

DoneThat with Fyne, DDD, EventSourcing, and CQRS

The story of building a great personal management app



Simon Dassow

Consult With Simon

September 20, 2024



Topics

- But First
- Much Journey
- Very Development
- Giant Foundations
- Wow Application
- Some Code
- Such Conclusions



But First

A Few Personal Details

- Software Developer: 25+ years
- House Music: 20+ years
- Dad: 17+ years
- Calisthenics: 9+ years
- Vegan: 8+ years
- Certified Calisthenics Trainer: 2+ years
- New company: 1+ years



Some Professional Things

- Web, RAC Export Trading (1998)
- Web/admin, L&B GmbH (1998)
- Web/backend, Flipside (1999)
- Admin/backend, Vivendi Universal Games (2000)
- Web/backend/admin, Delphi Management Beratung (2003)
- Web/backend, RealNetworks (2006)
- Founder/dev/admin, PuzzWorks (2008)
- Lead/backend/dev, Booking (2010)
- Founder/coach/dev/admin, Consult With Simon (2023)



Much Journey

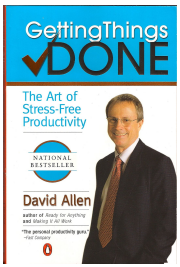
Becoming A Parent

- Active open source contributor
- Lots of ideas
- Kid was born
- Plans to start company
- Many responsibilities
- Looking for ways to manage



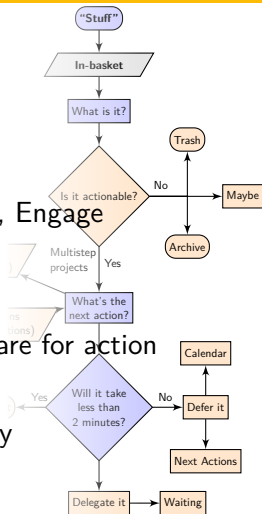
Searching For Books

- So. Many. Books.
- Looking for practical help
- One book in particular stood out
- Getting Things Done by David Allen
- Avid reading with kid on arm



Quick Explanation Of GTD

- TL;DR: wholesome list management
- Outsourcing mental load to trusted system
- Five steps: Capture, Clarify, Organize, Reflect, Engage
- Capture: collect everything
- Clarify: describe actions, sort into buckets
- Organize: add context and other details, prepare for action
- Reflect: get an overview, do weekly review
- Engage: prioritize by context, time, and energy



Researching The Concepts

- Does it make sense?
- Based on experience with knowledge workers
- Sound psychological concepts
- Study done by a university in Brussels agrees
- Key to success: reviewing/monitoring
- Is it easy to implement?



So It Begins

- Armed with knowledge (and kid)
- Pen, paper, action!
- Works well, doesn't scale
- Needs discipline for weekly review
- Software developer gets his hammer



Fast Forward Almost Twenty Years

- Many iterations
- Lots of experience
- Studied and learned a lot
- Development or evolution?
- Some milestones are worth mentioning



Looking Back At Milestones

- 2007: New parent, reading GTD
- 2008: Pen + paper version
- 2009: Command line version using PgSQL
- 2012: Server side web app
- 2016: CQRS/ES experiments
- 2017: Client side web app using CQRS/CS
- 2018: Server side switch to GoLang
- 2020: Fyne experiments
- 2022: Fyne app, clean CQRS code
- 2023: Fyne rewrite, reuse CQRS code



Very Development

Pen And Paper First

- Ultimate local first approach
- Still the baseline as reference
- Initially used as reference for visual design
- Cumbersome and tricky for complex scenarios
- Mental paradigm shift more effort than anticipated



Local First Is Natural

- Done this before
- Before mobile phones really took off
- Working with a computer anyways
- Very familiar with PostgreSQL
- Command line version using stored procedures



Everything Becomes More Available

- Internet more wide-spread, public/open wifi, mobile connections
- Server running at home
- Command line tools used via SSH
- Available \nrightarrow Reliable
- Server side web app using existing database



Meanwhile At Work

- CRUD models deadly for high performance
- Reads and writes using the same infrastructure often a bottleneck
- Updating materialized views at scale can be tricky
- Research inspired by optimizations
- CQRS and EventSourcing is a different world
- Conceptual solution to many problems at once
- How to apply it?
- Time to experiment



Client Side Web Application

- Started building a client side app
- CQRS and CommandSourcing in JavaScript
- JS ecosystem, NPM, Riot.JS
- Perl server receiving commands



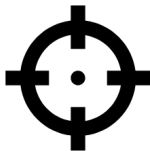
Server Side Language Switch

- Perl causing maintenance work
- Experimented with GoLang for a while
- Like statically typed languages
- Good start due to simplicity



Targeting Multiple Platforms

- Web applications fragile
- Web application support disappointing
- Clunky to package as mobile apps
- Mobile devices using Go?
- Nicer API than HTML/CSS?
- Experiments with Fyne



Diving Into Fyne

- First actual Fyne app
- Native UI code smooth using Fyne
- Fast realization of simple screens
- Implementation of CQRS/ES in Go
- Local first without synchronization
- Very same code on server



Getting Serious

- Clean rewrite of Fyne code
- Full reuse of CQRS/ES code
- Decision to offer application and service
- Intense focus and development
- Getting active in the community
- Contributions to Fyne



Giant Foundations

Keeping Things Practical

- Lots of theory behind every single thing
- Easy to get lost in details
- Experience, filtering, gut feeling, coherence
- Simplified high-level view
- Strong focus on practical value
- Leaving out details (see second point)



Fyne Is More Than Fine

- Targeting all relevant platforms with ease
- Clean and effective API
- Moves focus from design to functionality
- Encourages portable UI design
- Keeps winning UI toolkit comparisons
- Awesome community



CQRS

- Command Query Responsibility Segregation
- Simple concept with important implications
- Reads and writes using different methods/models
- Reads are free of side effects (immutable)
- Only writes change state
- Hint: Building command APIs is straight forward



EventSourcing

- Message based
- Sending commands to make changes
- Resulting events can be used to update state
- Single source of truth: event store
- Subscribe to event bus and build (many) read models
- Store read models optimized for reading
- Commands can be queued while operations continue
- Eventual consistency
- Business logic/process oriented
- Hint: Mock read models to decouple UI from business logic



Domain Driven Design

- Design approach to target problem space
- Division into bounded contexts (microservice)
- Specific models in each context/domain
- Terminology matching the business domain
- Specific high-level concepts and practices
- Domain is primary driver
- Aggregate design is important
- Repositories persist aggregates
- Recommends CQRS, EventSourcing, model driven design
- Hint: Event storming is simple and effective



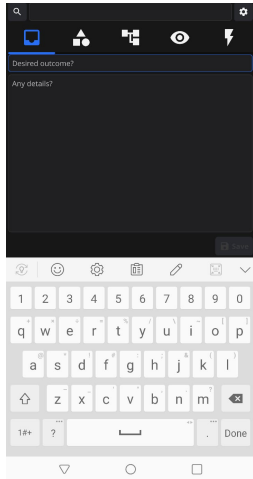
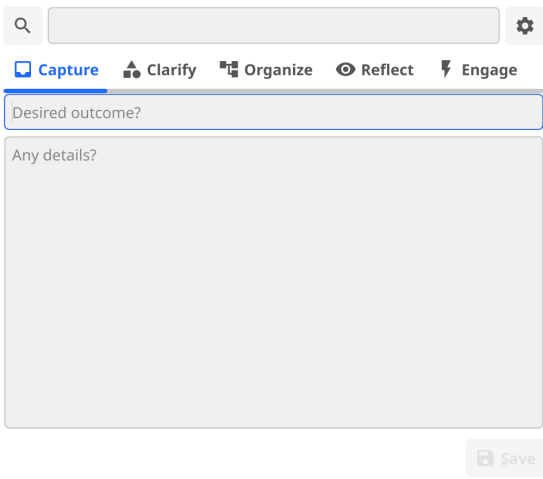
Wow Application

Combining Everything

- Very efficient UI design
- Fun and lean codebase
- 26791 lines of code
- 117 files of Go code for the Fyne/GUI app
- Pictures or it didn't happen...



Step 1: Capture



Step 2: Clarify

⚙️

📄 Capture
📊 Clarify
🗂️ Organize
👁️ Reflect
⚡ Engage

Integrated golang based CSS/JS minifier on consultwithsimon.tech

Found a project that looks promising: <https://github.com/tdewolff/minify>

Unactionable?

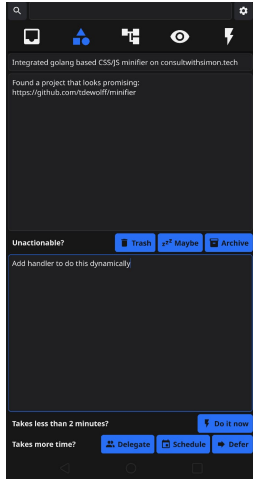
🗑️ Trash
z^z Maybe
📁 Archive

Very next physical action step?

Takes less than 2 minutes?
⚡ Do it now

Takes more?

👤 Delegate
📅 Schedule
➡️ Defer

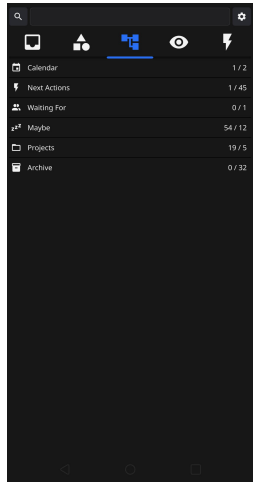


Step 3: Organize

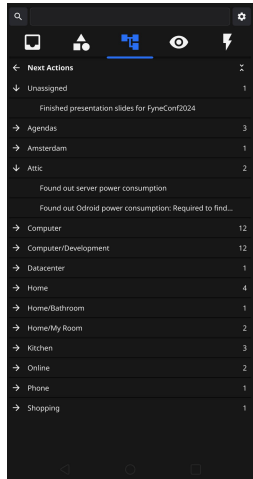
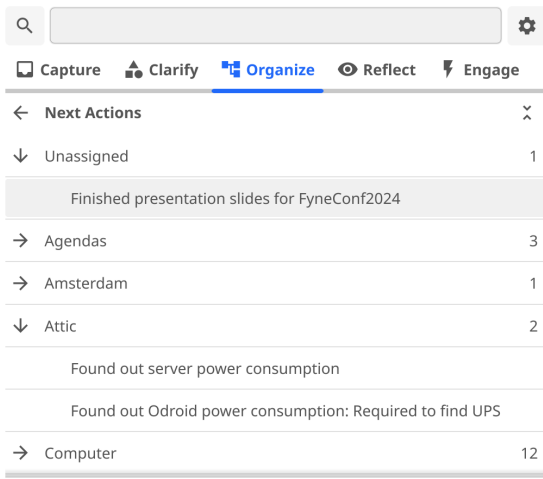
⚙️

📅 Capture
 🏠 Clarify
 📁 Organize
 👁️ Reflect
 ⚡ Engage

📅 Calendar	2 / 1
⚡ Next Actions	2 / 44
👤 Waiting For	0 / 1
zzz Maybe	55 / 12
📁 Projects	19 / 5
🗳️ Archive	0 / 32



Step 3: Organize: Next Actions



Step 3: Organize: Next Action: Detail

⚙️

📄 Capture
🗨️ Clarify
🗂️ Organize
👁️ Reflect
⚡ Engage

← Next Actions

Finished presentation slides for FyneConf2024

Got 25 minutes to cover, and a couple of specifics from the proposal submitted.

Have to work out structure of the talk, order of topics for smooth logical

Create slides, use logo, use images, ...

Context

Project

Move to ...

Duration (Select one)

Energy (Select one)

📁 Save

🔍
⚙️

📄
🗨️
🗂️
👁️
⚡

← Next Actions

Finished presentation slides for FyneConf2024

Got 25 minutes to cover, and a couple of specifics from the proposal submitted.

Have to work out structure of the talk, order of topics for smooth logical transitions.

List of things to check for on every slide:

- what's useful/interesting
- what would the audience miss if the didn't see it

Start with the main topics as outlined in description of talk to cover all bases, and from there fill in more.

Create slides, use logo, use images, ...

Context

Project

Duration (Select one)

Energy (Select one)

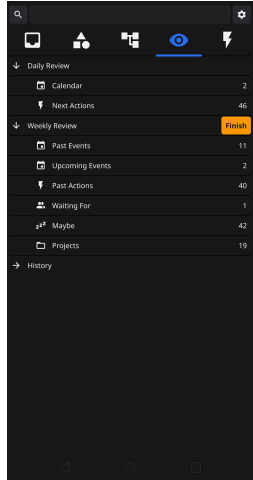
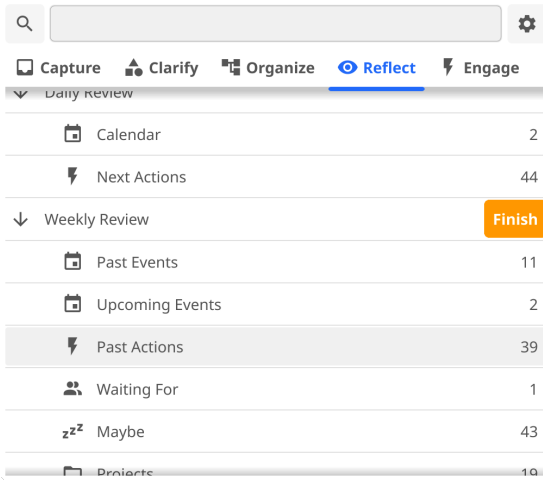
Move to ...

Duration (Select one)

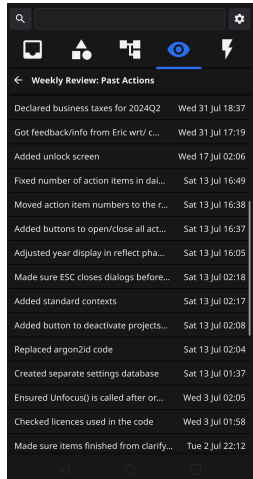
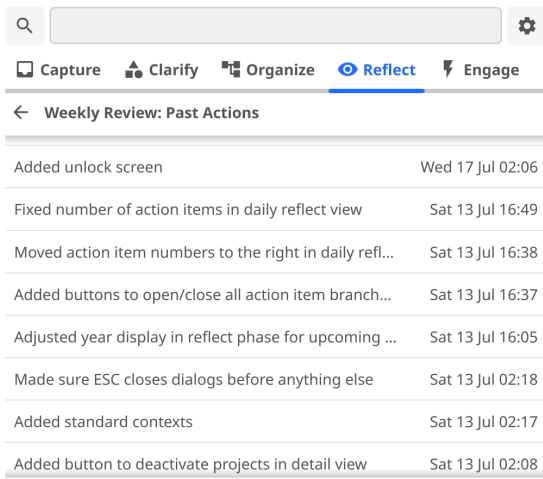
Energy (Select one)

📁 Save

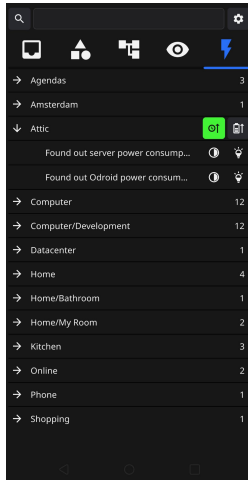
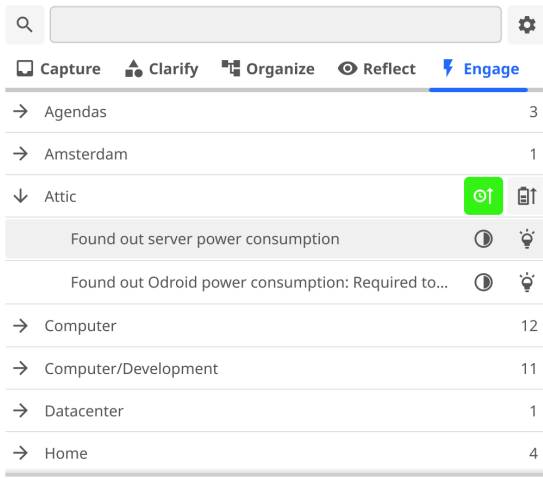
Step 4: Reflect



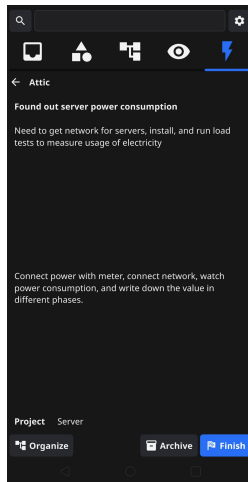
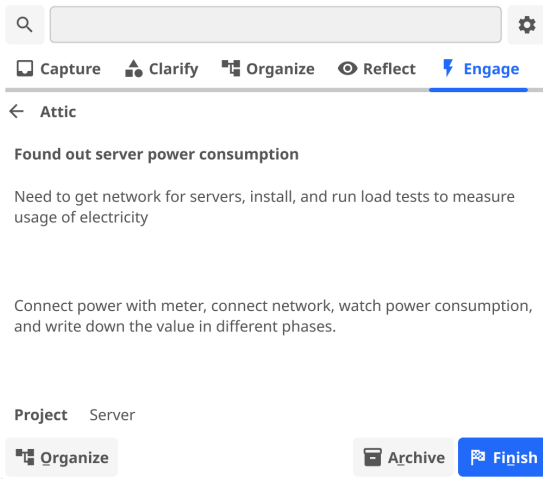
Step 4: Reflect: Past Actions



Step 5: Engage



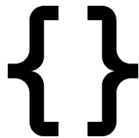
Step 5: Engage: Detail



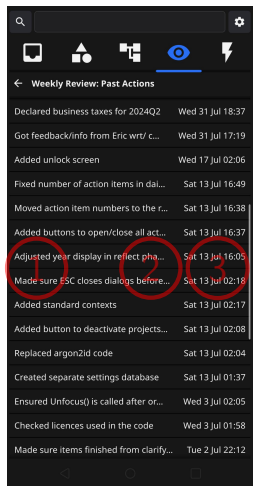
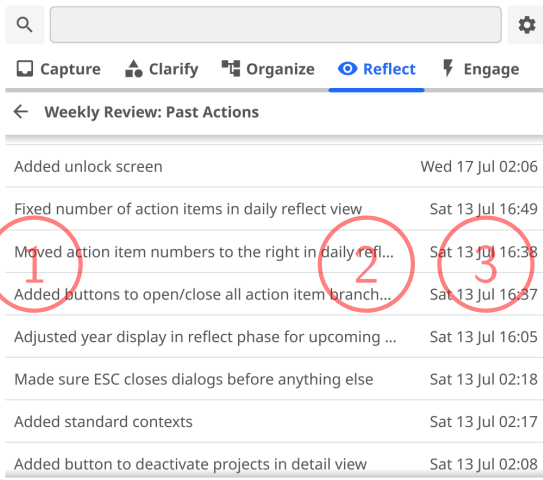
Some Code

Actual Code Example

- Using this in a lot of places
- Same code in official documentation
- List item with some left aligned text
- And a right aligned something
- Truncating the text with ellipsis



What It Looks Like



List Item Data Structure

- Using this very construct

```
// Basic Fyne widget  
type ListItem struct {  
    widget.BaseWidget  
    Title *widget.Label  
    Value *widget.Label  
}
```

New List Item

- Without truncation the `MinSize()` would grow

```
// Create new item
func NewListItem(title, value string) *ListItem {
    item := &ListItem{
        Title: widget.NewLabel(title),
        Value: widget.NewLabel(value),
    }

    // Configure label to truncate
    item.Title.Truncation = fyne.TextTruncateEllipsis

    // Important
    item.ExtendBaseWidget(item)

    return item
}
```

Simple List Item Renderer

- Border layout is super useful
- SimpleRenderer is simple

```
// Setup of layout and renderer
func (item *ListItem) CreateRenderer() fyne.WidgetRenderer {
    // The resulting widget wraps a container
    c := container.NewBorder(nil, nil, nil, item.Value, item.Title)

    // Writing a custom renderer would mean a bunch more code
    return widget.NewSimpleRenderer(c)
}
```

Another Example With Code

- Dark mode on a web site/app
- Good luck
- Different story with Fyne
- Custom themes
- Dark mode defaults available
- Scaling very easy



Custom Theme Wrapper

- Using as much Fyne code as possible
- Only adjusting two aspects
- Two custom methods

```
// Simple theme wrapper  
type customTheme struct {  
    fyne.Theme  
    variant fyne.ThemeVariant  
    scale   float32  
}
```

Custom Theme Methods

■ Minimal code

```
// Use custom theme variant to get colors
func (t *customTheme) Color(name fyne.ThemeColorName, _ fyne.ThemeVariant)
    color.Color {
    return t.Theme.Color(name, t.variant)
}

// Scale original theme size up or down
func (t *customTheme) Size(name fyne.ThemeSizeName) float32 {
    return t.Theme.Size(name) * t.scale
}
```

Custom Theme Helper

- Choice between light and dark theme
- Select from list of scale factors

```
// Internal helper to get custom theme
func getCustomTheme(name string, scale float32) fyne.Theme {
    def := theme.DefaultTheme()
    if name == "dark" {
        return &customTheme{def, theme.VariantDark(), scale}
    } else if name == "light" {
        return &customTheme{def, theme.VariantLight(), scale}
    }
    return def
}
```

Custom Theme Usage

- Using Preferences API
- Provide fallback values

```
// Using the preferences API for multiple things  
prefs := a.App.Preferences()  
  
// Set possibly user selected theme  
themeName := prefs.StringWithFallback("themeName", "default")  
themeScale := float32(prefs.FloatWithFallback("themeScale", 1.0))  
a.App.Settings().SetTheme(getCustomTheme(themeName, themeScale))
```

Such Conclusions

DoneThat In The End

- Putting it all together
- 15+ years of experience
- Battle tested best practices
- It actually does what it should
- Hint: always keep using the data



Closing Thoughts

- Fyne looks and works great
- DDD helps a lot with focus and scope
- CQRS maintains separation of concerns
- EventSourcing provides version control of data
- DoneThat benefits from the paradigm shift



Time For Questions

- Ask now
- Or later
- Community chat



Thanks

- Andrew for the constant engagement
- FyneLabs for organizing and sponsoring
- Community members for contributing
- Audience and you for listening

