Design After Agile

@stuarthalloway

Datomic

design

to prepare the plans for (a work to be executed), especially to plan the form and structure of

design

to prepare the plans for (a work to be executed), especially to plan the form and structure of

~~decide upon the look of~~

design

to prepare the plans for (a work to be executed), especially to plan the form and structure of

~~decide upon the look of~~

make a plan, write it down

agility

ability to move quickly and easily

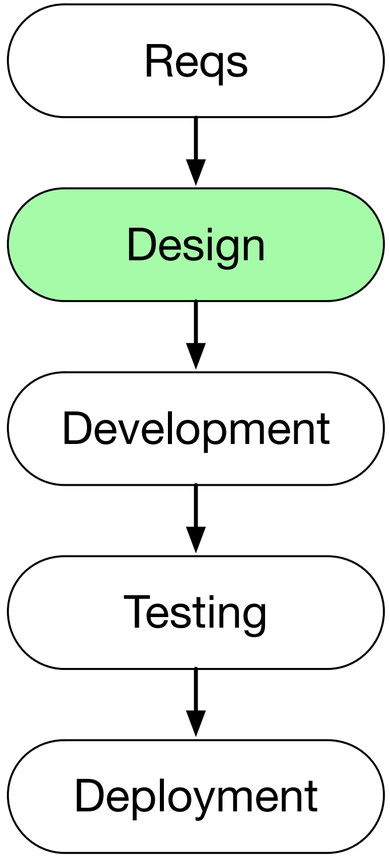
nimbleness, quickness

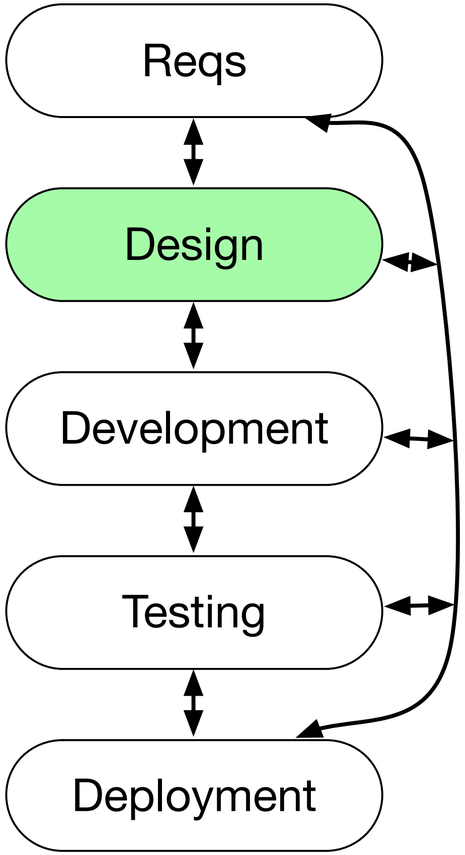
The Manifesto

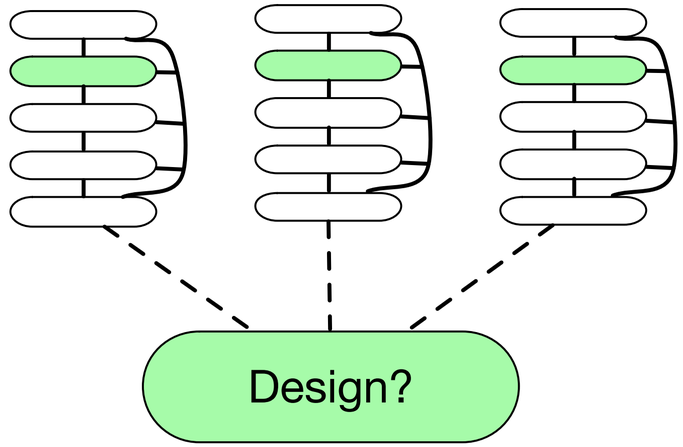
Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

https://agilemanifesto.org/

Fixing Waterfall







Waterfall ----► Feedback Loops ----► Smaller Increments

The Manifesto

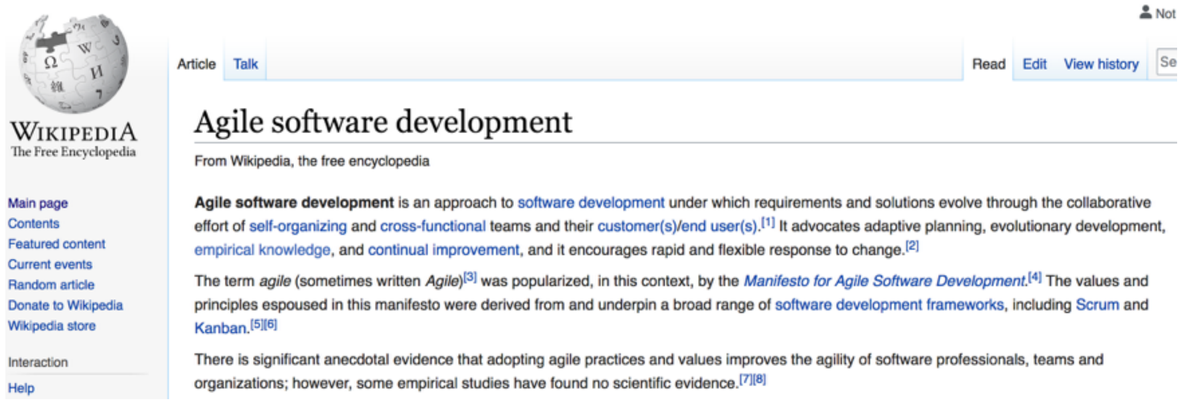
Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over *following a plan*

uh-oh

https://agilemanifesto.org/

“**plans** are useless but **planning** is indispensable”

Agile Without Design … Isn’t



WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help

Not

Article Talk Read Edit View history Se

Agile software development

From Wikipedia, the free encyclopedia

Agile software development is an approach to software development under which requirements and solutions evolve through the collaborative effort of self-organizing and cross-functional teams and their customer(s)/end user(s).[1] It advocates adaptive planning, evolutionary development, empirical knowledge, and continual improvement, and it encourages rapid and flexible response to change.[2]

The term *agile* (sometimes written *Agile*)[3] was popularized, in this context, by the *Manifesto for Agile Software Development*.[4] The values and principles espoused in this manifesto were derived from and underpin a broad range of software development frameworks, including Scrum and Kanban.[5][6]

There is significant anecdotal evidence that adopting agile practices and values improves the agility of software professionals, teams and organizations; however, some empirical studies have found no scientific evidence.[7][8]

6.2 Pitfalls
Lack of overall product design

https://en.wikipedia.org/wiki/Agile_software_development#Common_agile_software_development_pitfalls

https://en.wikipedia.org/wiki/Pitfall!

Design

design is taking things apart

make diagrams

make tables

write prose

Design is Taking Things Apart

requirements

time / order / flow

place / participants

information / mechanisms

solutions

https://github.com/matthiasn/talk-transcripts/blob/master/Hickey_Rich/DesignCompositionPerformance.md

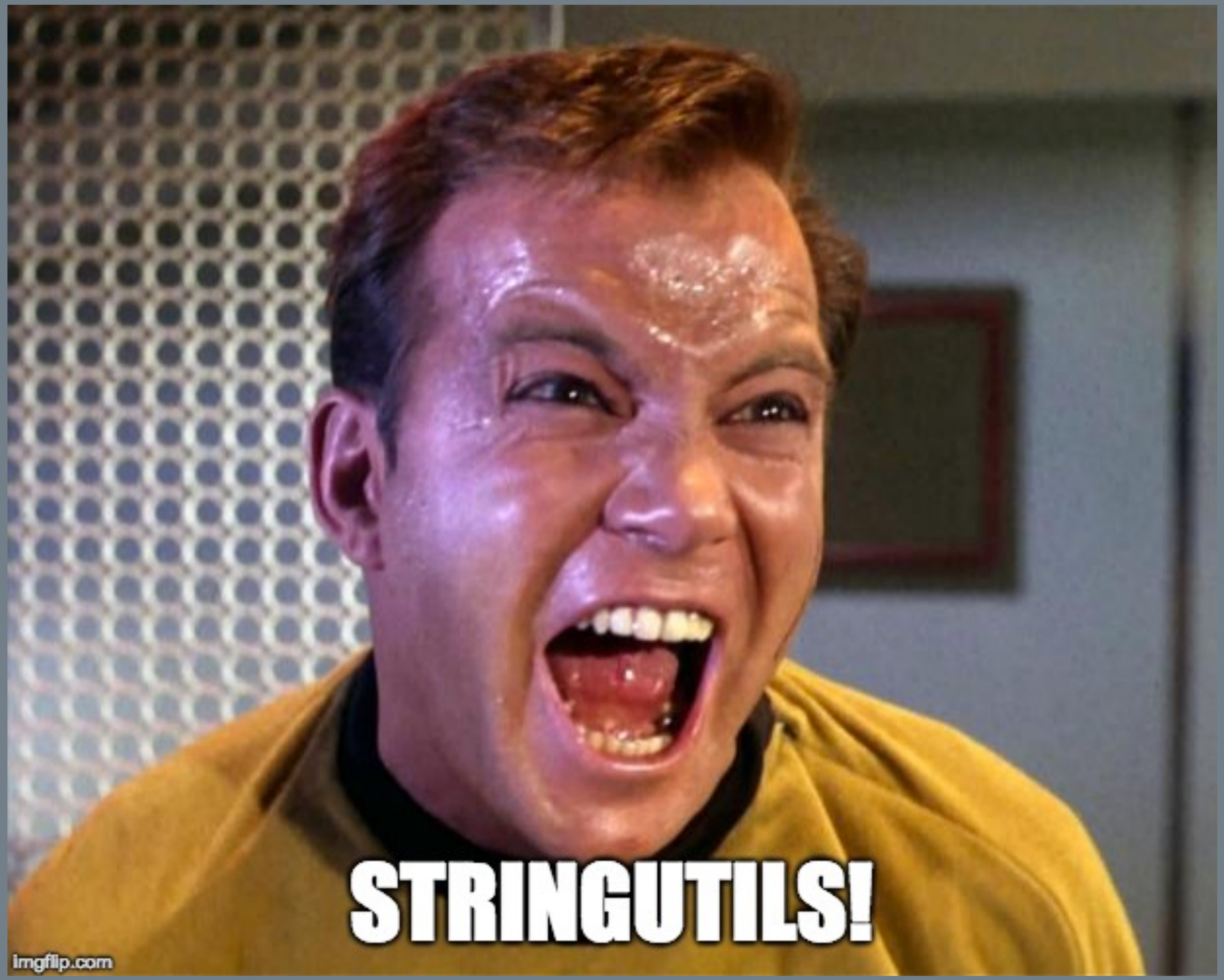
Designing Polymorphism

interfaces

implementations

extension

language	interface	impl	extension
Java	interface definition	class definition	class definition
Clojure	protocol	type	extend


STRINGUTILS!
imgflip.com

Make Diagrams

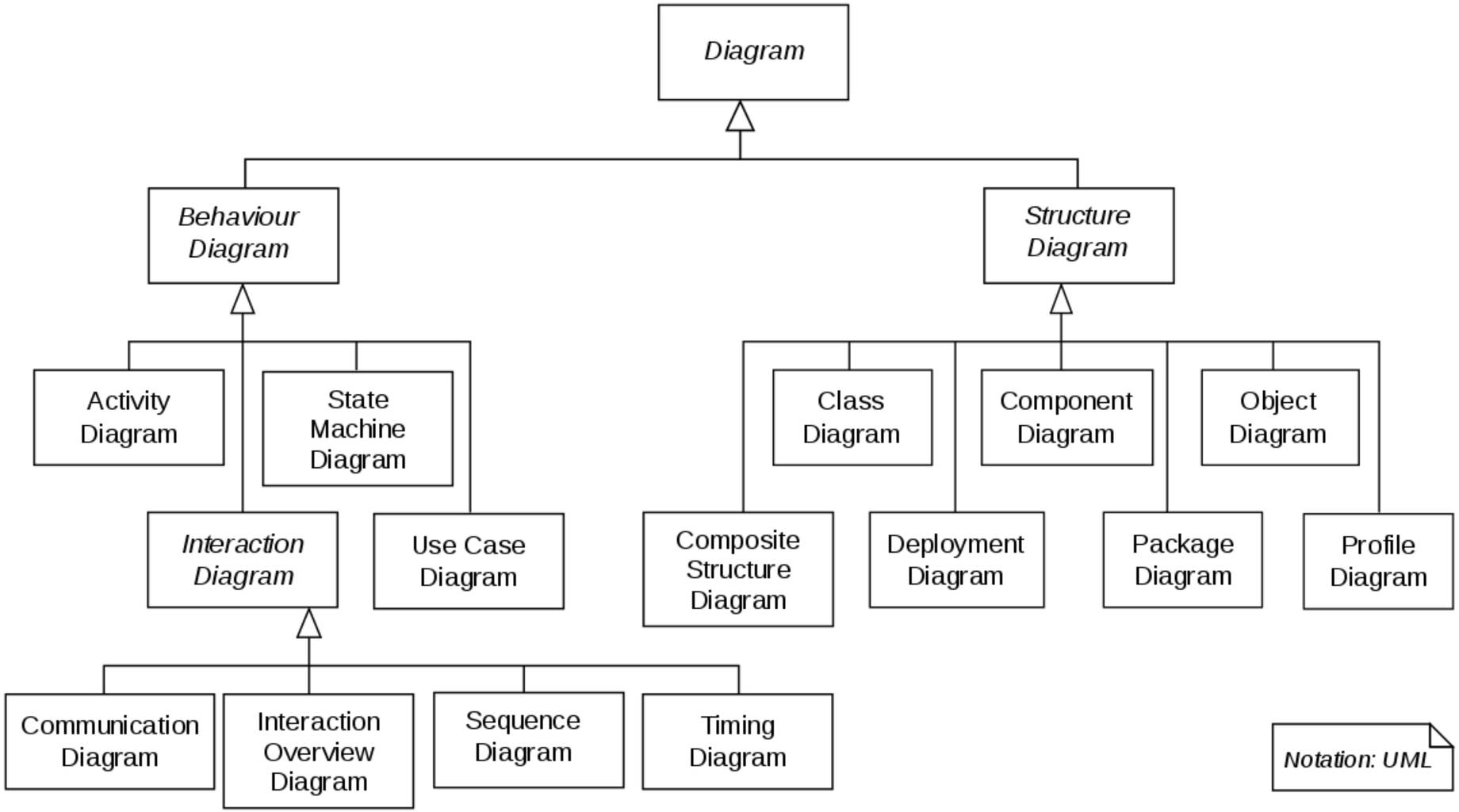
design is taking things apart

make diagrams

make tables

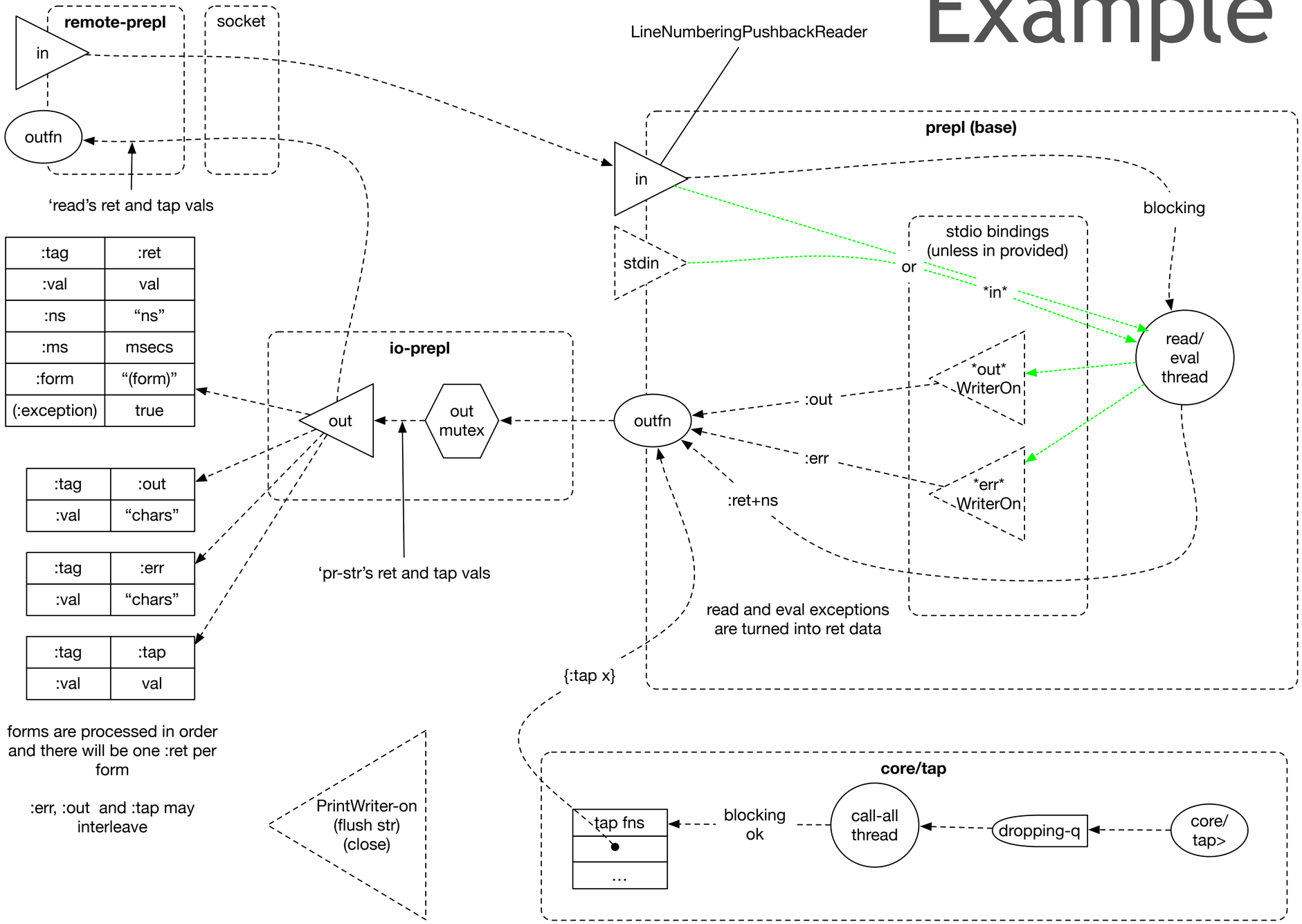
write prose

Don’t Let UML Drag You Down



https://en.wikipedia.org/wiki/Unified_Modeling_Language

Example


remote-prepl
socket
in
outfn
'read's ret and tap vals
LineNumberingPushbackReader
prepl (base)
in
blocking
stdio bindings
(unless in provided)
stdin
or
in
read/
eval
thread
:tag :ret
:val val
:ns "ns"
:ms msecs
:form "(form)"
(:exception) true
io-prepl
out
out
mutex
outfn
out
WriterOn
:out
:err
err
WriterOn
:ret+ns
:tag :out
:val "chars"
:tag :err
:val "chars"
:tag :tap
:val val
'pr-str's ret and tap vals
read and eval exceptions
are turned into ret data
{:tap x}
forms are processed in order
and there will be one :ret per
form
:err, :out and :tap may
interleave
PrintWriter-on
(flush str)
(close)
core/tap
tap fns
...
blocking
ok
call-all
thread
dropping-q
core/
tap>

Make Diagrams

combine structure, flow, invocation, etc. in one place

keep everything!

- separate diagram per option explored

design is process, not product

- include questions

Make Tables

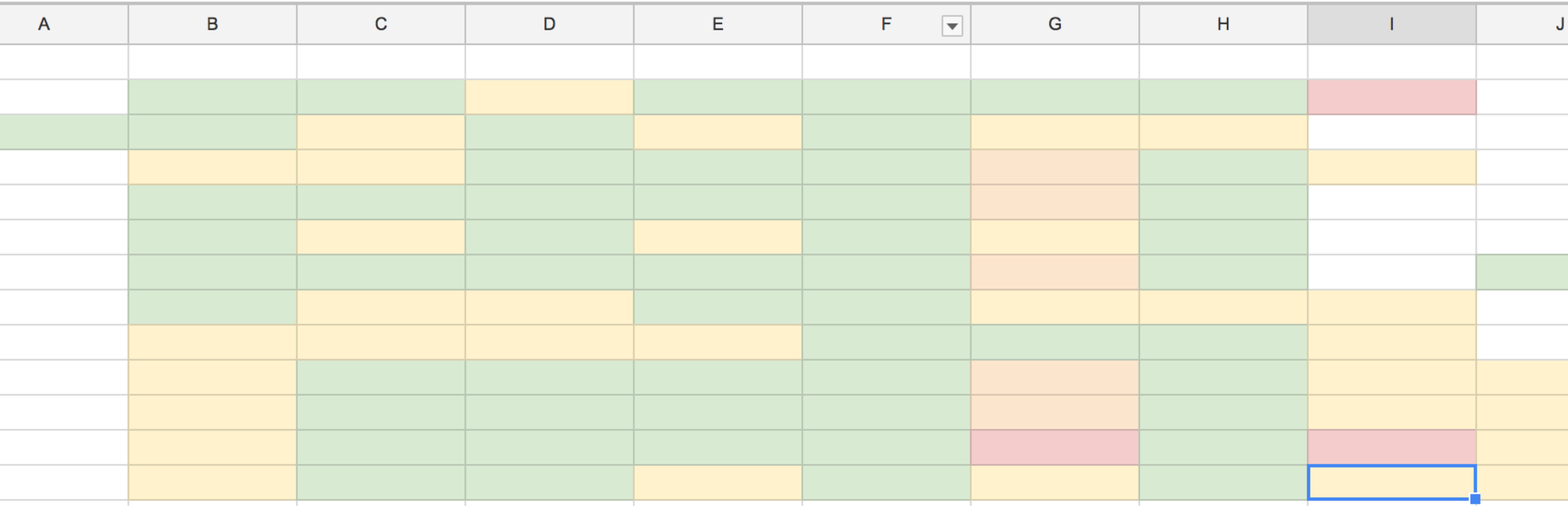
design is taking things apart

make diagrams

make tables

write prose

	A	B	C
1	User wants	VPC Link	Endpoint Service
2	Client Ion	No	No
3	AWS Events	No	No
4	Expose to other service through VPC	No	Yes
5	Expose to Internet through API Gateway	No	No
6	High Perf expose to Internet	Yes	No
7	Both a service exposed through VPC and high perf exposed to Internet	Yes	Yes
8	Expose to service in my VPC	No	No
9			


A
B
C
D
E
F
G
H
I
J

Make Tables

consider multiple options

- against multiple positive *and negative* factors

merely naming the rows and columns helps

- it make take several tries

keep everything!, include ?s

Write Good Prose

design is taking things apart

make diagrams

make tables

write prose

Iterating on Problem Statement

“developers hate ugly stacktraces”

take apart context (users, ops, programmers, REPL)

take apart errors (data, interpretation, serialization)

take apart consumers (human vs. program)

Prose Example: CLJ-2373

Problems

- Discussions about "make errors better" often conflate error message, stacktraces, exception data
- Errors do not clearly indicate in what "phase" of execution they occur (read, compile, macroexpand, evaluation, printing)
- Errors during macroexpansion do not capture the location in caller source (like read and compile exceptions do), and thus the wrong "location" is reported
- Clojure prints data into Throwable messages
  - big and ugly
  - outside user control

Principles

- exception messages should be small strings
  - with the expectation that `getMessage` will be printed
  - totally controlled by throwing code
  - never print arbitrary data into a message
- flow data all the way to the edge
  - `ex-info` & `ex-data`
  - macroexpand spec errors have a well known format
- edge printing should be concise by default, configurable by user
  - edge functions like `repl-caught` respect print and spec bindings
  - concise summary by default
  - all the details when you want

Proposed impl changes

- stop printing data into message strings
  - `ExceptionInfo.toString` should list keys, not entire map contents
  - `spec.alpha/macroexpand-check` should stop including `explain-out` in message
- and instead print data (configurably) at the edges `repl/pst` and `main/repl-caught`
  - print spec per `explain-out`
  - print `ExceptionInfo` keys
- make `CompilerException` wrappers special at print time instead of construct time
  - CE message is "CompilerException + file/line"
  - wrapped message is whatever it is
  - edge printers should print both

Discussion

Some Prose Examples

problem statements: CLJ-2373

principles: CLJ-2373

proposals: CLJ-2373

rationales: spec, ClojureScript, Clojure

Fallacies

not enough time for design

design and code will get out of sync

design emerges from an agile process

No Time For What? Maps? Steering?

https://weheartit.com/entry/133813285

Accumulate, Don’t Coordinate

you keep every iteration of your code

without complaining that last year is out of sync with this year

every artifact (code or design) captures a moment in time

including understanding and misconception

some code gets more attention than other code

so also with design

“Emergent Design”

https://www.extremetech.com/extreme/97311-millions-of-monkeys-kind-of-recreate-a-shakespearean-poem

Agile: The (Many) Good Parts

putting people first

continuous activities with feedback loops

- not discrete phases that end

designers can and do write code

evolutionary architecture

Agile makes
change cheap

design produces
valuable change

Prioritize Design

add design activities and artifacts to your Agile workflow

- not as window dressing
- design is a skill, give yourself time to learn it
- then make design pay its way like anything else

pair on design, not code

Design After Agile

@stuarthalloway