All you wanted to know about Open Development community building but didn't know who to ask

by Stefano Mazzocchi

me

italian

italian

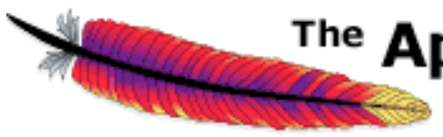
31

research scientist


MIT Libraries

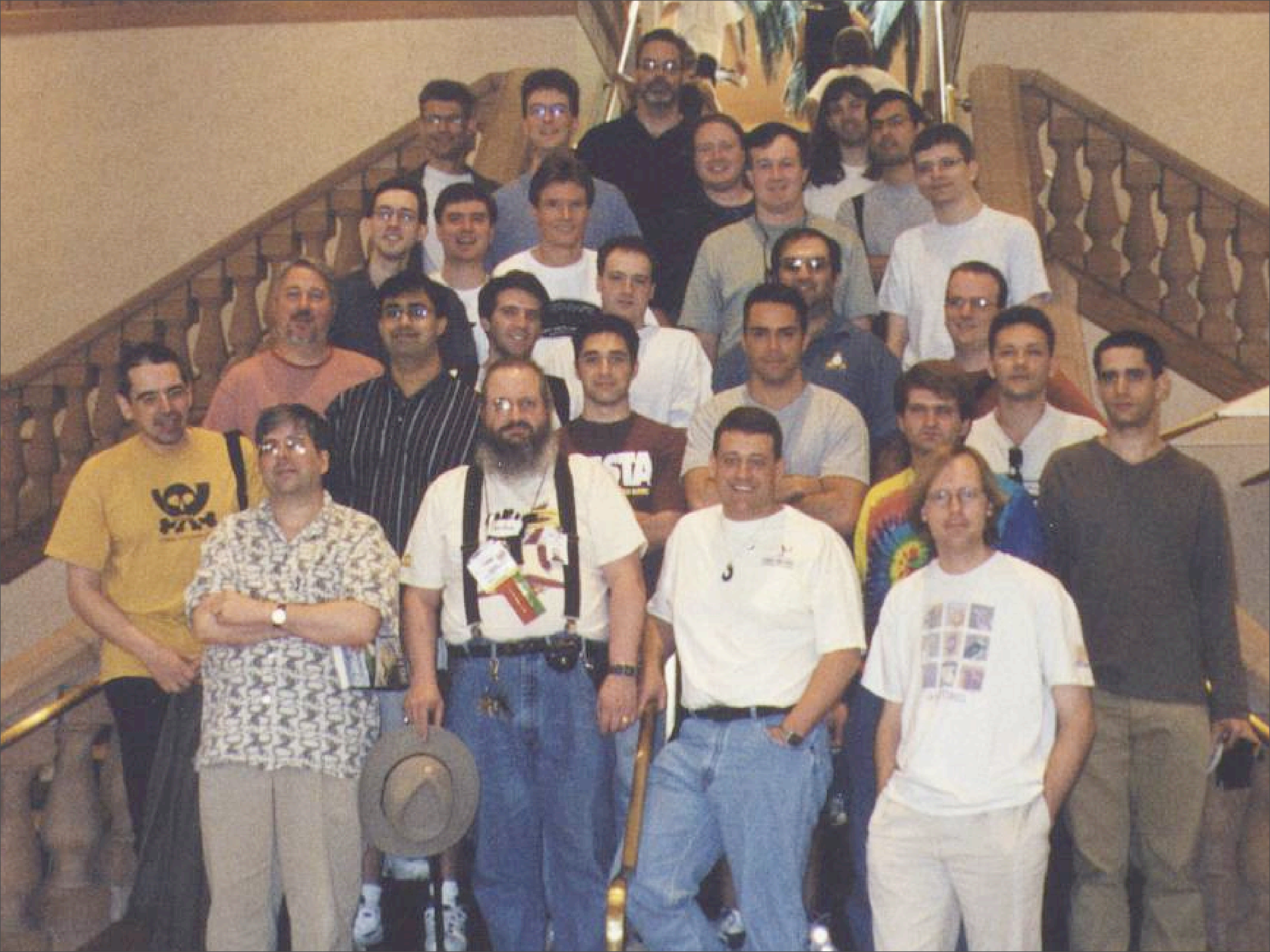
member

former director



The **Apache Software Foundation**

http://www.apache.org/



projects

JServ

[committer, release manager]

JMeter

[original author, committer]

Avalon

[original co-author, committer]

Cocoon

[original author, committer, PMC chair]

James

[original co-designer, committer]

Jakarta

[part of the first PMC]

Ant

[committer]

Tomcat

[committer]

Gump

[committer, PMC member]

Slide

[committer]

Commons

[contributor]

XML

[part of the first PMC]

Forrest

[original author, committer]

Xalan

[community sponsor (pre incubator)]

FOP

[community sponsor (pre incubator)]

Batik

[community sponsor (pre incubator)]

POI

[community sponsor (pre incubator)]

XIndice

[community sponsor (pre incubator)]

Lenya

[incubation sponsor]

Jackrabbit

[incubation co-mentor]

Harmony

[incubator co-sponsor]

Part 1

Communities aren't built, they are grown!

Gardening 101

Gardening 101

- You need a seed or a branch to grow a plant

Gardening 101

- You need a seed or a branch to grow a plant
- You need soil or some other support

Gardening 101

- You need a seed or a branch to grow a plant
- You need soil or some other support
- You need water, air and sun light for photosynthesis to work

Gardening 101

- You need a seed or a branch to grow a plant
- You need soil or some other support
- You need water, air and sun light for photosynthesis to work
- You need to fertilize to drive the plant growth

Incubation Rules

Incubation Rules

- Provide the "growing style" for ASF projects

Incubation Rules

- Provide the "growing style" for ASF projects
  - what kind of seeds (working code + sponsoring members)

Incubation Rules

- Provide the "growing style" for ASF projects
  - what kind of seeds (working code + sponsoring members)
  - what kind of soil (infrastructure)

Incubation Rules

- Provide the "growing style" for ASF projects
  - what kind of seeds (working code + sponsoring members)
  - what kind of soil (infrastructure)
  - exposes to the sun and air (visibility + users)

Incubation Rules

- Provide the "growing style" for ASF projects
  - what kind of seeds (working code + sponsoring members)
  - what kind of soil (infrastructure)
  - exposes to the sun and air (visibility + users)
  - mentors and committers provide water

Leaving the Nursery

Leaving the Nursery

- The plant (project) can leave the nursery (incubation) when:

Leaving the Nursery

- The plant (project) can leave the nursery (incubation) when:
  - there is guarantee that somebody will continue to water it

Leaving the Nursery

- The plant (project) can leave the nursery (incubation) when:
  - there is guarantee that somebody will continue to water it
  - the gardeners know how to grow it and deal with problems and parasites

What's missing?

What's missing?

- Fertilizers

What's missing?

- Fertilizers
- Pesticides

Fertilizers

Fertilizers

- Chemicals that plants need and can't synthesize themselves, mostly:

Fertilizers

- Chemicals that plants need and can't synthesize themselves, mostly:
- Nitrogen (N)

Fertilizers

- Chemicals that plants need and can't synthesize themselves, mostly:
  - Nitrogen (N)
  - Phosphorous (P)

Fertilizers

- Chemicals that plants need and can't synthesize themselves, mostly:
  - Nitrogen (N)
  - Phosphorous (P)
  - Potassium (K)

Fertilizers' Actions

Fertilizers' Actions

- Nitrogen -> grows leaves

Fertilizers' Actions

- Nitrogen -> grows leaves
- Phosphorous -> grows flowers and fruits

Fertilizers' Actions

- Nitrogen -> grows leaves
- Phosphorous -> grows flowers and fruits
- Potassium -> grows roots

Organic vs. Inorganic

Organic vs. Inorganic

- Inorganic fertilizers are water-soluble meaning that are washed away and need to be continuously added or soil is not fertile

Organic vs. Inorganic

- Inorganic fertilizers are water-soluble meaning that are washed away and need to be continuously added or soil is not fertile
- Organic fertilizers are water-insoluble, they are not washed away and keep the soil fertile for longer

Parassites

Parassites

- Most are bad:

Parassites

- Most are bad:
  - Feed off the plant leaves and fruits

Parassites

- Most are bad:
  - Feed off the plant leaves and fruits
  - Use the plant to nest and multiply (causing damage to the plant itself)

Parassites

- Most are bad:
  - Feed off the plant leaves and fruits
  - Use the plant to nest and multiply (causing damage to the plant itself)
- Some are good:

Parassites

- Most are bad:
  - Feed off the plant leaves and fruits
  - Use the plant to nest and multiply (causing damage to the plant itself)
- Some are good:
  - Create a symbiotic relationship that fertilizes the plant

Observations I

Observations I

- All plants need the same basic things, but in different quantities

Observations I

- All plants need the same basic things, but in different quantities
- Plants of the same specie can grow very differently in different environments

Observations I

- All plants need the same basic things, but in different quantities
- Plants of the same specie can grow very differently in different environments
- Even in controlled environments, fertilizers can change how and what the plant grows

Observations II

Observations II

- Organic fertilizers are more expensive and less effective but last longer and are more environmental friendly (reduce soil maintenance needs)

Observations II

- Organic fertilizers are more expensive and less effective but last longer and are more environmental friendly (reduce soil maintenance needs)
- Parasite control is critical but pesticides could also harm the food chain and resources (such as water, soil and air) or other plants

Part 2

Community Fertilizers

good ideas and bad code

“good ideas and bad code” lifecycle

“good ideas and bad code” lifecycle

- if code is good and ideas as good, users just use (committers pool doesn’t grow)

“good ideas and bad code” lifecycle

- if code is good and ideas as good, users just use (committers pool doesn’t grow)
- if ideas are bad, they walk away, no matter how good the code is

“good ideas and bad code” lifecycle

- if code is good and ideas as good, users just use (committers pool doesn’t grow)
- if ideas are bad, they walk away, no matter how good the code is
- if ideas are good, but code is bad, somebody will fix it, learning the codebase and reducing the ‘look under the hood’ cognitive barrier

“good ideas and bad code”

properties

“good ideas and bad code” properties

- harmful in closed development models!

“good ideas and bad code” properties

- harmful in closed development models!
- counterintuitive

“good ideas and bad code” properties

- harmful in closed development models!
- counterintuitive
- clashes with developers’ egos

“good ideas and bad code” properties

- harmful in closed development models!
- counterintuitive
- clashes with developers’ egos
- can be used as an excuse

“good ideas and bad code” properties

- harmful in closed development models!
- counterintuitive
- clashes with developers’ egos
- can be used as an excuse
- can alienate users if discovered

“good ideas and bad code”

when to use

“good ideas and bad code”

when to use

- when you need to grow your committer’s pool

“good ideas and bad code”

when to use

- when you need to grow your committer’s pool
- when you’re not sure of what’s the best design

“good ideas and bad code”

when to use

- when you need to grow your committer’s pool
- when you’re not sure of what’s the best design
- when your reputation as a developer is already established

“good ideas and bad code”

when to use

- when you need to grow your committer’s pool
- when you’re not sure of what’s the best design
- when your reputation as a developer is already established
- when it’s not obvious that your code is sloppy on purpose

“good ideas and bad code”

when not to use

“good ideas and bad code”

when not to use

- when the reputation of your project depends on the perceived quality of your code

“good ideas and bad code”

when not to use

- when the reputation of your project depends on the perceived quality of your code
- when you’re sure of the code design

“good ideas and bad code”

when not to use

- when the reputation of your project depends on the perceived quality of your code
- when you’re sure of the code design
- when your reputation as a developer is not already established

delegate early and often

“delegate early and often” lifecycle

“delegate early and often” lifecycle

- users that care submit patches

“delegate early and often” lifecycle

- users that care submit patches
- you can thank them with praise and a name on the credit list or you can thank them with commit access

“delegate early and often” lifecycle

- users that care submit patches
- you can thank them with praise and a name on the credit list or you can thank them with commit access
- praise and credits don’t increase ownership

“delegate early and often” lifecycle

- users that care submit patches
- you can thank them with praise and a name on the credit list or you can thank them with commit access
- praise and credits don’t increase ownership
- commit access does

“delegate early and often” properties

“delegate early and often” properties

- early delegation increases awareness of openness

“delegate early and often” properties

- early delegation increases awareness of openness
- increases the likelihood of contribution

“delegate early and often” properties

- early delegation increases awareness of openness
- increases the likelihood of contribution
- with limited risks if the infrastructure supports revision control

“delegate early and often”

when to use

“delegate early and often”

when to use

- when you need to increase your development community

“delegate early and often”

when to use

- when you need to increase your development community
- when the energy/time/interest you dedicate to the project is limited and/or decreasing

“delegate early and often”

when to use

- when you need to increase your development community
- when the energy/time/interest you dedicate to the project is limited and/or decreasing
- when the community is aware that quality is not a function of filtering but of proper adaptation feedback

“delegate early and often”

when not to use

“delegate early and often”

when not to use

- when you do NOT need to increase your development community

commit then review

“commit then review” lifecycle

“commit then review” lifecycle

- if you have a patch to apply and you know that nobody else is working on it, apply it without asking for feedback

“commit then review” properties

“commit then review” properties

- minimizes locking (optimistic locking strategy)

“commit then review” properties

- minimizes locking (optimistic locking strategy)
- increases development parallelism

“commit then review” properties

- minimizes locking (optimistic locking strategy)
- increases development parallelism
- counterintuitive for those who are not used to revision controlled environments

“commit then review” properties

- minimizes locking (optimistic locking strategy)
- increases development parallelism
- counterintuitive for those who are not used to revision controlled environments
- harmful for environments without revision control

“commit then review”

when to use

“commit then review”

when to use

- when environment is under revision control (i.e. changes can be easily rolled back)

“commit then review”

when to use

- when environment is under revision control (i.e. changes can be easily rolled back)
- when communication between developers and their activities is effective and up-to-date

“commit then review”

when not to use

“commit then review”

when not to use

- when changes are expensive to roll back

“commit then review”

when not to use

- when changes are expensive to roll back
- when communication is spotty and chances of collisions increase

scratch your own itch

“scratch your own itch” lifecycle

“scratch your own itch” lifecycle

- you find a problem

“scratch your own itch” lifecycle

- you find a problem
- you fix it the way it works for **you**

“scratch your own itch” lifecycle

- you find a problem
- you fix it the way it works for **you**
- without thinking of what other problem you can fix that you have only partial knowledge of

“scratch your own itch”

properties

“scratch your own itch” properties

- forces incremental development

“scratch your own itch” properties

- forces incremental development
- optimizes development energy

“scratch your own itch” properties

- forces incremental development
- optimizes development energy
- avoids unnecessary complexity

“scratch your own itch” properties

- forces incremental development
- optimizes development energy
- avoids unnecessary complexity
- counterintuitive for young and enthusiastic programmers (or those who love to abstract, or those with too much free time on their hands)

“scratch your own itch”

when to use

“scratch your own itch”

when to use

- always

“scratch your own itch”

when to use

- always
- even when your ego is in the way

“scratch your own itch”

when not to use

“scratch your own itch”

when not to use

- never

avoid “flexibility syndrome”

“avoid flexibility syndrome” lifecycle

“avoid flexibility syndrome” lifecycle

- you solve the problem for A

“avoid flexibility syndrome” lifecycle

- you solve the problem for A
- you solve the problem for B

“avoid flexibility syndrome” lifecycle

- you solve the problem for A
- you solve the problem for B
- so you solve the problem for all alphabet letters (even if you only need it solved for A and B)

“avoid flexibility syndrome” properties

“avoid flexibility syndrome” properties

- also known as “yagni” (you ain’t gonna need it!)

“avoid flexibility syndrome” properties

- also known as “yagni” (you ain’t gonna need it!)
- neglected by the same who don’t scratch their own itches

“avoid flexibility syndrome” properties

- also known as “yagni” (you ain’t gonna need it!)
- neglected by the same who don’t scratch their own itches
- causes unnecessary complexity in the architecture and in the social ecosystem (since FS-prone designers tend to need vast consensus)

“avoid flexibility syndrome”

when to use

“avoid flexibility syndrome”

when to use

- always

“avoid flexibility syndrome”

when not to use

“avoid flexibility syndrome”

when not to use

- never

value laziness as a virtue

“value laziness as a virtue” lifecycle

“value laziness as a virtue” lifecycle

- beware of those who want to fix things that work just because of esthetic reasons

“value laziness as a virtue” lifecycle

- beware of those who want to fix things that work just because of esthetic reasons
- show them examples where this caused issues in the past

“value laziness as a virtue” lifecycle

- beware of those who want to fix things that work just because of esthetic reasons
- show them examples where this caused issues in the past
- praise those who commit small contributions more than those who commit big ones

“value laziness as a virtue” properties

“value laziness as a virtue” properties

- optimizes development energy

“value laziness as a virtue” properties

- optimizes development energy
- reduces disruption in the ecosystem

“value laziness as a virtue” properties

- optimizes development energy
- reduces disruption in the ecosystem
- favors incremental development

“value laziness as a virtue” properties

- optimizes development energy
- reduces disruption in the ecosystem
- favors incremental development
- minimizes coordination cost overhead

“value laziness as a virtue” properties

- optimizes development energy
- reduces disruption in the ecosystem
- favors incremental development
- minimizes coordination cost overhead
- counterintuitive (even if still valid!) in closed development environments

“value laziness as a virtue”

when to use

“value laziness as a virtue”

when to use

- always

“value laziness as a virtue”

when not to use

“value laziness as a virtue”

when not to use

- never

“value laziness as a virtue”

when not to use

- never
- even when you have an amazingly productive developer!

small, reversible steps

“small, reversible steps” lifecycle

“small, reversible steps” lifecycle

- development should continue in small, reversible steps

“small, reversible steps” lifecycle

- development should continue in small, reversible steps
- incremental mode

“small, reversible steps” properties

“small, reversible steps” properties

- thermodynamics indicates how entropy (disorder in a system) is not increased by ‘change’, but only by ‘irreversible change’

“small, reversible steps” properties

- thermodynamics indicates how entropy (disorder in a system) is not increased by ‘change’, but only by ‘irreversible change’
- reversible steps do not increase disorder in the system

“small, reversible steps” properties

- thermodynamics indicates how entropy (disorder in a system) is not increased by ‘change’, but only by ‘irreversible change’
- reversible steps do not increase disorder in the system
- small steps reduce coordination costs and maximize the precision on the feedback on the change

"small, reversible steps"

when to use

“small, reversible steps”

when to use

- always!

“small, reversible steps”

when not to use

“small, reversible steps”

when not to use

- never

rules for revolutionaries

“rules for revolutionaries” lifecycle

“rules for revolutionaries” lifecycle

- you can’t avoid revolutions

“rules for revolutionaries” lifecycle

- you can’t avoid revolutions
- sometimes they are needed, sometimes they are not

“rules for revolutionaries” lifecycle

- you can’t avoid revolutions
- sometimes they are needed, sometimes they are not
- and you can’t know in advance

“rules for revolutionaries” lifecycle

- you can’t avoid revolutions
- sometimes they are needed, sometimes they are not
- and you can’t know in advance
- so define ahead of time what to do with revolutionaries

“rules for revolutionaries” properties

“rules for revolutionaries” properties

- creates an escape valve for those who feel locked into incremental development

“rules for revolutionaries” properties

- creates an escape valve for those who feel locked into incremental development
- allows innovation without community destabilization

“rules for revolutionaries”

when to use

“rules for revolutionaries”

when to use

- when creativity and innovation is being blocked by the ‘small, incremental steps’ pattern

“rules for revolutionaries”

when to use

- when creativity and innovation is being blocked by the ‘small, incremental steps’ pattern
- when a developer feels the need for a clean slate to show his/her peers his/her intentions

“rules for revolutionaries”

when not to use

“rules for revolutionaries”

when not to use

- when a developer feels constrained by the ‘small, incremental steps’ pattern and wants to find ways to avoid it

the busy list pattern

“the busy list pattern” lifecycle

“the busy list pattern” lifecycle

- somebody suggests that the mailing list is too noisy and should be split in multiple ones

“the busy list pattern” properties

“the busy list pattern” properties

- restaurants and night clubs know that “packed” rooms help marketing

“the busy list pattern” properties

- restaurants and night clubs know that “packed” rooms help marketing
- strive to keep the list as “packed” as possible

“the busy list pattern” properties

- restaurants and night clubs know that “packed” rooms help marketing
- strive to keep the list as “packed” as possible
- but not too packed

“the busy list pattern” properties

- restaurants and night clubs know that “packed” rooms help marketing
- strive to keep the list as “packed” as possible
- but not too packed
- when necessary, split a list by audience (dev/users), not by subject

“the busy list pattern”

when to use

“the busy list pattern”
when to use

- as much as possible, open developers have a selected ability to process very large email volumes

“the busy list pattern”
when to use

- as much as possible, open developers have a selected ability to process very large email volumes
- suggest the use [...] subject prefixes to “tag” topics instead of splitting the mail list, if complains persist or if the mail list is really too verbose (100 msg/day)

“the busy list pattern”

when not to use

“the busy list pattern”

when not to use

- never

"the busy list pattern"
when not to use

- never
- especially not when people suggest it as a way to help them manage their own email filters

always reply

“always reply” lifecycle

“always reply” lifecycle

- somebody writes an email

“always reply” lifecycle

- somebody writes an email
- nobody replies for a while

“always reply” lifecycle

- somebody writes an email
- nobody replies for a while
- you must reply something

“always reply” properties

“always reply” properties

- even if your reply doesn’t solve the issue indicate in the email, it makes the sender (and all the other lurkers!) feel welcome

“always reply” properties

- even if your reply doesn’t solve the issue indicate in the email, it makes the sender (and all the other lurkers!) feel welcome
- it also establishes you as part of the core of the social network

“always reply”

when to use

“always reply”
when to use

- always

“always reply”
when to use

- always
- even if the message is inflammatory (more patterns on that later)

“always reply”

when not to use

“always reply”

when not to use

- never

Part 3

Community Pesticides

In every community,
there is a sociopath

In every community, there is a sociopath

If you can't spot the sociopath, it's you!

In every community, there is a sociopath

If you can't spot the sociopath, it's you!

[inspired by Mark Pilgrim & Joe Gregorio]

double email

“double email” lifecycle

“double email” lifecycle

- somebody writes something that triggers in you a negative emotional response (anger, frustration, despair, sadness, pain)

“double email” lifecycle

- somebody writes something that triggers in you a negative emotional response (anger, frustration, despair, sadness, pain)
- reply trying to ‘vindicate’ your state by inflicting it back on the offender

“double email” lifecycle

- somebody writes something that triggers in you a negative emotional response (anger, frustration, despair, sadness, pain)
- reply trying to ‘vindicate’ your state by inflicting it back on the offender
- when about to hit ‘send’, hit ‘delete’

“double email” lifecycle

- somebody writes something that triggers in you a negative emotional response (anger, frustration, despair, sadness, pain)
- reply trying to ‘vindicate’ your state by inflicting it back on the offender
- when about to hit ‘send’, hit ‘delete’
- write another email until you can send it without emotional attachment

“double email” properties

“double email” properties

- effectively removes emotional negativity from the communication channel

“double email” properties

- effectively removes emotional negativity from the communication channel
- teaches emotional control

“double email” properties

- effectively removes emotional negativity from the communication channel
- teaches emotional control
- makes lurkers feel welcome

“double email”

when to use

“double email”

when to use

- when you feel that your reply contains negative emotions toward the person you are replying to

“double email”

when not to use

“double email”

when not to use

- when your message contains no negative emotion

flame the flamer's flamer

“flame the flamer’s flamer” lifecycle

“flame the flamer’s flamer” lifecycle

- somebody writes an inflammatory email

“flame the flamer’s flamer” lifecycle

- somebody writes an inflammatory email
- somebody else flames the flamer

“flame the flamer’s flamer” lifecycle

- somebody writes an inflammatory email
- somebody else flames the flamer
- you publicly flame the flamer’s flamer

“flame the flamer’s flamer” lifecycle

- somebody writes an inflammatory email
- somebody else flames the flamer
- you publicly flame the flamer’s flamer
- you privately contact the flamer’s flamer telling him about this pattern

“flame the flamer’s flamer” properties

“flame the flamer’s flamer” properties

- flamers like to feel unwelcomed, restricted or otherwise hated

"flame the flamer's flamer" properties

- flamers like to feel unwelcomed, restricted or otherwise hated
- the flamer's flamer is trying to protect the community (but doesn't know the "double email" pattern)

"flame the flamer's flamer" properties

- flamers like to feel unwelcomed, restricted or otherwise hated
- the flamer's flamer is trying to protect the community (but doesn't know the "double email" pattern)
- flaming the flamer's flamer will confuse the flamer, who normally leaves

"flame the flamer's flamer" properties

- flamers like to feel unwelcomed, restricted or otherwise hated
- the flamer's flamer is trying to protect the community (but doesn't know the "double email" pattern)
- flaming the flamer's flamer will confuse the flamer, who normally leaves
- private contact with the flamer's flamer guarantees a respectful relationship

“flame the flamer’s flamer”

when to use

“flame the flamer’s flamer”

when to use

- when you are respected member of the community

“flame the flamer’s flamer”

when not to use

“flame the flamer’s flamer”

when not to use

- when your flaming the flamer’s flamer could be interpreted by others as just another flame

archives are forever

“archives are forever”
lifecycle

“archives are forever” lifecycle

- somebody starts a flame

“archives are forever” lifecycle

- somebody starts a flame
- you reply indicating that last time you hired somebody for a job, you looked up their email communication behavior on the web.

“archives are forever”
properties

“archives are forever” properties

- takes the wind out of most flamer’s pipes

“archives are forever” properties

- takes the wind out of most flamer’s pipes
- informs others potential flamers

“archives are forever” properties

- takes the wind out of most flamer’s pipes
- informs others potential flamers
- it’s emotionally neutral (won’t feed the fire)

“archives are forever”

when to use

“archives are forever”

when to use

- when other emotionally neutral responses have no effect

“archives are forever”

when not to use

“archives are forever”

when not to use

- when the flamer is already widely known as such and has accepted his destiny (yes, ‘his’ because it’s never a woman!)

fear balkanization

“fear balkanization” lifecycle

“fear balkanization” lifecycle

- system is highly modular

“fear balkanization” lifecycle

- system is highly modular
- modules become ‘owned’ by their maintainers

“fear balkanization” lifecycle

- system is highly modular
- modules become ‘owned’ by their maintainers
- modules become ‘balkanized’ as people tend to write a similar module instead of coordinating with the owner

“fear balkanization” properties

“fear balkanization” properties

- turns a positive design (modularity) into a dangerous practice (isolation)

“fear balkanization” properties

- turns a positive design (modularity) into a dangerous practice (isolation)
- hardly reversible

“fear balkanization” properties

- turns a positive design (modularity) into a dangerous practice (isolation)
- hardly reversible
- very infective

“fear balkanization”

when to use

“fear balkanization”

when to use

- when a system is modular

“fear balkanization”

when to use

- when a system is modular
- when modules have less than 3 developers actively maintaining it

“fear balkanization”

when to use

- when a system is modular
- when modules have less than 3 developers actively maintaining it
- when developers invoke the ‘revolutionary’ principle to avoid coordination with an existing module owners

“fear balkanization”

when not to use

“fear balkanization”

when not to use

- when systems aren’t modular

“fear balkanization”

when not to use

- when systems aren’t modular
- when each module is ‘owned’ by at least three developers

Part 4

Tools

Apache Agora

Apache Agora

- Interactive Community Visualizer
- Emerges social network by mining email communication in mailing lists
- Completely automatic
- http://people.apache.org/~stefano/agora/

??

Part 5

Ending Notes and Disclaimers

please, take with a grain of salt

yes, I come from a family of farmers

you do need a “green thumb” with communities as well

some communities die

some communities die

jserv

some communities die

jserv

xindice

sometimes **you** have to kill them

sometimes **you** have to kill them

avalon

and be able to move on

Concepts, slides and pictures by

Stefano Mazzocchi

<stefano@apache.org>

<stefanom@mit.edu>

When you know a thing, to hold that you know it;
and when you do not know a thing, to allow that
you do not know it - this is knowledge.

Confucius

The end