How we approach product development at Nebraska Global

Doug Durham

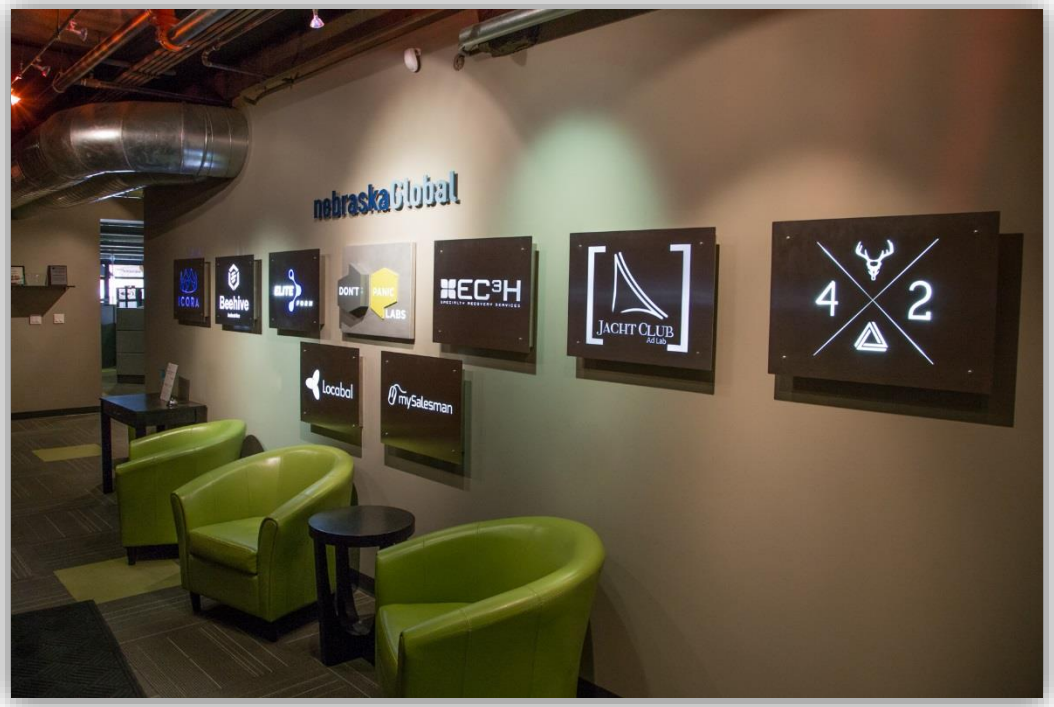
Chad Michel

nebraskaGlobal
/* Develop here */

nebraskaGlobal

/* Develop here */

About Nebraska Global



- Formed in April, 2010
- Located in Lincoln, NE
- Founded by people with decades of experience in commercial software product development
- Unique model…
  - Establish an investment fund and combine these dollars with a product development team to build software products and companies
  - Develop young software engineers / entrepreneurs
- Core development team has been together since early 2000's

NG product development challenges

- We are building startups
  - Radically different problem domains
- Product development team is an “elastic resource”
  - Move folks around
  - Integration of new and inexperienced team members
- Development of software designers/architects
- Development of young engineers

Lessons learned from prior lives

- Big jump from requirements to code
  - No software architecture / design
- Resulting code tended to be…
  - Inconsistent/incoherent
  - Overly complex and “clever” code
  - Difficult to maintain
  - Poorly documented/commented
- Lack of ongoing integration meant end of release cycles required lengthy “stabilization” phases
- Releases were quickly followed by a string of hot fixes
  - Painful “first birth”

Lessons learned from prior lives

- Within a few years you start to hear and see things like…
  - “Jeff is the one who designed that system and only he really knows how it works”
  - “If we are going to change this much we should just start over and re-write the system but do it right this time”
  - Every new feature requires an individual spend more and more time trying to understand the broad system to determine potential for unintended behavior
  - More and more time (and anxiety) associated with each release

Core Values

Proven Patterns and Designs

Layered Approach to Quality

Always Have a Plan

Adaptive Processes

Expect Change

Individual Accountability

"Make everything as simple as possible, but not simpler." -- Albert Einstein

How we approach product development

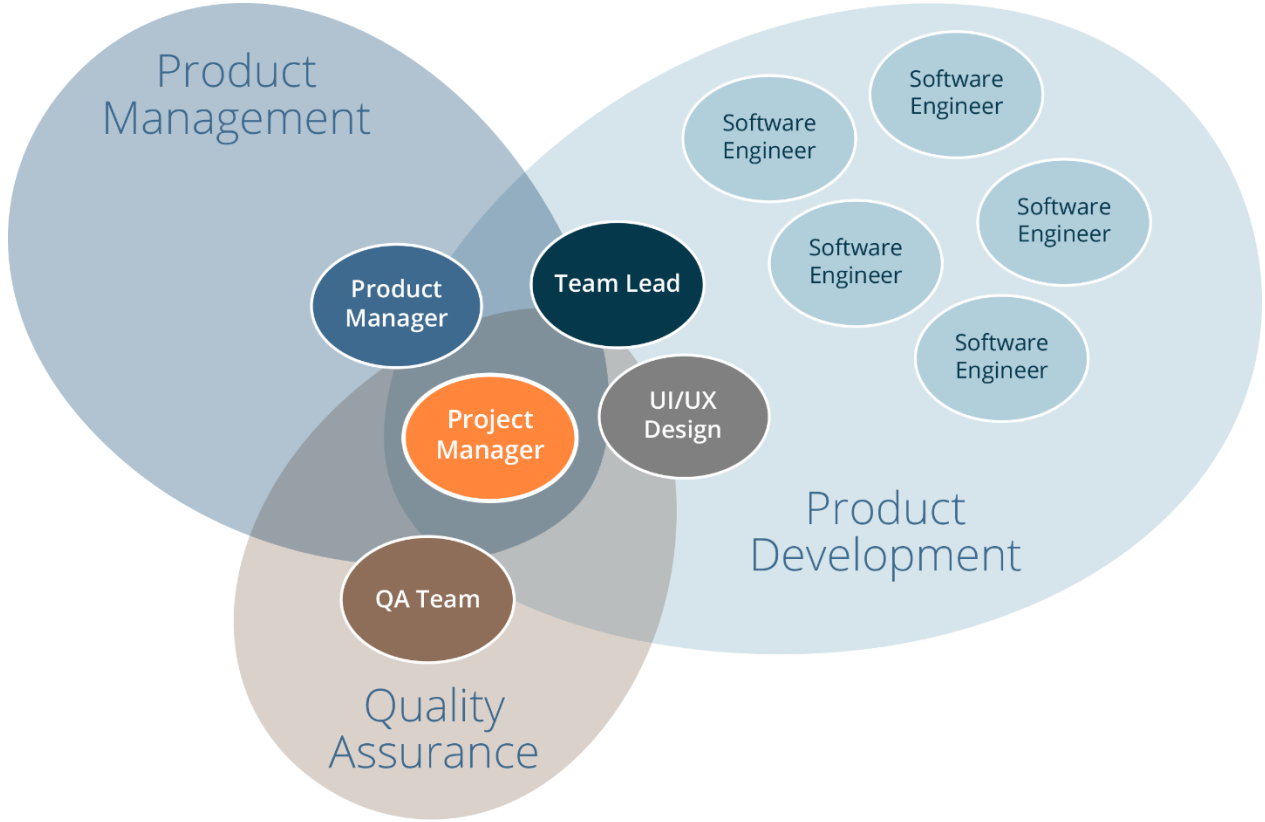
- We needed a “disruptive” vs “evolutionary” change
  - How we are organized
  - How we plan
  - How we design software
  - How we take a layered approach to quality
  - Additional factors improving developer productivity

Our roles

- Project Managers
  - aka Scrum Master
  - Process managers
  - Our “force multiplier” – make the development process far more efficient
- Lead Engineers
  - Responsible for ensuring the integrity of the design
  - Have the “big picture” in their head
- Architects
  - Work with lead engineers to develop the service architecture of the application
  - Collaborate across projects and products
- Software Engineers
  - Responsible for the design, build, test and quality of features
- UI / UX
  - Responsible for the analysis, design, and build of user interfaces
- Quality Assurance
  - Focus is more on customer experience and regression testing

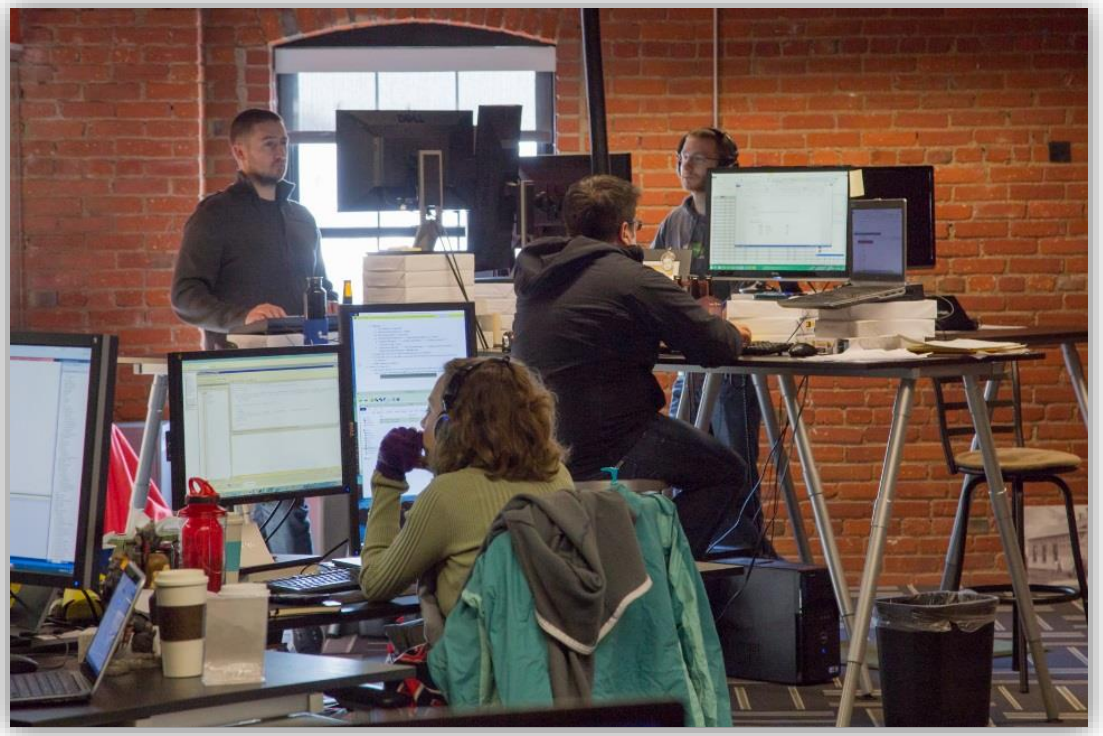
Typical team dynamics

- Teams: 4-6 engineers
- Project manager is the “hub” for coordination
- Team lead / architect works closely with product and project management
- Communication flows freely between product management and all team members
- QA operates at a product level
- NOTE: Very flat org structure. Accountability is to the team vs a manager



Office layout

- Observation: Only a small amount of “friction” is required before impacting collaboration and communication
- Our solution…
  - Open spaces
  - Adjustable desks
  - Easy access to breakout rooms
  - Movement to mobile computing
  - Headphones ☺
- Does have some drawbacks



How we plan

- Goals
  - Create a prioritized backlog of stories
    - Each story represents <= 1 week of effort
  - Create a process that allows for predictable throughput
  - Treat requirements analysis/story development as a design exercise
  - Incorporate risk reduction

Story development

- Collaboration with product manager/owner to define product/release requirements
- Develop UI/UX wireframes as necessary to help with requirement insight and effort estimation
- Continue to decompose stories until development team feels effort is <= 1 week
- Fixed estimate “buckets”
  - 1 point = 1 week
  - 0.5 = 2.5 days
  - 0.2 = 1 day
  - 0 <= 1 hour

Multi-Sprint Release Planning

- Stories are assigned to sprints
  - Available resources dictate total points available per sprint (i.e. velocity)
  - Story dependencies, priorities and risks influence where stories appear in the schedule
- Resulting sprint plan provides…
  - Estimate of likely release date (provides visibility to folks outside product development)
  - Visibility into rationale for sequence and timing
- Important: Sprint plan is just a plan, not the law
  - Every new sprint presents opportunity for adjustment and change in priorities

Area	Story	Description	Priority	ROM	Assigned to
	Sprint 0	**4/24 - 4/30**			
	Develop MVP architecture design	Controllers, areas and views		0.5	beebs, Hari, Chad, Norm
	Database schema		High	0.5	beebs, Hari, Chad, Norm
Tablet: UI setup	Get the basic tablet UI setup		High	0.5	Hari
Tablet: Login	Salesman need a unique login to sign into the tablet to create estimates	**See comp on page 20** -Reject login if they try to use it to log into admin	High	0.2	Hari
Tablet: Create estimate	Implement Create an estimate from scratch	**See comp on page 22** -From scratch takes an address and zip code. When [Begin] hit, they are taken to the fences & gates tab with the map centered over the address entered	High	0.2	Hari
	Sprint 1	**5/1-5/7**			
	Chain link rules engine		High	1	beebs
Super Admin: Businesses	Need ability to enable/disable budget vs estimator	See page 1: Businesses comp	High	0.1	Hari

“Maintenance” development

- Move toward a “kanban“ model
  - Stories are stack ranked and worked in priority order
  - Releases typically done at end of each sprint
- We often ebb and flow back and forth between multi-sprint releases and kanban releases.
  - Sometimes at the same time

Sprint planning

- IMPORTANT: This is treated as a design process!!!
- Stories are broken down into tasks
  - Tasks are the steps to implement
  - White papers used for more fluid or complex stories
    - Observation: Writing things down increases insights into requirements and design
- Tasks are estimated
  - Small (< 4 hrs)
  - Medium (4-12 hours)
  - Large (> 12 hrs) (further decomposition?)
- Tasks estimates are compared to story estimates
  - Adjustments made as necessary
- Sprints are 1 week (typically)

How we design software

"Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live."

How we design software

Software engineering is…

“the multi-person construction of multi-version programs.”-- David Parnas

Consumers use products developed by teams of engineers and these products are continuously changing and evolving

- Our solution
  - Formal software design role/process
  - Leverage existing design principles
  - Consistency of design pattern
  - Consistency of design conventions
  - “Global” design criteria

Leveraging existing principles

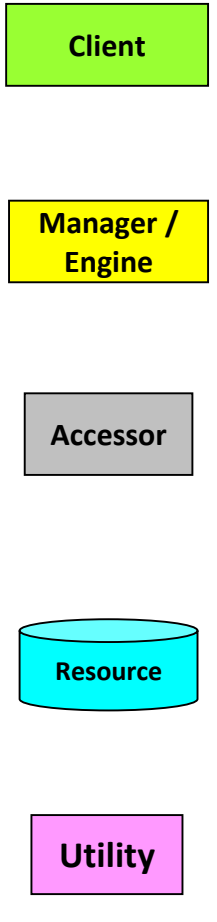
- Information hiding
- Designing for change
- Separation of concerns
- Maximize functional cohesiveness
- Minimize coupling
- Favor closed vs open architecture

Consistent software design process

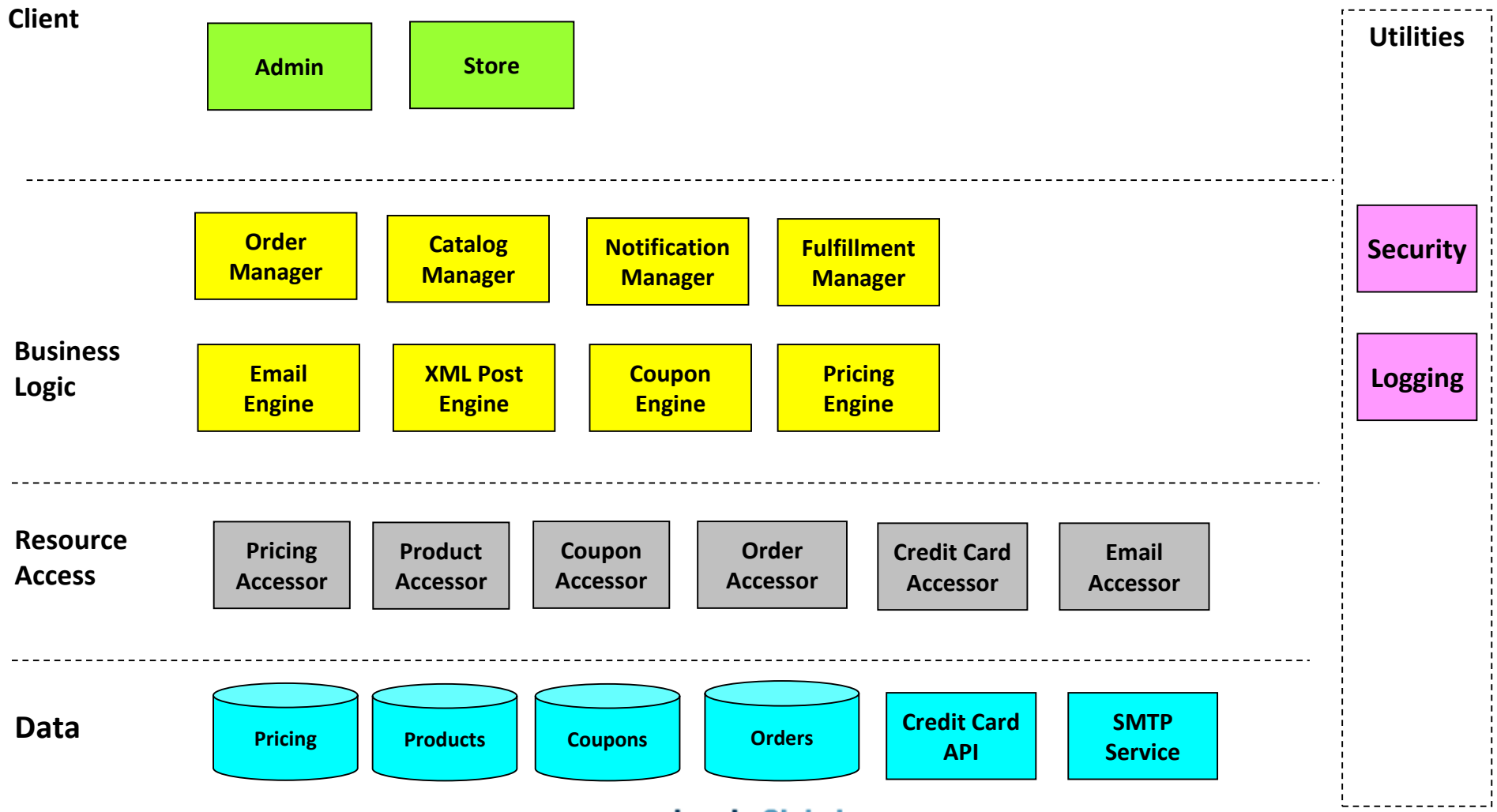
- IDesign “Method”
  - http://www.idesign.net
  - Search “Zen of Architecture”
- Simplified conventions and rules
- Encapsulation of volatility / design for change
  - Information hiding
- Every class is a “service”
  - Service orientation vs object-orientation
- Multiple views of the design

Consistent design conventions

- Basic abstractions
  - Managers: Manages sequence of actions
  - Engines: Applies algorithm/business rule
  - Accessors: Data/resources
  - Utilities
- Layers (typical)
  - Clients
  - Business
  - Resource Access
  - Resources
- Communication rules
  - Closed architecture
  - Services can only call down
    - Utility exception
  - Not allowed:
    - Manager -> Manager (sync)
    - Client -> Engine
    - Client -> Accessor



Static diagram



Characteristics of a Good Design

- Minimal complexity
  - Cleverness == complexity
  - Minimize your required “field of view”
- Ease of maintenance
  - How easy will it be for any programmer to understand?
- Loose coupling
  - Components with few connections/dependencies on other components
- Extensibility
  - Enhancements to the system do not cause a ripple effect
- Encapsulation of volatility (future change)
  - Inevitable changes are localized
- Closed system
  - Restrictions on inter-component communication
- Class/unit level testability
  - Peace of mind and ability to refactor

Layered approach to quality

- Effectiveness of test processes (source: “Code Complete”, 2nd Edition)
  - Regression Test (30%)
  - Code Reviews (35%)
  - Integration Test / Design Reviews (40%)
  - Unit Test (50%)
  - Desk-Checking (60%)
- Our layered approach
  - Pair programming
  - Unit tests
  - Code review via GitHub Pull Requests
  - Regression tests
  - Integration tests
- Automation
  - Leverage CI tools for build and deployment (even if we are not doing CI)

Layered approach to quality

- Lead engineer with “big picture”
  - Ensuring design integrity is maintained
- Focus on developer accountability for quality
  - Removal of QA “safety net”
  - Code + Tests == Feature
  - No separate “develop tests” tasks

Layered approach to quality

Sprint done

- Coding standards met
- User Stories code complete
- Unit test developed/passed
- Test Clients completed
- Priority bugs fixed (blocking bugs)
- Priority bugs validated
- User Stories documented for QA/documentation

- NOTE: These activities fall within the development sprint (i.e. no QA involvement here)

Release done

- QA Acceptance
- Stress testing
- Scalability testing
- Performance testing
- Finalize Customer documentation
- Migration checklist
- Final QA regression passed
- Final Integration test passed

- NOTE: these activities fall outside of the development sprints (i.e. QA focus here)

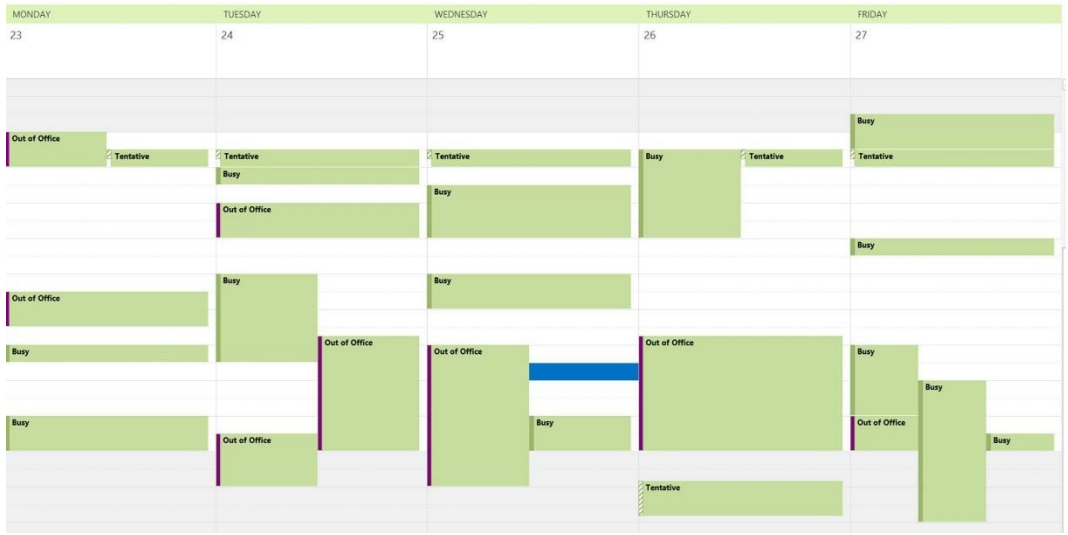
Additional factors improving developer productivity

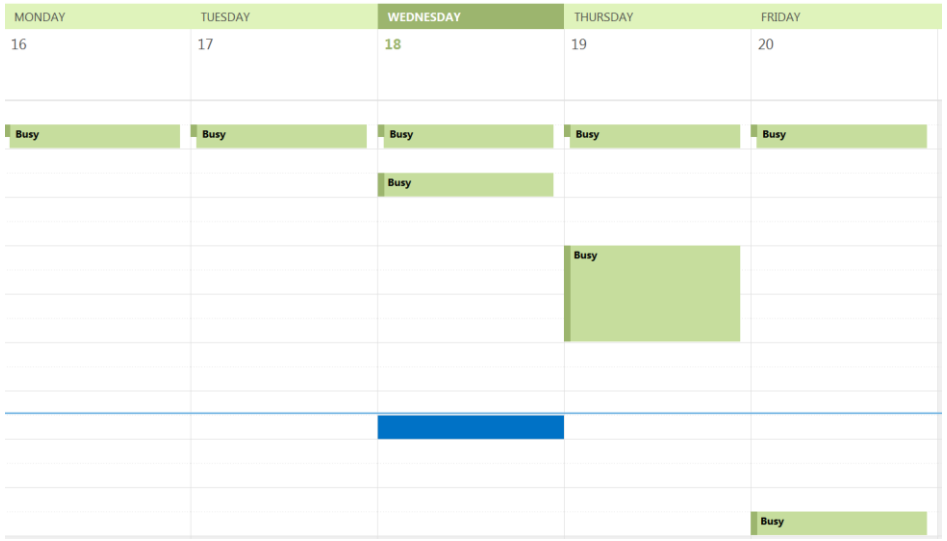
Project manager as “force multiplier”…

- Primary responsibility: Process facilitator
  - Ensure steps are followed
  - Maintain consistency
  - Keep a productive rhythm
  - Schedule/facilitate meetings
  - Keep meetings productive
  - Ensure proper task prioritization
- Central communication for project
- Tight coordination with lead engineer, UI/UX, QA and product manager
- Decision tracking and documentation
- Release plan development
- Task/action item tracking
- Project status monitoring / reporting
- Project health monitoring / reporting
- Information/decision coordination
- Retrospectives
- Management of external communications
- Lead daily standups

Additional factors improving developer productivity

- Maker vs Manager Schedule
  - Paul Graham (Y Combinator): http://www.paulgraham.com/makersschedule.html



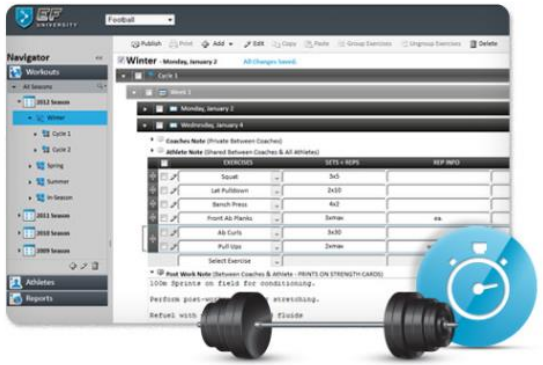


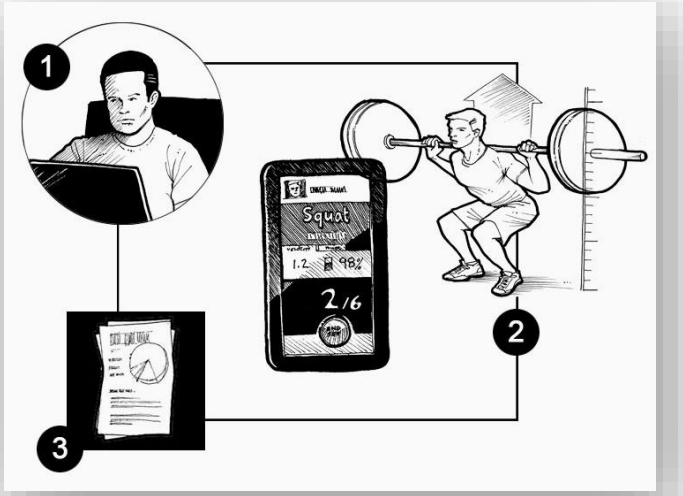
Case Studies

- EliteForm
- Beehive

EliteForm Product

1. Design, Personalize, Deliver





2. Receive Instant Feedback and Automatically Track Results

3. Track and Analyze Data

nebraskaGlobal
/* Develop here */

EliteForm

- Key Challenges and Design Criteria:
  - Providing a simple to use, yet flexible, user interface for design of strength and conditioning programs
  - Patent Pending Algorithms for tracking athlete weight-lifting exercises with extremely high degree of accuracy
  - Creating a seamless, flexible, and low-touch user experience for athletes
  - Providing mobile access for coaches and athletes
  - Enabling access to data for ad hoc analysis by researches and caches
  - Managing remote embedded systems
  - Creating a device and experience that becomes a seamless part of an athlete's workout and survives the harsh conditions
  - R&D Process for Building Technology
- Application Architecture Highlights
  - Power Tracker runs on a pc-based embedded appliance running Windows Embedded OS
  - Strength planner is a web-based client-server system with servers hosted by Nebraska Global
  - Paperless is a platform-agnostic web app
  - Microsoft SQL Server for database back-end
  - Cloud or on-premise deployment
  - Multi-tenant where appropriate or private, segmented databases as necessary
  - OLAP database for ad hoc analysis & data mining
- http://www.eliteform.com

EliteForm

- In use by dozens of elite athletic teams and military units
- Millions of repetitions tracked

- 8 developers and 2 applied mathematicians
- 10 months from start to use by NU Football team
- 1 hot fix in last 12 months
- Developed an automated test cluster for algorithm validation

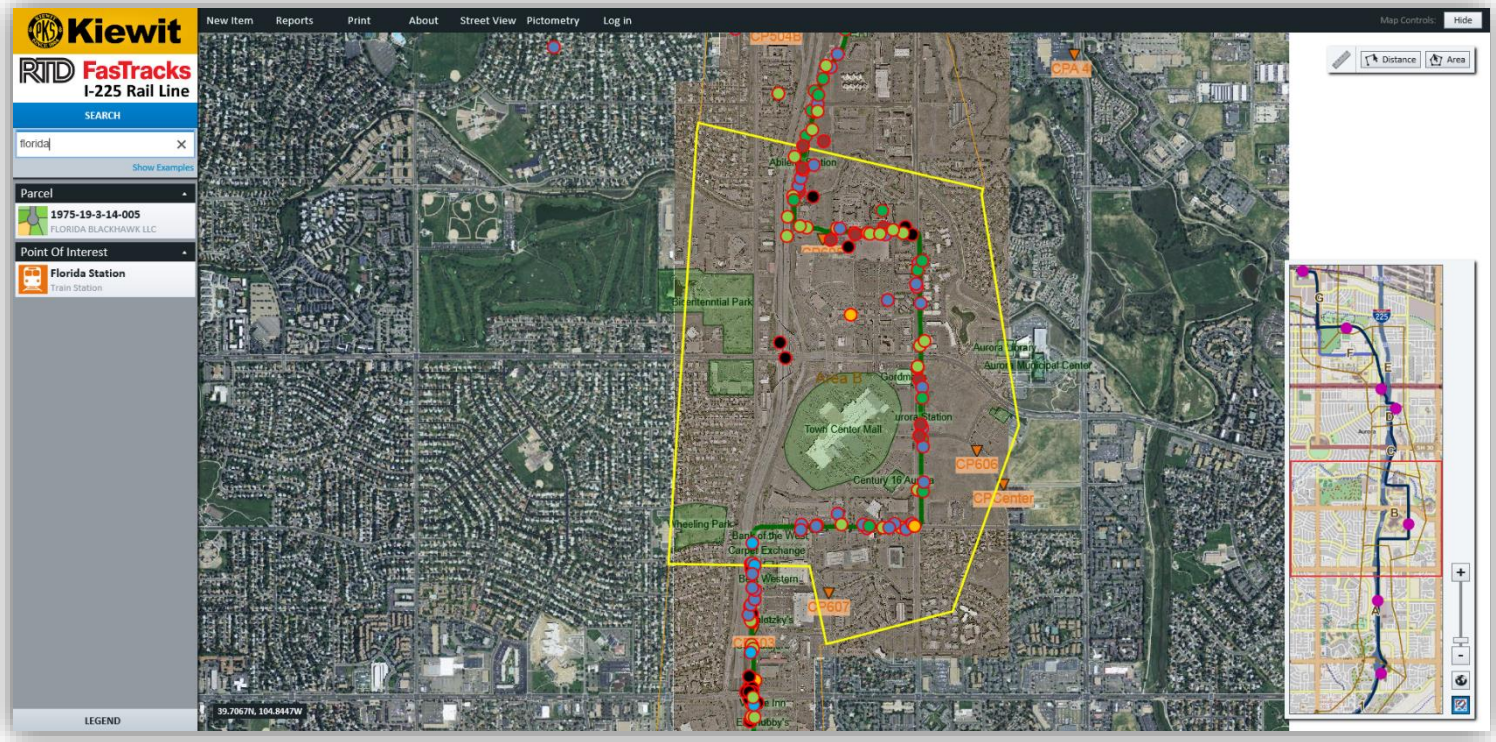
Beehive Public Sector

nebraskaGlobal

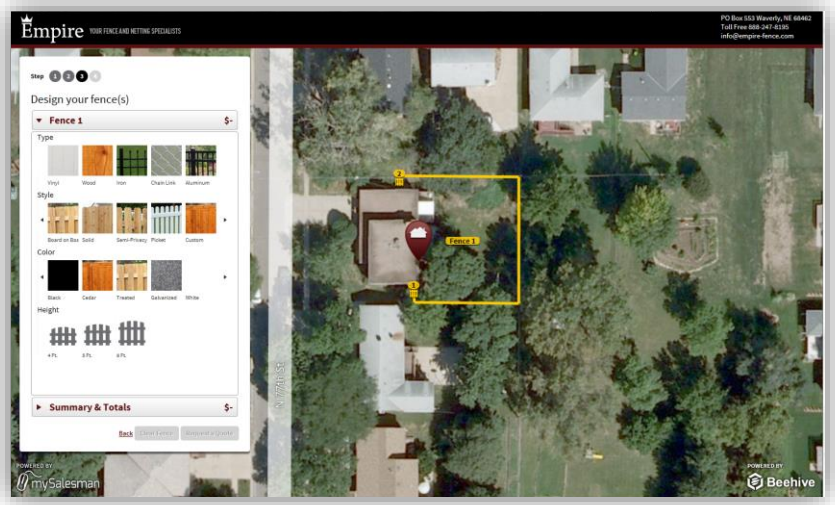
/* Develop here */

Beehive as a Platform

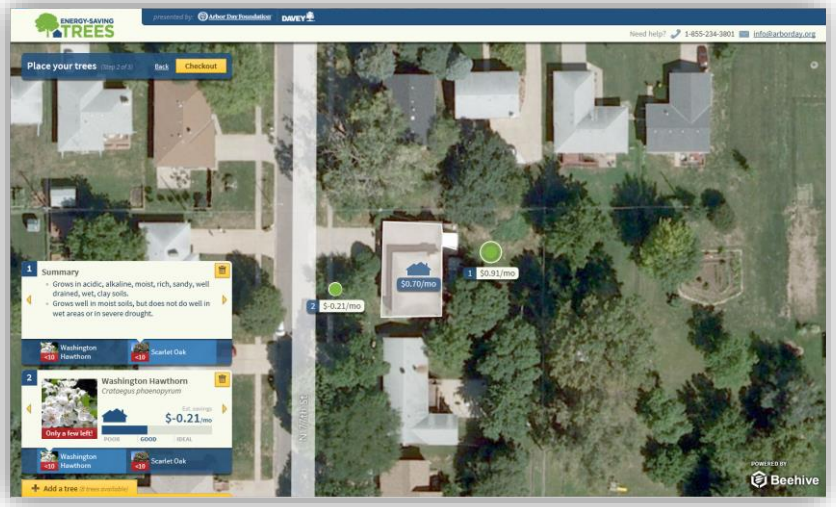
Highway construction management



Self-service fence design



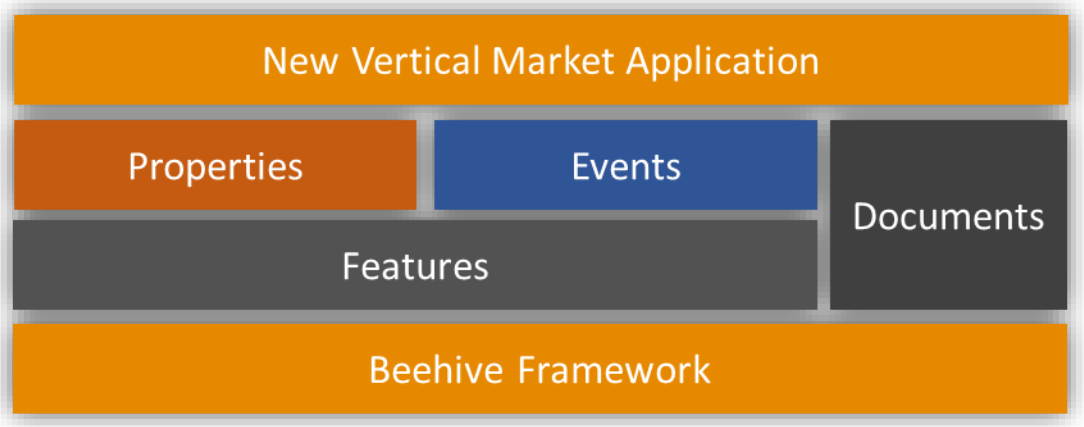
Optimized tree placement



nebraskaGlobal
/* Develop here */

Beehive

- Key Challenges and Design Criteria:
  - A platform that could be extended to a variety of industries and applications with minimal dev
  - Provide for disconnected use while leveraging a cloud-based infrastructure
  - Intuitive interface for knowledge workers and general public
  - Comprehensive, yet user-friendly public web portal for access to GIS data
  - Integrate with existing GIS database systems as well as working standalone
  - Provide a mobile experience that enables data access and collection in the field
- Application Architecture Highlights
  - Combines rich desktop client with web portal
  - Data synchronization across clients and cloud
  - Hybrid on premise/cloud system
  - Hosted in Microsoft Azure and Amazon AWS
- http://www.beehiveindustries.com/



Beehive

- In use by dozens of public works departments and other agencies

- 8 developers, 1 QA engineer
- 6 months from start to use by first city
- Automated the configuration control and nightly testing of all customer configurations
- Weekly unattended releases
  - Include code and database schema updates

Key Takeaways

- Agile processes work but don’t be afraid to hybridize
- Building software is a complex activity – why wouldn’t you have a plan?
  - Project plan
  - Architecture/design plan
- If you design your software to be unit testable, you are “80% there”
- Don’t be afraid to make developers accountable for quality – remove the training wheels (i.e. QA)
- Find a great project manager and trust them to handle all of the “noise”
- Protect developer’s time

Combining experience and the body of knowledge

Process Methodologies	Design Patterns and Principles	Business Models	Thought Leaders
Agile	Information Hiding	Business Model Canvas	David Parnas
Scrum	Service Orientation	Lean Startup	Fred Brooks
Extreme Programming	High Cohesion	Minimum Viable Product	Martin Fowler
Kanban	Loose Coupling		Robert C. Martin
Test-Driven Development	Dependency Injection		Steve McConnell
Waterfall	Closed Architectures		Kent Beck

Thanks!

"If builders built buildings the way programmers wrote programs, then the first woodpecker that came along wound destroy civilization."

Gerald Weinberg

- ddurham@dontpaniclabs.com
- cmichel@beehiveindustries.com

- http://blog.dontpaniclabs.com