WEBFORLESS · SINCE 2003

Software Discovery Workbook

Custom software is valuable when generic tools force important workflows into awkward compromises. The goal is not to create more technology. It is to understand the operation, remove friction, connect reliable systems and create software the organization can own and improve.

Use this guide to:	Clarify priorities, assign ownership, identify risk and verify implementation.
Website:	webforless.ca

Project / website: _____

Owner: _____ Review date: _____

1. Start with the operation, not the feature list

A request such as “build a dashboard” is not yet a requirement. Discovery examines users, decisions, data, exceptions, handoffs and risks. The team should understand the current workflow and the desired outcome before choosing screens or technology.

- ■ Map the current process and its pain points.
- ■ Identify users, permissions and critical decisions.
- ■ Separate required outcomes from preferred interface ideas.

Notes / evidence:

2. Define a useful first release

A minimum viable product should be the smallest coherent system that creates value and can be safely operated. It should not be a random collection of half-finished features. Prioritization considers user value, operational dependency, technical risk and learning.

- ■ Choose one complete workflow.
- ■ Include security, auditability and support needs.
- ■ Defer features without blocking future architecture.

Notes / evidence:

3. Architecture should fit the problem

Architecture balances simplicity, reliability, cost, scale, integration and team capability. A modular monolith can be better than premature microservices. A managed platform can be better than owning every server. The right choice is the simplest one that meets real constraints.

- ■ Document boundaries and data ownership.
- ■ Prefer proven components for commodity needs.
- ■ Make trade-offs visible to business stakeholders.

Notes / evidence:

4. APIs connect systems and responsibilities

APIs are contracts between systems. Good contracts use clear resources, validation, authentication, versioning, errors and documentation. Integrations also require retry behaviour, rate limits, monitoring and plans for external outages.

- ■ Define who owns each data field.
- ■ Design predictable errors and idempotent operations.
- ■ Protect secrets and validate all external input.

Notes / evidence:

5. Security and privacy belong in the design

Security is not a final penetration test. Threat modelling, least privilege, encryption, secure defaults, logging, dependency management and recovery planning should influence architecture and implementation from the beginning.

- Collect only data the workflow requires.
- Separate roles and sensitive operations.
- Plan incident response and tested backups.

Notes / evidence:

6. Quality comes from layered testing

Useful testing combines automated unit and integration checks with workflow, accessibility, security, performance and user acceptance testing. Production monitoring then verifies behaviour in the real environment.

- Test business rules and exception paths.
- Use realistic data without exposing personal information.
- Make release criteria explicit.

Notes / evidence:

7. Ownership continues after launch

Custom software becomes an operational asset. It needs documentation, support, observability, dependency updates, user feedback and a roadmap. Contracts should make code, data, credentials and deployment responsibilities clear.

- ■ Keep source code and deployment access under client control.
- ■ Maintain architecture and runbook documentation.
- ■ Budget for improvement, not only emergency repair.

Notes / evidence:

Launch and follow-up review

Item	Owner	Status / date
Critical journey tested		
Access and ownership documented		
Backups or rollback verified		
Analytics and monitoring confirmed		
Outstanding risks recorded		
30-day review scheduled		

Good websites and software are never finished. They are monitored, maintained and improved as evidence and needs change.