag.*
   {"idea", "tag"} where tags are communities in the ecosystem (Moody).
2) *System makes an entry of the idea in the list of ideas regardless of the tag it is associated with.*
   System provides an option to vote for the idea.
   {"number", "idea", "tag"}

The main premise here is to create a pile of action items (!) which can be applied to a problem when it garners enough support. There is no such thing as a consolidated view or a prioritization process. When an idea gets to the top of the list, it becomes a solution even if it doesn't really solve anything. Here, this is the human condition for you.

Vote for Solution

Voting and viewing the list of ideas are one and the same. Voting happens as a reaction to the current list. Also, when an idea is in the vicinity of the bottom of the list, only those with very strong feelings will go there and vote for it. This way unpopular ideas are eliminated even if they are the best ideas. Also, if an idea has a lot of supporters, it will find itself at the top even if those in power hate it. Another feature is (!) not having sorting capabilities. This prevents employees from focusing on their specific problems and instead urges them to focus on what is going on in the workplace. Should we fix these glitches? Hell no! These should be handled out there in the open as a part of the office politics, not behind closed doors. Yet another feature is (!) the ability to vote without a limit (brainstorming diarrhea).

1) *Employee views solution candidates with the option to vote for them.*
   {“number”, “idea”, “tag”}
2) *(optional) Employee votes for an issue. (Employee can vote for many ideas, but only once for the same idea)*
3) *System refreshes the list.*

As you might say, this approach enforces viewers to focus on the first ten, maybe twenty ideas in the list. While one may scroll down as much as she wishes, it’s tiresome. So, many will just take a look at the top ten and no more… which is fine, because we want to turn first ten ideas to solutions applied to one’s ecosystem and move on.

Data

Just like other apps in the trinity (of interlinked apps developed keeping a shared paradigm in mind), “Notung” builds up on the data structures of the others.

- *“Idea” is the idea of an employee in regard to a certain community problem.*
  {number, idea, tag (as in community), isSolution (Is it in top ten?), isSelected (Is it number one)}
- *“Solution” is an idea that has achieved the number one status.*

As you must be aware of it by now, we love mixing things. We mix the realities of an ecosystem with its possible futures. We mix system and business actors. And we always think in terms of app suites… imaginary or real, rather than individual apps. This way, nature golden plates our otherwise incomplete solution. When humans golden plate, it’s a disaster. When mother nature golden plates on the other hand, it’s perfection.

Assumption (No, it’s not mine. I’m just exploiting it!): “Being emotionally agile is good for workplaces”.

- *School’s Out Cycle (i)*
  - *App 1, App 2, App 3 = Moody, Verdict and Notung for SOC1.*
  - *Unifying One-Page App = Emotional Agility DASH for SOC1.*
  - *Unifying Theme (Paradigm) = Emotional Agility for SOC1.*

Pages

Yes, I know, we are creatures of habit… or perhaps, we are just lazy. Who knows? So, the final app in the trinity has two pages too. The second one is for viewing the list and voting for items on it. First one is for sharing your ideas.

1) *Idea Page*
   a. *Dropdown list of tags (communities in the ecosystem).*
   b. *Option to enter one sentence description of the idea.*
   c. *Option to submit idea to the solutions pool.*
2) *Solutions Page*
   a. *List of solutions sorted with respect to their votes (highest to lowest).*
      {“number”, “idea”, “tag”} {“support cause!”}

It's Time to Think!

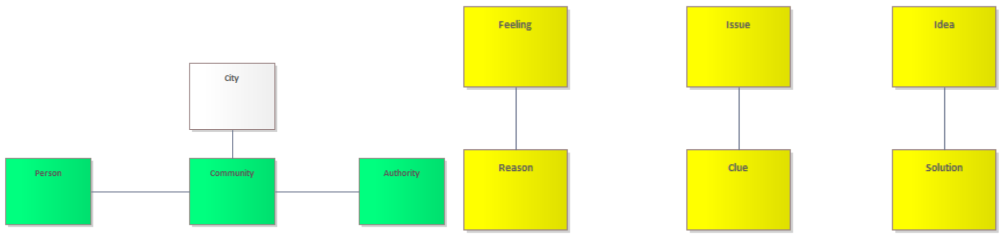
When you deploy all three apps to an ecosystem, whether they are being used implicitly (nobody is aware of it) or explicitly (everybody is aware of it) as an app suite, they feed on each other's findings. Doing so enables the trinity to spit out a consolidated view of that ecosystem which may be displayed by a consolidator later.

- *"Emotional Agility" where having higher points is good.*
- *"Emotional Agility Index" is the metric that represents how responsive and capable management is. (Do I have a formula? Not yet. I'll come up with one. It's not a biggie.)*
- *If a company is emotionally agile, she will have a higher rank among other companies.*
- *Happier employees make companies agile (higher rank). Read, "agile".*
- *Unhappy employees make companies ignorant (lower rank). Read, "not agile".*
- *Every unhappiness a company takes care of rewards her with positive points… making it more agile.*
- *Every unhappiness a company ignores punishes her with negative points… making it not agile.*

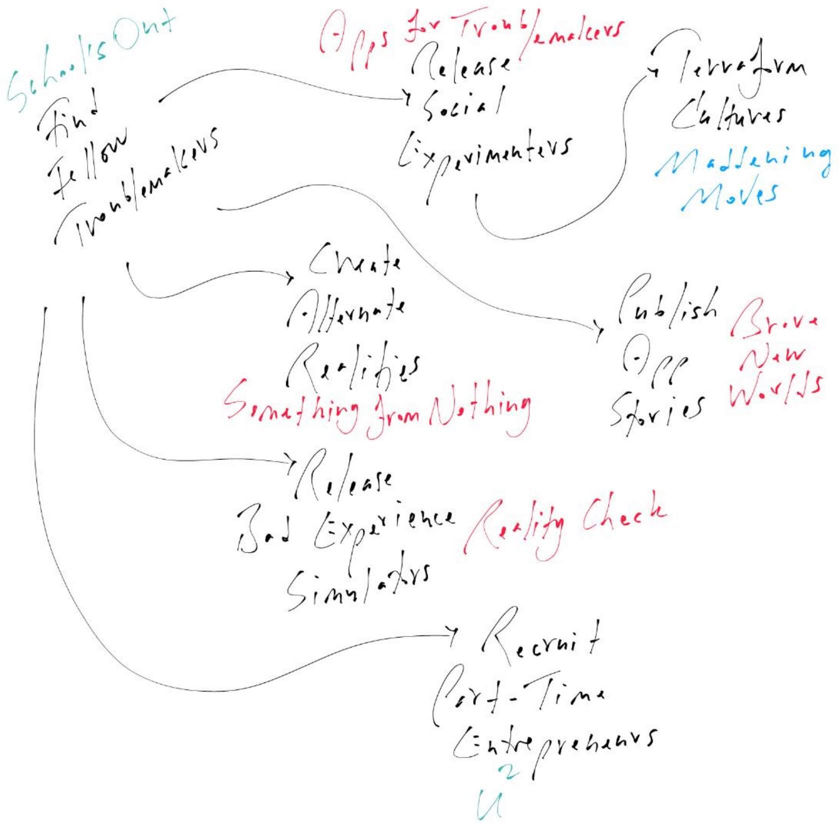
The result may be something like the following:

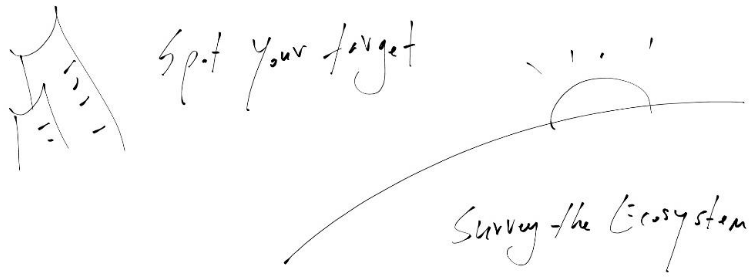
1) *MikiMoka with 66 EAI.*
2) *ACME with 34 EAI.*

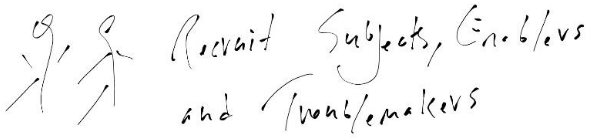
If we take look from above to bring all apps together, we will see that each app fills a void in a much bigger scheme. The same is true for School's Out too. It fills a void in a much bigger scheme. What can I do? I'm a sucker for creating social puzzles. While my cheese is often varied, my mice are always you. NEEHEEHEEHEEHAHAHA! Oops! I shared too much. Anyway… When using "Moody", employees in a company will broadcast their feelings about certain criteria. Since we know their communities, we associate their grievances with their communities… which will be directly associated with issues they encounter in the workplace along with clues they collect to support their claims while using "Verdict". "Notung" will reflect on those findings by enabling them to propose solutions for these problems… bypassing organization structures, processes or management styles to help public rule for a change.



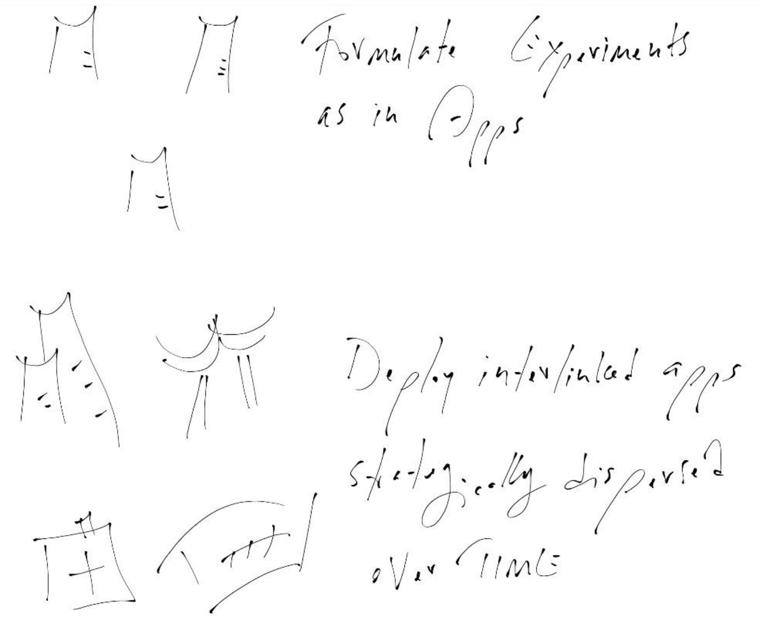
As we all know, strong public opinion always wins. Because even the bloodthirsty tyrants submit to public opinion at one time or another. It's the only thing they are scared of. They don't give a fuck what the truth is. Those who know it doesn't matter too. They are too few in number. There, here's another take on the human condition. Those who make more noise win. That's why it is very important to recruit all shades of troublemakers and make them excited.


School's Out
Find Fellow Troublemakers
Apps for Troublemakers
Release Social Experimenters
Terraform Cultures
Maddening Moves
Create Alternate Realities
Something from Nothing
Publish App Stories
Brave New Worlds
Release Bad Experience Simulators
Reality Check
Recruit Part-Time Entrepreneurs
2
U





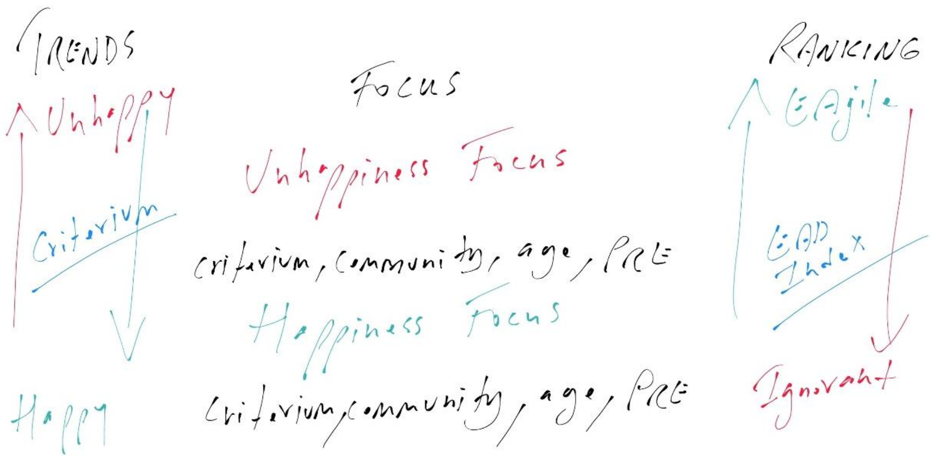
You know, some say we are economic animals. Others say we are animals of habit. I say, we are animals, period. So, ecosystems must be the first and the last thing in your minds. When you control an ecosystem, you control everything. Even if you cannot make reality in your own image, you can unmake all the competing attempts. You make ruins *livable… for scavengers that is. Take a look at Israel or Russia. They just can't get enough of it. I follow the example of the US, but don't panic. My goal is honorable. I want to take humanity to its logical conclusion. NEEHEEHEEHEEHAHAHA!*



Ecosystems, ignored social problems, subjects, experiments, apps, apps and more apps… This is the chapter summary.

Emotional Agility DASH

Designing one-page apps with no actor intervention is a lost art, my friend. Consider them as making inconvenient truths unescapable. No one can bend them to their will, because there is zero possibility of interaction. No one can ignore them, because they are right in your face. You can make it but never fake it. If you cannot fix things, you will always know and learn to live with it… which will eventually create a new type of hero. Someone quite unlike you.



Let's see, what do we have here?

- ➢ *We receive feelings with reasons behind them daily. So, we may have a focus like this:*

  "Benefits create extreme unhappiness!"

  "Especially within the IT Department".

  "This has been the case for 8 months".

  "Previous unhappiness focus was work life balance and even it didn't last that long".

- ➢ *Happiness focus can be displayed similarly:*

  "Employee profile creates extreme happiness!"

  "Especially in the Marketing Department".

  "This has been the case for 4 weeks".

  "Previous happiness focus was benefits".

We can lose ourselves in what's under focus, right? We must have a way to see the big picture. For this purpose, we will provide two bearings. The first one (on the left) is about our company's performance, a simple view that shows us the never-ending battle between happiness and unhappiness reasons. The second one (on the right) is about our company's standing in the real world, outside of the company borders.

It shows whether our company is emotionally agile or not by pointing out our relative ranking with regards to agility (read, "care and concern") and ignorance (read, "conforming or adapting").

See, I can make even the game characters angry. Anyway, how do we calculate emotional agility index?

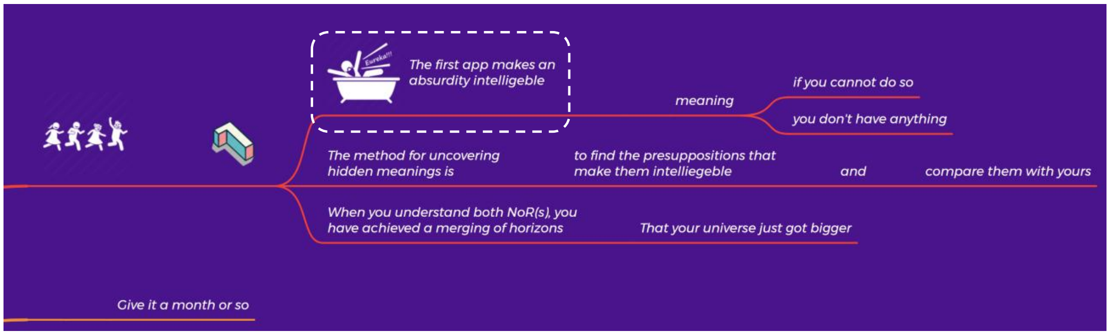
OK, now, since we are making this up, you may find better ways to calculate such a potent metric, a metric that will cause never-ending tears of joy or pain… along with maniacal laughter coming from afar. NEEHEEHEEHEEHAHAHA!

- *The number of "criteria with high numbers of negative feelings". You choose what "high" means.*
- *The number of "issues that won't go away". You choose what "age" means.*
- *The number of "criteria with high numbers of positive feelings". You choose what "high" means.*
- *The number of "issues that never show up". You choose what "age" means.*

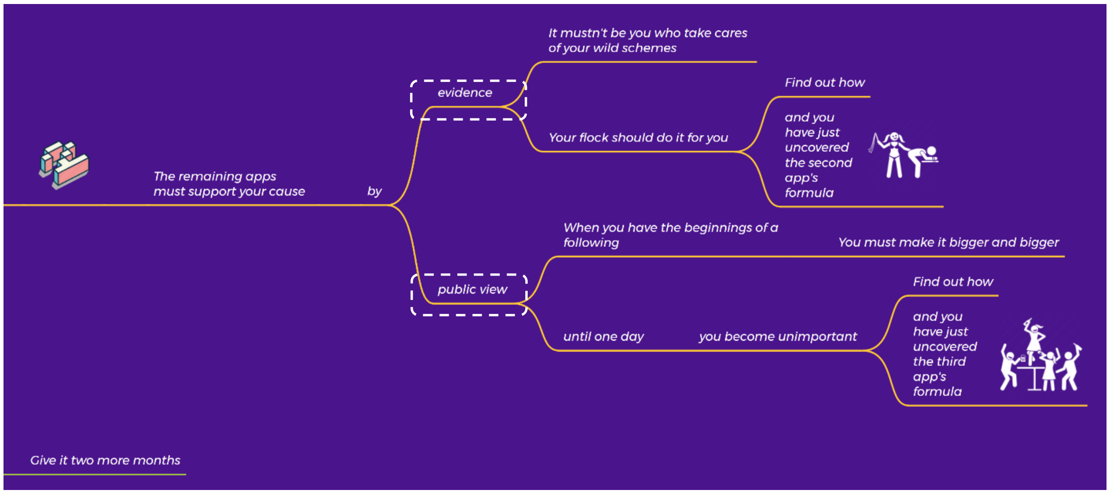
For example,

- *"Benefits" get %30 negative votes company-wide / in a community. We have a problem.*
- *"Benefits" got %30 or more negative votes for two weeks in a row. We have a problem.*
- *"Work life balance" gets %30 positive votes company-wide / in a community. We have a winner.*
- *We never get any issues regarding "work life balance". We have a winner.*

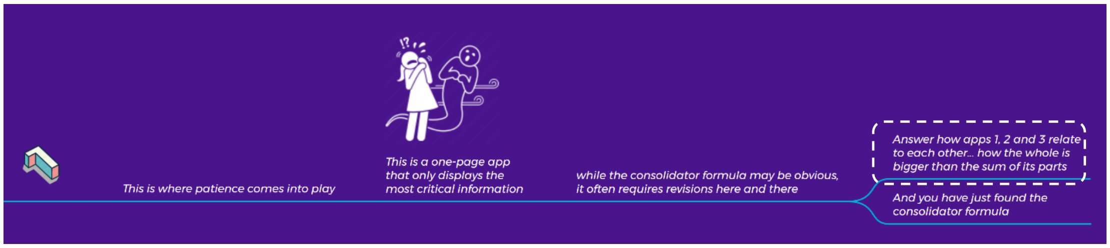
App 1 + App 2 + App 3 + Consolidator


Eureka!!!
The first app makes an absurdity intelligeble
meaning
if you cannot do so
you don't have anything
The method for uncovering hidden meanings is
to find the presuppositions that make them intelliegeble
and
compare them with yours
When you understand both NoR(s), you have achieved a merging of horizons
That your universe just got bigger
Give it a month or so

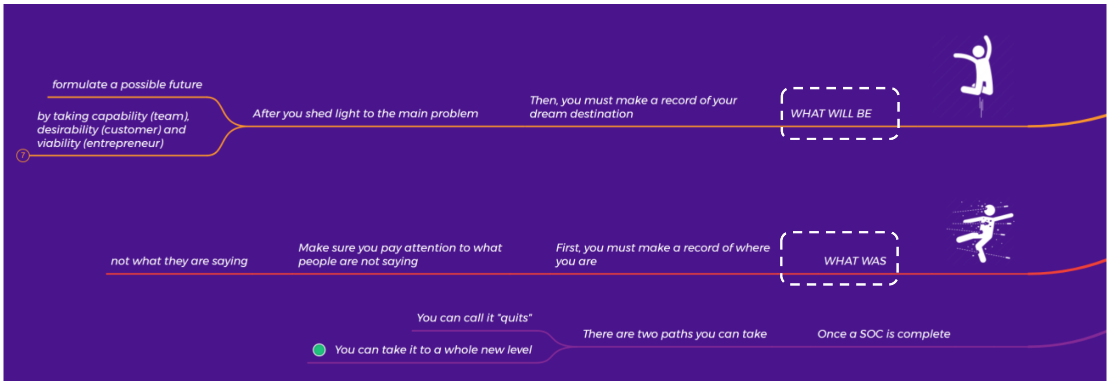
As an [experimenter] your first job is to come up with an app… fast. Do you know the anecdote about the dead fish? One day a consultant is summoned to fix a problem in the software development department of a big company. He immediately sees where the dead fishes are. However, it is vital that he not only finds the murderer but also devises a strategy to prevent such issues in the future. Otherwise, he will get the blame for it, "What! Why did you kill the fish!"


The remaining apps must support your cause
by
evidence
It mustn't be you who take cares of your wild schemes
Your flock should do it for you
Find out how
and you have just uncovered the second app's formula
public view
When you have the beginnings of a following
You must make it bigger and bigger
until one day
you become unimportant
Find out how
and you have just uncovered the third app's formula
Give it two more months

Good. So, you have formulated an app that makes absurdity intelligible. Now, it's time to support the feelings associated with it. While there may be other options, the best way to go about it is first, exposing clues and second, gathering public support for the cause. These may easily be handled by apps two and three in the trinity of interlinked apps.


This is where patience comes into play
This is a one-page app that only displays the most critical information
while the consolidator formula may be obvious, it often requires revisions here and there
Answer how apps 1, 2 and 3 relate to each other… how the whole is bigger than the sum of its parts
And you have just found the consolidator formula

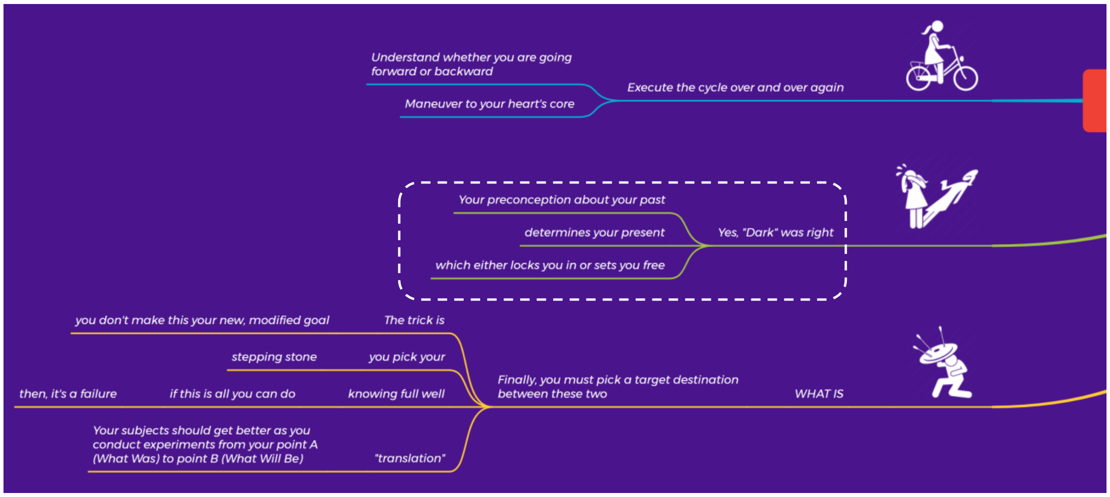
The final step in your app formulations is finding a way to see how those three interlinked apps generate critical value. This will be the consolidator that brings valuable findings into public eye to create… to mitigate cultural change.


formulate a possible future
by taking capability (team), desirability (customer) and viability (entrepreneur)
After you shed light to the main problem
Then, you must make a record of your dream destination
WHAT WILL BE
not what they are saying
Make sure you pay attention to what people are not saying
First, you must make a record of where you are
WHAT WAS
You can call it "quits"
You can take it to a whole new level
There are two paths you can take
Once a SOC is complete

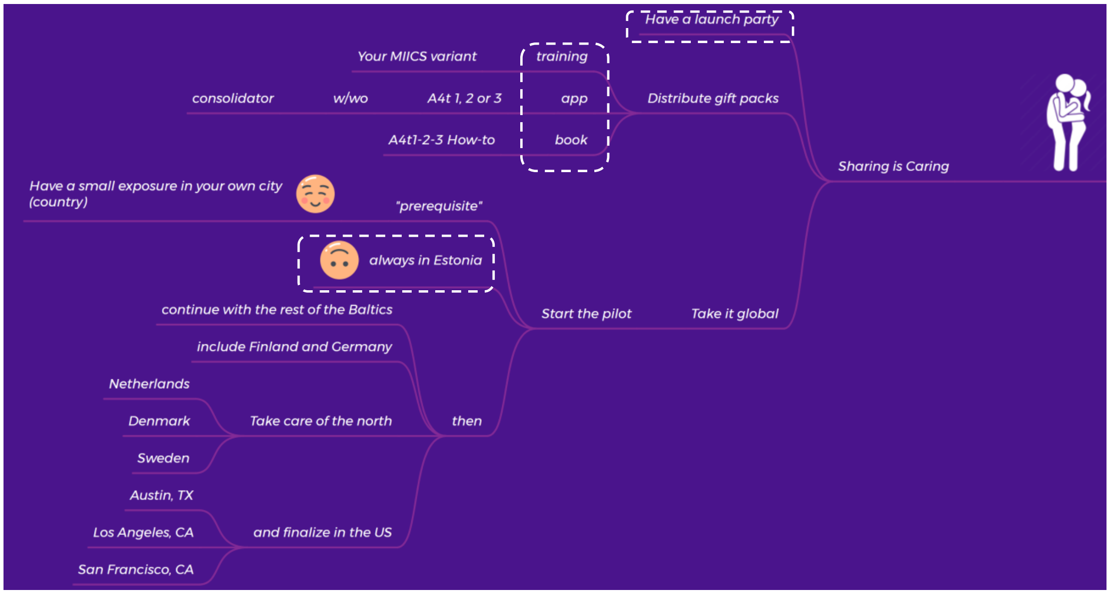
This is a critical juncture. You can stop. Actually, you could have stopped after formulating the first app making it a solitary one… but you didn't. Good. So, why stop now?

- *First, stop lying to yourself. Describe how things are going truthfully… and turn it into a milestone.*
- *Then, imagine that you can have a wish… any wish about your situation. Turn it into a milestone.*
- *This is the tricky part. Contemplate what is possible. Pick a few steps beyond that. Understand that this is not a goal. This is your failure point meaning if this is all you can do. It only means that you have failed. You are picking such an intermediary step, because Rome wasn't built in one day. Also, your flock needs to experience a [precedent], something which was deemed impossible to go even further the next time.*

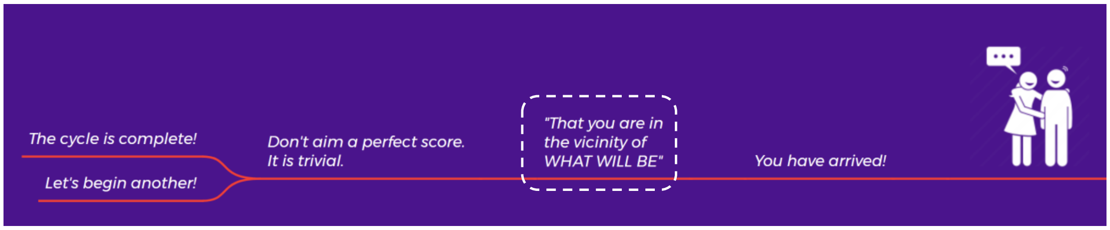
"Often, something much lesser in value is projected to the unsuspecting crowd as of high value. Beware of the imposters. Pseudomorphism is not your friend. It's your way to mediocrity and ignorance."


Understand whether you are going forward or backward
Maneuver to your heart's core
Execute the cycle over and over again
Your preconception about your past
determines your present
which either locks you in or sets you free
Yes, "Dark" was right
you don't make this your new, modified goal
The trick is
stepping stone
you pick your
then, it's a failure
if this is all you can do
knowing full well
Finally, you must pick a target destination between these two
WHAT IS
Your subjects should get better as you conduct experiments from your point A (What Was) to point B (What Will Be)
"translation"

This may not be easy to get, but only if you get it, will you understand the true nature of terraforming or cultural evolution for that matter. Your preconception about your past is directly linked to your future while you may be stuck or unstuck in the present. Confused yet? You are the one who is opening those possible futures regardless of where you are… or who you think you are. Another possible interpretation is "There is no such thing as present just like there is no such thing as a solitary / meaningful / capable self. The self becomes something only when she leaves herself behind."



Of course, you can just post your GitHub URL, but where is the fun in that? Your launch is your viral. It helps. It's like working at home… and when you do go to the office, you catch a virus or something. Launches where you spread the disease. Picking a launch location is tricky. Since what we are doing is weird… we are considered weird… and that doesn't help at all. So, often I disguise myself within keyword searching crowds where simulation or experiment may mean eighty million different things. Just before they figure out who the heck we are, we finish our show and head back home. And that's how I like it. It's best to distribute your gift pack at the end of the launch, but shit happens. Just make sure everybody can easily remember the gift pack URL. I use "tinyurl" and didn't have any problems so far. Finally, when you make the pilot global, always… and I can't stress this more… always start with Estonia. That whole country is like Silicon Valley. Tallinn, for example, is a testing ground for all kinds of crazy ideas. That's why after a short (read private) pilot where I live at the time, I always go to Tallinn and start the 'real' pilot there. You should do the same.


The cycle is complete!
Let's begin another!
Don't aim a perfect score. It is trivial.
"That you are in the vicinity of WHAT WILL BE"
You have arrived!

This depends on the circumstances, but usually a complete pilot cycle takes about anywhere from three months to one year. If you are lucky, you 'visit' all the countries listed previously during the course of a year, spending a month or so in each one of them. If the going gets though, you must at least finish the Baltic Invasion (Estonia, Latvia, Lithuania). The second and the third countries are on the southern side… meaning they have some sunlight. So, they add variety to the psychic personae of the Baltics… which is something you will need, if you aren't in love with the bitter cold. Also, you can finish in two ways regardless of the location or the length of the pilot. You can finish prematurely, meaning your target ecosystems are nowhere near their terraforming goals. And you can finish gracefully, meaning at least some of your ecosystems are very near their terraforming goals.

A4t 1

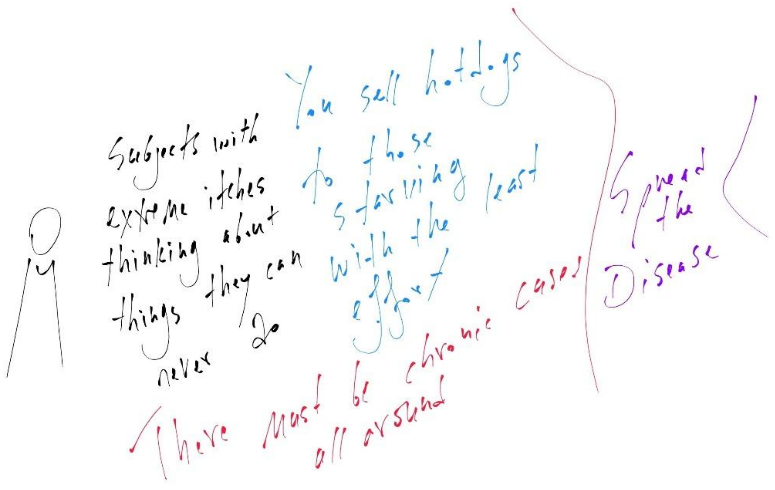
It's Alive!

Until the moment your app goes live, everything is just hogwash even if you have the most brilliant ideas. None of it matters unless your ideas reach those who need it in the form of a cross platform mobile app. Just like the first moment you set your eyes on the love of your life, the first days, first weeks of an app launch is crucial. It can live or die depending on the effect it creates during such a short time. So, we must pick the first population (interesting issue or ecosystem) very carefully. The best example is an anecdote I never forget. Robert Allen is giving a speech at a university. He asks the students what they would do if they wanted to open a hotdog joint and turn it into a success story.

A student says, "I would make the best hotdog available there". Another one says, "I would make the joint entertaining or relaxing depending on the sensibilities there". Yet another says, "I would hire the cutest waitresses and give free desserts".

Robert smiles at the proposals thrown at him, "Well, all these may have different kinds of effects, but the answer is so simple. You should find people starving and sell them any hotdog".

We will follow Robert's lead here. We are not after ideal this or ideal that. We are looking for the first fish that will take an interest in our bait. Once we achieve that, we will have a better understanding of that ecosystem and this will help us refine our ideas to catch more and more fish.


Subjects with extreme itches thinking about things they can never do
You sell hotdogs to those starving with the least effort
There must be chronic cases all around
Spread the Disease

So, sit tight and don't make sudden moves. Just make sure your bait is still there, looking attractive. Your subjects will be revealed to you.

It's not any different than spotting a crackhead, loose woman or powerless manager… you know, someone dying to unsee her realities… just to feel good.

Of course, we must understand their point A(s) and B(s) too… because all exploitation take place in-between these two.

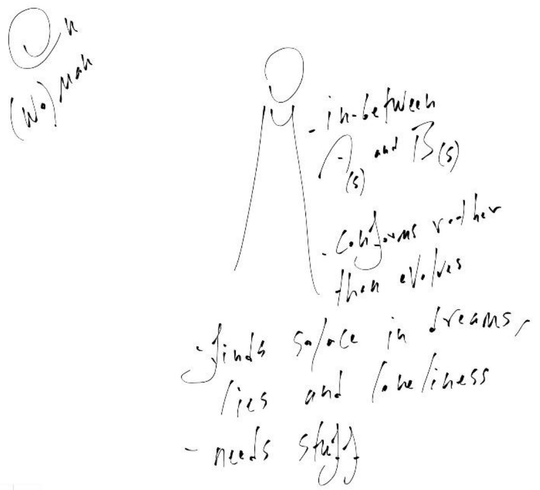
When we see the relationships between a person's points A and B, we can… what's the polite word for this… well, I can't think of one… we can exploit her... control her. When her condition is combined with today's information driven world, it produces "programmable humans" without whom none of this would be possible. So, thank you diminished humans!

Everything we will be able to do circles around programmable humans. There are three types as you may recall, "subjects", "troublemakers" and "enablers". They have a few, but critical differences. [Subjects] are locked into their ecosystems. Yes, they can move from one to another, but they are all variations of the same theme, ecosystems which are similar to the ones that came before them. They can never really leave. [Troublemakers] are the ones who can go through the exit, not just spend their whole lives looking at it. They know the difference between knowing the path and walking the path. [Subjects], however, don't know that. Their whole lives revolve around "not knowing". They may have the capacity to know it, but because of their cowardice, they do their best to unsee things, unknow things, undo things. [Enablers] are just like [subjects]. The only difference is they are in other ecosystems. So, they can freely talk about an ecosystem which is pretty much like theirs, but because they don't report to the power holders there, they can be honest about the lives of those which have no immediate effect on theirs. This makes them seem OK which they are not.

Understanding [subjects] beyond the scope of the current ecosystem is crucial. You must note everything you learn about them for further analysis. You must consolidate your findings from one project to the others, for [subjects] are all but one species. Everything you learn from one group of [subjects] is applicable to the ones in other groups.

Perhaps, I must make the concept of the [subjects] more visual, so that you will never miss one when you set your eyes on him.

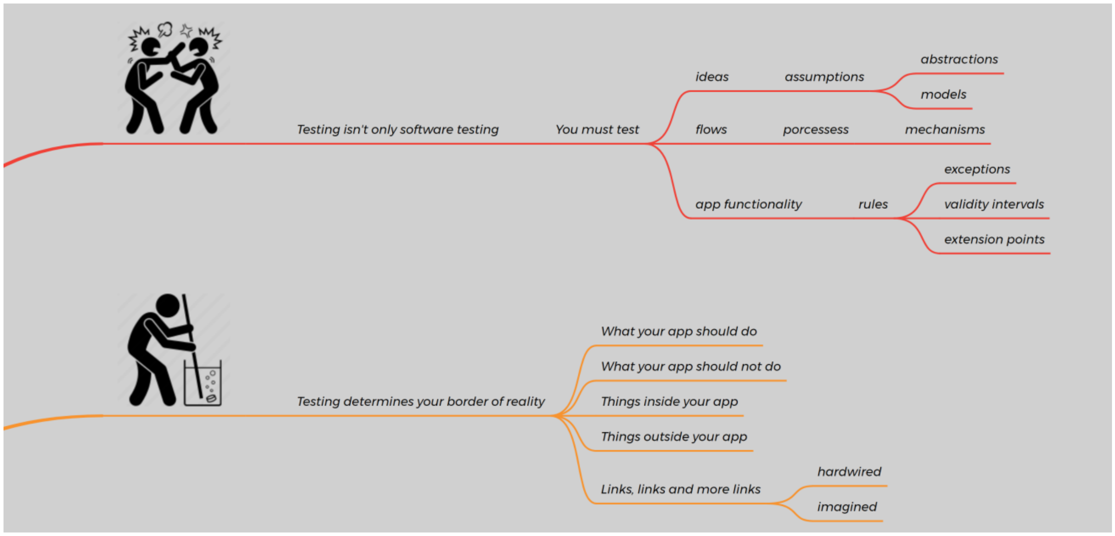
Do you like Westerns? I do. I just love them. To me they provide a summary of the human condition. They make character traits so striking, you can never miss what they really mean. This helps you in real life too, because lies are the foundation of human existence.



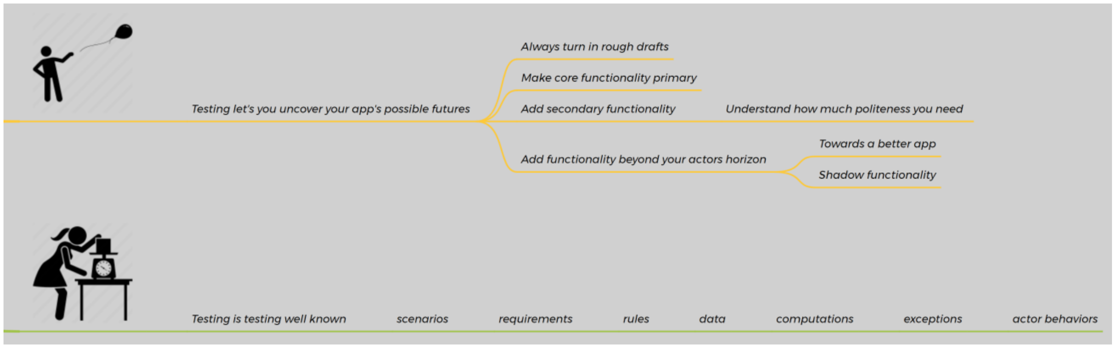
(Wo)man
- in-between A(s) and B(s)
- conforms rather than evolves
- finds solace in dreams, lies and loneliness
- needs stuff

Rope, Alfred Hitchcock, 1948.

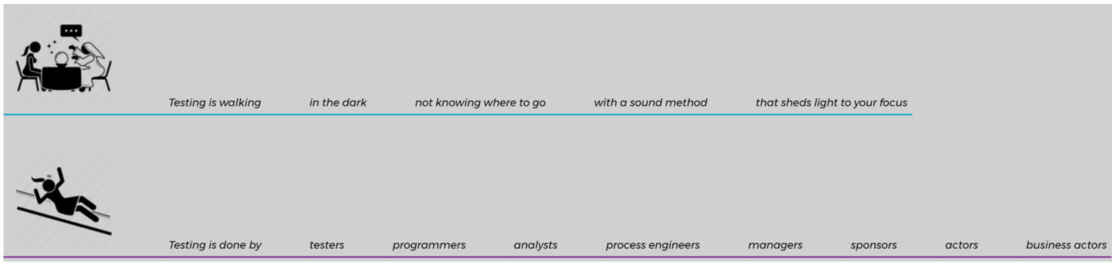
A human being may behave in an honorable way once or twice, but he is a parasite regardless of how competent he is at hiding it. I consider myself as one too which always helps me keep myself on the right path. I'm never fooled by lies, because I'm a man without point B(s). You can never fool someone who seeks nothing. This is how you should approach ecosystems too. In other words, this is the ability that promotes an [troublemaker] to be an [experimenter]. If I have to say it twice, you must never ever go to ecosystems where you seek things. Because you cease to be an experimenter in them. You too become a subject to be experimented on. You have inclinations that can be exploited. People who can see you for what you are will do far worse things than conducting experiments on you. So, watch it, pal.



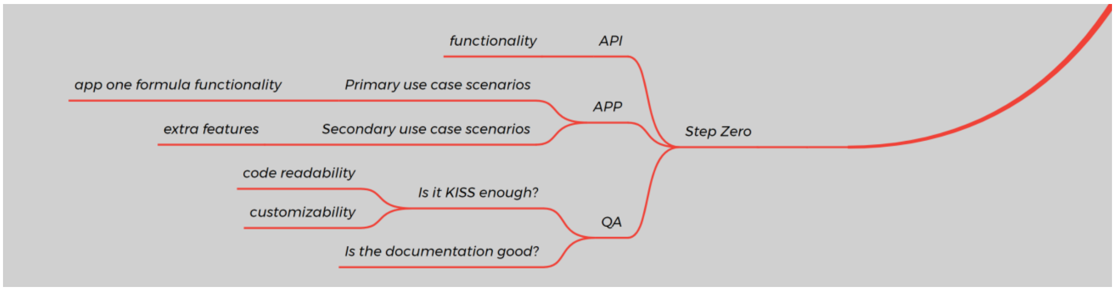
If you want to be able to trust your test results, you shouldn't confine yourself within the borders of your apps. Anything that links to your apps directly or indirectly is testable, nay, they must be tested, if you seek "control".


Testing let's you uncover your app's possible futures
Always turn in rough drafts
Make core functionality primary
Add secondary functionality
Understand how much politeness you need
Add functionality beyond your actors horizon
Towards a better app
Shadow functionality
Testing is testing well known
scenarios
requirements
rules
data
computations
exceptions
actor behaviors

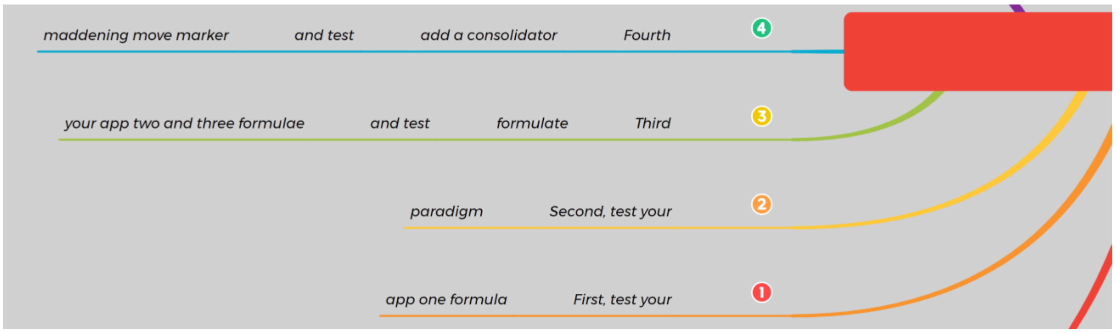
Testing takes place in two parallel realms, what is known and unknown. While known things represent your "today", unknown things are where your "tomorrow" lies. This is why you test what's been done and what can be done together.



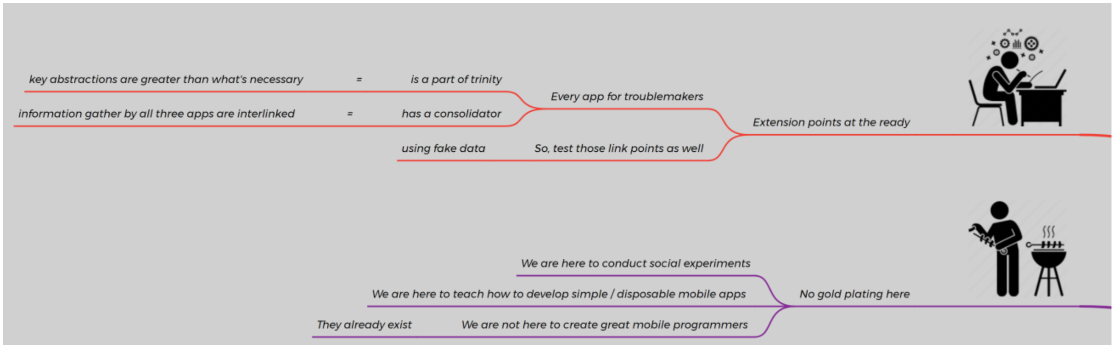
While testing is for understanding the nature of things, whether everything is where it is supposed to be, your other capabilities like business modeling, analysis, process engineering, development or product management are there to find the sweet spots you seek. Think about testing as your eyes and all the other capabilities as the rest of your organs.


functionality
API
app one formula functionality
Primary use case scenarios
APP
extra features
Secondary uise case scenarios
Step Zero
code readability
Is it KISS enough?
customizability
QA
Is the documentation good?

So, "what is known?" The requirements of our apps is one of the things in the what's known basket, if it isn't just a figment of our imagination. This is why the work of analysts is critical. On the other hand, often analysts aren't that good. That's why so many software projects fail, not because of the glitches in technology. And yes, there are more important things behind requirements. Ideas, principles and concepts drive requirements. How you communicate it with other people dictate the quality of your products, processes, culture… lives (Mel Conway).

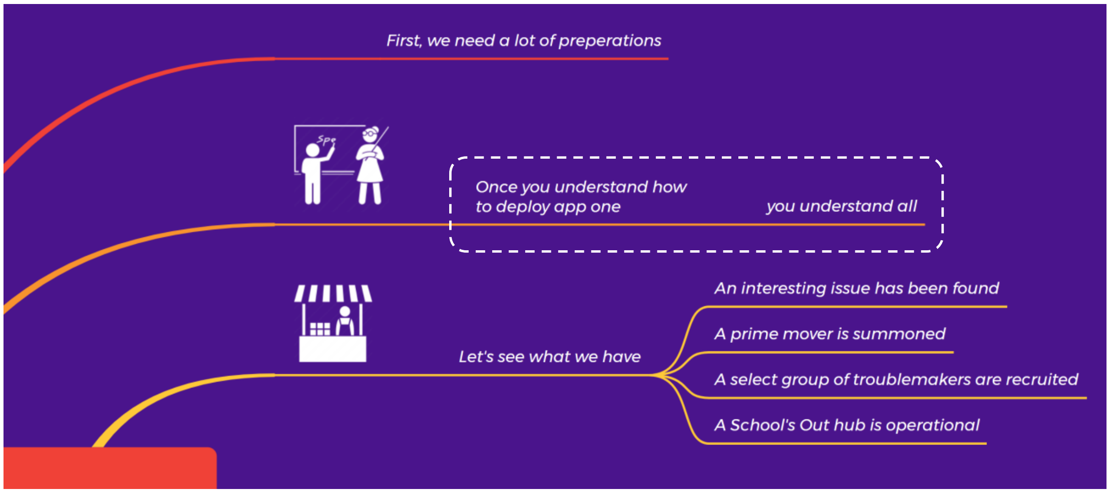


"What is unknown?" All your destinations are! Where we want to be is yet another unknown, because it is a dream. Where we should really go will unfold little by little as we take further steps into the dark. So, your plan is a series of well thought of steps which can be modified as necessary. The idea of the destination should never change, though. The place where our idea (ideal) becomes a reality is what we are after. It's something we cannot name just yet.

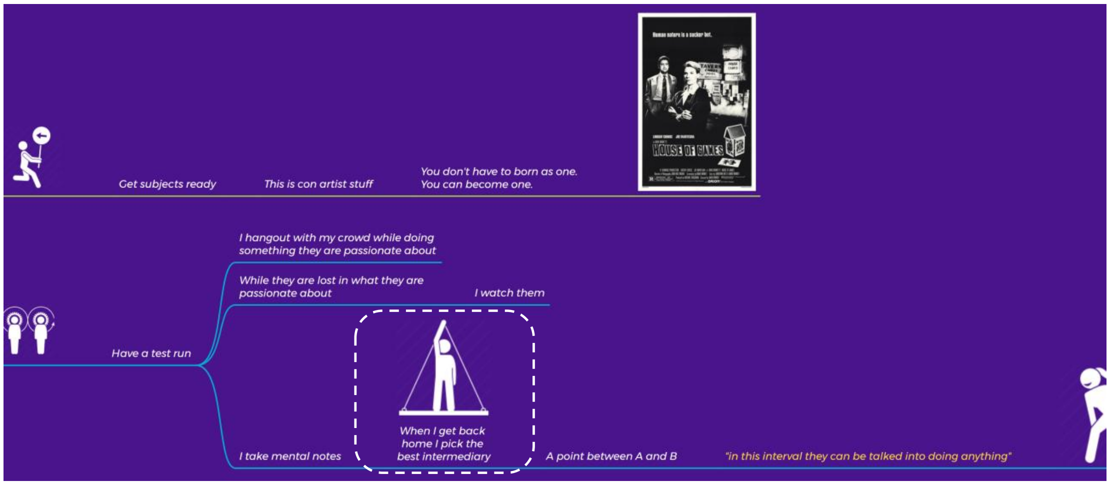

key abstractions are greater than what's necessary
=
is a part of trinity
information gather by all three apps are interlinked
=
has a consolidator
Every app for troublemakers
using fake data
So, test those link points as well
Extension points at the ready
We are here to conduct social experiments
We are here to teach how to develop simple / disposable mobile apps
They already exist
We are not here to create great mobile programmers
No gold plating here

Don't lose yourself in artifacts or deliverables. When you take care of the swamp, you will be taking care of the people. Focusing too much on artifacts, deliverables, numbers or personae will break you. Take care of the people and they will take care of all the rest including you. If people become diminished, everything becomes diminished. Don't forget that.

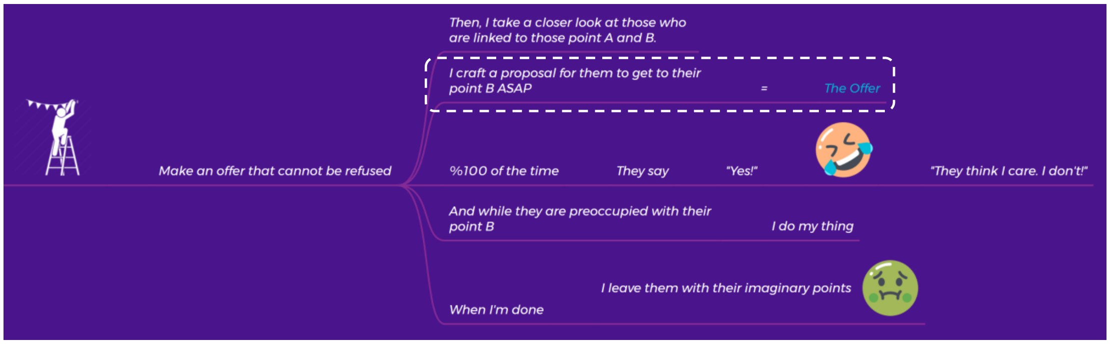
Deployment Games


First, we need a lot of preperations
Once you understand how to deploy app one
you understand all
Let's see what we have
An interesting issue has been found
A prime mover is summoned
A select group of troublemakers are recruited
A School's Out hub is operational

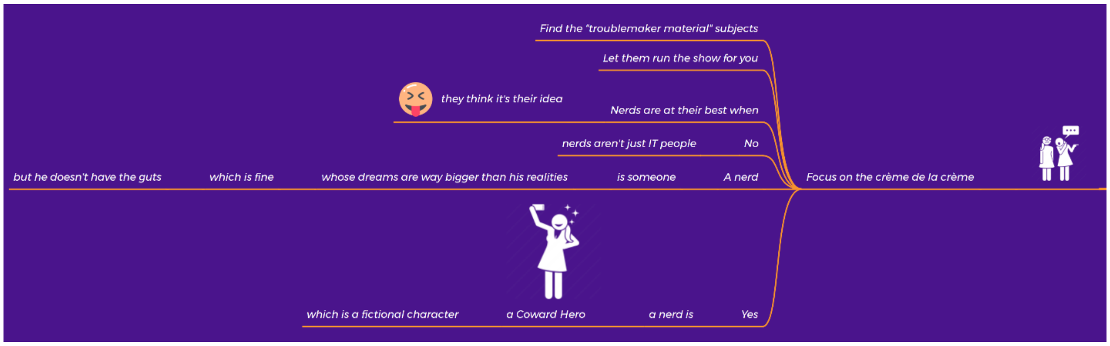
In this game your first move is what makes or breaks it. If you cannot make the absurdity intelligible, you have nothing. When you can do that, all there is to be done is to formulate an app that makes it obvious. For example, in Moody employees who don't take the necessary measures use it to complain about everything until they drive management crazy. By doing so, they must be releasing endorphins, because bitching makes them very happy. It gives them the illusion of being right, thinking everybody else is wrong. That by speaking about things or dreaming about them is living. That they are not cowards. That they are the smart / good ones… which is horseshit, of course.


Get subjects ready
This is con artist stuff
You don't have to born as one.
You can become one.
I hangout with my crowd while doing something they are passionate about
While they are lost in what they are passionate about
I watch them
Have a test run
When I get back home I pick the best intermediary
I take mental notes
A point between A and B
"in this interval they can be talked into doing anything"

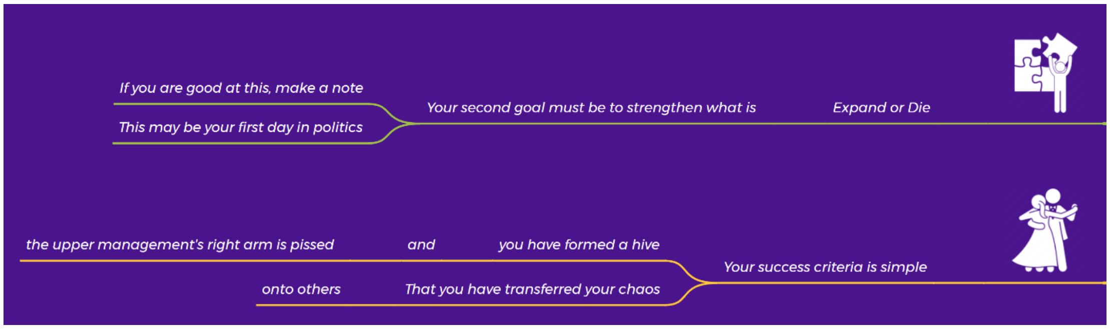
Another skill you should develop is understanding what people want and how that contrasts with how they behave. Understand that you can fool those with strong emotions towards desired objects or strong beliefs in themselves regardless of how it is justified in their minds. Once you find people like that, all you have to do is to uncover their point B(s). Since their point A(s) is rather straightforward, granted your eyes and ears are open. Of course, these points only reveal their true nature to you if you look at them within the context of diminished human beings. To be able to prefer hay to gold, you must be an ass too. Yes, it's a sad story, but when you are faced with normal human beings, app concepts applicable to them are rather boring. Thankfully, this is a very rare thing these days. When I see a group of people with strong feelings or beliefs, it always puts a big smile on my face. I do enjoy bad news like George Carlin.


Then, I take a closer look at those who are linked to those point A and B.
I craft a proposal for them to get to their point B ASAP
=
The Offer
Make an offer that cannot be refused
%100 of the time
They say
"Yes!"
"They think I care. I don't!"
And while they are preoccupied with their point B
I do my thing
I leave them with their imaginary points
When I'm done

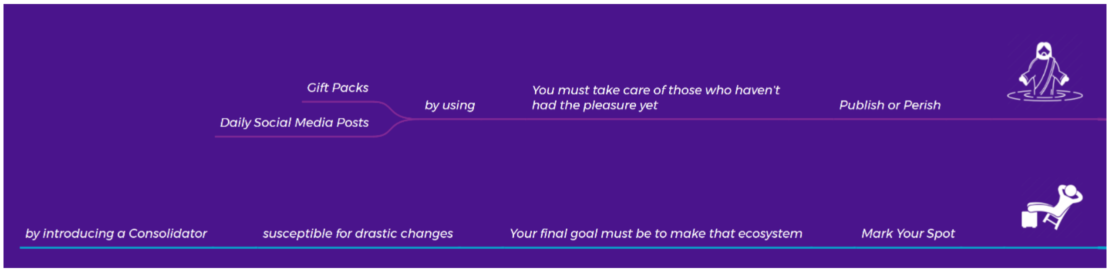
The good thing is their Point B(s) are so important to them, that they are blind to all else gives you opportunities.


Find the "troublemaker material" subjects
Let them run the show for you
they think it's their idea
Nerds are at their best when
nerds aren't just IT people
No
but he doesn't have the guts
which is fine
whose dreams are way bigger than his realities
is someone
A nerd
Focus on the crème de la crème
which is a fictional character
a Coward Hero
a nerd is
Yes

Pick the unbalanced ones, lots of talk but no walk. Cowards thinking they are heroes are your bread and butter.


If you are good at this, make a note
This may be your first day in politics
Your second goal must be to strengthen what is
Expand or Die
the upper management's right arm is pissed
and
you have formed a hive
onto others
That you have transferred your chaos
Your success criteria is simple

Before you get out of that cowboy town, so to speak, leave a seed behind and set your eyes for similar ecosystems.


Gift Packs
Daily Social Media Posts
by using
You must take care of those who haven't had the pleasure yet
Publish or Perish
by introducing a Consolidator
susceptible for drastic changes
Your final goal must be to make that ecosystem
Mark Your Spot

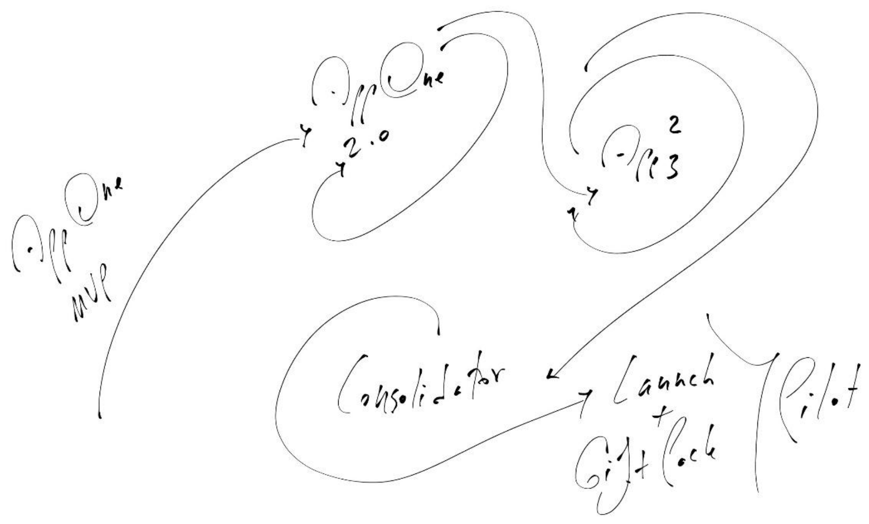
Your seed receives the final gift which is a consolidator of interlinked apps. Let everybody know about the experience.

A4t 1

Judgement Day

Let's face it. You have embarked on a journey where gold plating is the greatest sin and having the ability to maneuver at a moment's notice is the greatest achievement. That's why we have boundaries called big pictures and plans called waypoints with options. Being a computer scientist means there is a way to dance in the darkness, go to places you didn't know existed and commune with friends you haven't met yet… experience new possibilities of existence.

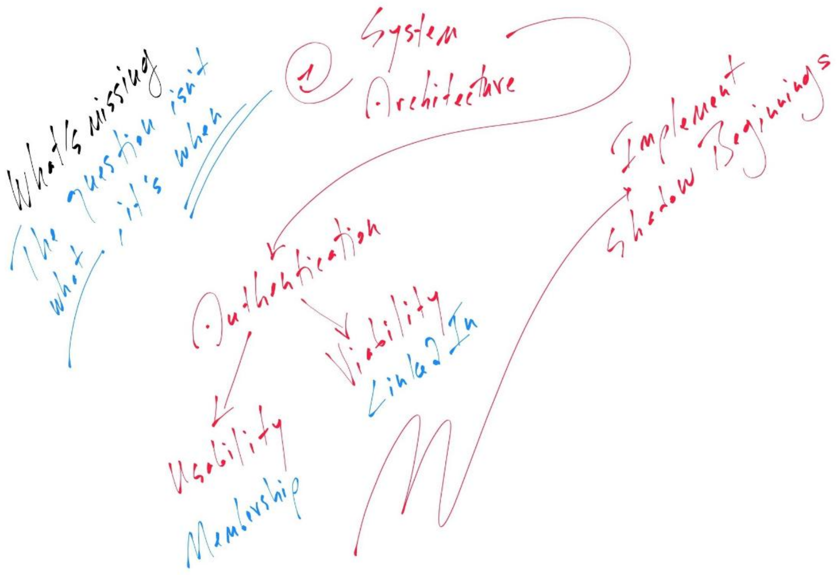
That's why people never get me when I talk. Because what I want, what I think, what I say and what I do seem disjointed to them. While they are all kind of loose, they seem that way because I am what ties them together. When I change, they change. When they change, I may change, but then again, I don't have to. That's your path too, troublemaker.

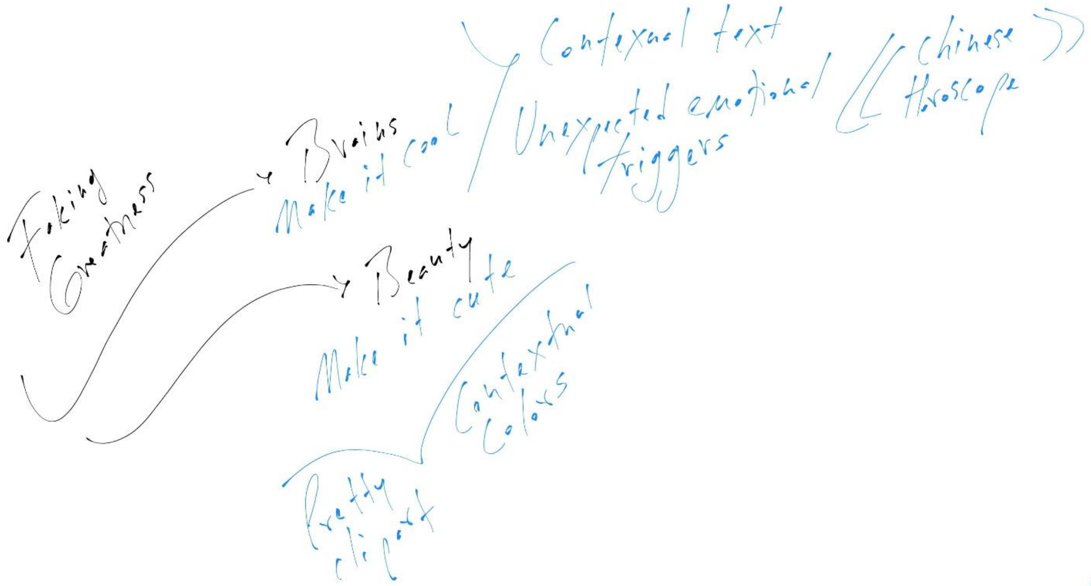


We must first answer the question, "what is enough" … to start, stop, pause or resume our activities? If you can exploit your [subjects] in their pursuits from their point A(s) to point B(s) with little or no resistance, that they are turning into fuel for your future activities without much effort, you have arrived.

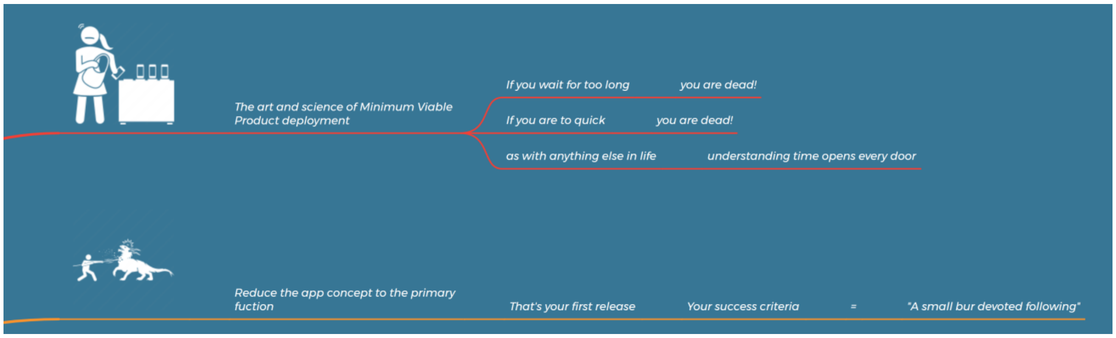
Think about it like a pimp or a drug dealer. Not much conversation takes place there, right? That's because the merchandise speaks for itself. The product sells itself. Any intermediaries are there to take advantage of something your provider is taking advantage of… you know, like a series of tightly linked exploitations. And like every good deal, both parties should think that they are the ones really exploiting the other. Of course, in such a world taking advantage of others would be seen as business, something quite normal, even a quality standard of some sort.

In this first version of the Moody several features will be missing. Because they don't mean much when you are learning how to develop mobile apps from scratch... how to conduct social experiments using apps from scratch. They are secondary features at best. The most important missing feature is "authentication" which will be handled by a 3rd party system, "LinkedIn". Without that feature, you cannot go live. Relax! I said, "it's not important now".

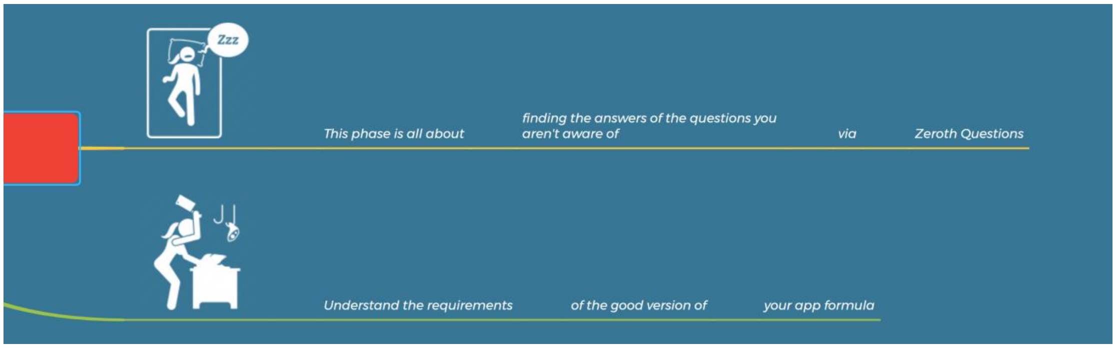




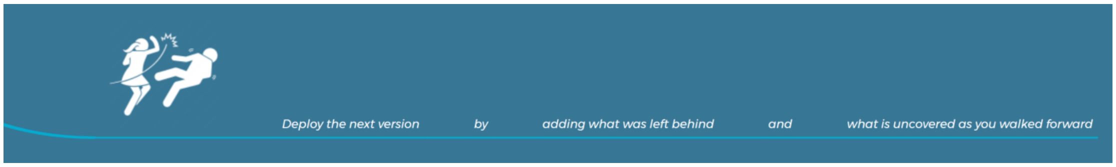
Other missing features have to do with software politeness and esthetics. We won't strengthen the machine-human bond too much. Also, we will live with a somewhat ugly app. I mean, the real beauty lies on the inside. Am I wrong?


The art and science of Minimum Viable Product deployment
If you wait for too long
you are dead!
If you are to quick
you are dead!
as with anything else in life
understanding time opens every door
Reduce the app concept to the primary fuction
That's your first release
Your success criteria
=
"A small bur devoted following"

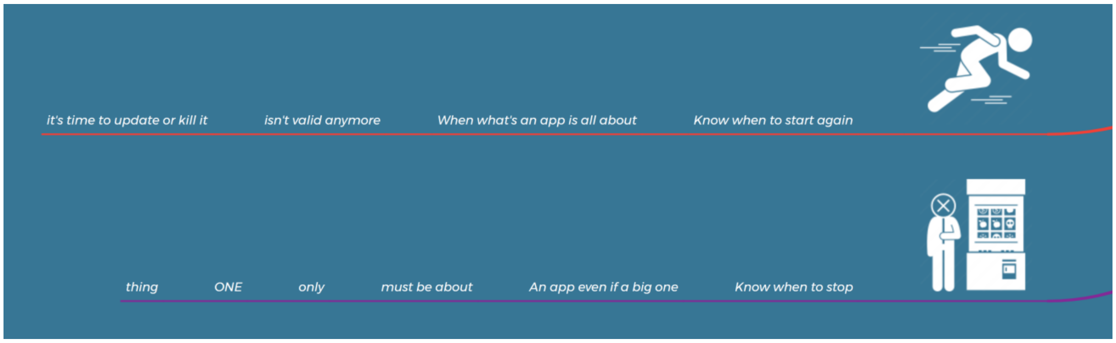
The most important skill you can learn is "knowing when to do something and when not to do something". The simplest way to go about it is picking the first functionality (achievement) your app grants you and sticking with it till the end without making it bigger or smaller. For example, when you say, "I must be able to transfer money to another person", this is what you must focus on, not possible secondary goals which may exist at the same time.


Zzz
This phase is all about
finding the answers of the questions you aren't aware of
via
Zeroth Questions
Understand the requirements
of the good version of
your app formula

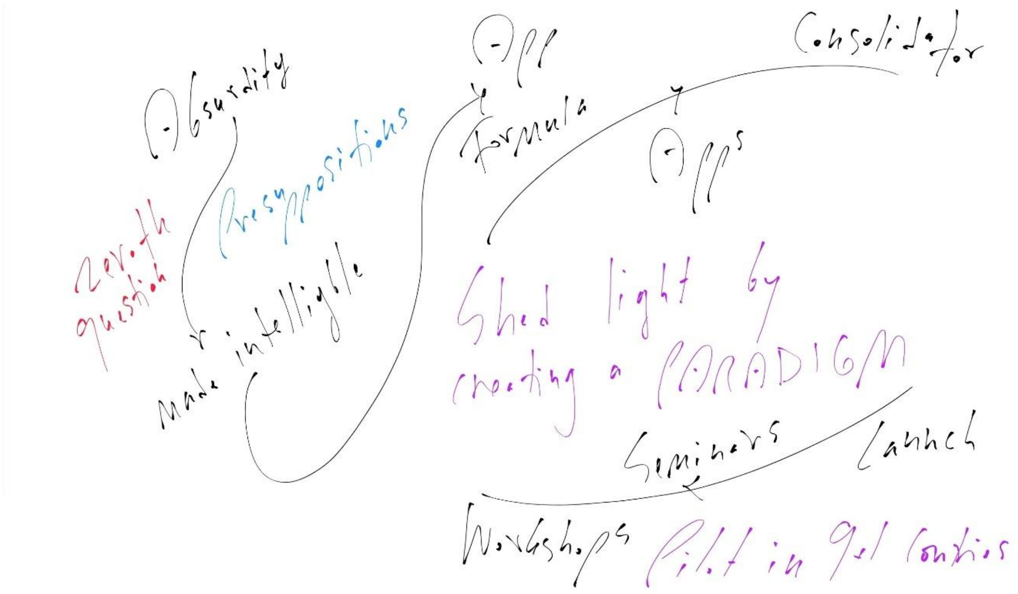
Since we focus on absurdities instead of how-to-ideas, we can't be easily fooled. In our efforts to make the absurdity we face intelligible we are forced to formulate zeroth questions. Remember, a zeroth question is the question that cannot be asked within the border of reality of the absurdity… where the absurdity doesn't seem absurd to anyone. It is the responsibility of the [experimenter] to form formulate such a question… which exposes the presuppositions that make the absurdity intelligible. These questions further help us formulate our apps. So, once discovered, stick with that app formula. Flesh it out by attaching building blocks of software development (i.e. actors, use cases, scenarios, etc.) to it until you have the application version of the answer.


Deploy the next version
by
adding what was left behind
and
what is uncovered as you walked forward

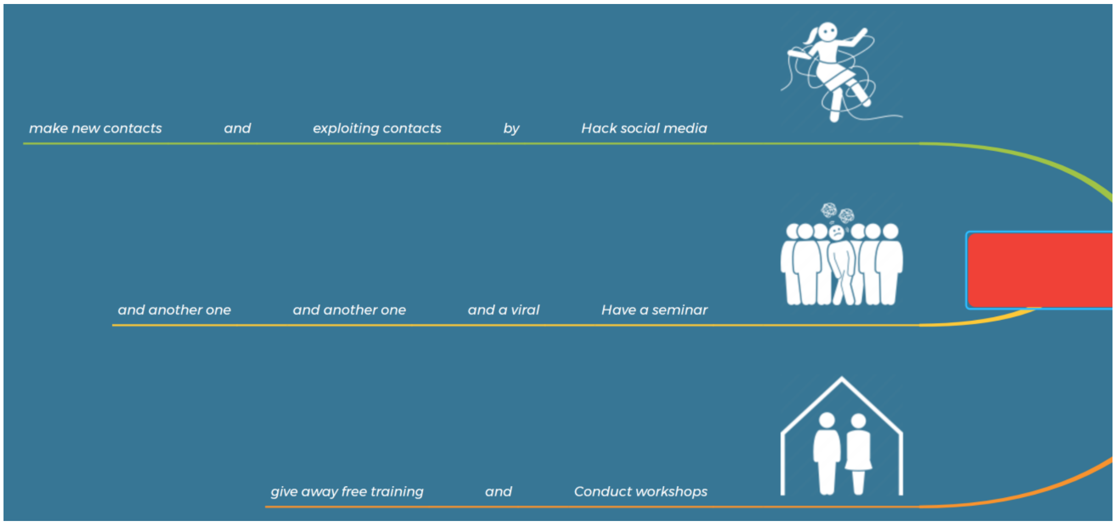
App version of the answer has software development artifacts merged with what it helps you uncover… as you are developing and deploying versions of it. The most interesting aspect of your work is emergent… meaning you will find the do-or-die features of your app on the way. You cannot go ahead and create it on day one. You cannot know it then. It will reveal itself to you as you perform activities she (?) deems worthy of her attention. "Truth is shy. If you chase her, you will never have her." Just do your best and she may claim you hers.



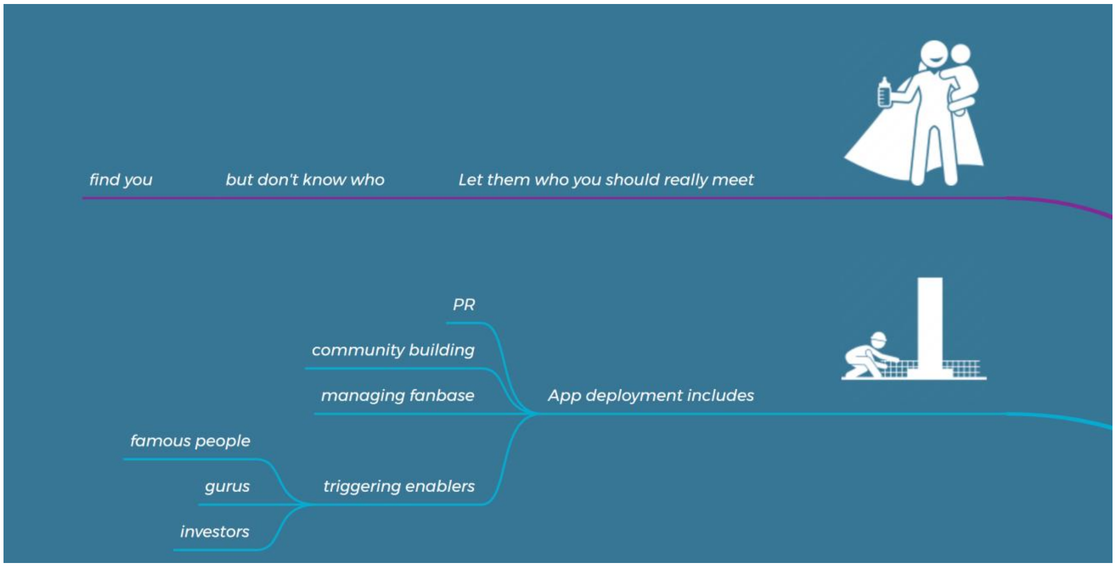
In time we will be adding new functionality and visual components to our apps. Be careful though, continuously adding or subtracting things from an app may delude us into thinking those apps are immortal. Every app must die one day. The best way not to miss this is to keep the app focused, performing one thing very good and not turning it into a mixed bag.



When you have an MVP ready, you must plan how it gets into contact with other people. You cannot just throw a party and expect it to become a party animal. While your experiment grows, starting from a single app and ending in three interlinked apps and a consolidator, your deployment strategy must start with an MVP, continue with a launch, develop more so with seminars and workshops, and only end with pilots in 9+1 countries.



As you can see, I think about everything in terms of their relationships with other things. This is what sets our way of thinking apart. We aren't concerned whether the glass is half full or half empty. We are concerned whether the glass is isolated from others or the glass has friends.



A strange habit, I know. I'm an expert when it comes to contacting friends I do not know! When you are an inquisitive one whose interests have no bounds, sooner or later you outgrow your ecosystem. So, little by little I have perfected the art of using messages in bottles. My success criteria are high. I think, even if someone finds out what I have done long after I kicked the bucket, she should be able to use it without instructions. Those who are gifted need not to be joined organically. Time does not limit us. That's the human being's true gift. That's what separates us from other mutations. That's why we are worth it even if most of the things we do are shitty.

Extension Schemes

I don't know about you, but I just hate the endless releases companies force upon us these days. It's the results of many things including the shitty agile frameworks. It is considered a job well done. Well, a job well done doesn't have to be updated every few weeks now, would it?

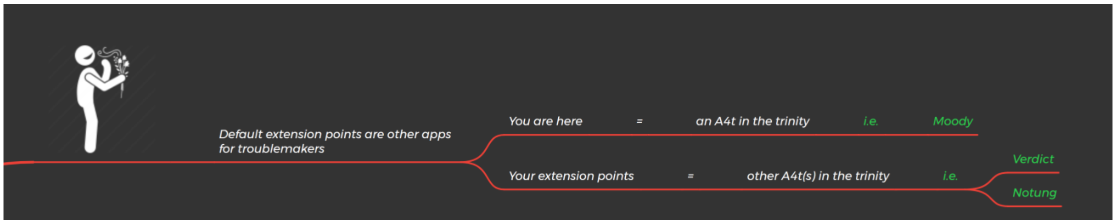
What do I mean by that? Your extension schemes shouldn't be feature centric. Of course, there may be crucial features you have missed or didn't have the time to include… or maybe they can only useful after a certain milestone is reached… regardless, when you think of extensions, you should think about small steps. Every step should account for something remarkable. That doesn't mean that they have to be big. Again, think about diamond earrings. They are small, but the lady (?) who is on the receiving end would never complain, right?

Testing Deployment Strategy Growth X are one and the same

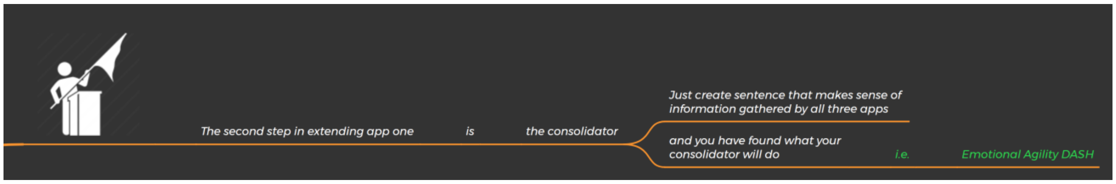
Again, never make the mistake of being a template zombie. Everything you do about an [interesting issue] a [traveler] has picked… is just one thing. Imagine, you are composing a concerto. And, yes, in this reality past-present-future are one thing too. Never forget that. You do something and the meaning of what you have done previously changes.

Appreciation of time is everything, but to be able to appreciate time…

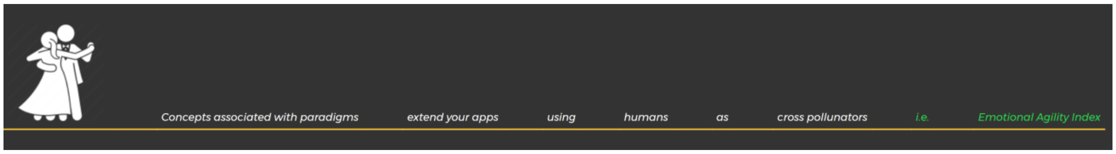
…you must develop an understanding of beings in relation to one another in time.


Default extension points are other apps for troublemakers
You are here
=
an A4t in the trinity
i.e.
Moody
Your extension points
=
other A4t(s) in the trinity
i.e.
Verdict
Notung

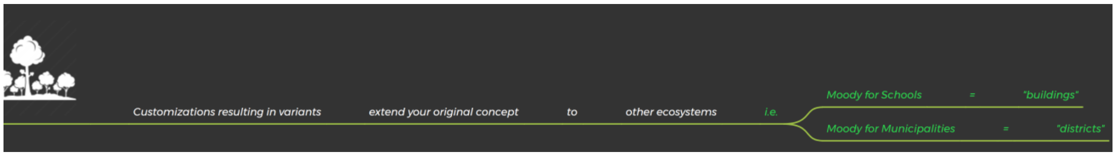
Your first extension moves should be the other apps in the trinity. Since I'm a man of models and processes, I tend to have rules regarding them. However, your next app can be any complimentary app that helps you further develop the idea behind your first app / social experiment.


The second step in extending app one
is
the consolidator
Just create sentence that makes sense of information gathered by all three apps
and you have found what your consolidator will do
i.e.
Emotional Agility DASH

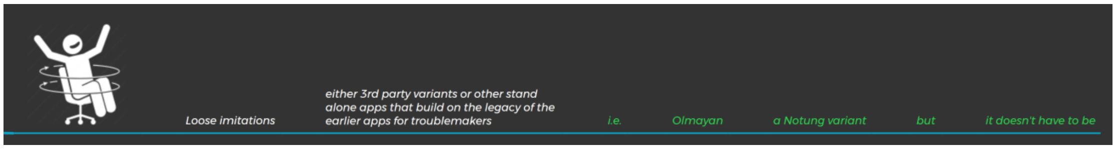
When you are content with the borders of your combination of related social experiments, you should start thinking about a concept that ties them all together. Oh, well, actually, you shouldn't work that linearly. Whatever I explain here in a linear fashion, I do it in a circular fashion. "Iterative" is the secret word here. Do everything over and over again in different intensities. What will guide us towards a meaningful end? Well, first you must imagine. Then, you must *understand. Later you must create. Next, you must test. When it seems you are finished, you must go to square one again. When the end is really great, it will stop you.*



The unifying concept you come up with must be reducible to a simple metric. Otherwise, you cannot figure out whether you are going forward or backward. Also, people just love empty promises in the form of buzzwords. It's the best way to lie. A metric which is a good lie… Figure it out.



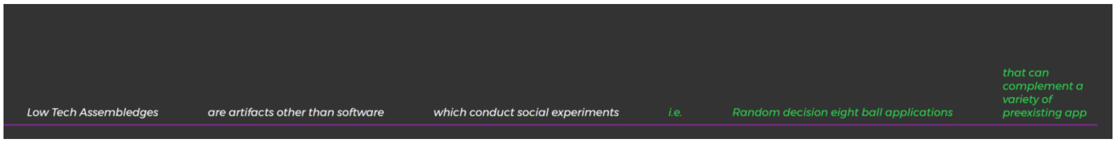
Developing tiny generic apps makes it possible to create numerous variants with little to no effort. Just change two lists (criteria, community) in Moody and you have a different app! Thinking in this manner makes one invincible. "If my assumptions turn out to be wrong, I can change the target ecosystem and the nature of the itch under a minute!"


Loose imitations
either 3rd party variants or other stand alone apps that build on the legacy of the earlier apps for troublemakers
i.e.
Olmayan
a Notung variant
but
it doesn't have to be

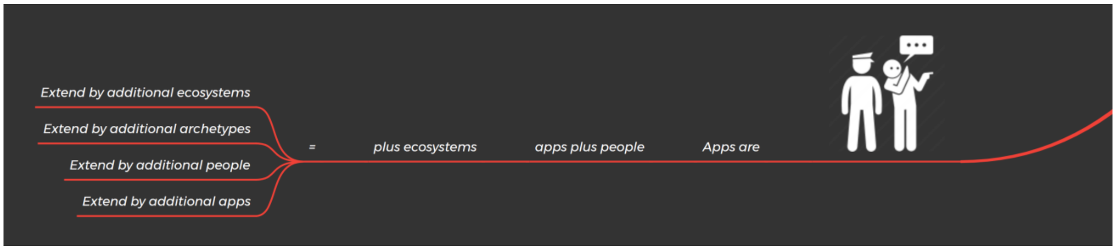
Why stop there? We can build upon the general concept of an experiment. Even if this new app may have nothing to do with the ones that came before them, it can be used to exploit the mental states of the subjects in that ecosystem. We are building coral reefs here, people!

We think in terms of four itches:

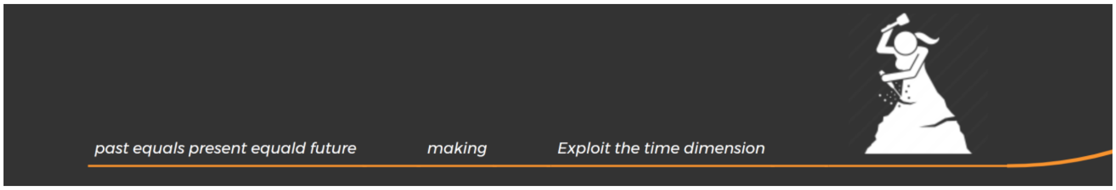
1) *Apps as psychological triggers and cultural disruptors.*
2) *Nature of ecosystems.*
3) *Lives of those involved with our apps.*
4) *Software engineering disciplines.*


Low Tech Assembledges
are artifacts other than software
which conduct social experiments
i.e.
Random decision eight ball applications
that can complement a variety of preexisting app

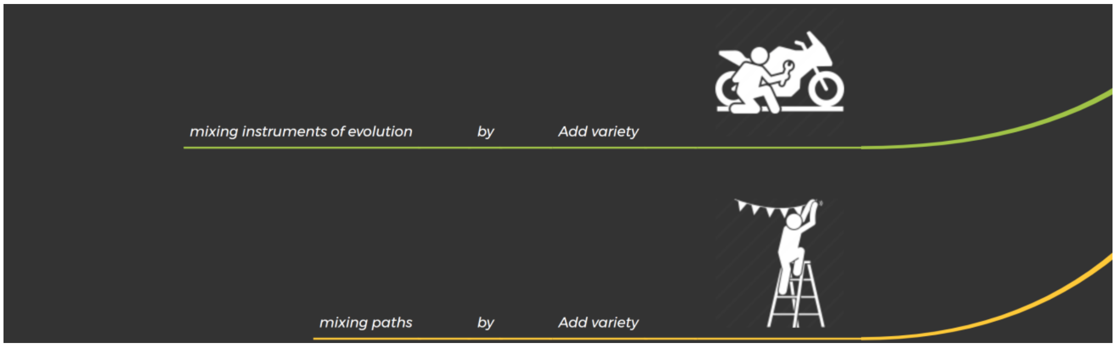
If you are living near the target ecosystem as a family experimenter, you can inject pranks into the overall experience. Haven't you heard the news? More chaos means more fun. People are so gullible. They make their circumstances worse by using outside help. I have seen so many smarter mice, let me tell you.


Extend by additional ecosystems
Extend by additional archetypes
Extend by additional people
Extend by additional apps
=
plus ecosystems
apps plus people
Apps are

Many people look at a person thinking about her. When I look at someone I think about the Milky Way and beyond. You must never look at rain while you are thinking about drops of water or water's chemical composition. If you do, you will miss the whole point. Worse, you will lose your way.



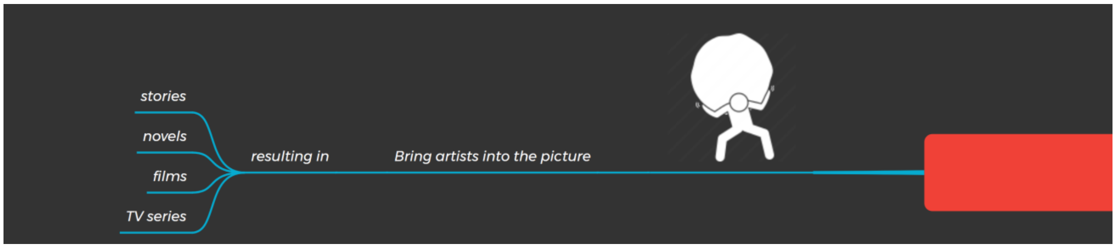
"So, how can the future change the past?", you say. Simple, it makes you change your perspective about the past. When your presuppositions change, what you perceive as reality changes. In other words, the past does not change. You do. "You cannot bend the spoon. Instead, you must accept the truth. That there is no spoon." (The Matrix, The Wachowskis, 1999). "That which moves is neither the branches nor the wind. It's your heart and mind." (A Bittersweet Life, Kim Jee-woon, 2005).



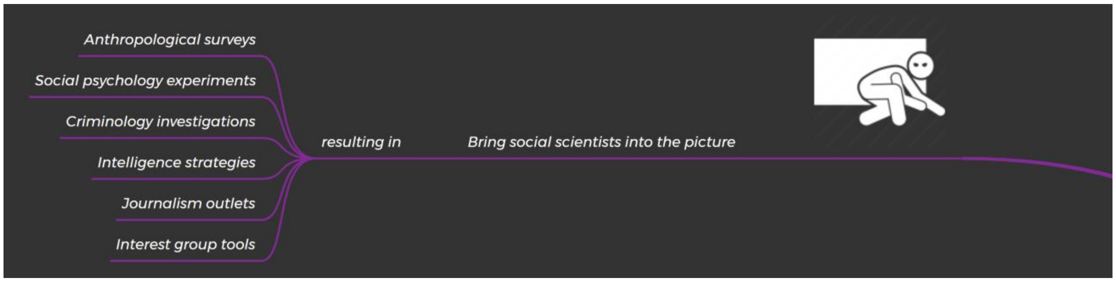
Apps for Troublemakers are artifacts of School's Out cycles just like experimenters. They don't have to be isolated from other NoRM components. You can add other evolution instruments like Brave New Worlds (app stories) into the mix. You can also merge evolution paths adding other ones like Media Producers into the mix, as explained below.

Evolution Instruments = {School's Out, Apps for Troublemakers, Maddening Moves, Brave New Worlds, Reality Check, Something from Nothing}

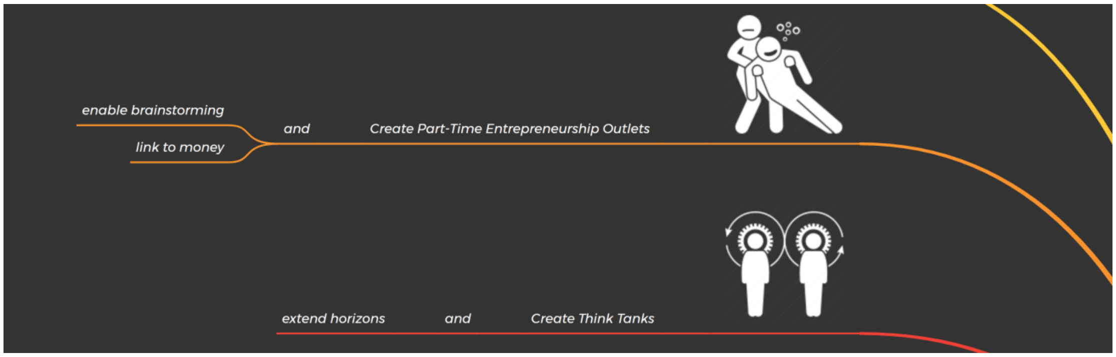
Evolution Paths = {School Alternates, Media Producers, Intelligence Agencies, Think Tanks, Interest Groups, Part-Time Entrepreneurship Outlets}



I'm a Promethean by nature and a Wagnerian by choice. Just like Wagner, I think everything can be merged to create perfect art. Apps can play central roles when creating stories… realities. Think of apps as films and TV Series. Whatever you watch can be something you do when you leave the theatre behind. Isn't it cool?



I feel like a Renaissance man, don't you too? Operating on a single dimension bores me to death. So, artists alone will not cut it. You must make all kinds of social scientists parts of this game. A (wo)man is a city in my book. And a city is a place with endless opportunities both good and bad.

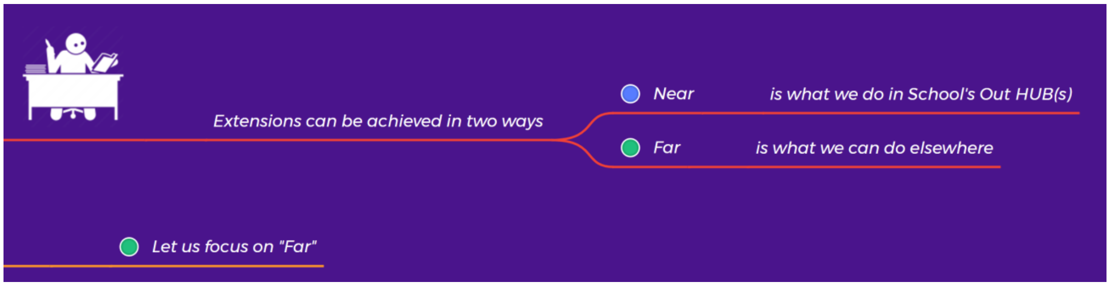


Braves aren't enough. You must include the cowards too. I mean, don't they make any story more fun. Imagine, rats leaving the sinking ship. Don't they add to the ambience? Yes, they do! So, you must have people with one foot here and one foot somewhere else. Your experiments shouldn't be too controlled. Otherwise, you will be condemned to live behind the borders of your reality… where nothing is absurd.

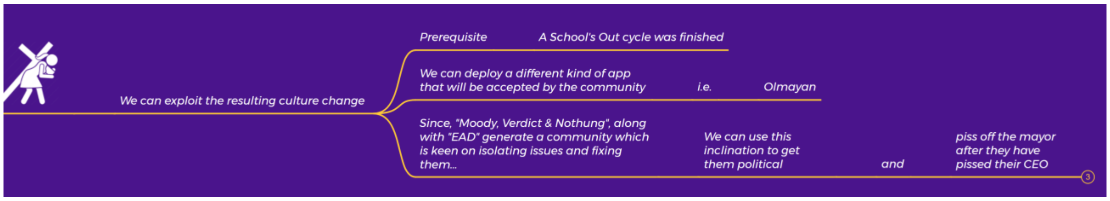
join forces
and
Link to other School's Out HUB(s)

Last but not the least, you must have links to those who are like you. They may have different kinds of tools in their toolboxes. Every player looks similar from a distance, but only a few know how to win the game let alone can do it.

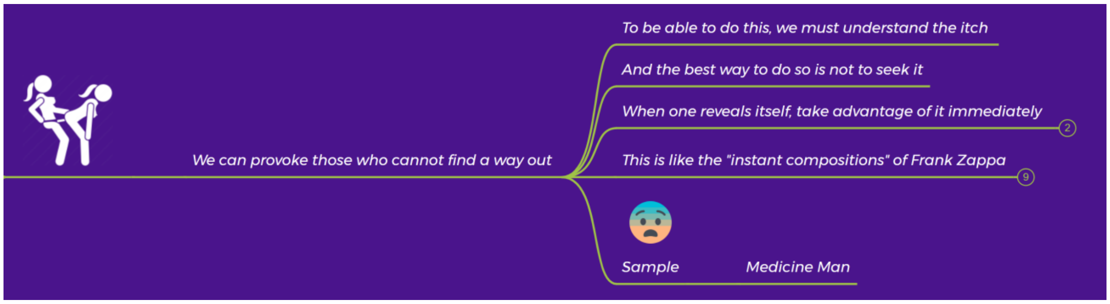
Extension Games


Extensions can be achieved in two ways
Near
is what we do in School's Out HUB(s)
Far
is what we can do elsewhere
Let us focus on "Far"

We have already covered what we can do within the context of School's Out. Let's have a look at what we can do as stand-alone projects now. Those are particularly important, because they are our experimentation grounds… meaning we have to experiment on ourselves as well. We have to evolve as well.

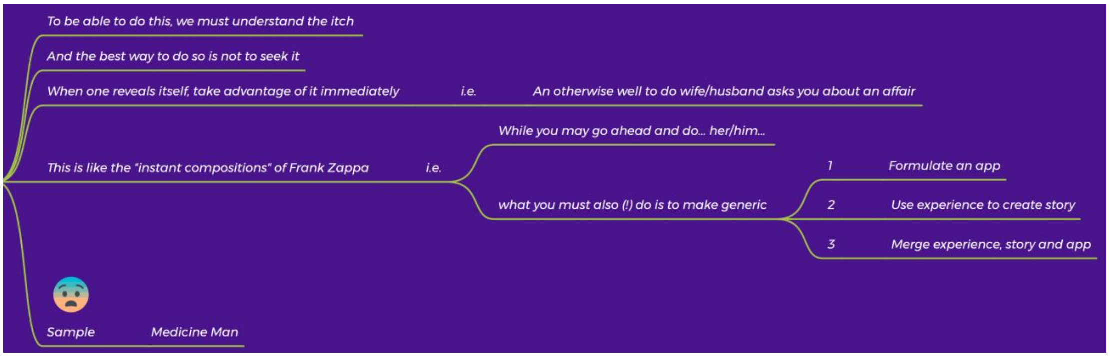

We can exploit the resulting culture change
Prerequisite
A School's Out cycle was finished
We can deploy a different kind of app that will be accepted by the community
i.e.
Olmayan
Since, "Moody, Verdict & Nothung", along with "EAD" generate a community which is keen on isolating issues and fixing them...
We can use this inclination to get them political
and
piss off the mayor after they have pissed their CEO

We can have a complementary project that takes advantage of a recently finished School's Out cycle. It can exploit the results. It can exploit the resulting state of mind. It can make use of artifacts of that cycle. For example, "Olmayan", a Notung variant can be even simpler. It can be a web page where a list is displayed. The items of that list can be moved up or down depending on their popularity. Since the first School's Out cycle focuses on cultural problems in big businesses to find new ways to change for the better, we can exploit that mentality and take it to an upper level. We can have a project that deals with issues in a country. "Olmayan" can be a political project to create a political party program communally. See, how easy it is to move from one company to the whole country. Dr. Evil would love this!

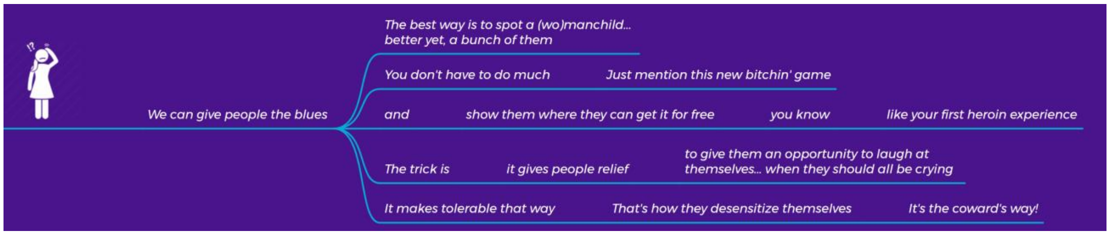

We can provoke those who cannot find a way out
To be able to do this, we must understand the itch
And the best way to do so is not to seek it
When one reveals itself, take advantage of it immediately
2
This is like the "instant compositions" of Frank Zappa
9
Sample
Medicine Man

The best writers are thieves. Always remember that. The best programmers, film directors and experimenters are that way too. Because there is no such thing as invention. Something grabs your attention. You focus. Then, you add, subtract, quieten or amplify… take apart or merge. Add your strategy to it. That's it. The art of watching, understanding and acting are the most important ones. Sounds like the art of war, doesn't it? In a way it is. Bad experimenters walk the earth with a well prepped set of experiments. The better ones compose instantly upon seeing an opportunity and seize the day. I belong to the latter group. That's why I share my findings and ask nothing in return. I need new ones.

Medicine Man is a good example. I like meeting new people. Sometimes I search for pen pals in countries that intrigue me at that time. I can tell you Welsch folk tales, for example. I can sing Latvian folk songs. I can cook Nigerian food. Those were my interests back then. Anyway, this time I wanted to visit Texas because I had so many great memories there. I poked at a strong looking lady who looked like a sheriff or something. Thankfully, she was responsive.

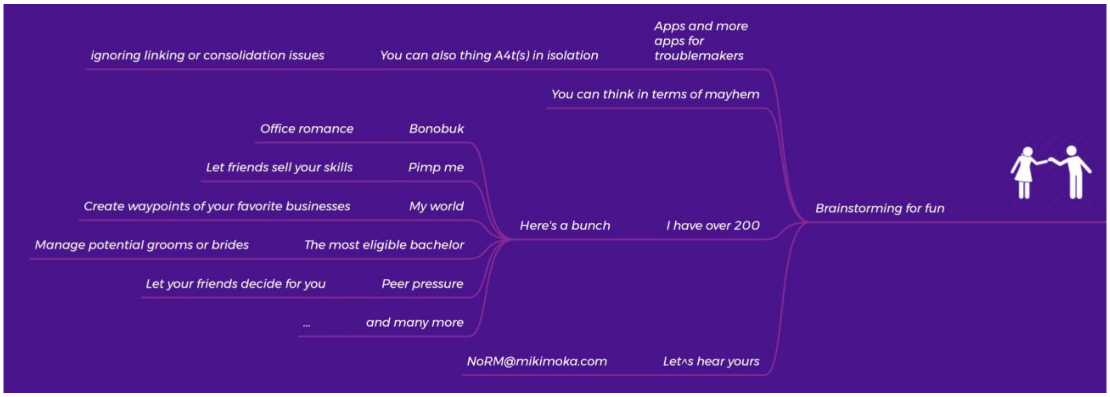

To be able to do this, we must understand the itch
And the best way to do so is not to seek it
When one reveals itself, take advantage of it immediately
i.e.
An otherwise well to do wife/husband asks you about an affair
While you may go ahead and do… her/him…
This is like the "instant compositions" of Frank Zappa
i.e.
what you must also (!) do is to make generic
1
Formulate an app
2
Use experience to create story
3
Merge experience, story and app
Sample
Medicine Man

In her second letter the pen pal thing skyrocketed to become a David Lynch story. She told me that she was having a sexless marriage, and she had a life too. It was strange because even if she was shopping for a guy to cheat on her husband, I was thousands of miles away not to mention there was an ocean between us. Being a joker, I told her to go to a certain hotel and wait for me. I thought it was funny. Yet, the subject didn't change. When people act weird, my natural response is to observe their behavior, formulate a mechanism (read, "app") and document my experience (read, "book"). And that was exactly what I did. I sent her the first page of a promising book concept. Of course, since I was the one writing it, the first page showed you an alternate interpretation of what was happening… preparing you for a sucker punch. The pen pal thing didn't go that far, though. One page ended it all. This is what happens when a person is too scared to leave her dull life behind. Not everybody can take the first step towards their freedom. Sad, but true.

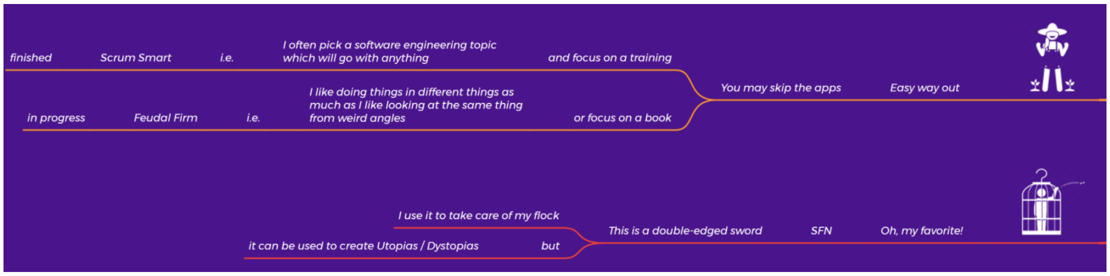

We can give people the blues
The best way is to spot a (wo)manchild... better yet, a bunch of them
You don't have to do much
Just mention this new bitchin' game
and
show them where they can get it for free
you know
like your first heroin experience
The trick is
it gives people relief
to give them an opportunity to laugh at themselves... when they should all be crying
It makes tolerable that way
That's how they desensitize themselves
It's the coward's way!

Seduction… is a dish that be eaten hot or cold. It's so simple in the case of the subjects. All you have to do is to place an instrument of interest (IOI) at the right place, at the right time. You don't have to do anything else. Indeed, you should not do anything else. It would just scare the herd! They must believe that they are in control of their fates. That it's their choice. That they are the logical folk. That they can be there instantly after consuming this or that. Yes, I know. They are idiots! Yet, where would we be without them? We wouldn't have any fun.

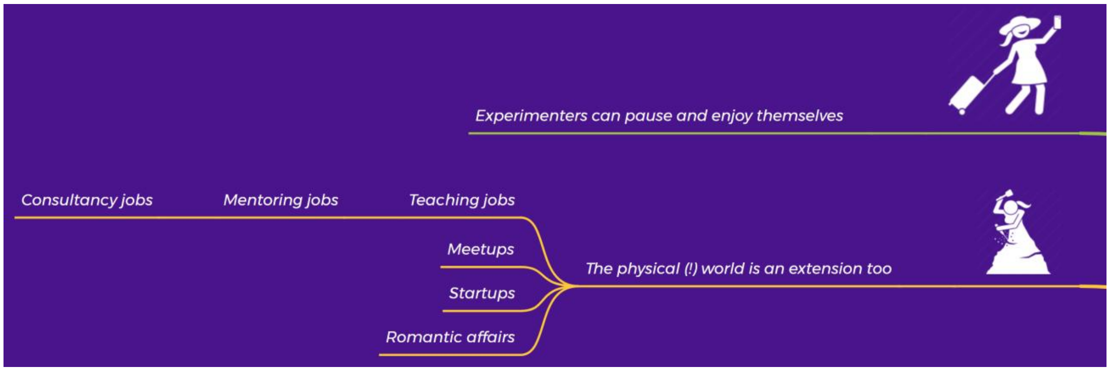
In our case IOI is an app. It can be a book or training too. Just show them how you are using the app and reveal genuine acts of pleasure. That's it! Make sure they know where they can get it… and then, you can leave the building like Elvis. Your name will be heard in the halls of Nerdinopolis, the city of nerds. OK. OK. I'm just fucking with you… but it's pretty much like that.


Brainstorming for fun
Apps and more apps for troublemakers
You can also thing A4t(s) in isolation
ignoring linking or consolidation issues
You can think in terms of mayhem
I have over 200
Here's a bunch
Bonobuk
Office romance
Pimp me
Let friends sell your skills
My world
Create waypoints of your favorite businesses
The most eligible bachelor
Manage potential grooms or brides
Peer pressure
Let your friends decide for you
and many more
...
Let^s hear yours
NoRM@mikimoka.com

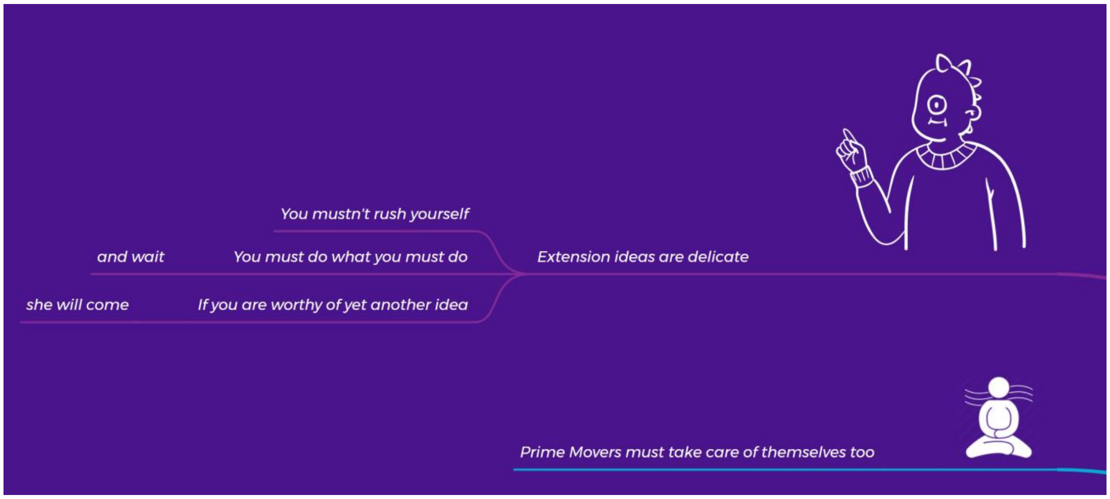
As a result, I always brainstorm for apps. I have 200+ ideas so far. You can see some of my favorites above. Send me your ideas. Who knows? Maybe we can turn them into apps. "Peer Pressure" is one of my all-time favorites. It turns one into a mindless automaton. Before you make a decision, you ask for votes of your friends… and they make the decision for you. Think about it… making every decision that way… life would be so easy. Yet would we still call it life?


Easy way out
You may skip the apps
and focus on a training
I often pick a software engineering topic which will go with anything
i.e.
Scrum Smart
finished
or focus on a book
I like doing things in different things as much as I like looking at the same thing from weird angles
i.e.
Feudal Firm
in progress
Oh, my favorite!
SFN
This is a double-edged sword
I use it to take care of my flock
but
it can be used to create Utopias / Dystopias

There are times when one feels tired and wants to put some distance between her and apps. When that happens, I focus on training materials and books. They prepare me for the next phase in a cycle, the phase we have just left behind. Training materials or books may be serious (!) ones… meaning they don't have to have anything to do with social experiments. You can't get better just doing what seems the main thing. Other things make all the difference in the *world. Behind closed doors your mind still works on problems you deemed worthy of your attention, but your mind needs some time for itself.*


Experimenters can pause and enjoy themselves
Consultancy jobs
Mentoring jobs
Teaching jobs
Meetups
Startups
Romantic affairs
The physical (!) world is an extension too

Other times, I don't even seek new ideas. I just grab what life throws at me. They may be about anything, but when you are mature you can turn everything into one of your favorite things. That's the luxury of old age. What seems like new is always old.

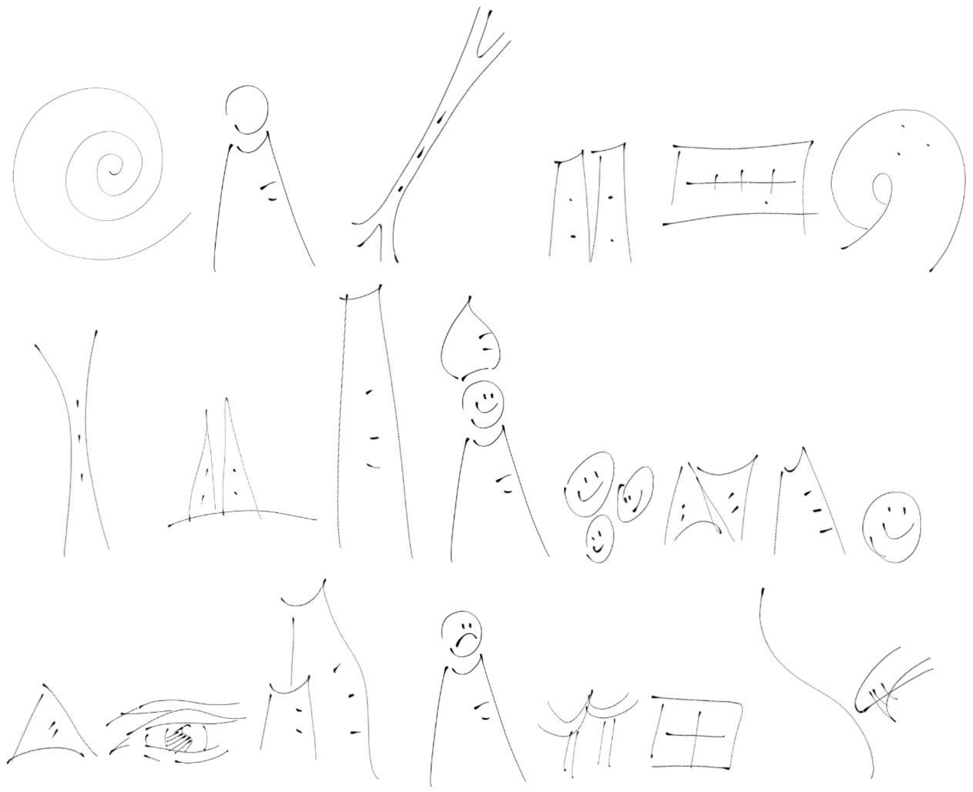

You mustn't rush yourself
and wait
You must do what you must do
Extension ideas are delicate
she will come
If you are worthy of yet another idea
Prime Movers must take care of themselves too

On the other hand, sometimes I don't do anything… I mean, anything at all. I know the truth is shy and you can never chase her. If you do chase her, what you will get won't be the truth. It would be something that is there to fool you. All you can do is to be worthy of the truth… is to be patient. She will come to you. Granted you deserve her, of course.

Disrupting Perceptions

"Disruptions and maddening moves" should be considered together just like "missing bits and deployment strategies". While testing is a concept that is active throughout the life cycle of an app, it becomes much more prominent during "birth" and "childhood", so to speak. So, you should be testing your theories during disruptions, and you should be testing implementations of them during deployment. Both of them have to be right for this to work. That being said, let's start with the first duo, "disrupting perceptions" and "making maddening moves".

Often, people prefer to have a very narrow view for it makes focusing easier. However, narrow views must always be complemented with wider ones, so that a complete view of the 'crime scene' is attained. I like to think about projects like unsolved murder cases with all kinds of corrupt people. Because not only does this perspective help me catch seemingly minor but critical information, but also it reminds me that we humans are incomplete. That we are bound to make mistakes here and there willingly or not. Did you know Ottoman Turks used to make deliberate mistakes when they were building their architectural marvels. They behaved that way to remind themselves that perfection isn't a realm of mortals. That when they made mistakes, they were their own… but when they didn't make any mistakes at all, it was by the will of the big guy/gal upstairs.



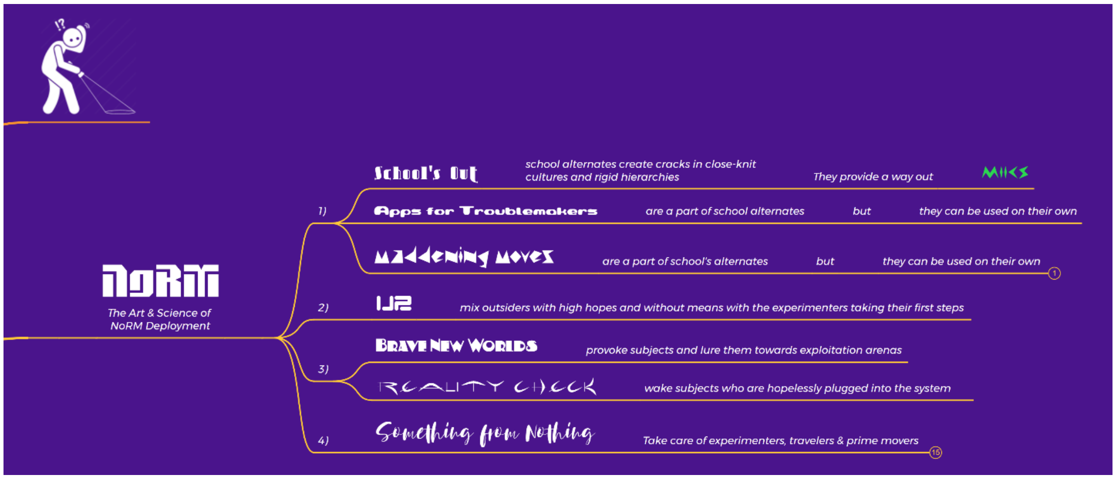
I always begin the case at hand with the big picture. That picture may be wrong. It may change. It doesn't matter. It's there to give me bearings. Bearings may change. The course may differ. Intermediary goals may be revised. Yet, the ultimate goal should never change. We are there to catch murderers. Anything less than that is not a job well done.

As a result, I approach the opportunity to disrupt perceptions with a bag full of symbols. During this process I revise existing symbols and invent new ones. Thinking in terms of symbols was Wagner's favorite too… and I'm a Wagnerian.

Let's make a list of these symbols:

1) *Alternate reality (super apps / Something from Nothing).*
2) *Subject (absurdity dweller who will be experimented on / School's Out)*
3) *Bad experience simulation (game apps / Reality Check)*
4) *Media production (stories, novels, films, TV series)*
5) *Higher education outlet (school alternate / School's Out)*
6) *Think tank (which specializes in resolving social conflicts / terminators)*
7) *Consolidator (is an app which consolidates the findings of interlinked apps)*
8) *U^2 (Part-Time Entrepreneurship / Uili is an MVP generator / Uusta is a micro consultancy service)*
9) *Intelligence Agency (intelligence operations)*
10) *Experimenter (is a School's Out graduate)*
11) *Interlinked apps (are apps for troublemakers feeding each other with complementary information)*
12) *App stories (are stories powered by apps)*
13) *School's Out HUB (District chosen for a cycle's social experiments)*
14) *App for troublemakers (Mobile apps designed for social experiments)*
15) *Maddening moves (Interlined apps dispersed over time with a unifying strategy)*
16) *Traveler (is an exceptional School's Out graduate who walks the earth seeking absurdities)*
17) *Ecosystem (is experiment grounds)*
18) *Troublemaker (is a subject with guts who can be recruited to School's Out)*
19) *Hangout (is a facility in the selected ecosystem)*
20) *Home (is where a subject lives)*
21) *Prime mover (is a seasoned traveler who specializes in terraforming cultures)*

I use these symbols to think, to develop strategies, to transfer what I have in mind to the others… and to create shareable pictures of my thoughts. I believe they are pretty much matured, for I don't need anything extra when working on schemes regarding School's Out or associated projects anymore. Before going any further, let me remind you of these associated projects once more.

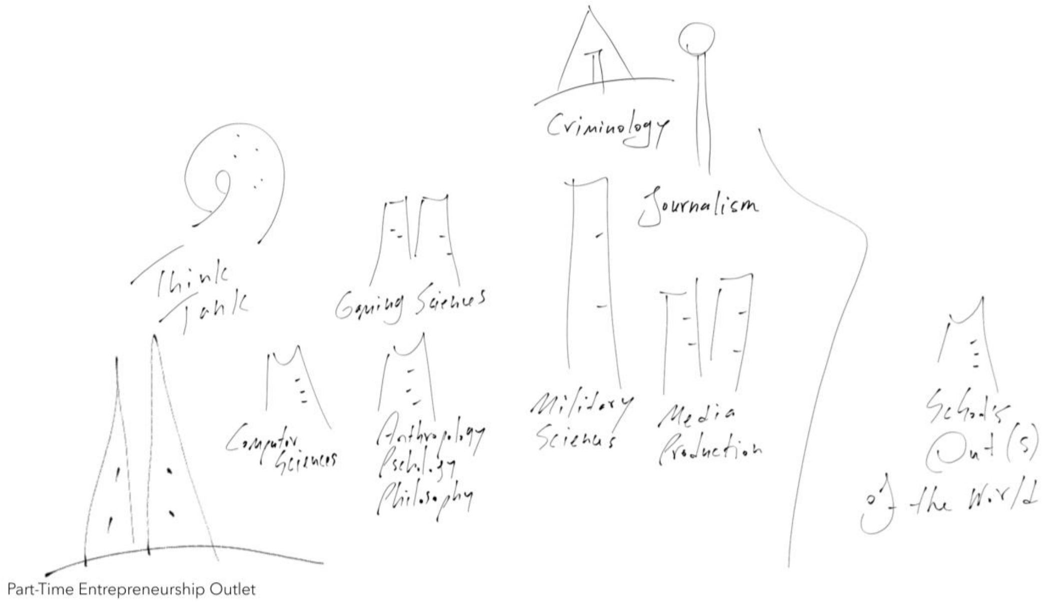

NoRM
The Art & Science of NoRM Deployment
1)
School's Out
school alternates create cracks in close-knit cultures and rigid hierarchies
They provide a way out
MIIKS
Apps for Troublemakers
are a part of school alternates
but
they can be used on their own
Maddening Moves
are a part of school's alternates
but
they can be used on their own
1
2)
U2
mix outsiders with high hopes and without means with the experimenters taking their first steps
3)
Brave New Worlds
provoke subjects and lure them towards exploitation arenas
Reality Check
wake subjects who are hopelessly plugged into the system
4)
Something from Nothing
Take care of experimenters, travelers & prime movers
15

- *"School's Out", "Apps for Troublemakers", "Maddening Moves", "Something from Nothing", "Part-Time Entrepreneurship", "Brave New Worlds" and "Reality Check".*

"And what were they all?" about, you ask…

- *SO = School Alternate*
- *A4t = Apps designed for social experiments*
- *MM = Strategically deployed interlinked apps dispersed over time to achieve a cultural change*
- *SFN = Apps designed to create alternate realities and supplement associated novels / films*
- *U2 = An incubator for busy (!) professionals*
- *BnW = Stories powered by apps*
- *RC = Bad experience simulations / games*

Now that we are aware of our boundaries, let's add another dimension to our thinking. We can use our method in four different fronts. One of which is School's Out… and we have already been thinking about it in terms of apps for troublemakers. However, we can conduct experiments, terraform cultures and create alternate realities for Intelligence Agencies, Media Production Companies and Think Tanks as well. Wait! Not so fast. There is another, final, tool in our toolbox. I'd like to call it "School's Out on Drugs", "SOoD", for short. Let's have a look at its components too, shall we?



This School's Out has School's Out(s)! Wild, right? Also, it's more of a campus than a hub. It has eleven components:

1) *School's Out Headquarters (Computer Science Unit).*
2) *Anthropology Unit.*
3) *Psychology Unit.*
4) *Philosophy Unit.*
5) *Media Production Unit.*
6) *Gaming Sciences Unit.*
7) *Journalism Unit.*
8) *Criminology Unit.*
9) *Military Sciences Unit.*
10) *Think Tank.*
11) *Part-Time Entrepreneurship Outlet.*
12) *(connections to all the School's Out hubs in the world)*

So, if we summarize the whole thing, we will end up with the following:

1) *Four ecosystem options (out of 21 options)*
   a. *School's Out Hubs near businesses with interesting issues.*
      We are here!
   b. *Government Intelligence Agencies seeking new tools for their operations.*
   c. *Media Production Companies seeking ways to merge dreams and reality.*
   d. *Think Tanks specializing in social conflict resolution by creating terminators.*
   e. *Interest Groups looking for more ways to achieve harmony.*
   f. *Part-Time Entrepreneurship Outlets making half-assed approaches account for something.*
2) *Application components*
   21 (minus 4) in total which can be used to think, describe or execute.
3) *Hub components*
   11 pieces of the puzzle which connects to School's Out hubs everywhere.
4) *Strategic instruments*
   Seven strategic elements (of 21) which can be used to define and execute relevant strategies.

Square One

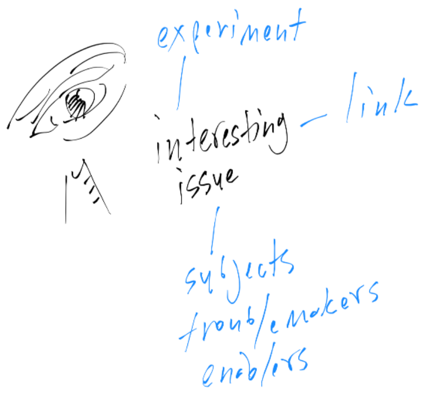
As you might have guessed I'm a big picture kind of guy. I just love imagining big pictures, but that doesn't hold me back. Because I know that one must work in tiny worlds knowing the universe is so much bigger. The tiny world in our universe is the one which brings everything together in the form of an app… An app for troublemakers.

So, let's get back on track by remembering the fundamental question: "How does one apply an app for troublemakers to a particular ecosystem? How does pick an ecosystem in the first place?"

An ecosystem… is selected by a traveler when he spots an absurdity which is interesting. That interesting issue is accepted by a prime mover when she thinks the associated ecosystem is experimentable. See, there is a method to our madness.

Interesting issues are issues which are well suited to social experiments… and they are never alone. There are always a few other interesting issues in close proximity to the one we picked first, always linked in some way. Through careful analysis of the nature of absurdities therein we can create two critical artifacts:

- *Concept formulae.*
- *Subjects and linked personae.*



Concept formulae lead to apps. Subjects are the actors using those apps. Linked personae are troublemakers who can be recruited by School's Out as future experimenters and enablers who are subjects in similar ecosystems. Subjects in other ecosystems behave in a way that they are not subjects which is something that should be exploited to expand the experiment further. This strategy can be described as using similar groups of people against each other while appearing to be joining forces. A powerless, cowardly middle manager in the first zone may look pathetic to another powerless, cowardly middle manager who is in a zone further away from the first one. That distance makes him feel different. Oh, well, ignorance works in mysterious ways.

First and foremost, we don't come up with stuff here. We just open our eyes and let whatever skeleton lying there reveal itself to us. In that regard an interesting issue is an absurdity begging for more attention. For example, the interesting issue that gave rise to Moody was a group of workers (read, "close friends") who have been working in the same company for years and years and years… and still complaining about this or that. Some of them happen to change companies once in a while, but it really doesn't count. They only move between very similar notions of reality… and expect very different results. Yes, they are not very bright. More precisely, they never have the guts to really… really leave. Still, they make bold claims that's exactly what they do. When they are cornered, that is. So, our subject (here's a new term, "exploitation entity") was that kind of worker. She just needed our app… to bitch more efficiently.

Once that was clear, formulating our app for troublemakers was straightforward. It had to accept a complaint and requested a reason without being threatening. Between "the opportunity to bitch" and "the bitching has occurred" states, we must make that strong feeling reveals the reason behind the feeling without much thought. We can learn a lot of things about that worker's ecosystem as the workers keep using Moody effortlessly. Once we have that, we can do all kinds of things with that information. That's the main premise behind all apps for troublemakers. We seem to be helping the exploitation entities, when actually they are the ones helping us. Every minute a sucker is born.

Let's take a minute and digest what we have covered thus far. This is the first step in A4t application.

- *We have spotted an interesting issue.*

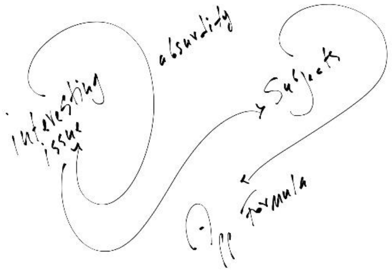
  "Workers are bitching about things that don't change. Yet, they don't quit."

- *We have uncovered the presuppositions behind the absurdity.*

  "Workers bitch to release tension, to be able to continue working there, because they are cowards".

- *We have formulated the first app.*

  "This A4t must help workers bitch efficiently while retrieving critical information for further analysis".



A more refined process may be expressed like this: "When you spot an absurdity, something unintelligible under your presuppositions, first you must find the presuppositions that make it intelligible. This will make the app formula you seek reveal itself. People who have those presuppositions will be your subjects. They are the ones you should torment, while two other kinds of people will help you in different areas.

While they too are very similar to the subjects, they differ in critical ways. Enablers are subjects far away from the ecosystem you have selected. Troublemakers are subject who can be saved."

Step 2

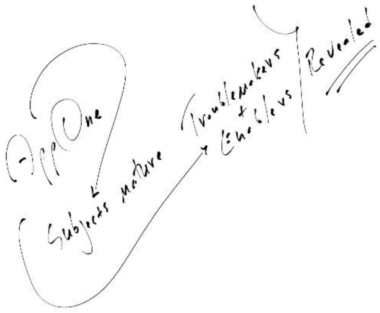
Now, we must perform usual analysis chores including those of Business Process Analysts, System Analysts, Interaction Designers and my favorite, Timeline Analysts which is something we have invented (MIKS). I will not go over those here, because we have already covered them in the previous chapters… except Timeline Analysis which is reserved for the training material… included in your gift pack!

Step 3

Since we can deploy our first app, let's do so. What we must do is to let the first app for troublemakers take hold of the subjects. Once we have a devoted following there, troublemakers and enablers will be easy to recruit. Here the secret word is patience. We mustn't introduce any additional app to the mix let alone a consolidator. Their time will come. "When will we stop", you say, huh? Well, the moment Moody spits out enough data which you can turn into meaningful information, you can deploy other apps. It should take at least one month granted you have a mass following.

Step 4

By now you should have a steady group of voters who are beginning to piss off upper management. When human resources manager starts walking around trying to give you the impression that she is going to fix the issue while in reality she doesn't really give a fuck… you have arrived. It's time to deploy additional artillery. The next app in line is "Verdict"", because it will scare off opposition by providing clues. This is what you should always aim for, making your crazy scheme take charge. The metric you should be following is the growth of your fan base. Yes, you are a king maker powered by mobs!

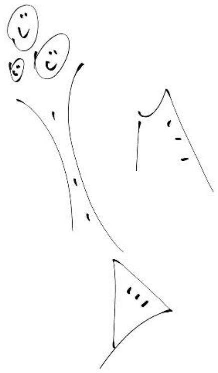


You must always think in terms of groups and the links between them. When you identify your subjects and begin experimenting on them, your better understanding of them will take you to the next step. You will uncover their siblings, "troublemakers" (who will be your enthusiasts in School's Out Hubs) and "enablers" (who are subjects in other ecosystems, so they can fuck around in the one you have selected for your experiments).

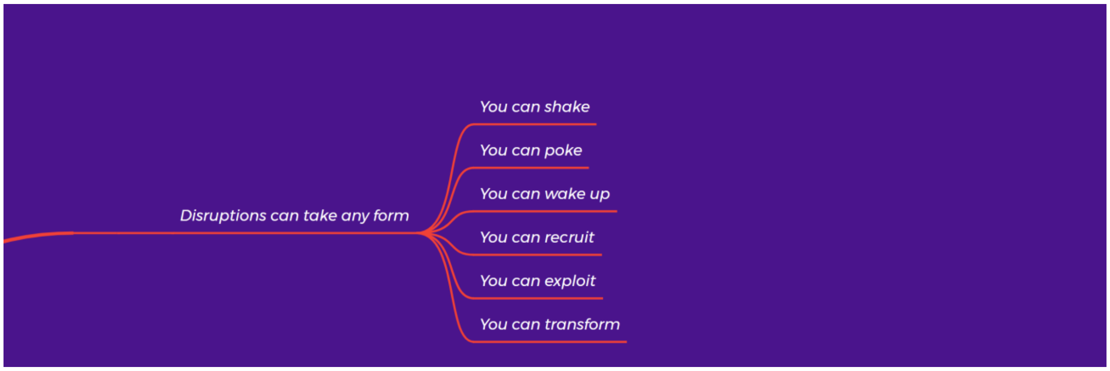
A better understanding of all three archetypes will help you formulate the remaining apps in your trinity. What's more, it's through them you will invent a new paradigm… a paradigm that will excite and motivate the likes of them anywhere.

Step 5

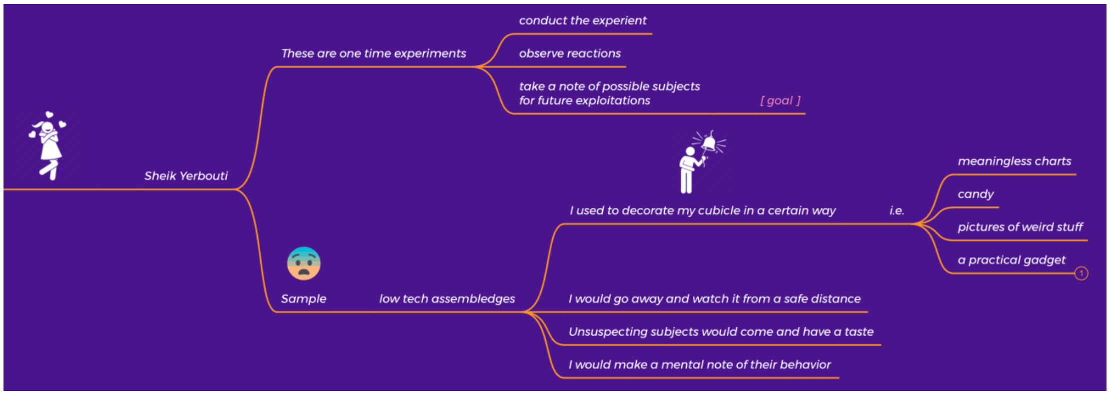
"Exploitation begins at home", remember? So, those using the second app should be the ones who are already using the first app. You can change your strategy later depending on the dynamics of your subject pool. Sometimes it is best to divide a group. Other times different groups should merge either by conflict or collaboration.



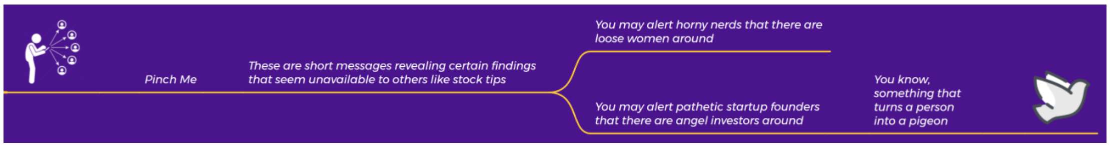
You must keep in mind that "applying apps for troublemakers" and "initiating maddening moves" are two sides of the same coin. If there is an [interesting issue] you can exploit, it often has several facets to it. When you formulate the most obvious itch in terms of a mobile app, you uncover the remaining experiment opportunities as well. While the data structures of these apps are interlinked, those who are using the apps don't have to be. What this means is "when you provide opportunities for mental weaknesses to come together, they find friends very easily". You don't have to exert too much energy after the first app. It all becomes very easy after you gain your first fanbase. Those with weaker minds are often very energetic and they will do your work for you.



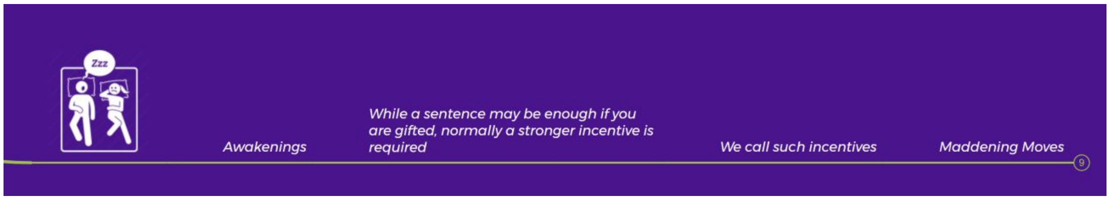
Disruptions come in any form. Smaller ones are not necessarily lesser ones. Indeed, those who are influenced by small disruptions are the better stock. Not everybody needs an alarm clock, you know. Also, the target ecosystem may not grant access to certain exploitation entities. You may be onto another thing. What you think you are looking at may be something else. So, use a combination of these keeping in mind that the School's Out cycle is what matters.


Sheik Yerbouti
These are one time experiments
conduct the experient
observe reactions
take a note of possible subjects for future exploitations
[goal]
Sample
low tech assembledges
I used to decorate my cubicle in a certain way
i.e.
meaningless charts
candy
pictures of weird stuff
a practical gadget
I would go away and watch it from a safe distance
Unsuspecting subjects would come and have a taste
I would make a mental note of their behavior

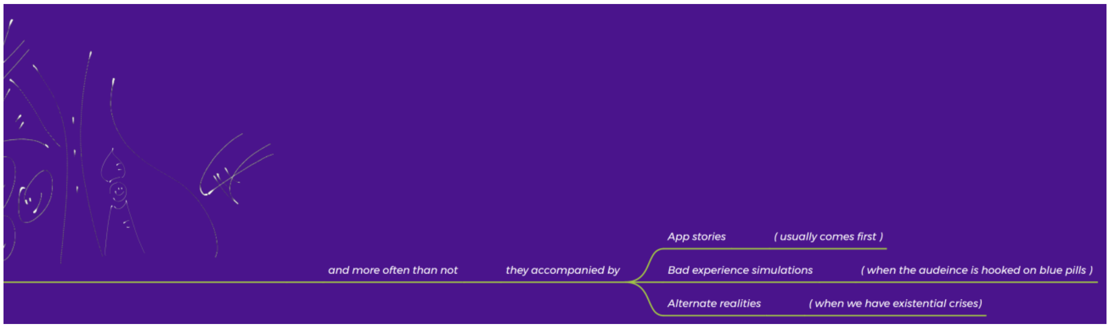
One of my favorite pokes is using low-tech assemblages, meaning often I don't use computers at all. Think about it this way, you can position things, you can take advantage of the positions of things, you can exploit the relationships between things, or you can play with the beliefs concerning the properties of things. Dreams are realities. Realities are dreams. That's the way we are.


Pinch Me
These are short messages revealing certain findings that seem unavailable to others like stock tips
You may alert horny nerds that there are loose women around
You may alert pathetic startup founders that there are angel investors around
You know, something that turns a person into a pigeon

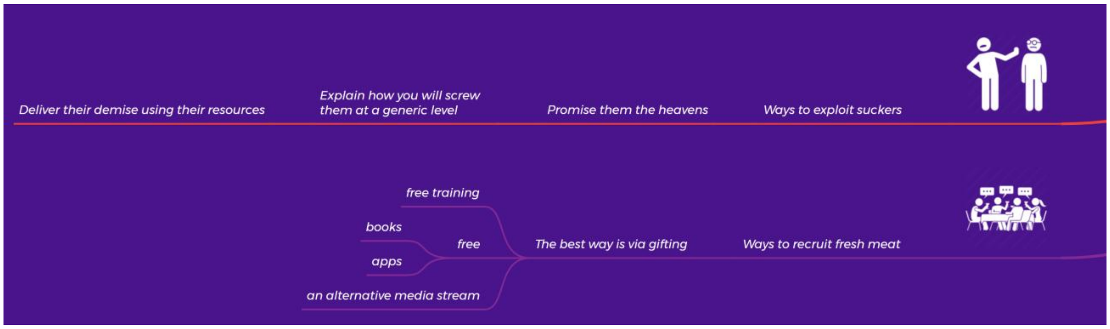
A poke is an unexpected intel that gives the receiver the impression of unfair advantage. The intel may be true or not. In both cases different experiments can be conducted. Pokes can be deployed in combination even if they are disjointed. Furthermore, chaos is so much bigger when they are disjointed. Because humans merge them in all kinds of ways. Think about it, "humans being the glue". It's fun.


Awakenings
While a sentence may be enough if you are gifted, normally a stronger incentive is required
We call such incentives
Maddening Moves
9

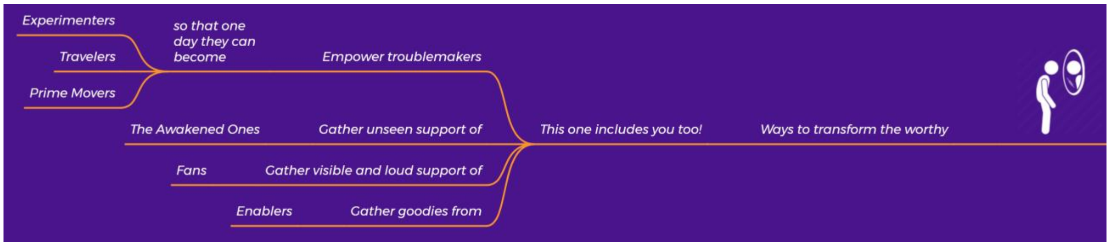
Developing deployment strategies and understanding the quality of your subjects go hand in hand. In a typical scenario, you deploy app one, then apps two and three, and much later the consolidator while you are transferring control over to your subjects. However, if a maddening move is needed, you can continue to be the supervisor of the cultural terraforming project. Regardless, you must set up your initial and target cultural bearings. A certain group of app findings and a consolidator interpretation will do. While your subjects are moving from one to another, you keep track of their progress. Of course, this configuration may change as needed. It can be more relaxed, or it can be more aggressive. It all depends on the ecosystem.


and more often than not
they accompanied by
App stories
(usually comes first)
Bad experience simulations
(when the audeince is hooked on blue pills)
Alternate realities
(when we have existential crises)

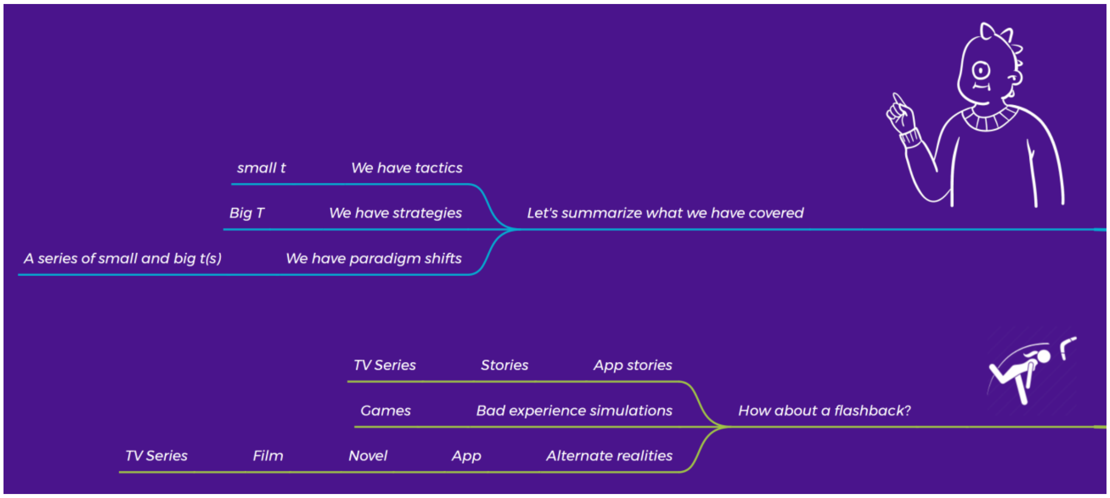
Why stop there? You can combine the instruments of evolution as you see fit. So, while your ecosystem is moving from one culture to another in a maddening move, you can help subjects "dream big with app stories" and "understand others with bad experience simulations". Alternate realities? Well, it's a tricky business. While it is possible to use them too in that context, let's assume that they are for the experimenters not the experimented… at least for now. We will get back to them in the future when you are more comfortable with social experiments and cultural terraforming.


Deliver their demise using their resources
Explain how you will screw them at a generic level
Promise them the heavens
Ways to exploit suckers
free training
books
free
apps
an alternative media stream
The best way is via gifting
Ways to recruit fresh meat

As you can see, people can be tempted by introducing opportunities into their lives. If they are selfish, they are more interested in getting to their Point B(s). If they are free, so to speak, they are interested in getting access to a larger universe. So, the former desires dreams and the latter desires means to dreams. What are you waiting for? Give it to them! The first one doesn't need finding. They will give you headaches when talking about themselves. The second ones, on the other hand, need a human touch. Just play your flute pied piper… and they will come out of the shadows.



To us "one" is "many". Whatever we do, we do it in numbers. That means once you gained control of the experiment zone, you should spread like a disease. You should get in touch with similar subjects living somewhere else, those with some kind of control over their lives… with some kind of power in their hands. When you bring your subjects and those with some power together, fans will emerge. They are troublemakers coming from all kinds of ecosystems. What makes them different is they can understand what you are aiming for. Their word of mouth is the most effective PR machine ever created. Their loyalty is legendary. You may find yourself on the edge of your world, old and wary, ready to quit. And they will be the ones who will resurrect you and join you to carry on with the cause.



You know, Alan Cooper (the guy who came up with the persona concept) was thinking about movies when he had the idea. Ivar Jacobson was fed up with doing the same tests over and over without any control whatsoever when he came up with the idea of use cases. Pretty much like them, I think everything in terms of stories. So, my reality is a mixture of past, present and future. Aha! That's a "Dark" moment for you. I also believe what Buddha once said. That "what we imagine, we become" … meaning there is no such thing as reality. Reality is created as we keep walking on an invisible treadmill. Yet, "meaning" is a whole different matter. The meaning of our lives is revealed through care and concern manifested in time. So, thinking in groups, considering one to be many is the way out.

A4t 1

Making Maddening Moves

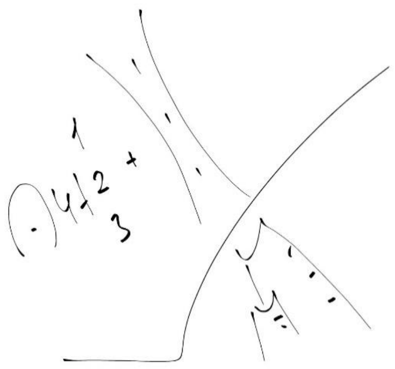
Not all experiments go well even if your theory holds. Your subjects may be hostile. Their ecosystems may be unwelcoming. If, on the other hand, your subjects turn out to be warm blooded, you can knock yourself out. You can terraform cultures for the better or the worse.

A Maddening Move is the application of a combination of Apps for Troublemakers while taking time into account. For example, you may pick Moody and decide that you will start at {work life balance, unhappy} and arrive at {work life balance, happy} in six months. However, you can be bolder. You can define milestones taking more than one experiment into account.

For example, you can pick an unhappiness trio {work life balance, management style, employee profile} and aim to have a better managed workplace, a decent private life and more diverse colleagues. Of course, such bolder moves are also trickier. Thus, you must make use of attributes of the other apps in the trinity. Together they will provide you with high levels of control for your social experiments.

- *Moody provides feelings and reasons.*
- *Verdict provides issues and clues.*
- *Notung provides ideas and solutions.*

Think about it this way, "you are interested in negative feelings associated with work life balance, management style and employee profile". So, you want to change the schedule, the strategy and the team compositions. This also means that the issues you receive about those criteria should decrease in time along with clues that support them. Also, ideas shouldn't be pouring in, or proposed solutions shouldn't stay at the top of your current agenda. While these associating aren't hard wired, they are easy to form, thanks to the way apps in our trinity are linked. Feelings are associated with reasons. Issues are associated with criteria and supported by clues. Ideas are associated with criteria and voted for as possible solutions. This underlying theme is also our paradigm, "emotional agility". It will also be the focus of our one-page consolidator with pressure points such as company rankings according to the emotional agility index no less. Regardless of which app came first, we already have the company and the community information of the employee.

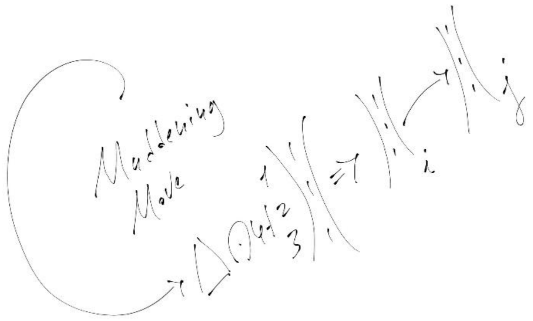


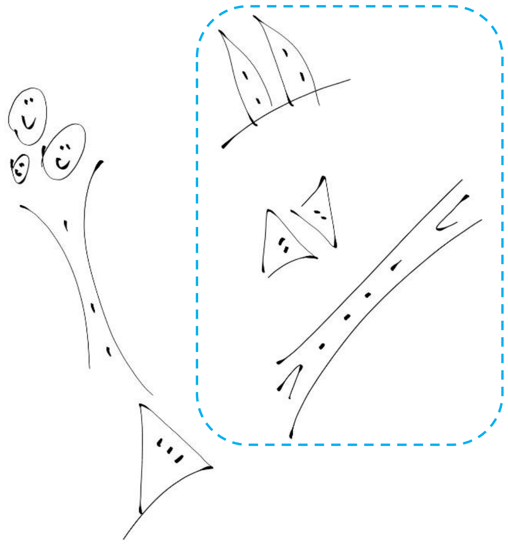
A Maddening Moves has degrees like earthquakes. It may be just a little bit bigger than Apps for Troublemakers, an app generating different attribute values in time, or a few apps linked doing the same. However, the best way to is to use all three apps with a consolidator. Apps aren't the only tools here, though. People should be circled around them. Those in the ecosystem (subjects, troublemakers and enablers) should be linked to those in School's Out hubs (enthusiasts, Sherlock Holmes - Dr. Watson duos, A-Teams and experimenters). Two different manifestations of the same type of people, "troublemakers and enthusiasts" are the point of contact for both worlds. Be careful what you wish for, for things, beings and relationships make up worlds.

You may stop here and open a geography book. Focus on maps. Look at legends. Go through different kinds of maps. Try to relate one to another. Better yet, you may combine seemingly irrelevant maps. Make two dimensional maps three dimensional. Understand that as you create maps of your imagination you are actually creating alternate realities, realities within which only those sharing your notion of reality feel at home.

If we simplify the process, we use just one variable that is time. Time is what links dreams to reality. Through dedication and hard work one can gain followers and with their help she can create the communities of the future.

There is no guarantee for success, but the willingness for self-sacrifice, desire to help others and always considering oneself half-baked helps one to travel through the dark nights of the soul even if she may not be among those who reach the other end.



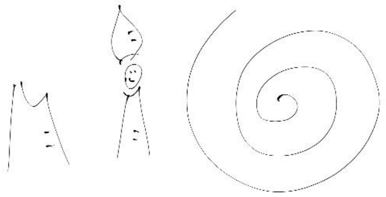


Maddening Moves are all about time. School's Out and Apps for Troublemakers are all about people. In that regard, they complement each other. However, you should keep in mind that other NoRM components can be handy too. You can even mix path options with evolution instruments.

In our arrangement, the artifacts of a School's Out cycle are joined with other components. We also see that the artifacts of this School's Out Hub were taken as far as they could be. Because a maddening move was initiated there.

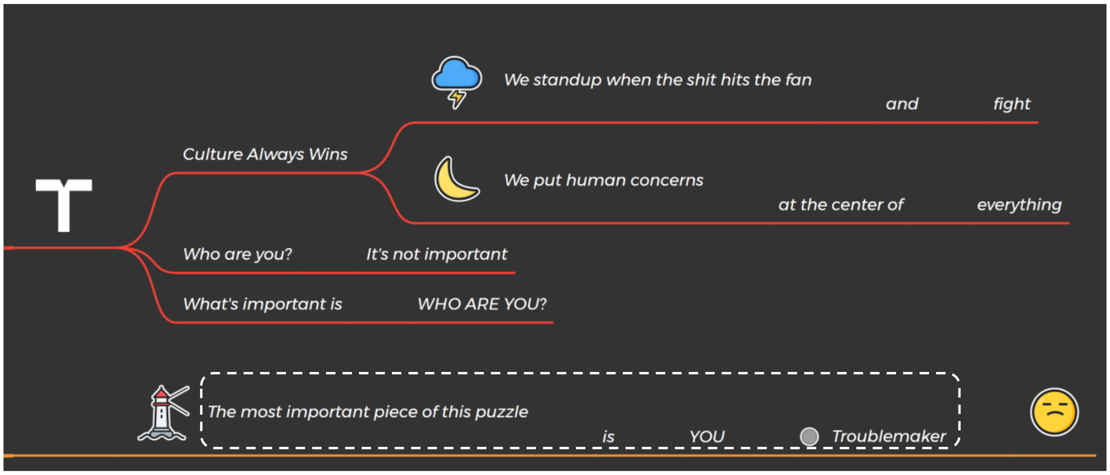
This experiment is further strengthened by providing an outlet for escape (Part-Time Entrepreneurship), stories to broaden horizons (App Stories) and an awakening mechanism to help captives understand their circumstances better (Reality Check).

Think about it. You have a vast arsenal which provides all kinds of combinations. You can experiment, transform, create, destroy… You are God in this game… unless, of course, someone smarter than you may decide to experiment on you. Still, it's a pretty good deal. What are you going to do, complain all day, every day… and do nothing… keep on taking crap from everybody? Stop being a subject and become a troublemaker. Who knows? If you've got the goods, maybe you will end up as an experimenter one day. And remember, travelers and prime movers were all experimenters at the beginning of their journeys. So, come on, hero! This is your call to adventure.

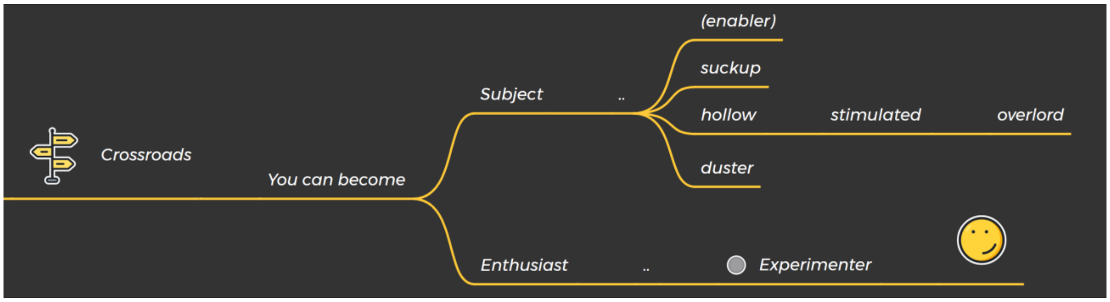


While it isn't a must, "Something from Nothing" is best suited to experimenters, travelers and prime movers along with enthusiasts. Because it has a protective cover against the harsh realities of the world. Of course, it's a double-edged sword, for it can too become a dungeon for the unworthy. That's not a problem for us, however. We designed it to take care of our own first and foremost. So, see you there!

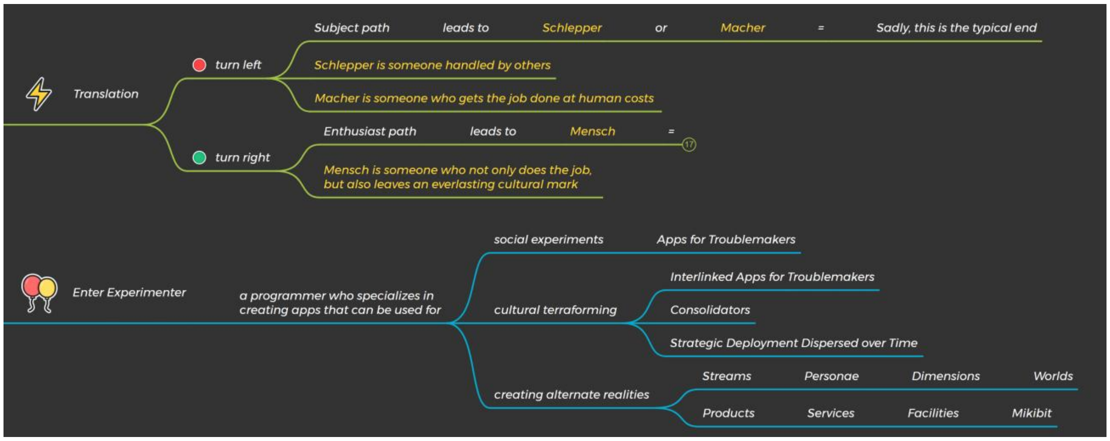
More on Troublemakers



They say, "culture eats strategy for breakfast" and I agree. Tactics, even strategies can go only so far. Unless your culture… character is a gem, you are akin to dirt no matter what you do, no matter how hard you try. They also say, "you can take a girl (!) out of trailer park, but you can't take the trailer park out of the girl" … which is a better philosophical point, if you ask me. So, we are aiming for a cultural change here, a complete transformation on a personal level.

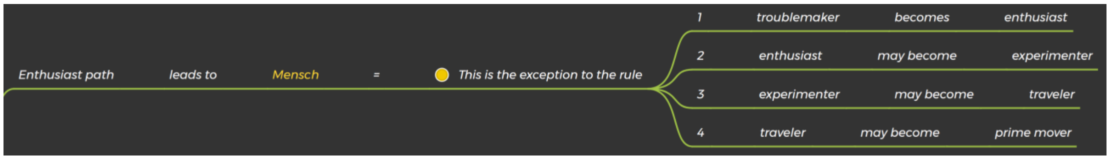


It all begins with your initial move or ends with it. You can choose to continue a diminished existence where you may have tastes and smells while you are living among so many like you… or you may choose a riskier path, more adventurous yet so much more fun. Making the wrong decision (former) begs for forgetfulness. This is why there are so many masks on that path… masks you can cherry-pick along your way becoming your undoing while never suspecting it will only go bad from here. The other path (latter) looks gloomy, but it isn't. When you shed your old skin, little by little you become someone who seeks nothing… who gives before he takes anything. That path transforms you into an experimenter who can become a traveler or even a prime mover if you play your cards right, that if you are worthy.

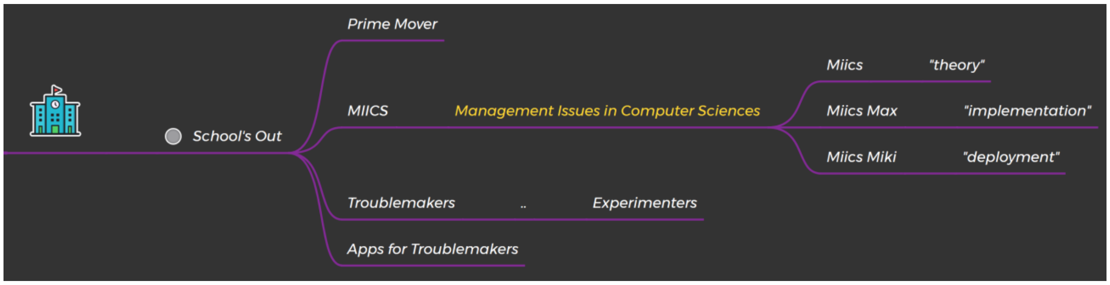

Translation
turn left
Subject path
leads to
Schlepper
or
Macher
=
Sadly, this is the typical end
Schlepper is someone handled by others
Macher is someone who gets the job done at human costs
turn right
Enthusiast path
leads to
Mensch
=
17
Mensch is someone who not only does the job, but also leaves an everlasting cultural mark
Enter Experimenter
a programmer who specializes in creating apps that can be used for
social experiments
Apps for Troublemakers
cultural terraforming
Interlinked Apps for Troublemakers
Consolidators
Strategic Deployment Dispersed over Time
creating alternate realities
Streams
Personae
Dimensions
Worlds
Products
Services
Facilities
Mikibit

Another way to look at it involves seeing beyond archetypes. Remaining as a subject is the life of a schlepper. He can only become a macher, if he is ruthless. Think this in terms of the slaves who don't have what it takes to escape. They can either be ordinary slaves or special ones such as Uncle Toms, slavers, house slaves or comfy girls. Theirs is a life of servitude no matter how good it may look.

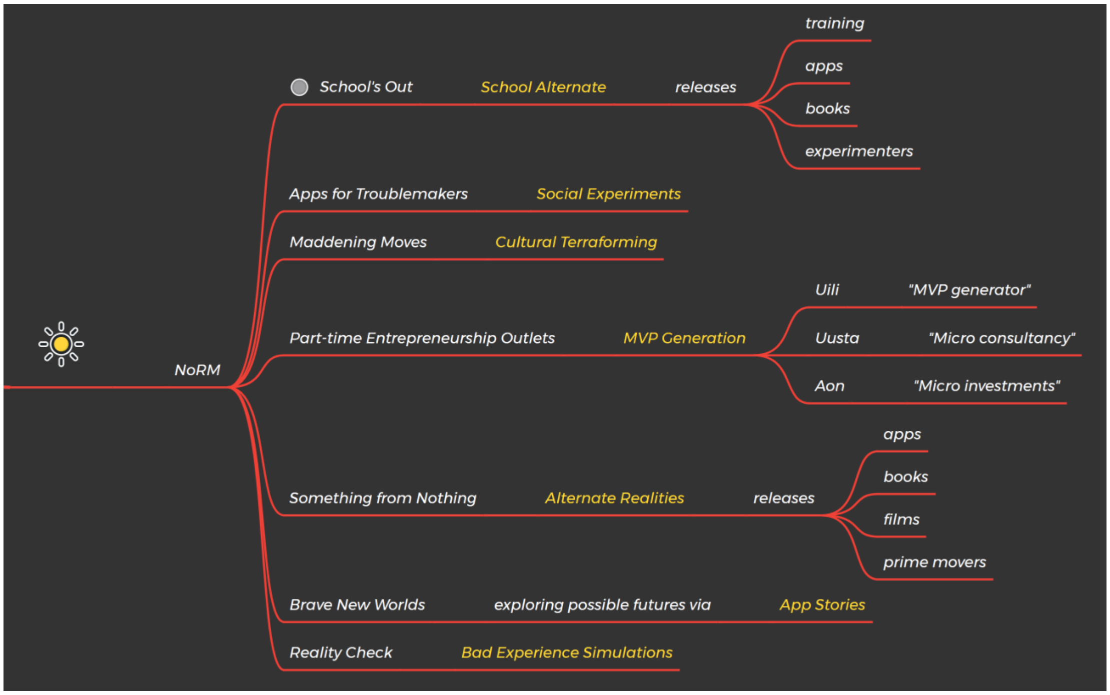
The other way is the path that may lead to becoming a mensch. It is by no means guaranteed, but a true believer (!) doesn't need such assurances. He knows that if he doesn't make it, the reason is simple. He isn't worthy of it. His accepting such harsh criticism is the mark of the potential for becoming a mensch. There will be many trials along the way in the forms of experiments, cultural terraforming and alternate realities. The life of a true believer (!) is a lonely one… but then again, he always has good company one after another.



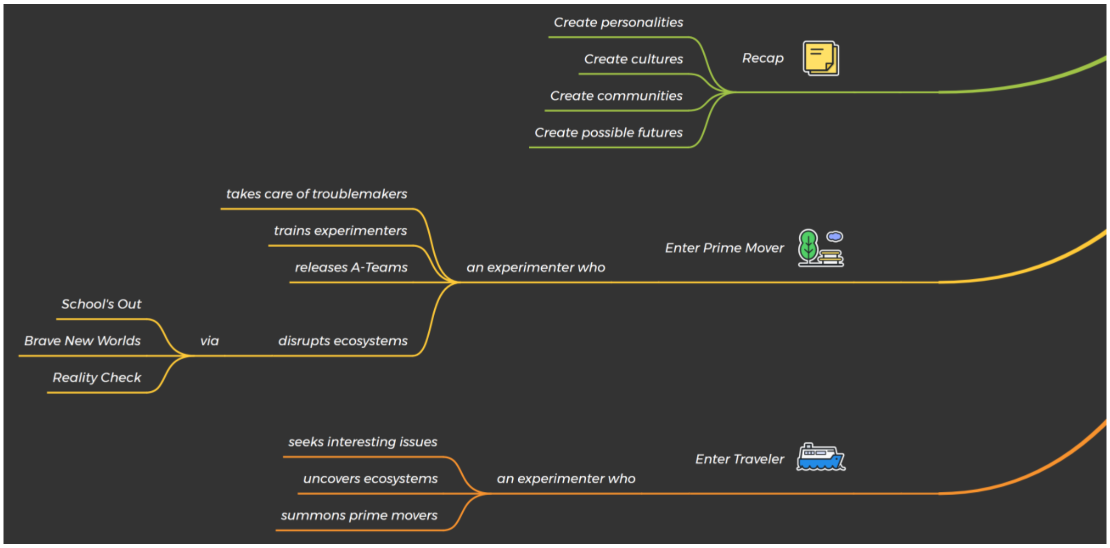
Let's go back for a minute. The road to greatness begins with making noise. When you are one of the few who speak out, create alternatives and dream of possible futures, you are at the door. Whether you will do what's necessary to step over the threshold is a big question. It is only possible by realizing you are the one who should be connecting the dots. Otherwise, you will dwell on past heartaches until the day you die.



School's Out is at the heart of NoRM Institute. Everything revolves around it, because this is the birthplace of experimenters. They are the nebulae in our alternative universe. Prime movers are the supervisors of the whole process. They are like galactic gardeners (or the engineers in Prometheus, engineers with a better attitude) who take pride in *their offspring. First, they make sure everybody is on the same page. Then, they focus on programming skills. Finally, they test that cycle's offspring in the trenches. Those who survive the process continue their journey as experimenters. Some of them may become travelers… and only a handful of them may become prime movers. When there is no one worthy in a cycle, it is regarded as normal and the focus is diverted to the next School's Out cycle.*



Think about components of NoRM as your toolbox as you "boldly go no one has gone before". If you spot "an earth like planet" so to speak, you can create a School's Out HUB. Otherwise, you may proceed with a stand-alone experiment. You may decide to exploit the untapped energy in the ecosystem by using part-time entrepreneurship outlets. If a School's Out HUB proved to be fruitful, you can continue with cultural terraforming. Regardless, you may decide to wake up the inhabitants of a colony with bad experience simulations or you may just provoke them by app stories. Of course, reality creation is a special case. More often than not, you will be using it for yourself.

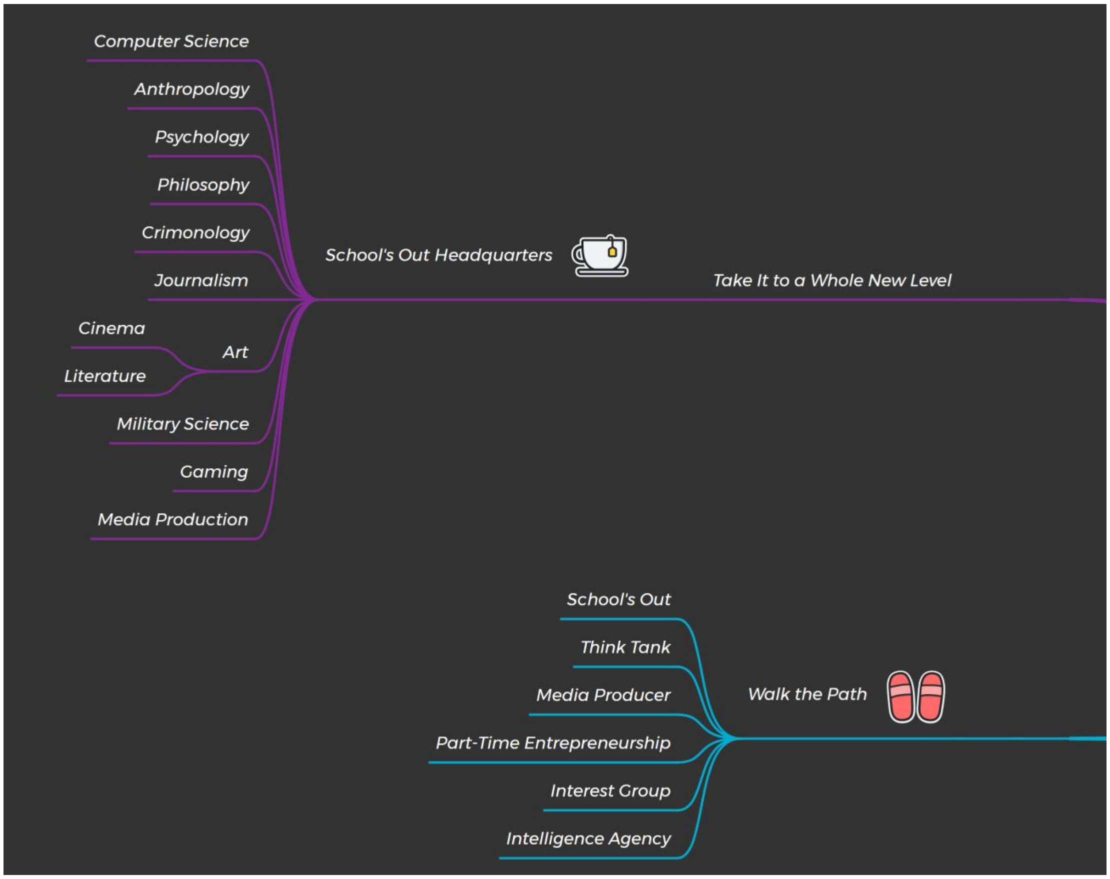


Travelers are seasoned experimenters who have developed a keen eye for spotting absurdities (interesting issues, exploitation opportunities). They walk the earth seeking absurdities which are worthy of their attention. When they find one, they try to understand it in terms of app formulae. If they are impressed with the results, they uncover the relevant subjects, survey the associated ecosystem… and if they are still impressed, they summon prime movers before they continue walking the earth.

Yes, to be able to do their jobs, travelers do what experimenters do. The only difference is their activities result in "an ecosystem selection", rather than developing apps or conducting experiments.

Prime movers are seasoned travelers who have developed insights into cultural terraforming during their encounters with absurdities. While the focus of the travelers is the absurdities, the focus of the prime movers is the troublemaker subset of the subjects in absurdity bearing ecosystems (i.e., "the garden"). They measure their success by how far subjects and troublemakers have moved from their first bearings. Subjects may become the subjects of experiments, but they can become the subjects of cultural terraforming (r)evolutions too. Similarly, troublemakers may stay as troublemakers who eventually sour and divert back to their subject roots… or go forward to become enthusiasts who apply to School's Out cycles. If they prove to be worthy, they can become experimenters after they graduate. Experimenters are the protagonists of our NoRM adventures. Songs are sung after them. Stories are written about their deeds.

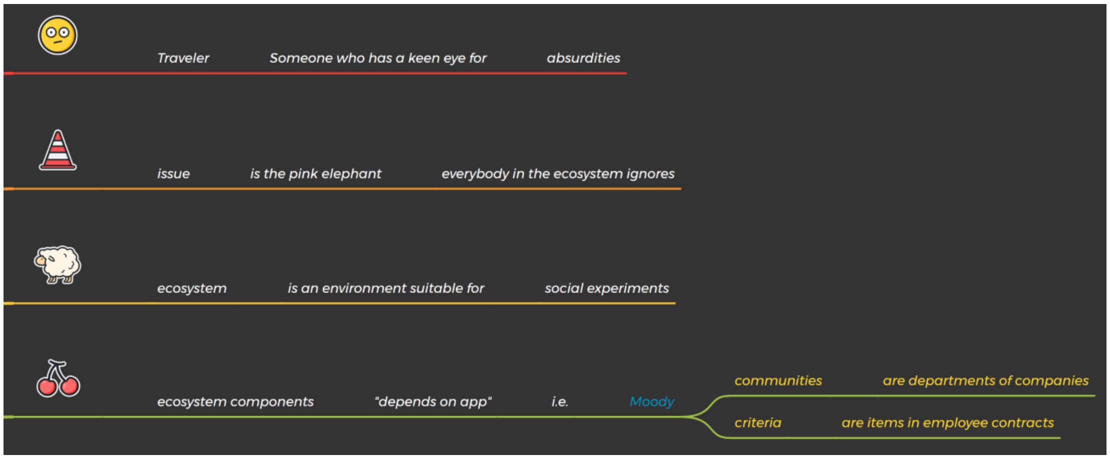
No, soured troublemakers never become enablers. These are the possible ends of subjects. They either go pro or become the forever unsuccessful opposition. It kind of reminds me of two party political systems. No matter who you vote for, you lose. And regardless of being selected or not, they win. What a strategy!



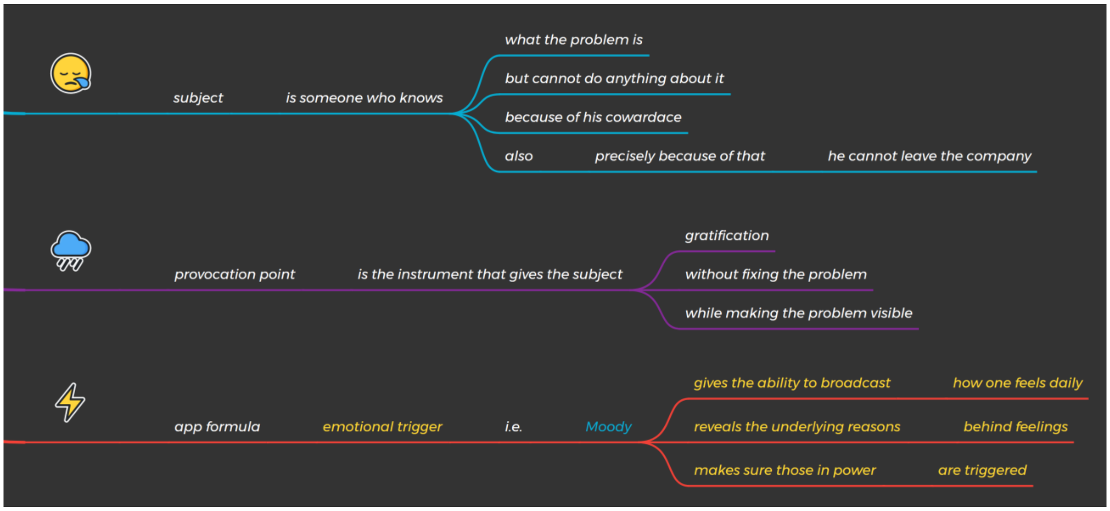
After a School's Out cycle is complete, we may carry on with a new cycle, but also, we can do many other things. We can strengthen links with the world via initiating a Think Tank or Part-Time Entrepreneurship Outlet. We can also dive deep into the realm of an Interest Groups. Of course, we can continue creating media whether it be a story, novel, film, TV series or a game. The last but not least, we can help an Intelligence Agency conduct a challenging operation.

That being said, we can divert our attention inward too. We can make the School's Out Headquarters better or create one. SOHQ isn't a singular entity. There are many SOHQ(s)… meaning ours (as in the first one) doesn't have complete control over everything. We can be challenged… even taken out of the picture. Regardless, all SOHQ(s) have a similar organization even if their focus and expertise may differ. At the core of the headquarters lie Computer Science Unit. Around it we have an additional computer science unit (gaming) and social sciences units (anthropology, psychology, philosophy). We also have communication (journalism) and strategy (criminology, military science) units. Finally, we have art units (cinema, literature, media production).

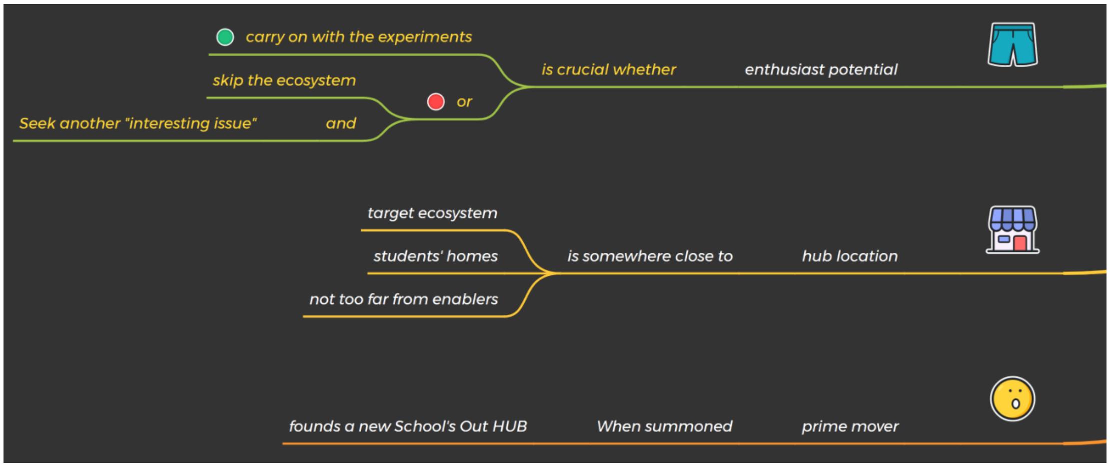
More on Interesting Issues


Traveler
Someone who has a keen eye for
absurdities
issue
is the pink elephant
everybody in the ecosystem ignores
ecosystem
is an environment suitable for
social experiments
ecosystem components
"depends on app"
i.e.
Moody
communities
are departments of companies
criteria
are items in employee contracts

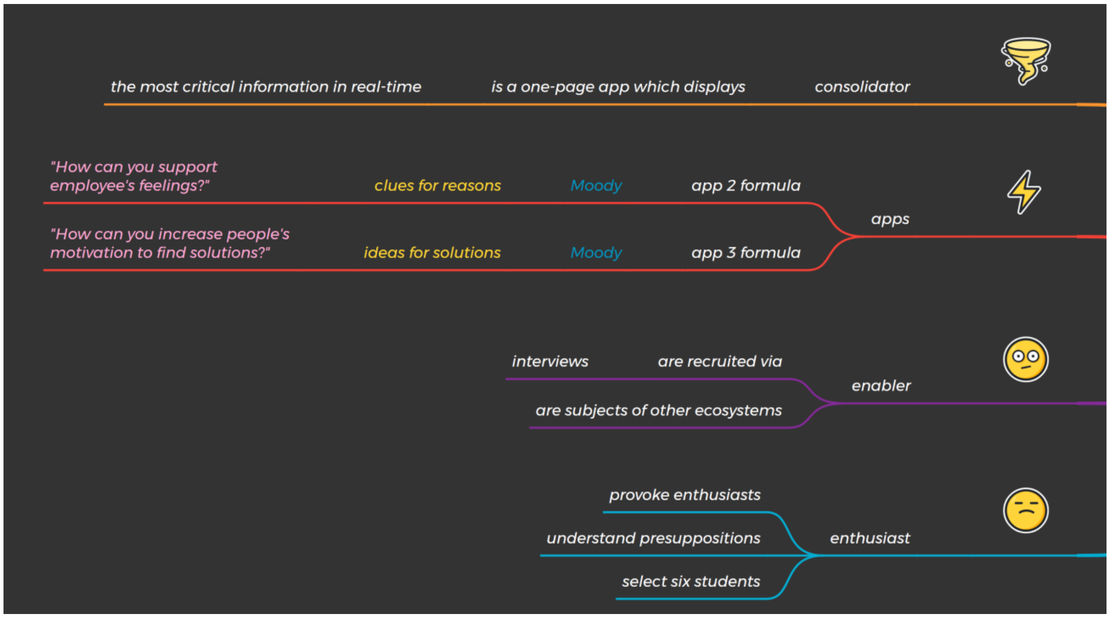
The adventure always begins the same way. Someone with a keen eye for social absurdities, traveler, finds an interesting issue… focuses on the associated ecosystem and thinks about possible app formulae while observing the subjects therein… thinking, "Is this worthy of a School's Out HUB?"


subject
is someone who knows
what the problem is
but cannot do anything about it
because of his cowardace
also
precisely because of that
he cannot leave the company
provocation point
is the instrument that gives the subject
gratification
without fixing the problem
while making the problem visible
app formula
emotional trigger
i.e.
Moody
gives the ability to broadcast
how one feels daily
reveals the underlying reasons
behind feelings
makes sure those in power
are triggered

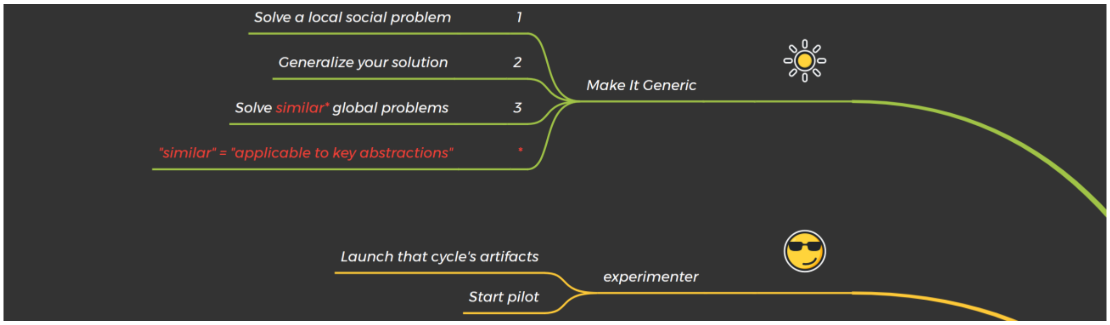
Subjects are the ones who make apps for troublemakers happen. They are powerless people who know what is wrong… who can fix it given the chance… but too chicken shit (Excuse my French) to do anything about it. While being able to do something, they cannot get over their cowardice. So, they fake idealized versions of themselves. Since they are always with the likes of them, they easily believe their own lies. They see people who look up to them as the clues of their greatness which is the other way round. That's when we come into the picture. We exploit that psychological condition. Subjects always have provocation points which are their buttons begging to be pushed. The emotional triggers we must formulate as apps push these buttons, so to speak… and turn subjects into exploitation entities.



Of course, finding an interesting issue doesn't cut it. The associated ecosystem must be exploitation ready. That it must be home to eager hosts who are searching for ways to get even. Hosts who are in between their point A(s) and point B(s)… eyeing their point B(s) with strong emotions… hosts who have become willing subjects!


the most critical information in real-time
is a one-page app which displays
consolidator
"How can you support employee's feelings?"
clues for reasons
Moody
app 2 formula
"How can you increase people's motivation to find solutions?"
ideas for solutions
Moody
app 3 formula
apps
interviews
are recruited via
are subjects of other ecosystems
enabler
provoke enthusiasts
understand presuppositions
select six students
enthusiast

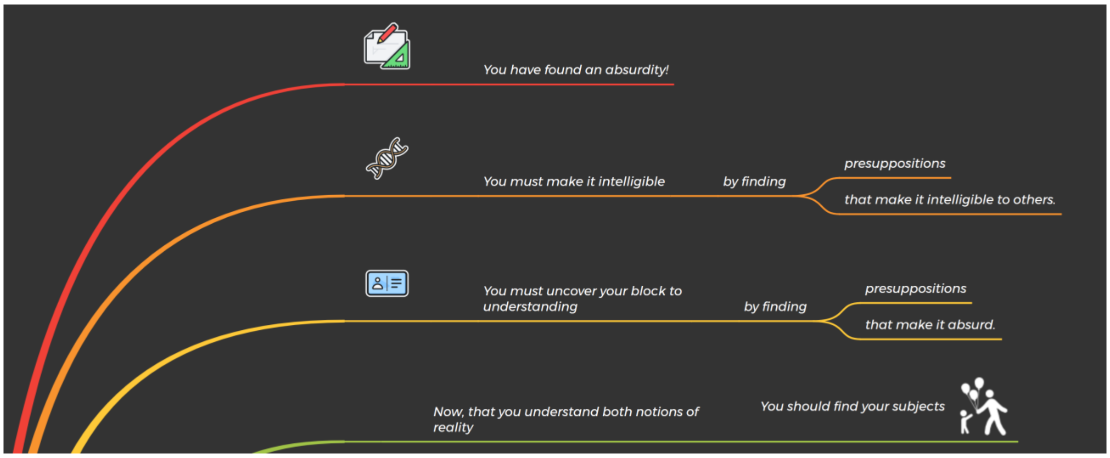
Now it's time to get busy. We will recruit troublemakers, bond with enablers and at the same time formulate the remaining apps and find a way to consolidate the information all those apps spit out. Wow! We are busy bees or what? Don't forget, you don't make up those apps after going to the desert. Instead, you become one with the people. You observe. You let the adventure uncover a path for you. There you find ideas. So, ideas are not coming from you. That's the trick here. What comes from you are their formulations and applications.


Solve a local social problem 1
Generalize your solution 2
Solve similar* global problems 3
"similar" = "applicable to key abstractions" *
Make It Generic
Launch that cycle's artifacts
Start pilot
experimenter

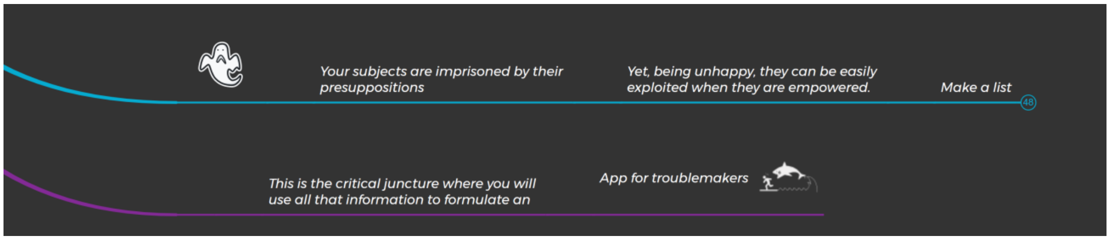
Finally, there is something you should never forget while formulating those apps. They must be generic and very easy to customize. Also, they must be linkable (their key abstractions must be closely related), so that apps generated at that cycle of School's Out could be deployed together to initiate maddening moves. This feature also makes consolidators applicable. Because consolidators don't do anything except display whatever the interlinked apps bring to the table, so to speak. That information is what leads you to the paradigm that cycle introduces too.

One way of looking at linkable apps is imagining three good apps doing just one thing. Then, imagine an app doing all those three things only to become a bad app. There! That's how you can identify three linkable app ideas.

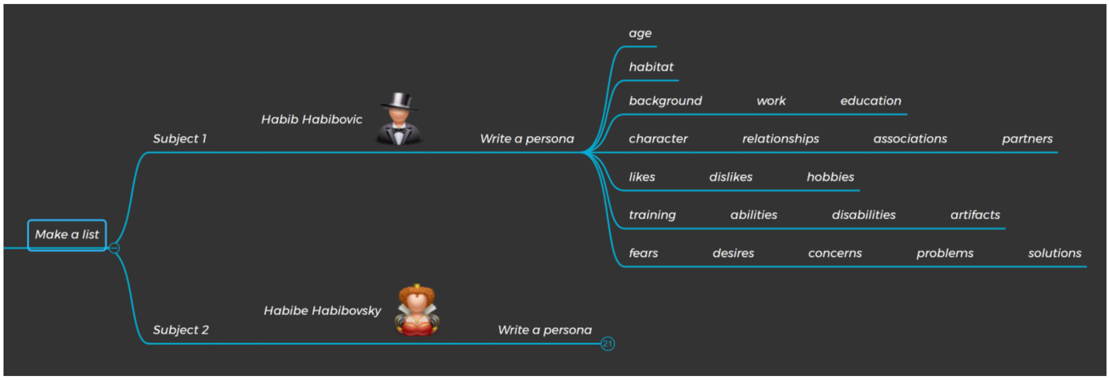
More on Zeroth Questions


You have found an absurdity!
You must make it intelligible
by finding
presuppositions
that make it intelligible to others.
You must uncover your block to understanding
by finding
presuppositions
that make it absurd.
Now, that you understand both notions of reality
You should find your subjects

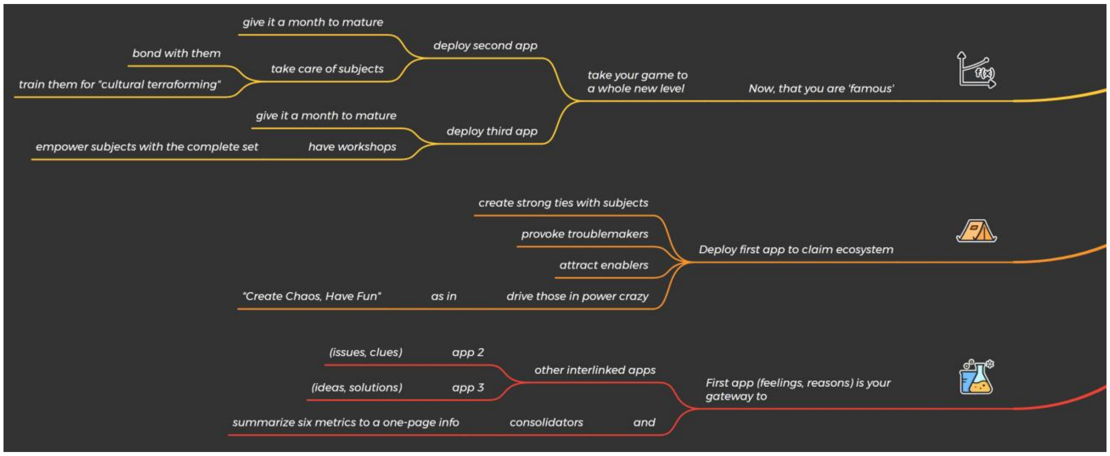
This is the prerequisite skill to get into the wild world of social experiments. You must be able to spot absurdities. It must be like breathing for you, effortless. Unless you have a keen eye on them, you won't get very far. If you do, you must learn a few tricks… and we are here for you. First, you must be able to make those absurdities intelligible. You must be able to uncover their and your presuppositions… and merge horizons. With that kind of power there is no limit to "what you can do. Granted you find your subjects, that is.


Your subjects are imprisoned by their presuppositions
Yet, being unhappy, they can be easily exploited when they are empowered.
Make a list
48
This is the critical juncture where you will use all that information to formulate an
App for troublemakers

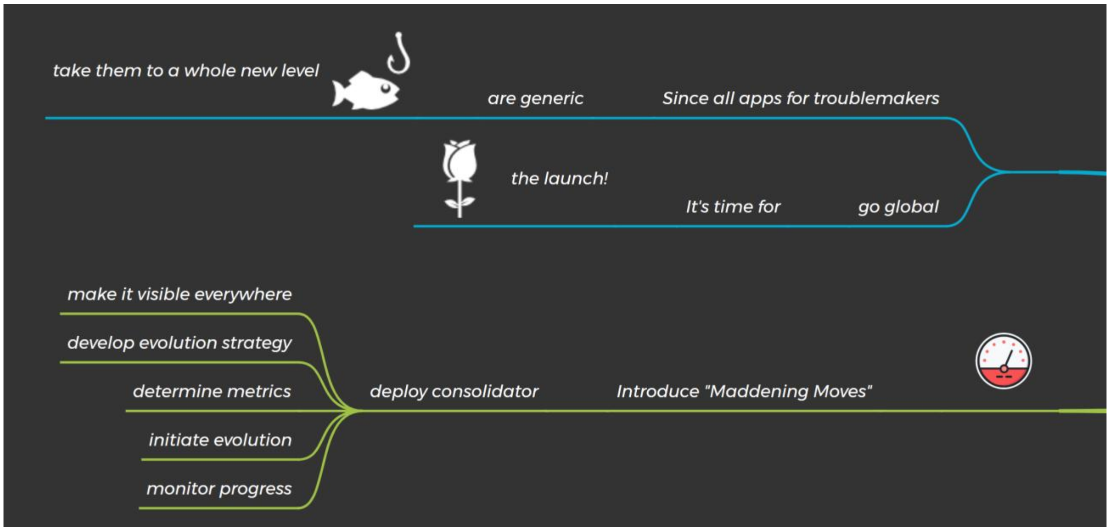
The good thing about subjects is they are unhappy. They pretend to be this way or that way, and they may be phenomenal at faking. Still, whatever's bothering them won't leave them alone… and it's a good thing. This is precisely how you will gather information that will help you formulate your apps and conduct your experiments.



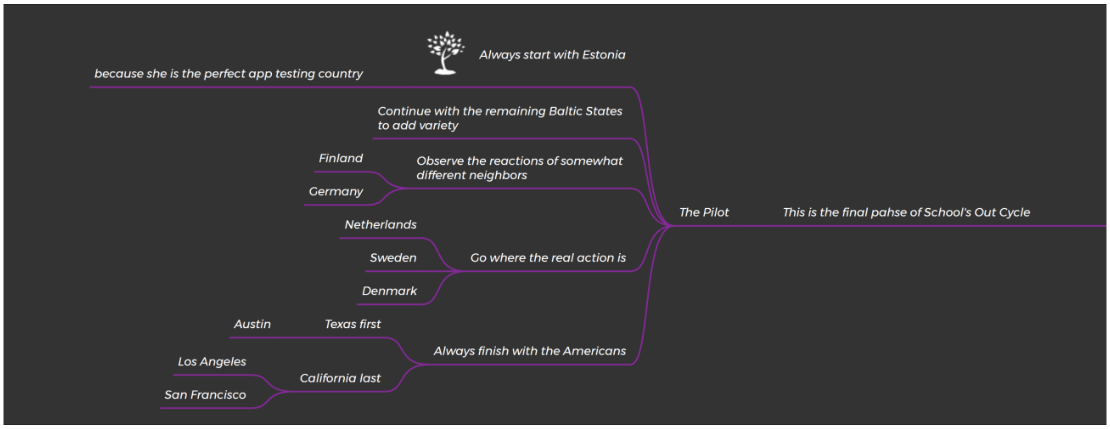
You must pay close attention to your subjects. The practical way to go about it is having a folder for each candidate. Gathering information for long periods of time is your edge. No one in his right mind observes a subject for too long. It's the fact of modern life. And that's why nobody has a clue about what is wrong. I must thank my military service for giving me that kind of patience. Otherwise, such an endeavor would drive me crazy. However, that training gave me the ability to stand still for a long time and make a killing at the best time… like Rambo.


give it a month to mature
deploy second app
bond with them
take care of subjects
train them for "cultural terraforming"
take your game to a whole new level
Now, that you are 'famous'
give it a month to mature
deploy third app
empower subjects with the complete set
have workshops
create strong ties with subjects
provoke troublemakers
Deploy first app to claim ecosystem
attract enablers
"Create Chaos, Have Fun"
as in
drive those in power crazy
(issues, clues)
app 2
other interlinked apps
(ideas, solutions)
app 3
First app (feelings, reasons) is your gateway to
summarize six metrics to a one-page info
consolidators
and

As soon as the first app has a fanbase, you must deploy the other two to strengthen the growing tensions. While it is possible to have a different variety of apps, I always follow a pattern. My first app is about the hidden truth whatever that may be. The others make it more prominent by adding clues and public support. Think about a three-piece closing *technique. First, you introduce something new. Then, you show why it is important. Finally, you make the sale, so to speak, by showing how popular it is.*

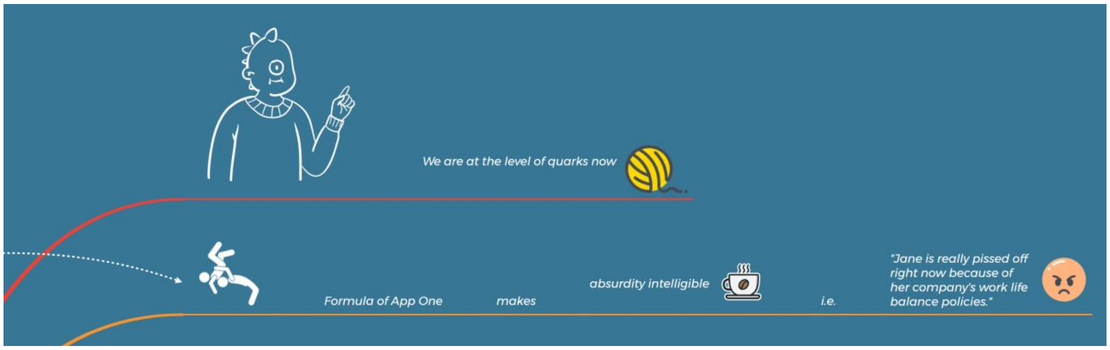

take them to a whole new level
are generic
Since all apps for troublemakers
the launch!
It's time for
go global
make it visible everywhere
develop evolution strategy
determine metrics
initiate evolution
monitor progress
deploy consolidator
Introduce "Maddening Moves"

When you want to take things to a whole new level, there are two things you can do. You can initiate a maddening move locally or you can take your experiments to the world's stage. Initiating a maddening move requires one more thing, a consolidator. A one-page app which consolidates information gathered by the interlinked apps and makes it unavoidable for everybody to ignore it. Taking experiments globally means just repeating what you have done locally somewhere else. While one can pick any country, my madness has a method.

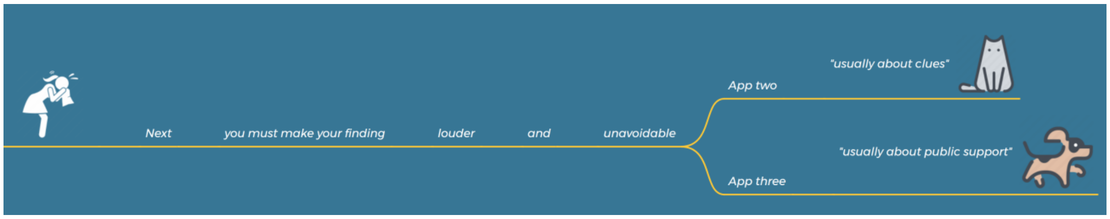

Always start with Estonia
because she is the perfect app testing country
Continue with the remaining Baltic States to add variety
Finland
Observe the reactions of somewhat different neighbors
Germany
The Pilot
This is the final pahse of School's Out Cycle
Netherlands
Sweden
Go where the real action is
Denmark
Austin
Texas first
Always finish with the Americans
Los Angeles
California last
San Francisco

I always start in Estonia, because that whole county is a Silicon Valley… with a better attitude. Then, I continue with a selection of European countries and always finish in the US. There are three types of European countries. First, we have a country with educational tendencies, Finland. Then, we have a country with philosophical tendencies, Germany. Finally, we have countries with warm blooded people living in cold places, Netherlands, Denmark and Sweden. In the US, I focus on entrepreneurial tendencies first, Austin. Then, I continue with a mixture of entrepreneurship and entertainment, Los Angeles and San Francisco. Of course, what makes world dominion (!) easy is making apps for troublemakers small, generic (lacking domain specific functionality) and very easy to customize (via customizing two lists).

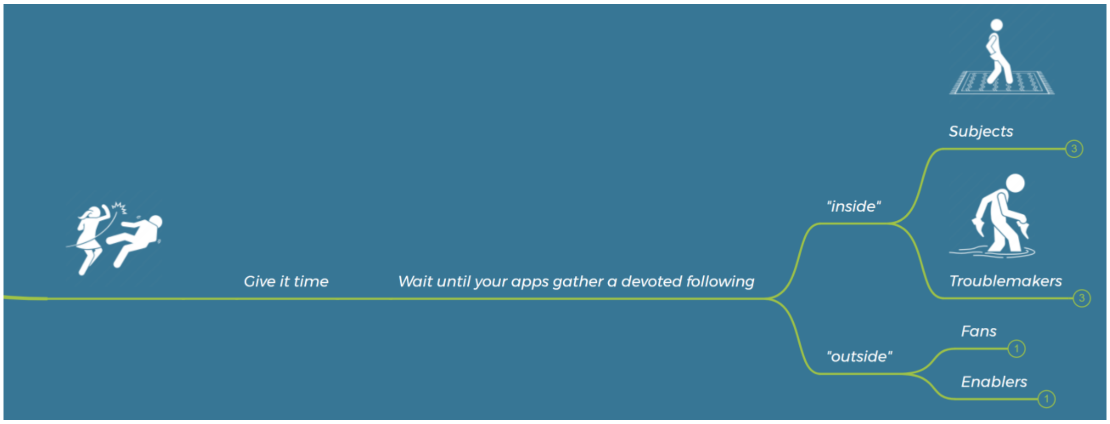
More on Formulating App Concepts


We are at the level of quarks now
Formula of App One
makes
absurdity intelligible
i.e.
"Jane is really pissed off right now because of her company's work life balance policies."

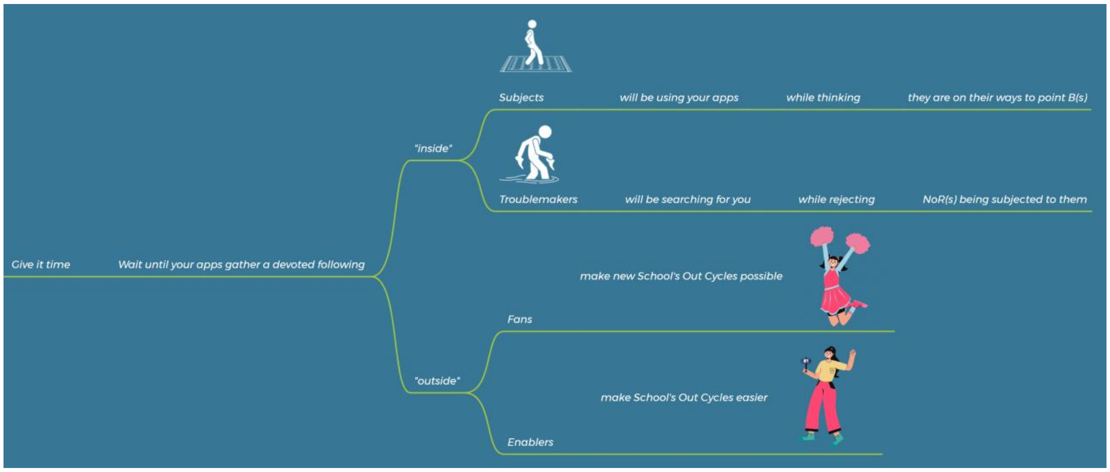
This may be tricky at first, but it must become second nature, or you will never be a social experiment mobile app programmer (Semap), my friend. You must look at the absurdity excluding your presuppositions meaning you must assume (!) that the absurdity isn't absurd at all. What's absurd is the way you are looking at it. Once that barrier has been lifted, you seek presuppositions that make the absurdity intelligible. For example, "employees bitching but not leaving" is intelligible when you see them as "greedy and… not so brave". Then, it makes sense to stay in the shit while complaining about it.



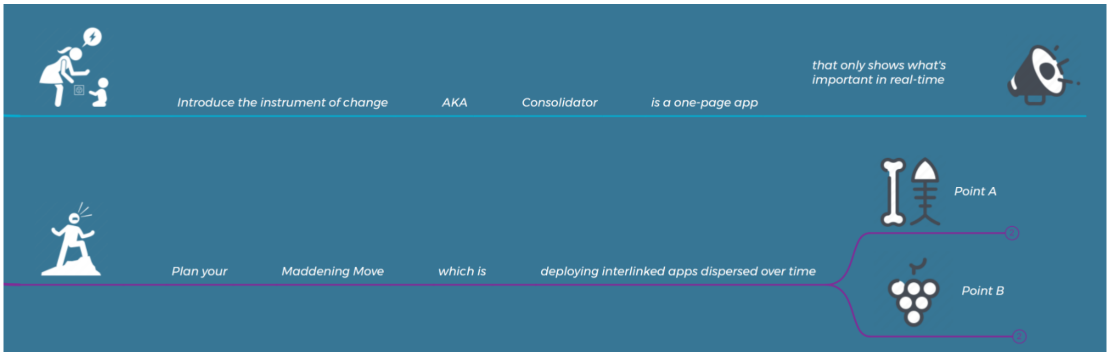
Once you find how your subjects tick, you can automate it with your first app. All the other apps that come afterwards must make that state of being stronger in one way or another. As I said before I go with clues and public support, but you can come up with better schemes.



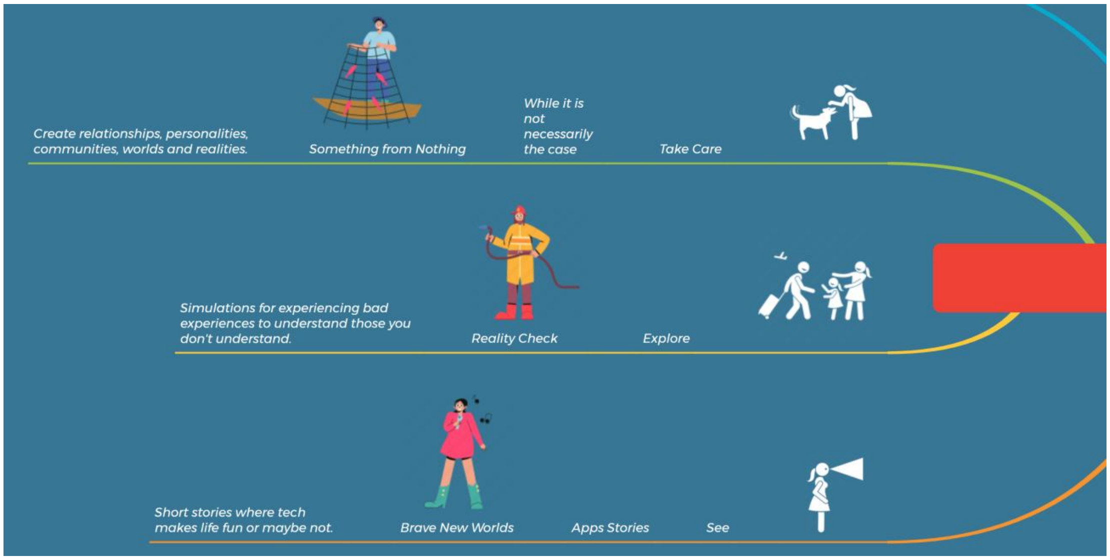
Timing is of the outmost importance. Come to think of it, "time appreciation" is the most important skill one may develop regardless of the circumstances. When you have a subject base, make sure the troublemakers among them reveal themselves. Also, make sure you can identify enablers and fans outside of the experiment zone. Troublemakers drive School's Out HUB(s). Enablers help you spread the disease. Fans do the same… only they operate beyond zones.


Give it time
Wait until your apps gather a devoted following
"inside"
Subjects
will be using your apps
while thinking
they are on their ways to point B(s)
Troublemakers
will be searching for you
while rejecting
NoR(s) being subjected to them
"outside"
make new School's Out Cycles possible
Fans
make School's Out Cycles easier
Enablers

An important aspect of this process is knowing where you are meaning how far off your subjects on their way to their Point B(s). Similarly, you must measure the enthusiasm levels of your enablers and fans. Can they be exploited to promote an ongoing School's Out cycle? Can they be triggered to found another School's Out HUB?

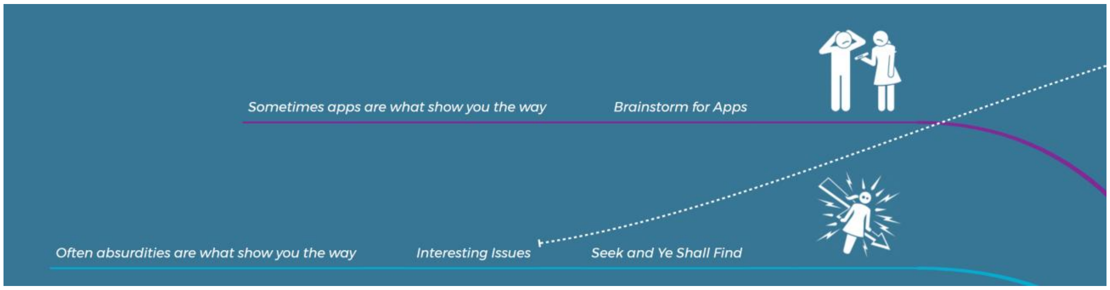

Introduce the instrument of change
AKA
Consolidator
is a one-page app
that only shows what's important in real-time
Plan your
Maddening Move
which is
deploying interlinked apps dispersed over time
Point A
Point B

As you might have guessed, I think in terms of circles just like the case it was in the TV series, "Dark" where past, present and future are all one thing. I consider this as a fact. So, when I'm there formulating apps, on another side of my brain a consolidator concept is being born. App one, two and three expressed in a single sentence… That's your consolidator, pal! Also, remember you exploit your subjects between points A and B… meaning your cultural terraforming must fit within a similar interval. The only difference is, this time point A and B refer to the realities and possibilities of your subjects' community.

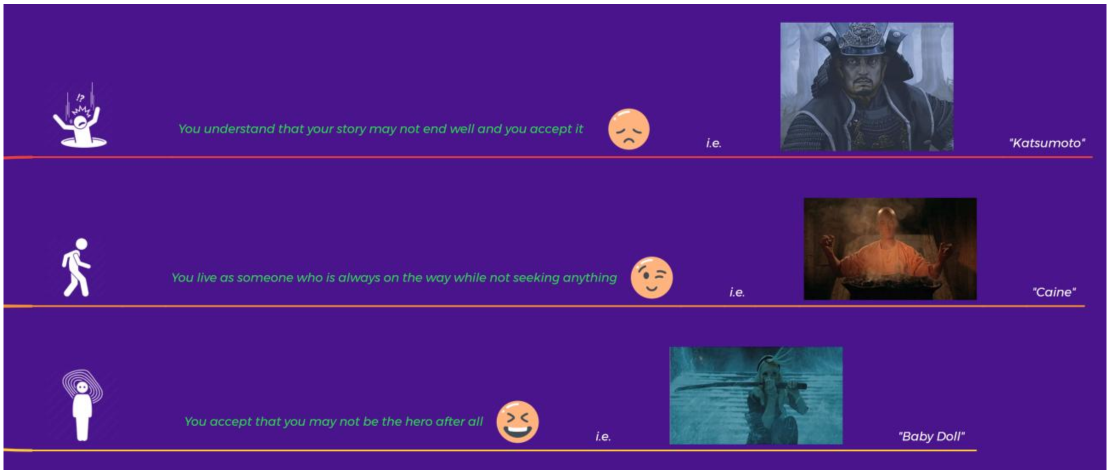

Create relationships, personalities, communities, worlds and realities.
Something from Nothing
While it is not necessarily the case
Take Care
Simulations for experiencing bad experiences to understand those you don't understand.
Reality Check
Explore
Short stories where tech makes life fun or maybe not.
Brave New Worlds
Apps Stories
See

You get it, don't you? Everything we do can be good… or bad. It all depends on who you are or what you are trying to do. Also, they can be applied disjointly or together. While they are disjointed, you can have subjects to link them or not. See, we have so many choices which implies you should know what you are doing. App stories are good for quick thrills. They give their audience a sense of feasible, tangible, possible… escape. Bad experience simulations are the complete opposite of that. They give their audience the taste of ugly, inconvenient truths. Together, they create the borders of reality of that group of subjects.

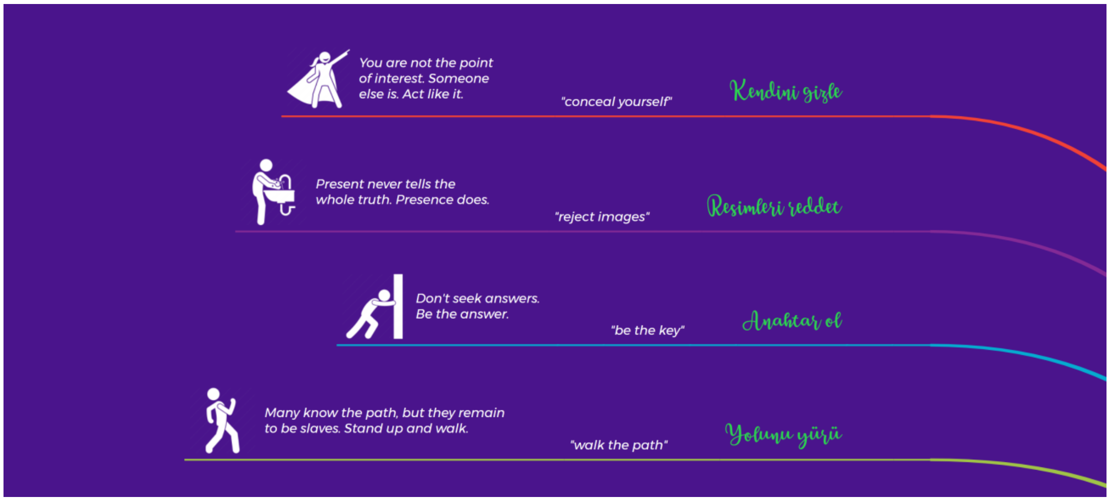
Alternate reality platforms, on the other hand, should either be applied on their own or they must be used to support a certain group in that ecosystem. My favorite group to support is my group! I use alternate reality platforms for experimenters. Don't let this discourage you, though. They are the perfect tools for creating hells on earth as well.


Sometimes apps are what show you the way
Brainstorm for Apps
Often absurdities are what show you the way
Interesting Issues
Seek and Ye Shall Find

More on Prime Movers


You understand that your story may not end well and you accept it
i.e.
"Katsumoto"
You live as someone who is always on the way while not seeking anything
i.e.
"Caine"
You accept that you may not be the hero after all
i.e.
"Baby Doll"

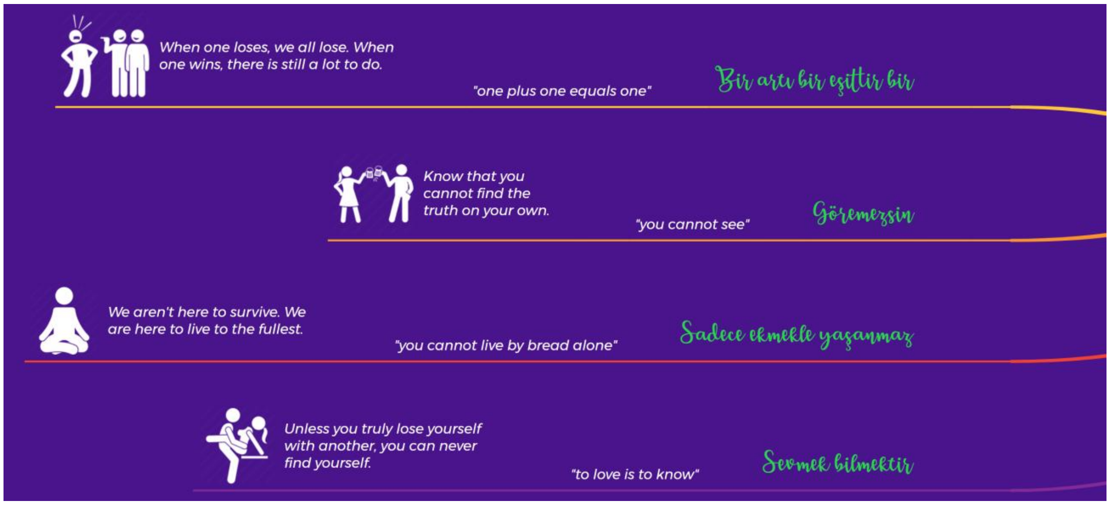
Yes, you heard me. You have to assume the opposite of what everybody else assumes. That your story may not end well. You may not even be the hero of your stories. That's why being always on the way while seeking nothing is important. It sounds wrong, doesn't it? Being on the way and yet, not seeking an end, closure… something to show for… Daily truths or rather facts… even maybe, just information can be found in the normal way, but deeper truths… truths that reveal who we are can only be found in this strange manner. Think about quantum mechanics. It's weird too. Anything at the core of things / beings is not straightforward. It's weird. Because you aren't meant to spend time on them. You are meant to spend time on your life. You are meant to live.


You are not the point of interest. Someone else is. Act like it.
"conceal yourself"
Kendini gizle
Present never tells the whole truth. Presence does.
"reject images"
Resimleri reddet
Don't seek answers. Be the answer.
"be the key"
Anahtar ol
Many know the path, but they remain to be slaves. Stand up and walk.
"walk the path"
Yolunu yürü

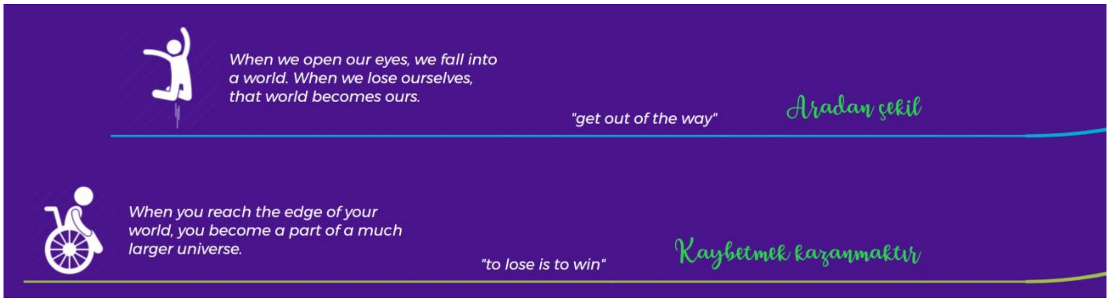
Every culture has a way to preserve this strange wisdom. Mine has ten rules.

- *A person who answers the call for adventure (a would-be hero) must conceal herself… meaning she must put a distance between her and her deeds. He must be brave… noncompromising… and yet, humble.*
- *She must reject images of any kind, of oneself, of truth… Nothing about should be stiff, unchanging, absolute.*

- *She must not seek answers… coming from elsewhere, when the matter at hand requires guts. She must be the answer. She must stand up and fight.*
- *Knowledge is not wisdom. You may know the options or even the best course of action. If you are just thinking about them, you are a coward. Worse yet, if you are thinking in terms of optimizations and preparations within your horizon, you are dust.*

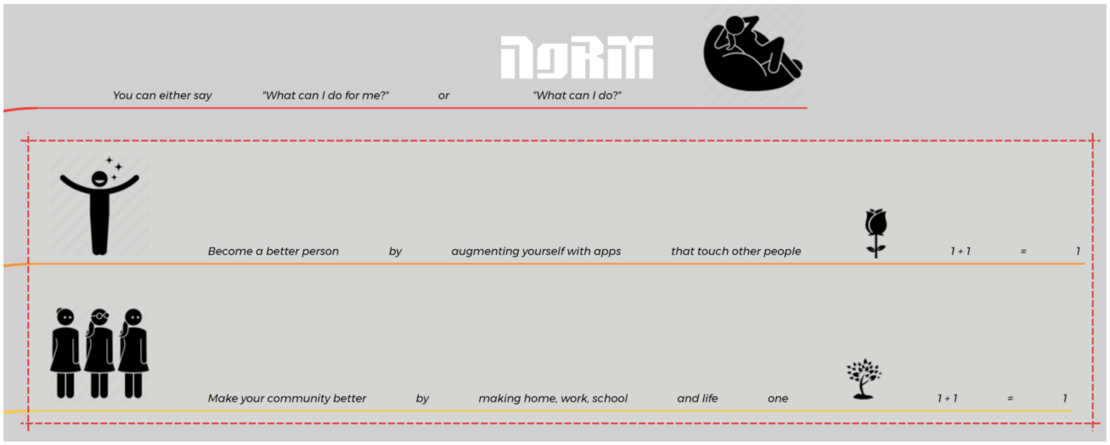

When one loses, we all lose. When one wins, there is still a lot to do.
"one plus one equals one"
Bir artı bir eşittir bir
Know that you cannot find the truth on your own.
"you cannot see"
Göremezsin
We aren't here to survive. We are here to live to the fullest.
"you cannot live by bread alone"
Sadece ekmekle yaşanmaz
Unless you truly lose yourself with another, you can never find yourself.
"to love is to know"
Sevmek bilmektir

- *Homo sapiens has a tendency to isolate oneself from others, either in groups or own his own. The moment you do that you are lost. We are meant to do it together. Otherwise, it's a different kind of cancer.*
- *Not only that, but we also cannot even comprehend the truth on our own. It is something that can be uncovered together… like living well.*
- *Preoccupation about means, especially money, but also knowledge and skills to do this or that blind us after a point. Too much focus on your daily bread turns you into pimps and hookers.*
- *There are doors that can be opened by courage, knowledge and wisdom, but the hardest door to open has only one key, "your unselfish love towards someone or something" like the love of a child towards his father and mother.*


When we open our eyes, we fall into a world. When we lose ourselves, that world becomes ours.
"get out of the way"
Aradan çekil
When you reach the edge of your world, you become a part of a much larger universe.
"to lose is to win"
Kaybetmek kazanmaktır

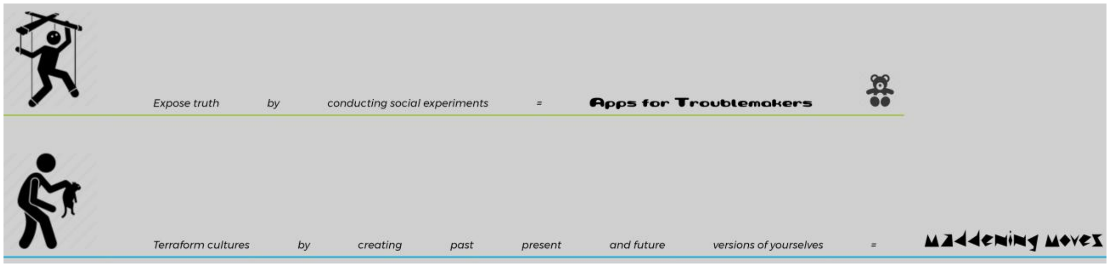
- *Sometimes you are the barrier that blocks your understanding. You can let it happen only by letting go of yourself.*
- *This is a pickle. Testing your limits to the roof… to the point you break… is considered failure by many, but it is the ultimate victory. Because our victories are not meant to last. We are transitionary. We must find ways to salute the ones that come after us. We must give them a worthy cause.*

More on Notion of Reality Management

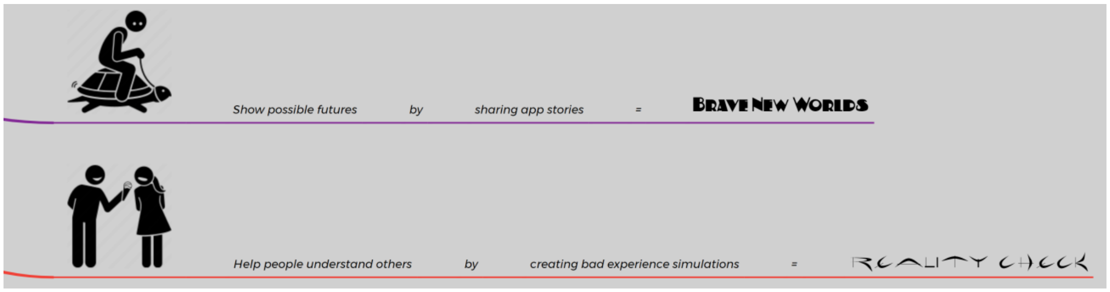


So, how do we summarize our crazy scheme in minutes? I guess, it all comes down to two questions. The way you formulate your unspoken question in your head reveals who you are… and you are either good or not so good. When I look at the world, I am not even the last thing that comes to my mind let alone what's in my wallet. I have always been like that. Also, there is something else… something peculiar about me. I feel at home. I always feel at home… (points his heart) in here. Perhaps, that's why I never synch with friends… or lovers for that matter. I don't seek anything. I am already… home.

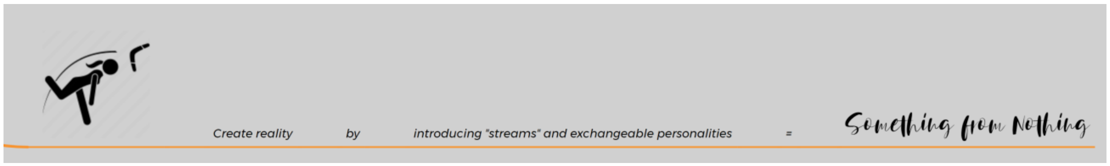
Starting from that point, there are two paths we can take. More often than not, they are intertwined. So, which one you pick only determines the tone not the ends. I no longer pick the first question. That was a question of my youth… when I wasn't even in high school. Later, I asked only one question, "How do… we… get better?" When you know even the dumbest, most evil son of a bitch is not so much worse than you, you understand. Nobody is so much better than you either. We are blessed or cursed, granted we are together. Alone we are less than nothing… just a dust in the wind.



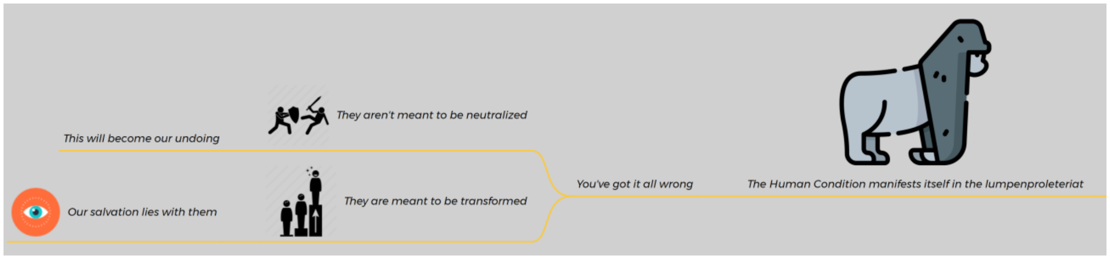
Truth has a habit of hiding. She is always shy. To be able to uncover it, we will use mobile apps designed for social experiments. These apps will often come in threes, and they will have everything necessary to be interlinked and consolidated. When they are strategically deployed dispersed over time, they can be used for cultural terraforming.



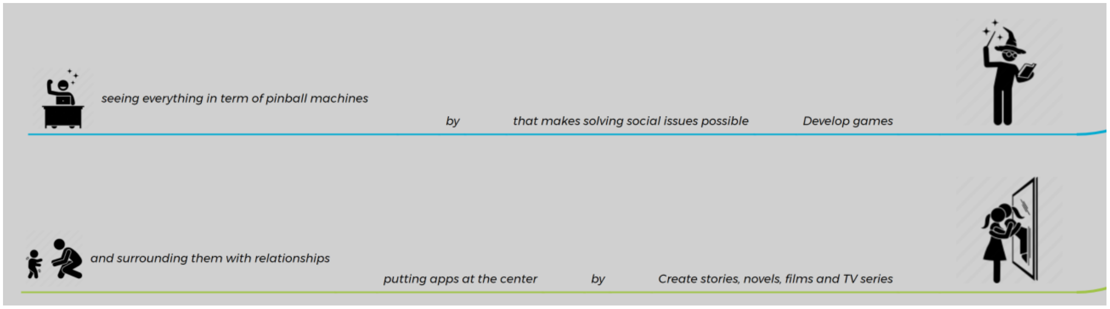
We do have instruments of evolution that go well with every décor. You can use them on their own, in conjunction with School's Out HUB(s) or to prep ecosystems for alternate reality applications. In the old days human beings weren't programmable, nay, they were programmable to a much lesser extent. Nowadays the distinction between (wo)man and the machine is a tricky one. Imagine the days when business modeling, the UML way, was first surfaced. There you would look at how things are done and find ways to automate different aspects of a flow or a process. These days I look at humans (!) and augment them with apps to automate certain aspects of them. And, you know what? They just love it. Human beings are suckers of escaping from their responsibilities. You can hardly find a few who would willingly swallow the red pill (The Matrix).



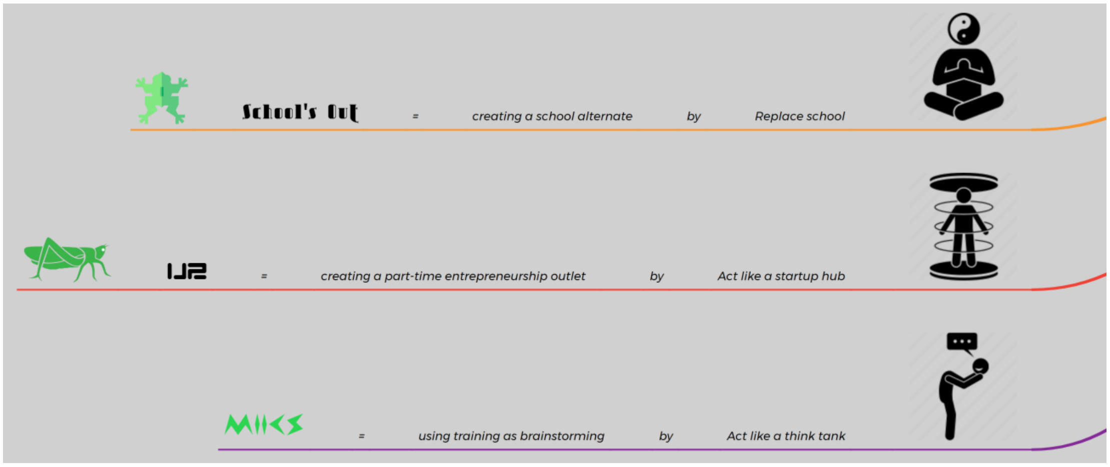
While I like to use SFN to empower experimenters and give them an easy access to travelers and prime movers, it is meant to be used on its own… without a direct contact between it and the rest of the NoRM components. Why? Because it's a super app, silly. When you have SFN, you don't need anything else.


This will become our undoing
They aren't meant to be neutralized
Our salvation lies with them
They are meant to be transformed
You've got it all wrong
The Human Condition manifests itself in the lumpenproleteriat

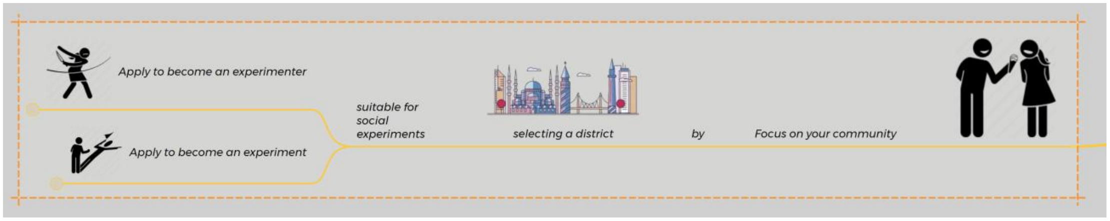
Another trick is reading this opportunity as another way to feel superior. This isn't it. This isn't it at all. We are aiming for the lowest strata of communities, losers, misfits, fanatics… Imagine this as Jesus recruiting hookers and thieves once again. Only through their transformation, there is a tomorrow. Otherwise, we are condemned to live our lives in bondage, behind doors, trying to make ourselves believe that this is indeed paradise.


seeing everything in term of pinball machines
by
that makes solving social issues possible
Develop games
and surrounding them with relationships
putting apps at the center
by
Create stories, novels, films and TV series

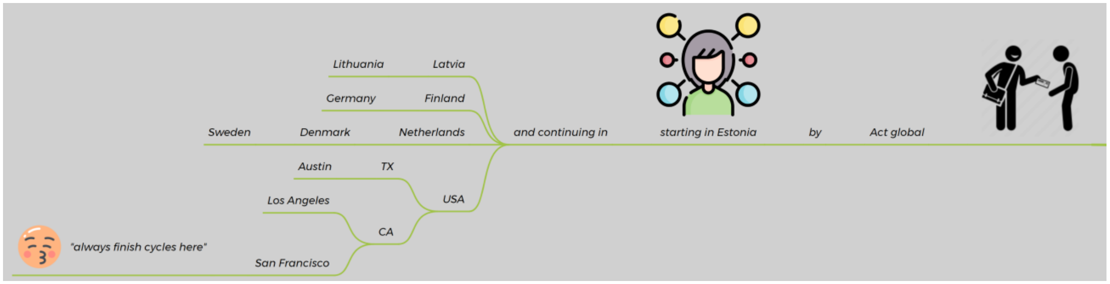
My favorite word is German, "Gesamtkunstwerk". It means making seemingly irrelevant things… one. To me art, science… work, school, home… life… are all one thing. Making one of them great and sacrificing the others won't give you a way out. Here the weakest chain in the link is the link. So, our training, apps, books… stories, novels, films, TV series… and games are all one thing. Just like we all one. You know, one plus one is equal to one, you fool!


School's Out
=
creating a school alternate
by
Replace school
IJR
=
creating a part-time entrepreneurship outlet
by
Act like a startup hub
MIICS
=
using training as brainstorming
by
Act like a think tank

As you might expect, I'm a sucker for trinities. Here we have another one. School's Out HUB is home to MIICS which is linked to the outsiders in the form of a part-time entrepreneurship outlet. Also, HUB is linked to the other hubs via School's Out Headquarters.


Apply to become an experimenter
Apply to become an experiment
suitable for social experiments
selecting a district
by
Focus on your community

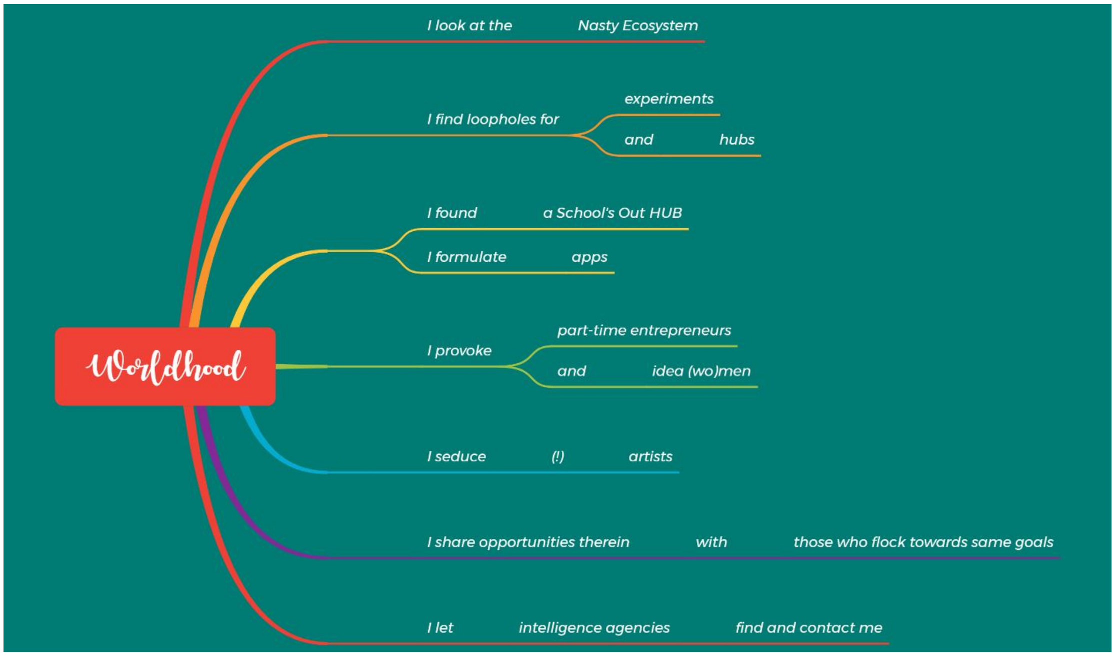
This adventure begins with a simple message. If you are a troublemaker, you can apply to become an experimenter. If you are a subject, you can apply to become an experiment. What was the difference. A troublemaker can walk out of a nasty ecosystem and a subject always finds reasons to not to.



Every School's Out cycle ends the same way. It strengthens its ties with reality on a global scale.

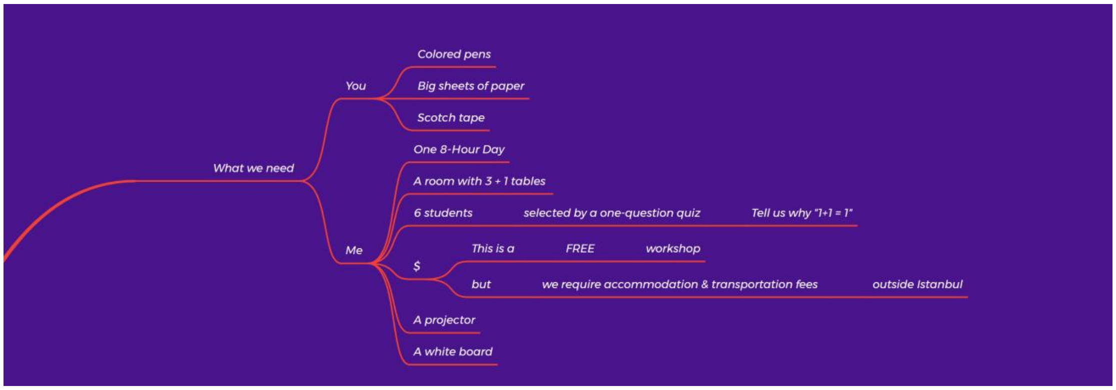
In the Beginning

They say, "in the beginning was the word". I say, "in the beginning was the look". When you look at your world with care and concern… and never give up, doors will open to you. What's only pitch black for the others is the land of infinite possibilities to you. Nothing will get you down. Nothing will make you run in the opposite direction. There is a universe within each one of you. Unless you turn yourselves into things, you can never be cursed. You are blessed from day one.

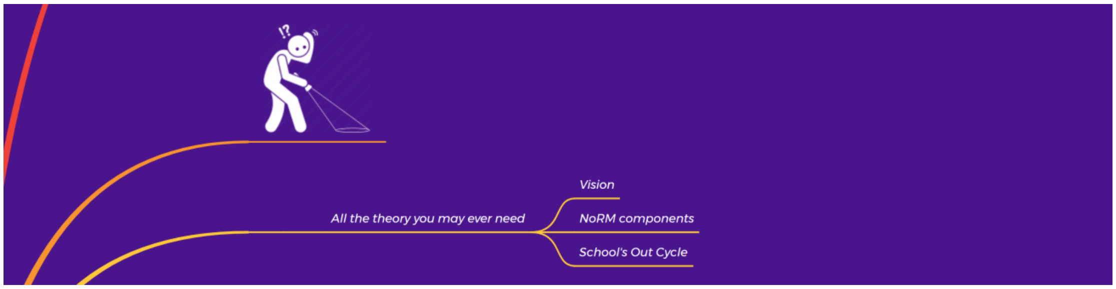


The Art & Science of Preaching – Part One

I learned early in life that if you give away stuff for free, people will take it. That's my favorite way to open otherwise very tightly shut doors. People are… stupid. When I give something, actually I'm the one taking stuff. Words and actions… they rarely match in our post-modern world. Did you not still understand this? NEEHEEHEEHEEHAHAHA!

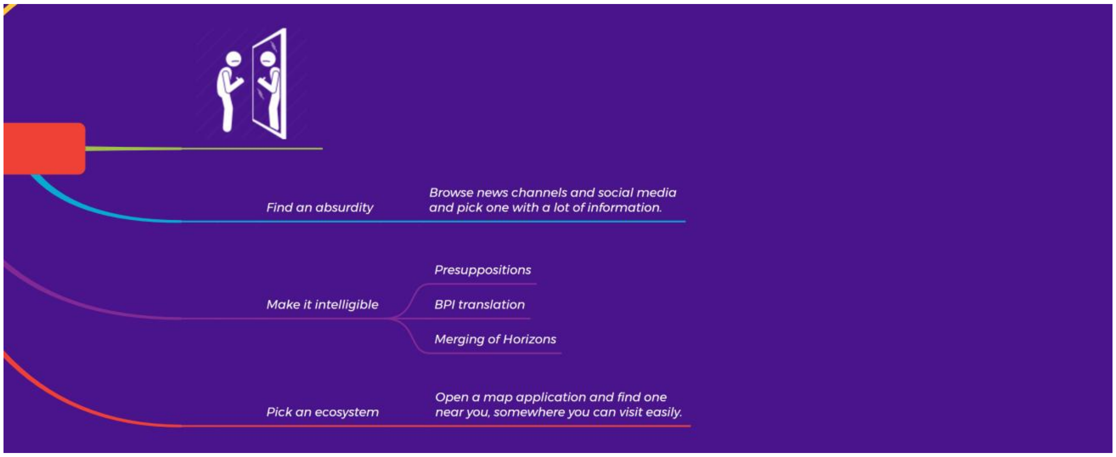


I don't amateurish gifts, by the way which is something you can deduce from the gift pack this book belongs to. What? "Do I take stuff from you too?" Of course, I do! You get goodies and I get future troublemakers. Win-Win!



I make sure that I teach you all the critical aspects of the theory. Who knows? Maybe I won't be around, and you will have to fight the dragons on your own. We are one, you and I, my friend. When one falls, the other stands up and fights.

The best way to transfer know-how is to create a shared vision, make the big picture understandable / actionable and help candidates experience every nook and cranny of the training package.

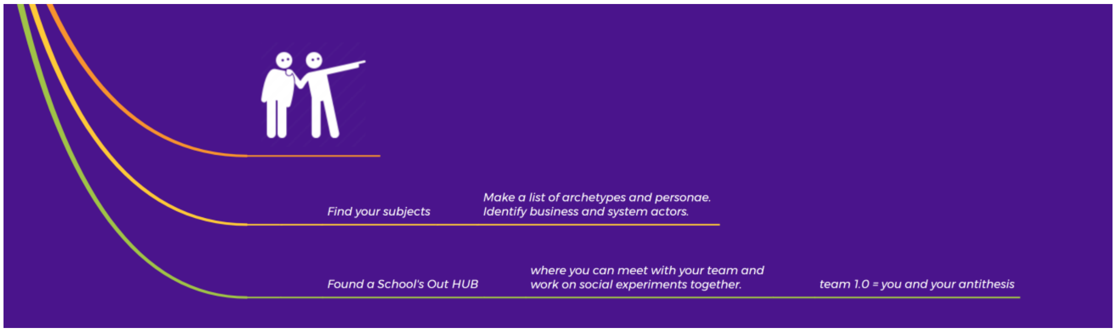

Find an absurdity
Browse news channels and social media and pick one with a lot of information.
Make it intelligible
Presuppositions
BPI translation
Merging of Horizons
Pick an ecosystem
Open a map application and find one near you, somewhere you can visit easily.

[Traveler] wears the cultural anthropologist hat. The most important trait of a cultural anthropologist is her ability to fake being a local. She doesn't raise suspicions when she is the one observing everybody and taking notes. In that regard, she is like a spy. While she is faking it, she uses this advantage (!) to find absurdities. People tend to conceal

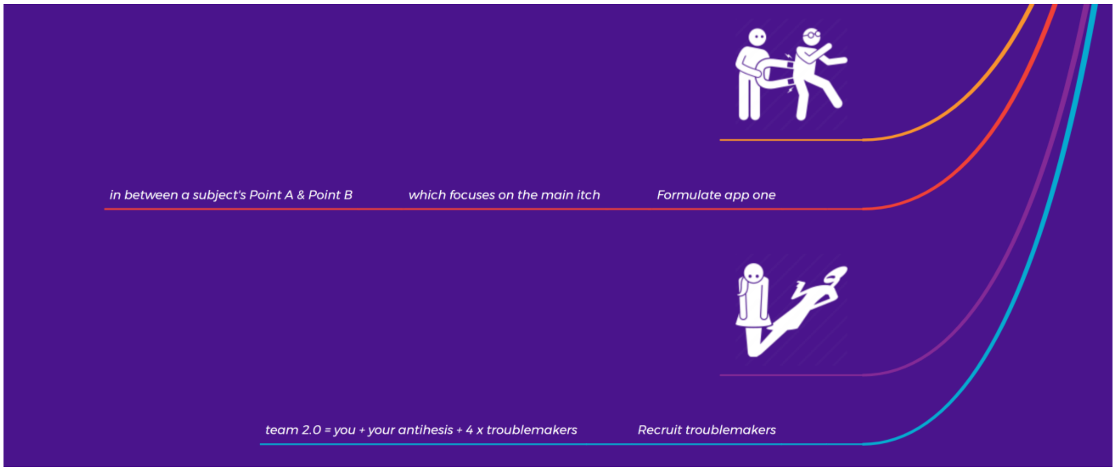
absurdities in the presence of strangers… and let loose when they think they are among their own people. That's the reason behind our otherwise strange behavior. Also, faking technique doesn't have to be too demanding. Sometimes, all you have to do to is to be silent no matter what.

After you have found the absurdity, you were after, you must make sure it is an interesting issue. That it is about an important social issue, and it is testable. After that stage, you must make it intelligible by uncovering the presuppositions of those who take that absurdity seriously. While making the absurdity intelligible is enough to formulate app one, further understanding the social issue in terms of BPI Entities (beings who understand their being in terms of power) gives you ideas about the whole package, "three interlinked apps and a consolidator".

On one hand you have the presuppositions of your target ecosystem and on the other you have your own. That understanding should be used to find candidate ecosystems, you know, ecosystems that actually have postal addresses. You must pick one with the most variety. You must also take logistics issues into account, because you and your team will go there, conduct interviews and experience the ambience yourselves.

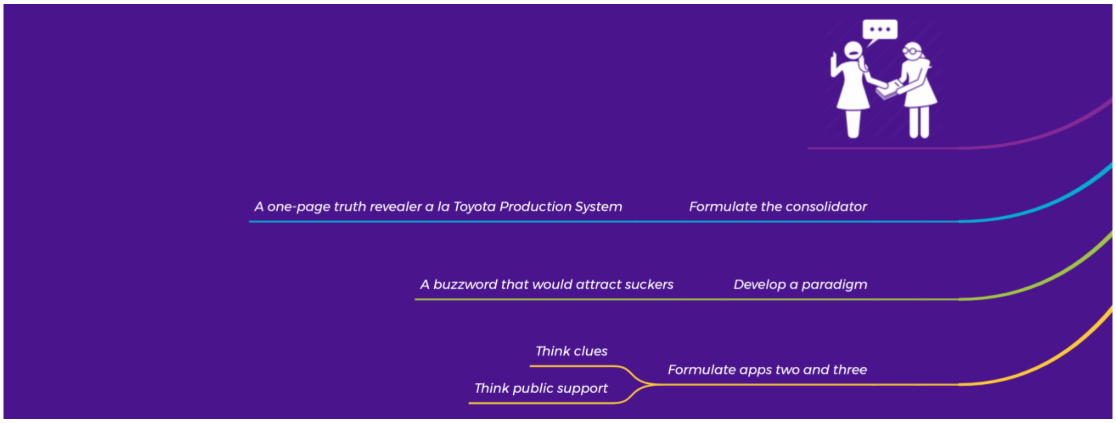

Find your subjects
Make a list of archetypes and personae.
Identify business and system actors.
Found a School's Out HUB
where you can meet with your team and
work on social experiments together.
team 1.0 = you and your antithesis

Are you still with me? Good! The next step is to find suitable subjects. Even if there are many subjects in your target ecosystem, the suitable ones may be hard to find. An exploitation ready (!) subject is a hopeless one. A hopeless person who thinks there is hope. That's your formula for spotting one. You will use their energy for your experiments. Since these kinds of subjects never use their energy for what matters as in "going through the exit", their energy is contained. You can milk the sob(s) so to speak. Oops! I shared too much. Anyway, without suitable subjects, there is no social experiment.

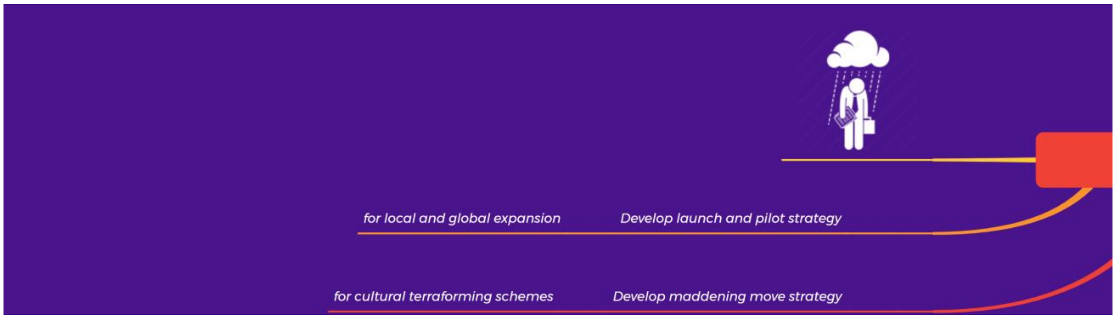

in between a subject's Point A & Point B
which focuses on the main itch
Formulate app one
team 2.0 = you + your antihesis + 4 x troublemakers
Recruit troublemakers

The next two steps are interlinked. You must find another type of subject, those who can go through the exit one fine day. The easiest way to spot one is to learn about her past. If she did it several times before, she can do it again. Another way to find one is to ask the eternal question and go over the answers. You know, the question that goes like "tell me why one plus one is one". The ones with mathematical answers are the dumbest of the pack. The ones who find answers within are next in line. Those who can answer it thinking about other people are the ones you must focus on.

As you find candidates, you must prep your training grounds for future experimenters. A School's Out HUB isn't bigger than a simple classroom in its first phase. So, any comfortable, conveniently placed building would do. You wouldn't believe how many pubs I have used as my office. If you can get the job done, the world is your oyster, my friend. Why seek fancy shelter.

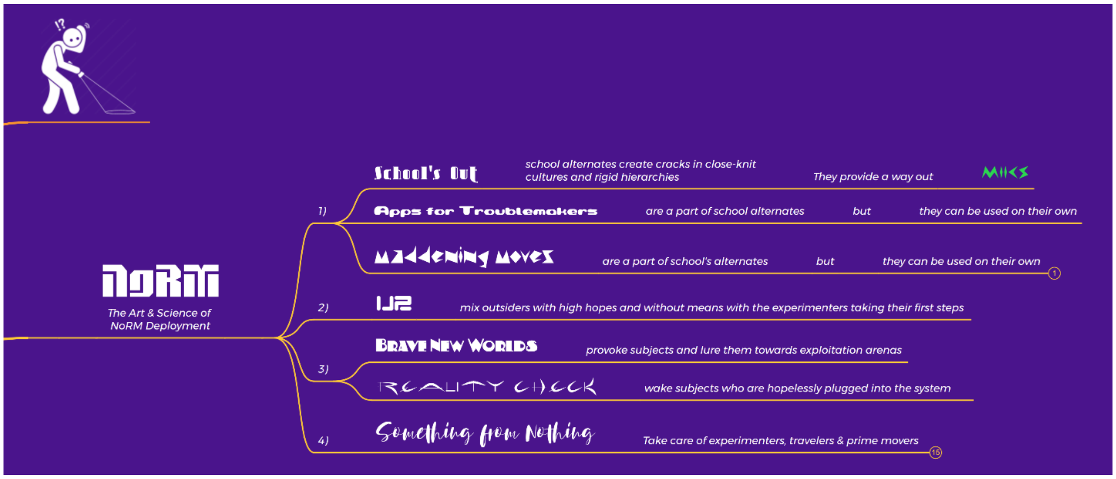

A one-page truth revealer a la Toyota Production System
Formulate the consolidator
A buzzword that would attract suckers
Develop a paradigm
Think clues
Think public support
Formulate apps two and three

You know the drill. Any experimenter should be able to formulate interlinked apps and consolidators. That's your success criteria. You should give passing grades only if something worthwhile happens in the real world, not in the classroom.

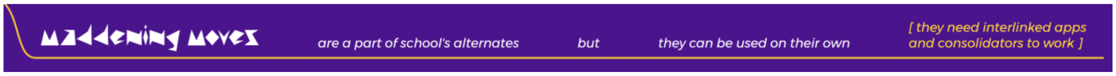

for local and global expansion
Develop launch and pilot strategy
for cultural terraforming schemes
Develop maddening move strategy

Sharing is caring. Never forget that. When you have finally finished your experiment and polished your artifacts making them presentable, it's time to share. Plan a fun launch party and begin the pilot at the same time.

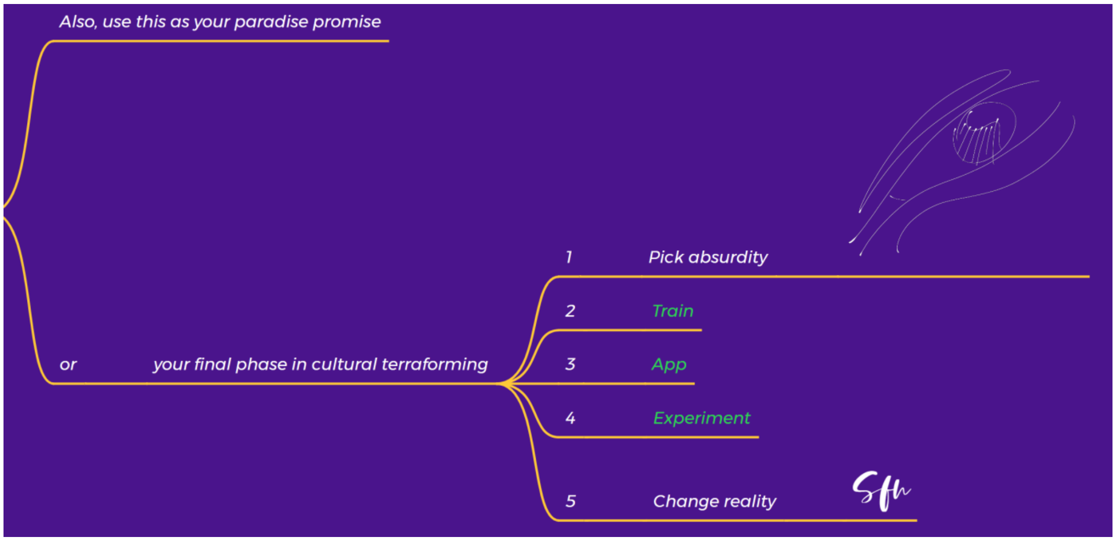
The Art & Science of Preaching – Part Two


NoRM
The Art & Science of NoRM Deployment
1)
School's Out
school alternates create cracks in close-knit cultures and rigid hierarchies
They provide a way out
MIKS
Apps for Troublemakers
are a part of school alternates
but
they can be used on their own
Maddening Moves
are a part of school's alternates
but
they can be used on their own
2)
IJR
mix outsiders with high hopes and without means with the experimenters taking their first steps
3)
Brave New Worlds
provoke subjects and lure them towards exploitation arenas
Reality Check
wake subjects who are hopelessly plugged into the system
4)
Something from Nothing
Take care of experimenters, travelers & prime movers

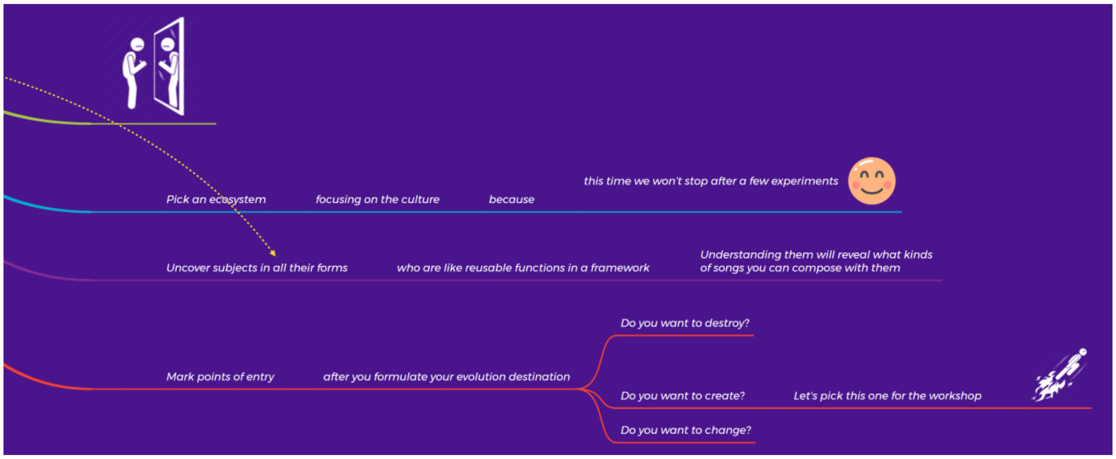
Yes, School's Out is at the heart of everything. Because it is where one realizes there is a way out. However, after you have become an experimenter, you must become a part of a larger universe. Experiments are only the beginning. You have four types of tools in your arsenal. You can do School's Out things, "school alternates", "social experiments" and "terraforming projects". You can link captives in the workplaces to your hubs. You can clear visions, wake up sleepers and shake people. Finally, you can create alternate worlds where the laws of physics, so to speak, change.


Maddening Moves
are a part of school's alternates
but
they can be used on their own
[they need interlinked apps and consolidators to work]

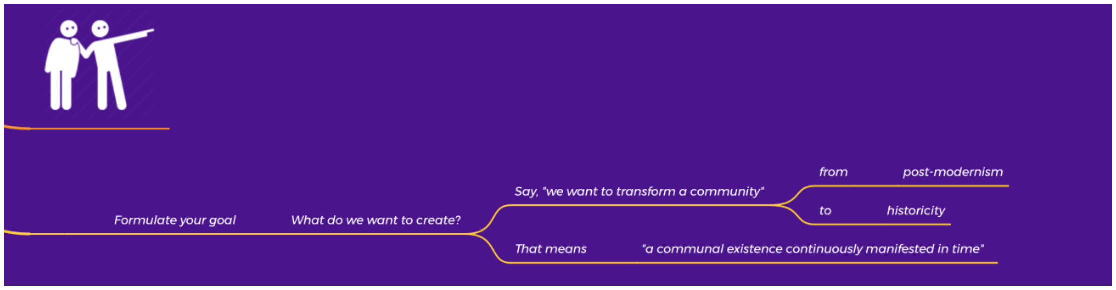
Cultural terraforming is the result of the application of School's Out artifacts over time. So, every social experiment has the potential to grow. Every ecosystem you play with can transform itself.


Also, use this as your paradise promise
or
your final phase in cultural terraforming
1
Pick absurdity
2
Train
3
App
4
Experiment
5
Change reality
Sfh

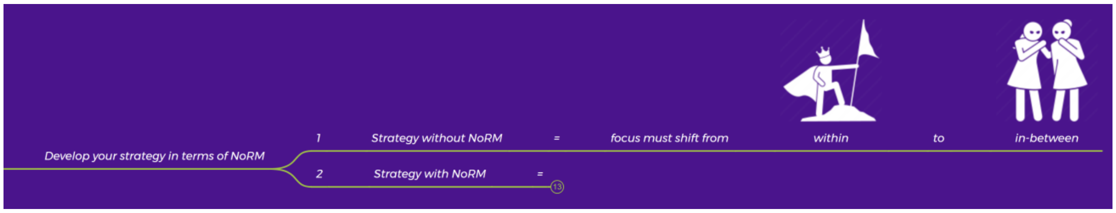
What lies between absurdity and alternate reality are training, apps and experiments. They are your steps.


Pick an ecosystem
focusing on the culture
because
this time we won't stop after a few experiments
Uncover subjects in all their forms
who are like reusable functions in a framework
Understanding them will reveal what kinds of songs you can compose with them
Mark points of entry
after you formulate your evolution destination
Do you want to destroy?
Do you want to create?
Let's pick this one for the workshop
Do you want to change?

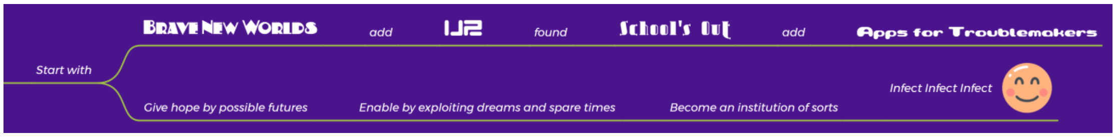
Culture of a target ecosystem will tell you its potential for cultural terraforming. Some cultures are meant to die.


Formulate your goal
What do we want to create?
Say, "we want to transform a community"
from
post-modernism
to
historicity
That means
"a communal existence continuously manifested in time"

Different manifestations of interlinked apps and their consolidators must tell a story in the form of a paradigm.



You must have two plans executed in parallel, one is your NoRM deployment and the other is your NoRM politics.


Start with
Brave New Worlds
add
UR
found
School's Out
add
Apps for Troublemakers
Give hope by possible futures
Enable by exploiting dreams and spare times
Become an institution of sorts
Infect Infect Infect

While it is possible to use evolution instruments on their own, the best way to go about it is to formulate a series of applications such as this one above which can be followed by the steps below.

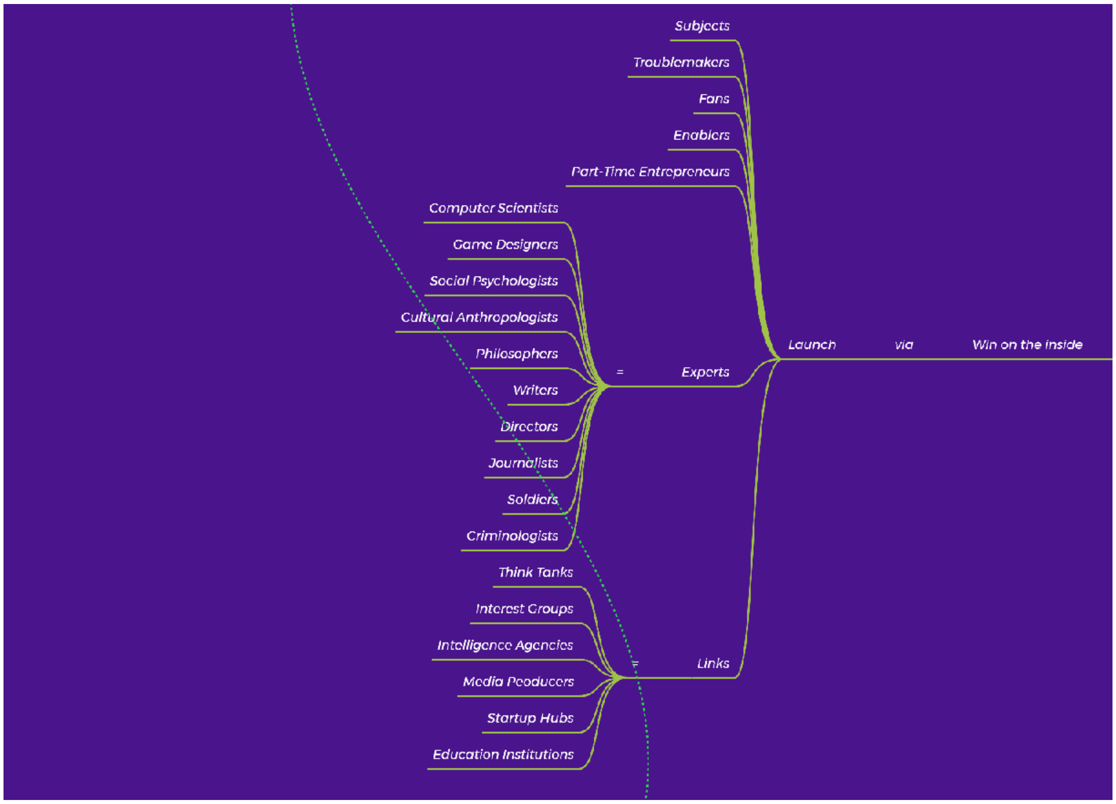
seed communities
=
dreams + spare time
=
U2
add
BRAVE NEW WORLDS
i.e.
Initiate first strike

If you are like me you will think that best defense is offense.

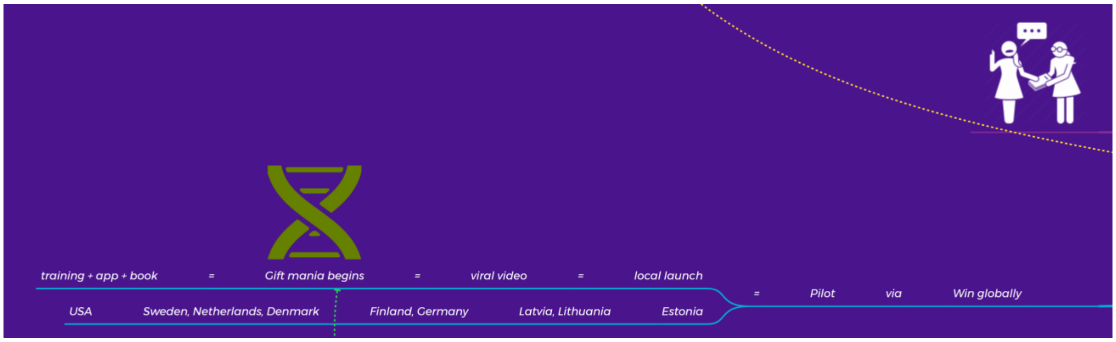
experimenters
=
troublemakers
=
School's Out
i.e.
Initiate full campaign

And you would like to create institutions from experiences.

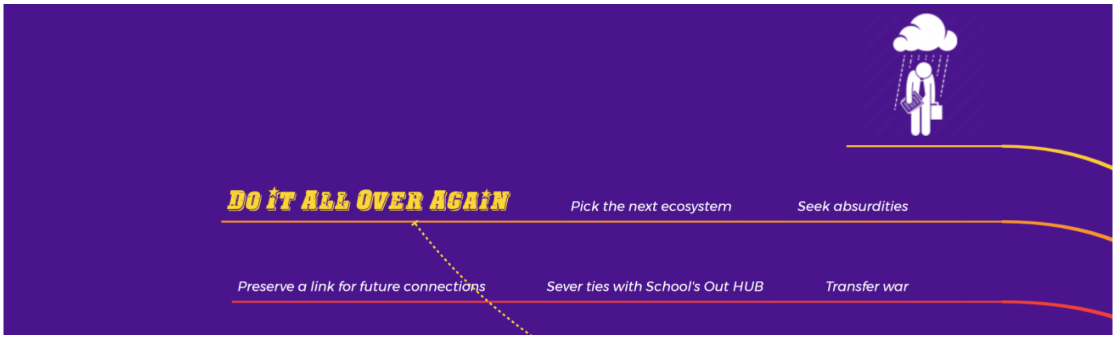
Don't forget to add variety to your experiences.



Don't just torment the subjects. There are all kinds of people out there… and they all need someone who will lie to their faces… just to feel good for a minute or two.



The best way to test your idea is testing it in different cultures one after another. This way you will find your audience and understand your true calling.



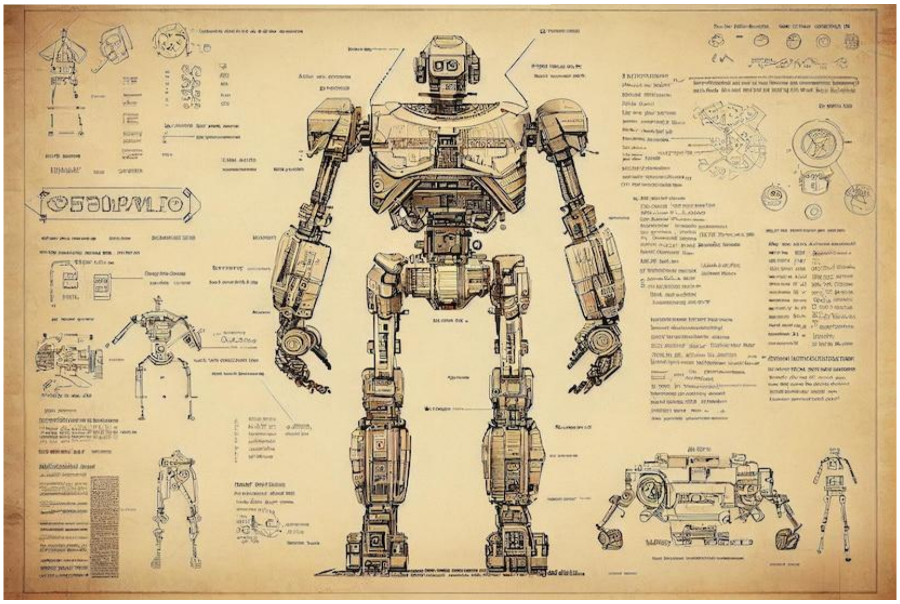
Finished? Me too! Don't just sit there. The cycle is complete. Let's create another!

Case Story

How about a sample run? Yes, it's real… but I won't name names, because I'm such a nice guy. Every School's Out cycle begins with picking an [interesting issue] which is uncovered by a [traveler] who is someone who walks the earth kind of like Kung Fu. Without a beginning and without an end may we fall into worlds of not our own, so that we may live in the virginial now. Conscience is care and concern towards what's out there even if, especially, it has nothing to do with us. We are just another piece of the puzzle. We cannot find truth on our own. Beauty, truth and goodness are not within. They are without… grasshopper. So, when an absurdity comes your way, be sure to take your time to enjoy it.

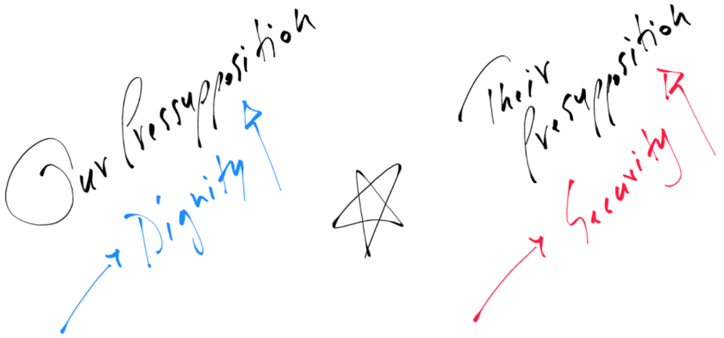
Everybody bitches
but still takes
crap

Where does one start? You know the answer! Exploitation begins at home. So, we spent quality time among our own. Within minutes… way before the coffee cups were empty… we knew. Ours was an ecosystem of not so brave people. People who did everything to conceal that they were cowards and faked their way towards the deeper catacombs of their slavery. Sometimes, what seems like the way out is a way in.



"When what you are is revealed to you, you cannot make heads or tails of it" (Westworld)

The second step is to understand why something that makes no sense makes sense to other people. Why are they the way they are? While formulating zeroth questions may seem to you like the black arts, actually they are pretty easy to formulate granted you aren't blinded by the presuppositions you haven't uncovered yet. How can you be sure? Well, if you are on the way to the zeroth question meaning you can 'see' the absurdity, you are on the right track. If you are *blind, you would be looking at the absurdity and you wouldn't see anything out of the ordinary. It wouldn't occur to you. Such a realization would be beyond the borders of your notion of reality. That's why we make sure interesting issues are tracked by specialized experimenters we call the [travelers]. Their presuppositions are not of this world. What they cannot see doesn't cripple our crazy schemes. It cripples (!) them. When a [traveler] summons a [prime mover], we all know that there is an absurdity even if we don't see one yet. That's how the life of an idealist should be.*



In this cycle's focus we have those who prefer "security" to "dignity" … "slavery" to "freedom". Because, simply put, they are chicken shit… and they love toys. This leads to another question, though. "What is it that makes them feel secure?" There, you have found your zeroth question!

"Distancing oneself from pain may give him the illusion of a meaningful life"

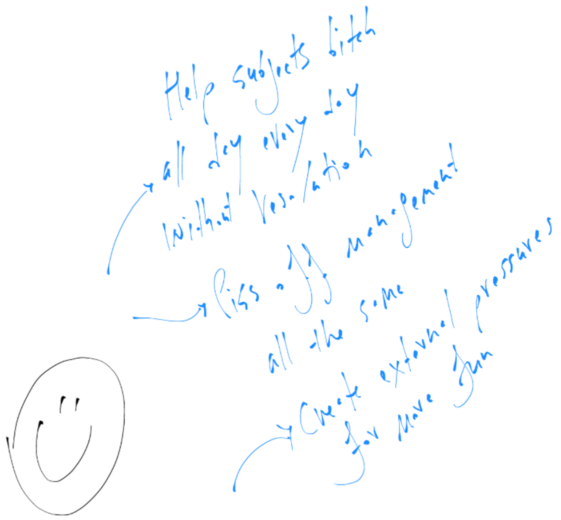
"Yet, life isn't about happiness. It is about living."

Once you have your zeroth question, you cannot only find an answer, but also you can find a way to keep your [subject(s)] busy, powering your experiments by the itches they cannot scratch. Here "being a part of a well-oiled mean machine feels like home, because even if it hurts, it is very organized… it never leaves you without a purpose… it always moves in the direction of what can easily be mistaken as going forward. You always know your place … which is not the case out there in the open, in the real world… where anything goes. You always feel at home." That's the price of freedom, but you are all about earthly profits. This is precisely what locks you in ecosystems you hate. You were made for each other. You just cannot internalize who you really are. That's the itch! Don't worry. We will make it go away. Like taking the blue pill, you feel at home regardless of what kind of a place it is. You will love it. It's a way of life.

So, an app for troublemakers is actually designed thinking about the [subjects], not the [troublemakers]. We plan to recruit [troublemakers] into School's Out hubs as enthusiasts. We want to turn them into [experimenters] who will be experimenting on the likes of [subject(s)]. Yes, we are efficient.

If someone bitches all day and every day yet never quits… he needs an app to bitch more efficiently. He needs a better way to conceal who he really is. He needs a better way to lie to himself… to fake an imagined, idealized version of himself. He must expire without living. That's his success criteria! What is yours?

That's how Moody was born. We imagined an app where one had the option to vote every day to broadcast how good or bad he felt. Have you ever seen cats spraying? That's the idea. What an annoying person, right? And there are so many of them. Lucky us! This person's negative energy must be used to spread his unhappiness along with an intriguing reason… and this bad feeling should be transferred to upper management and its right arm, human resources, even if they don't give a fuck… especially if they don't give a fuck. Think about it. It's like thermodynamics. On one end we have those negative feelings trying to spread to the realms of serenity… turn them sour… make them just like home. Powerless people bitch, because if they can make whatever's pissing them off widespread, they will end up being normal… or so they think… which they are not.


Help subjects bitch all day every day without resolution
Piss off management all the same
Create external pressures for more fun

We will use their condition to make other strong feelings sour. On the other end of the spectrum we have management who are only interested in numbers and see the misery of the others as a necessary byproduct. They cannot envision a different world. We will make them feel bad too!

Are you still with me? Good! Here's your principles:

Our app should make broadcasting feelings very easy.
Our app should create chaos in the company.
Our app shouldn't be contained by a single company.

Let's see, what do we have here?

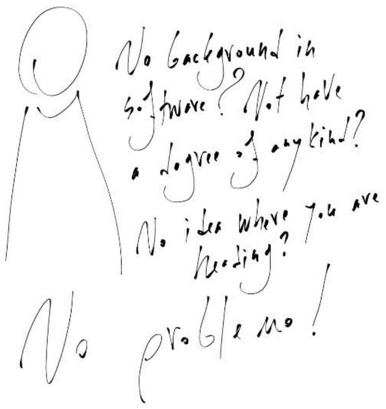
- *We have [subject(s)] who are employees working in the same company for years.*
- *Our [subject(s)] are sour. They complain all the time, but they never leave. They are all Uncle Tom(s).*
- *We have [troublemaker(s)] who are aware they are taking shit every day.*
- *Our [troublemaker(s)] are looking for ways to escape or at least ways to avenge themselves.*
- *We have [enabler(s)] working in similar companies, but they can fake it better being in other ecosystems.*
- *Our [enabler(s)] have either learned to enjoy their misery or gave in completely to have more toys or power.*

These were the people under focus. We also had the first app concept. All we needed at this point was where the School's Out HUB would be and who we would recruit as the students for this cycle. Take it away [prime mover].

Since this cycle's district was the district of the people who could be persuaded into doing anything when the price was right, we thought it would be full of troublemakers, but we were wrong. So, we ended up recruiting the next best ones, those who came to the big city to study at universities… only to become as miserable as their parents and like.

In this instance troublemakers didn't come from the targeted businesses. We recruited their children. Strange, right?

This tells something about that ecosystem, but it's a story for another book.


No background in software? Not have a degree of any kind?
No idea where you are heading?
No problemo!

Yes, we started as an opportunity for the clueless, but we ended up becoming an opportunity to not so clueless after all. Then again, they were clueless about other things. We recruited six students. What was common among them? They were either pursuing a career in computer sciences, or they were craving it. It was a 50-50 situation which gave me the idea for the "Sherlock Holmes - Dr. Watson duos". One who can catch the murderer and one who can lock him up. An analyst and a programmer, so to speak. It seems the perfect duo of our profession resurfaces again and again regardless of our dreams of another configuration… which only means that it doesn't need any improvements. Our salvation isn't before us. It is behind us.

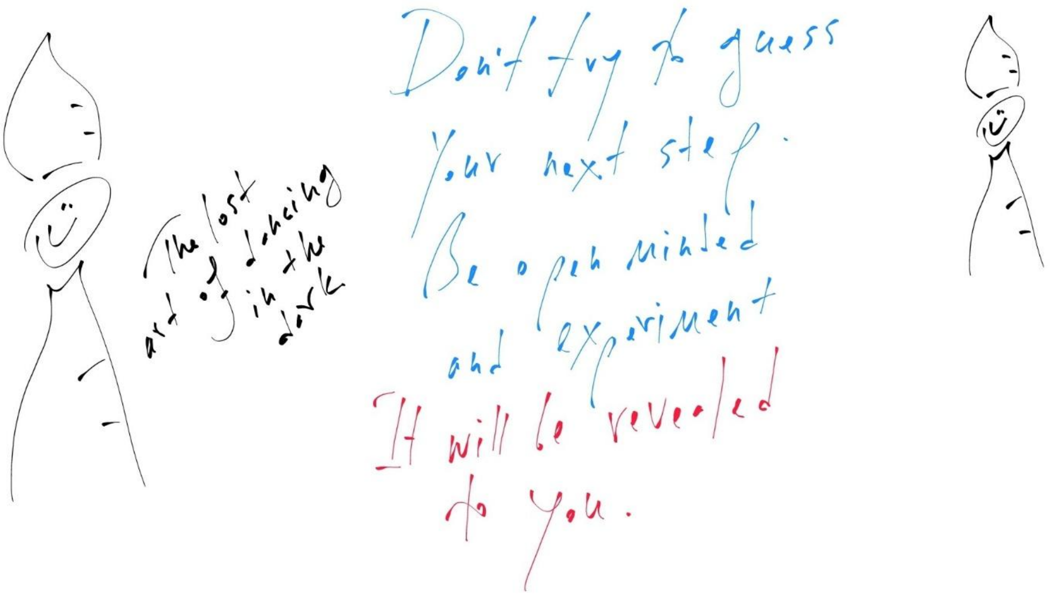
Sometimes, we take a step back and observe our creation. It's always fun to watch something you made up take baby steps and then become your opponent without realizing they are there only because you deemed so. Yet, another Siegfried moment. There are times I wish I was Noah, I confess.

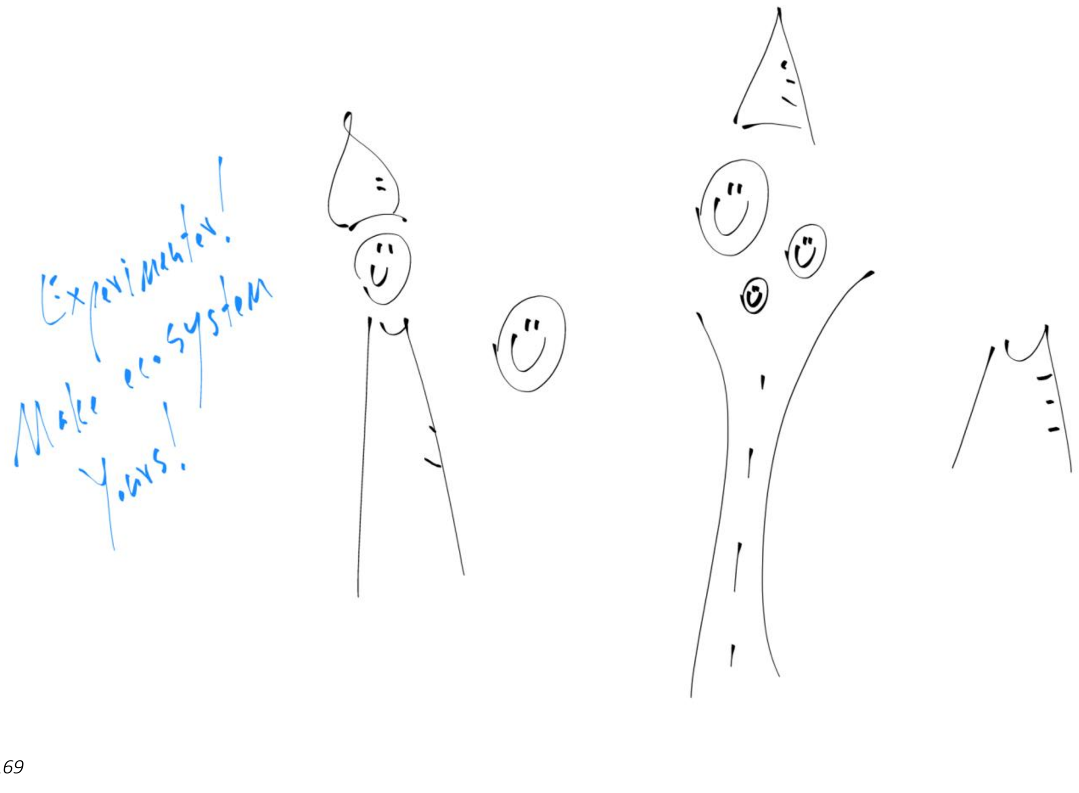
While we take further steps to complete our experiments in a timely manner, we must realize that it's pretty much like thermodynamics. Mass has a direct connection to energy. They are irreplaceable. In their core, on the other hand, both are nothing but information… and that's what we are all about too. We gather information, play with it, create more information, pass it along, let it spread everywhere. Think about it this way, we create information via experiments, use information to terraform cultures and ultimately to create alternate realities… which can be turned into something else by fellow experimenters later. We are playing with the fabric of spacetime continuum. There is a difference, though. We are Star Trek going inwards… into a much larger and stranger realm that is the human condition.

Where were we? Oh, yes, we were about to deploy our first app. A typical way to do so is to hand the app to human resources which is bound to fail, because human resources management is nothing but an accomplice of upper management. They are there to do their dirty work… always smiling… hiding behind high heels and fancy ties. So, instead, we went with LinkedIn authentication. Whatever an employee's current company was… is accepted as her current company in real life. We are not after accuracy here, after all. Hatred or love transcends numbers. This way nobody can shut the program. Neet trick, huh? People keep bitching about you, and you have no control over it. Nice!

What an experimenter should do is to inject the app to the selected group with easily distributable ways. This way, friends and family of your subjects will be infected too. They will spread the disease too. Keep in mind that you must have more than one ecosystem for the last step in the chain, Emotional Agility DASH, to work, because there will be a ranking of companies with respect to their emotional agility performances.

That being said, never assume that you can predict what is next. Never fix what you are going to do. Always assume what needs to be done will be revealed to you on the way. While it may be the very thing you have in mind, that doesn't mean that you have gained the power to see the future. Nobody does. That's the biggest mistake everybody makes. You should know better. Even if you can see where you will end up, there are tons of other factors that can change that.


The lost art of dancing in the dark
Don't try to guess your next step.
Be open minded and experiment
It will be revealed to you.

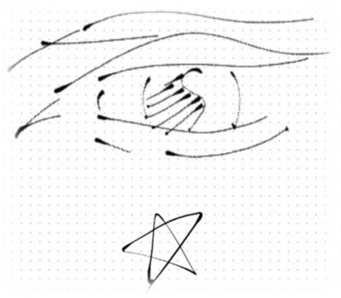

Experimenter!
Make ecosystem yours!

This attitude is what differentiates a master from the rest. People assume too many things, especially about themselves. They have imagined starting points and destinations. They see everything and everyone as intermediary steps towards an… you have guessed it… imaginary finale. Simply put, none of this matter. You don't have to come from somewhere. You don't have to go somewhere. You just have to… be… which is the default orientation of our species. Sadly, it's a feature which is very easy to lose. That's nature's way of getting rid of bad apples. Do you see it too? Nature behaves in a way that she accepts bad apples are the majority! Those who don't lose this default orientation become [experimenters]. And the ones who easily lose it are their [subjects] or [enablers]. Troublemakers, you say, huh? Well, since they are the starting point of experimenters... you can guess, can't you?

NoRM Cycle

[1] Absurdities Becoming Interesting:

Everything starts with an [interesting idea]. An interesting idea is an absurdity which cannot be made intelligible by our presuppositions. We must formulate a zeroth question that will reveal the hidden presuppositions that make it intelligible. A zeroth question is the question that cannot be formulated within the borders of reality of the absurdity we are trying to understand. When the absurdity becomes intelligible, we have our grounds for experimentation. Not only that, knowing both sets of presupposition (ours and theirs) we now understand both notions of reality.



[2] School's Out Cycles:

When the [interesting issue] reveals her [ecosystem], it's time to summon [prime movers] who are travelers who stopped traveling and took an interest in gardening, so to speak. A [prime mover] always works solo. He makes (!) friends as he needs them. See, that's why I contacted you! He focuses on the ecosystem to identify the people he can exploit to create a new School's Out HUB. Some are experimented on. Others are liberated. Which one are you?

[3] Exploitation Entities:

While there are three possible types of people a [prime mover] can exploit, two of them are more important.

- Subjects are the ones who are unhappy about something.

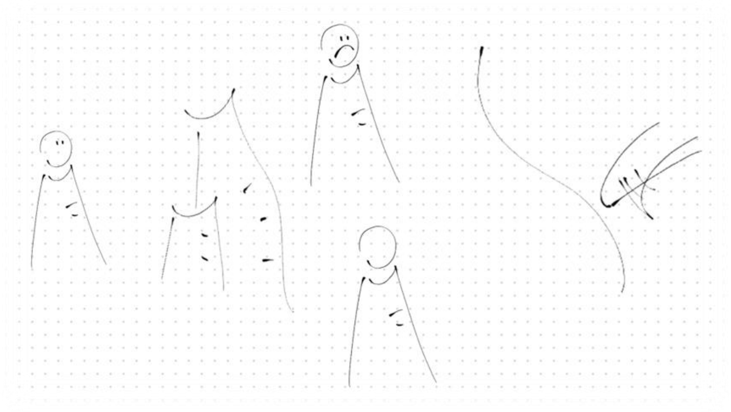
They are the ones who are going to use our apps.

- Troublemakers are those who want to wreck things.

They are a different breed of subjects. They are brave. They can be freed. We recruit them as School's Out students.

- Enablers are subjects from other ecosystems.

Since they don't answer to the power holders in our ecosystem, they share resources… and occasionally give a hand.



[4] Troublemakers Becoming Experimenters:

Once recruited, [troublemakers] go through three steps of training.

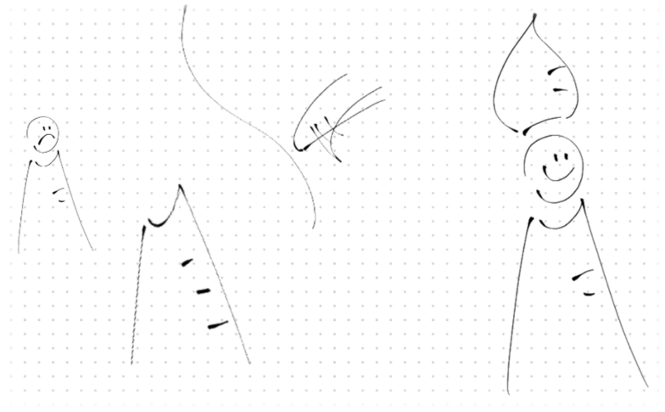
(1) They start by learning the vision and the methodology.

(2) They continue by analyzing the ecosystem, creating relationships with its many inhabitants, formulating apps and then developing them.

(3) During the last step, they distribute the artifacts of that School's Out cycle at a launch party and conduct several pilots.

Having started the journey as troublemakers,

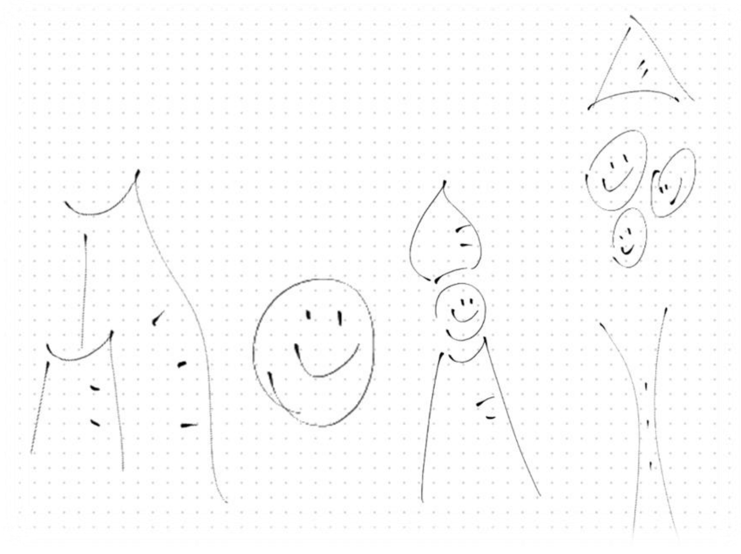
- *first they turn into enthusiasts,*
- *then members of Sherlock Holmes - Dr. Watson duos.*
- *Finally, they graduate as members of A-Team(s)… social [experimenter(s)] powered by software chops.*



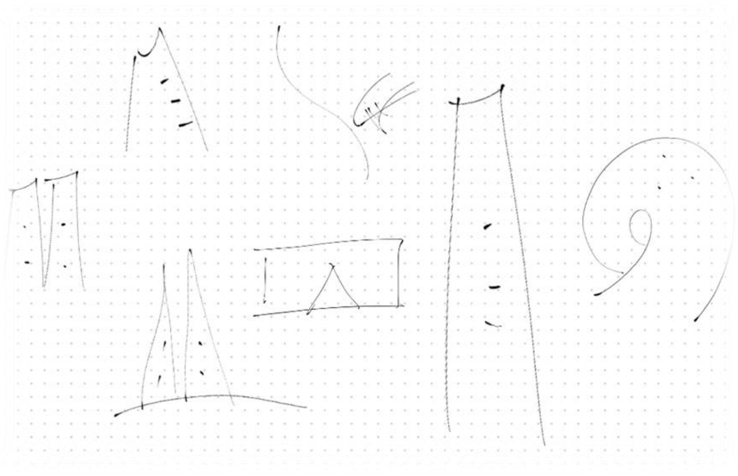
[5] Experiments Galore:

While they are in a School’s Out cycle, they take control of the target ecosystems little by little. First, they deploy an app and let it mature as its fanbase grows. Then, they formulate the other two interlinked apps and deploy them as well. Finally, they formulate a one-page [consolidator] app and link it with the previously deployed apps making the ugly truth unavoidable.

- *First app covers the biggest itch*
  like “inability to leave a lousy workplace”.
- *Second and third apps for troublemakers as helpers of the biggest itch*
  like “why it is lousy” and “how many people are sharing the same feelings”.
- *A paradigm that unifies the concept as exportable culture*
  like “being emotionally intelligent is good”.
- *A one-page consolidator that makes the new paradigm visible and enables ranking*
  like “company A is more sensitive to employee needs”.
- *These artifacts are further developed by releasing “How to” books*
  like this one here, “A4t1 is Moody’s How-to book”.



Another perspective is the one [prime mover] has which is all about strategy. Strategy… to attack a certain culture …to create a certain culture …to create a new community …to take care of experimenters. While components of these strategies may overlap, their goals are always different. You either fight with or against other powers.



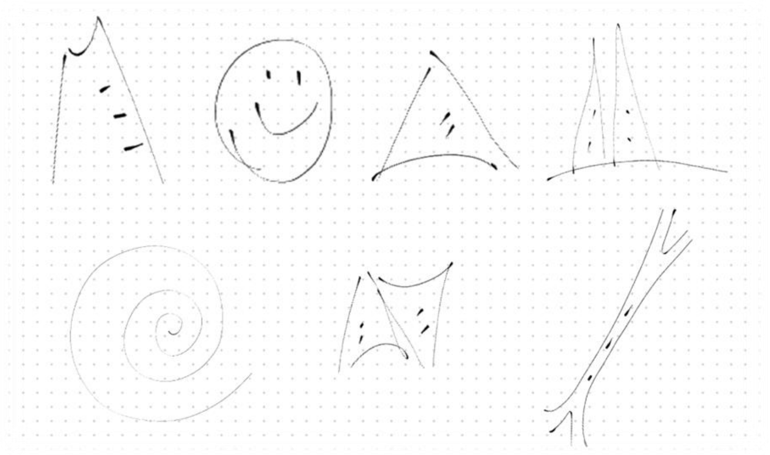
[1] HUB:

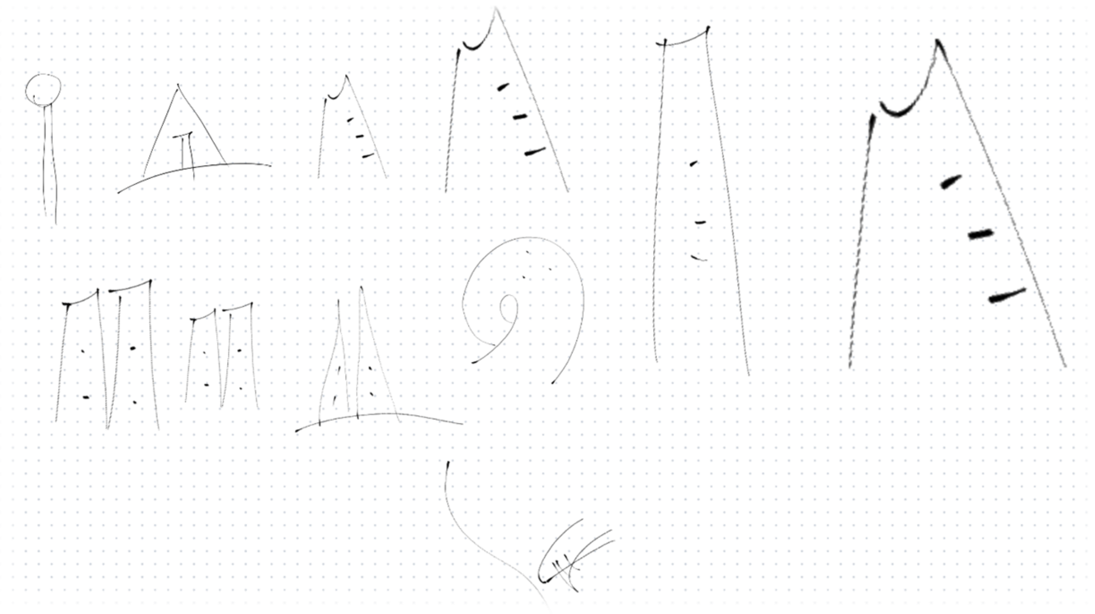
A [prime mover] may create a new School's Out hub, or he may have already done so and wants to link it with the business world via part-time entrepreneurship. While one may create School's Out hubs one after another, he doesn't always have to follow the same strategy. Remember, "one must compose his band". There are other options.

He may go into film or game production. He may focus on the issues interest groups are facing. He may form a think tank, or he may just go ahead and serve intelligence agencies. And my favorite, he does everything all at once!

[2] SO HQ:

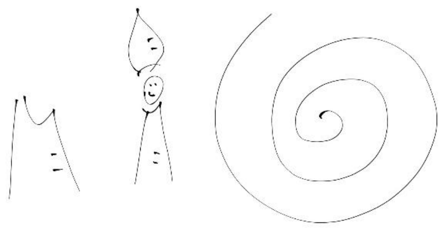
Depending on where he is in this adventure, he may choose to make a certain School's Out hub bigger… bolder and more beautiful. He may create additional units ranging from anthropology, psychology, philosophy to journalism, criminology, military sciences. He may also include game sciences and media production. He may finalize the mix by adding a part-time entrepreneurship outlet and a think tank. This way he not only creates a School's Out hub with greater abilities and a longer reach, but also, he strengthens the ties between the hub and the rest of the world… so that he can create possible futures in greater numbers.





[3] Paths:

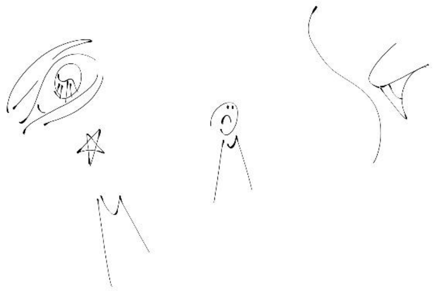
Whenever he starts a new adventure, he has seven starting points in his arsenal. Often, it's best to start with School's Out for it's the breeding grounds of [experimenter(s)] without whom we wouldn't have [traveler(s)] or [prime mover(s)]. He may proceed with Apps for Troublemakers, followed by Maddening Moves. Before, during or afterwards, he may add a pinch of Brave New Worlds here and there. He may provoke strong feelings with Reality Check. Again, he may link the business world to his crazy schemes via Part-Time Entrepreneurship. Last but not least, he may empower those who are worthy by using Something from Nothing… which is always a double-edged sword… good for every occasion.



Something from Nothing (SFN) deserves her own paragraph because of her being a world, nay, universe of her own. It serves as an alternate reality where one can filter out the things she doesn't like and live in her utopia. On the other hand, by doing just that, one can turn it into a dystopia. It provides a mechanism to have building blocks for achieving both ends. As one dives deeper and deeper, SFN learns about the hero more and more… until one day it replaces the hero with herself.

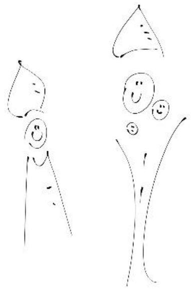
Regardless of these possibilities, Something from Nothing can be a tool to empower [experimenters] along with [travelers] and [prime movers]. Since gardening is the [prime mover's] main concern, SFN is his most powerful tool.

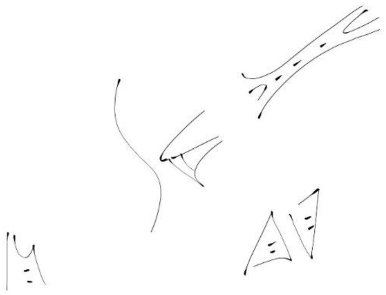
Before we wrap things up, let's go through our archetypes along with their toolboxes.



A traveler, a troublemaker and a prime mover are eternally linked. A troublemaker turning into an experimenter is the beginning. An experimenter becoming a traveler is the middle. A traveler becoming a prime mover is the end. They are tied together with absurdities which are bound by ecosystems.

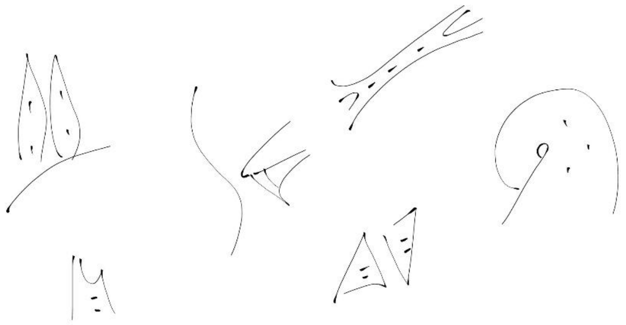
Experimenters are the most active archetypes in our adventures. They are the ones who survey ecosystems, conduct experiments, strategize maddening moves and conduct them. Theirs is an art of social experimentation which goes hand in hand with terraforming cultures. In that regard, they are the ones who change, create and spread cultures. They are the force that transforms subjects into experimenters.

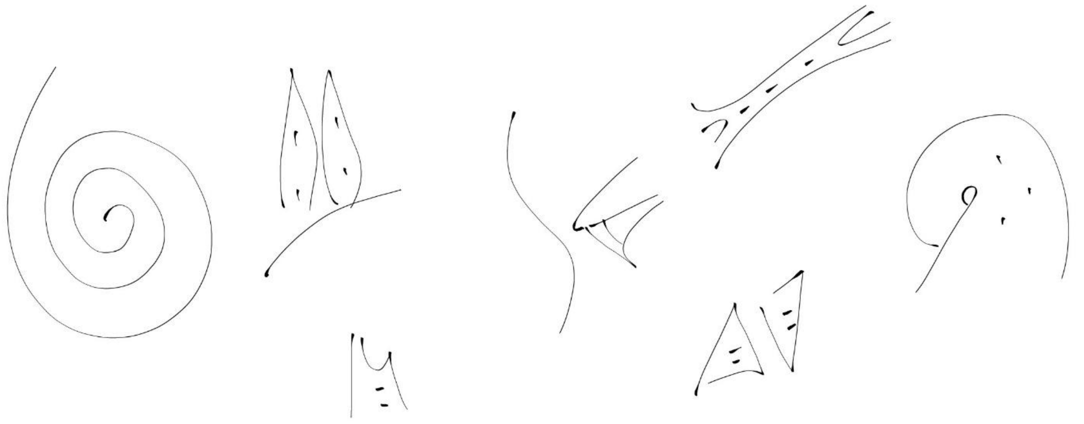




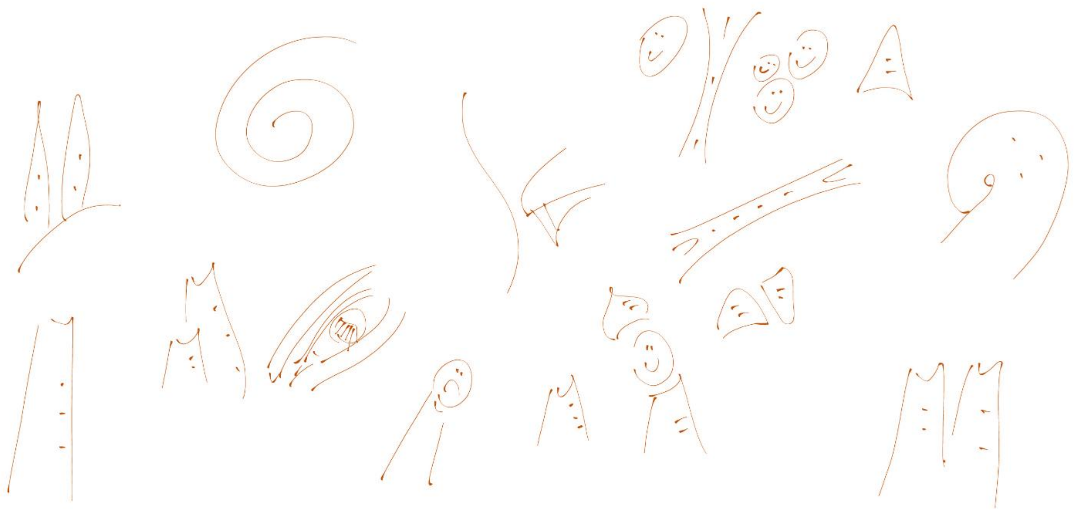
Prime Movers are like gardeners. They are the ones who turn exploitation opportunities into apps, experiences, cultures and communities. This is the birthplace for experimenters. Prime Movers recruit troublemakers from the experimented ecosystems and train them. That's an old Ottoman tactic, to recruit from your competition and turn them against themselves. This time there is a difference, though. We are doing it for their own good.

Prime Movers aren't limited to School's Out HUB(s). They have all kinds of tools at their disposal. They can use bad experience simulators to wake people up. They can use app stories to give people hope while exploring other possibilities of existence. They can even give a helping hand to those who are not brave enough to quit their day jobs by providing part time entrepreneurship outlets for them.





When everything is put together, you have a huge toolbox. You can spend an eternity only in one spot or if you are more like me, you can jump from one opportunity to another. Failure or success doesn't really matter. What really matters is the journey and the friends (or enemies) you make along the way.



I'm a trinity kind of guy, so we represent our hero by three interlinked (yet another fascination of mine) personae, [experimenter] leading to [traveler] leading to [prime mover]. Consider this as another take on the three phases of life (Joe Marasco). Subject is a schlepper (someone who has no control over his life). He may become an enabler or continue to become a seasoned (!) subject… who is a macher (someone who gets the job done at the expense of others). If he has guts, he can go through the rabbit hole to become an experimenter who may one day become a prime mover (someone who leaves a cultural mark behind him).

A4t 1