Funability

Doug Durham

Chad Michel

nebraskaGlobal
/* Develop here */

DON'T PANIC LABS.

Why did we get into software development?

- Rapid return on our effort
- Work on tough problems
- Build tools that people use
- Enriching our lives
- Building something innovative
- Impacting people's lives
- Saving money / creating wealth
- Automating complex activities

Problem is…

… the journey many of us are on to seek fulfillment of those goals has required enduring a lot of “pain” along the way.

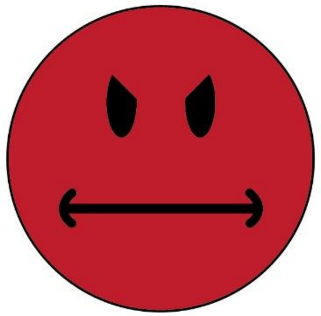
Cool office spaces is not enough

The pain…

- Swim through 5 layers of inheritance
- 12 hour product releases
- 6 weeks of stabilization
- Estimates <> reality
- Dreading project status reviews
- Hours wading through code to determine how something works
- Hope and prayers during releases
- Edge of seat waiting for support calls about system down
- Looking for new projects to avoid maintaining ugly code
- Test environment cumbersome and shared
- Estimates driven by deadlines vs reality
- Silos of design philosophy throughout the system
- …

What we want…

What we enjoy



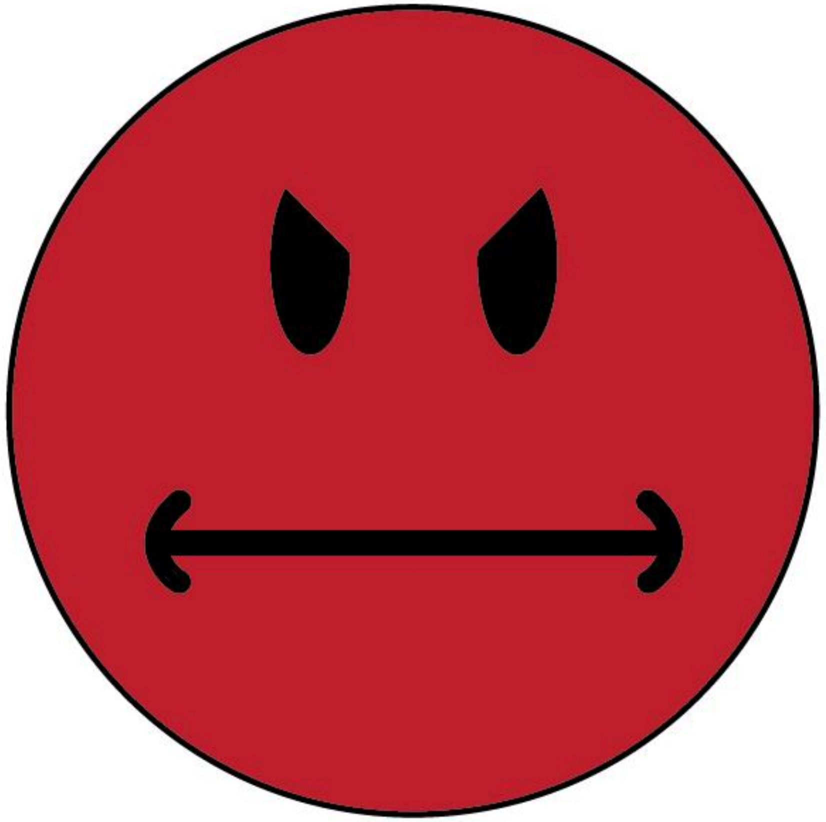
What we loathe

nebraskaGlobal
/* Develop here */

DON'T PANIC LABS.

Reality for most of us…

What we enjoy


What we loathe

How can we turn this around?

We have put a lot of emphasis on the ability of our software teams and development culture to achieve fun and personal fulfillment in our work…

…helping to realize the things that got us into software while minimizing the pain…

… we call this Funability

> … the measure of how well our culture and process enable us to realize the motivations that got us into this business in the first place

What contributes to Funability?

- Frequent Delivery of Value to Customers
- Being Part of a Team
- Maintainability of the System
- Effective Management of Technical Debt
- Sound Software Design
- Consistent Quality of Product Releases
- Productivity and Efficiency of the Developers

What contributes to Funability?

- Frequent Delivery of Value to Customers
  - Visibility of progress by internal stakeholders
  - Regular and frequent releases to external customers
  - Minimizing long, drawn-out development efforts

What contributes to Funability?

- Being Part of a Team
  - Frequent interactions with teammates on project items
  - Ability to leverage pair programming when necessary
  - Esprit de Corps - a feeling of pride, fellowship, and common loyalty shared by the members of a particular group.
  - Mutual accountability amongst the team

What contributes to Funability?

- Maintainability of the System
  - Ability to efficiently read and understand the code throughout the system
  - Ability to effectively debug the system
  - Ability to understand the impact of a change on the entire system
  - Ability to avoid unintended behavior changes
  - Maximizing the useful life of a software system
  - Avoidance of silos in the System

What contributes to Funability?

- Effective Management of Technical Debt
  - Recognizing when choices will lead to technical debt
  - Ability to efficiently reduce technical debt as part of normal feature development
  - Leveraging tools to identify technical debt

What contributes to Funability?

- Consistent Quality of Product Releases
  - Stress-free release days
  - Automation of processes
  - No stabilization phases
  - Hot fixes as the exception, not the rule

What contributes to Funability?

- Productivity and Efficiency of the Developers
  - Creating a project management discipline that reduces the mental burden on developers and leads
  - Enabling designers and developers enough time to actually do some work

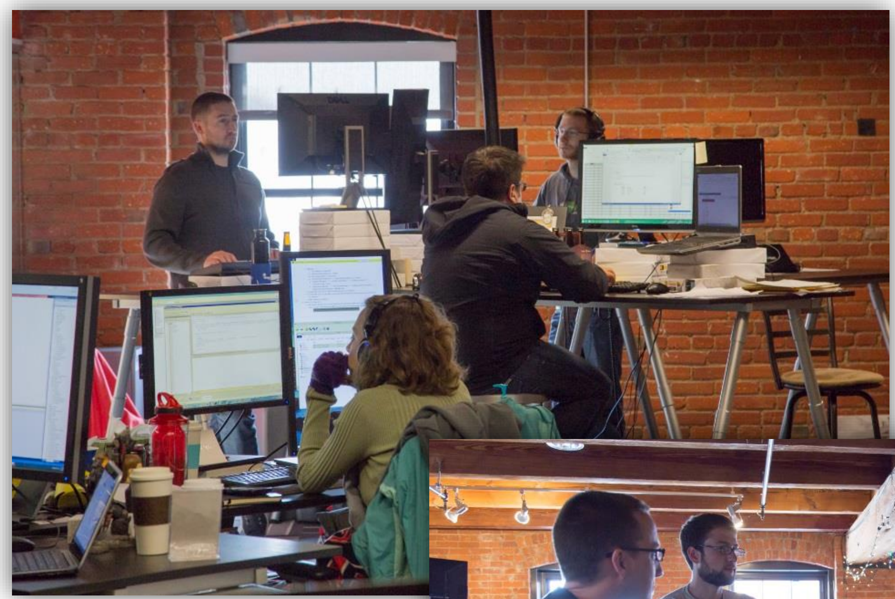
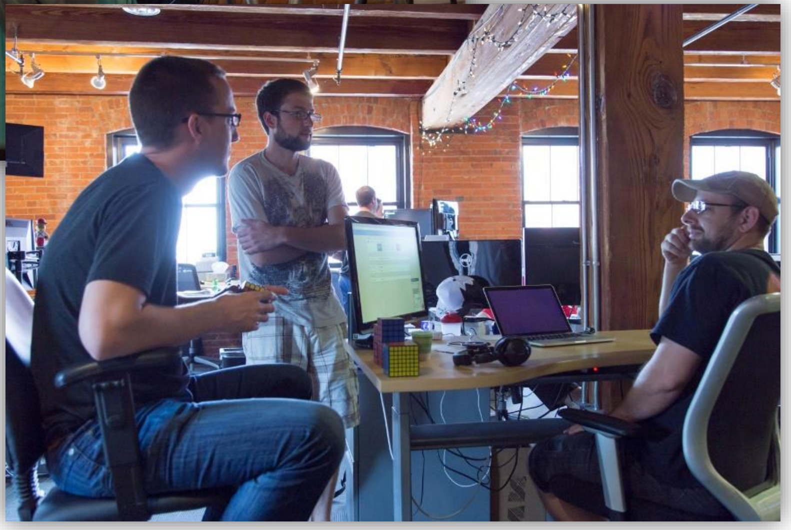
What contributes to Funability?

- Sound Software Design
  - Consistency of the conceptual design of the software
  - Consistent adherence to common design principles and criteria
  - Simplicity over complexity and cleverness
  - Disciplined and consistent approach to decomposition and estimation
  - This is the “bedrock” of our culture

Can’t a lot of this be managed through office layout and agile processes?

Cool spaces are important

- Creates a relaxed and collegial environment
- Enables collaboration
- Helps with recruiting
- Enables play
- Increases socialization

… and Agile methods are essential, but…

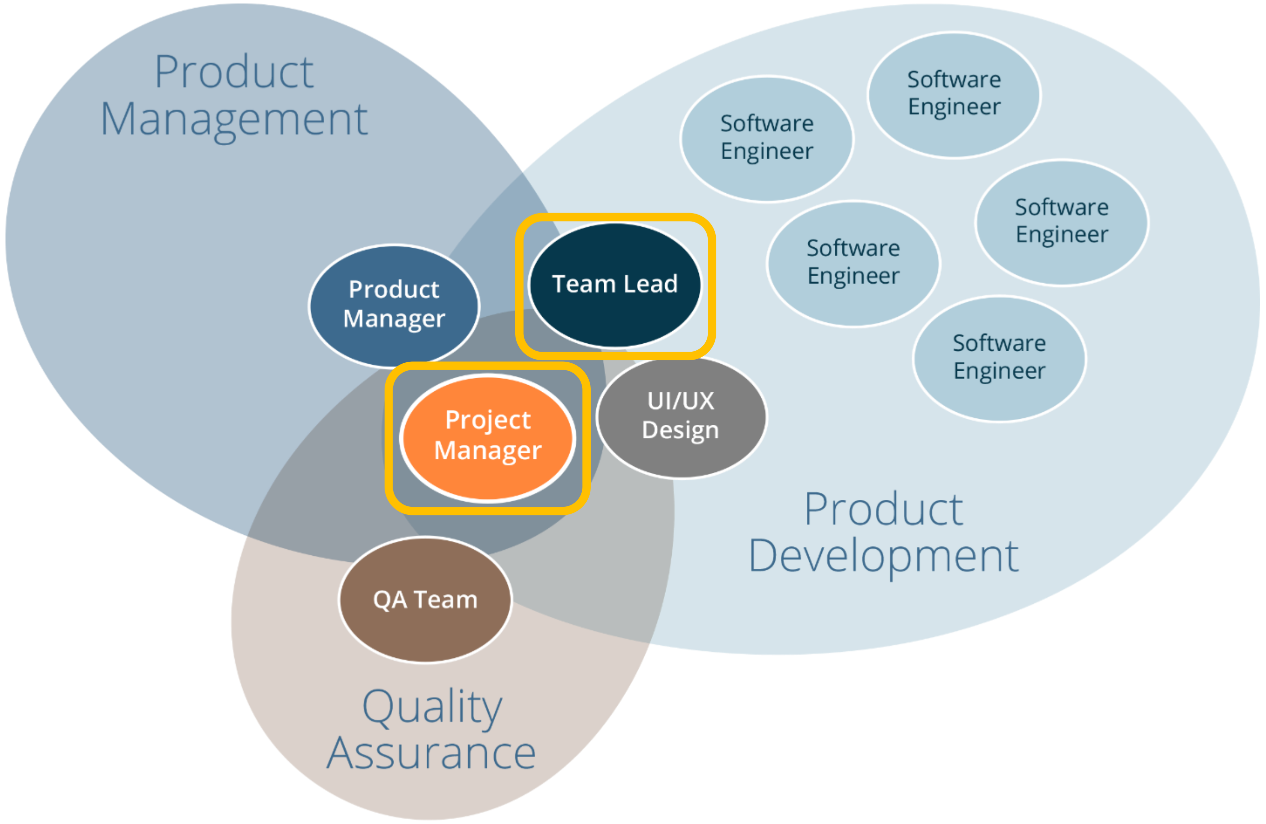
… they are not enough to effectively address and manage the ever-increasing essential complexity of the problems we are trying to solve with software

Bottom line: Agile is not a silver bullet

	Frequent Value Delivery	Part of a Team	Maintainability	Managing Technical Debt	Sound Software Design	Consistent Release Quality	Productivity & Efficiency
“Cool” Space		Ease of interaction and collaboration					A comfortable, exciting, relaxing environment
Agile Methods	Sprints and Kanban methods ensure value prioritization	Daily standups and mutual accountability			Estimation as part of the planning process	Reduced scope of short sprints reduces release risk	Rituals enable collaboration and early issue resolution

Strategies and techniques for increasing Funability

Key Team Roles



Establish a strong development lead role

- Lead Developer
  - Qualities of good programmer +…
  - Coaches Jr programmers
  - Works with programmers to design new features

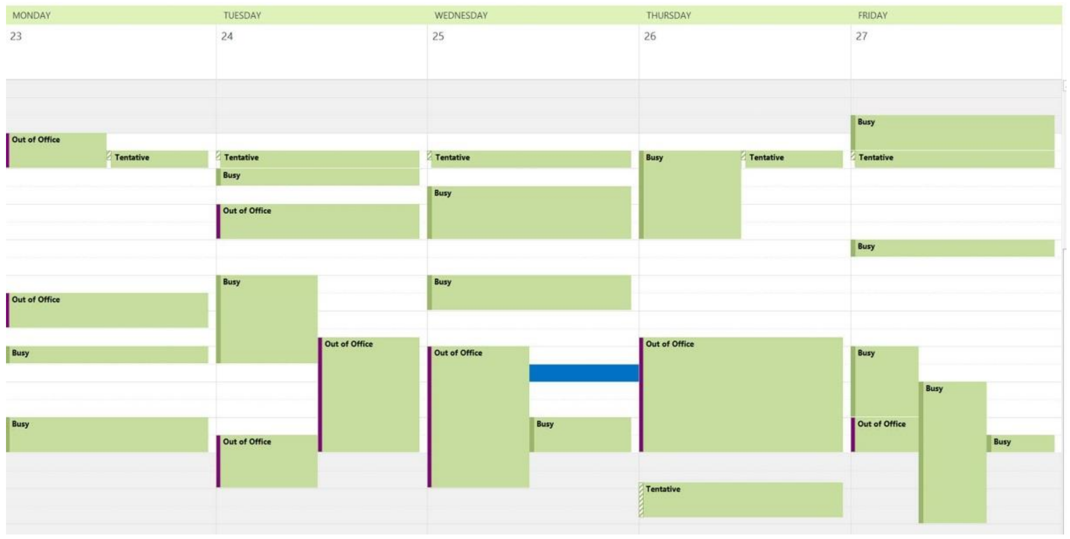
- Lead Engineer
  - Qualities of good engineer +…
  - Coaches and mentors team on design principles and standards
  - Responsible for maintaining the conceptual design
  - Maintains big picture of product
  - Proactive communicator
  - Responsible for performance of product
  - Ensures engineers are testing their code
  - Performs code reviews

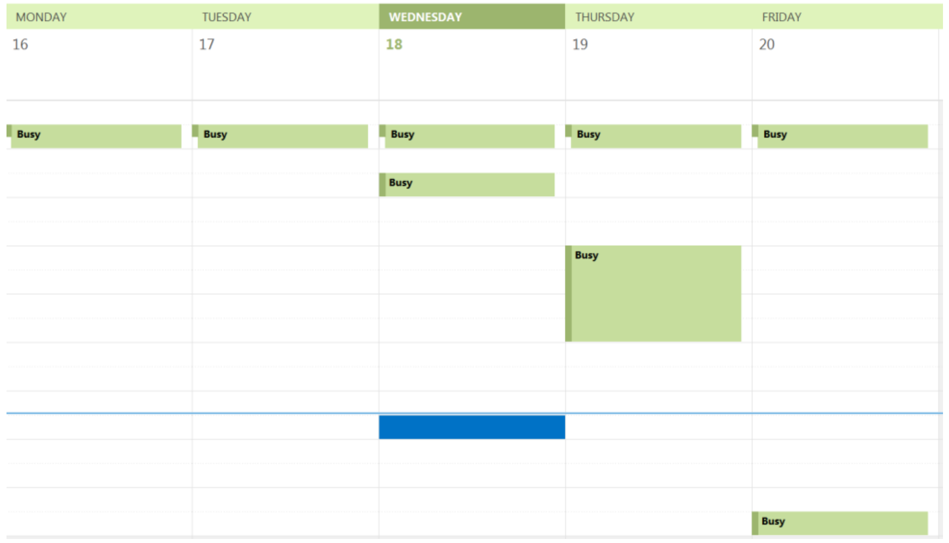
Establish a strong project management role

- 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
- Keeps sprint planning < 1 hour

Protect the schedules of your "makers"

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





nebraskaGlobal
/* Develop here */

DON'T PANIC LABS.

Establish a consistent design identity

- Avoid treating every new project as a unique design effort
  - When done right, the methodology for decomposing a system can (and should) be the same for every project
- System feels created by a single mind
- Ensures things such as testability remain high in all areas of system
- Enables movement of developers from one area of the system to another, and from project to project

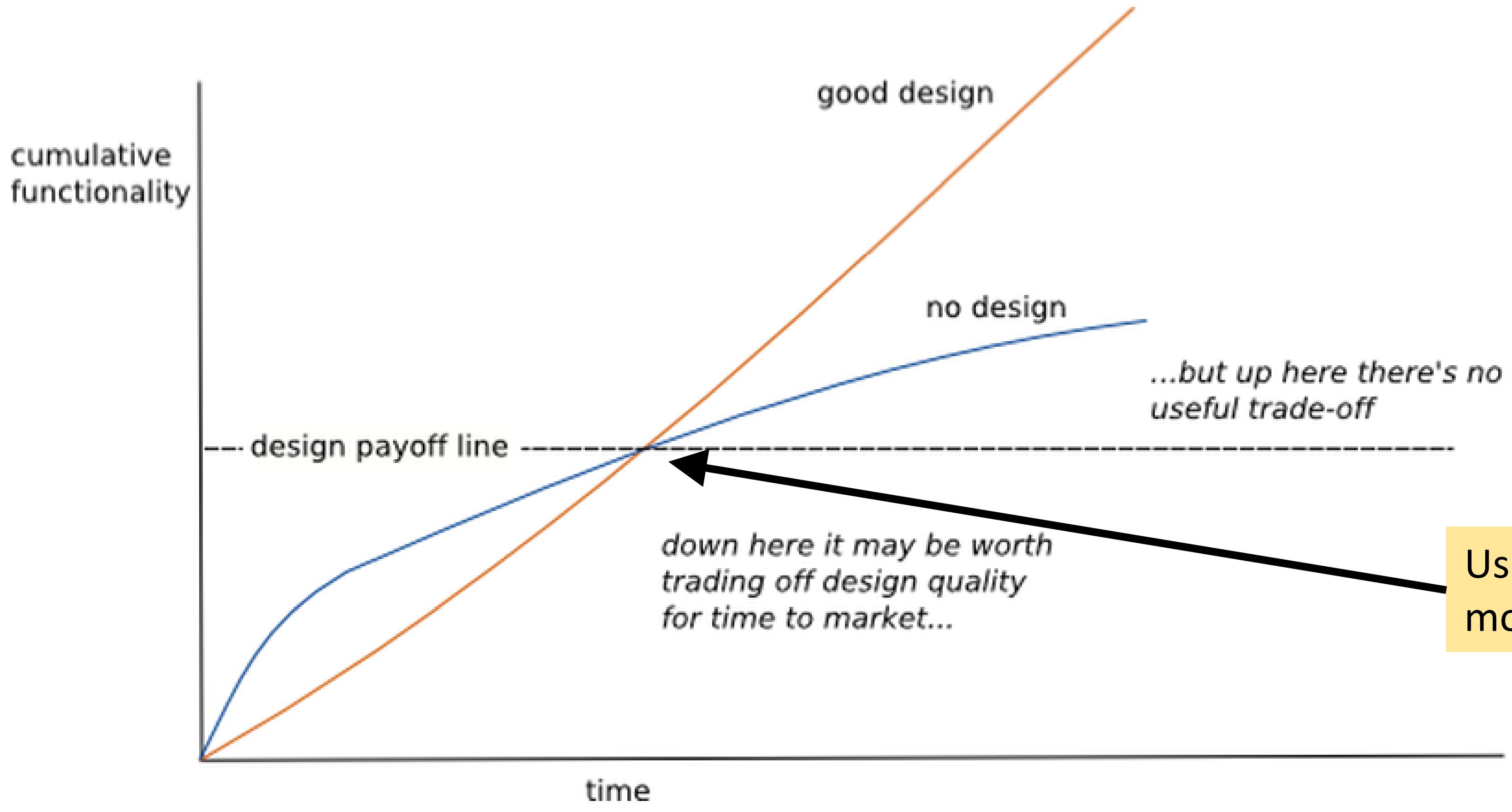
- Examples: object-orientation, services, micro-services, IDesign

Establish a consistent design identity

"I will contend that <u>Conceptual Integrity is the most important consideration in system design</u>. It is better to have a system omit certain anomalous features and improvements, but to reflect one set of design ideas, than to have one that contains many good but independent and uncoordinated ideas."

Fred Brooks (1975)

Design Stamina Hypothesis



Usually weeks, not months

http://www.martinfowler.com/bliki/DesignStaminaHypothesis.html

Practice test-driven design

- Forces consumption awareness in your code because you create the first consumer
  - Gets you focused on the interface rather than the implementation
  - Tends to create interfaces that are conveniently callable
- Forces the software to be more testable which usually requires more decoupling from its surroundings
- Things that are difficult to test tend to be simplified in order to achieve testability
- Tests allow you to play “what if” games with broad changes to assess the impact to the design
- Makes the code more understandable/readable
  - Unit tests that describe how the developer intended the code to be consumed
  - Built in example code!

Require code review of every pull request

- Code reviews single best way to improve quality
- Reduces the stabilization cycle
- Enables us to develop confidently
- Provides mechanism for coaching and mentoring
- Ensures the code is consistent with the software architecture/design

Use continuous integration with tests

- We want confidence that our systems work
- We don’t want that dreaded support call
- Enables us to sleep at night
- Avoids “broken window” syndrome

Require ability to do integration tests on the desktop

- Enables a developer to build and run end-to-end tests and validation on their own machines
  - Includes hosting of databases locally
  - Avoids collisions with other developers
  - Enables development in isolation
- Ability to do this must be a conscious design all along the way
  - Your system design choices can enable or prevent this

	Frequent Value Delivery	Part of a Team	Maintainability	Managing Technical Debt	Sound Software Design	Consistent Release Quality	Productivity & Efficiency
“Cool” Space		Ease of interaction and collaboration					A comfortable, exciting, relaxing environment
Agile Methods	Sprints and Kanban methods ensure value prioritization	Daily standups and mutual accountability				Reduced scope of short sprints reduces release risk	Flexible processes Rituals enable collaboration and early issue resolution
Strong Lead Engineer		Mentoring junior developers	Ensure code standards met	Maintains “big picture” enabling proactive tradeoff evaluation and decisions	Maintains conceptual design integrity	Accountability for product releases	Maintains focus of the team and prevents thrashing
Strong Project Management	Ensures clear definition of done and user expectations Maintains alignment between product owners and development team			Maintains backlog list of items where we need to come back because of shortcuts taken		Manages expectations across all stakeholders through proactive communication	Mental burden of planning, decision, and stakeholder details lifted from dev team
Maker Schedules		Increased availability for spontaneous collaboration				Less disruptions can mean increased focus and better results	Large blocks of time to “get into” the problem at hand
Consistent Design Identity	More efficient estimation and planning of work	Increased collaboration efficiency as a result of reduced design silos	Software designed to encapsulate volatility making future changes easier	Enables explicit decision for design deviation	Consistent adherence to common design principles and criteria helps reduce entropy/software rot Consistent approach to system decomposition		Narrow developer FOV Increased shared understanding of the whole system
Test-Driven Design	Helps ensure main branch has releaseable code	Quality/design accountability increased and shared mutually amongst the team	Code-base is constantly under some form of test Reduces unintended changes in behavior	Helps identify areas of potential technical debt Enables refactoring	Enables adherence to best practices related to design	Defect detection prior to formal QA testing Reduces level of stress during releases	Narrow developer FOV
Pull Request Code Review	Helps ensure main branch has releaseable code	Shared responsibility for the entire code base and shared design	Can enable simplicity over cleverness	Helps identify areas of potential technical debt	Increased likelihood of design problems identified early	Defect detection prior to formal QA testing	Provides coaching and mentoring opportunities to improve skills of the team
Continuous Integration w/Tests	Provides option for continuous deployment	Accountability to the team for consistently successful builds	Keeps a steady benchmark of a verified system	Can integrate code quality static analysis		QA can focus more on acceptance and regression testing vs defect detection	Broken builds encourage attention to detail and discipline
Desktop integration Tests	Increases velocity of development team		Helps prevent/avoid failed builds		Encourages modularization and decoupling of system and designing for subsystem isolation and ability to use mocking techniques	Defect detection prior to code review Easier to set expectations on developers for code quality	Increases ease of testing Reduces need for stabilization Reduces friction around environment setup

Summary

- No silver bullet to creating a dev culture with high Funability
- To truly change your development culture you need to go beyond cool spaces and agile/scrum and change the way the software is designed and constructed
  - Only an integrated view of these processes and best practices will get you where you want to be
- Constantly review practices and push for higher Funability
- Challenges remain
  - Still feeling a lot of pain in the web client tier and some mobile app development
  - How to actually measure funability in the workplace

Thanks!

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

Gerald Weinberg

- ddurham@dontpaniclabs.com / @dnsdurham
- cmichel@dontpaniclabs.com / @chadmichel

- http://blog.dontpaniclabs.com / @dontpaniclabs

nebraskaGlobal
/* Develop here */

DON'T PANIC LABS.

Help us answer the age-old question:

Dog or Cat or Robot?

And enter for a chance to win an Amazon Fire Stick

How it works:

1. Take the sticker of your choice.
2. Post a photo of it on Twitter and mention us (@dontpaniclabs).
3. Bonus: Follow us on Twitter for an additional entry.

We'll draw the winner on Monday, May 23 and announce it on Twitter.

nebraskaGlobal
/* Develop here */

DON'T PANIC LABS.