Enabling Innovation

Integrating Agile and Software Development Best Practices

Omaha .NET Users Group
May 2017

DON'T PANIC LABS

About Me

- Current Roles
  - Co-Founder of Nebraska Global
  - CTO at Don’t Panic Labs
  - Adjunct Faculty at UNL Raikes School
- Almost 30 years in software and systems engineering
  - Last 20 years primarily in startups and new product development

Themes of innovation processes

- Chunka Mui (“The New Killer Apps”):
  - Think big
  - Start small
  - Learn fast
- Eric Ries (“Lean Startup”)
  - Build
  - Measure
  - Learn
  - (and complete this cycle as quickly as possible)
- Minimum Viable Products

What is our responsibility?

- Enabling business agility
  - The ability to leverage what is learned in order to build

- How do we do this?
  - Rapid & effective response throughout
    - New information
    - New insights
    - New ideas
  - Collaborating with product owners
    - Balance feature value and complexity

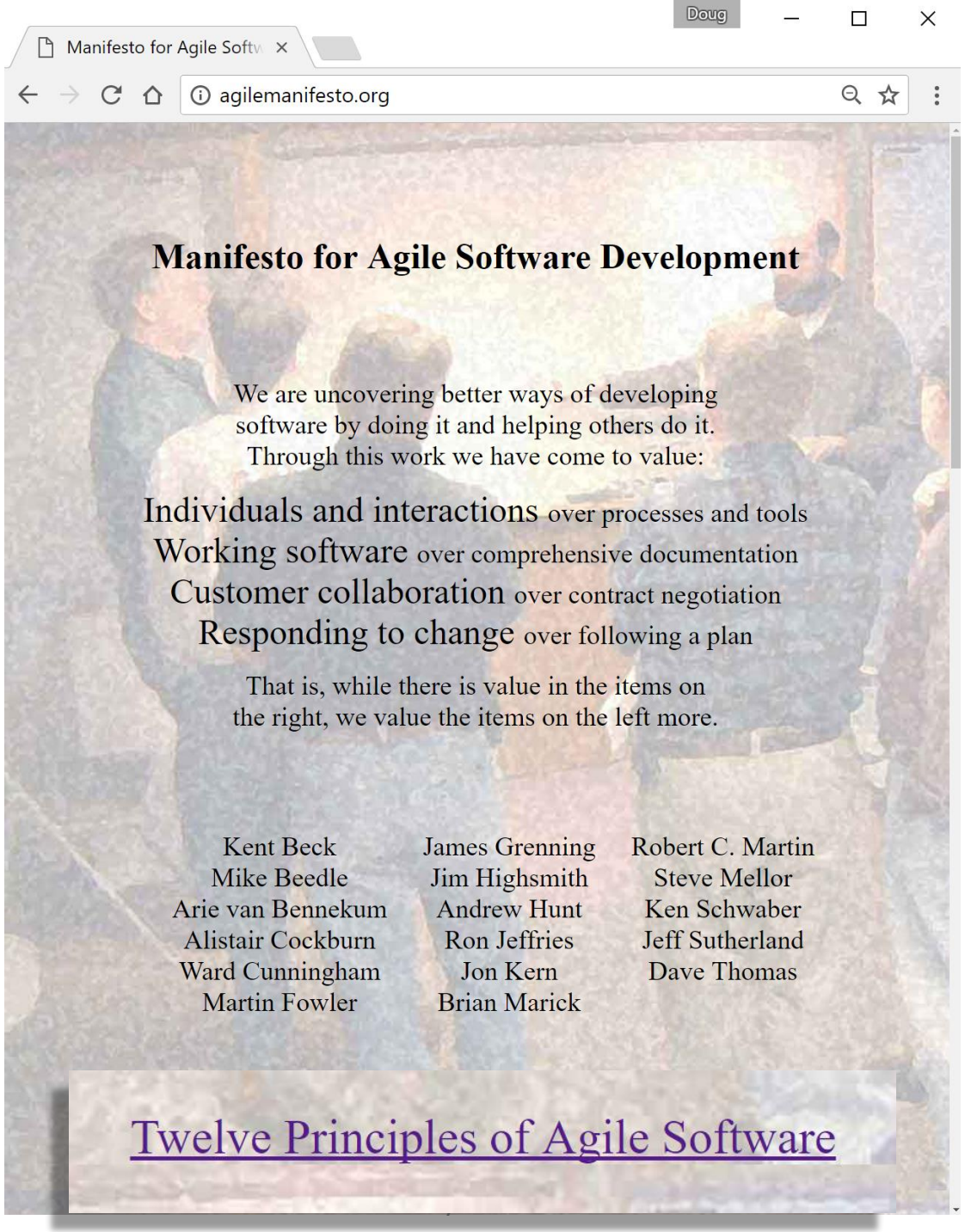
The role of agile processes

We love agile because…

… it establishes customer-centric expectations of value
… it facilitates a dialog between domain experts and engineers
… it focuses the development team on what the current priorities are
… it allows for pivots and re-prioritizations
… it empowers product owners
… it relieves engineers from having to fill a decision vacuum
… it increases accountability for fulfilling commitments
… it increases the predictability of delivery times
… it allows for early visibility by product owners

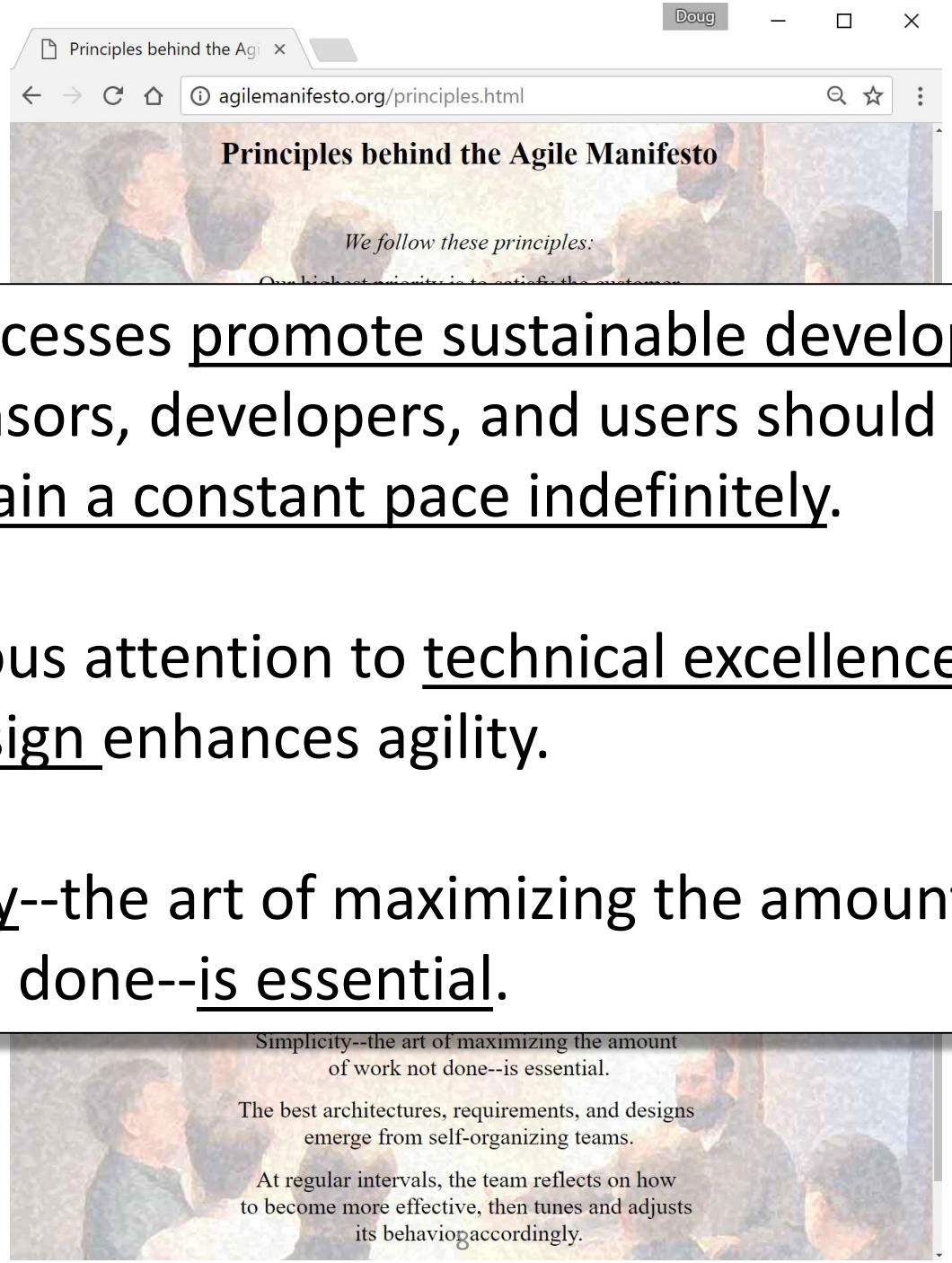
… it helps enable business agility

So, what's the problem?



http://agilemanifesto.org/

DON'T PANIC LABS



Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Continuous attention to technical excellence and good design enhances agility.

Simplicity--the art of maximizing the amount of work not done--is essential.

http://agilemanifesto.org/principles.html

Agile methods are essential, but…

… they are not a silver bullet.

The problems are getting increasingly complex

Our development teams and processes vary in ability and maturity

Increasing complexity

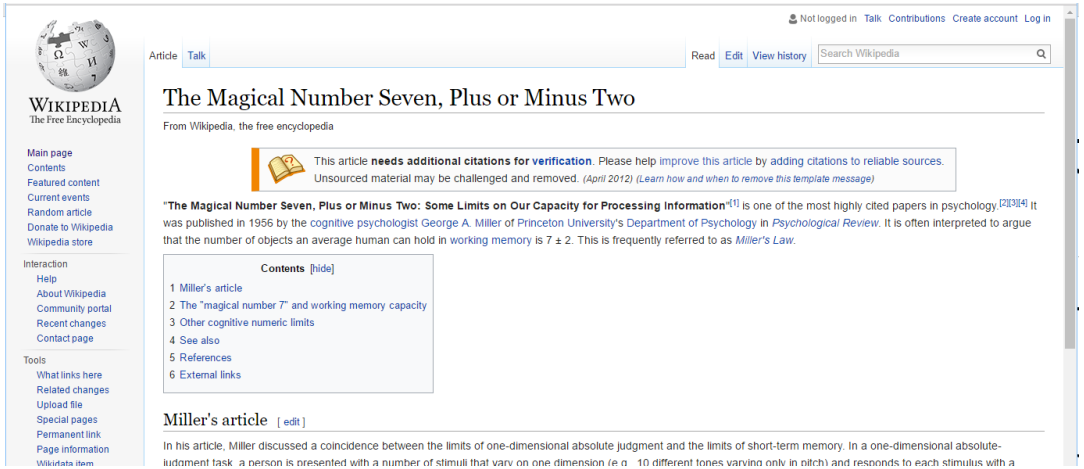
- The problems we are trying to solve today are far more complex than 10-20 years ago
- No signs of slowing

- 20 years ago we might have been building a dog house, now we are trying to build 3000 SF ranch-style homes…

  … and nobody builds a house without a design

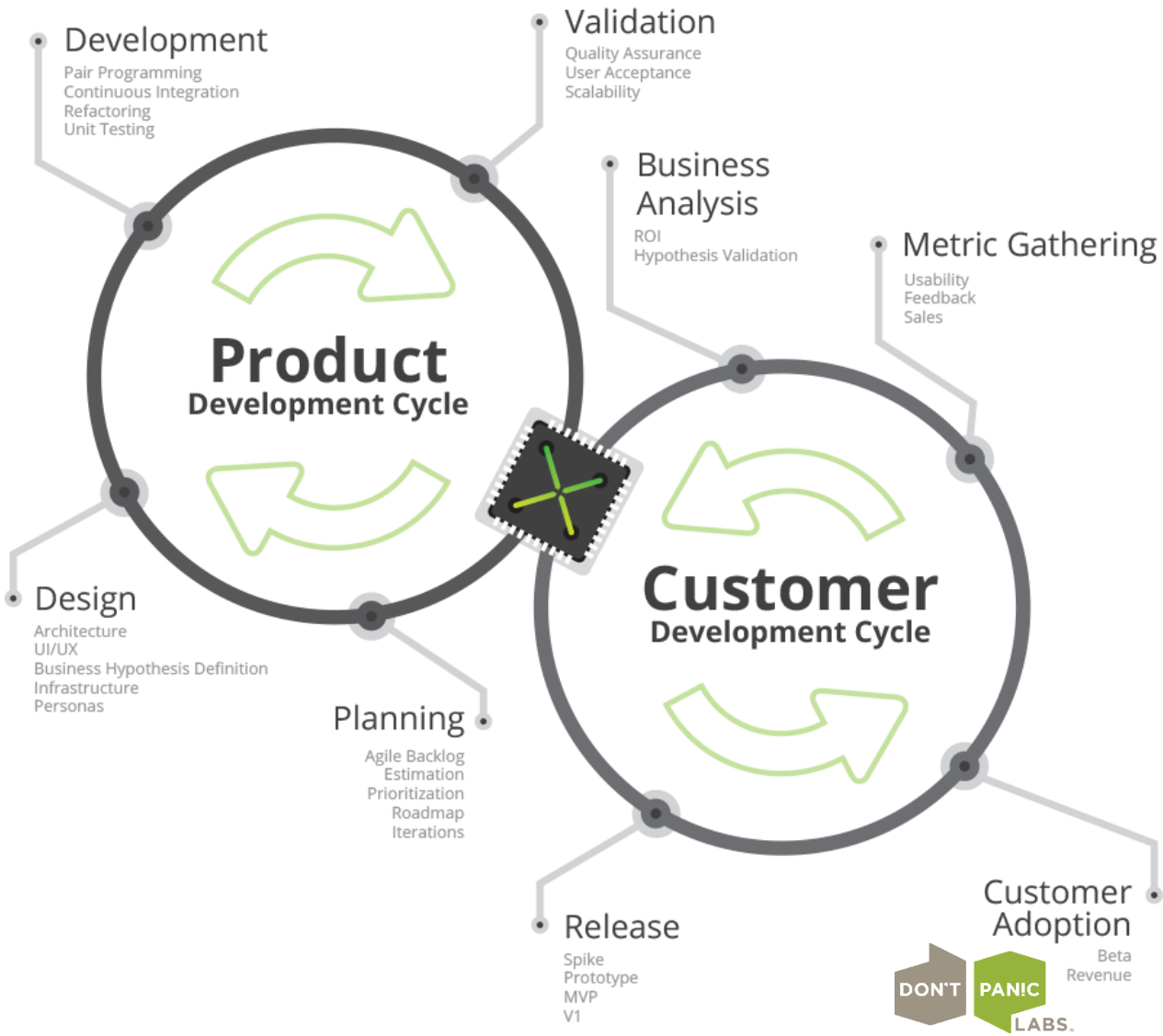
Managing essential complexity

- Unmanaged complexity is the most common technical reason projects fail
  - Humans are not capable of managing all details of a complex program at once
  - Some systems can grow so complex that no one really understands what it does, making further development very difficult



Integrated Agile Product Development

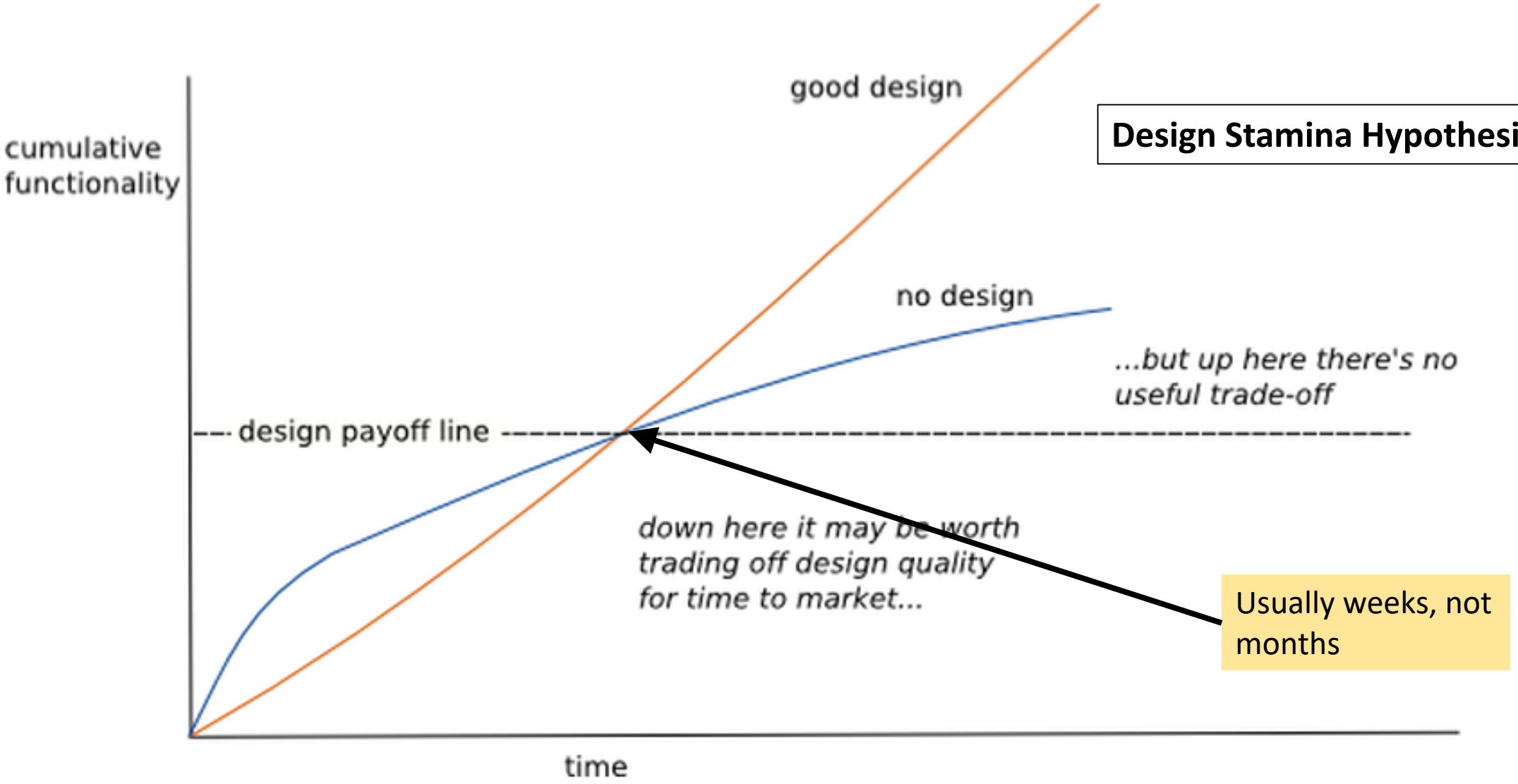
Strong collaboration with tight feedback loops between business and engineering disciplines is critical to a successful launch and ROI.



But, you say…

- “We can’t afford to do this now, we need to get the MVP out!”
- “If this gets traction we will come back later and re-architect or refactor it correctly”
- “No one does design upfront – its not agile!”
  - Building a dog house without a plan

Design Stamina Hypothesis



Usually weeks, not months

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

Software design always happens

The question becomes, do we want our design to be random or coherent and consistent

> Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
>
> Continuous attention to technical excellence and good design enhances agility.
>
> Simplicity--the art of maximizing the amount of work not done--is essential.

Bottom Line…

… we cannot enable business agility if our development environment is on a path to chaos and decreasing throughput

… we cannot escape the realities of software entropy and the accumulated effect of technical debt

https://en.wikipedia.org/wiki/Software_entropy

Our solution for maintaining agility…

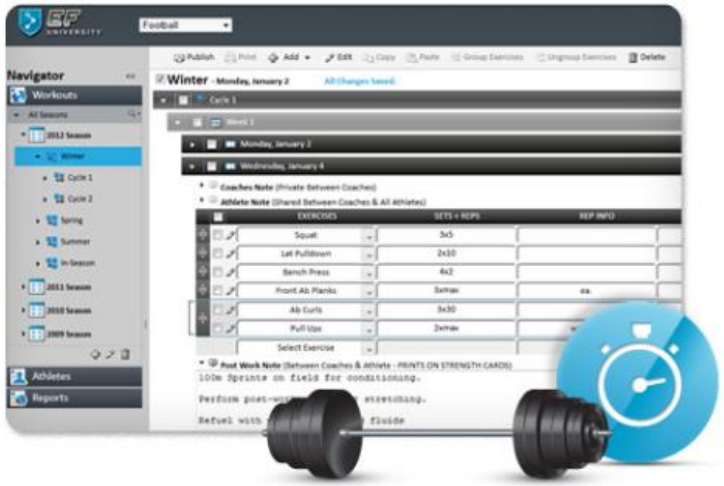
1. Always engineering
2. Reducing uncertainty
3. Designing for change
4. Designing for testability
5. Services vs Objects

Before we get into the details…

Some case studies from our experience…

EliteForm

1. Design, Personalize, Deliver



2. Receive Instant Feedback and Automatically Track Results

3. Track and Analyze Data

EliteForm

- In use by dozens of elite athletic teams and military units
- Millions of repetitions tracked
- http://www.eliteform.com

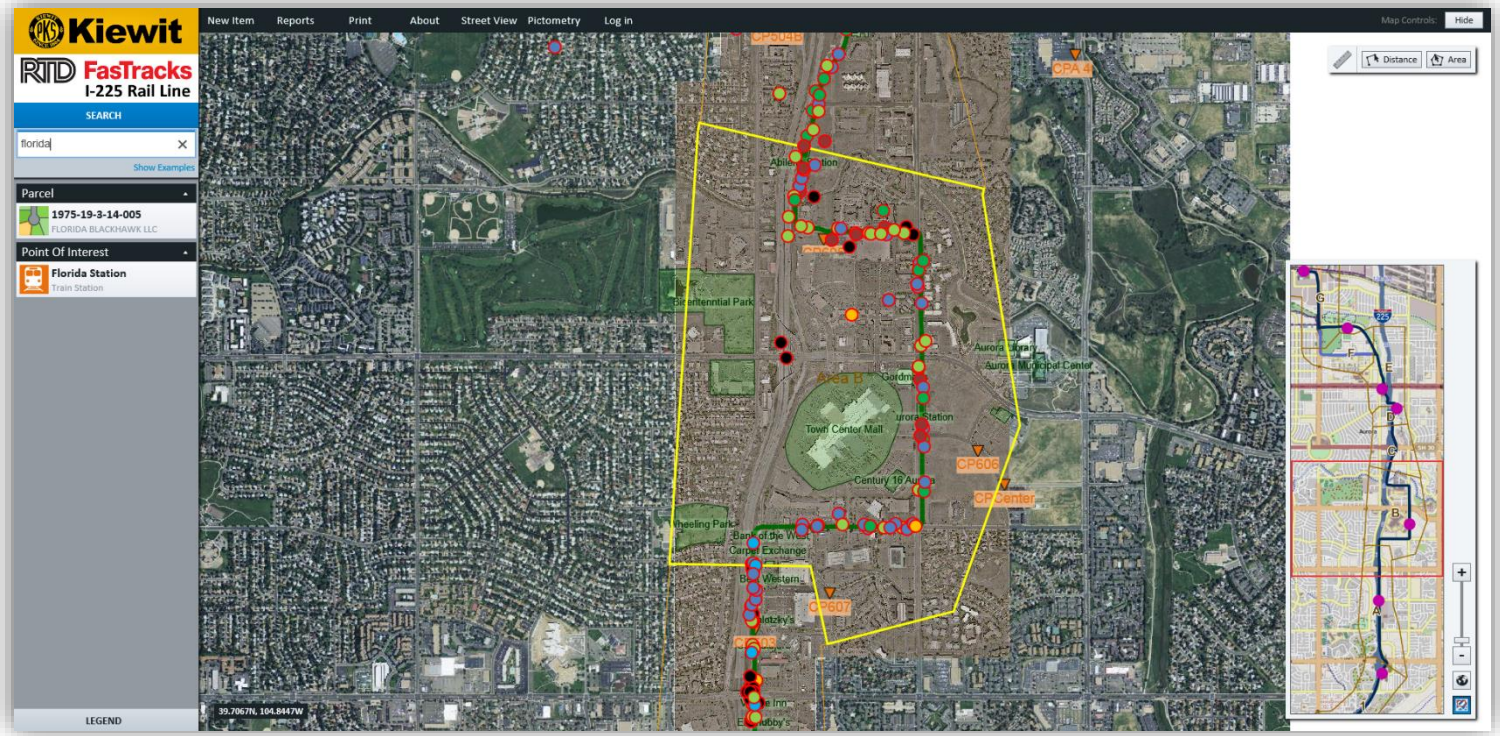
- 10 months from start to use by NU Football team
- 8 developers and 2 applied mathematicians

- 1 hot fix in last 12 months
- Developed an automated test cluster for algorithm validation

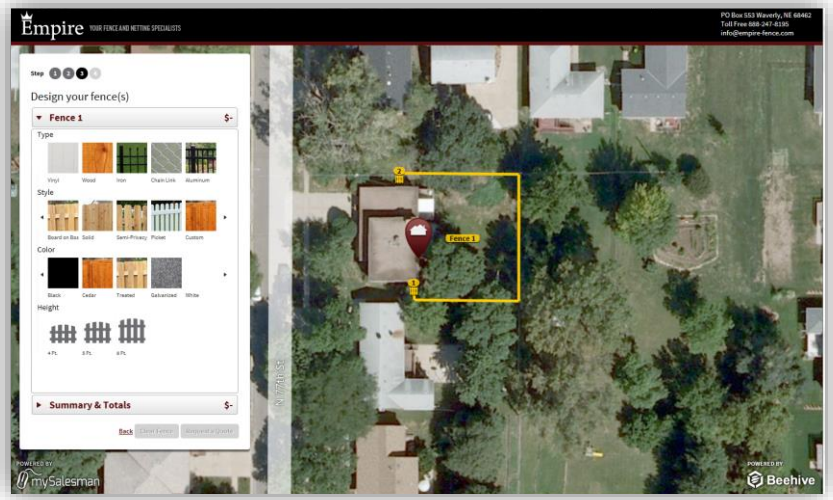
Beehive Public Sector

Beehive as a Platform

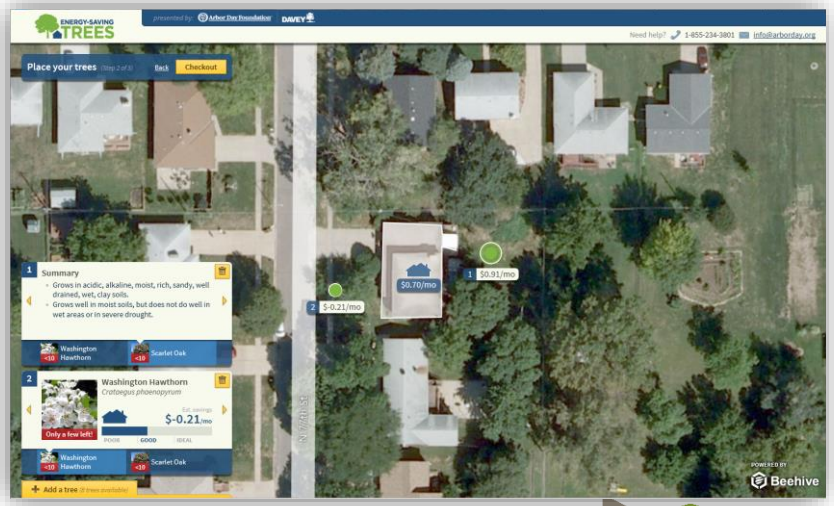
Highway construction management



Self-service fence design



Optimized tree placement



DON'T PANIC LABS

Beehive

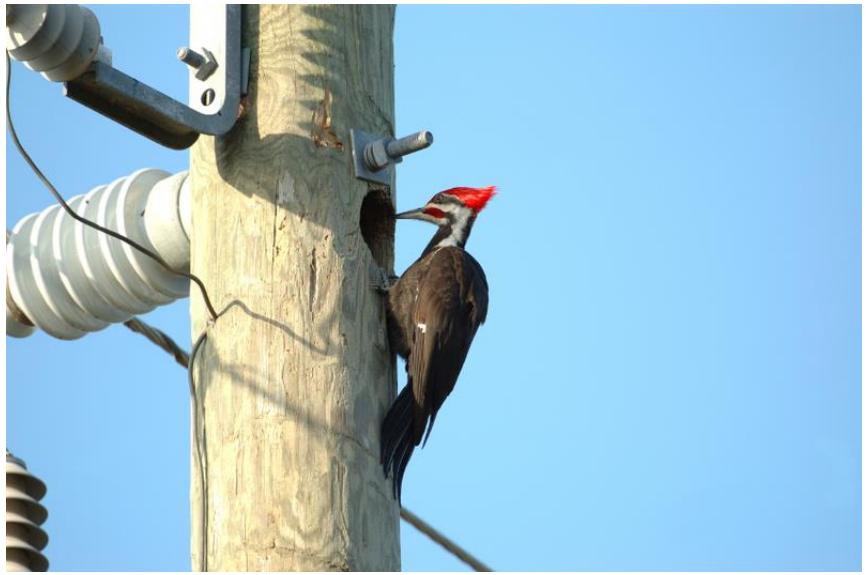
- In use by dozens of public works departments and other agencies
  - Populations of 5,000 up to 300,000 (including Lincoln)
- http://www.beehiveindustries.com/

- 6 months from start to use by first city
- 8 developers, 1 QA engineer

- Automated the configuration control and nightly testing of all customer configurations
- Weekly unattended releases
  - Include code and database schema updates

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

Gerald Weinberg
https://en.wikipedia.org/wiki/Gerald_Weinberg



Always engineering

Design/engineering from story to code…

- Story decomposition
  - How well do we all understand the requirements?
  - What is my plan?
  - What are other factors?
- System decomposition
  - How can I isolate future changes?
- Task decomposition
  - What is the sequence of tasks I will complete to implement this?
- Test-Driven Design
  - How can I maximize the adherence to best practices and principles to enable sustained velocity and refactoring?

Reducing uncertainty

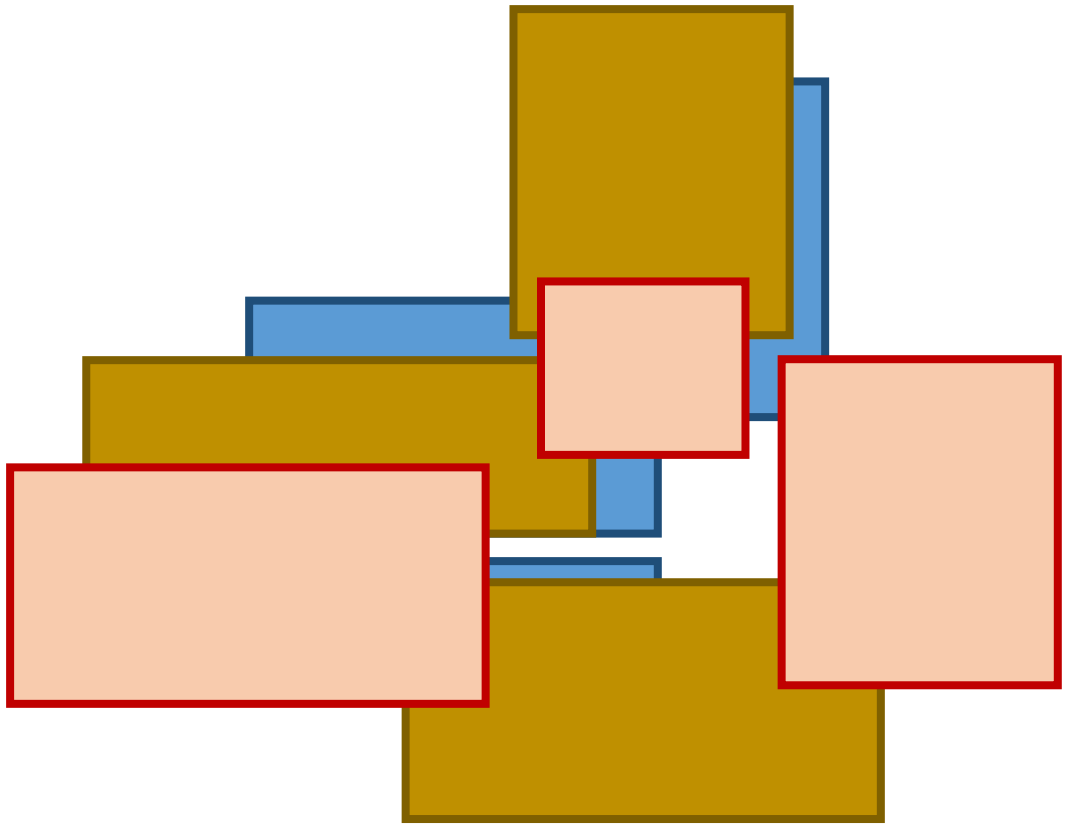
The Danger of Incomplete Pictures

User Story

Product Owner "Picture"

Your "Picture"

QA "Picture"



The Danger of Incomplete Pictures

User Story

Product Owner “Picture”

Your “Picture”

QA “Picture”

Aggregate “Picture”

The Danger of Incomplete Pictures

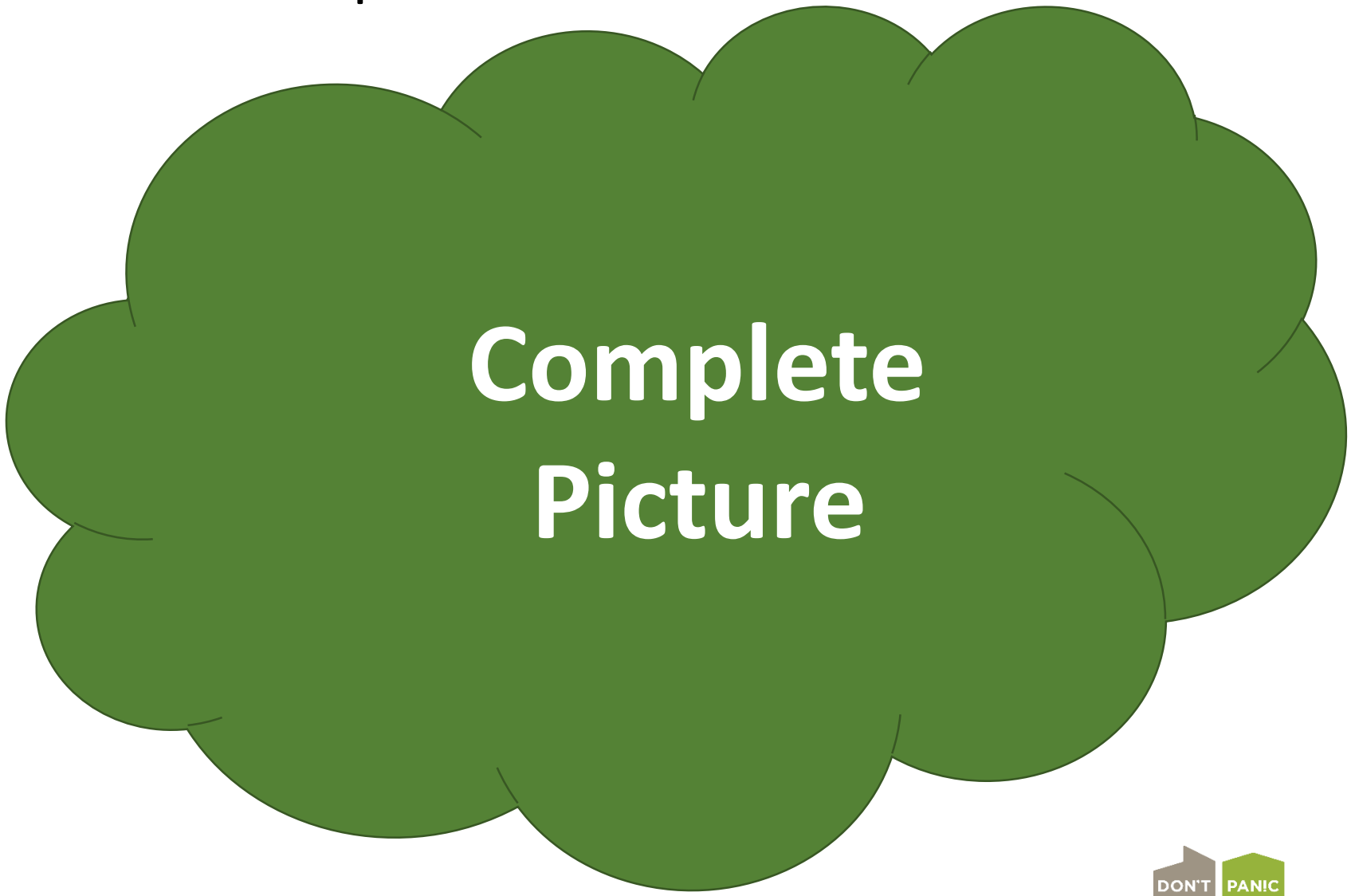
User Story

Product Owner “Picture”

Your “Picture”

QA “Picture”

Aggregate “Picture”



The Danger of Incomplete Pictures

- Two problems:
  - Our assumption that we all have a shared picture
  - Our assumption that all assumptions and requirements are known

- Our solution:
  - Engineering and analysis via design documents

Working through complex problems

- Goal:
  - Facilitate analysis and conversation between engineering and business
  - Gain increased clarity prior to implementation
  - Improve the quality and completeness of the requirements
  - Increase the value of outside input on design
  - Create a shared understanding of requirements → design
  - Improve the quality of estimates
  - Maintain the design integrity of the system

Working through complex problems

- Sample of items to consider:
  - What is my understanding of the requirements?
  - What assumptions am I making about these requirements and the system?
  - What are the existing areas impacted?
  - How will the design/architecture need to change?
  - How can I encapsulate current and future change?
  - How should I test and validate the system?
  - What risks am I aware of?
  - What concerns do I have?
  - What are the specific steps to implement and what is their level of effort?

Design document process guidance

- Identify those stories/epics with significant uncertainty, technical risk, or estimated effort
- Work through the process of analysis by writing things down
  - Use the exercise of writing down your thoughts and decisions as a way to uncover hidden assumptions and holes in the requirements
- Structure yours thoughts around the specific considerations
- Time box your effort to an hour or two
- Use whatever format works best for you
  - Pictures, words, flowcharts, etc
- Final section should be a set of implementation tasks with estimates

[Design Document Title]

Author(s):

Created Date:

Last Updated:

What is my understanding of the requirements? What is the primary goal or objective?

What are the desiderata or secondary goals/objectives?

-

Applicable Stories in the backlog:

Persona	Story	Version	Call Chain

What are the known design constraints?

-

What are the acceptance criteria for the solution (including non-functional)?

-

What assumptions am I making about these requirements and the system?

-

What are the unknowns? (i.e. what information is not currently available?)

-

What areas/features are most likely to change over time?

-

What are anticipated/possible failure scenarios and how should they be handled?

-

What are the existing areas impacted?

-

How will the design/architecture need to change?

-

How can I encapsulate current and future change?

-

How should I test and validate the system?

-

What risks am I aware of?

-

What concerns do I have?

-

What are the specific steps to implement and what is their level of effort?

-

Rev A-20170425

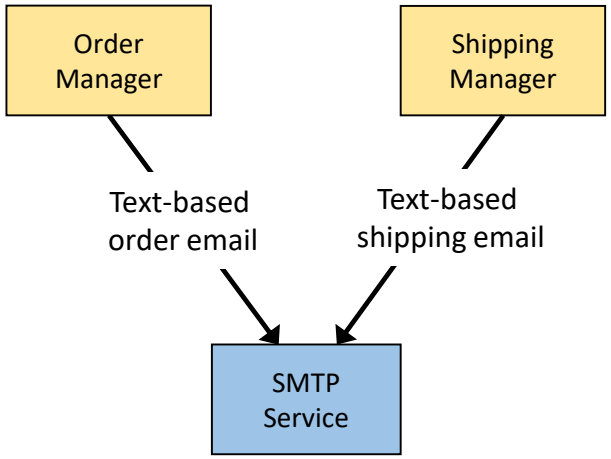
Designing for change

Decomposition based on isolating change

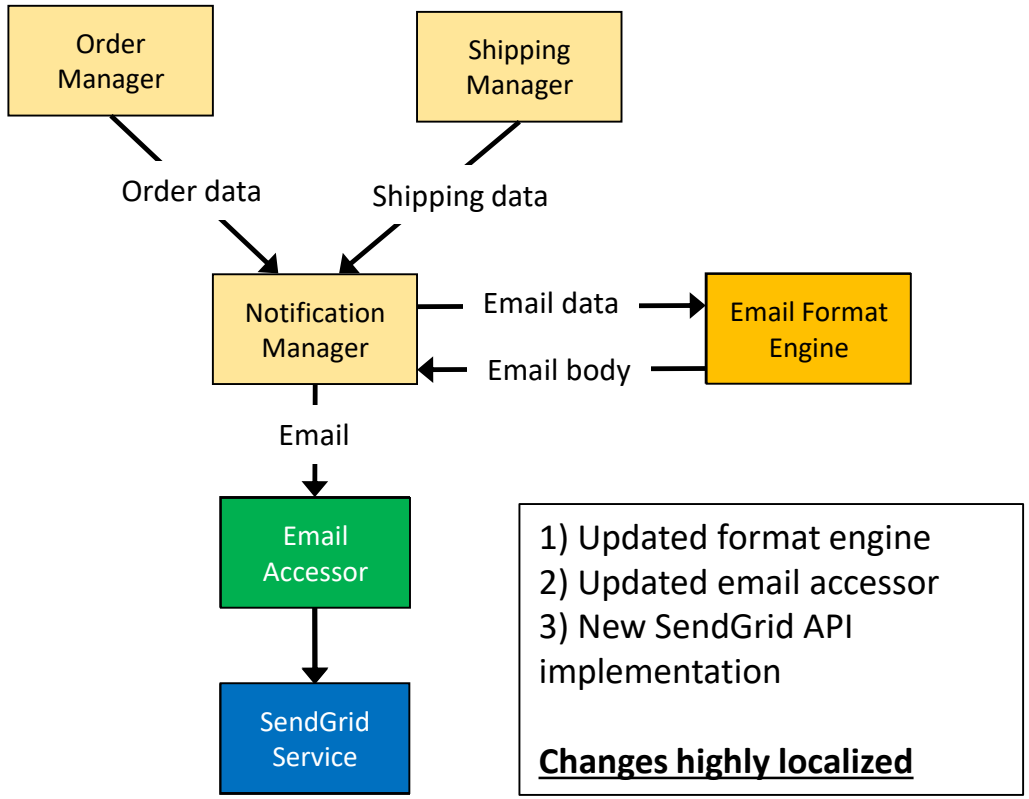
- Principle of Information Hiding
  - David Parnas (1972)
- Designing for Change
  - “Design Patterns, Elements of Object-Oriented Software”, Gamma, et al
- Volatility-based decomposition
  - Juval Lowy, “The Method”
  - www.idesign.net

Example: Isolating change

- Sending email notifications for order placement and shipping



Design changes:
HTML Emails
Send via SendGrid



1) Updated format engine
2) Updated email accessor
3) New SendGrid API implementation

Changes highly localized

Designing for testability

Impact of test-driven design on design quality

- 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!
- Enables refactoring and extensibility → Business Agility!

Services vs Objects

Services vs Objects

- Separate behavior from data
- Easier programming model to…
  - design
  - maintain
  - test
  - extend
  - scale
  - comprehend
- Positioned for cloud PaaS

Key Takeaways from Our Experience

- Our processes and designs need to allow for tight integration of product development and customer development cycles
- Design happens from story to code
- Let's not skip the "engineering and analysis" part of our job
- We need a design identity to help provide the structure and conceptual integrity to maintain design "stamina" and enable continued innovation
  - Encapsulating change minimizes/localizes impact of new/changing requirements
- TDD increases the quality of your design AND enables refactoring as insights occur and features are added
  - Gets you "80% there"
- Services are easier to grasp than objects

Questions?

More on DPL Product Development Process: http://bit.do/dpldev

Doug Durham
ddurham@dontpaniclabs.com
http://blog.dontpaniclabs.com
@dnsdurham

Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Continuous attention to technical excellence and good design enhances agility.

Simplicity--the art of maximizing the amount of work not done--is essential.