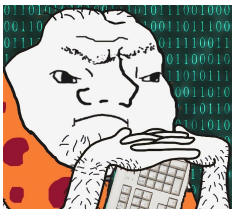


The Grug Brained Developer

A layman's guide to thinking like the self-aware smol brained

Introduction

this collection of thoughts on software development gathered by grug brain developer

grug brain developer not so smart, but grug brain developer program many long year and learn some things although mostly still confused

grug brain developer try collect learns into small, easily digestible and funny page, not only for you, the young grug, but also for him because as grug brain developer get older he forget important things, like what had for breakfast or if put pants on

big brained developers are many, and some not expected to like this, make sour face

THINK they are big brained developers many, many more, and more even definitely probably maybe not like this, many sour face (such is internet)

(note: grug once think big brained but learn hard way)

is fine!

is free country sort of and end of day not really matter too much, but grug hope you fun reading and maybe learn from many, many mistake grug make over long program life

The Eternal Enemy: Complexity

apex predator of grug is complexity

complexity bad

say again:

complexity *very* bad

you say now:

complexity *very*, *very* bad

given choice between complexity or one on one against t-rex, grug take t-rex: at least grug see t-rex

complexity is spirit demon that enter codebase through well-meaning but ultimately very clubbable non grug-brain developers and project managers who not fear complexity spirit demon or even know about sometime

one day code base understandable and grug can get work done, everything good!

next day impossible: complexity demon spirit has entered code and very dangerous situation!

grug no able see complexity demon, but grug sense presence in code base

demon complexity spirit mocking him make change here break unrelated thing there what!?! mock mock mock ha ha so funny grug love programming and not becoming shiney rock speculator like grug senior advise

club not work on demon spirit complexity and bad idea actually hit developer who let spirit in with club: sometimes grug himself!

sadly, often grug himself

so grug say again and say often: complexity *very*, *very* bad

Saying No

best weapon against complexity spirit demon is magic word: "no"

"no, grug not build that feature"

"no, grug not build that abstraction"

"no, grug not put water on body every day or drink less black think juice you stop repeat ask now"

note, this good engineering advice but bad career advice: "yes" is magic word for more shiney rock and put in charge of large tribe of developer

sad but true: learn "yes" then learn blame other grugs when fail, ideal career advice

but grug must to grug be true, and "no" is magic grug word. Hard say at first, especially if you nice grug and don't like disappoint people (many such grugs!) but easier over time even though shiney rock pile not as high as might otherwise be

is ok: how many shiney rock grug really need anyway?

Saying ok

sometimes compromise necessary or no shiney rock, mean no dinosaur meat, not good, wife firmly remind grug about young grugs at home need roof, food, and so forth, no interest in complexity demon spirit rant by grug for fiftieth time

in this situation, grug recommend "ok"

"ok, grug build that feature"

then grug spend time think of 80/20 solution to problem and build that instead.
80/20 solution say "80 want with 20 code" solution maybe not have all bell-whistle that project manager want, maybe a little ugly, but work and deliver most value, and keep demon complexity spirit at bay for most part to extent

sometimes probably best just not tell project manager and do it 80/20 way. easier forgive than permission, project managers mind like butterfly at times overworked and dealing with many grugs. often forget what even feature supposed to do or move on or quit or get fired grug see many such cases

anyway is in project managers best interest anyway so grug not to feel too bad for this approach usually

Factoring Your Code

next strategy very harder: break code base up properly (fancy word: "factor your code properly") here is hard give general advice because each system so different. however, one thing grug come to believe: not factor your application too early!

early on in project everything very abstract and like water: very little solid holds for grug's struggling brain to hang on to. take time to develop "shape" of system and learn what even

doing. grug try not to factor in early part of project and then, at some point, good cut-points emerge from code base

good cut point has narrow interface with rest of system: small number of functions or abstractions that hide complexity demon internally, like trapped in crystal

grug quite satisfied when complexity demon trapped properly in crystal, is best feeling to trap mortal enemy!

grug try watch patiently as cut points emerge from code and slowly refactor, with code base taking shape over time along with experience. no hard/ fast rule for this: grug know cut point when grug see cut point, just take time to build skill in seeing, patience

sometimes grug go too early and get abstractions wrong, so grug bias towards waiting

big brain developers often not like this at all and invent many abstractions start of project

grug tempted to reach for club and yell "big brain no maintain code! big brain move on next architecture committee leave code for grug deal with!"

but grug learn control passions, major difference between grug and animal

instead grug try to limit damage of big brain developer early in project by giving them thing like UML diagram (not hurt code, probably throw away anyway) or by demanding working demo tomorrow

working demo especially good trick: force big brain make something to actually work to talk about and code to look at that do thing, will help big brain see reality on ground more quickly

remember! big brain have big brain! need only be harness for good and not in service of spirit complexity demon on accident, many times seen

(best grug brain able to herd multiple big brain in right direction and produce many complexity demon trap crystals, large shiney rock pile awaits such grug!)

also sometimes call demo approach "prototype", sound fancier to project manager

grug say prototype early in software making, *especially* if many big brains

Testing

grug have love/hate relationship with test: test save grug many, many uncountable time and grug love and respect test

unfortunately also many test shamans exist. some test shaman make test idol, demand things like "first test" before grug even write code or have any idea what grug doing domain!

how grug test what grug not even understand domain yet!?

"Oh, don't worry: the tests will show you what you need to do."

grug once again catch grug slowly reaching for club, but grug stay calm

grug instead prefer write most tests after prototype phase, when code has begun firm up

but, note well: grug must here be very disciplined!

easy grug to move on and not write tests because "work on grugs machine"!

this very, very bad: no guarantee work on other machine and no guarantee work on grug machine in future, many times

test shaman have good point on importance of test, even if test shaman often sometimes not complete useful feature in life and talk only about test all time, deserve of club but heart in right place

also, test shaman often talk unit test very much, but grug not find so useful. grug experience that ideal tests are not unit test or either end-to-end test, but in-between test

unit tests fine, ok, but break as implementation change (much compared api!) and make refactor hard and, frankly, many bugs anyway often due interactions other code. often throw away when code change.

grug write unit test mostly at start of project, help get things going but not get too attached or expect value long time

end to end tests good, show whole system work, but! hard to understand when break and drive grug crazy very often, sometimes grugs just end up ignoring because "oh, that break all time"

very bad!

in-between tests, grug hear shaman call "integration tests" sometime often with sour look on face. but grug say integration test sweet spot according to grug: high level enough test correctness of system, low level enough, with good debugger, easy to see what break

grug prefer some unit tests especially at start but not 100% all code test and definitely not "first test". "test along the way" work pretty well for grug, especially as grug figure things out

grug focus much ferocious integration test effort as cut point emerge and system stabilize! cut point api hopefully stable compared implementation and integration test remain valuable many long time, and easy debug

also small, well curated end-to-end test suite is created to be kept working religiously on pain of clubbing. focus of important end-to-end test on most common UI features and few most important edge cases, but not too many or become impossible maintain and then ignored

this ideal set of test to grug

you may not like, but this peak grug testing

also, grug dislike mocking in test, prefer only when absolute necessary to (rare/never) and coarse grain mocking (cut points/systems) only at that

one exception "first test" dislike by grug: when bug found. grug always try first reproduce bug with regression test *then* fix bug, this case only for some reason work better

Agile

grug think agile not terrible, not good

end of day, not worst way to organize development, maybe better than others grug supposes is fine

danger, however, is agile shaman! many, many shiney rock lost to agile shaman!

whenever agile project fail, agile shaman say "you didn't do agile right!" grug note this awfully convenient for agile shaman, ask more shiney rock better agile train young grugs on agile, danger!

grug tempted reach for club when too much agile talk happen but always stay calm

prototyping, tools and hiring good grugs better key to success software: agile process ok and help some but sometimes hurt taken too seriously

grug say no silver club fix all software problems no matter what agile shaman say (danger!)

Refactoring

refactoring fine activity and often good idea, especially later in project when code firmed up

however, grug note that many times in career "refactors" go horribly off rails and end up causing more harm than good

grug not sure exactly why some refactors work well, some fail, but grug notice that larger refactor, more likely failure appear to be

so grug try to keep refactors relatively small and not be "too far out from shore" during refactor. ideally system work entire time and each step of finish before other begin.

end-to-end tests are life saver here, but often very hard understand why broke... such is refactor life.

also grug notice that introducing too much abstraction often lead to refactor failure and system failure. good example was J2EE introduce, many big brain sit around thinking too much abstraction, nothing good came of it many project hurt

another good example when company grug work for introduce OSGi to help manage/trap spriit complexity demon in code base. not only OSGi not help, but make complexity demon much more powerful! took multiple man year of best developers to rework as well to boot! more complex spirit and now features impossible implement! very bad!

Chesterton's Fence

wise grug shaman chesterton once say

> here exists in such a case a certain institution or law; let us say, for the sake of simplicity, a fence or gate erected across a road. The more modern type of reformer

> goes gaily up to it and says, “I don’t see the use of this; let us clear it away.” To which the more intelligent type of reformer will do well to answer: “If you don’t see the use of it, I certainly won’t let you clear it away. Go away and think. Then, when you can come back and tell me that you do see the use of it, I may allow you to destroy it.”

many older grug learn this lesson well not start tearing code out willy nilly, no matter how ugly look

grug understand all programmer platonists at some level wish music of spheres perfection in code. but danger is here, world is ugly and gronky many times and so also must code be

humility not often come big brained or think big brained easily or grug even, but grug often find "oh, grug no like look of this, grug fix" lead many hours pain grug and no better or system worse even

grug early on in career often charge into code base waving club wildly and smash up everything, learn not good

grug not say no improve system ever, quite foolish, but recommend take time understand system first especially bigger system is and is respect code working today even if not perfect

here tests often good hint for why fence not to be smashed!

Microservices

grug wonder why big brain take hardest problem, factoring system correctly, and introduce network call too

seem very confusing to grug

Tools

grug love tool. tool and control passion what separate grug from dinosaurs! tool allow grug brain to create code that not possible otherwise by doing thinking for grug, always good relief! grug always spend time in new place learning tools around him to maximize productivity: learn tools for two weeks make development often twice faster and often have dig around ask other

developers help, no docs

code completion in IDE allow grug not have remembered all API, very important!

java programming nearly impossible without it for grug!

really make grug think some time

good debugger worth weight in shiney rocks, in fact also more: when faced with bug grug would often trade all shiney rock and perhaps few children for good debugger and anyway debugger no weigh anything far as grug can tell

grug always recommend new programmer learn available debugger very deeply, features like conditional break points, expression evaluation, stack navigation, etc teach new grug more about computer than university class often!

grug say never be not improving tooling

Type Systems

grug very like type systems make programming easier. for grug, type systems most value when grug hit dot on keyboard and list of things grug can do pop up magic. this 90% of value of type system or more to grug

big brain type system shaman often say type correctness main point type system, but grug note some big brain type system shaman not often ship code. grug suppose code never shipped is correct, in some sense, but not really what grug mean when say correct

grug say tool magic pop up of what can do and complete of code major most benefit of type system, correctness also good but not so nearly so much

also, often sometimes caution beware big brains here!

some type big brain think in type systems and talk in lemmas, potential danger!

danger abstraction too high, big brain type system code become astral projection of platonic generic turing model of computation into code base. grug confused and agree some level very elegant but also very hard do anything like record number of club inventory for Grug Inc. task at hand

generics especially dangerous here, grug try limit generics to container classes for most part where most value add

temptation generics very large is trick! spirit demon complex love this one trick! beware!

always most value type system come: hit dot see what grug can do, never forget!

Expression Complexity

grug once like to minimize lines of code much as possible. write code like this:

```
if(contact && !contact.isActive() && (contact.inGroup(FAMILY) || contact.inGroup(FRIENDS))
  // ...
}
```

over time grug learn this hard debug, learn prefer write like so:

```
if(contact) {
  var contactIsInactive = !contact.isActive();
  var contactIsFamilyOrFriends = contact.inGroup(FAMILY) || contact.inGroup(FRIENDS);
  if(contactIsInactive && contactIsFamilyOrFriends) {
      // ...
  }
}
```

grug hear screams from young grugs at horror of many line of code and pointless variable and grug prepare defend self with club

club fight start with other developers attack and grug yell: "easier debug! see result of each expression more clearly and good name! easier understand conditional expression! EASIER DEBUG!"

definitely easier debug and once club fight end calm down and young grug think a bit, they realize grug right

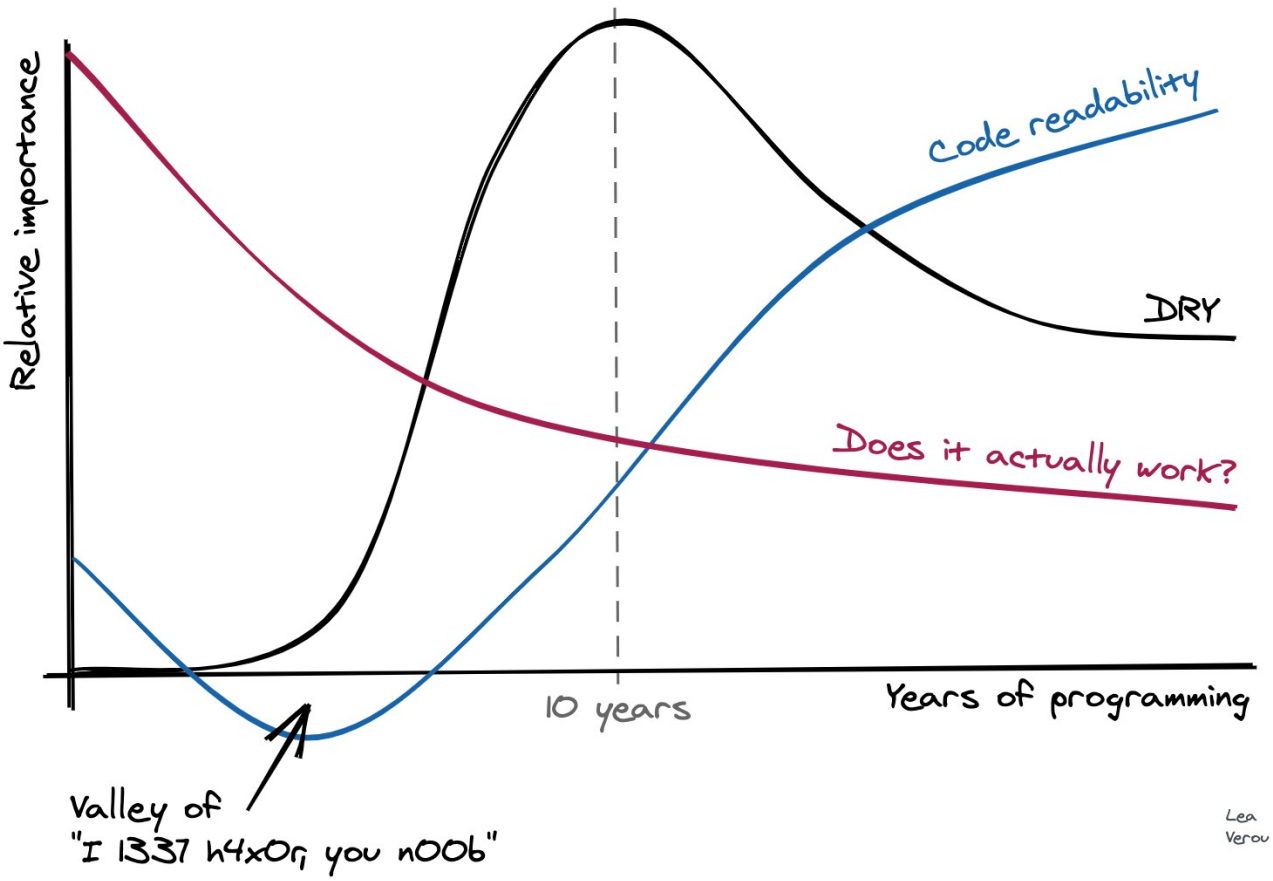
grug still catch grug writing code like first example and often regret, so grug not judge young grug

DRY

DRY mean Don't Repeat Self, powerful maxim over mind of most developers

grug respect DRY and good advice, however grug recommend balance in all things, as gruggest big brain aristotle recommend

grug note humourous graph by Lea Verou correspond with grug passion not repeat:



over time past ten years program grug not as concerned repeat code. so long as repeat code simple enough and obvious enough, and grug begin feel repeat/copy paste code with small variation is better than many callback/closures passed arguments or elaborate object model: too hard complex for too little benefit at times

hard balance here, repeat code always still make grug stare and say "mmm" often, but experience show repeat code sometimes often better than complex DRY solution

note well! grug encourage over literal developer not take does work line too serious, is joke

Separation of Concerns (SoC)

Separation of Concern (SoC) another powerful idea over many developer mind, idea to separate different aspects of system into distinct sections code

canonical example from web development: separation of style (css file), markup (html file) and logic (javascript file)

here grug much more sour faced than DRY and in fact write big brained essay on alternative design principle locality of behavior (LoB) against SoC

grug much prefer put code on the thing that do the thing. now when grug look at the thing grug know the thing what the thing do, alwasy good relief!

when separate of concern grug must often all over tarnation many file look understand what how button do, much confuse and time waste: bad!

Closures

grug like closures for right job and that job usually abstracting operation over collection of objects

grug warn closures like salt, type systems and generics: small amount go long way, but easy spoil things too much use give heart attack

javascript developers call very special complexity demon spirit in javascript "callback hell" because too much closure used by javascript libraries very sad but also javascript developer get what deserved let grug be frank

Logging

grug huge fan of logging and encourage lots of it, especially in cloud deployed. some non-grugs say logging expensive and not important. grug used think this way no more

funny story: grug learn idol rob pike working on logging at google and decide: "if rob pike working on logging, what grug do there?!?" so not pursue. turn out logging *very* important to google so of course best programmer work on it, grug!

don't be such grug brain, grug, much less shiney rock now!

oh well, grug end up at good company anyway and rob pike dress habit increasingly erratic, so all work out in end, but point stand: logging very important!

grug tips on logging are:

- log all major logical branches within code (if/for)
- if "request" span multiple machine in cloud infrastructure, include request ID in all so logs can be grouped
- if possible make log level dynamically controlled, so grug can turn on/off when need debug issue (many!)
- if possible make log level per user, so can debug specific user issue

last two points are especially handy club when fighting bugs in production systems very often

unfortunately log libraries often very complex (java, why you do?) but worth investing time in getting logging infrastructure "just right" pay off big later in grug experience

logging need taught more in schools, grug think

Concurrency

grug, like all sane developer, fear concurrency

as much as possible, grug try to rely on simple concurrency models like stateless web request handlers and simple remote job worker queues where jobs no interdepend and simple api

optimistic concurrency seem work well for web stuff

occasionally grug reach for thread local variable, usually when writing framework code

some language have good concurrent data structure, like java ConcurrentHashMap but still need careful grug work to get right

grug has never used erlang, hear good things, but language look wierd to grug sorry

Optimizing

ultra biggest of brain developer once say:

> premature optimization is the root of all evil

this everyone mostly know and grug in humble violent agreement with ultra biggest of big brain

grug recommend always to have concrete, real world perf profile showing specific perf issue before begin optimizing.

never know what actual issue might be, grug often surprise! very often!

beware only cpu focus: easy to see cpu and much big o notation thinking having been done in school, but often not root of all slowness, surprise to many including grug

hitting network equivalent of many, many millions cpu cycle and always to be minimized if possible, note well big brain microservice developer!

inexperienced big brain developer see nested loop and often say "O(n^2)? Not on my watch!"

complexity demon spirit smile

APIs

grug love good apis. good apis not make grug think too much

unfortunately, many apis very bad, make grug think quite a bit. this happen many reasons, here two:

- API creators think in terms of implementation or domain of API, rather than in terms of use of API
- API creators think too abstract and big brained

usually grug not care too deeply about detail of api: want write file or sort list or whatever, just want to call `write()` or `sort()` or whatever

but big brain api developers say:

"not so fast, grug! is that file *open for write*? did you define a *Comparator* for that sort?"

grug find self restraining hand reaching for club again

not care about that stuff right now, just want sort and write file mr big brain!

grug recognize that big brain api designer have point and that *sometime* these things matter, but often do not. big brain api developers better if design for simple cases with simple api, make complex cases possible with more complex api

grug call this "layering" apis: two or three different apis at different level complexity for various grug needs

also, if object oriented, put api on thing instead of elsewhere. java worst at this!

grug want filter list in java

"Did you convert it to a stream?"

fine, grug convert to stream

"OK, now you can filter."

OK, but now need return list! have stream!

"Well, did you collect your stream into a list?"

what?

"Define a Collector<? super T, A, R> to collect your stream into a list"

grug now swear on ancestor grave he club every single person in room, but count two instead and remain calm

put common thing like `filter()` on list and make return list, listen well big brain java api developer!

nobody care about "stream" or even hear of "stream" before, is not networking api, all java grugs use list mr big brain!

Parsing

grug love make programming language at drop of hat and say recursive descent most fun and beautiful way create parser

unfortunately many big brain school teach only parser generator tool. here grug usual love of tool is not: parser generator tool generate code of awful snakes nest: impossible understand, bottom up, what? hide recursive nature of grammar from grug and debug impossible, very bad according grug!

grug think this because while complexity demon bad for code base and understand, complexity demon very good for generation of much academic papers, sad but true

production parser almost always recursive descent, despite ignore by schools! grug furious when learn how simple parse is! parsing not big brain only magic: so can you!

grug very elated find big brain developer Bob Nystrom redeem the big brain tribe and write excellent book on recursive descent: Crafting Interpreters

book available online free, but grug highly recommend all interested grugs purchase book on general principle, provide much big brain advice and grug love book *very* much except visitor pattern (trap!)

The Visitor Pattern

bad

Front End Development

some non-grugs, when faced with web development say:

"I know, I'll split my front end and back end codebase up and use a hot new SPA library talking to a GraphQL JSON API back end over HTTP (which is funny because I'm not transferring hypertext)"

now you have two complexity demon spirit lairs

and, what is worse, front end complexity demon spirit even more powerful and have deep spiritual hold on entire front end industry as far as grug can tell

back end developers try keep things simple and can work ok, but front end developers make very complex very quickly and introduce lots of code, demon complex spirit

even when website just need put form into database or simple brochure site!

everyone do this now!

grug not sure why except maybe facebook and google say so, but that not seem very good reason to grug

grug not like big complex front end libraries everyone use

grug make htmx and hyperscript to avoid

keep complexity low, simple HTML, avoid lots javascript, the natural ether of spirit complexity demon

maybe they work for you, but no job post, sorry

react better for job and also some type application, but also you become alcolyte of complexity demon whether you like or no, sorry such is front end life

Fads

grug note lots of fads in development, especially front end development today

back end better more boring because all bad ideas have tried at this point maybe (still retry some!)

still trying all bad ideas in front end development so still much change and hard to know

grug recommend taking all revolutionary new approach with grain salt: big brains have working for long time on computers now, most ideas have tried at least once

grug not saying can't learn new tricks or no good new ideas, but also much of time wasted on recycled bad ideas, lots of spirit complexity demon power come from putting new idea willy nilly into code base

Fear Of Looking Dumb

note! very good if senior grug willing to say publicly: "hmmm, this too complex for grug"!

many developers Fear Of Looking Dumb (FOLD), grug also at one time FOLD, but grug learn get over: very important senior grug say "this too complicated and confuse to me"

this make it ok for junior grugs to admit too complex and not understand as well, often such case! FOLD major source of complexity demon power over developer, especially young grugs!

take FOLD power away, very good of senior grug!

note: important to make thinking face and look big brained when saying though. be prepare for big brain or, worse and much more common, *thinks* is big brain to make snide remark of grug

be strong! no FOLD!

club sometimes useful here, but more often sense of humor and especially last failed project by big brain very useful, so collect and be calm

Impostor Syndrome

grug note many such impostor feels in development

always grug one of two states: grug is ruler of all survey, wield code club like thor OR grug have no idea what doing

grug is mostly latter state most times, hide it pretty well though

now, grug make softwares of much work and moderate open source success , and yet grug himself often feel not any idea what doing! very often! grug still fear make mistake break everyone code and disappoint other grugs, imposter!

is maybe nature of programming for most grug to feel impostor and be ok with is best: nobody imposter if everybody imposter

any young grug read this far probably do fine in program career even if frustrations and worry is always to be there, sorry

Reads

grug like these:

- Worse is Better
- Worse is Better is Worse
- Is Worse Really Better?
- A Philosophy of Software Design

Conclusion

you say: complexity *very*, *very* bad