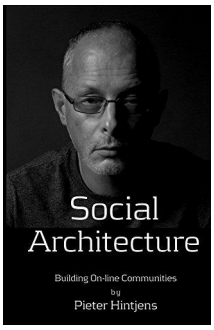
kindle



YOUR KINDLE NOTES FOR:

Social Architecture: Building On-line Communities

by Pieter Hintjens

Free Kindle instant preview: http://a.co/cYkqLDn

67 Highlights | 3 Notes

Highlight (Yellow) | Location 58

> process for building smart, self-guiding, successful on-line communities that could beat expert groups every time. It is a discipline I named Social Architecture,

Highlight (Yellow) | Location 126

> Any intense group, family, business, or team starts to resemble a cult, in little or larger ways.

Highlight (Yellow) | Location 128

> There was a straight causal effect: as the group became more cult-like, they became more useless.

Highlight (Yellow) | Location 130

> Define a powerful mission to attract newcomers. Make it really easy for people to get involved. Embrace argument and conflict; it's where good ideas come from. Delegate systematically, and create competition. Work with volunteers more than employees. Get diversity and scale. Make people own the work; don't let the work own the people.

Highlight (Yellow) | Location 138

> In my Social Architect's toolbox, I have 20 tools,

Highlight (Yellow) | Location 140

> Strong mission -- the stated reason for the group's existence Free entry -- how easy it is for people to join the group Transparency -- how openly and publicly decisions are made Free contributors -- how far people are paid to contribute Full remixability -- how far contributors can remix each others' work Strong protocols -- how well the rules are written Fair authority -- how well the rules are enforced Non-tribalism -- how far the group claims to own its participants Self-organization -- how far individuals can assign their own tasks Tolerance -- how the group embraces conflicts Measurable success -- how well the group can measure its progress High scoring -- how the group rewards its participants Decentralization -- how widely the group is spread out Free workspaces --

> how easy it is to create new projects Smooth learning -- how easy it is to get started and keep learning Regular structure -- how regular and predictable the overall structure is Positivity -- how far the group is driven by positive goals Sense of humor -- how seriously the group takes itself Minimalism -- how much excess work the group does Sane funding -- how the group survives economically

---

Highlight (Yellow) | Location 167

> Use your mission as a slogan, on your website, marketing, presentations, and so on.

Highlight (Yellow) | Location 168

> Without a clear mission, an on-line community won't grow.

Highlight (Yellow) and Note | Location 168

> A group of friends who start a project may agree what they want to do, yet anyone new coming on board has to guess what they had in mind. People will guess wrong, and will change their minds over time. This leads to confusion, disagreement, and disappointment as people find that their hard work was wasted because the rest of the group headed off in a different direction.

We need this for our community

Highlight (Yellow) | Location 267

> Promote the most active contributors into positions of authority, and do this rapidly. You have a short window for promoting new contributors before they disappear to other projects.

Highlight (Yellow) | Location 277

> Stay away from formal membership models, especially those that try to convert people to belonging to the group. Allow anonymous or unidentified participation. Encourage people to create their own competing projects as spaces to experiment and learn.

Highlight (Yellow) | Location 286

> Top-down task assignment is an anti-pattern with many weaknesses. It makes it impossible for individuals to act when they recognize new problems.

Highlight (Yellow) | Location 294

> A diverse group has conflicting opinions, and a healthy group has to embrace and digest these conflicts.

---

kindle

Highlight (Yellow) | Location 313

> An overriding motivation is to be admired for success. That can be as an individual, or as part of a team. Success is relative so we need metrics, some high score that people can see and track.

Highlight (Yellow) | Location 345

> Make it absolutely simple for logged-in users to create new projects.

Highlight (Yellow) | Location 350

> We seem bad at learning structures deeper than three or four levels. However, we're happy to explore very wide structures with thousands or millions of boxes if those boxes correspond to separate units of work, or projects. Think of a city.

Highlight (Yellow) | Location 364

> Think of your community as a video game with levels that become increasingly difficult, and have bigger and bigger payoffs. People will play "up to their level." If you can do this right, you attract the most people.

Highlight (Yellow) | Location 389

> You make a racing car faster by removing weight, not by adding power. You can make your community lighter, faster, and more agile by being dogmatically minimalist about the work you do. Though it sounds lazy, it's often harder to not do something that seems fun than to just go ahead and do it.

Highlight (Yellow) | Location 420

> When you have a real, usable product, it attracts early adopters. These are people making real products yet who are good at taking and managing risk. They still don't need much help, though they do expect some guarantee that things won't break randomly. This is the bulk of your community.

Highlight (Yellow) | Location 427

> Once you understand the market curve, you see why it's counterproductive to, for instance, write perfect tutorials for the early versions. You won't get the mass market regardless and it will feel patronizing to the pioneers.

Highlight (Yellow) | Location 457

> You will have gathered by now that I'm not a great fan of the brilliance of individuals. Mostly this is because despite being a Mensa member, I've seen myself make such amazingly clever mistakes. Over time I've come to think that the very notion of individual intelligence is a dangerously simplified myth.

---

kindle

Highlight (Yellow) | Location 485

> The size and diversity of the community is a key factor. Larger, more diverse communities collect more relevant problems, solve them more accurately, and do this faster than a small expert group.

Highlight (Yellow) | Location 487

> when we trust the solitary experts, they make classic mistakes. They focus on ideas, not problems. They focus on the wrong problems. They make misjudgments about the value of solving problems. And they don't use their own work.

Highlight (Yellow) and Note | Location 549

> Any market follows a power curve where a few players dominate the market, and a majority of players are frustrated. It's by promising this frustrated crowd a way out, that you can convince them to invest in something new and open and potentially game-changing.
>
> **!**

Highlight (Yellow) | Location 619

> If you decide to register a mark, do it in the US (via the USPTO) first. That's cheap, and simple. Then over time you can register in the EU (via the OHIM), if you find your project is worth it.

Highlight (Yellow) and Note | Location 668

> Around <tt>libzmq</tt>, there are about 50 bindings. These are individual projects that create higher-level APIs for ZeroMQ, or at least map the low-level API into other languages.
>
> **Modules**

Highlight (Yellow) | Location 703

> My main takeaway from a long career of projects of every conceivable format is: if you want to build truly large-scale and long-lasting software, aim to build a free software community.

Highlight (Yellow) | Location 725

> This is the number one rule: simplicity beats functionality, every single time. If you can't understand an architecture on a cold gray Monday morning before coffee, it is too complex.

Highlight (Yellow) | Location 726

> the architecture must create space and opportunity for selfish acts that benefit the whole. Selfishness is often indirect and subtle.

Highlight (Yellow) | Location 730

> we'll overcome our stupidity and laziness to prove others wrong and beat them in competition. The architecture thus has to create space for public competition based on fair rules that anyone can understand.

Highlight (Yellow) | Location 734

> Reciprocity: we'll pay extra in terms of hard work, even money, to punish cheats and enforce fair rules.

Highlight (Yellow) | Location 738

> The architecture has to make sure every piece we make has our name on it, so we'll have sleepless nights stressing about what others will say about our work.

Highlight (Yellow) | Location 752

> a collective work has two extreme outcomes. Either it's a failure, irrelevant, and worthless, in which case every sane person walks away, without a fight. Or, it's a success, relevant, and valuable, in which case we start jockeying for power, control, and often, money.

Highlight (Yellow) | Location 764

> Ask yourself, "if this project had a big fight, and split three ways, which license would save us?" Or, "if the whole team was bought by a hostile firm that wanted to turn this code into a proprietary product, which license would save us?"

Highlight (Yellow) | Location 778

> a good contract--and I consider the modern GPL to be the best for software--lets programmers work together without upfront agreements, organizations, or assumptions of decency and goodwill. It makes it cheaper to collaborate, and turns conflict into healthy competition. GPL doesn't just define what happens with a fork, it actively encourages forks as a tool for experimentation and learning. Whereas a fork can kill a project with a "more liberal" license, GPL projects thrive on forks since successful experiments can, by contract, be remixed back into the mainstream.

Highlight (Yellow) | Location 809

> BSD is an excellent strategic tool, but only if you're a large well-funded institution that can afford to use Option One. The Apache license is BSD in a suit.

Highlight (Yellow) | Location 827

> With ZeroMQ, we said we were going to make "the Fastest. Messaging. Ever.", which qualifies as a good motivator. If we'd said, we're going to make "a smart transport layer that'll connect your moving pieces cheaply and flexibly across your enterprise", we'd have failed.

Highlight (Yellow) | Location 829

> Then your work must be beautiful, immediately useful, and attractive. Your contributors are users who want to explore just a little beyond where they are now. Make it simple, elegant, and brutally clean. The experience when people run or use your work should be an emotional one. They should feel something, and if you accurately solved even just one big problem that until then they didn't quite realize they faced, you'll have a small part of their soul.

Highlight (Yellow) | Location 903

> as founder of a community, you are asking people to invest in your property, trademark, and branding. In return, and this is what we do with ZeroMQ, you can use that branding to set a bar for quality. When you download a product labeled "ZeroMQ", you know that it's been produced to certain standards. It's a basic rule of quality: write down your process; otherwise you cannot improve it.

Highlight (Yellow) | Location 945

> If you don't document the code style you use, you have no basis except prejudice to reject patches.

Highlight (Yellow) | Location 961

> we make software by slowly synthesizing the most accurate knowledge, much as we make Wikipedia articles.

Highlight (Yellow) | Location 973

> Each Contributor SHALL be responsible for identifying themselves in the project Contributor list. In other words, the maintainers are not karma accountants. Anyone who wants credit has to claim it themselves.

Highlight (Yellow) | Location 980

> patch SHOULD be a minimal and accurate answer to exactly one identified and agreed problem. This implements the Simplicity Oriented Design process that I'll come to later in this chapter. One clear problem, one minimal solution, apply, test, repeat.

Highlight (Yellow) | Location 1002

> This is a unapologetic ramming through of thirty years' software design experience. It's a profoundly simple approach to design: make minimal, accurate solutions to real problems, nothing more or less.

Highlight (Yellow) | Location 1004

> we don't have feature requests. Treating new features the same as bugs confuses some newcomers. But this process works, and not just in open source. Enunciating the problem we're trying to solve, with every single change, is key to deciding whether the change is worth making or not.

Highlight (Yellow) | Location 1007

> The user or Contributor SHOULD write the issue by describing the problem they face or observe. "Problem: we need feature X. Solution: make it" is not a good issue. "Problem: user cannot do common tasks A or B except by using a complex workaround. Solution: make feature X" is a decent explanation. Because everyone I've ever worked with has needed to learn this, it seems worth restating: document the real problem first, solution second.

Highlight (Yellow) | Location 1026

> Anyone who submits a patch is a contributor, and all contributors follow the same rules. No special privileges to the original authors, because otherwise we're not building a community, only boosting our egos.

Highlight (Yellow) | Location 1044

> Maintainers MAY merge incorrect patches from other Contributors with the goals of (a) ending fruitless discussions, (b) capturing toxic patches in the historical record, (c) engaging with the Contributor on improving their patch quality. It turns out that accepting imperfect patches rapidly, which I call "optimistic merging", works better all-round than insisting that contributors deliver perfect work.

Highlight (Yellow) | Location 1109

> To misquote Stallman: "your freedom to create an ideal world stops one inch from my application."

Highlight (Yellow) | Location 1114

> New contracts SHOULD be marked as "draft" until they are stable and used by real users.

Highlight (Yellow) | Location 1119

> Old contracts SHOULD be deprecated in a systematic fashion by marking them as "deprecated" and replacing them with new contracts as needed.

Highlight (Yellow) | Location 1159

> We had road maps, and we deleted them. Instead of a few experts trying to lay out the next steps, we were allowing this to happen organically.

Highlight (Yellow) | Location 1166

> by defining the road map, we in effect claimed territory, making it harder for others to participate. People do prefer to contribute to changes they believe were their idea. Writing down a list of things to do turns contribution into a chore rather than an opportunity.

Highlight (Yellow) | Location 1185

> Here thus is an alternative theory of innovation: There is an infinite problem/solution terrain. This terrain changes over time according to external conditions. We can only accurately perceive problems to which we are close. We can rank the cost/benefit economics of problems using a market for solutions. There is an optimal solution to any solvable problem. We can approach this optimal solution heuristically, and mechanically. Our intelligence can make this process faster, but does not replace it. There

---

Highlight (Yellow) | Location 1227

> Ideas are cheap. No exceptions. There are no brilliant ideas. Anyone who tries to start a discussion with "oooh, we can do this too!" should be beaten down with all the passion one reserves for traveling evangelists.

Highlight (Yellow) | Location 1250

> is insanely hard for engineers to stop extending a design to cover more potential problems. They argue, "What if someone wants to do X?" but never ask themselves, "What is the real value of solving X?"

Highlight (Yellow) | Location 1268

> We take the simplest, most dramatic problem and we solve this with a minimal plausible solution, or "patch". Each patch solves exactly a genuine and agreed-upon problem in a brutally minimal fashion.

Highlight (Yellow) | Location 1274

> We do not do anything that is not a patch. We enforce this rule with formal processes that demand that every activity or task is tied to a genuine and agreed-upon problem, explicitly enunciated and documented.

Highlight (Yellow) | Location 1284

> SOD is a hill-climbing algorithm, a reliable way of finding optimal solutions to the most significant problems in an unknown landscape. You don't need to be a genius to use SOD successfully, you just need to be able to see the difference between the fog of activity and the progress towards new real problems.

Highlight (Yellow) | Location 1330

> The Lazy Perfectionist Never design anything that's not a precise minimal answer to a problem we can identify and have to solve.

Highlight (Yellow) | Location 1342

> The Open Door The accuracy of knowledge comes from diversity. The Open Door accepts contributions from almost anyone. She does not argue quality or direction, instead allowing others to argue that and get more engaged. She calculates that even a troll will bring more diverse opinion to the group. She lets the group form its opinion about what goes into stable code, and she enforces this opinion with help of a Benevolent Tyrant.

Highlight (Yellow) | Location 1408

> most successful large-scale software systems are Living Systems. That is, in a competitive market, a Living System will wipe out any competing Planned Systems. It will recognize and solve real problems faster, cheaper, and more accurately. If your business depends on a Planned System, you are vulnerable to attack by a Living System.

Highlight (Yellow) | Location 1412

> when two Living Systems meet, they don't usually fight. Rather, they specialize into different areas, and they then merge to form a single Living System. Competition and conflict usually work for the benefit of the Living System, even if individual components fail.

Highlight (Yellow) | Location 1417

> A Planned System is essentially trying to act as a single individual, and cannot tolerate internal competition, nor failure of individual components.

Highlight (Yellow) | Location 1425

> don't believe individuals components -- including you and me -- can be "intelligent" at all, except in a narrow and superficial sense: intelligence is a property of systems.

Highlight (Yellow) | Location 1486

> components are lazy and opportunistic. They only work when there are tasks waiting, and they only change and grow when there are new, profitable opportunities. This means components can remain lightweight and minimalistic. Further, they can solve the "problem landscape" much more accurately, without excess baggage. In a Planned System by contrast, components are built upfront, on the assumption of future problems, or at best, knowledge of past problems.

Highlight (Yellow) | Location 1512

> Some Living Systems use earned trust, together with identity, in place of verifiable contracts. This can be a valuable short-hand, especially when exchanging knowledge, though it is also vulnerable to cheats (frauds). An alternative is to ensure that every contract is verifiable, backed by meta-contracts on performance. This is often better for trading work. Any taxi driver is fine, so long as drive to the right address and don't over-charge. However we want our news from trusted sources.
