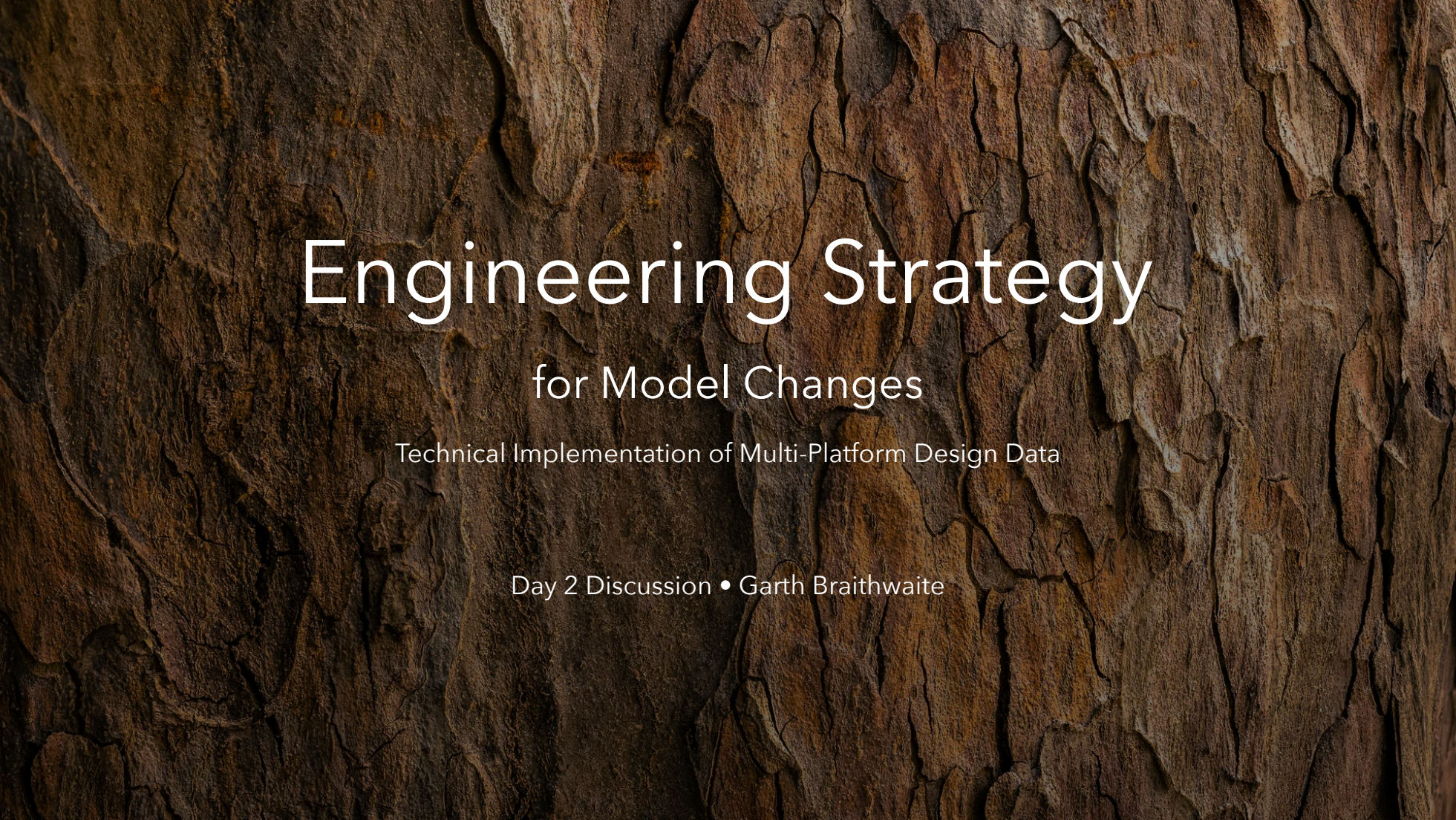


Engineering Strategy

for Model Changes

Technical Implementation of Multi-Platform Design Data

Day 2 Discussion • Garth Braithwaite

Engineering Strategy Overview

Technical Components

- **Design Data** structure and schemas
- **Component Options API** for states and variants
- **Component Anatomy** definitions
- **Design Tokens** hierarchy (Global → Base → Alias)
- **Component Token Relationships** (CTR)
- **Mappings & Parsers** for transformation

Multi-Platform Customization

- **Platform-specific implementation** formats
- **Business-specific flavoring** (future)
- **Product specialization** layers
- **Parser transforms** for data conversion
- **Filtered data** for platform needs
- **Value overrides** and mappings

From Anonymous Tokens to CTR Objects

Current: String-Based Problem

```
# Hard to validate, prone to errors
slider-handle-border-width-down-small: "7px"
button-minimum-height-medium: "40px"
```

Issues:

- No schema validation
- Naming inconsistency
- Manual spec maintenance
- Implementation guesswork

CTR Solution: Structured Objects

```
id: 683fb538-290c-423f-990b-d7134e485f51
$schema: adobe/spectrum/schema/0.0.0/token-types/dimensior
component: slider
part: handle
property: border-width
value: 7px
options:
  state: down
  size: small
  platform: mobile
```

Benefits:

- Schema validation
- Automatic generation
- Multi-platform support
- Clear relationships

Data Architecture: From Scattered to Structured

Current: Scattered Data

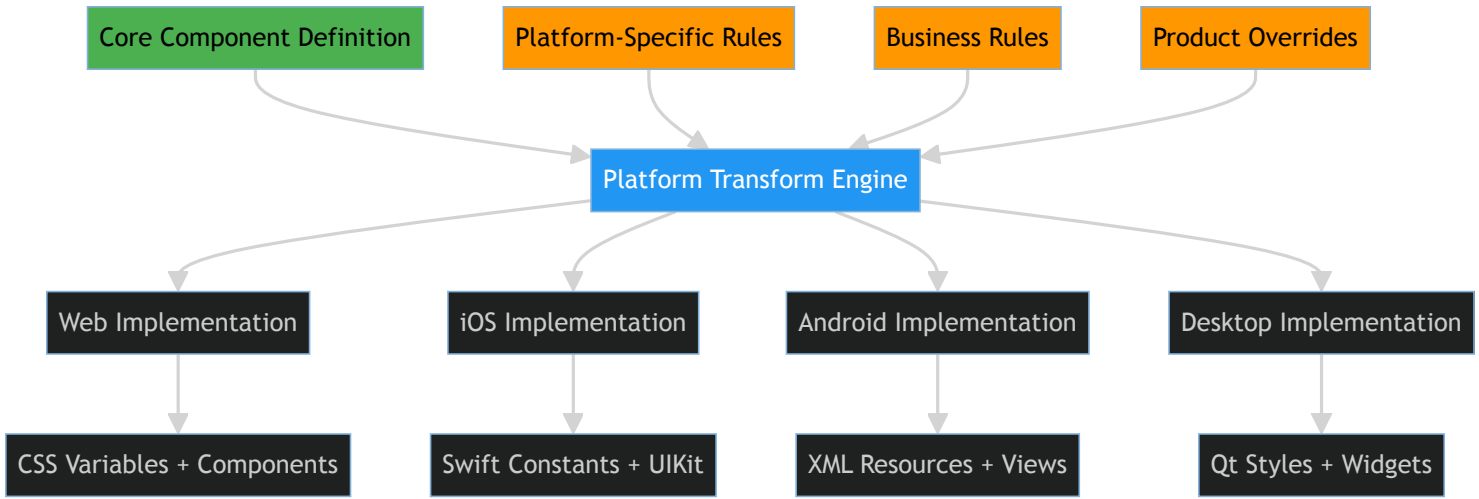
- **Design Data**: Multiple repos, formats
- **Component Specs**: Manual authoring
- **Figma Assets**: Disconnected from code
- **Documentation**: Multiple sources
- **Partnership Info**: Slack, meetings, heads

Proposed: Unified Schema

```yaml
# Unified Component Definition
component:
  name: "button"
  anatomy: ComponentAnatomy
  options: OptionsAPI
  tokens: ComponentTokenRelations
  platforms: PlatformMappings
  documentation: DocumentationRefs
  partnership: CollaborationData
```

Key Insight: Structured data enables automation, validation, and multi-platform generation

Multi-Platform Customization Strategy



Component Options API

Current Challenges

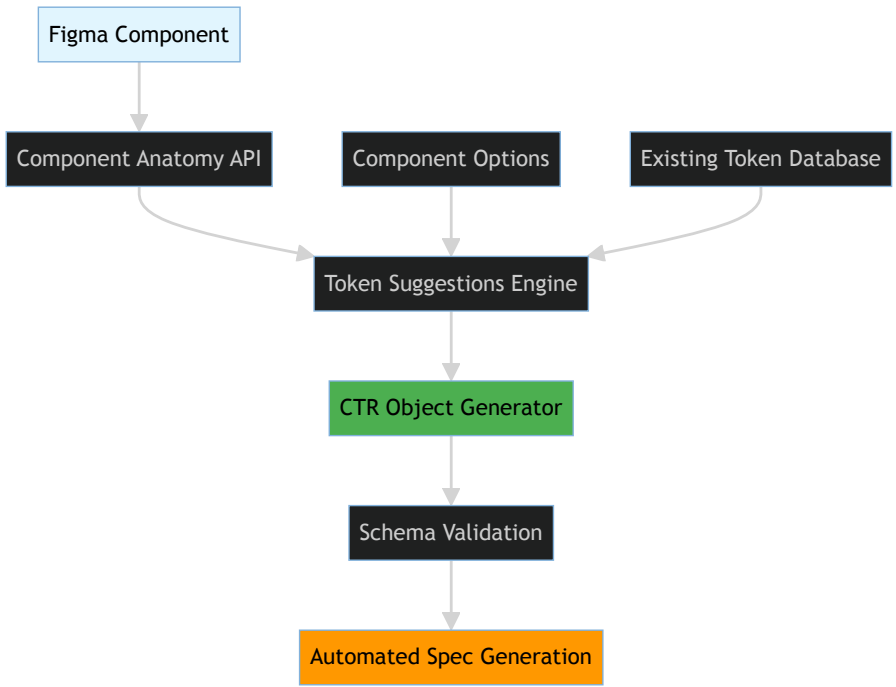
- **State variations** manually maintained
- **Different platforms** implement states differently
- **Inconsistent naming** across implementations
- **Missing states** on some platforms
- **Complex spec files** hard to maintain

Proposed API Structure

```
interface ComponentOptions {
  states: {
    default: StateDefinition
    hover: StateDefinition
    active: StateDefinition
    focus: StateDefinition
    disabled: StateDefinition
  }
  variants: {
    primary: VariantDefinition
    secondary: VariantDefinition
    tertiary: VariantDefinition
  }
  dimensions: DimensionDefinitions
}
```

Technical Architecture: Designer Workflow Integration

Figma Integration Architecture



Designer Experience Flow

1. **Designer selects component** in Figma
2. **System queries Component Anatomy** for valid parts
3. **Suggests existing tokens** from CTR database
4. **Guides naming** if new token needed
5. **Validates against schema** in real-time
6. **Generates CTR object** automatically
7. **Updates specs** across all platforms
8. **Creates changelog entry** automatically

Result: No manual spec maintenance

Addressing Real Designer & Engineering Pain Points

Current S2 Spec Authoring Issues

- **Manual "Token definition" artboards** in each component page
- **No changelog tracking** for spec updates
- **Token naming confusion** - hard to know how to name new tokens
- **Difficult to find existing tokens** to reuse
- **Complex semantic alias decisions** without guidance
- **Third-party Tokens Studio dependency** creates risk
- **Manual sync meetings** required for implementation

Technical Solutions via CTR Objects

- **Automated spec generation** from structured CTR data
- **Built-in change tracking** via schema versioning
- **Guided token authoring** with component anatomy
- **Automatic token discovery** and reuse suggestions
- **Schema-driven semantic aliases** with validation
- **Native authoring tools** reduce external dependencies
- **API-driven sync** eliminates manual handoffs

> **Goal**: Solve Allison's team workflow while improving engineering reliability

Parser & Transform Strategy

Input Parsers

- **Figma API** data extraction
- **Design token** file parsing
- **Component spec** analysis
- **Documentation** scraping
- **Schema validation**

Transform Engine

- **Platform mapping** rules
- **Value conversion** (px → pt → dp)
- **Naming convention** transformation
- **State generation** from base values
- **Validation & linting**

Output Generators

- **CSS custom properties**
- **iOS Swift constants**
- **Android XML resources**
- **Desktop platform formats**
- **Documentation** generation

Architecture: Pluggable parsers and generators enable platform extension

Implementation Phases

Phase 1: Core Infrastructure

- **Unified schema** definition
- **Transform engine** foundation
- **Basic parsers** (Figma, tokens)
- **One platform** implementation (Web)
- **Governance tooling**

Phase 2: Platform Expansion

- **iOS implementation**
- **Android implementation**
- **Desktop platform** support
- **Advanced transforms**
- **Cross-platform validation**

Phase 3: Advanced Features

- **AI-assisted authoring**
- **Automated state generation**
- **Performance optimization**
- **Visual diff tooling**
- **Advanced governance**

Technical Discussion Topics

Architecture Decisions

- **Schema format**: JSON Schema, TypeScript, or custom?
- **Transform engine**: Build vs. adapt existing tools?
- **Storage**: Git-based vs. database vs. hybrid?
- **API design**: REST vs. GraphQL vs. custom protocol?
- **Validation**: Compile-time vs. runtime vs. both?

Platform Integration

- **Web**: How to integrate with existing Spectrum CSS?
- **iOS**: Swift package vs. Xcode integration?
- **Android**: Gradle plugin vs. build integration?
- **Desktop**: Qt resource system integration?
- **Tooling**: IDE plugins and developer experience?

Risk Mitigation

Technical Risks

- **Complex migration** from current system
- **Performance impact** of transform engine
- **Platform compatibility** issues
- **Tool integration** challenges
- **Scale limitations** as system grows

Mitigation Strategies

- **Gradual migration** with parallel systems
- **Performance benchmarking** and optimization
- **Platform pilot programs** before full rollout
- **Extensive testing** and validation
- **Scalable architecture** from day one

Success Metrics for Engineering

Development Velocity

- **Faster component implementation** across platforms
- **Reduced bugs** from inconsistent token usage
- **Automated validation** catches errors early
- **Easier platform onboarding** for new implementations

System Reliability

- **Consistent behavior** across all platforms
- **Automated testing** of design-code alignment
- **Version control** for all design decisions
- **Clear rollback** strategies for changes

Ultimate goal: Engineering teams focus on user experience, not token management

Discussion Time

Let’s dive into the technical details

Ready to tackle the engineering challenges...