A Witch's Guide to Go: 3 Charms to Ehance your Software

inspired by question: "is what we do so straightforward that it cna be replicated by generative AI?"

speaker is builder of NLP apps, LLMs, and distributed systems

one
"if a module hides information we increase the funcitonaliy provided while reducing its interface. This makes the module deeper"

two
"define errors out of existence"

In Go we rely on errors for contorl flow, but this makes their interpretation tricky; distributed tracing is hard.

could we get compound errors to help tell a better story

three
"code is most obvious if it conforms to the conventions that readers will be expecting."

Enchantment Charms
- creativity - see one above
- hope - take a holistic view of distributed tracing with errors that communicate
- empathy - deliver go-flavored features to your users who use languages other than Go

---

The Blueprints to Building Your Own Badass Community

Go is becoming a global language - community is key

understnading goals is important to help with your motivation and planning

does a community already exist? Get involved rather than just starting a new one?

Are there meetups you can take inspiration from?

planning your event:
understand your limitations and manage your expectations

Gather interest from work, friends or othes (aka seeding) - others see at least some people there. (Like when a busker puts dollars in their guitar case)

securing a venue (see slide)

define comms channels ahead of time - what if it gets canceled.

Make sure you understand the Go communicty code of conduct

what if your first event was a disaster?
1. review and reflect
2.
3. remember to celebrate - you can still be sad and eat ice cream

what if your first event was good enough to do another?
1. same as before
2. streamline processes - you will notice repetition and boilerplate
3. Try not to give into ego

"Can a tech meetup have Code Smell?" - interesting article to look up

Introduced Priority Queue because they were getting too many potential attendees. They save 20%

for underrepresented in tech spots

what if you want out - after one or more events you no longer want to be the organizer
1.
2. it can be hard but know you can leave at any time. Make sure to get those who might want to take over.
3. Remember to celebrate

Cleaning Your GOOOP

How to berak oject oriented programming muscle memory and become better Gophers

Applying OOP patterns to Go is less than ideal

OOP
- grew out of work in MIT in the 50s/60s
- focuse don systemed based on objects that capture state ahd methods on those objects to modify state

Go is apracticaly language that doens't reward fancy
- structs are not classes

closer to C structs than Java or C++ classes

methods are just syntax sugar

constructors are reg functions and are 100% optional

no destructors

- embedding is not inheritance

focus on what things a type can do (has-a) not which kind of thing it is (is-a)

fields and methods on the mebedded type, but are promoted to the outer type

thinking in terms of based classes and derived classes can lead to GOOOP

- packages are the smalest unit of code

all code list in apcakage

visibility is enforced w/in and outside the package

importing the package bring sin the entire package

all consumer refs to symbols include the package name

examples of GOOOP

1. creating seaparate shared components to resolve co-dependency

When A depends on B and B depends on A

- usually you make a third ocmponent C and have the common stuff in there that both A and B use
- reduces cohesion an dcreate unnecessary coupling
- often leads to bad module/package names
- modifying multiple modules at once can be difficult

Signs of GOOOP
- package names like base, clients, interfaces tha tdoesn't represent functionality
- packages that only container interfaces and structs with not behavior

2. Declaring interfaces as exported provider abstractions

Interfaces in OOP
- abc
- enumerate the available operations on some type
- commonlu used as insulation layer between class and user

interfaces in Go
- define one or more methods that represent some set of behavior
- satisfied implicitly by any type having the hright methods
- insulation is "free" - consumer not bound to any implementation
- simplifies testing
- eases future development - can satisfy interfaces that don't even exist yet

Signs of GOOOP
- used the "good" name for the interface
- implementatoin has an IMpl (or similar) suffix
- interface redeclares every method
- there are no refs to the interface in the package that declared it

Architectural patterns

How to structure my project?

Clean Architecture - defined by "Uncle Bob" Martin in 2012
- clean boundaries
- combines other patterns with a focus
- 4 layers
        - entities
        - use cases
        - interfaces and adapters
        - Frameworks and drivers
-deps can only point inward

Model, View, *
- focus on building UIs
- common in web apps

Signs of GOOOP
- packages name dfor things that mirror the pattern layers
- types are repeated in each package
- each entity/model change means updating multiple packages
- manageable in isolation, but rough for cross-project integration

These have stuck around for a reason - they're good.

(see slide to see how Clean Architecture is done in a more idiomatic way)

there should never be more than one reason for a class to change

we should be able ot provide new behavior without needing ot rewrite existing code

interfaces should be as narrow a possible - idealy single-method

compose narrower interfaces to combine behaviors

duck typing means consumers aren't stuck

depend on abstractions, not ocncretions.

A great rule of thumb for Go is accept interfces, return structs

definte interfaces for input parameters

Everything you alwawy wanted to know abou ttype inference

type inference is the ability toe automaticlaly deduce the type of an expression at compile time....without explicit type annotations

it's already done in go from what's on the righthand side of the equals sign

it's good for removing repetition

with generics in Go, there can be many more types.

(don't be dogmatic - if it makes the code easier to read - use them)

Type inference can be seen as solving Type equations

In Go 1.21 type inference is much more powerful

Closing remarks Thursday: A fascinating talk about vulnerability analysis along the supply chain with govulncheck. It checks your deps (both the ones you listed and the ones they brought in). Then it tells you if you have to worry about a CVE on one of the deps (or deps of deps) based on whether you actually use the function or functionality from the dep where the vulnerability lies.

From Zero to Hero - Implementing a Game in 45 min

author wants more games to be builtin in Go instead of .net

cooperative idle game called "feed the gopher" - compete with friends to see who can feed the gopher the most. Automate a feeding empire to maximize growth

features:
- leaderboard
- player chat
- rate limiting

client:
single static page HTML
feed buttom
leaderboard chatUI
auto feeder upgrades

see slide for architecture

see slide for endpoints/code layout

leaderboard powered by a sorted set

rate limiter to enforce the rules of the game

Go Microservices for ML at the Edge

speaker wants to add data processing as a use case for Go

apply ML to cellular virtual straining. Want to do image anomaly detection.

watch the talk because there were LOTS of lessons learned from lots of complications - including needing to be able to install on Windows (including Win11)

Wed keynote - Go Changes
Rus Cox

Go has to change because it has to adapt.

Talk is about how change is decided for the language.

Changes they want

It's all based on the goal - "better software engineering at scale"

"software engineering is what happens to programming when you add time and other programmers"

the go goal is why it had testing built in from the beginning

gofmt - avoids spuroius diffs just because you disagree on formatting.

Unambiguous import paths - make for better software engineering when there's a package name collision

other examples including removing support for insecure systmes

Data-driven decisions

"consensus is more likely when the poeple look at the same problem with the same information"

where do we get the data?

talking ot users - go user survey, in-editor surveys, research interviews and user exp studies

they also do open-source code analysis. They look at the go modules that have been published.

however, these data come from very small number of folks. Sampling helps.

Go toolchain telemetry (opt-in)
---
telemetry will be a small amount of go usage, but it will give the Go devs different data.

It will be voluntary and opt-in

they want usage information and breakage informatoin

usage - eg who is using features they are considering removing?

breakage - eg the Mac build system bug

Go compiler doesn't stop at the first error. It tries to find them all so that you can fix them all at once before trying to compile again.

Conclusions

- Go needs to change over time
- go changes aim for better software engr
- data is critical to reaching consenus

- telemetry tiwll be an ipmortant data source

Lightning Talks (Wed)

Can ChatGPT do my job?

first pass result from ChatGPT was OK.

it was helpful to get a little bit started. But it requires a LOT of back and forth with the speaker.

can be a powerful helper , but cannot be trusted to write good code. Can reintroduce problems that you already asked it to remove

==========

Tis But a Scratch

Shrinking container images usein FROM scratch

they will be faster to ship and more secure

using alpine ended up with 298MB image

because Go is compiling to a binary, we don't need to bring an OS along. It can just run off the host kernel.

see slide to see how they use first alpine and then scratch to get 6.72MB image size

because no OS - malicous actor can't open a shell

you don't need to worry about keeping an OS up to date

if you need just a bit more, you might want to use distroless image instead of scratch - includes CA certs, tzdata, and a few more things

==========

State of the Art in Randomness

math/rand v2 will have multiple sources avialable

uniformity - do we have statistical quality;

unpredictability - used for security; chacha20 is the state of the art

efficiency - speed and size; many state of the art algorithms

==========

Cognitive Load and Go

reduced cognitive load can lead to less issues with bugs

shows how, wihout type hints, there's actually a high cognitive load in Python compared to Go

"understranding what types of information variables hold is key to being abl eto reason about making changes to code. If you don't understand what the variable is supposed to represent, thinking aobut the code will be hard"

60% of dev time is spent reading and understanding code

You eventually have to "pay the piper" on types

"If you've chosen the right data structure, the algorithms are self-evident. data structures, not algorithms are central to programming"

==========

A supply chain issue lurking in your API

issues:
- typosquatting
- phishing of maintainers
- the code you see on Github is not necessarily what you get
- weaponized dependencies

crypto/rsa package supply chain attack

nil error means things worked (from me: weird!!)

you can poison the package by changing the global variable in that package

see demo package the speaker has made

==========

Go Proverbs to Overcome Imposter Syndrome

go-proverbs.github.io

don't communicate by sharing memory, share memory by communicating - talk to others about what you're working on so you can celebrate together

clear is better than clever - you might not have all the fancy words to talk about how you feel, but you can still talk about it

reflection is never clear - hard to see the ways we've improved

errors are values - failure is necessary for growth. don't make perfection the goal

don't just check errors, handle them gracefully - give yourself time and space to learn from errors

design the architecture, name the components, document the detalis - ask youself how you might support a friend who minimizes their accomplishments

documentation is for users - track your accomplishments and praise

don't panic - builda good network of friends and fam

==========

Freeze - a meeting muter with a twist

bluetooth gradget in tinygo

---

structures logging

log/slog

structured logging - using key/value pairs in your logging

why make slog package when there are many 3rd party ones? Well, to unify things so they all work the same

Goals
- easy to use
- fast
- interoperable with other packages

see slide on how you do it - there is text handler and json handler

LogAttrs (is faster) you specify the type (so it doesn't need to do inference) - see slide

Levels
- are ints
- some have names (Debug, Info, Warn, Error)
- room for your own names
- info is 0

use groups to make json sub-objects

groups can also help with the issue of conflicting names

the Logger.With - allows you to have something added to everything in that log

---

Terminal UI Apps from the Ground Up with Bubble Tea

@andyhaskell@mastodon.social

Bubble Tea is based on the Elm Architecture

Model is the data of your app

View - what the data looks like to the user

Update - how we handle input

see slides to see how the view is built

Lip Gloss is like CSS for TUIs

The Future of JSON in Go

this has been a multi-year effort

JSON datatypes correlate to Go types (see slide)

encoding/json package used to Marshal and Unmarshal data into Go structs

it's the #5 most imported Go package - even over net/http, os, errors, and strconv

What's bad with encoding/json v1?

1. missing functionality

92 open requests on the GH repo

2. api deficiencies

these can't be fixed w/o making a breaking chance

options cannot be passed to Marshal or Unmarshal

no easy way to unmarshal from an io.Reader

compact, indent, and htmlescape have limitations

3. performance limitations

the methods perform poorly

mashalljson always allocates

they also force a second parse - On^2 runtime may be experienced

this (On^2) has caused issues w/ kubernetes in the real world

4. behavioral flaws

methods are inconsistently called depending on whether the underlying Go value is addressable

unmarshalling allows for invalid UTF-8 (violates JSOn standard)

unmarshallling allow for duplicate names in JSON object - which can be a security issue

uses case-insentive match to match struct names - a security vuln.

see slides for priv esc using duplicate names

draft design for v2 (not a formal proposal)
==========

(see slide)

their idea for jasontext package makes it smaller which is good for TinyGo

they have byte slices, io writers and readers ,and jsontext

there will be easier ability to set options

Looks like v2 will call v1 with options - to preserve compat?

omitempty - is more about the JSON representation of types

omitzero - omits a Go struct field that is the zero Go value or possesses an IsZero method that reports true

inline - JSOn equiv of Go struct embedding

Understanding Supply Chain Threats with static analysis

problems in open source security

600% increase in software supply chain attacks in 2022 vs 2021

in general Go has a more secure ecoystem than other languages
- has well-defined standard lib - you don't need to rely on 3rd party packages
- consistent package mgmt system -

- easy to read/write - harder to write bugs
- great tooling for code analysis -
- mem safety
- great community and security culture

for any package - does it do what it claims to do? Will it *continue* to be trustworthy? What about the dependencies in this package?

common dep issues - overly broad permissions, malicious updates, malicious packages

overly broad perms example - log4j

malicious package eg - see slides

malicious updates are extra bad because a package you audited and trust is no longer trustworthy. Also might be a transitive dep (not one of your direct deps)

Solution of checking all code and deps is not scalable.

capability analysis

A way of understanding what your deps do without having to audit it manually

capabilities are permissions, not vulnerabilities

So you want principle of least capability

We can to make package capbilities explicit and inform devs when those have changed

see slide for list of types of capabilities

std lib cals interact with the system in certain ways and those are the types of capabilities

eg network, files, system, cgo, unsafe

caveats
- only lets us detect a superset of package's capabilities
- mem safe violations could be used by bad actors to avoid being detected
- doesn't detect - logic errors, bugs, vulns

the capslock analyzer

CLI and lib for analyzing package capabilities

Goal: app permissions model, but fo rpackage imports

see slide on how to run it

output tells the caps and how many calls for each capability

You can also get the call paths to understand what's using which capability

Has JSON output for analysis and automatoin

next steps: integrate cap into package odcumentation sites (eg deps.dev) and OpenSSF Scorecards in Github

Also, could they automate capability restriction

Using Go to Track Satellites in the Sky

social pmconseco

citizen science

there are 8100 active sat orbiting earth

comms are usually unencrypted

Cna use the Go-Satellite lib

github.com/joshuaferrara/go-satellite

Port of SGP4

provides sat tls and other stuff

see Go code in the talk notes
